Technical University of Denmark

Written examination, date June 3rd, 2013

Course name: Computer Science Modelling

Course number: 02141

Aids allowed: All written aids are permitted

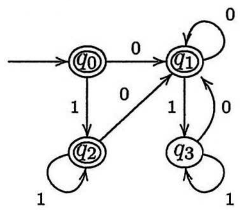
Exam duration: 4 hours

Weighting: 7 step scale

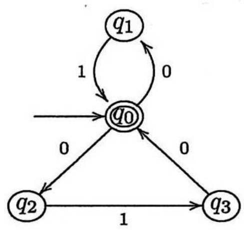
Regular languages (30pt)

Exercise 1 (5%) Let $\Sigma = \{0, 1\}$ and consider the following four languages:

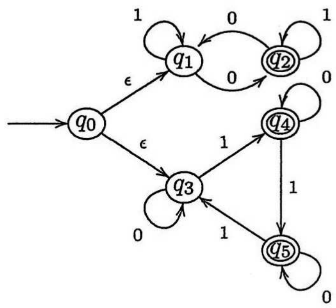
- L_1 is given by the DFA with the following transition relation:



- L_2 is defined by the NFA with the following transition relation:



- L_3 is defined by the ϵ-NFA with the following transition relation:



- L_4 is given by the regular expression $(0 + 11^*0)(0 + 11^*0)^* + \epsilon + 11^*$

Fill out the following table with a yes if the string in the column is a member of the language in the row and a no if it is not a member of the language:

	0101	01010	0101001	110110
L_1				
L_2				
L_3				
L_4				

Exercise 2 (15%) Let L be a set of strings over the alphabet Σ. Define

$$\begin{array}{lrcl} \text{the prefixes of } L: & \mathsf{Pre}(L) & = & \{x \in \Sigma^* \mid \exists y \in \Sigma^* : xy \in L\} \\ \text{the suffixes of } L: & \mathsf{Suf}(L) & = & \{z \in \Sigma^* \mid \exists x \in \Sigma^* : xz \in L\} \\ \text{the substrings of } L: & \mathsf{Sub}(L) & = & \{y \in \Sigma^* \mid \exists x, z \in \Sigma^* : xyz \in L\} \end{array}$$

Prove the following results:

(a) If L is a regular language then so is $\mathsf{Pre}(L)$.

(b) If L is a regular language then so is $\mathsf{Suf}(L)$.

(c) If L is a regular language then so is $\mathsf{Sub}(L)$.

Exercise 3 (10%) Consider the following language

$$L = \{a^i b^j c^k \mid i, j, k \geq 0; i = 1 \Rightarrow j = k\}$$

Is L a regular language? Present an argument for why this is the case – or why it is not the case.

Context-Free Languages (30pt)

A Grammar For Lists We design a small part of a programming language to represent lists of integers. We have a symbol " `nil` " for the empty list, and an infix operator " `cons` " that works as follows: if i is an integer and l is an integer list, then i `cons` l is the list the results from pre-pending i to list l. For instance the integer list $0, 2, 1, 2$ can be represent as

`0 cons 2 cons 1 cons 2 cons nil`

To describe this language, we use the following Context-Free Grammar $G = (V, T, S, P)$ where

$$\begin{aligned} V &= \{\, \mathsf{S} \,,\, \mathsf{E} \,\} \\ T &= \{\, \texttt{nil} \,,\, \texttt{cons} \,,\, \texttt{0} \,,\, \texttt{1} \,,\, \texttt{2} \,,\, \texttt{3} \,\} \\ S &= \mathsf{S} \end{aligned}$$

and P contains the following productions:

$$\begin{aligned} \mathsf{S} &\rightarrow \texttt{nil} \mid \mathsf{E}\ \texttt{cons}\ \mathsf{S} \\ \mathsf{E} &\rightarrow \mathsf{0} \mid \mathsf{1} \mid \mathsf{2} \mid \mathsf{3} \end{aligned}$$

Here for simplicity, we consider here only the integers $0, \ldots, 3$.

Exercise 4 (5%) Show how to derive the word `0 cons 2 cons 1 cons 2 cons nil` from the start symbol S and draw the parse tree.

A More Convenient Notation While this notation with `nil` and `cons` is convenient for programming with lists it looks a bit ugly. Therefore we want to additionally give the programming language a notation of comma-separated lists that are enclosed by square brackets, for instance

$$[0, 2, 1, 2]$$

should be an alternative notation for list 0 `cons` 2 `cons` 1 `cons` 2 `cons` `nil` .

Exercise 5 (7%) Extend the set of terminal, variable symbols and productions of the above grammar to allow this notation of lists.

Abstract Syntax In order to actually realize this part of a programming language, we need to define abstract data structures to store the result of parsing an input program.

Exercise 6 (5%) Define Java data structures that can hold the integer lists after parsing. Hint: you can either use vectors of integers or linked lists.

Exercise 7 (6%) Describe the link between the concrete syntax—the grammar from *Exercise* 1 and *Exercise* 2—and the abstract syntax—the Java data structures from *Exercise* 3. You are free to choose how to describe this (e.g. by a diagram or by ANTLR-style annotation of the grammar).

Going to Far A crazy programmer that uses our programming language would like an extension that allows for "lists of lists" and suggests to simply add this production rule to the grammar:

$$\mathsf{E} \rightarrow \mathsf{S}$$

thus allowing that a list element can also be list.

Exercise 8 (7%) Show that with this addition the grammar becomes ambiguous. Hint: give two different parse trees for the word:

`0 cons 1 cons nil cons nil`

Semantics (40pt)

Assuming that the domain of numerals and arithmetic expressions are bounded, say $\{0, 1, \ldots, 7\}$. The semantics of numerals is given by:

$$
\begin{aligned}
\mathcal{N}[\![0]\!] &= \mathbf{0} \\
\mathcal{N}[\![1]\!] &= \mathbf{1} \\
\mathcal{N}[\![n\ 0]\!] &= (\mathbf{2} \cdot \mathcal{N}[\![n]\!]) \mod 8 \\
\mathcal{N}[\![n\ 1]\!] &= (\mathbf{2} \cdot \mathcal{N}[\![n]\!] + \mathbf{1}) \mod 8
\end{aligned}
$$

and the semantics for arithmetic expressions (see Table 1.1 in NN) are given by:

$$
\begin{aligned}
\mathcal{A}[\![n]\!] &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!] &= (s\ x) \mod 8 \\
\mathcal{A}[\![a_1 + a_2]\!] &= (\mathcal{A}[\![a_1]\!] + \mathcal{A}[\![a_2]\!]) \mod 8 \\
\mathcal{A}[\![a_1 * a_2]\!] &= (\mathcal{A}[\![a_1]\!] * \mathcal{A}[\![a_2]\!]) \mod 8 \\
\mathcal{A}[\![a_1 - a_2]\!] &= (\mathcal{A}[\![a_1]\!] - \mathcal{A}[\![a_2]\!]) \mod 8
\end{aligned}
$$

The semantics for boolean expressions, the natural and structural operational semantics are not changed. We refer to the extended language *Bounded While.*

Exercise 9 (8%)

1. In the semantics for arithmetic expressions given above, please indicate for each line whether it
   - corresponds to a basic element, or
   - corresponds to a composite element of arithmetic expressions.
2. Suppose we want to prove the following statement:
   - Let s and s' be two states satisfying that $sx = s'x$ for all x in $FV(a)$. Show that it holds $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$

   What proof technique should be used? Moreover, assume we are now considering the case that a has the form $a_1 + a_2$. Please write down the induction hypothesis (you do not need to carry out the proof itself).

Exercise 10 (12%)

Consider the following statement in the Bounded While, denoted by S:

$$i := n;\ \texttt{while}\ (i > 0)\ \texttt{do}\ (sum := sum + i;\ i := i - 1)$$

1. Does the following hold?

$$\langle S,\ [\mathtt{n} \mapsto \mathbf{2}, \mathtt{sum} \mapsto \mathbf{0}, \mathtt{i} \mapsto \mathbf{1}]\rangle \to [\mathtt{n} \mapsto \mathbf{2}, \mathtt{sum} \mapsto \mathbf{3}, \mathtt{i} \mapsto \mathbf{0}] \qquad (1)$$

   Explain why if not, and construct the derivation tree if yes.

2. Does the following hold?

$$\langle S,\ [\mathtt{n} \mapsto \mathbf{4}, \mathtt{sum} \mapsto \mathbf{0}, \mathtt{i} \mapsto \mathbf{2}]\rangle \to [\mathtt{n} \mapsto \mathbf{4}, \mathtt{sum} \mapsto \mathbf{10}, \mathtt{i} \mapsto \mathbf{0}] \qquad (2)$$

   Explain why if not, and construct the derivation tree if yes.

3. Does the statement terminate for all state? Why?

You can use the following abbreviations in your derivations:

$$\underline{w} = \texttt{while}\ (i > 0)\ \texttt{do}\ (sum := sum + i;\ i := i - 1)$$
$$S_0 = sum := sum + i;\ i := i - 1$$

and you can write $s_{k_1 k_2 k_3}$ for the state $[\mathtt{n} \mapsto \mathbf{k_1}, \mathtt{sum} \mapsto \mathbf{k_2}, \mathtt{i} \mapsto \mathbf{k_3}]$.

Exercise 11 (8%) We say a configuration $\langle S, s\rangle$ is stuck free, if there exists at least one γ such that $\langle S, s\rangle \Rightarrow \gamma$. Prove that for any statement S of the Bounded While, $\langle S, s\rangle$ is stuck free.

Exercise 12 (12%) Consider the following statement S:

$$(\texttt{while}\ (x < 7)\ \texttt{do}\ (x := x + 2))\ \texttt{par}\ (x := x * 2\ \texttt{or}\ x := x * 3)$$

in the Bounded While extended with nondeterminism and parallelism extensions (semantics for them are not changed, i.e., the same as in pages 50 and 52 in [NN07] respectively). We assume the statement is executed in a state where x has the value 2.

1. Construct *two different* derivation sequences in structural operational semantics showing that the execution of the statement can terminate in the same state where x has the value 8.
2. State all other possible results of the execution of the statement (you don't have to provide derivation sequences).

You can use the following abbreviations in your derivations:

$$\begin{aligned} S_0 &= x := x + 2 \\ S_1 &= \texttt{while}\ (x < 7)\ \texttt{do}\ (x := x + 2) \\ S_2 &= x := x * 2\ \texttt{or}\ x := x * 3 \end{aligned}$$

and you can write s_k for the state $[\mathtt{s} \mapsto \mathbf{k}]$.

Hint. Always indicate which rules you have applied by mentioning the appropriate rule name(s) for each derivation step.