Department of Computer Science, Australian National University
COMP2600 — Formal Methods in Software Engineering
Semester 2, 2015

Week 6 Tutorial

Non-Deterministic Finite State Automata and Grammars

You should hand in attempts to the questions indicated by (*) to your tutor at the start of each tutorial. Showing effort at answering the indicated questions will contribute to the 4% "Tutorial Preparation" component of the course; your attempts will not be marked for correctness. You may collaborate with your fellow students or others, so long as you hand in your work individually and indicate who you have worked with.

Objectives:

- convert a NFA to a DFA (Q 1.2);
- convert a NFA to a right linear grammar (Q 2.2);
- convert a right linear grammar to a NFA (Q 1.1);
- draw parse trees (Q 3.1).

Question 1

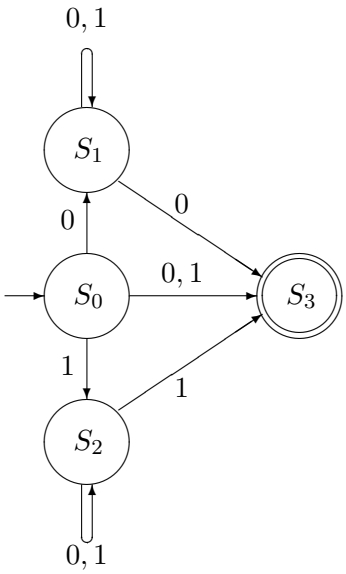
1. The following right linear grammar describes a language of binary strings that begin and end with the same bit:

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

   where S_0 is the start symbol. Use the algorithm presented in lectures, convert this right linear grammar to a NFA. $(*)$

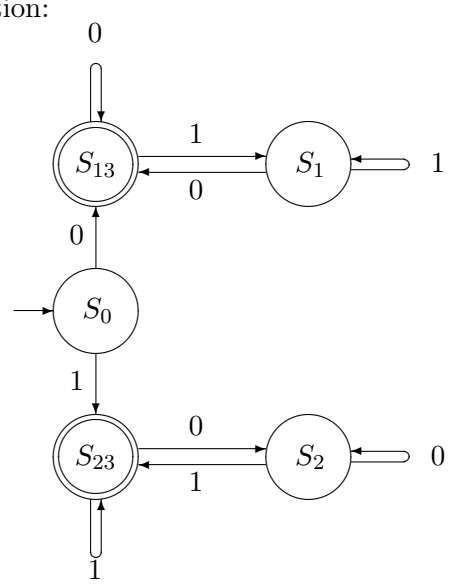
   Note that the final state S_f in the algorithm does not play any role in this case, so you can safely delete it.

   Solution:



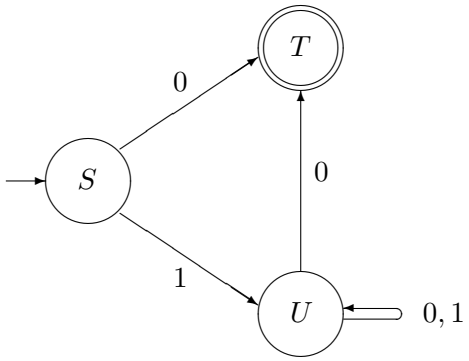
2. Use the subset construction algorithm to construct an equivalent ***deterministic*** FSA from your previous answer. $(*)$

   Solution:



Question 2

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), but without unnecessary leading 0s.

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

   Solution:

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

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:

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

Question 3

Consider the context-free grammar G:

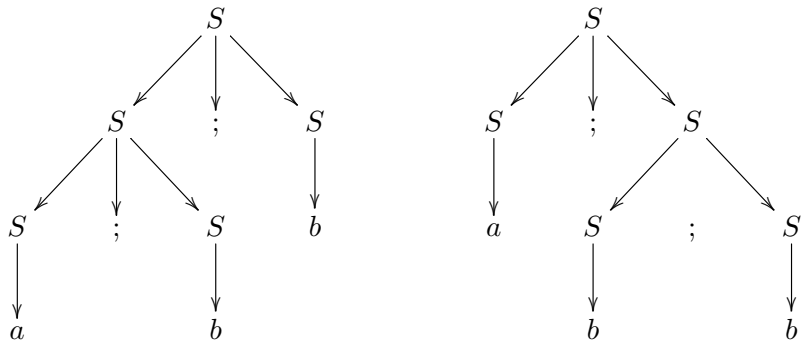
$$S \rightarrow a \mid b \mid S; S$$

generating non-empty lists of a's and b's separated by semicolons.

1. Give two different parse trees generating the string $a; b; b$ with this grammar. $(*)$

   Note that the semi-colon ;, like a and b, is a terminal symbol (but | isn't!).

   Solution:



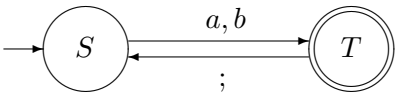
2. Give a ***right-linear*** grammar that generates the same language.

   Solution:

$$S \rightarrow aT \mid bT$$
$$T \rightarrow \epsilon \mid ;S$$

3. Convert that grammar into a non-deterministic finite automaton using the algorithm given in lectures.

   Solution:



   Note: we did not need to introduce a new final state S_f here, as was done in lectures, because our right-linear grammar contains no productions of the form $U \rightarrow a$ that would bring S_f into play.

Question 4

Given any word w over the alphabet $\{a, b\}$, let $n_b(w)$ be the number of b's appearing in w. Consider the language

$$L = \{a^i w \mid w \in \{a, b\}^* \wedge i = n_b(w)\}$$

1. Give a context-free grammar that generates L.

   Solution:

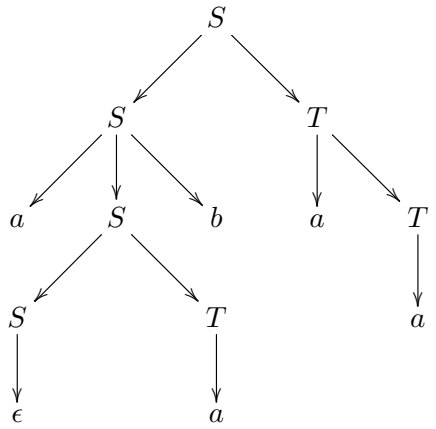
   There may be several correct CFGs that generate L, but here's my answer:

$$S \rightarrow aSb \mid ST \mid \epsilon$$
$$T \rightarrow a \mid aT$$

Here S expands by either (i) adding b to the end and incrementing the 'counter' a at the start, (ii) adding a string of a's to the end, via the non-terminal T, or (iii) transitioning to the empty string ϵ to end the expansion.

2. Give a parse tree that shows that this grammar generates the string $aabaa$.

   Solution:



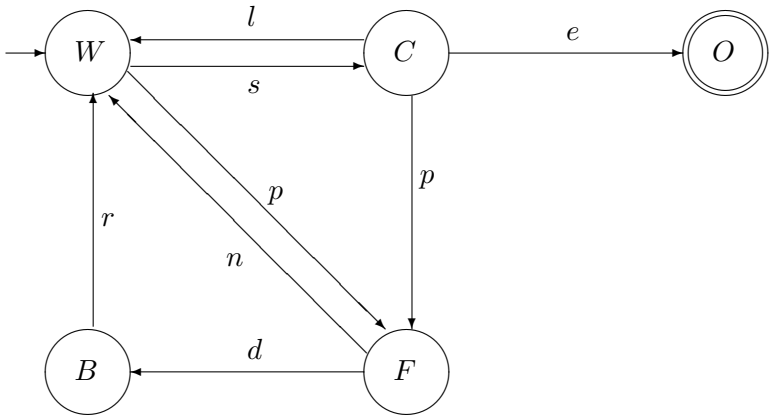
Question 5

The epic Pac-Man game basically runs as follows: the player (i.e., Pac-Man) navigates through a maze, eating pellets and avoiding the ghosts who chase the player in the maze. After eating a power pellet, the player is granted a temporary super power to eat the ghosts, during this period, the ghosts do not chase the player, but avoid him. The behaviours (states) of ghosts in Pac-Mac can be simplified as below:

- randomly wander the maze;
- chase Pac-Man (the player) when within line of sight;
- flee Pac-Man after he has consumed a power pellet;
- return to the central base to regenerate after eaten by Pac-Man;
- game over after eats Pac-Man.

Design a NFA to model the above game AI for the ghosts.

Solution:



States: W for "wander the maze"; C for "chase Pac-Man"; F for "Flee Pac-Man"; B for "return to base"; O for "game over".

Transitions: s for "spots Pac-Man"; p for "Pam-Man eats power pellet; l for "loses Pac-Man"; e for "eats Pac-Man"; n for "power pellet expires, Pac-Man returns to normal"; d for "eaten by Pac-Man and dies"; r for "regenerate from the central base."