Exercise 7.3.1

Show that the operation *cycle* preserves context-free languages, where cycle is defined by:

$$cycle(L) = \{xy \mid yx \in L\}$$

Informally, $cycle(L)$ allows all strings constructed as follows: take a string w from L, and choose an arbitrary position in the middle. Take the second part (this is x), then wrap around and concatenate the first part (this is y). For example, if $abcd \in L$, then all of the following strings are in $cycle(L)$: $abcd, bcda, cdab, dabc$.

Solution

We will prove this by PDA construction. Given a PDA P for L, we'll create PDA P_c for $cycle(L)$. Both accept by empty stack. P_c simulates an accepting computation in P, but it starts in the middle and simulates P on x, then "wraps around" and simulates P on y.

The difficulty lies in the stack. When P_c begins consuming x, it is missing the stack symbols that P would have from consuming y. To resolve this, we allow P_c to guess the top stack symbol and "pop" it by pushing a negated version. Later, when simulating P on y, we confirm these guesses: instead of pushing symbols, we confirm and pop the guesses.

We use a stack marker # to delimit the guessed symbols from the rest of the stack. Start by guessing the top symbol (the first one we'll pop) left by y. Place a copy above and below the marker. Then begin consuming x by simulating P normally on the portion above the stack above the marker. Eventually we return to the marker, meaning we popped the symbol we guessed (note: it is now recorded below the marker). Here is an example where we guess that y left A as the top stack symbol:

Start configuration	Z_0
Mark the stack	$\#Z_0$
Guess top symbol	$A\#AZ_0$
Arbitrary build up	$DEFA\#AZ_0$
	...
Eventually pop A	$\#AZ_0$

On return to the marker, one thing we can do is guess the next symbol to pop and repeat the step described above:

Guess B	$B\#BAZ_0$
...	...
Eventually pop B	$\#BAZ_0$
Guess C	$C\#CBAZ_0$
...	...
Eventually pop C	$\#CBAZ_0$

Another option when returning to the marker is to guess that we have finished consuming x – this means we must have emptied the stack left by y, so it is time to start consuming y and verifying that it builds the stack we guessed.

The general idea is that instead of pushing symbols onto the stack, we'll pop the previous guesses, checking that they match up. There are several technical details to this phase.

First, we must encode the top stack symbol for the simulation of P into the state, since we are now using the stack itself to hold guesses. For example, in the beginning of this phase P would be reading Z_0 so we transition from state p to p_{Z_0} to indicate that we are reading Z_0. If we "push" a C by popping the C below the marker, we'll transition to state p_C to record the the top symbol (C is no longer on the stack).

Second, when pushing a symbol in this phase, there are two options. If we guess that the symbol we're pushing will eventually be popped during this phase, we push it on top of the marker and simulate P normally above the marker. If we guess that the symbol we're pushing will be the last one pushed at this position, then we verify the previous guess by popping the symbol below the marker (assuming they match up). Here is an example:

Begining of phase	p_{Z_0}	$\#CBAZ_0$
Arbitrary buildup	p_{Z_0}	$XYZ\#CBAZ_0$
...	p_{Z_0}	...
Eventually return	p_{Z_0}	$\#CBAZ_0$
"Push" a C	p_C	$\#BAZ_0$
...	p_C	...
"Push" a B	p_B	$\#AZ_0$
...	p_B	...
"Push" an A	p_A	$\#Z_0$
Accept!	p_A	

You might notice that there are even more technical details. The example I've shown pops the symbol *below* the marker in one step. We can simulate this with three steps: (1) pop the marker, (2) pop the symbol and (3) replace the marker. Likewise, a transition in δ may push multiple symbols — we can pop multiple symbols by breaking this into several steps using intermediate states.

Comments

An English description with less detail than this was sufficient. You'll notice that even this solution did not cover every technical detail. This was a very difficult question and only a few students got full credit, so don't be too concerned if you did poorly. Some comments about the grading, from which you can extract some general lessons:

Students who demonstrated clear understanding of the problem and gave a plausible approach to solving it were awarded the most partial credit, even if that approach was not correct.

Many students submitted solutions with elements of the correct solution but it was obvious they got hints either from office hours or other students and understood very little about the solution. These students received little credit.

CS 381 Homework 10
Solutions

2. Show that half is not closed for CFL's.

Let L = { $a^n b^n c^i d d^{3i} d$ | n>0, i>0 }

Since CFL is closed under intersection with regular languages, we can do

half(L) $\cap$ a*b*c*d

which splits L into two halves:

$a^n b^n c^i d$ and $d^{3i} d$

In order for these to be of equal length, i = n,

So half(L) $\cap$ a*b*c*d = $a^n b^n c^n d$, which we know is not a CFL.

Therefore half(L) is not closed for CFLs.

Note: There are many choices for L which will work.

CS381, Homework #10 Solutions *Question 3 (7.3.4)*

Show that $shuffle(L, R)$ is context free, where L is a CFL and R is a regular language

This can be shown via a machine construction. First assume that L can be represented by a one-state PDA P which accepts by empty stack. Here, $P = (\{q_p\}, \Sigma, \Gamma, \delta_p, q_p, Z_0)$. R is represented by a DFA $D = (Q_d, \Sigma, \delta_d, q_0, F_d)$.

Now, for production in $\delta_p(q_p, \alpha, \beta) = (q_p, \gamma)$, define a new production for each state $s \in Q_d$, such that:

$$\delta'(s, \alpha, \beta) = (s, \gamma)$$

Now for transition in $\delta_d(q, \alpha) = \gamma$, define a new production:

$$\delta'(q, \alpha, S) = (\gamma, S)$$

Now finally define transition functions for all final states $f \in F_d$:

$$\delta(f, \epsilon, \epsilon) = (f_{final}, \epsilon)$$

This transition says that whenever we're in a final state in our DFA D, and we have an empty stack on our PDA P, then we should transition to a final state in the new PDA, because the acceptance criteria for both languages have been met.

Now the final PDA, $P' = (Q_d \cup \{f_{final}\}, \Sigma, \Gamma, \delta', q_0, \{f_{final}\})$. So the states of this new PDA are the states of D, and the transition function allows us to change state in D, and accept everything that would be accepted by P.

Give a counterexample to show that if L_1 and L_2 are both CFL's, then the $shuffle(L_1, L_2)$ need not be a CFL.

If we make $L_1 = a^n b^n$, and $L_2 = c^n d^n$, then by taking $shuffle(L_1, L_2) \cap a^* c^* b^* d^*$, we can produce $a^n c^m b^n d^m$, a language which clearly is not context free. Since context free languages are closed under intersection with regular sets, it must be that $shuffle$ is not context free.

A common mistake that people made here was assuming that if they had some a^n in each language, that it was necessarily the same n in the final language. That is not sufficient to show that the output might not be a CFL, because unless the terms are interwovel like they are here, it is possible to write a CFL for say $a^n b^n c^m d^m$. Declaring $m = n$ to be a special case proves nothing, it just happens by coincidence.

Question 4

We start with a machine with contents 01^n01^m0 and want to finish with contents $01^{nm}0$ with some possible zeroes at the end.

Note: Multiplication is just repeated addition. In other words:

$$n * m = \underbrace{m + m + m + ... + m}_{\text{n times}}$$

Therefore, with the pointer starting at the first 0, for each 1 we see we:

- Go to the next 0
- We copy the string of 1's we see to the end of the string (passed the 0)
- We delete the 1 we were looking at

The resulting tape should look like $0^{n+2}1^m01^{nm}$, so we just have to delete everything in front of the 01^{nm} term and add a zero after it, and we are done.