

Week 7 Tutorial  
Languages, Grammars and Pushdown Automata

---

**Exercise 1**

Consider the grammar

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

1. Identify the terminal and non-terminal symbols.
2. Show that this grammar is ambiguous by constructing two different parse trees for *abab*.
3. describe the language this grammar generates.

**Exercise 2**

Consider the grammar:

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

1. Demonstrate that this grammar is ambiguous.
2. Do the parse trees reflect the precedence of boolean operators that you would expect?
3. Modify the grammar to eliminate the ambiguity and to reflect normal operator precedence.

**Exercise 3**

For the 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. Derive a (non-deterministic) PDA and show that it *accepts* the string: **begin dec; dec; st end**
2. Show that it *rejects* the string: **begin dec; st; end**

#### Exercise 4

For the same grammar

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

suppose we wanted to construct a *deterministic* PDA by using the lookahead symbol to direct the choice of productions to expand on the stack.

1. Assuming a single lookahead symbol, which productions will cause difficulties?
2. Can you suggest sensible modifications of the grammar to overcome those problems? Of course, the *language* must remain unchanged.
3. For this new grammar, derive a PDA with lookahead.
4. Demonstrate that it accepts **begin dec; dec; st end**
5. Demonstrate that it rejects **begin dec; st; end**

#### Exercise 5

In fact, the *block* language of exercise 3 is *regular*. Define a regular expression that matches this language and derive a NFA by the process used in lectures.

If time permits, convert this NFA into

- a right-regular grammar
- a left-regular grammar

by the method described in lectures. You may be able to see some easy ways to simplify this NFA before producing the grammars.

#### Exercise 6

The method described in lectures for converting a NFA to a regular grammar can produce a grammar containing productions of the form  $A \rightarrow B$ . Here we consider how to convert such a grammar to one which does not contain such productions, but generates the same language (and is otherwise similar).

- (a) Is it satisfactory to simply change all occurrences of  $A$  in the grammar to  $B$ ? If so, why? If not, why not (give an example where doing this changes the language)?
- (b) Devise a systematic method of changing a grammar to avoid such productions, generates the same language (and is otherwise as similar as possible).
- (c) Does your solution to (b) work when  $A$  is the starting symbol? Does it work when you have a “cycle” of such productions, such as  $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ ? If no, can you adapt it satisfactorily?