The Australian National University
School of Computing
Dirk Pattinson and Victor Rivera

Semester 2, 2021
Practical Session 9

Foundations of Computation

The practical contains a number of exercises designed for the students to practice the course content. During the practical session, the tutor will work through some of these exercises while students will be responsible for completing the remaining exercises in their own time. There is no expectation that all the exercises will be covered in the practical session.

Covers: Lecture Material Week 9

At the end of this tutorial, you will be able to

- construct a grammar given an automaton;
- design Context Free Grammars
- construct an automaton given a grammar;
- design push-down automata given a language;
- show that a given grammar is ambiguous.
- convert a Context Free Grammar to a Push-down Automata.

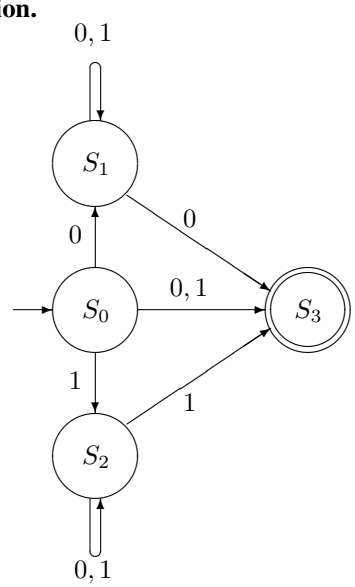
Exercise 1 **From Grammars to Automata**

The following right linear grammar describes a language of binary strings that begin and end with the same bit:

$$\begin{aligned} S_0 &\to 0S_1 \mid 1S_2 \mid \epsilon \\ S_1 &\to 0S_1 \mid 0S_3 \mid \epsilon \\ S_2 &\to 1S_2 \mid 1S_3 \mid \epsilon \\ S_3 &\to \epsilon \end{aligned}$$

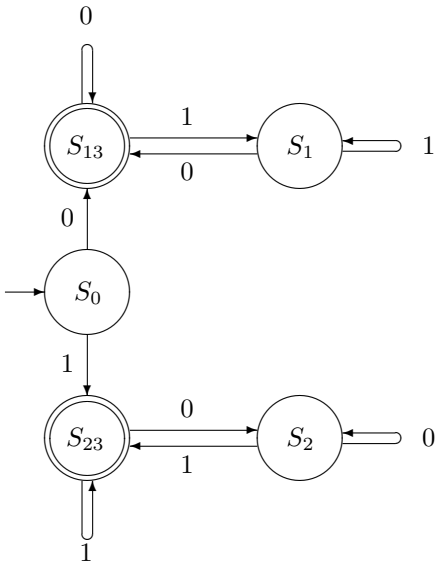
where S_0 is the start symbol.

1. Use the algorithm presented in lectures, convert this right linear grammar to a NFA.
   Note that the final state S_f in the algorithm does not play any role in this case, so you can safely delete it.
   Solution.



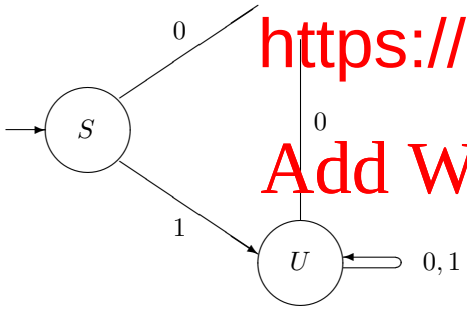
2. If you need more practice with finite automata, use the subset construction algorithm to construct an equivalent DFA from your previous answer.

   Solution.



Exercise 2 **From Automata to Grammars**

Consider the non-deterministic finite automaton A:



1. Describe the language that this automaton accepts.

   Solution. The language $L(A)$ is the set of all binary integers that are *even* (or, equivalently, end in 0) and have no unnecessary leading 0s.

2. Following the algorithm presented in lectures, give a right-linear grammar accepting the same language.

   Solution.

$$\begin{aligned} S &\to 0T \mid 1U \\ T &\to \epsilon \\ U &\to 0T \mid 0U \mid 1U \end{aligned}$$

3. Can you see a simpler right-linear grammar that would accept the same language?

   Solution. There are no transitions out of the final state T, so we could eliminate it and use the smaller grammar:

$$\begin{aligned} S &\to 0 \mid 1U \\ U &\to 0 \mid 0U \mid 1U \end{aligned}$$

Exercise 3 **Deterministic PDAs**

Consider the context-free grammar

$$\begin{array}{lcl} \textit{block} & \rightarrow & \textbf{begin}\ \textit{decls}\ \textit{stmts}\ \textbf{end} \\ \textit{decls} & \rightarrow & \textbf{dec ;} \mid \textit{decls}\ \textbf{dec ;} \\ \textit{stmts} & \rightarrow & \textbf{st} \mid \textbf{st ;}\ \textit{stmts} \end{array}$$

where $\{\textbf{begin}, \textbf{end}, \textbf{dec}, \textbf{;}, \textbf{st}\}$ are the terminals, and $\textit{block}$ is the start symbol.

1. Derive a (non-deterministic) PDA from this grammar.

   Solution.

$$\begin{array}{rcll} \delta(S_0, \epsilon, Z) & \mapsto & S_1/\textit{block}\ \mathrm{Z} & \\ \delta(S_1, \epsilon, \textit{block}) & \mapsto & S_1/\textbf{begin}\ \textit{decls}\ \textit{stmts}\ \textbf{end} & \\ \delta(S_1, \epsilon, \textit{decls}) & \mapsto & S_1/\textbf{dec ;} & \\ \delta(S_1, \epsilon, \textit{decls}) & \mapsto & S_1/\textit{decls}\ \textbf{dec ;} & \\ \delta(S_1, \epsilon, \textit{stmts}) & \mapsto & S_1/\textbf{st} & \\ \delta(S_1, \epsilon, \textit{stmts}) & \mapsto & S_1/\textbf{st ;}\ \textit{stmts} & \\ \delta(S_1, x, x) & \mapsto & S_1/\epsilon & (\text{for all } x \in \{\textbf{begin}, \textbf{end}, \textbf{dec}, \textbf{;}, \textbf{st}\}) \\ \delta(S_1, \epsilon, Z) & \mapsto & \underline{S_2}/\epsilon & \end{array}$$

   with start state S_0 and final state $\underline{S_2}$.

   We used the variable x above to compress five transitions (for each terminal) into one. This is fine to do e.g. to save time in an exam *as long as you are completely clear and explicit about the meaning of your abbreviations.*

2. Give a trace to show that your PDA

   Solution.

$$\begin{array}{lrr} (S_0, \textbf{begin dec; dec} & &) \\ \Rightarrow (S_1, & \textbf{begin dec; dec; st end}, & \textbf{begin}\ \textit{decls}\ \textit{stmts}\ \textbf{end}\ Z) \\ \Rightarrow (S_1, & \textbf{dec; dec; st end}, & \textit{decls}\ \textit{stmts}\ \textbf{end}\ Z) \\ \Rightarrow (S_1, & \textbf{dec; st end}, & \textbf{dec ;}\ \textit{stmts}\ \textbf{end}\ Z) \\ \Rightarrow (S_1, & \textbf{; st end}, & \textbf{;}\ \textit{stmts}\ \textbf{end}\ Z) \\ \Rightarrow (S_1, & \textbf{st end}, & \textit{stmts}\ \textbf{end}\ Z) \\ \Rightarrow (S_1, & \textbf{st end}, & \textbf{st end}\ Z) \\ \Rightarrow (S_1, & \textbf{end}, & \textbf{end}\ Z) \\ \Rightarrow (S_1, & \epsilon, & Z) \\ \Rightarrow (\underline{S_2}, & \epsilon, & \epsilon) \\ \Rightarrow\ \textit{accept} & & \end{array}$$

Exercise 4 **Deterministic Pushdown Automata**

Design a deterministic push-down automaton that recognises language L and give the transition function of the automaton.

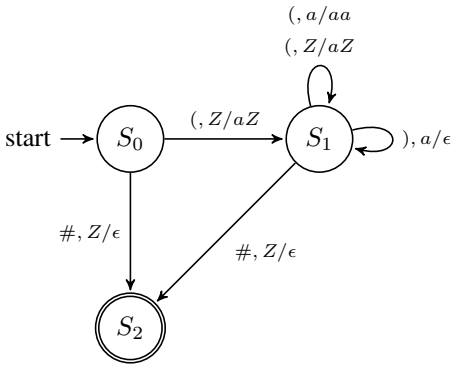
$$L = \{w\# \mid w \in \{(,)\}^* \text{ and } w \text{ is a well-parenthesised string}\}.$$

Here the alphabet is $\{(,), \#\}$, e.g $(()())()\#$ belongs to the language, and $(()(\#)()$ does not belong.

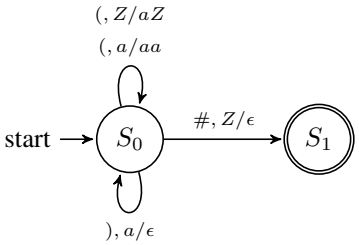
Solution.

$$\begin{array}{rcl} \delta(S_0, (, Z) & \mapsto & S_1/aZ \\ \delta(S_0, \#, Z) & \mapsto & \underline{S_2}/\epsilon \\ \delta(S_1, (, a) & \mapsto & S_1/aa \\ \delta(S_1, (, Z) & \mapsto & S_1/aZ \\ \delta(S_1,), a) & \mapsto & S_1/\epsilon \\ \delta(S_1, \#, Z) & \mapsto & \underline{S_2}/\epsilon \end{array}$$

And its graphical representation:



Here's another solution (another PDA that recognises language L):



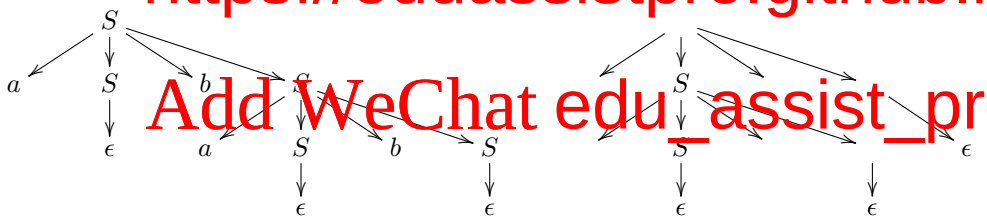
Exercise 5 **Ambiguity**

Consider the grammar

$$S \to aSbS \mid bSaS \mid \epsilon$$

1. Show that this grammar is ambig b.

   Solution.



2. Describe the language this grammar generates.

   Solution. This grammar generates the language of all strings over $\{a, b\}$ with the same number of as and bs.

3. Optional and hard: Give a formal proof of the fact that the language generated by this grammar is indeed the language that you have identified above.

   Solution. One direction is easy: every *sentential form* produced by this grammar has an equal number of as and bs, so the language specified by this grammar must be included in the language of strings with an equal number of as and bs.

   The other direction – showing that all strings with an equal number of as and bs can be derived by this grammar – is a bit harder. The proof is by induction on the length of strings: assume that for all strings s *shorter* than the given string, if s has equal numbers of as and bs, then s is generated by the grammar.

   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 generated by the grammar. Therefore, using the first production for S, $s = a\alpha' b\beta$ is generated by the grammar.

   (If α started with b then exactly equivalent reasoning would hold, using the second production for S).

   Incidentally, you may be wondering if there exists any ***unambiguous*** grammar for this language. The answer is yes, but the simplest example I've found uses 7 non-terminals and 17 productions – rather an increase on 1 and 3! This shows that disambiguating a grammar can in general be quite difficult.

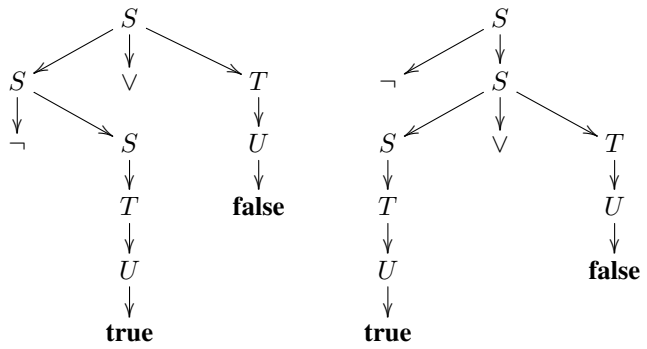
Exercise 6 **Operator Precedence**

Consider the grammar:

$$\begin{array}{lcl} \text{S} & \rightarrow & \neg\ \text{S} \mid \text{S} \vee \text{T} \mid \text{T} \\ \text{T} & \rightarrow & \text{T} \wedge \text{U} \mid \text{U} \\ \text{U} & \rightarrow & (\ \text{S}\ ) \mid \textbf{true} \mid \textbf{false} \end{array}$$

1. Demonstrate that this grammar is ambiguous.

   Solution. There are two parse trees for $\neg$ **true** $\vee$ **false**, for example:



2. Modify the grammar to eliminate the ambiguity and to reflect the normal precedence for logical operators ($\neg$ binds the tightest, then $\wedge$, then $\vee$.)

   Solution. $\begin{array}{lcl} \text{S} & \rightarrow & \text{S} \vee \text{T} \mid \text{T} \\ \text{T} & \rightarrow & \text{T} \wedge \text{U} \mid \text{U} \\ \text{U} & \rightarrow & \neg\ \text{U} \mid (\ \text{S}\ ) \mid \textbf{true} \mid \textbf{false} \end{array}$ There are other answers possible here; for example we could have another non-terminal V necessary).

Exercise 7 **Context-Free Grammars and Automata**

1. Give a *context free* grammar H that generates the language

   $$M = \{a^m b^n \mid 2m > n > m\}.$$

   Hint: m can't be 0, because in that case $2m = m$. m can't be 1, because in that case $2 > n > 1$, such a natural number n doesn't exist. So the shortest string in the language M is $aabbb$. For longer strings, you need to make sure that the number n of bs and the number m of as satisfy $2m > n > m$.

   Solution. It is impossible for the number of as to be less than two, so the grammar below starts off by adding two as and three bs, then proceeds by adding either one or two bs for each a added.

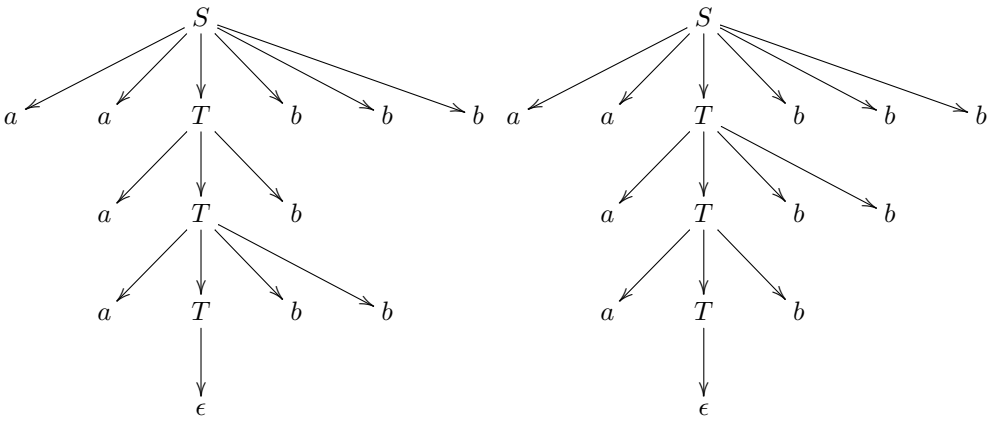
   $$\begin{array}{lcl} S & \rightarrow & aaTbbb \\ T & \rightarrow & \epsilon \mid aTb \mid aTbb \end{array}$$

   where S is the start symbol.

   This is not the only correct answer – most context-free languages are definable via many different context-free grammars, and the language M is no exception to this. Of note, the grammar above is *ambiguous* (see below); designing an unambiguous grammar for M is an worthwhile exercise.

2. Using the grammar H you defined above, draw a parse tree for the string a^4b^6, i.e., $aaaabbbbbb$.

   Solution. For the grammar above either of these trees are correct:



3. Consider the following context free grammar in the lectures:

$$S \rightarrow T \mid W$$

$$\begin{array}{ll} T \rightarrow UV & W \rightarrow XY \\ U \rightarrow aUb \mid \epsilon & X \rightarrow aX \mid \epsilon \\ V \rightarrow cV \mid \epsilon & Y \rightarrow bYc \mid \epsilon \end{array}$$

Convert this grammar to a (non-deterministic) PDA and give the transition relation of the PDA.

Solution. The constructed (non-deterministic PDA) has an initial state S_0, with the following transitions:

$$\begin{array}{lll} \text{Initialisation:} & \delta(S_0, \epsilon, Z) & \mapsto \quad S_1/SZ \\ \text{Non-terminals} & \delta(S_1, \epsilon, S) & \mapsto \quad S_1/T \\ & \delta(S_1, \epsilon, S) & \mapsto \quad S_1/W \\ & \delta(S_1, \epsilon, T) & \quad\quad S_1/UV \\ & & \vdots \\ & \delta(S_1, \epsilon, V) & \quad\quad S_1/\epsilon \\ & \delta(S_1, \epsilon, W & \\ & \delta(S_1, \epsilon, X) & \\ & \delta(S_1, \epsilon, X) & \\ & \delta(S_1, \epsilon, Y) & \quad\quad 1 \\ & \delta(S_1, \epsilon, Y) & \mapsto \quad S_1/\epsilon \\ \text{Terminals:} & \delta(S_1, x, x) & \mapsto \quad S_1/\epsilon \\ \text{Termination:} & \delta(S_1, \epsilon, Z) & \mapsto \quad \underline{S_2}/\epsilon \end{array}$$

where $x \in \{a, b, c\}$.

4. Give a PDA trace for processing the string $abbcc$ using the PDA you constructed in the previous question. State whether the string is accepted or rejected.

Solution. The PDA trace for processing the string $abbcc$ is as below.

$$\begin{array}{lll} (S_0, abbcc, Z) & \Rightarrow & (S_1, abbcc, SZ) \\ & \Rightarrow & (S_1, abbcc, WZ) \\ & \Rightarrow & (S_1, abbcc, XYZ) \\ & \Rightarrow & (S_1, abbcc, aXYZ) \\ & \Rightarrow & (S_1, bbcc, XYZ) \\ & \Rightarrow & (S_1, bbcc, YZ) \\ & \Rightarrow & (S_1, bbcc, bYcZ) \\ & \Rightarrow & (S_1, bcc, YcZ) \\ & \Rightarrow & (S_1, bcc, bYccZ) \\ & \Rightarrow & (S_1, cc, YccZ) \\ & \Rightarrow & (S_1, cc, ccZ) \\ & \Rightarrow & (S_1, c, cZ) \\ & \Rightarrow & (S_1, \epsilon, Z) \\ & \Rightarrow & (\underline{S_2}, \epsilon, \epsilon) \end{array}$$

Accept.

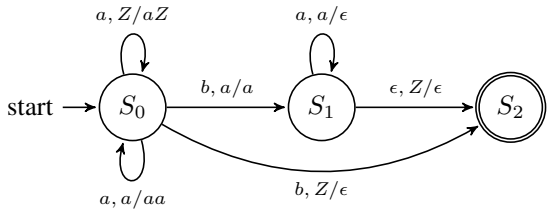
Exercise 8 **More on Deterministic Pushdown Automata**

For each of the following languages, design a deterministic push-down automaton that recognises the language and give the transition function of the automaton.

1. $\{a^n\ b\ a^n \mid n \geq 0\}$;

   Solution.

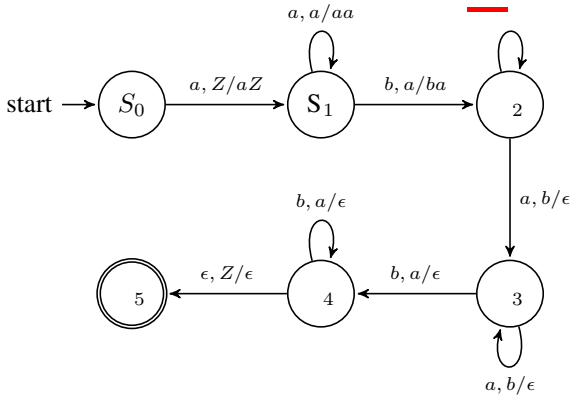
$$
\begin{array}{lcll}
\delta(S_0, a, Z) & \mapsto & S_0/aZ & \\
\delta(S_0, a, a) & \mapsto & S_0/aa & \\
\delta(S_0, b, Z) & \mapsto & \underline{S_2}/\epsilon & \text{to accept immediately if string is } b \\
\delta(S_0, b, a) & \mapsto & S_1/a & \\
\delta(S_1, a, a) & \mapsto & S_1/\epsilon & \\
\delta(S_1, \epsilon, Z) & \mapsto & \underline{S_2}/\epsilon &
\end{array}
$$



2. $\{a^n b^m a^m b^n \mid m > 0 \wedge n > 0\}$;

   Solution.

$$
\begin{array}{lcl}
\delta(S_0, a, Z) & \mapsto & S_1/aZ \\
\delta(S_1, a, a) & \mapsto & S_1/aa \\
\delta(S_1, b, a) & \mapsto & S_2/ba \\
 & \vdots & \\
\delta(S_4, b, a) & \mapsto & S\ /\epsilon \\
\delta(S_4, \epsilon, Z) & \mapsto & \_\_
\end{array}
$$



We can come up with a more optimised PDA: