

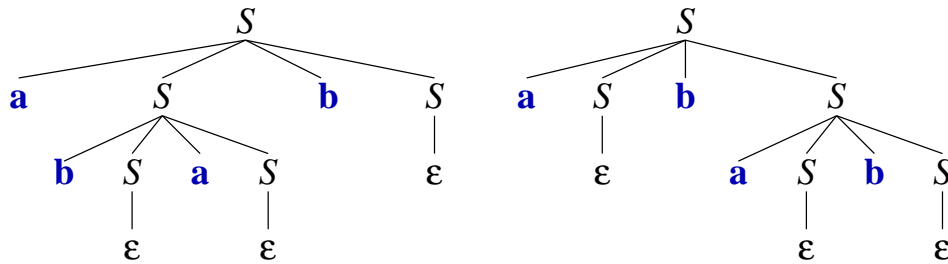
Week 7 Tutorial Solutions

Languages, Grammars and Pushdown Automata

Exercise 1

$$S \rightarrow aSbS \mid bSaS \mid \epsilon$$

1. S is the only non-terminal symbol (hence the start symbol). The terminals (tokens) are a and b .
2. Two parse trees for $abab$:



3. What language (call it L) does this grammar generate? Don't expect to be able to write a "regular-expression" style description of context-free languages — most context-free languages are *not* regular. The language is all strings over ab with the same number of as and bs .

How to prove this? It's easy to see that each production brings into a sentential form which is "under construction" an equal number of as and bs . So each sentential form, and therefore each sentence of L will have equal numbers of as and bs . A *formal* proof of this fact would be by induction on the length (number of steps) of the derivation of the sentence.

The other way is a bit harder. The proof is by induction on the length of string: assume that for all strings s *shorter* than the given string, if s has equal numbers of as and bs , then s is in L . (This is slightly different from the induction principle discussed in lectures, which involved proving $P(n+1)$ from the inductive hypothesis which was simply $P(n)$ alone).

Let the given string be s , with equal numbers of as and bs . If $s = \epsilon$ this is easy; assume s is non-empty. Write $s = \alpha\beta$ where α is as short as possible, but non-empty, such that each of α and β have equal numbers of as and bs (β may be empty). Suppose the given string s (and α) starts with a . Then α ends with b (why? — you work this one out!).

Then write $s = a\alpha'b\beta$: by induction both α and β are in L . Therefore, using the first production for S , $s = a\alpha'b\beta$ is in L .

Exercise 2

Grammar:

$$\begin{aligned} bexp &\rightarrow \mathbf{not} \ bexp \mid bexp \ \mathbf{or} \ bterm \mid bterm \\ bterm &\rightarrow bterm \ \mathbf{and} \ bfactor \mid bfactor \\ bfactor &\rightarrow (\ bexp \) \mid \mathbf{true} \mid \mathbf{false} \end{aligned}$$

Note that in this grammar (and in Exercise 3) it should be understood that the words in italics are non-terminal symbols, and the others are terminal symbols.

1. There are two parse trees for **not true or false**, for example. (Draw them.)
2. The relative precedence of **and** and **or** is as we would expect (**and** has higher precedence than **or**). The ambiguity relating to **not** makes this question slightly nonsensical — one parse of the expression above is correct, the other is not. When we eliminate the ambiguity we want to choose the right one.
3. Modified grammar with **not** as highest precedence:

$$\begin{aligned} bexp &\rightarrow bexp \ \mathbf{or} \ bterm \mid bterm \\ bterm &\rightarrow bterm \ \mathbf{and} \ bfactor \mid bfactor \\ bfactor &\rightarrow \mathbf{not} \ bfactor \mid (\ bexp \) \mid \mathbf{true} \mid \mathbf{false} \end{aligned}$$

Exercise 3

Grammar:

$$\begin{aligned} block &\rightarrow \mathbf{begin} \ decls \ stmts \ \mathbf{end} \\ decls &\rightarrow \mathbf{dec} \ ; \mid decls \ \mathbf{dec} \ ; \\ stmts &\rightarrow \mathbf{st} \mid \mathbf{st} \ ; \ stmts \end{aligned}$$

1. Non-deterministic PDA:

$$\begin{aligned} \delta(q_1, \epsilon, block) &\mapsto q_1/\mathbf{begin} \ decls \ stmts \ \mathbf{end} \\ \delta(q_1, \epsilon, decls) &\mapsto q_1/\mathbf{dec} \ ; \\ \delta(q_1, \epsilon, decls) &\mapsto q_1/decls \ \mathbf{dec} \ ; \\ \delta(q_1, \epsilon, stmts) &\mapsto q_1/\mathbf{st} \\ \delta(q_1, \epsilon, stmts) &\mapsto q_1/\mathbf{st} \ ; \ stmts \end{aligned}$$

Also the following (the first for each terminal x):

$$\begin{aligned} \delta(q_1, x, x) &\mapsto q_1/\epsilon \\ \delta(q_0, \epsilon, Z) &\mapsto q_1/block \ Z \\ \delta(q_1, \epsilon, Z) &\mapsto \underline{q_2}/\epsilon \end{aligned}$$

Exercise 4

1. The two productions for *stmts* both start with **st** so a single lookahead symbol cannot differentiate.

One of the productions for *decls* is left-recursive, so a lookahead of **dec** is compatible with both *decls* productions. (An unbounded lookahead would be required, which is not very useful.)

2. There are standard techniques for dealing with these issues, but we haven't covered them in this course. Nevertheless, with a bit of thought it's reasonably clear that changing the productions as follows is satisfactory:

$$\begin{aligned} \text{stmts} &\rightarrow \mathbf{st\ stmts'} \\ \text{stmts'} &\rightarrow \epsilon \mid ; \text{stmts} \end{aligned}$$

The left recursion solution can be modelled on the expression example from lectures:

$$\begin{aligned} \text{decls} &\rightarrow \mathbf{dec\ ;\ decls'} \\ \text{decls'} &\rightarrow \mathbf{dec\ ;\ decls'} \mid \epsilon \end{aligned}$$

3. The PDA with lookahead is as follows with other transitions as in Exercise 3: (we use the notation $\delta(q, \epsilon[x], \alpha) \mapsto q' / \alpha'$ for a transition which looks at but does *not* consume the input symbol x)

$$\begin{aligned} \delta(q_1, \epsilon[\mathbf{begin}], \text{block}) &= q_1 / \mathbf{begin\ decls\ stmts\ end} \\ \delta(q_1, \epsilon[\mathbf{dec}], \text{decls}) &= q_1 / \mathbf{dec\ ;\ decls'} \\ \delta(q_1, \epsilon[\mathbf{dec}], \text{decls'}) &= q_1 / \mathbf{dec\ ;\ decls'} \\ \delta(q_1, \epsilon[\mathbf{st}], \text{decls'}) &= q_1 / \epsilon \\ \delta(q_1, \epsilon[\mathbf{st}], \text{stmts}) &= q_1 / \mathbf{st\ stmts'} \\ \delta(q_1, \epsilon[;], \text{stmts'}) &= q_1 / \mathbf{;\ stmts} \\ \delta(q_1, \epsilon[\mathbf{end}], \text{stmts'}) &= q_1 / \epsilon \end{aligned}$$

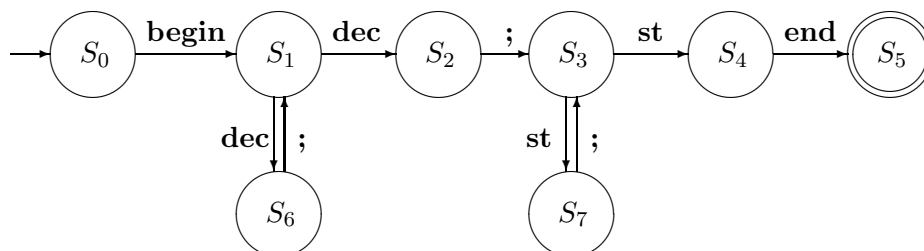
Exercise 5

Here is a regular expression:

begin (dec ;)* dec ; (st ;)* st end

Here is a NFA (somewhat simplified from what you would get by blindly applying the constructions in lectures). You'll notice that you could shift S_6, S_7 and the vertical arrows to the right, and get an NFA which doesn't require look-ahead or backtracking (ie, it would be a DFA except for the fact that it would have missing transitions).

Converting that NFA to a right-linear grammar would give a grammar which doesn't require lookahead beyond the next input symbol (unlike the one below).



Note that another regular expression for the same language would be

begin dec ; (dec ;)* st (; st)* end

In the same way the productions for *decls* and *stmts* could be changed to

$$\begin{aligned} \text{decls} &\rightarrow \mathbf{dec} ; \mid \mathbf{dec} ; \text{decls} \\ \text{stmts} &\rightarrow \mathbf{st} \mid \text{stmts} \mathbf{st} ; \end{aligned}$$

This would remove the left recursion, and the need for unbounded lookahead, but still requires lookahead of more than one symbol.

Corresponding grammars:

Right-linear grammar, with start symbol S_0 :

$$\begin{aligned} S_0 &\rightarrow \mathbf{begin} S_1 & S_1 &\rightarrow \mathbf{dec} S_6 \\ S_1 &\rightarrow \mathbf{dec} S_2 & S_6 &\rightarrow ; S_1 \\ S_2 &\rightarrow ; S_3 & S_3 &\rightarrow \mathbf{st} S_7 \\ S_3 &\rightarrow \mathbf{st} S_4 & S_7 &\rightarrow ; S_3 \\ S_4 &\rightarrow \mathbf{end} S_5 \\ S_5 &\rightarrow \epsilon \end{aligned}$$

Left-linear grammar, with start symbol S_5 :

$$\begin{aligned} S_0 &\rightarrow \epsilon & S_1 &\rightarrow S_6 ; \\ S_1 &\rightarrow S_0 \mathbf{begin} & S_6 &\rightarrow S_1 \mathbf{dec} \\ S_2 &\rightarrow S_1 \mathbf{dec} & S_3 &\rightarrow S_7 ; \\ S_3 &\rightarrow S_2 ; & S_7 &\rightarrow S_3 \mathbf{st} \\ S_4 &\rightarrow S_3 \mathbf{st} \\ S_5 &\rightarrow S_4 \mathbf{end} \end{aligned}$$

Exercise 6

(a) No. Here is an example:

$$S \rightarrow A, A \rightarrow aA, A \rightarrow B, B \rightarrow bB, B \rightarrow \epsilon$$

If you change A to B (or vice versa), it will generate the string ba

(b) Firstly, if there are any sets of productions which form a “cycle”, such as

$$A \rightarrow B \quad B \rightarrow C \quad C \rightarrow A$$

(so that $A \Rightarrow^* A$), replace each of A, B, C wherever it occurs by A .

Secondly, for any production $A \rightarrow B$, where the grammar also contains productions $B \rightarrow \beta_i$ (for some set of i s) replace $A \rightarrow B$ by $A \rightarrow \beta_i$ for each such i . (Do not delete $B \rightarrow \beta_i$!).

(An alternative to this second step is to say: for any production $A \rightarrow B$, where the grammar also contains productions $C_i \rightarrow \alpha_i$ where α_i contains A , let β_i be α_i with A replaced by B . Then replace $A \rightarrow B$ by $C_i \rightarrow \beta_i$ for each i . However this step doesn't quite work when A is the starting symbol S).

It may be easier to visualise these ideas by considering the corresponding NFA than by thinking directly about a grammar.