Technical University of Denmark

Written examination, May 23, 2017

Course name: Computer Science Modelling

Course number: 02141

Aids allowed: All written aids are permitted

Exam duration: 4 hours

Weighting: 7-step scale

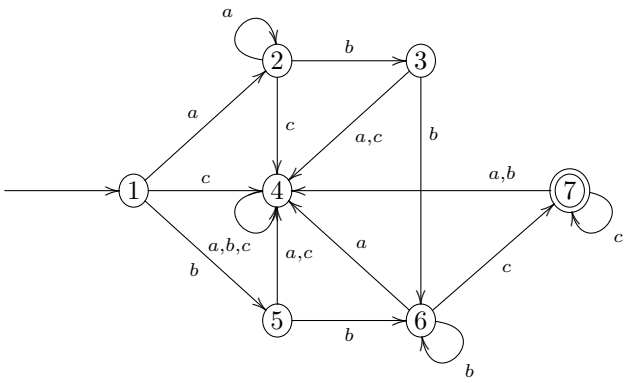
Old exam sets are not indicative of new exam sets.

Exercises on Regular Languages

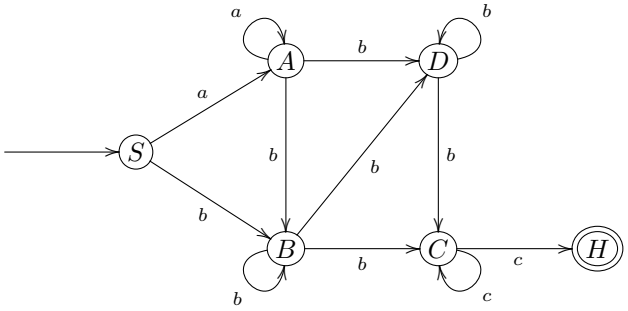
Exercise 1 (20%)

Let $\Sigma = \{a, b, c\}$ and consider the following languages:

- L_1 is given by the regular expression $a^*(bb)^*c^*$.
- L_2 is the language given by the DFA



- L_3 is given by the NFA



- L_4 is the language $\{a^i b^{2j} c^k \mid 0 \leq i, j \leq i+1, 1 \leq k\}$.

In this exercise we shall investigate the relationship between these languages.

(a) 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:

w	ϵ	bbc	aac	$abbc$	$abbbc$	$aabbcc$
L_1						
L_2						
L_3						
L_4						

(b) Use the subset construction algorithm (Hopcroft, Motwani and Ullman chapter 2) to construct a DFA corresponding to the NFA for L_3.

(c) Fill out the following table with a *yes* if the language in the leftmost column is a subset of the language in the topmost row and a *no* if this is not the case.

$\subseteq$	L_1	L_2	L_3	L_4
L_1	*yes*			
L_2		*yes*		
L_3			*yes*	
L_4				*yes*

If you answer *yes*, please give an argument for why, and if you answer *no*, please give a counter example.

(d) Prove that L_4 cannot be a regular language.

Exercise 2 (10%)

Let $L \subseteq \Sigma^*$ and define

$$\mathsf{pre}(L) = \{x \in \Sigma^* \mid \exists y \in \Sigma^* : xy \in L\}$$

Give a construction showing that if L is regular then so is $\mathsf{pre}(L)$; a formal mathematical proof is not required.

Exercises on Formal Methods

Exercise 3 (30% in all)

Program Graphs and Guarded Commands (10%)

These questions relate to the lecture material *Formal Methods: An Appetizer* as used on the course and to the program graphs AA and BB in Figure 1 below.

(a) For each of the two program graphs write a complete execution sequence (Section 1.2) starting in the initial memory $\sigma = [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 9, \mathtt{z} \mapsto 23]$.

(b) For each of the two program graphs determine whether or not they constitute a deterministic system (Section 1.4); motivate your answer.

(c) For each of the two programs determine whether or not they constitute a system that may run forever in the sense that there is an execution sequence $\langle q_\triangleright; \sigma_0\rangle \Longrightarrow \langle q_1; \sigma_1\rangle \Longrightarrow \langle q_2; \sigma_2\rangle \cdots \Longrightarrow \langle q_n; \sigma_n\rangle \cdots$ that goes on forever; motivate your answer.

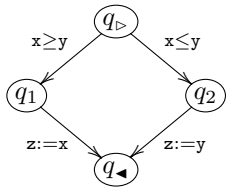
(d) For each of the two program graphs determine whether or not there is a program in the Guarded Command language (Section 2.1) that gives rise to the program graph; motivate your answer by giving the program in case of an affirmative answer and an explanation in case of a negative answer.

Program Analysis (10%)

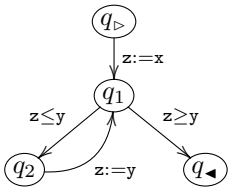
These questions relate to the lecture material *Formal Methods: An Appetizer* as used on the course and to the program graphs AA and BB in Figure 1. We shall assume that `x` is initially any integer but that `y` is positive and `z` is 0.

(e) What is the set of abstract memories (Section 4.1) corresponding to this assumption? (Write the signs in the order `x`, `y` and `z` so that we may write `-+0` to indicate that `x` is negative, `y` is positive, and `z` is 0.)

(f) For each of the two program graphs construct an analysis assignment (Section 4.2) $\mathbf{A}$ based on $\mathbf{A}(q_\triangleright)$ being the set of abstract memories of the previous question and that is not larger than necessary.



Program Graph AA



Program Graph BB

Figure 1: The two program graphs AA and BB.

(g) Determine for each of the two programs whether or not your analysis assignment allows you to conclude that the final value of `z` is positive.

Model Checking (10%)

These questions relate to the lecture material *Formal Methods: An Appetizer* as used on the course and to the program graphs AA and BB in Figure 1. Suppose that `x`, `y`, and `z` can only take values in the set $\{1, 2, 3\}$ and let a memory mention the values of the variables in the order `x`, `y`, and `z`; then $\#_{132}$ means that `x` is 1, `y` is 3, and `z` is 2.

(h) For each of the two programs determine the number of states in the transition system obtained from the program graph and memories (Section 5.4).

(i) For each of the two programs determine whether or not the CTL formula

$$(\rhd \wedge \#_{321}) \Rightarrow \exists \diamond (\#_{323} \wedge \blacktriangleleft)$$

holds (Section 5.3) of the transition system obtained from the program graph and memories; motivate your answer (in the manner of either Section 5.2 or 5.3).

(j) For each of the two programs determine whether or not the CTL formula

$$(\rhd \wedge \#_{321}) \Rightarrow \forall \diamond (\#_{323} \wedge \blacktriangleleft)$$

holds (Section 5.3) of the transition system obtained from the program graph and memories; motivate your answer (in the manner of either Section 5.2 or 5.3).

Exercise 4 (10%)

These questions relate to the lecture material *Semantics with Applications: An Appetizer* as well as *Formal Methods: An Appetizer*.

Consider the following version of the Guarded Command language (used in *Formal Methods: An Appetizer*) given by

$$\begin{array}{rcl} C & ::= & x := \texttt{a} \mid \texttt{skip} \mid \texttt{C}_1 \,;\, \texttt{C}_2 \mid \texttt{if GC fi} \mid \texttt{do GC od} \\ GC & ::= & b \rightarrow \texttt{C} \mid \texttt{GC}_1 \,[]\, \texttt{GC}_2 \end{array}$$

where arithmetic expressions a and boolean expressions b now are as in *Semantics with Applications: An Appetizer*.

(a) Write a natural semantics (in the style of Section 2.1 of *Semantics with Applications: An Appetizer*) for the language of Guarded Commands. You should be defining transitions for commands

$$\langle C, \sigma \rangle \longrightarrow \sigma'$$

and transitions for guarded commands

$$\langle GC, \sigma \rangle \longrightarrow \sigma'$$

(Hint: It may be helpful to still use the notation $\mathsf{done}(b)$ from *Formal Methods: An Appetizer*).

(b) Consider a command C and the program graph PG generated from it as in Chapter 2 of *Formal Methods: An Appetizer*.

If we have a finite execution sequence

$$\langle q_{\triangleright}; \sigma_0 \rangle \Longrightarrow^* \langle q_{\blacktriangleleft}; \sigma_n \rangle$$

how is that reflected in your natural semantics?

If we have an infinite execution sequence

$$\langle q_{\triangleright}; \sigma_0 \rangle \Longrightarrow \langle q_1; \sigma_1 \rangle \Longrightarrow \langle q_2; \sigma_2 \rangle \cdots \Longrightarrow \langle q_n; \sigma_n \rangle \cdots$$

how is that reflected in your natural semantics?

Exercises on Context-free Languages

Exercise 5 (30%)

Go is a programming language created at Google to easily build concurrent programs. This exercise is based on a small subset of the language that we call here tinyGo, whose syntax is given by the following grammar:

S	$\rightarrow$	`[`P`]`	(scoped program)
P	$\rightarrow$	A	(actions)
	$\mid$	P `;` P	(sequential composition of programs)
	$\mid$	P `+` P	(non-deterministic choice between programs)
A	$\rightarrow$	`skip`	(do nothing)
	$\mid$	C`!`	(output on a channel)
	$\mid$	C`?`	(input on a channel)
	$\mid$	`go` P	(spawn a parallel program)
	$\mid$	S	(scoped program)
C	$\rightarrow$	`tick` $\mid$ `tack`	(finite set of channels)

where the set of non-terminal symbols (or variables) is $\{S, P, A, C\}$, the set of terminal symbols (or tokens) is $\{$`[`, `]`, `;`, `+`, `skip`, `!`, `?`, `go`, `tick`, `tack`$\}$, and the initial symbol is S.

(a) Show that the following program is accepted by the grammar of tinyGo by providing a parse tree for it:

```
[ tick? ; go tick! ]
```

(b) Show that the grammar is ambiguous by providing a tinyGo program with two distinct parse trees.

(c) Enumerate all sources of ambiguities very briefly (1-3 sentences).

(d) Provide an unambiguous grammar for tinyGo. Your grammar should accept exactly those programs accepted by the original grammar. Explain how you transformed the grammar to obtain a new one. Hint: You can obtain the new grammar by applying the transformations seen during the course and in the mandatory assignment. Show that the program provided as a solution to (b) has a unique parse tree in the new grammar.

(e) Consider again the original grammar of tinyGo described at the beginning of the exercise. Show that the grammar is not an LL(1) by providing an example of a valid program where an LL(1) parser would not be able to choose a production based on the lookahead token.

(f) Consider the following alternative grammar for tinyGo

$$
\begin{array}{lcl}
S & \rightarrow & \texttt{[}P\texttt{]} \\
P & \rightarrow & A \mid A \texttt{ ; } P \mid A \texttt{ + } P \\
A & \rightarrow & \texttt{skip} \mid C\texttt{!} \mid C\texttt{?} \mid \texttt{go } P \mid S \\
C & \rightarrow & \texttt{tick} \mid \texttt{tack}
\end{array}
$$

Apply the transformations seen in class (e.g. left-factorisation) to obtain an LL(1) grammar. Show that the grammar is LL(1) by providing a deterministic parsing table for it, where rows correspond to non-terminal symbols and columns correspond to terminal symbols. In the cell corresponding to non-terminal X and terminal y you should write the production that parser should choose if it is trying to parse an expression generated by X and the lookahead symbol is y. You can use a copy of the following template for providing your solution:

	`;`	`+`	`skip`	`!`	`?`	`go`	`[`	`]`	`tick`	`tack`
S										
P										
A										
C										

(g) Consider the subset of tinyGo programs that are *deterministic* and *sequential*. These are programs generated by P in our original grammar which do not contain the tokens `+`, `go`, `[` and `]`. Such language can be described with the following grammar (with start symbol P):

$$
\begin{array}{lcl}
P & \rightarrow & A \mid P \texttt{ ; } P \\
A & \rightarrow & \texttt{skip} \mid C\texttt{!} \mid C\texttt{?} \\
C & \rightarrow & \texttt{tick} \mid \texttt{tack}
\end{array}
$$

We are now interested in a subset of such language that we call *balanced programs*. Those are programs that contain the same number of input and output operations on each channel. For example, the program

```
tick! ; tack? ; tick? ; tack! ; tick! ; tick?
```

is balanced the number of times `tick` is used as input and output is the same (2) and the number of times `tack` is used as input and output is also the same (1). Instead program

```
tick! ; tick? ; tick?
```

is *not* a balanced program since the program performs two read operations on channel `tick` but it performs only one output on the same channel. Show that the set of balanced programs is a context-free language by providing a context-free grammar for such programs.