Technical University of Denmark

Written examination, May 24, 2018

Page 1 of 14 pages

Course number: 02141

Aids allowed: All written aids are permitted

Exam duration: 4 hours

Weighting: 7-step scale

Answers Included

Exercises on Formal Methods

Exercise 1 (25%)

Consider the program graph in Figure 1.

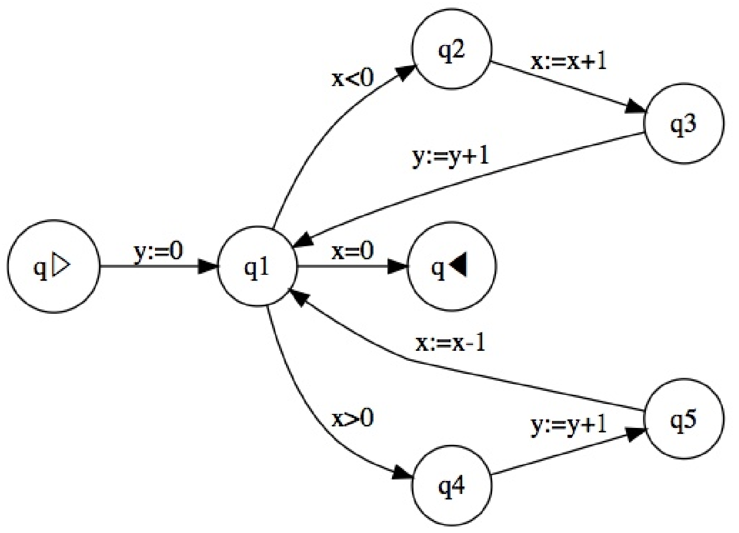


Figure 1: Program Graph for computing the absolute value.

A: Suppose that the initial memory $\sigma_\triangleright$ has $\sigma_\triangleright(\mathrm{x}) = -4$ and $\sigma_\triangleright(\mathrm{y}) = 27$ and imagine a complete execution sequence

$$\langle q_\triangleright ; \sigma_\triangleright \rangle \overset{k}{\Longrightarrow} \langle q_\blacktriangleleft ; \sigma_\blacktriangleleft \rangle$$

A1: What is the final memory $\sigma_\blacktriangleleft$?
A2: How many steps k were needed?
A3: Is the program graph deterministic?
A4: Is the program graph evolving?

Answer (4 %):
A1: $\sigma_\blacktriangleleft(\mathrm{x}) = 0$, $\sigma_\blacktriangleleft(\mathrm{y}) = 4$
A2: $k = 14$
A3: Yes, according to Proposition 1.26, because no node has a memory where more than one outgoing edge can be taken.
A4: Yes, according to Proposition 1.29, because no non-final node has a memory where none of the outgoing edges can be traversed.

B: Is there a program in Guarded Commands that gives rise to the program graph in Figure 1? If your answer is yes you should write the program in Guarded Commands. If your answer is no you should argue why this cannot be the case.

Answer (5 %): I will accept two different answers.
The first one says yes and gives the program:

```
y:=0;
do x<0 -> x:=x+1; y:=y+1
   []
   x>0 -> y:=y+1; x:=x-1
od
```

The other one says no and argues that $\mathtt{x} = \mathtt{0}$ does not have the form of $\neg(\mathtt{x} > \mathtt{0}) \wedge \neg(\mathtt{x} < \mathtt{0})$ as constructed by the edges function (although it is semantically equivalent to it).

C: Recall that the absolute value $\mathsf{abs}(z)$ is defined by

$$\mathsf{abs}(z) = \begin{cases} z & \text{if } z \geq 0 \\ -z & \text{if } z < 0 \end{cases}$$

Complete the following incomplete partial predicate assignment

$$\begin{array}{rcl} \mathbf{P}(q_\triangleright) & = & \mathtt{x} = \underline{\mathtt{n}} \\ \mathbf{P}(q_1) & = & \cdots \\ \mathbf{P}(q_\blacktriangleleft) & = & \mathtt{y} = \mathsf{abs}(\underline{\mathtt{n}}) \end{array}$$

into one that is correct. Furthermore, argue that it is correct.

Answer (8 %):

$$\begin{array}{rcl} \mathbf{P}(q_\triangleright) & = & \mathtt{x} = \underline{\mathtt{n}} \\ \mathbf{P}(q_1) & = & \mathsf{abs}(\underline{\mathtt{n}}) = \mathtt{y} + \mathsf{abs}(\mathtt{x}) \\ \mathbf{P}(q_\blacktriangleleft) & = & \mathtt{y} = \mathsf{abs}(\underline{\mathtt{n}}) \end{array}$$

We need to consider 4 short path fragments:

- $q_\triangleright$ $\mathtt{y} := \mathtt{0}$ $\mathtt{q}_1$**: From $\mathtt{x} = \underline{\mathtt{n}}$ we get $\mathsf{abs}(\underline{\mathtt{n}}) = \mathsf{abs}(\mathtt{x})$ and because $\mathtt{y} = \mathtt{0}$ we also get $\mathsf{abs}(\underline{\mathtt{n}}) = \mathtt{y} + \mathsf{abs}(\mathtt{x})$.**
- q_1 $\mathtt{x} = \mathtt{0}$ $\mathtt{q}_\blacktriangleleft$**: From $\mathsf{abs}(\underline{\mathtt{n}}) = \mathtt{y} + \mathsf{abs}(\mathtt{x})$ and $\mathtt{x} = \mathtt{0}$ we get $\mathsf{abs}(\underline{\mathtt{n}}) = \mathtt{y}$.**
- q_1 $\mathtt{x} > \mathtt{0}$ $\mathtt{y} := \mathtt{y} + \mathtt{1}$ $\mathtt{x} := \mathtt{x} - \mathtt{1}$ $\mathtt{q}_1$**: The net effect is to increment $\mathtt{y}$ by 1 and to decrement $\mathsf{abs}(\mathtt{x})$ by 1 and therefore the invariant is maintained.**
- q_1 $\mathtt{x} < \mathtt{0}$ $\mathtt{x} := \mathtt{x} + \mathtt{1}$ $\mathtt{y} := \mathtt{y} + \mathtt{1}$ $\mathtt{q}_1$**: The net effect is to increment $\mathtt{y}$ by 1 and to decrement $\mathsf{abs}(\mathtt{x})$ by 1 and therefore the invariant is maintained.**

D: Suppose that the initial value of `x` is positive and that the initial value of `y` is positive and consider the detection of signs analysis.

D1: Construct the analysis assignment as computed by the chaotic iteration algorithm; for succinctness you may write $(+,-)$ for the abstract memory $\widehat{\sigma}$ that has $\widehat{\sigma}(\mathtt{x}) = +$ and $\widehat{\sigma}(\mathtt{y}) = -$.

Answer (6 %):

$$\begin{array}{rcl}
\mathbf{A}(q_{\triangleright}) & = & \{(+,+)\} \\
\mathbf{A}(q_1) & = & \{(-,+),(0,+),(+,0),(+,+)\} \\
\mathbf{A}(q_2) & = & \{(-,+)\} \\
\mathbf{A}(q_3) & = & \{(-,+),(0,+),(+,+)\} \\
\mathbf{A}(q_4) & = & \{(+,0),(+,+)\} \\
\mathbf{A}(q_5) & = & \{(+,+)\} \\
\mathbf{A}(q_{\blacktriangleleft}) & = & (0,+)\}
\end{array}$$

D2: In case $\mathbf{A}(q_2)$ and $\mathbf{A}(q_3)$ are non-empty you should explain why this is so.

Answer (2 %): The analysis does not distinguish 1 from any other positive number, so when a positive `x is decremented by 1, there is no way to argue that it cannot become negative.`

Exercise 2 (15%)

Consider the following program C in Guarded Commands:

```
i:=0;
j:=0;
do  i<n -> A[i]:=A[i]+3; i:=i+1
 [] j<m -> B[j]:=B[j]*B[j]; j:=j+1
od
```

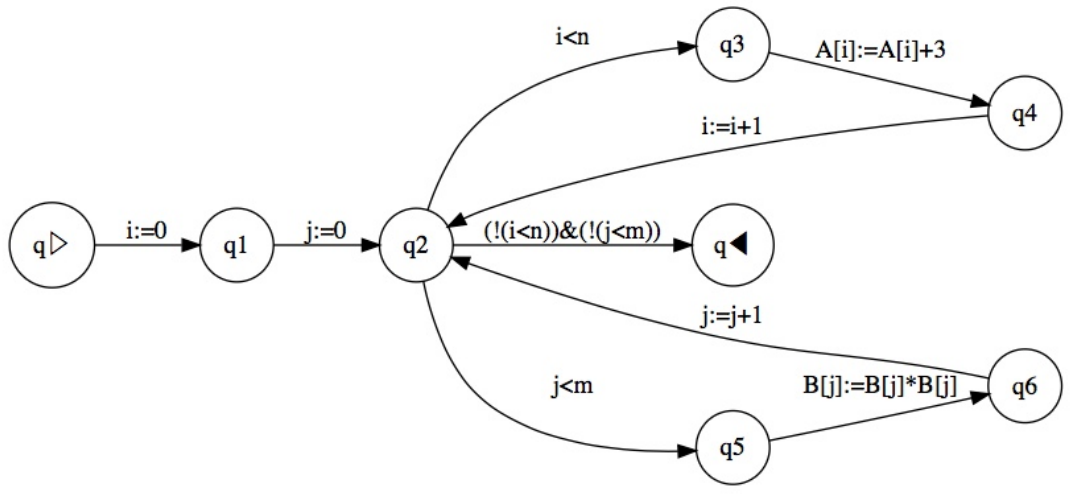
Here `A` is intended to be an array of length `n` and `B` is intended to be an array of length `m`.

A: Draw the program graph as constructed by

$$\mathbf{edges}(q_{\triangleright} \rightsquigarrow q_{\blacktriangleleft})[\![C]\!]$$

of Chapter 2.

Answer (5 %):

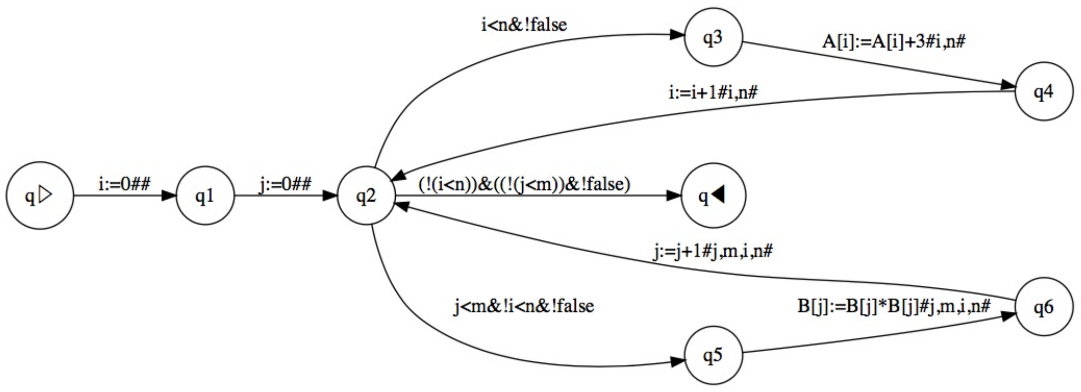


B: Draw the instrumented program graph as constructed by

$$\mathbf{edges_s}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![C]\!](\{\,\})$$

of Chapter 5.

Answer (5 %): (For typographical reasons writing $\{\cdots\}$ as $\#\cdots\#$.)



It is essential that the program graph has been made deterministic and that the implicit flows are correctly stated.

C: Suppose that the permissible information flow is given by $\{\mathtt{i},\mathtt{A},\mathtt{n}\} \rightrightarrows \{\mathtt{i},\mathtt{A},\mathtt{n}\}$ and $\{\mathtt{j},\mathtt{B},\mathtt{m}\} \rightrightarrows \{\mathtt{j},\mathtt{B},\mathtt{m}\}$.

Is the program secure, i.e. what is the value of $\mathbf{sec}[\![C]\!](\{\,\})$?

You should motivate your answer (but do not need to exhibit the detailed computation).

Answer (5 %): The computation of sec$[\![C]\!](\{\,\})$ is going to check all the explicit and implicit flows made visible on the edges in the instrumented program graph. Looking at the edge with the action `j:=j+1`$\{$`j,m,i,n`$\}$ it is clear that there is an implicit flow `i`$\rightarrow$`j` that is not permitted.

Exercises on Context-free Languages

Exercise 3 (30%)

The exercises in this part of the exam are based on PingPong, a programming language whose syntax is given by the following context-free grammar. The productions are

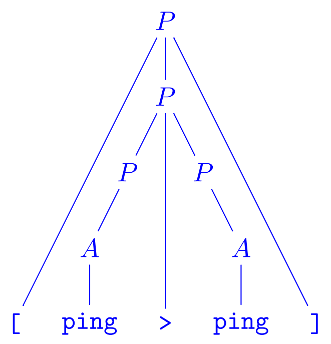
P	$\rightarrow$	A	(actions)
	$\mid$	P `>` P	(sequential composition of programs)
	$\mid$	P `+` P	(non-deterministic choice between programs)
	$\mid$	P `*` P	(parallel composition of two programs)
	$\mid$	`[`P`]`	(scoped program)
A	$\rightarrow$	`ping`	(do ping)
	$\mid$	`pong`	(do pong)

the set of non-terminal symbols is $N = \{P, A\}$, the set of terminal symbols is $T = \{$`[`, `]`, `>`, `+`, `*`, `ping`, `pong`$\}$, and the initial symbol is P.

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

`[ ping > pong ]`

Solution: the (unique) parse tree is:



(b) Show that the grammar is ambiguous by providing one PingPong program and two distinct parse trees for it.

Solution: the key observation is that the three binary operators to combine programs give raise to ambiguities due to associativity and precedence. A simple example is

`ping > pong + ping`

Two distinct parse trees can be given, each encoding different associativity or precedence order between sequential composition and non-deterministic choice. The parse trees would arise from the following different groupings (denoted with parentheses here):

```
(ping > pong) + ping          ping > (pong + ping)
```

(c) Consider the following associativity and precedence rules:

- `>` has the highest priority and associates to the right;
- `+` has priority over `*` and associates to the right;
- `*` has the lowest priority and associates to the left.

which of the two parse trees you provided in (b) adheres to these rules?

Solution: It depends on the answer to (b). For example, for the solution example we provided for (b), only the parse tree on the left adheres to the rules.

(d) Consider again the rules of (c). As explained in class, many grammar languages used by parser generators allow you to specify those rules in the grammar specification. Most likely, this is what you did in the mandatory assignment. Describe very briefly the grammar annotations that would implement the rules of (c) in one of the two grammar languages we saw in class (FsLexYacc or ANTLR4). Please specify which grammar language you have chosen. You don't need to give the entire grammar specification but just the part where the rules are specified.

Solution:

In FsLexYacc associativity can be specified with "`%left op`" statements for each of the operator tokens `op`, and precedence can be specified by the order of such statements (from low to high):

```
%left '*'
%right '+'
%right '>'
```

In ANTLR4 associativity can be specified with `<assoc=left/right>` annotations in the productions, while precedence can be specified by the order of the productions:

```
P : <assoc=right> P '>' P
  | <assoc=right> P '+' P
  | <assoc=left>  P '*' P
```

(e) Consider again the rules of (c). Transform the grammar of PingPong to encode the rules directly in the grammar. You can apply the techniques seen in class, i.e. enforce associativity by forbidding recursion on one of the sides, and enforce precedence by stratifying the grammar into precedence layers.

Solution: A possible solution is:

$$\begin{array}{lcl} P & \to & P \texttt{ * } Q \mid Q \\ Q & \to & R \texttt{ + } Q \mid R \\ R & \to & A \texttt{ > } R \mid \texttt{[}P\texttt{]} \texttt{ > } R \mid \texttt{[}P\texttt{]} \mid A \\ A & \to & \texttt{ping} \mid \texttt{pong} \end{array}$$

(f) Construct a Pushdown Automaton (PDA) that accepts the same language as the the grammar of PingPong described at the beginning of the exercise. Use the construction that we saw in class to translate a grammar into an equivalent PDA.

Solution: The PDA obtained accepts by empty stack and is defined by the tuple $(\{q\}, T, T \cup N, \delta, P, \{q\})$ where the transition function δ is defined as follows:

$$\begin{array}{rcll} \delta(q, P) & = & \{(q, A), (q, P\texttt{>}P), (q, P\texttt{+}P), (q, P\texttt{*}P), (q, \texttt{[}P\texttt{]})\} & \\ \delta(q, A) & = & \{(q, \texttt{ping}), (q, \texttt{pong})\} & \\ \delta(q, t, t) & = & \{(q, \epsilon)\} & \text{if } t \in T. \end{array}$$

A graphical representation of the PDA using the notation seen in class is also a valid answer.

(g) Use the PDA you have constructed to show that the example of (a) is in the language of the PDA. Show the sequence of configurations (instantaneous descriptions) of the PDA starting with the initial one $(q, \texttt{[ping > pong]}, P)$, where q is the initial state of your PDA, `[ ping > pong ]` is the input and P is the initial stack.

Solution:

$$\begin{array}{ll} & (\text{q}, \texttt{[ping > pong]}, \text{P}) \\ \vdash & (\text{q}, \texttt{[ping > pong]}, \texttt{[P]}) \\ \vdash & (\text{q}, \texttt{ping > pong]}, \text{P]}) \\ \vdash & (\text{q}, \texttt{ping > pong]}, \text{P > P]}) \\ \vdash & (\text{q}, \texttt{ping > pong]}, \text{A > P]}) \\ \vdash & (\text{q}, \texttt{ping > pong]}, \texttt{ping} \text{ > P]}) \\ \vdash & (\text{q}, \texttt{> pong]}, \text{> P]}) \\ \vdash & (\text{q}, \texttt{pong]}, \text{P]}) \\ \vdash & (\text{q}, \texttt{pong]}, \text{A]}) \\ \vdash & (\text{q}, \texttt{pong]}, \texttt{pong} \text{]}) \\ \vdash & (\text{q}, \texttt{]}, \text{]}) \\ \vdash & (\text{q}, ,) \end{array}$$

(h) Design a set of datatypes that are suitable to store abstract representations (ASTs) of PingPong programs. You can take inspiration from your solution to the mandatory assignment, i.e. you can use F# types or Java classes. Show how the example program

```
[ ping > pong ] * [ ping + pong ]
```

would be represented as a value of your datatype.

Solution: In F# we could define the following types:

```
type P =
  | a of A
  | seq of (P * P)
  | choice of (P * P)
  | par of (P * P)
type A =
  | ping
  | pong
```

and the program would by represented by

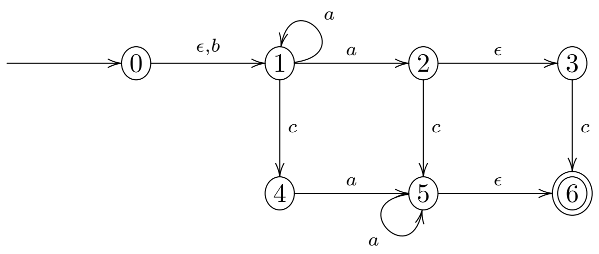
```
par(seq(a(ping),a(pong)),choice(a(ping),a(pong)))
```

Regular languages

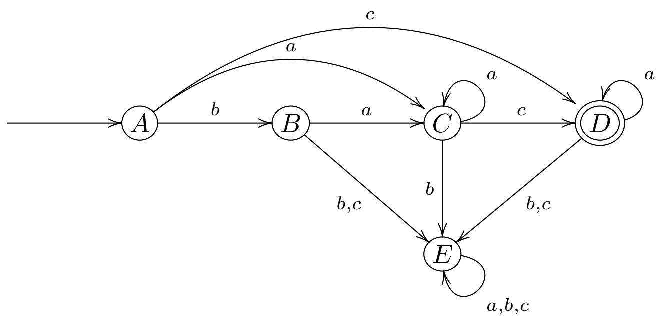
Exercise 4 (30%)

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

- L_1 is given by the ϵ-NFA:



- L_2 is given by the regular expressions $(\epsilon + b)(a + c)^*$
- L_3 is given by the DFA:



- L_4 is given by $\{ba^n ca^m \mid n, m \geq 0, n \geq m\}$
- L_5 is given by the regular expression $(\epsilon + (b + \epsilon)aa^*)ca^*$

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	c	bc	bac	bca	aac	$acaca$
L_1							
L_2							
L_3							
L_4							
L_5							

Answer (6%): The table is as follows:

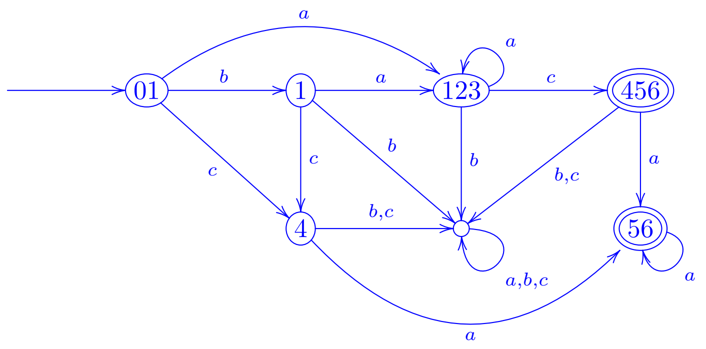
w	ϵ	c	bc	bac	bca	aac	$acaca$
L_1	*no*	*no*	*no*	*yes*	*yes*	*yes*	*no*
L_2	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*
L_3	*no*	*yes*	*no*	*yes*	*no*	*yes*	*no*
L_4	*no*	*no*	*yes*	*yes*	*no*	*no*	*no*
L_5	*no*	*yes*	*no*	*yes*	*no*	*yes*	*no*

(b) Use the subset construction algorithm (Hopcroft, Motwani and Ullman chapter 2) to construct a DFA corresponding to the ϵ-NFA for L_1.

Answer (6%): We use the subset construction algorithm of Section 2.3.5 extended to eliminate the ϵ-transitions as described in Section 2.5.5. The initial state is $ECLOSE(\{0\})$ which amounts to $\{0,1\}$. The transition relation is computed as follows:

S	a	b	c
$\{0,1\}$	$\{1,2,3\}$	$\{1\}$	$\{4\}$
$\{1,2,3\}$	$\{1,2,3\}$	$\{\ \}$	$\{4,5,6\}$
$\{1\}$	$\{1,2,3\}$	$\{\ \}$	$\{4\}$
$\{4\}$	$\{5,6\}$	$\{\ \}$	$\{\ \}$
$\{4,5,6\}$	$\{5,6\}$	$\{\ \}$	$\{\ \}$
$\{5,6\}$	$\{5,6\}$	$\{\ \}$	$\{\ \}$
$\{\ \}$	$\{\ \}$	$\{\ \}$	$\{\ \}$

The final states are those containing 6, that is, the states $\{4,5,6\}$ and $\{5,6\}$. Pictorially (not required):



(c) Prove that L_4 is not a regular language.

Answer (6%): We use the Pumping Lemma directly. So assume that L_4 is regular and consider $w = ba^Nca^N \in L_4$ where N is the constant given by the Pumping Lemma. Then there exists x, y and z such that $w = xyz$, $|xy| \leq N$, $y \neq \epsilon$ and for all $k \geq 0$ also

$xy^kz \in L_4$. **Clearly it must be the case that the** c **of** w **is in the substring** z **so the string** xz **will have fewer than** N **occurrences of** a **before the** c **which contradicts that** $xz \in L_4$**,**

(d) 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_5
L_1	*yes*				
L_2		*yes*			
L_3			*yes*		
L_4				*yes*	
L_5					*yes*

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

Answer (12%):

$\subseteq$	L_1	L_2	L_3	L_4	L_5
L_1	***yes***	*yes*[3]	*no*[6]	*no*[4]	*no*[6]
L_2	*no*[3]	***yes***	*no*[5]	*no*[2]	*no*[5]
L_3	*no*[6]	*yes*[5]	***yes***	*no*[7]	*yes*[1]
L_4	*no*[4]	*yes*[2]	*no*[7]	***yes***	*no*[7]
L_5	*no*[6]	*yes*[5]	*yes*[1]	*no*[7]	***yes***

(1) $L_3 = L_5$. **We can convert the DFA of** L_3 **into a regular expression. Formally, this can be done using the elimination algorithm of Section 3.2.1 that gives** $((ba + a)a^*c + c)a^*$ **(first eliminate** B**, then** C**). Rewriting this gives us that** $L_3 = L_5$**.**

(2) $L_4 \subseteq L_2$ **but** $L_2 \not\subseteq L_4$**. We observe that** L_4 **is a subset of** ba^*ca^* **and that** $ba^*ca^* \subseteq b(a + c)^* \subseteq L_2$ **so** $L_4 \subseteq L_2$**. However** $L_2 \not\subseteq L_4$ **because** $\epsilon \in L_2$ **but** $\epsilon \notin L_4$**.**

(3) $L_1 \subseteq L_2$ **but** $L_2 \not\subseteq L_1$**. We can convert the** ϵ**-NFA of** L_1 **(or the corresponding DFA constructed in exercise 1b)into a regular expression using the elimination algorithm of Section 3.2.1 and we get** $(\epsilon + b)a^*(ca + ac)a^*$ **(after some rewritings) which is a subset of** $(\epsilon + b)(a + c)^*$ **so** $L_1 \subseteq L_2$**. However** $L_2 \not\subseteq L_1$ **because** $\epsilon \in L_2$ **but** $\epsilon \notin L_1$**.**

(4) $L_4 \not\subseteq L_1$ **but** $L_1 \not\subseteq L_4$**. We have** $L_4 \not\subseteq L_1$ **since** $bc = ba \in L_4$ **but** $bc \notin L_1$**. Also** $L_1 \not\subseteq L_4$ **since** $ca \in L_1$ **but** $ca = a^0ca^1 \notin L_4$ **(violates** $0 \geq 1$**)**

(5) $L_3 \subseteq L_2$ **but Ł$_2$** $\not\subseteq L_3$**. Recall that** $L_3 = L_5 = (\epsilon + (b + \epsilon)aa^*)ca^*$ **and** $L_2 = (\epsilon + b)(a + c)^*$ **and it is easy to see that** $L_3 \subseteq L_2$**; however Ł$_2$** $\not\subseteq L_3$ **as** $bca \in L_2$ **but** $bca \notin L_3$**. The same holds when replacing** L_3 **with** L_5**.**

(6) $L_1 \not\subseteq L_3$ **and** $L_3 \not\subseteq L_1$**. We have** $bca \in L_1$ **but** $bca \notin L_3$ **so** $L_1 \not\subseteq L_3$**; also** $L_3 \not\subseteq L_1$ **because** $c \in L_3$ **but** $c \notin L_1$**. The same holds when replacing** L_3 **with** L_5**.**

(7) $L_4 \not\subseteq L_3$ **and** $L_3 \not\subseteq L_4$**. We have** $bc \in L_4$ **but** $bc \notin L_3$ **so** $L_4 \not\subseteq L_3$**; also** $L_3 \not\subseteq L_4$ **as strings in** L_3 **do not have to start with** b **so** $c \in L_3$ **but** $c \notin L_4$**. The same holds when replacing** L_3 **with** L_5**.**

Technical University of Denmark

Written examination, May 24, 2019

Course number: 02141

Aids allowed: All written aids are permitted

Exam duration: 4 hours

Weighting: 7-step scale

Answers Included

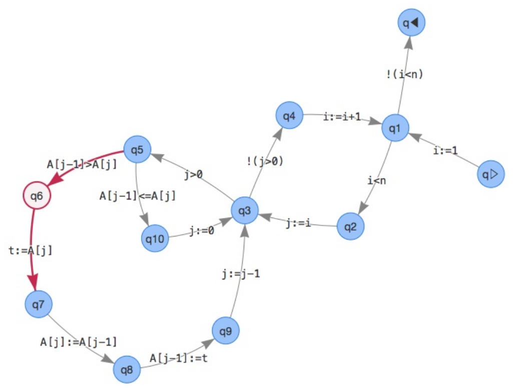
Exercises on Formal Methods

Exercise 1 (15%) Semantics and Program Analysis

Consider the following Guarded Commands program in the notation of the on-line system at FormalMethods.dk:

```
i:=1;
do i<n ->
   j:=i;
   do j>0 ->
          if A[j-1]>A[j] -> t:=A[j];
                            A[j]:=A[j-1];
                            A[j-1]:=t;
                            j:=j-1
          [] A[j-1]<=A[j] -> j:=0
          fi
   od;
   i:=i+1
od
```

1a Draw the program graph for this program. How many nodes does it have?



Solution:

12.

1b What is the final configuration in case the initial configuration of the execution sequence is $\langle q_\triangleright; [\mathtt{i} = 0, \mathtt{j} = 0, \mathtt{n} = 2, \mathtt{t} = 0, \mathtt{A} = [4, 3, 2, 1]]\rangle$? (You do *not* have to show all the intermediate configurations.)

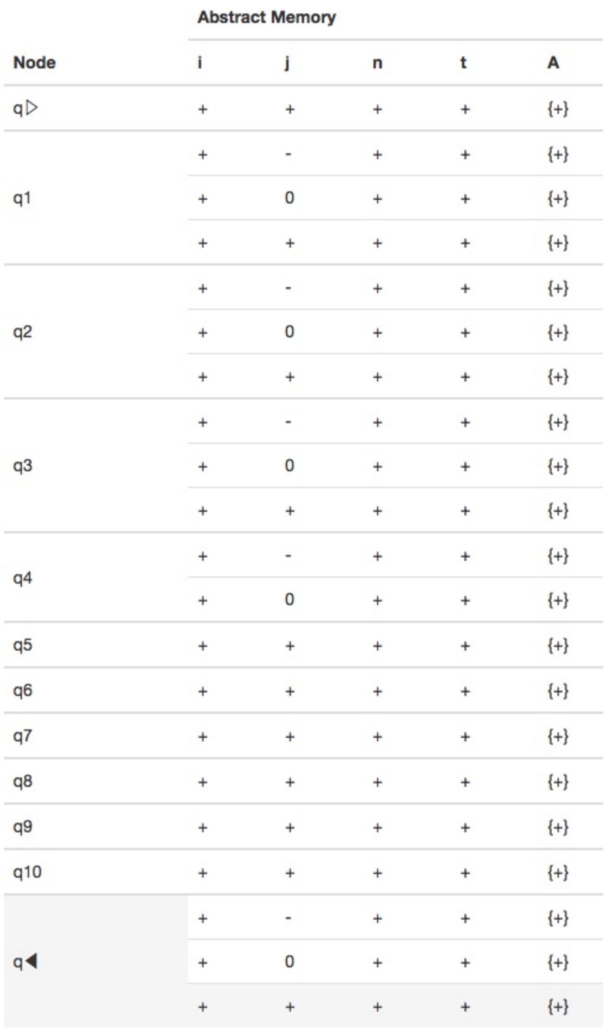
Solution: $\langle q_\blacktriangleleft; [\mathtt{i} = 2, \mathtt{j} = 0, \mathtt{n} = 2, \mathtt{t} = 3, \mathtt{A} = [3, 4, 2, 1]]\rangle$.

1c What is the final configuration in case the initial configuration of the execution sequence is $\langle q_\triangleright; [\mathtt{i} = 0, \mathtt{j} = 0, \mathtt{n} = 3, \mathtt{t} = 0, \mathtt{A} = [4, 3, 2, 1]]\rangle$? (You do *not* have to show all the intermediate configurations.)

Solution: $\langle q_{\blacktriangleleft}; [\mathtt{i} = 3, \mathtt{j} = 0, \mathtt{n} = 3, \mathtt{t} = 2, \mathtt{A} = [2, 3, 4, 1]]\rangle$.

1d Compute the analysis assignment $\mathbf{A}$ when there is just one initial abstract memory as given by $\mathbf{A}(q_{\triangleright}) = \{[\mathtt{i} = +, \mathtt{j} = +, \mathtt{n} = +, \mathtt{t} = +, \mathtt{A} = \{+\}]\}$. (Hint: make abbreviations like $[+ + + + \{+\}]$ for the abstract memory above, in order not to have to write too much.)

What is the set of abstract memories at the final node, i.e. what is $\mathbf{A}(q_{\blacktriangleleft})$?



	Abstract Memory				
Node	i	j	n	t	A
q▷	+	+	+	+	{+}
q1	+	-	+	+	{+}
	+	0	+	+	{+}
	+	+	+	+	{+}
q2	+	-	+	+	{+}
	+	0	+	+	{+}
	+	+	+	+	{+}
q3	+	-	+	+	{+}
	+	0	+	+	{+}
	+	+	+	+	{+}
q4	+	-	+	+	{+}
	+	0	+	+	{+}
q5	+	+	+	+	{+}
q6	+	+	+	+	{+}
q7	+	+	+	+	{+}
q8	+	+	+	+	{+}
q9	+	+	+	+	{+}
q10	+	+	+	+	{+}
q◀	+	-	+	+	{+}
	+	0	+	+	{+}
	+	+	+	+	{+}

Solution:

$$\mathbf{A}(q_{\blacktriangleleft}) = \left\{ \begin{array}{l} [\mathtt{i} = +, \mathtt{j} = -, \mathtt{n} = +, \mathtt{t} = +, \mathtt{A} = \{+\}] \\ [\mathtt{i} = +, \mathtt{j} = 0, \mathtt{n} = +, \mathtt{t} = +, \mathtt{A} = \{+\}] \\ [\mathtt{i} = +, \mathtt{j} = +, \mathtt{n} = +, \mathtt{t} = +, \mathtt{A} = \{+\}] \end{array} \right\}.$$

Exercise 2 (15%) Information Flow

Consider the following program C in Guarded Commands:

```
m:=0;
d:=0;
do  m<10000 -> M[m]:=M[m]+27; m:=m+1
 [] d<20000 -> D[d]:=(D[d]*103)/100; d:=d+1
od
```

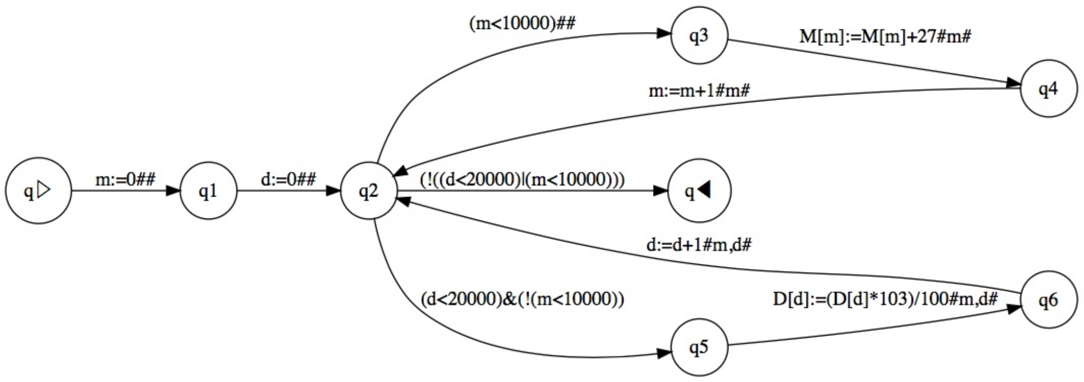
Here `M` is intended to be an array of length `10000` and `D` is intended to be an array of length `20000`. (This may be seen as a cloud provider updating costs with a fixed value for one company and with a percentage for another company.)

2a Draw the instrumented program graph as constructed by

$$\mathbf{edges_s}(q_{\triangleright} \rightsquigarrow q_{\blacktriangleleft})[\![C]\!](\{\,\})$$

of Formal Methods an Appetizer Section 5.2.

Solution: (For typographical reasons writing $\{\cdots\}$ as $\#\cdots\#$.)



It is essential that the program graph has been made deterministic and that the implicit flows are correctly stated; it is OK to do a bit of simplification of the boolean expressions on the edges.

Preparing for the next questions let us choose a security lattice with elements clean, Mcompany, Dcompany, corrupted partially ordered by

$$\text{clean} \sqsubset \text{Mcompany} \sqsubset \text{corrupted} \qquad \text{clean} \sqsubset \text{Dcompany} \sqsubset \text{corrupted}$$

in the manner of Formal Methods an Appetizer Section 5.4.

For each of the following security associations determine whether or not the security analysis $\mathbf{sec}[\![C]\!](\{\,\})$ of Formal Methods an Appetizer Section 5.3 deems the program secure.

You should motivate your answer (but do not need to exhibit the detailed computation).

2b $\mathbf{L}(\mathsf{m}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{M}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{d}) = \mathsf{Dcompany}$, $\mathbf{L}(\mathsf{D}) = \mathsf{Dcompany}$.

Solution: The computation of sec$[\![C]\!](\{\,\})$ is going to check all the explicit and implicit flows made visible on the edges in the instrumented program graph.

Looking at the edges with the actions `D[d]:=(D[d]*103)/100{d,m}` and `d:=d+1{d,m}` it is clear that there are implicit flows `m`→`d` and `m`→`D` that are not permitted.

2c $\mathbf{L}(\mathsf{m}) = \mathsf{clean}$, $\mathbf{L}(\mathsf{M}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{d}) = \mathsf{clean}$, $\mathbf{L}(\mathsf{D}) = \mathsf{Dcompany}$.

Solution: The computation of sec$[\![C]\!](\{\,\})$ is going to check all the explicit and implicit flows made visible on the edges in the instrumented program graph.

There is no edge where the explicit or implicit flow violates the security classification.

2d $\mathbf{L}(\mathsf{m}) = \mathsf{clean}$, $\mathbf{L}(\mathsf{M}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{d}) = \mathsf{Dcompany}$, $\mathbf{L}(\mathsf{D}) = \mathsf{Dcompany}$.

Solution: The computation of sec$[\![C]\!](\{\,\})$ is going to check all the explicit and implicit flows made visible on the edges in the instrumented program graph.

There is no edge where the explicit or implicit flow violates the security classification.

2e $\mathbf{L}(\mathsf{m}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{M}) = \mathsf{Mcompany}$, $\mathbf{L}(\mathsf{d}) = \mathsf{clean}$, $\mathbf{L}(\mathsf{D}) = \mathsf{Dcompany}$.

Solution: The computation of sec$[\![C]\!](\{\,\})$ is going to check all the explicit and implicit flows made visible on the edges in the instrumented program graph.

Looking at the edges with the actions `D[d]:=(D[d]*103)/100{d,m}` and `d:=d+1{d,m}` it is clear that there are implicit flows `m`→`d` and `m`→`D` that are not permitted.

Exercise 3 (15%) Deterministic Systems

This exercise compares three different notions of what might constitute a deterministic system.

- In Formal Methods an Appetizer Section 1.4 we defined a program graph PG and its semantics $\mathcal{S}$ to constitute a *deterministic system* whenever

$$\langle q_\triangleright;\sigma\rangle \overset{\omega'}{\Longrightarrow}{}^{*} \langle q_\blacktriangleleft;\sigma'\rangle \text{ and } \langle q_\triangleright;\sigma\rangle \overset{\omega''}{\Longrightarrow}{}^{*} \langle q_\blacktriangleleft;\sigma''\rangle$$
$$\text{imply that } \sigma' = \sigma'' \text{ and that } \omega' = \omega''$$

- An alternative definition might be to define a program graph PG and its semantics $\mathcal{S}$ to constitute a *locally deterministic system* whenever distinct edges with the same source node have semantic functions that are defined on non-overlapping domains.

  This can be written more formally as $\mathsf{dom}(\mathcal{S}[\![\alpha_1]\!]) \cap \mathsf{dom}(\mathcal{S}[\![\alpha_2]\!]) = \{\,\}$ whenever $(q, \alpha_1, q_1) \neq (q, \alpha_2, q_2)$.

- Yet another definition might be to define a program graph PG and its semantics $\mathcal{S}$ to constitute a *strongly deterministic system* whenever

  $$\langle q_{\triangleright}; \sigma\rangle \overset{\omega'}{\Longrightarrow}^{*} \langle q'; \sigma'\rangle \text{ and } \langle q_{\triangleright}; \sigma\rangle \overset{\omega''}{\Longrightarrow}^{*} \langle q''; \sigma''\rangle \text{ and } |\omega'| = |\omega''|$$
  $$\text{imply that } \sigma' = \sigma'' \text{ and that } \omega' = \omega''$$

  where $|\omega|$ denotes the length of ω.

It follows from Formal Methods an Appetizer Section 1.4 that the statement "a locally deterministic system is also deterministic" is always true. It also follows that the statement "a deterministic system is also locally deterministic" is sometimes false.

For each of the statements below indicate whether or not you consider them to be always true or sometimes false.

If you consider them to be always true you should provide a short explanation (but no formal proof). If you consider them to be sometimes false you should provide a simple example (that is no more complex than necessary to make the point). You may refer to results and figures in Formal Methods an Appetizer if you do so precisely.

3a "A locally deterministic system is also strongly deterministic."

Solution: This is always true.
Proposition 1.26 in Formal Methods an Appetizer shows that a locally deterministic system is also a deterministic system. The proof can be adapted to show that a locally deterministic system is also strongly deterministic system by performing mathematical induction in n where $n = |\omega'| = |\omega''|$.

3b "A deterministic system is also strongly deterministic."

Solution: This is sometimes false.
The program graph for `do true -> x:=1 [] true -> x:=2 od` is clearly not strongly deterministic, but is deterministic because it never terminates (that is, reaches $q_{\blacktriangleleft}$).

3c "A strongly deterministic system is also locally deterministic."

Solution: This is sometimes false.
An example is provided by the program graph in Figure 1.9 in Formal

Methods an Appetizer that shows a system that is (strongly) deterministic but that does not satisfy the property called locally deterministic. (Basically the idea is to have lack of nondeterminism at a node that cannot be reached with a memory for which the nondeterminism manifests itself.)

Exercises on Context-free Languages

Exercise 4 (25%)

In the lectures on model checking you were introduced to CTL, a formal language to express properties of programs. A popular alternative to CTL is *Linear-time Temporal Logic* (LTL), which was introduced by Amir Pnueli in 1977, who later received the Turing Award in 1996 for his contributions on temporal logics and verification. The next set of exercises regard the syntax of LTL as provided by the following grammar.

The productions of the grammar are

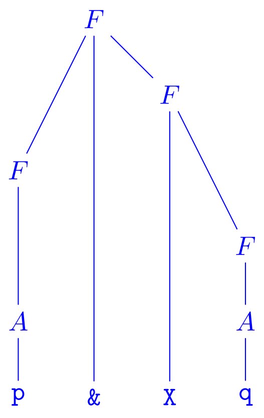
F	$\to$	$\mathtt{tt}$	(true)
	$\mid$	A	(atomic predicates)
	$\mid$	$\mathtt{!}F$	(negation)
	$\mid$	$F\ \mathtt{\&}\ F$	(conjunction)
	$\mid$	$\mathtt{X}F$	(next state)
	$\mid$	$\mathtt{G}F$	(globally)
	$\mid$	$F\ \mathtt{U}\ F$	(until)
	$\mid$	$\mathtt{[}F\mathtt{]}$	(grouping of a formula)
A	$\to$	$\mathtt{p} \mid \mathtt{q} \mid \mathtt{r}$	(atomic predicates)

the set of non-terminal symbols is $N = \{F, A\}$, the set of terminal symbols is $T = \{\mathtt{tt}, \mathtt{!}, \mathtt{\&}, \mathtt{X}, \mathtt{G}, \mathtt{U}, \mathtt{[}, \mathtt{]}, \mathtt{p}, \mathtt{q}, \mathtt{r}\}$. Strings generated by F are LTL formulas, and strings generated by A are atomic predicates.

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

```
p & X q
```

Solution: the (unique) parse tree is:



(b) Show that the grammar is ambiguous by providing one LTL formula and two distinct parse trees for it. You are not allowed to use negation (`!`) and conjunction (`&`) in the example formula.

Solution: the key observation is that the unary and binary temporal operators `X`, `G` and `U` give raise to the usual ambiguities due to precedence and associativity. A simple example of an ambiguous formula is

```
X tt U tt
```

Two distinct parse trees can be given, each encoding different precedences order between the next-state operator `X` and until `U`. The parse trees would arise from the following different groupings (denoted with parentheses here):

```
X (tt U tt)          (X tt) U tt
```

(c) Consider the following precedence and associativity rules.

- Precedence from highest to lowest for the following unary and binary operators is: `!`, `&`, `X`, `G`, `U`;
- `&` and `U` are left-associative.

which of the two parse trees you provided in (b) adheres to these rules?

Solution: It depends on the answer to (b). For example, for the solution example we provided for (b), only the parse tree on right adheres to the rules.

(d) Consider again the rules of (c). As explained in class, many grammar languages used by parser generators allow you to specify those rules in the grammar specification. Most likely, this is what you did in the mandatory assignment. Describe very briefly the grammar annotations that would implement the rules of (c) in one of the two grammar languages we saw in class (FsLexYacc or ANTLR4). Please specify which grammar language you have chosen. You don't need to give the entire grammar specification but just the part where the above precedence and associativity rules are specified.

Solution:

In FsLexYacc left-associativity can be specified with "`%left op`" statements for each of the operator tokens `op`, and precedence can be specified by the order of such statements (from low to high):

```
...
%left 'U'
%left 'G'
%left 'X'
```

```
%left '&'
%left '!'
...
```

Note: For unary operators precedence does not need to be specified (one can use `%precedence` instead of `%left`).

In ANTLR4 associativity can be specified with `<assoc=left/right>` annotations in the productions, while precedence can be specified by the order of the productions:

```
f : ...
  | '!' f
  | <assoc=left> f '&' f
  | 'X' f
  | 'G' f
  | <assoc=left> f 'U' f
  ...
```

NOTE: In the above sketched solutions, token names for each token would be used in an actual implementation.

(e) Consider again the rules of (c). Transform the grammar of LTL into a new grammar that encodes the rules directly. You can apply the techniques seen in class. For example, you can first enforce precedence by stratifying the grammar into precedence layers, and then enforce associativity by forbidding recursion on one of the sides.

Solution: A possible solution is as follows.

We first apply stratification, obtaining

$$\begin{array}{rcl}
F_0 & \to & F_0 \; \texttt{U} \; F_0 \mid F_1 \\
F_1 & \to & \texttt{G}F_1 \mid F_2 \\
F_2 & \to & \texttt{X}F_2 \mid F_3 \\
F_3 & \to & F_3 \; \texttt{\&} \; F_3 \mid F_4 \\
F_4 & \to & \texttt{!}F_4 \mid F_5 \\
F_5 & \to & \texttt{tt} \mid A \mid \texttt{[}F_0\texttt{]} \\
A & \to & \texttt{p} \mid \texttt{q} \mid \texttt{r}
\end{array}$$

Next, we enforce left-associativity on binary operators `&` and `U` by allowing recursion on the left only:

$$\begin{array}{rcl}
F_0 & \to & F_0 \; \mathtt{U} \; F_1 \mid F_1 \\
F_1 & \to & \mathtt{G} F_1 \mid F_2 \\
F_2 & \to & \mathtt{X} F_2 \mid F_3 \\
F_3 & \to & F_3 \; \mathtt{\&} \; F_4 \mid F_4 \\
F_4 & \to & \mathtt{!} F_4 \mid F_5 \\
F_5 & \to & \mathtt{tt} \mid A \mid \mathtt{[} F_0 \mathtt{]} \\
A & \to & \mathtt{p} \mid \mathtt{q} \mid \mathtt{r}
\end{array}$$

(f) Construct a Pushdown Automaton (PDA) that accepts the same language as the grammar of LTL described at the beginning of the exercise. Use the construction that we saw in class (and described in book [HMU]) to translate a context-free grammar into an equivalent, non-deterministic PDA.

Solution: The PDA obtained accepts by empty stack and is defined by the tuple $(\{q\}, T, T \cup N, \delta, P, \{q\})$ where the transition function δ is defined as follows:

$$\begin{array}{rcll}
\delta(q, \epsilon, F) & = & \{(q, \mathtt{tt}), (q, A), (q, \mathtt{!}A), (q, F\mathtt{\&}F), (q, \mathtt{X}F), (q, \mathtt{G}F), (q, F\mathtt{U}F), (q, \mathtt{[}F\mathtt{]}), \} & \\
\delta(q, \epsilon, A) & = & \{(q, \mathtt{p}), (q, \mathtt{q}), (q, \mathtt{r})\} & \\
\delta(q, t, t) & = & \{(q, \epsilon)\} & \text{if } t \in T.
\end{array}$$

A graphical representation of the PDA using the notation seen in class is also a valid answer.

(g) Show that the example formula of exercise (a) is accepted by the PDA you have constructed in exercise (f). Do so by showing the sequence of configurations (instantaneous descriptions) of the PDA starting with the initial one $(q, \mathtt{X\ tt\ U\ tt}, F)$, where q is the initial state of your PDA, `X tt U tt` is the input string and F is the initial stack symbol.

Solution:

$$\begin{array}{cl}
 & (\mathrm{q},\ \mathtt{X\ tt\ U\ tt},\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{X\ tt\ U\ tt},\ \mathtt{X}\mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{tt\ U\ tt},\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{tt\ U\ tt},\ \mathrm{F}\ \mathtt{U}\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{tt\ U\ tt},\ \mathtt{tt\ U}\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{U\ tt},\ \mathtt{U}\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{tt},\ \mathrm{F}) \\
\vdash & (\mathrm{q},\ \mathtt{tt},\ \mathtt{tt}) \\
\vdash & (\mathrm{q},\ ,\ )
\end{array}$$

(h) Design a set of datatypes that are suitable to store abstract representations (ASTs) of LTL formulas. You can take inspiration from your solution to the mandatory assignment, i.e. you can use F# types, Java classes, etc.. Show how the example formula

```
X [ p U G q ]
```

would be represented as a value of your datatype.

Solution: In F# we could define the following types:

```
type A =
       | P
       | Q
       | R
type F =
       | TT
       | A of A
       | Not of F
       | And of F * F
       | Next of F
       | Globally of F
       | Until of F * F
```

and the formula would by represented by

```
Next(Until(A P,Globally(A Q)))
```

Exercises on Regular Languages

Exercise 5 (10%)

In this question we are going to study four languages, L_1, L_2, L_3 and L_4 over the alphabet $\Sigma = \{0, 1\}$:

L_1 is described by the regular expression $(0 + 1)^*$.

L_2 is described by the context free grammar with productions

$$A \to 0A \mid A1 \mid \epsilon$$

L_3 is described by the ϵ-NFA with transition relation

	0	1
$\to \star p$	$\{q\}$	$\emptyset$
q	$\emptyset$	$\{p\}$
r	$\{r\}$	$\emptyset$

L_4 is described by the DFA with transition relation

	0	1
$\to p$	q	p
$\star q$	q	r
r	r	r

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:

	ϵ	0000	0101	0011	1100	1010
L_1						
L_2						
L_3						
L_4						

Solution:

	ϵ	0000	0101	0011	1100	1010
L_1	*yes*	*yes*	*yes*	*yes*	*yes*	*yes*
L_2	*yes*	*yes*	*no*	*yes*	*no*	*no*
L_3	*yes*	*no*	*yes*	*no*	*no*	*no*
L_4	*no*	*yes*	*no*	*no*	*yes*	*no*

Exercise 6 (20%)

Again consider the alphabet $\Sigma = \{0, 1\}$. For each of the six statements below determine whether it is true or false and explain your answer (by giving a counter example or a proof of the result).

(a) If L is a regular language and $L' \subseteq L$ then L' is a regular language.

(b) If L is a regular language then there exists a proper subset $L' \subset L$ such that L' is a regular language.

(c) If L and M are regular languages then so is $\{xy \mid x \in L \text{ and } y \notin M\}$.

(d) $\{w \mid w = w^R\}$ is a regular language – recall that w^R is the string w reversed so $(011)^R = 110$.

(e) If L is a regular language then so is $\{w \mid w \in L \text{ and } w^R \in L\}$.

(f) $\{xyx^R \mid x \in \Sigma^* \text{ and } y \in \Sigma^*\}$ is a regular language.

Solution:

(a) **false:** Σ^* is a regular language and $\{0^n1^n \mid n \geq 1\} \subseteq \Sigma^*$ but $\{0^n1^n \mid n \geq 1\}$ is not regular (Exercise 4.1.1, p. 131).

(b) **false:** : $\emptyset$ is a regular language and it has no proper subsets.

(c) **true:** When M is regular so is its complement $\overline{M}$ (Theorem 4.5 p. 135). Regular languages are closed under concatenation so $L\overline{M}$ is regular. Since $\{xy \mid x \in L \text{ and } y \notin M\} = L\overline{M}$ the result follows.

(d) **false:** Assume that it is a regular language. Then the pumping lemma (Theorem 4.1, p. 128) gives that there exists a constant n such that for every string w in the language with $|w| = n$ we can find strings x, y and z such that $w = xyz$ and furthermore $y \neq \epsilon$, $|xy| \leq n$ and for all $k \geq 0$ the string xy^kz is also in the language.

Now consider the string $w = 0^n10^n$ which clearly is in the language of interest. Taking $w = xyz$ we see that y will be a (non-empty) string of 0's. Thus it cannot be the case that $(xz)^R = xz$ and we have a contradiction.

(e) **true:** When L is regular then so is L^R (Theorem 4.1, p. 139). Regular languages are closed under intersection (Theorem 4.8, p. 136) so $L \cap L^R$ is a regular language. Since $\{w \mid w \in L \text{ and } w^R \in L\} = L \cap L^R$ the result follows.

(f) **true:** We simply observe that $\{xyx^R \mid x \in \Sigma^* \text{ and } y \in \Sigma^*\} = \Sigma^*$ since we can always take x to be ϵ. And Σ^* is a regular language.

Technical University of Denmark

Written examination, May 26, 2020

Page 1 of 19 pages

Course number: 02141

Aids allowed: All written aids are permitted

Changes due to the Covid-19 situation: The exam is conducted as a digital exam taken at home. For this reason answers that can be obtained using a computer need to be properly explained and will not be considered acceptable otherwise. You must submit your answer as a single pdf-file; it is fine to integrate pictures of drawings into the pdf-file, but no attachments to the pdf-file will be considered. If you think there are mistakes in the exam set (where in a written exam you would ask to be contacted by the teacher) please send an e-mail to fnie@dtu.dk for Formal Methods and to albl@dtu.dk for Regular Languages and Context Free Languages. Any announcements resulting from this will be communicated over inside.dtu.dk.

Exam duration: 4 hours

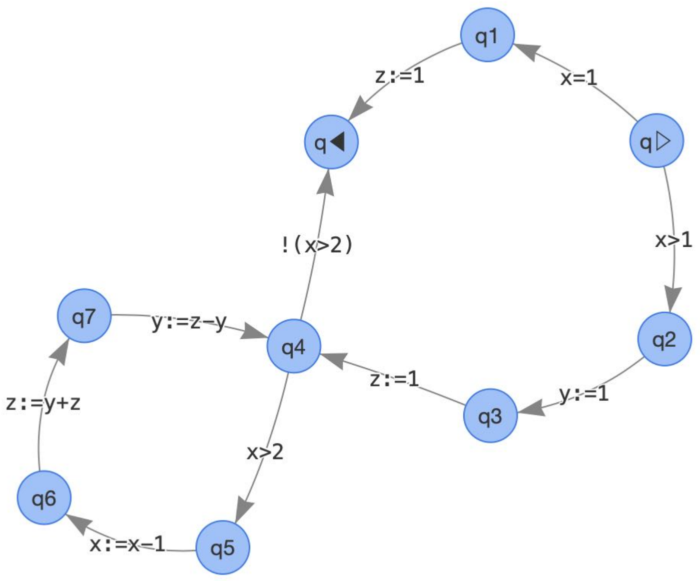
Weighting: 7-step scale

Answers Included

Exercises on Formal Methods

Exercise 1 (12%) Semantics

Consider the following program graph:



Question 1a: *Write a program in Guarded Commands for which* **edges**$(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\cdots]\!]$ *generates the above program graph.*

Solution :

```
if x=1 -> z:=1
[] x>1 -> y:=1;
          z:=1;
          do x>2 -> x:=x-1;
                    z:=y+z;
                    y:=z-y
          od
fi
```

Consider the complete execution sequence using the semantics in [FM]:

$$\langle q_\triangleright; [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 4, \mathtt{z} \mapsto 3]\rangle \overset{\omega}{\Longrightarrow}{}^{*} \langle q_\blacktriangleleft; \sigma\rangle$$

Question 1b: *What is σ and how long is ω (i.e. how many steps are taken)? (You do not need to write the entire execution sequence.)*

Solution : $\sigma = [\mathtt{x} \mapsto 2, \mathtt{y} \mapsto 3, \mathtt{z} \mapsto 5]$ and the length of ω is 16 (because the loop is executed three times, with four steps each time, and there are three steps to get into the loop and one step to get out).

Next we consider the concepts of a program graph being deterministic and/or evolving using the semantics in [FM].

Question 1c: *Is the program graph deterministic? (You should motivate your answer.)*

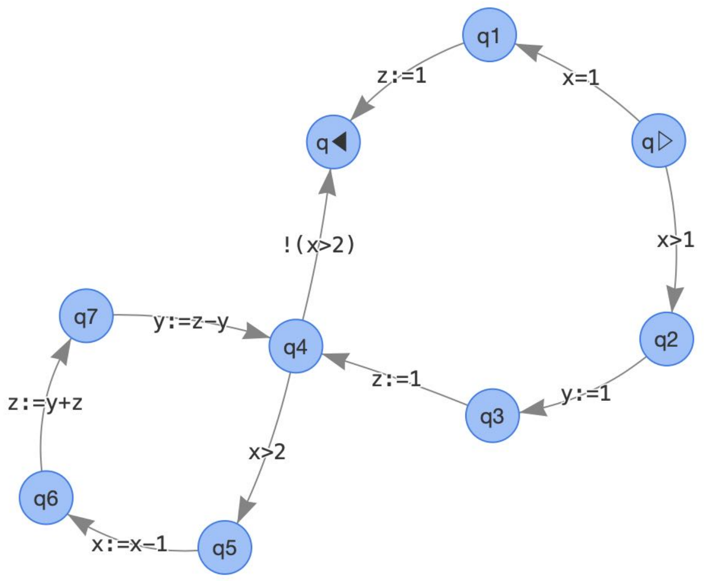
Solution : Yes, because the conditions of Proposition 1.22 are easily checked for the nodes $q_\triangleright$ and q_4.

Question 1d: *Is the program graph evolving? (You should motivate your answer.)*

Solution : No, because at node $q_\triangleright$ there is no edge to take if the initial value of $\mathtt{x}$ is non-positive.

Exercise 2 (15%) Program Verification and Analysis

Consider the following program graph that is exactly as in the previous exercise:



and consider the following predicate assignment $\mathbf{P}$ where $\mathbf{P}(q_5)$, $\mathbf{P}(q_6)$ and $\mathbf{P}(q_7)$ are missing:

$$\begin{array}{rcl}
\mathbf{P}(q_\triangleright) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_1) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_2) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_3) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_4) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_5) & = & \\
\mathbf{P}(q_6) & = & \\
\mathbf{P}(q_7) & = & \\
\mathbf{P}(q_\blacktriangleleft) & = & \mathtt{x} > 0, \mathtt{y} > 0, \mathtt{z} > 0
\end{array}$$

Question 2a: *Define $\mathbf{P}(q_5)$, $\mathbf{P}(q_6)$ and $\mathbf{P}(q_7)$ such that you get a correct predicate assignment in the sense of [FM].*

Solution :

$$\begin{array}{rcl}
\mathbf{P}(q_5) & = & \mathtt{x} > 2, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_6) & = & \mathtt{x} > 1, \mathtt{y} > 0, \mathtt{z} > 0 \\
\mathbf{P}(q_7) & = & \mathtt{x} > 1, \mathtt{y} > 0, \mathtt{z} > \mathtt{y}
\end{array}$$

Question 2b: *Argue that the predicate assignment is indeed correct in the sense of [FM].*

Solution : For 6 of the 10 edges we have the same predicate assignment at the source node as at the target node and it is straightforward to argue that the condition of Definition 3.8 holds and this leaves us with 4 edges.

The edge from q_4 to q_5 satisfies the condition as $\mathbf{P}(q_5)$ is $\mathbf{P}(q_4)$ strengthened to record that $\texttt{x} > 2$ because the test $\texttt{x} > 2$ was passed.

The edge from q_5 to q_6 satisfies the condition as $\mathbf{P}(q_6)$ is $\mathbf{P}(q_5)$ modified to record that $\texttt{x} > 1$ because 1 is subtracted from $\texttt{x}$.

The edge from q_6 to q_7 satisfies the condition as only $\texttt{z}$ is changed and indeed $\texttt{z} > \texttt{y}$ because the initial value of $\texttt{z}$ was positive.

The edge from q_7 to q_4 satisfies the condition as $\texttt{x} > 1$ implies $\texttt{x} > 0$ and the fact that $\texttt{z} > \texttt{y} > 0$ ensures that $\texttt{z} - \texttt{y} > 0$.

Moving on to program analysis, it is worth considering whether we could have obtained the same insights using the detection of signs analysis. So consider the following analysis assignment $\mathbf{A}$ where $\mathbf{A}(q_5)$, $\mathbf{A}(q_6)$ and $\mathbf{A}(q_7)$ are missing:

$$\begin{array}{rcl}
\mathbf{A}(q_\triangleright) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\} \\
\mathbf{A}(q_1) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\} \\
\mathbf{A}(q_2) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\} \\
\mathbf{A}(q_3) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\} \\
\mathbf{A}(q_4) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\} \\
\mathbf{A}(q_5) & = & \\
\mathbf{A}(q_6) & = & \\
\mathbf{A}(q_7) & = & \\
\mathbf{A}(q_\blacktriangleleft) & = & \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\}
\end{array}$$

Question 2c: *Is it possible to define $\mathbf{A}(q_5)$, $\mathbf{A}(q_6)$ and $\mathbf{A}(q_7)$ such that you get a semantically correct analysis assignment.? (You should argue for your answer.)*

Solution : The answer is no.

The most immediate choice would be to set $\mathbf{A}(q_5) = \mathbf{A}(q_6) = \mathbf{A}(q_7) = \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\}$ but the conditions on the edges from q_5 to q_6 and from q_7 to q_4 would then not hold.

In case of the edge from q_5 to q_6 we could rectify the problem by setting $\mathbf{A}(q_5) = \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\}$ and $\mathbf{A}(q_6) = \mathbf{A}(q_7) = \{[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +], [\texttt{x} \mapsto 0, \texttt{y} \mapsto +, \texttt{z} \mapsto +], [\texttt{x} \mapsto -, \texttt{y} \mapsto +, \texttt{z} \mapsto +]\}$.

However, the edge from q_7 to q_4 remains a problem as $\mathbf{A}(q_7)$ needs to contain the tuple $[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]$ in order to be semantically correct and this requires $\mathbf{A}(q_4)$ to contain the tuples $[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto +]$, $[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto 0]$, $[\texttt{x} \mapsto +, \texttt{y} \mapsto +, \texttt{z} \mapsto -]$ in order to be semantically correct and this is against the requirements.

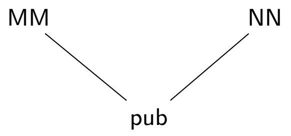
In short, the detection of signs analysis is not able to express the condition $\texttt{z} > \texttt{y} > 0$ any more precisely than $\texttt{z} > 0, \texttt{y} > 0$ and therefore the answer is no.

Exercise 3 (13%) Information Flow

Consider the following program in Guarded Commands

```
i:=0; do i<n -> N[i]:=N[i]+27; i:=i+1 od;
j:=0; do j<m -> M[j]:=M[j]+33; j:=j+1 od
```

together with the following partially ordered set of security properties



Next consider the following possibilities for the security classification $\mathbf{L}_i$:

i	$\mathbf{L}_i$(i)	$\mathbf{L}_i$(n)	$\mathbf{L}_i$(N)	$\mathbf{L}_i$(j)	$\mathbf{L}_i$(m)	$\mathbf{L}_i$(M)
1	pub	pub	NN	pub	pub	MM
2	pub	NN	NN	pub	MM	MM
3	NN	pub	NN	MM	pub	MM
4	NN	NN	NN	MM	MM	MM
5	pub	pub	NN	MM	MM	MM

Question 3a: *For which of the above possibilities for* $\mathbf{L}_i$ *will the security analysis* $\mathbf{sec}[\![\cdots]\!](\{\,\})$ *report that there are no explicit or implicit flows that violate* $\mathbf{L}_i$*? (It is essential that you motivate your answer.)*

Solution : The answer is $\mathbf{L}_1$, $\mathbf{L}_3$, $\mathbf{L}_4$ and $\mathbf{L}_5$.

One way to motivate the answer is to construct the program graph using $\mathbf{edges_s}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\cdots]\!](\{\,\})$ and observe that the actions on the edges for assignments within the loops will be

```
N[i]:=N[i]+27{i,n}
i:=i+1{i,n}
M[j]:=M[j]+33{j,m}
j:=j+1{j,m}
```

In case of $\mathbf{L}_1$, $\mathbf{L}_3$, $\mathbf{L}_4$ and $\mathbf{L}_5$ all explicit and implicit flows are permissible whereas in the case of $\mathbf{L}_2$ we have $\mathbf{L}_2(\mathtt{n}) \not\sqsubseteq \mathbf{L}_2(\mathtt{i})$ and $\mathbf{L}_2(\mathtt{m}) \not\sqsubseteq \mathbf{L}_2(\mathtt{j})$.

Another way to motivate the answer is to unfold the definition of $\mathbf{sec}[\![\cdots]\!](\{\,\})$ as when evaluating a functional program by hand.

Next let us consider the following program in Guarded Commands

```
j:=0; do j<m -> M[j]:=M[j]+33; j:=j+1 od;
i:=0; do i<n -> N[i]:=N[i]+27; i:=i+1 od
```

together with the same partially ordered set of security properties and the same possible security classifications $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{L}_3$, $\mathbf{L}_4$ and $\mathbf{L}_5$.

Question 3b: *Would the answer to the previous question be the same for this program? (It is essential that you motivate your answer.)*

Solution : The answer is yes because there are no flows between the two lines of the program so everything is symmetrical.

We now consider the following program in Guarded Commands

```
i:=0; j:=0;
do i<n -> N[i]:=N[i]+27; i:=i+1
[] j<m -> M[j]:=M[j]+33; j:=j+1
od
```

together with the same partially ordered set of security properties and the same possible security classifications $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{L}_3$, $\mathbf{L}_4$ and $\mathbf{L}_5$.

Question 3c: *For which of the above possibilities for* $\mathbf{L}_i$ *will the security analysis* $\mathbf{sec}[\![\cdots]\!](\{\,\})$ *report that there are no explicit or implicit flows that violate* $\mathbf{L}_i$*? (It is essential that you motivate your answer.)*

Solution : The answer is $\mathbf{L}_1$ and $\mathbf{L}_5$.

One way to motivate the answer is to construct the program graph using $\mathbf{edges_s}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\cdots]\!](\{\,\})$ and observe that the actions on the edges for assignments within the loops will be

```
N[i]:=N[i]+27{i,n}
i:=i+1{i,n}
M[j]:=M[j]+33{i,n,j,m}
j:=j+1{i,n,j,m}.
```

In case of $\mathbf{L}_1$ and $\mathbf{L}_5$ all explicit and implicit flows are permissible whereas in the remaining cases we will have $\mathbf{L}(\mathtt{i}) \not\sqsubseteq \mathbf{L}(\mathtt{M})$ or $\mathbf{L}(\mathtt{n}) \not\sqsubseteq \mathbf{L}(\mathtt{M})$.

Another way to motivate the answer is to unfold the definition of $\mathbf{sec}[\![\cdots]\!](\{\,\})$ as when evaluating a functional program by hand.

Next let us consider the following program in Guarded Commands

```
j:=0; i:=0;
do j<m -> M[j]:=M[j]+33; j:=j+1
[] i<n -> N[i]:=N[i]+27; i:=i+1
```

together with the same partially ordered set of security properties and the same possible security classifications $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{L}_3$, $\mathbf{L}_4$ and $\mathbf{L}_5$.

Question 3d: *Would the answer to the previous question be the same for this program? (It is essential that you motivate your answer.)*

Solution : The answer would be different.

One way to motivate the answer is to construct the program graph using $\mathbf{edges_s}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\cdots]\!](\{\})$ and observe that the actions on the edges for assignments within the loops will be

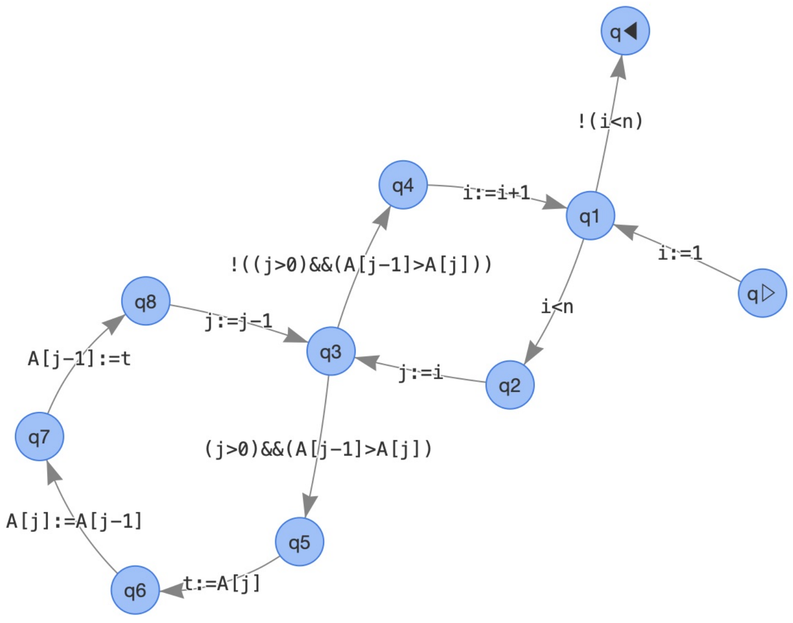
```
N[i]:=N[i]+27{i,n,j,m}
i:=i+1{i,n,j,m}
M[j]:=M[j]+33{j,m}
j:=j+1{j,m}.
```

In case of $\mathbf{L}_1$ all explicit and implicit flows are permissible whereas in the remining cases we will have $\mathbf{L}(\texttt{i}) \not\sqsubseteq \mathbf{L}(\texttt{M})$ or $\mathbf{L}(\texttt{n}) \not\sqsubseteq \mathbf{L}(\texttt{M})$.

A shorter answer with the same insight would also be fine.

Exercise 4 (10%) Model Checking

Consider the following program graph (for the version of insertion sort considered in the course):



We next consider a transition system obtained from the program graph (but in a different way compared to [FM] Section 6.4). Informally, just erase all the actions on the edges and take the initial node as the only initial state. More formally, consider a transition system obtained from the program graph such that

- the set of states is the set of nodes in the program graph,
- the set of initial states contains only the state corresponding to the initial node,
- there is a transition from a state to another if and only if there is a directed labelled edge from the former to the latter, i.e. we merely disregard the actions on the edges,
- the set of atomic propositions are the names of the nodes, and
- the labelling function for states therefore trivially maps a state to the set consisting of just that state.

Recall that a state formula Φ of CTL holds on a transition system if and only if it holds on all initial states, which in this case is merely the initial node $q_\triangleright$.

Next consider the following formula of CTL:

1 $\mathbf{EF}\, q_\blacktriangleleft$

2 $\mathbf{AF}\, q_\blacktriangleleft$

3 $\mathbf{AF}\,\mathbf{EF}\, q_\blacktriangleleft$

4 $\mathbf{EF}\,\mathbf{AF}\, q_\blacktriangleleft$

Question 4a: *Which of the above formulae hold on the transition system? (You should motivate your answer.)*

Solution : The answer is 1, 3, 4.

For 1 we note that there exists the finite path $q_\triangleright, q_1, q_\blacktriangleleft$.

For 2 we note that there exists the infinite path $q_\triangleright, q_1, q_2, q_3, q_4, q_1, \cdots, q_2, q_3, q_4, q_1, \cdots$.

For 3 consider an arbitrary path starting in $q_\triangleright$ and observe that already in $q_\triangleright$ we have $\mathbf{EF}\, q_\blacktriangleleft$.

For 4 consider the finite path $q_\triangleright, q_1, q_\blacktriangleleft$ and note that $\mathbf{AF}\, q_\blacktriangleleft$ holds in $q_\blacktriangleleft$.

An alternative approach is to determine all the states in which the formulae in 1 and 2 hold and to use that for the formulae in 3 and 4 (as we do in the algorithms for model checking).

Next consider the following formula of CTL:

5 $\neg\,\mathbf{E}(\neg\, q_\triangleright)\mathbf{U} q_\blacktriangleleft$

6 $\neg\,\mathbf{E}(\neg\, q_2)\mathbf{U} q_8$

7 $\neg\,\mathbf{E}(\neg\, q_8)\mathbf{U} q_2$

8 $\neg\,\mathbf{E}(\neg\,q_5)\mathbf{U}q_8$

9 $\neg\,\mathbf{E}(\neg\,q_8)\mathbf{U}q_5$

10 $\neg\,\mathbf{E}(\neg\,q_1)\mathbf{U}q_1$

Question 4b: *Which of the above formulae hold on the transition system? (You should motivate your answer.)*

Solution : The answer is 5, 6, 8.

Answers can be motivated in many ways and we provide two approaches.

Approach 1: Here we will use the insight that $\neg\,\mathbf{E}(\neg\,q_i)\mathbf{U}q_j$ holds at $q_\triangleright$ if and only if every path fragment from $q_\triangleright$ to q_j must pass through q_i strictly before reaching q_j. Using standard notation we shall say that q_i strictly dominates q_j.

In case of 5 clearly the initial node strictly dominates the final node.

In case of 6 clearly q_2 strictly dominates q_8 as a path fragment from $q_\triangleright$ to q_8 must start with $q_\triangleright, q_2$.

In case of 7 the path fragment $q_\triangleright, q_2$ provides the negative answer.

In case of 8 clearly q_5 strictly dominates q_8 as a path fragment from $q_\triangleright$ to q_8 must end with $q_5, q_6, q_7, q_8,$.

In case of 9 the path fragment $q_\triangleright, q_2, q_3, q_5$ provides the negative answer.

In case of 10 the path fragment $q_\triangleright, q_1$ provides the negative answer.

Approach 2: An alternative approach is to first consider the formula without the leading negation (as we do in the model checking algorithm) and work out whether or not that formula holds in $q_\triangleright$. For the formulae without the leading negation we get that 7, 9, 10 hold in $q_\triangleright$ (by considering the same path fragments as above), whereas 5, 6, 8 do not; in case of 5 because we cannot avoid starting at $q_\triangleright$, in case of 6 because any path from $q_\triangleright$ to q_8 must pass through q_2, and in case of 8 because any path from $q_\triangleright$ to q_8 must pass through q_5.

Incorporating the leading negation provides the desired answer.

Exercises on Context-free Languages

Exercise 5 (25%)

Function signatures are used in programming languages to specify the type of functions. Consider the following context-free grammar for expressing function signatures in an F#-like style:

$$
\begin{array}{rcll}
F & \rightarrow & C & \text{(composed type)} \\
 & | & F\texttt{->}F & \text{(function type)} \\
C & \rightarrow & B & \text{(basic type)} \\
 & | & C \texttt{ * } C & \text{(tuple type)} \\
 & | & C \texttt{ list} & \text{(list type)} \\
 & | & \texttt{[}F\texttt{]} & \text{(brackets for grouping types)} \\
B & \rightarrow & \texttt{int} & \text{(integer type)} \\
 & | & \texttt{string} & \text{(string type)}
\end{array}
$$

the set of non-terminal symbols is $N = \{F, C, B\}$, the set of terminal symbols is $T = \{\texttt{->}, \texttt{*}, \texttt{list}, \texttt{int}, \texttt{string}, \texttt{[}, \texttt{]}\}$ and the initial symbol is F. Function signatures are all strings accepted by the grammar.

(a) Design a set of datatypes that are suitable to store abstract representations (ASTs) of function signatures and how the function signature

```
[ int list * string ] -> string
```

would be represented as a value of your datatype.

Solution: In F# we could define the following types:

```
type F =
       | Composed of C
       | Function of F * F
and C =
       | Basic of B
       | Tuple of C * C
       | List of C
       | Group of F
and B =
       | Integer
       | String
```

The function signature would be represented by

```
Function (
    Composed(
        Group(
            Composed(
             Tuple(
                    List (Basic(Integer)) ,
                    Basic(String)
                 )
            )
        ) ,
    Composed(Basic(String))
    )
)
```

(b) Provide the precedence rules of the type composition operators `->`, `*` and `list` by filling out the following table. In cell (x, y) write "yes" if, according to the above grammar, x has precedence over y (i.e. x binds more tightly than y), and write "no" otherwise.

	`->`	`*`	`list`
`->`	X		
`*`		X	
`list`			X

Solution:

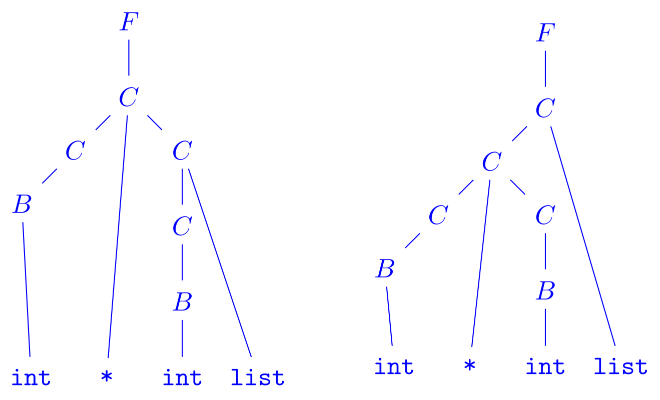
	`->`	`*`	`list`
`->`	X	no	no
`*`	yes	X	no
`list`	yes	no	X

(c) If the answer to exercise (b) includes two operators whose precedence is undefined according to the above grammar (they don't have precedence over each other), provide an example of a function signature that has two distinct parse trees, each encoding a different precedence. Otherwise, briefly explain why it is not possible to find such an expression.

Solution: The precedence between `list` and `*` is undefined. A simple example of a function signature where such ambiguity arises is

```
int * int list
```

Two distinct parse trees can be given for such expression, each encoding different precedences:



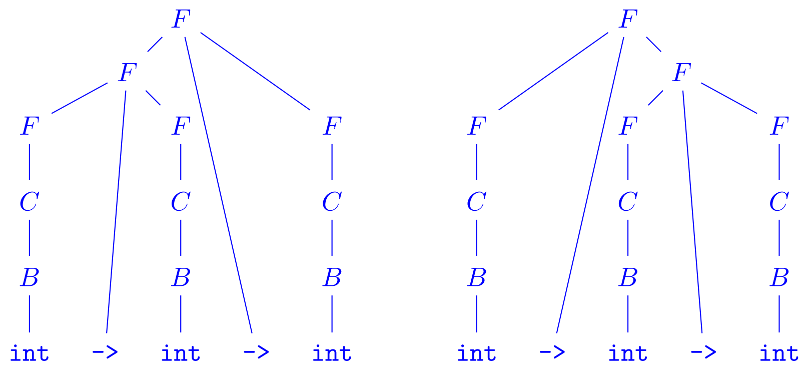
(d) In F# operator `->` is right-associative. Is it also like this in the provided grammar? If the answer is yes, provide a brief explanation. If the answer is no, explain why and provide a new grammar that enforces associativity of `->` as in F# and that accepts the same language of the original grammar.

Solution:

In the provided grammar the associativity of `->` is not defined. For example two distinct parse tress can be given to

```
int -> int -> int
```

namely



We transform the grammar by forbidding recursion on the left for `->`.

$$\begin{array}{lcll}
F & \to & C & \\
 & | & C\texttt{->}F & // \ F\texttt{->}F \text{ unfolded on the left with } C \\
C & \to & B & \text{(basic type)} \\
 & | & C * C & \text{(tuple type)} \\
 & | & C \ \texttt{list} & \text{(list type)} \\
 & | & \texttt{[}F\texttt{]} & \text{(brackets for grouping types)} \\
B & \to & \texttt{int} & \\
 & | & \texttt{string} &
\end{array}$$

(e) We focus now on the subset of function signatures given by the regular expression $(\texttt{int} + \texttt{string})(\texttt{->}(\texttt{int} + \texttt{string}))^*$. Provide a context-free grammar that accepts the same language as the regular expression.

Solution: One possible solution is

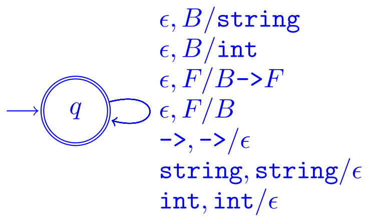
$$\begin{array}{lcl}
F & \to & B \mid B\texttt{->}F \\
B & \to & \texttt{int} \mid \texttt{string}
\end{array}$$

(f) Construct a Pushdown Automaton (PDA) that accepts the same language as the grammar in exercise (e). Use the construction that we saw in class (and described in book [HMU] 02141: Automata Theory and Languages - edited by Hanne Riis Nielson) to translate a context-free grammar into an equivalent, non-deterministic PDA that accepts by empty stack.

Solution: The PDA obtained accepts by empty stack and is defined by the tuple $(\{q\}, \{\texttt{int}, \texttt{string}, \texttt{->}\}, \{\texttt{int}, \texttt{string}, \texttt{->}\} \cup \{F, B\}, \delta, q, F, \{q\})$ where the transition function δ is defined as follows:

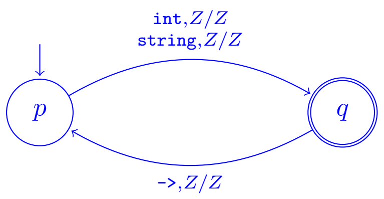
$$\begin{array}{rcl}
\delta(q, \epsilon, F) & = & \{(q, B), (q, B\texttt{->}F)\} \\
\delta(q, \epsilon, B) & = & \{(q, \texttt{int}), (q, \texttt{string})\} \\
\delta(q, \texttt{int}, \texttt{int}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{string}, \texttt{string}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{->}, \texttt{->}) & = & \{(q, \epsilon)\}
\end{array}$$

A graphical representation of the PDA using the notation seen in class is also a valid answer.


$\epsilon, B/\texttt{string}$
$\epsilon, B/\texttt{int}$
$\epsilon, F/B\texttt{->}F$
$\epsilon, F/B$
$\texttt{->}, \texttt{->}/\epsilon$
$\texttt{string}, \texttt{string}/\epsilon$
$\texttt{int}, \texttt{int}/\epsilon$
q

(g) The PDA obtained in exercise (f) is non-deterministic. Can you build a deterministic PDA instead? If no, explain why. If yes, provide the deterministic PDA.

Solution: The answer is yes. The language was provided in exercise (e) as a regular expression. It is therefore a regular language, and there is therefore a DFA for it that we can obtain with constructions seen in class: from regular expression to ϵ-NFA, and from ϵ-NFA to DFA. Every DFA is trivially a deterministic PDA. The thus obtained deterministic PDA is graphically as follows

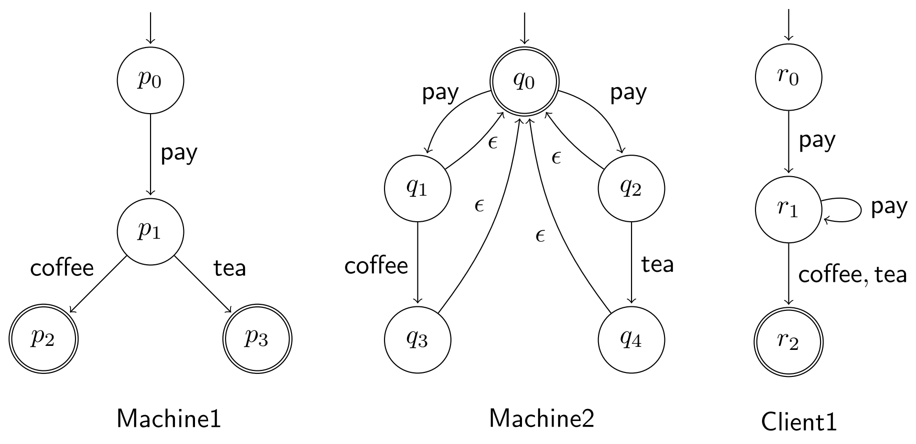


It accepts by final state. Note that the stack plays no role and remains constant (always containing the initial stack symbol Z).

Exercises on Regular Languages

Exercise 6 (25%)

In this exercise we consider models of coffee machines and their clients. Coffee machines Machine1, Machine2 and client Client1 are modelled with finite state automata given as transition diagrams below:



Client Client2 instead is modelled with the regular expression

$$\mathsf{pay}^*(\mathsf{coffee} + \mathsf{tea})$$

(a) For each client $C \in \{\mathsf{Client1}, \mathsf{Client2}\}$ and each coffee machine $M \in \{\mathsf{Machine1}, \mathsf{Machine2}\}$ answer the following question: Does client C accept words that are *not* accepted by machine M? Respond by

- providing a table, like the one below, filled with either "yes" or "no", and
- providing a justification for each answer: if the answer is "yes", provide one example (as short as possible), if the answer is "no" provide a very brief explanation.

	Machine1	Machine2
Client1		
Client2		

Solution:

	Machine1	Machine2
Client1	yes	no
Client2	yes	yes

- Client1 vs Machine1: Client1 accepts word pay pay coffee, which is not accepted by Machine1.

- Client1 vs Machine2: all words of Client1 are either of the form pay^{n}coffee or pay^{n}tea for $n > 0$. Consider the first case. It is easy to see that Machine2 can accept any such word by cycling through q_0, q_1 n times using pay and ϵ transitions, then taking the coffee transition to q_3 and finally the ϵ transition to the accepting state q_0. For words of the form pay^{n}tea the reasoning is similar.
- Client2 vs Machine1: Client2 accepts word coffee, which is not accepted by Machine1.
- Client2 vs Machine2: Client2 accepts word coffee, which is not accepted by Machine2.

(b) We want now to make clients interact with machines and for that purpose we define a composition operator $\triangleright$ as follows.

Let $A = \langle Q_A, \Sigma, \delta_A, q_0^A, F_0^A \rangle$ and $B = \langle Q_B, \Sigma, \delta_B, q_0^B, F_0^B \rangle$ be ϵ-NFA. We define the *composition* of A and B, denoted $A \triangleright B$, as the ϵ-NFA $\langle Q, \Sigma, \delta, q_0, F_0 \rangle$ such that

- $Q = (Q^A \times Q^B) \cup \{\mathsf{error}\}$;
- $\delta : Q \times \Sigma \to P(Q)$ is such that

$$\delta((s_A, s_B), a) = \begin{cases} \left(\begin{array}{l} \{(s'_A, s_B) \mid s'_A \in \delta_A(s_A, a)\} \cup \\ \{(s_A, s'_B) \mid s'_B \in \delta_B(s_B, a)\} \end{array} \right) & \text{if } a = \epsilon \\ \{\mathsf{error}\} & \text{if } a \in (\Sigma \setminus \{\epsilon\}) \text{ , } \delta_A(s_A, a) \neq \emptyset \\ & \text{ and } \delta_B(s_B, a) = \emptyset \\ \delta_A(s_A, a) \times \delta_B(s_B, a) & \text{otherwise} \end{cases}$$

- $q_0 = (q_0^A, q_0^B)$;
- $F = F_0^A \times F_0^B$.

where we assume that state error is a new state neither contained in Q^A nor in Q^B.

Construct the automaton Client1 $\triangleright$ Machine1 and provide it as a transition diagram. Depict only the reachable states.

Solution:

$$\begin{tikzcd}
 & \arrow[d] & \\
 & (p_0, r_0) \arrow[d, "\mathsf{pay}"] & \\
 & (p_1, r_1) \arrow[r, "\mathsf{pay}"] \arrow[dl, "\mathsf{coffee}"'] \arrow[dr, "\mathsf{tea}"] & \mathsf{error} \\
\boxed{(p_2, r_2)} & & \boxed{(p_3, r_2)}
\end{tikzcd}$$

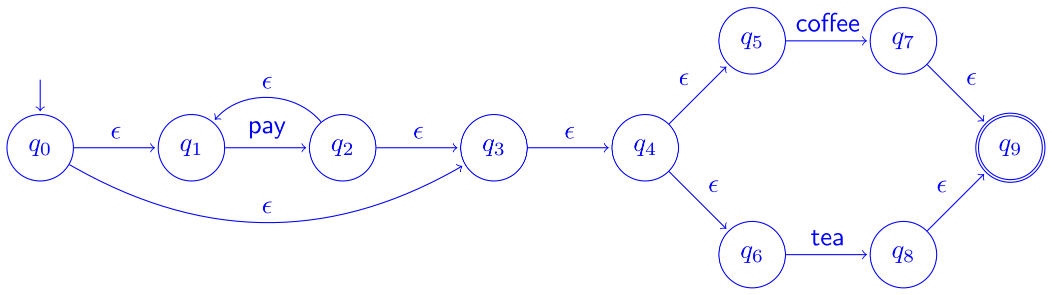
(c) Construct the automaton $\mathsf{Client1} \triangleright \mathsf{Machine2}$ and provide it as a transition diagram.

Solution:

$$\begin{tikzcd}
\arrow[d] & & \\
(q_0, r_0) \arrow[d, "\mathsf{pay}"'] \arrow[ddrr, "\mathsf{pay}"] & (q_0, r_1) \arrow[dl, bend right, "\mathsf{pay}"'] \arrow[dr, bend left, "\mathsf{pay}"] & \\
(q_1, r_1) \arrow[ur, bend right, "\epsilon"'] \arrow[r, "\mathsf{tea},\ \mathsf{pay}"] \arrow[d, "\mathsf{coffee}"'] & \mathsf{error} & (q_2, r_1) \arrow[ul, bend left, "\epsilon"] \arrow[l, "\mathsf{coffee},\ \mathsf{pay}"] \arrow[d, "\mathsf{tea}"] \\
(q_3, r_2) \arrow[r, "\epsilon"] & \boxed{(q_0, r_2)} & (q_4, r_2) \arrow[l, "\epsilon"]
\end{tikzcd}$$

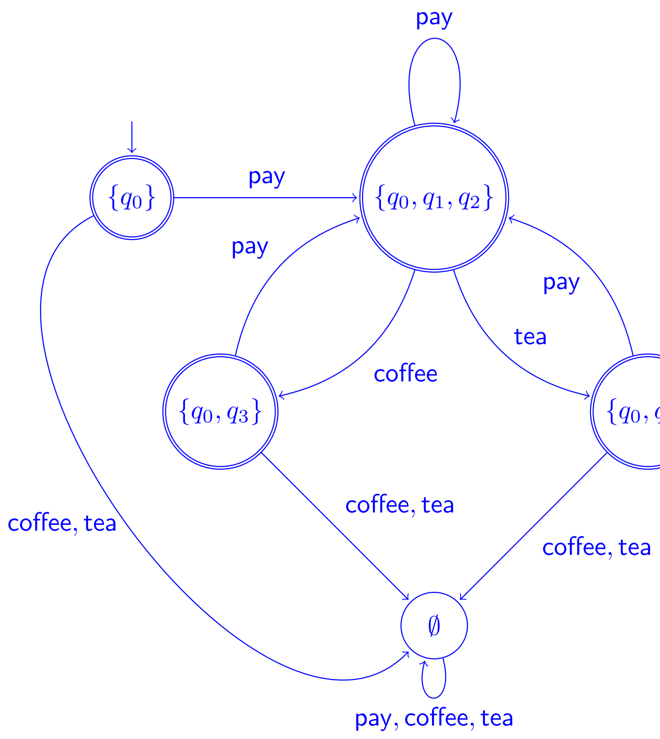
(d) Convert $\mathsf{Client2}$ into an ϵ-NFA using the procedure seen in the course. Provide the result as a transition diagram and do not apply any optimisation (i.e. do not remove ϵ-transitions).

Solution:



(e) Convert Machine2 into a DFA using the procedure seen in the course. Provide the result as a transition diagram. Do not include unreachable states. Do include the state $\emptyset$ if it is reachable.

Solution:



Technical University of Denmark

Written examination date: May 26, 2021

Page 1 of 16

Course title: Computer Science Modelling

Course number: 02141

Aids allowed: All aids allowed for digital exams

Exam duration: 4 hours

Weighting: See the individual exercises

Grading: 7-step scale

Due to the Covid-19 situation the exam is conducted as a digital exam taken at home with the aids allowed for such exams. For this reason answers that can be obtained using a computer need to be properly explained and will not be considered acceptable otherwise. You must submit your answer as a single pdf-file; it is fine to integrate pictures of drawings into the pdf-file, but no attachments to the pdf-file will be considered.

If you think there are mistakes in the exam set (where in a written exam you would ask to be contacted by the teacher) please send an e-mail to fnie@dtu.dk for Formal Methods and to albl@dtu.dk for Regular Languages and Context Free Languages. Any announcements resulting from this will be communicated over the Microsoft Teams channel for the exam.

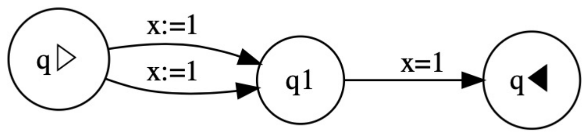
Answers Included

Exercises on Formal Methods

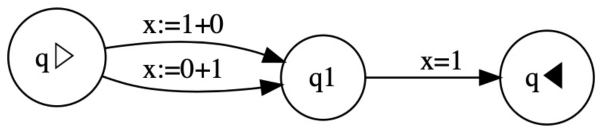
Exercise 1 (10%) Program Graphs

Consider the following program graphs:

Program graph A:



Program graph B:



Question 1a: *For each of program graphs A and B, does it satisfy the sufficient conditions of [FM Proposition 1.22] for being a deterministic system?*
(You should explain your answer.)

Solution (2%): No for both A and B because at $q_\triangleright$ there are two distinct edges emanating that do not have non-overlapping domains (they have the same non-empty domains).
Yes for A if you translate the graph into tuple notation and observe that the two distinct edges in the graph obtain the same representation and hence are the same.

Question 1b: *For each of program graphs A and B, does it satisfy the conditions of [FM Definition 1.21] for being a deterministic system?*
(You should explain your answer.)

Solution (2%): Yes for A and No for B. It is Yes for A because the two edges emanating from $q_\triangleright$ have the same actions. It is No for B because the two edges emanating from $q_\triangleright$ have different actions (even though they are semantically equivalent).

Question 1c: *For each of program graphs A and B, does it satisfy the sufficient conditions of [FM Proposition 1.25] for being an evolving system?*

(You should explain your answer.)

Solution (2%): No for both A and B because there is no edge emanating from q_1 that can be taken when the value of `x` is different from 1.

Question 1d: *For each of program graphs A and B, does it satisfy the conditions of [FM Definition 1.24] for being an evolving system?*
(You should explain your answer.)

Solution (2%): Yes for both A and B because all executions reaching q_1 will be with a memory where `x` is 1 and hence the emanating edge can be taken.

Question 1e: *For each of program graphs A and B, does there exist a program in Guarded Commands that generates the program graph according to [FM Section 2.2]?*
(You should explain your answer: in case of a positive answer show the program, in case of a negative answer argue why this cannot be the case.)

Solution (2%): No for both A and B. Inspection of [FM Section 2.2] shows that whenever we generate more than one edge that emanates from a node, then all these edges will be labelled with a boolean condition, which is not the case for neither programs A or B.

Exercise 2 (20%) Program Verification and Analysis

Consider program graph C from Figure 1 and consider the following execution sequence

$$\langle q_{\triangleright}; \sigma\rangle \overset{w}{\Longrightarrow}^{*} \langle q_{\blacktriangleleft}; \sigma'\rangle$$

where σ is given by $\sigma(\mathtt{x}) = 5$, $\sigma(\mathtt{y}) = 0$, $\sigma(\mathtt{z}) = 0$, $\sigma(\mathtt{zz}) = 0$.

Question 2a: *What is σ'?*
(You should explain your answer.)

Solution (4%): We get $\sigma'(\mathtt{x}) = 2$, $\sigma'(\mathtt{y}) = 3$, $\sigma'(\mathtt{z}) = 5$, $\sigma'(\mathtt{zz}) = 3$, and note that at q_5 the values of `y` and `z` will be two consecutive Fibonacci numbers (and the value of `y` will be equal to that of `zz`),

Recall the Fibonacci numbers 1, 1, 2, 3, 5, 8 ... and let us write them as $fib(1) = 1$, $fib(2) = 1$, $fib(n + 2) = fib(n) + fib(n + 1)$ (for $n > 0$)

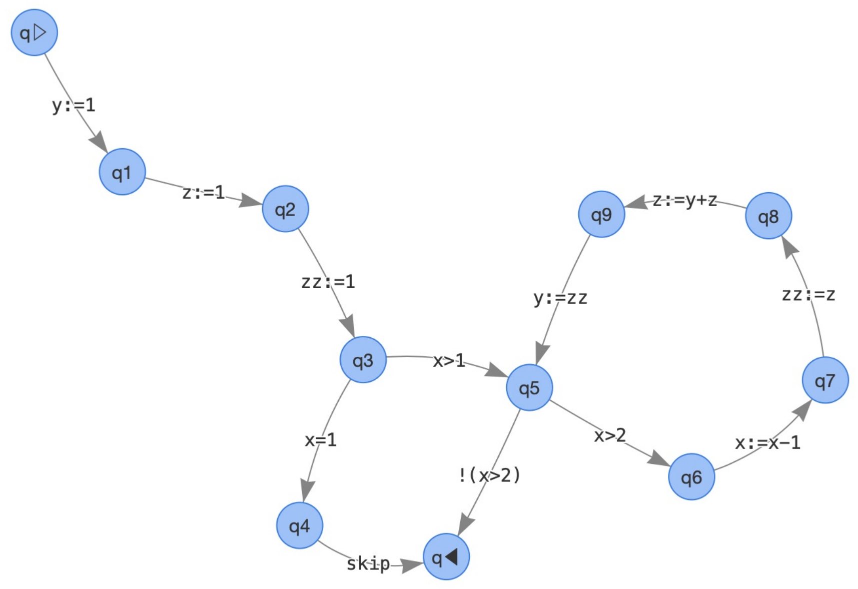


Figure 1: Program Graph C

.

Question 2b: *Complete the following partial predicate assignment*

$$\begin{aligned}\mathbf{P}(q_\triangleright) &= \mathtt{x}=\mathtt{n} \ \wedge \ \mathtt{n}>0 \\ \mathbf{P}(q_5) &= \cdots \\ \mathbf{P}(q_\blacktriangleleft) &= \mathtt{z}=fib(\mathtt{n})\end{aligned}$$

to a partially correct predicate assignment (in the sense of [FM, Section 3.3]), and list all the correct proof obligations that need to be checked. (You do not have to check them in detail provided that they are correct.)

Solution (8%): A solution is

$$\begin{aligned}\mathbf{P}(q_\triangleright) &= \mathtt{x}=\mathtt{n} \ \wedge \ \mathtt{n}>0 \\ \mathbf{P}(q_5) &= \mathtt{n}-\mathtt{x}\geq 0 \ \wedge \ \mathtt{x}>1 \ \wedge \ \mathtt{y}=fib(\mathtt{n}-\mathtt{x}+1) \ \wedge \ \mathtt{z}=fib(\mathtt{n}-\mathtt{x}+2) \\ \mathbf{P}(q_\blacktriangleleft) &= \mathtt{z}=fib(\mathtt{n})\end{aligned}$$

and note that at q_5 the value of $\mathtt{n}-\mathtt{x}$ gives the number of times the loop has been traversed.

We need to check the proof obligations:

$$\left[\begin{array}{l}\texttt{x} = \texttt{n} \land \\ \texttt{n} > 0\end{array}\right] \quad \begin{array}{l}\texttt{y} := 1 \\ \texttt{z} := 1 \\ \texttt{zz} := 1 \\ \texttt{x} > 1\end{array} \quad \left[\begin{array}{l}\texttt{n} - \texttt{x} \geq 0 \land \\ \texttt{x} > 1 \land \\ \texttt{y} = fib(\texttt{n} - \texttt{x} + 1) \land \\ \texttt{z} = fib(\texttt{n} - \texttt{x} + 2)\end{array}\right]$$

$$\left[\begin{array}{l}\texttt{x} = \texttt{n} \land \\ \texttt{n} > 0\end{array}\right] \quad \begin{array}{l}\texttt{y} := 1 \\ \texttt{z} := 1 \\ \texttt{zz} := 1 \\ \texttt{x} = 1 \\ \texttt{skip}\end{array} \quad \left[\ \texttt{z} = fib(\texttt{n})\ \right]$$

$$\left[\begin{array}{l}\texttt{n} - \texttt{x} \geq 0 \land \\ \texttt{x} > 1 \land \\ \texttt{y} = fib(\texttt{n} - \texttt{x} + 1) \land \\ \texttt{z} = fib(\texttt{n} - \texttt{x} + 2)\end{array}\right] \quad !(\texttt{x} > 2) \quad \left[\ \texttt{z} = fib(\texttt{n})\ \right]$$

$$\left[\begin{array}{l}\texttt{n} - \texttt{x} \geq 0 \land \\ \texttt{x} > 1 \land \\ \texttt{y} = fib(\texttt{n} - \texttt{x} + 1) \land \\ \texttt{z} = fib(\texttt{n} - \texttt{x} + 2)\end{array}\right] \quad \begin{array}{l}\texttt{x} > 2 \\ \texttt{x} := \texttt{x} - 1 \\ \texttt{zz} := \texttt{z} \\ \texttt{z} := \texttt{y} + \texttt{z} \\ \texttt{y} := \texttt{zz}\end{array} \quad \left[\begin{array}{l}\texttt{n} - \texttt{x} \geq 0 \land \\ \texttt{x} > 1 \land \\ \texttt{y} = fib(\texttt{n} - \texttt{x} + 1) \land \\ \texttt{z} = fib(\texttt{n} - \texttt{x} + 2)\end{array}\right]$$

We dispense with the detailed argument for why they are correct.

Next let us consider the detection of signs analysis as desribed in [FM, Chapter 4] and let us write an abstract state as (sx, sy, sz, szz) where sx is the sign of `x` etc. so that (0,+,-,0) represents an abstract state where `x` is 0, `y` is positive, `z` is negative and `zz` is 0.

Question 2c: *Construct the least analysis assignment* $\mathbf{A}$ *that has* $\mathbf{A}(q_\triangleright) = \{(+,+,+,+)\}$ *and give the results for* $\mathbf{A}(q_5), \mathbf{A}(q_6), \mathbf{A}(q_7), \mathbf{A}(q_\blacktriangleleft)$.
(You should briefly explain how you obtain the answer.)

Solution (8%): Using the algorithm for chaotic iteration we get

$$\begin{array}{l}\mathbf{A}(q_\triangleright) = \mathbf{A}(q_1) = \mathbf{A}(q_2) = \mathbf{A}(q_3) = \mathbf{A}(q_4) = \mathbf{A}(q_6) = \{(+,+,+,+)\} \\ \mathbf{A}(q_5) = \mathbf{A}(q_7) = \mathbf{A}(q_8) = \mathbf{A}(q_9) = \mathbf{A}(q_\blacktriangleleft) = \\ \qquad \{(-,+,+,+),(0,+,+,+),(+,+,+,+)\}\end{array}$$

so in particular

$$\begin{array}{l}\mathbf{A}(q_5) = \{(-,+,+,+),(0,+,+,+),(+,+,+,+)\} \\ \mathbf{A}(q_6) = \{(+,+,+,+)\} \\ \mathbf{A}(q_7) = \{(-,+,+,+),(0,+,+,+),(+,+,+,+)\} \\ \mathbf{A}(q_\blacktriangleleft)) = \{(-,+,+,+),(0,+,+,+),(+,+,+,+)\}\end{array}$$

We note that even though $\mathbf{A}(q_6) = \{(+,+,+,+)\}$ we do **not** get $\mathbf{A}(q_7) = \{(0,+,+,+),(+,+,+,+)\}$ because the analysis of [FM, Section 4.3] does not treat subtraction by 1 any more precisely than subtraction by any other positive number.

Exercise 3 (10%) Information Flow

Consider the program graph C from Figure 1.

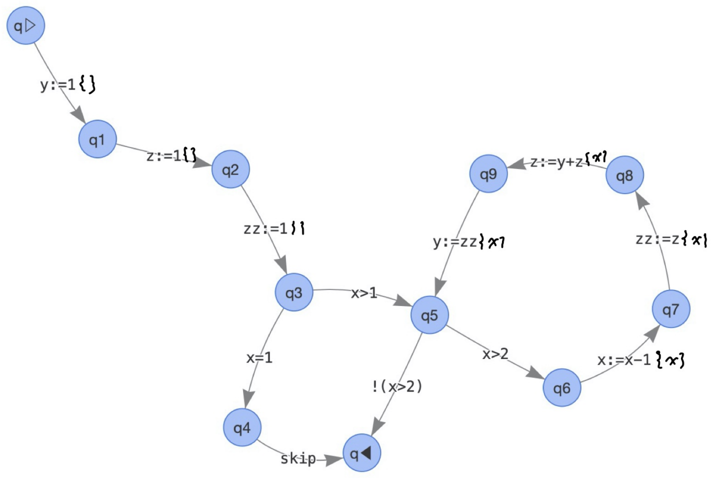
Question 3a: *Write a program in Guarded Commands for which the* **edges** *function of [FM, Section 2.2] generates program graph C.*

Solution (4%):

```
y:=1; z:=1; zz:=1;
if x=1 -> skip
[] x>1 ->
          do x>2 -> x:=x-1;
                    zz:=z;
                    z:=y+z;
                    y:=zz
          od
fi
```

Question 3b: *Draw the instrumented program graph [FM, Section 5.2] for the program constructed above (and that generates the program graph C).*

Solution (4%):



Question 34: *Would we obtain a secure program (in the sense that* **sec** *of [FM, Section 5.3] gives true) if we take* `x` *to be private and* `y`, `z`, *and* `zz` *to be public (using the confidentiality lattice [FM, Figure 5.7])?*
(You should explain your answer.)

Solution (2%): No, because the instrumented program graph shows an implicit flow from `x` to `y`, `z`, and `zz` which would not be permitted according to the confidentiality lattice [FM, Figure 5.7].

Exercise 4 (10%) Model Checking

Consider the program graph C from Figure 1 and the transition system constructed from it as explained in [FM, Section 6.4]. (Since the variables take values in the integers the transition system will have infinitely many states and therefore we shall not draw it.) But let us change the set of atomic propositions to include formulae of the form $x = n$ where x is a program variable (i.e. one of `x`, `y`, `z`, `zz`) and n is an integer. The label function we need is then defined by

$$\mathbf{L}\langle q;\sigma\rangle = \{\triangleright \mid q = q_{\triangleright}\} \cup \{\blacktriangleleft \mid q = q_{\blacktriangleleft}\} \cup \{x = n \mid \sigma(x) = n\}$$

Question 4a: *Does the formula* $\triangleright \wedge \ \mathtt{x} = 5 \Rightarrow \mathbf{EF}\,(\blacktriangleleft \wedge \mathtt{z} = 3)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): No, because the final result is `z`=5.

Question 4b: *Does the formula* $\triangleright \wedge \ \mathtt{x} = 5 \Rightarrow \mathbf{AF}\,(\blacktriangleleft \wedge \mathtt{z} = 3)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): No, because the final result is `z`=5.

Question 4c: *Does the formula* $\triangleright \wedge \ \mathtt{x} = 5 \Rightarrow \mathbf{EF}\,(\blacktriangleleft \wedge \mathtt{z} = 5)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): Yes, because the final result is indeed `z`=5.

Question 4d: *Does the formula* $\triangleright \wedge \ \mathtt{x} = 5 \Rightarrow \mathbf{AF}\,(\blacktriangleleft \wedge \mathtt{z} = 5)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): Yes, because the final result is indeed `z`=5, and the program graph is deterministic.

Question 4e: *Does the formula* $\triangleright \Rightarrow \mathbf{EF} \blacktriangleleft$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): No, the transition system will be stuck at $\langle q_3;\sigma\rangle$ when we have $\sigma(\mathtt{x}) < 1$.

Question 4f: *Does the formula* $\triangleright \Rightarrow \mathbf{AF} \blacktriangleleft$ *hold for the transition system?*

(You should explain your answer but do not have to be formal.)

Solution (1%): No, the transition system will be stuck at $\langle q_3; \sigma \rangle$ when we have $\sigma(\mathtt{x}) < 1$.

Question 4g: *Does the formula* $\triangleright \land\ \mathtt{x} = 5 \Rightarrow \mathbf{EF}\,(\mathtt{x} = 1)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): No, because a value of `x` higher than 1 will never be decremented to 1.

Question 4h: *Does the formula* $\triangleright \land\ \mathtt{x} = 5 \Rightarrow \mathbf{AF}\,(\mathtt{x} = 1)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): No, because a value of `x` higher than 1 will never be decremented to 1.

Question 4i: *Does the formula* $\triangleright \land\ \mathtt{x} = 5 \Rightarrow \mathbf{E}(\mathtt{x} = 5\ \mathbf{U}\ \mathtt{x} = 4)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): Yes, as witnessed by the path $q_\triangleright, q_1, q_2, q_3, q_5, q_6, q_7$.

Question 4j: *Does the formula* $\triangleright \land\ \mathtt{x} = 5 \Rightarrow \mathbf{A}(\mathtt{x} = 5\ \mathbf{U}\ \mathtt{x} = 4)$ *hold for the transition system?*
(You should explain your answer but do not have to be formal.)

Solution (1%): Yes, because the program graph is deterministic and we have the path $q_\triangleright, q_1, q_2, q_3, q_5, q_6, q_7$.

Exercises on Context-free Languages

Exercise 5 (25%)

In the lectures on program verification you were introduced to predicates. The following grammar provides a syntax for predicates similar to the one you have seen in the course:

$$\begin{array}{lcl} F & \rightarrow & \texttt{true} \mid F \texttt{ and } F \mid \texttt{not } F \mid \texttt{exists } V \texttt{ st } F \mid V \texttt{ equals } V \\ V & \rightarrow & \texttt{x} \mid V \texttt{ prime} \end{array}$$

where the set of non-terminal symbols is $N = \{F, V\}$, the set of terminal symbols is $T = \{\texttt{true}, \texttt{and}, \texttt{not}, \texttt{exists}, \texttt{st}, \texttt{equals}, \texttt{x}, \texttt{prime}\}$ and the initial symbol is F. Predicates are all strings accepted by the grammar.

(a) Design a set of datatypes that are suitable to store abstract representations (ASTs) of predicates and show how the formula

```
exists x prime st x equals x prime
```

would be represented as a value of your datatype.

Solution: In F# we could define the following types:

```
type F =
       | True
       | And of F * F
       | Not of F
       | Exists of V * F
       | Equals of V * V
and V =
       | X
       | Prime of V
```

The function signature would be represented by

```
Exists(Prime(X),Equals(X,Prime(X)))
```

(b) Consider the predicate composition operators `and`, `not` and `exist...st`. Fill the table below as follows. In the cell corresponding to row x and column y write YES if, according to the above grammar, the operators x and y can yield an ambiguity. Otherwise, write NO.

	not	and	exists...st
not			
and			
exists...st			

Solution:

	`not`	`and`	`exists...st`
`not`	NO	YES	NO
`and`	YES	YES	YES
`exists...st`	NO	YES	NO

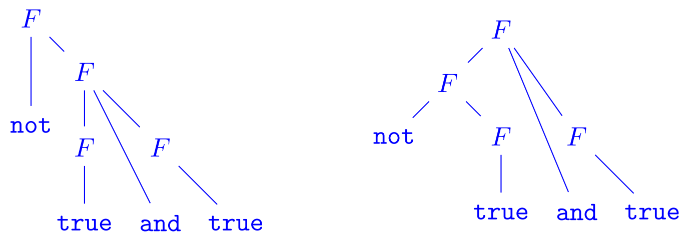
(c) For each of ambiguity identified in (b), showcase the ambiguity by providing an example of a formula that has at least two distinct parse trees.

Solution:

The ambiguity between `not` and `and` is due to undefined precedence and can be showcased by the formula

`not true and true`

and the two distinct parse trees:



The ambiguity between `not` and `exists...st` is due to undefined precedence and can be showcased similarly with the formula:

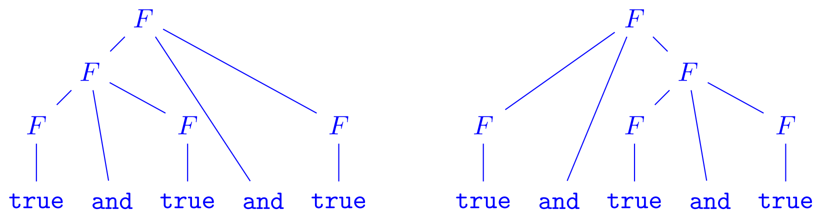
`exists x st true and true`

and the corresponding pair of parse trees (similar to the ones above).

The ambiguity between `and` and `and` is due to undefined associativity and can be showcased with the formula

`true and true and true`

and the two distinct parse trees



(d) Consider the PDA $(\{q\}, T, T \cup N, \delta, q, F, \{q\})$ where the transition function δ is defined as follows:

$$\begin{array}{rcl}
\delta(q, \epsilon, F) & = & \{(q, F\texttt{and}F), (q, V\texttt{equals}V)\} \\
\delta(q, \texttt{true}, F) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{not}, F) & = & \{(q, F)\} \\
\delta(q, \texttt{exists}, F) & = & \{(q, V\texttt{st}F)\} \\
\delta(q, \epsilon, V) & = & \{(q, V\texttt{prime})\} \\
\delta(q, \texttt{x}, V) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{true}, \texttt{true}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{and}, \texttt{and}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{not}, \texttt{not}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{exists}, \texttt{exists}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{st}, \texttt{st}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{equals}, \texttt{equals}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{x}, \texttt{x}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{prime}, \texttt{prime}) & = & \{(q, \epsilon)\}
\end{array}$$

Check if the example formula of exercise (a) is accepted by the PDA. Do so by showing the accepting sequence of configurations (instantaneous descriptions) of the PDA starting with the initial one

$$(q, \texttt{exists x prime st x equals x prime}, F)$$

where q is the initial state of your PDA, `exists x prime st x equals x prime` is the input string and F is the initial stack symbol.

Solution:

```
   (q, exists x prime st x equals x prime, F)
⊢  (q,        x prime st x equals x prime, V st F)
⊢  (q,        x prime st x equals x prime, V prime st F)
⊢  (q,          prime st x equals x prime, prime st F)
⊢  (q,                st x equals x prime, st F)
⊢  (q,                   x equals x prime, F)
⊢  (q,                   x equals x prime, V equals V)
⊢  (q,                     equals x prime, equals V)
⊢  (q,                            x prime, V)
⊢  (q,                            x prime, V prime)
⊢  (q,                              prime, prime)
⊢  (q,                                  ϵ, ϵ)
```

(e) Does the PDA of (d) accept programs that are not accepted by the grammar in (a)? Explain your answer. You do not need to provide a formal answer (e.g. a formal proof); it suffices to provide the main intuitive insights of your argument.

Solution:

It should be easy to recognize that the PDA provided is very similar to the one that one would obtain from the standard construction of PDAs from CFGs. The only difference is that some predictions are "accelerated" (although the PDA is still non-deterministic).

For example, the transitions of the provided PDA for V are

$$\begin{array}{rcl}\delta(q,\epsilon,V) & = & \{(q,V\texttt{prime})\}\\ \delta(q,\texttt{x},V) & = & \{(q,\epsilon)\}\end{array}$$

while with the standard construction we would obtain:

$$\delta'(q,\epsilon,V) \quad = \quad \{(q,V\texttt{prime}),(q,\texttt{x})\}$$

It is easy to see that accelerated transitions of δ as in

$$\begin{array}{rl} & (\mathrm{q},\ \texttt{x}\mathrm{w},\ \mathrm{Vs})\\ \vdash & (\mathrm{q},\ \mathrm{w},\ \mathrm{s})\end{array}$$

can be mimicked by δ' with

$$\begin{array}{rl} & (\mathrm{q},\ \texttt{x}\mathrm{w},\ \mathrm{Vs})\\ \vdash & (\mathrm{q},\ \texttt{x}\mathrm{w},\ \texttt{x}\mathrm{s})\\ \vdash & (\mathrm{q},\ \mathrm{w},\ \mathrm{s})\end{array}$$

The same arguments apply to F.

A formal proof (not requested by the exercise) can be constructed on this main insight.

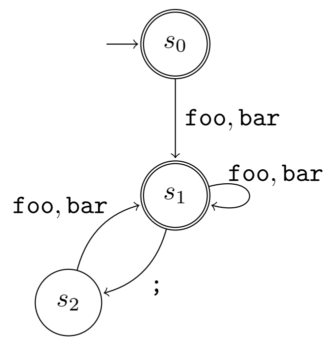
The conclusion is that all words accepted by the PDA are accepted by the grammar.

Exercises on Regular Languages

Exercise 6 (25%)

In this exercise we consider a simple programming language, in which two functions (`foo` and `bar`) that can be composed or invoked in sequence. We will consider the following three proposals for the syntax of the programming language:

(1) The automaton A



(2) The regular expression B

$$(\texttt{foo;} + \texttt{bar;})^*$$

(3) The grammar C

$$\begin{array}{lcl} S & \rightarrow & F \mid S\texttt{;}F \\ F & \rightarrow & \texttt{foo} \mid \texttt{bar} \end{array}$$

where the set $\{\texttt{foo}, \texttt{bar}, \texttt{;}\}$ acts as input alphabet for A and B, and as the set of terminal symbols for C, and where the non-terminal symbols in C are S (initial) and F.

(a) Consider the programs accepted by A, B and C, i.e. their languages $\mathcal{L}(\mathsf{A})$, $\mathcal{L}(\mathsf{B})$, $\mathcal{L}(\mathsf{C})$. Explain, for each pair of languages, their relation: Does one contain the other? Do they overlap? Answer by filling the below tables and explain your answers. You do not need to explain the (trivial) pre-filled cases.

More precisely, in the below table write YES in the cell corresponding to row x and column y if the language x is contained in the language y. Otherwise, write NO.

$\mathcal{L}(\dots) \subseteq \mathcal{L}(\dots)$	$\mathcal{L}(A)$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	YES		
$\mathcal{L}(\mathsf{B})$		YES	
$\mathcal{L}(\mathsf{C})$			YES

Moreover, in the table below write YES in the cell corresponding to row x and column y if the language x overlaps with language y (i.e. they have at least one word/program in common). Otherwise, write NO.

$(\mathcal{L}(\dots) \cap \mathcal{L}(\dots)) \neq \emptyset$	$\mathcal{L}(\mathsf{A})$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	YES		
$\mathcal{L}(\mathsf{B})$		YES	
$\mathcal{L}(\mathsf{C})$			YES

Solution:

$\mathcal{L}(\dots) \subseteq \mathcal{L}(\dots)$	$\mathcal{L}(A)$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	YES	NO	NO
$\mathcal{L}(\mathsf{B})$	NO	YES	NO
$\mathcal{L}(\mathsf{C})$	YES	NO	YES

$(\mathcal{L}(\dots) \cap \mathcal{L}(\dots)) \neq \emptyset$	$\mathcal{L}(\mathsf{A})$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	YES	YES	YES
$\mathcal{L}(\mathsf{B})$	YES	YES	NO
$\mathcal{L}(\mathsf{C})$	YES	NO	YES

A vs B: the languages overlap but neither of them contains the other. This can be shown with three programs:

- ϵ is accepted by both A and B;
- `foo` is accepted by A but not by B;
- `foo;` is accepted by B but not by A.

A vs C: the language of C is strictly contained in the language of A. To see that the language of A is not contained in C we just observe that the empty string is accepted by A but not by C.

To see that the language of C is contained in the language of A we can provide a proof by induction. We need to show that for every string w generated by S in grammar C, w is accepted by A.

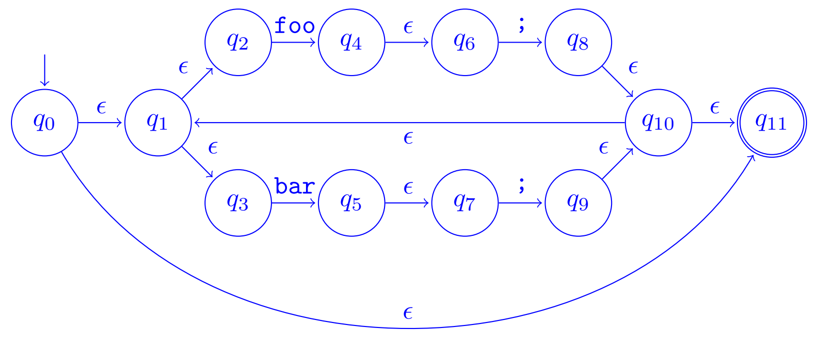
Base cases: The symbol S of grammar C generates strings `foo` and `bar` with production $S \rightarrow F$, which are clearly accepted by A.

Inductive step: Assume that w is derived as follows $S \Rightarrow S\,\texttt{;}\,F \Rightarrow^* w$. Then $w = w'\texttt{;}x$ with $x \in \{\texttt{foo}, \texttt{bar}\}$. The induction hypothesis is that w' is accepted by A. Hence, after reading w' the automaton is in state s_1 (it cannot be in s_0 since w cannot be the empty string). Reading input symbol `;` leads the automaton to state s_2 and reading then input symbol x (for any of the two options) leads the automaton back to the accepting state s_1. Hence, w is accepted by A.

B vs C: the languages do not overlap. B accepts the empty string but C does not. All other strings accepted by B finish with `;`, but no such word can be accepted by C. To see this we observe that the only occurrence of `;` in the body of a production is followed by F, which cannot generate the empty string.

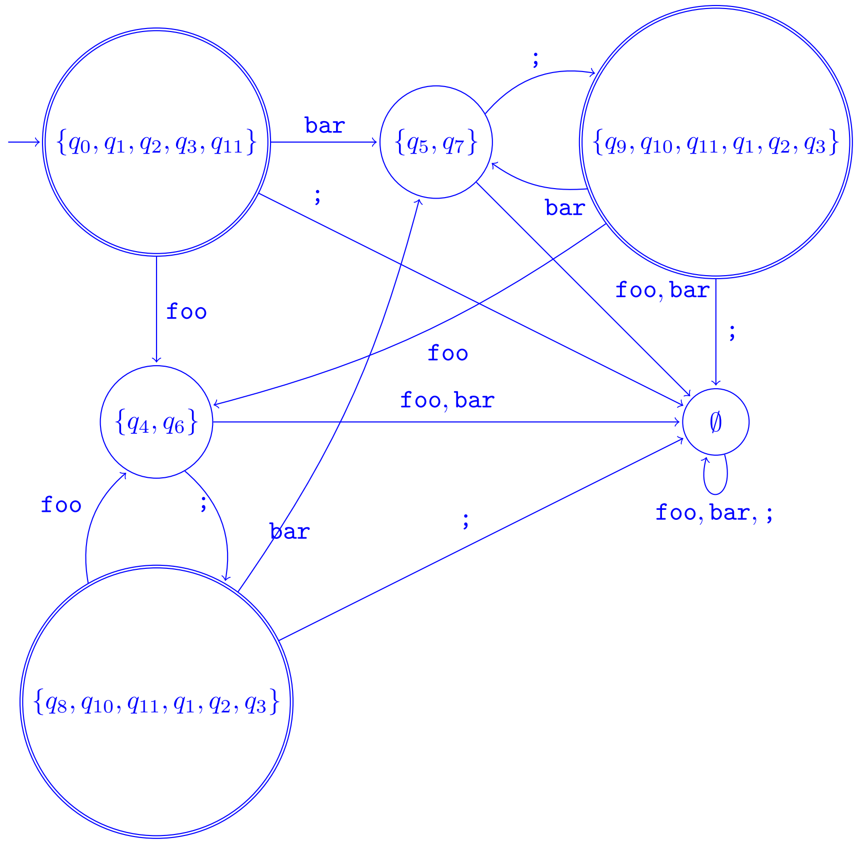
(b) Convert the regular expression B into an ϵ-NFA using the basic algorithm for translating a regular expression into an ϵ-NFA seen in the course. Provide the result as a transition diagram and do not apply any optimisation (i.e. do not remove ϵ-transitions).

Solution:



(c) Convert the ϵ-NFA obtained in (b) into a DFA using the procedure seen in the course. Provide the result as a transition diagram. Do not include unreachable states. Do include the state $\emptyset$ if it is reachable.

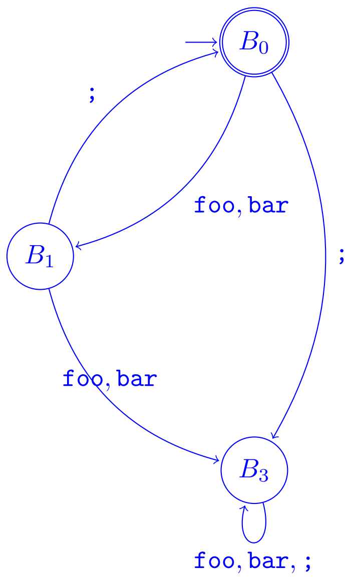
Solution:


$\{q_0, q_1, q_2, q_3, q_{11}\}$
$\{q_5, q_7\}$
$\{q_9, q_{10}, q_{11}, q_1, q_2, q_3\}$
$\{q_4, q_6\}$
$\emptyset$
$\{q_8, q_{10}, q_{11}, q_1, q_2, q_3\}$
bar
;
foo
foo, bar
foo, bar, ;

(d) Is the DFA obtained in (c) minimal or can you find an equivalent one that is smaller? Explain your answer.

Solution:

One solution is to apply the partition refinement algorithm seen in class. This will yield the following minimized DFA:



where

- $B_0 = \{\{q_0, q_1, q_2, q_3, q_{11}\}, \{q_8, q_{10}, q_{11}, q_1, q_2, q_3\}, \{q_9, q_{10}, q_{11}, q_1, q_2, q_3\}\}$
- $B_1 = \{\{q_4, q_6\}, \{q_5, q_7\}\}$
- $B_3 = \{\emptyset\}$

Correctness of the solution (equivalence to the departing automata) follows from the correctness of the algorithm.

Technical University of Denmark

Written examination, May 20, 2022

Page 1 of **??** pages

Course number: 02141

Aids allowed: All aid

The exam is conducted as a digital exam taken at home with the aids allowed for such exams. For this reason answers that can be obtained using a computer need to be properly explained and will not be considered acceptable otherwise. You must submit your answer as a single pdf-file; it is fine to integrate pictures of drawings into the pdf-file, but no attachments to the pdf-file will be considered. You must clearly indicate on each page of your submission which task you attempt to solve.

If you think there are mistakes in the exam set (where in a written exam you would ask to be contacted by the teacher) please send an e-mail to chmat@dtu.dk for Formal Methods and to albl@dtu.dk for Regular Languages and Context Free Languages. Any announcements resulting from this will be communicated over DTU Learn.

Exam duration: 4 hours

Weighting: 7-step scale

Answers Included

Exercise 1 (15%) Semantics

Consider the program graphs PG1 - PG3 in Figures **??** - **??** on page **??**.

Question 1a: *For each of the program graphs on page* **??**, *determine whether there exists a program in Guarded Command Language such that* **edges**$(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\cdots]\!]$ *generates it. If you believe that a suitable program exists, it suffices to provide it. Otherwise, briefly justify why no such program exists.*

Solution : Figure **??**: this is possible.

```
r:=0;
do i<n -> j:=0;
          do j<n -> r:=r+1;
                    j:=j+1
          od;
          i:=i+1
od
```

Figure **??**: not possible; the loop requires some node to have two outgoing edges labeled with Boolean expressions.

Figure **??**: this is possible; one could remark, however, that analyses require that there are no edges flowing into the initial node.

```
do x <= y ->
    if
       x < y -> y := x
    [] !(x < y) -> y := x
    fi
[] x >= y ->
    if
       y > 0 -> y := x
    [] x > 0 -> x := y
    fi
od
```

Question 1b: *For the program graphs PG2 and PG3 in Figures* **??** *and* **??** *on page* **??**, *how many different execution sequences of the form*

$$\langle q_\triangleright; [\mathtt{x} \mapsto 5, \mathtt{y} \mapsto 5]\rangle \overset{\omega}{\Longrightarrow}^* \langle q_\blacktriangleleft; \sigma\rangle$$

exist for each program graph? For each such execution sequence, also provide σ. *(You do not need to write entire execution sequences.)*

Solution :

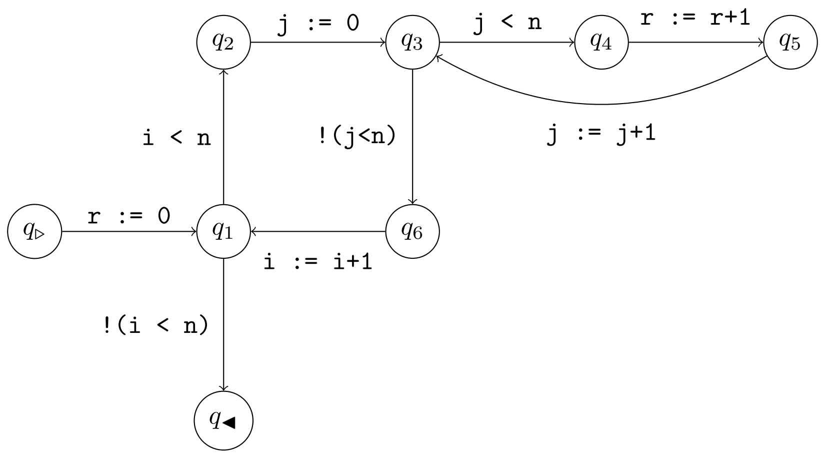


Figure 1: Program graph `PG1`

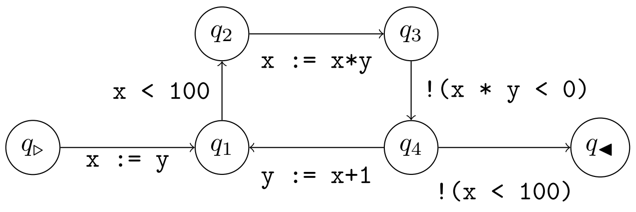


Figure 2: Program graph `PG2`

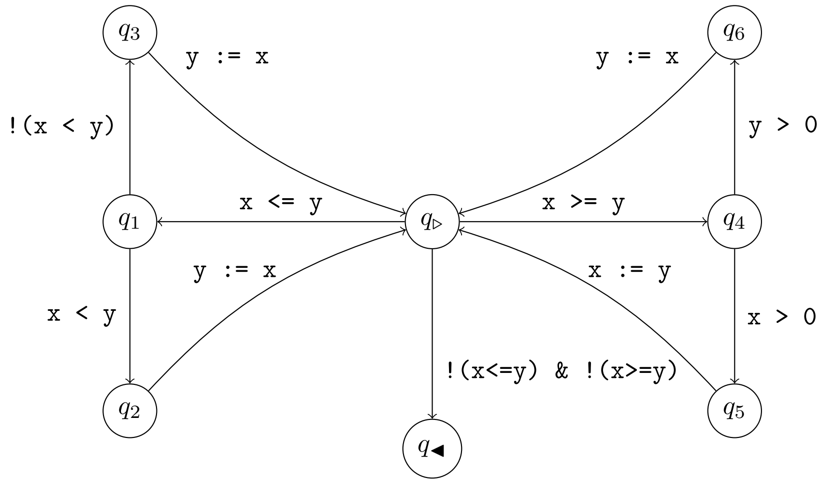


Figure 3: Program graph `PG3`

- Figure **??**: there is only one execution sequence that terminates; the corresponding memory is $\sigma = [x = 650, y = 26]$. All other execution sequences (note that the PG is not deterministic in q_4!) get stuck in q_1.
- Figure **??**: There is no terminating execution sequence, since the condition `!(x<=y) & !(x>=y)` is unsatisfiable.

Question 1c: *For each of the program graphs* `PG1` *-* `PG3` *in Figures* **??** *-* **??** *on page* **??**, *determine whether it is an evolving system. Justify your answer.*

Solution :

- Figures **??**: this is an evolving system; it suffices to check the definition, particularly for q_1 and q_3.
- Figures **??**: not an evolving system since it can get stuck in q_1 as noted in the previous task.
- Figures **??**: not an evolving system, since $[x \mapsto -1, y \mapsto -2]$ may get stuck in q_4.

Question 1d: *An alternative definition of a deterministic system, called an* observationally deterministic system *in the following, is as follows: for all memories* $\sigma, \sigma', \sigma''$ *and all* ω, ω', *we have*

$$\langle q_\triangleright; \sigma\rangle \overset{\omega}{\Longrightarrow}^* \langle q_\blacktriangleleft; \sigma'\rangle \text{ and } \langle q_\triangleright; \sigma\rangle \overset{\omega'}{\Longrightarrow}^* \langle q_\blacktriangleleft; \sigma''\rangle \text{ implies } \sigma' = \sigma''.$$

Give a program graph that is both an evolving system and an observationally deterministic system but not a deterministic system in the sense of [FM]. Justify your answer.

Solution :

```
if x=x -> x := 17
[] x+1=x+1 -> x := 17
fi
```

The above program is observationally deterministic (all executions terminate with `x=17`). However, they may take different actions and, consequently, do not constitute a deterministic system.

Question 1e: *We extend the admissible actions of the Guarded Command Language considered in [FM]: We add a new statement*

```
x := a1 DIV a2 REM y
```

to the Guarded Command Language, which performs an integer division of the value of arithmetic expression `a1` *by the value of arithmetic expression* `a2`*; the result (which is always rounded down) is assigned to variable* `x` *and the remainder is assigned to variable* `y` *afterward.*

Extend the definition of the semantics function $\mathcal{S}[\![\ldots]\!]$ in [FM] by a new action for the statement `x := a1 DIV a2 REM y` *from above.*

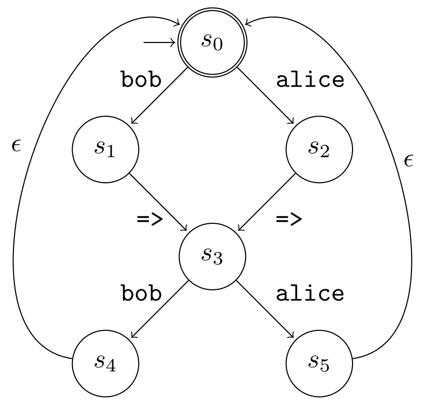
Solution :

$$\mathcal{S}[\![\texttt{x := a1 DIV a2 REM y}]\!]\sigma = \begin{cases} \sigma[x \mapsto u][y \mapsto v], & \text{if } \mathcal{A}[\![a_1]\!]\sigma = r, \mathcal{A}[\![a_2]\!]\sigma = s, \\ & r, s \text{ defined}, \\ & s \neq 0, u = \lfloor r/s \rfloor, v = r\%s \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Exercise 2 (20%) Formal Languages

In this exercise we consider languages for describing information flows. In particular, we consider the following 3 languages:

(1) The automaton A



(2) The regular expression B

$$(\texttt{(alice+bob)=>(alice+bob)})^+$$

(3) The grammar C

$$\begin{array}{lcl} S & \rightarrow & SS \mid F \mid \epsilon \\ F & \rightarrow & \texttt{alice=>bob} \mid \texttt{bob=>alice} \end{array}$$

where the set $\{\texttt{alice}, \texttt{bob}, \texttt{=>}\}$ acts as input alphabet for A and B, and as the set of terminal symbols for C, and where the non-terminal symbols in C are S (initial) and F.

(a) Consider the strings accepted by A, B and C, i.e. their languages $\mathcal{L}(\mathsf{A})$, $\mathcal{L}(\mathsf{B})$, $\mathcal{L}(\mathsf{C})$.

Fill the table below as follows: in the cell corresponding to row x and column y write YES if the language x overlaps with language y and provide a (preferably short) string contained in both languages. Otherwise, write NO and briefly explain (in 2-3 lines) why the two languages do not overlap.

You do not need consider the diagonal.

$(\mathcal{L}(\dots) \cap \mathcal{L}(\dots)) \neq \emptyset$	$\mathcal{L}(\mathsf{A})$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	—		
$\mathcal{L}(\mathsf{B})$		—	
$\mathcal{L}(\mathsf{C})$			—

Solution:

$(\mathcal{L}(\dots) \cap \mathcal{L}(\dots)) \neq \emptyset$	$\mathcal{L}(\mathsf{A})$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	—	YES: `alice => bob`	YES: ϵ
$\mathcal{L}(\mathsf{B})$	YES: `alice => bob`	—	YES: `alice => bob`
$\mathcal{L}(\mathsf{C})$	YES: ϵ	YES: `alice => bob`	—

(b) Consider again the sets of strings accepted by A, B and C, that is, the languages $\mathcal{L}(\mathsf{A})$, $\mathcal{L}(\mathsf{B})$, $\mathcal{L}(\mathsf{C})$.

Fill the below table as follows: in the cell corresponding to row x and column y write YES and a brief explanation (in 2-3 lines) if the language x is contained in the language. Otherwise, write NO and provide a (preferably short) string that is accepted by x but not by y.

You do not need consider the diagonal.

$\mathcal{L}(\dots) \subseteq \mathcal{L}(\dots)$	$\mathcal{L}(A)$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	—		
$\mathcal{L}(\mathsf{B})$		—	
$\mathcal{L}(\mathsf{C})$			—

Solution:

$\mathcal{L}(\dots) \subseteq \mathcal{L}(\dots)$	$\mathcal{L}(A)$	$\mathcal{L}(\mathsf{B})$	$\mathcal{L}(\mathsf{C})$
$\mathcal{L}(\mathsf{A})$	—	NO: ϵ	NO: `alice => alice`
$\mathcal{L}(\mathsf{B})$	YES	—	NO: `alice => bob`
$\mathcal{L}(\mathsf{C})$	YES	NO: ϵ	—

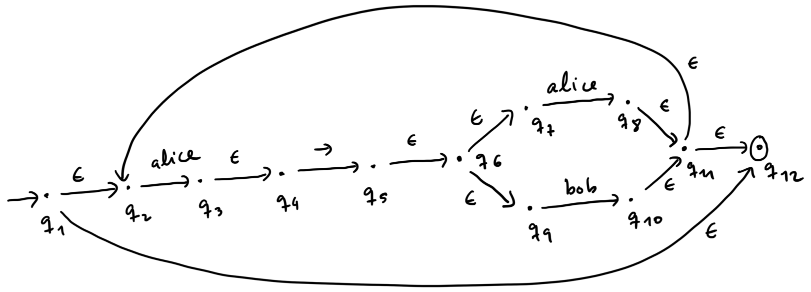
B is contained in A: Both languages consist of sequences x`=>`y where x and y can both be `alice` or `bob`. The only difference is that ϵ is accepted by A but not by B. This could be proven by adding ϵ to B (e.g. replacing $^+$ by *), transforming both into DFA and running an equivalence check.

C is contained in A: Both languages consist of sequences x`=>`y with x and y in $\{$`alice`$,$`bob`$\}$. However, in C x and y must be different, whereas in A they can be equal.

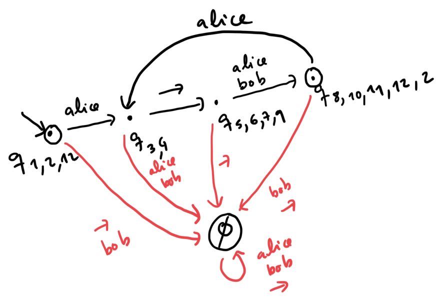
(c) Consider the regular expression $\mathsf{D} = ($`alice=>`$($`alice`$+$`bob`$))^*$. Use the transformations seen in the course to obtain an equivalent DFA. Briefly explain each step (that is, what you did and why you did it).

Solution:

Transforming the regular expression into an ϵ-NFA yieds

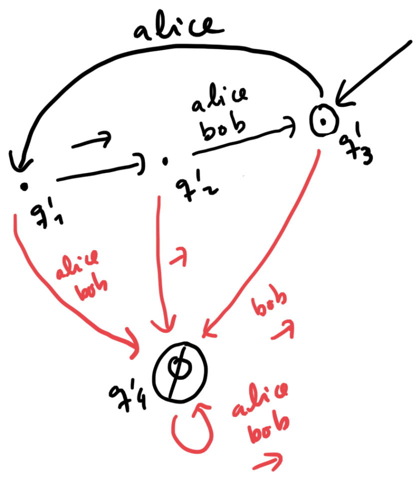


We can then transform the ϵ-NFA into a DFA applying subset construction and epsilon closure to obtain



where a state name q_i, j, k abbreviates a state $\{q_i, q_k, q_k\}$:

We can also apply minimisation to obtain (after some renaming)



(d) Consider the following languages. For each language L, state whether or not L is a regular language. Briefly explain your answer in 2-3 lines.

- The language $\mathcal{L}(\mathsf{A})$, where A is described at the top of the exercise.
- The language $\mathcal{L}(\mathsf{B})$, where B is described at the top of the exercise.
- The language $\mathcal{L}(\mathsf{C})$, where C is described at the top of the exercise.
- The language L_4 of all strings accepted by A but not by C.
- The language L_5 of all strings accepted by C, where the substring `alice=>bob` appears as many times as the substring `bob=>alice`.

Solution:

- $\mathcal{L}(\mathsf{A})$ the language of automaton A described at the beginning of the exercise. This language is trivially regular, since it is defined by a ϵ-NFA automaton.

- $\mathcal{L}(\mathsf{B})$ the language of the regular expression B described at the beginning of the exercise. This language is trivially regular, since it is defined by a regular expression.
- $\mathcal{L}(\mathsf{C})$ the language of the grammar C described at the beginning of the exercise. This language is regular. It is fairly easy to see that it is the language of possibly empty sequences of F's, which can be easily described with DFA (e.g. slight modification of A by "splitting" state q_3) or a regular expression (e.g. $(\texttt{alice => bob} + \texttt{bob => alice})^*$)
- L_4, the language of all strings accepted by both A but not by C. This language is regular. Since $\mathcal{L}(\mathsf{C})$ is regular, its complement is regular too. Since also $\mathcal{L}(\mathsf{A})$ is regular, the intersection of $\mathcal{L}(\mathsf{A})$ and the complement of C is a regular language.
- L_5, the language of all strings accepted by C, where the substring `alice=>bob` appears as many times as the substring `bob=>alice`. This language is not regular. A simple argument is to observe that replacing `alice=>bob` and `bob=>alice` by "(" and ")" would yield a language of balanced parenthesis, which is not regular. Intuitively, an automaton would need to count the occurrences of the substrings, and this would require as many states as the natural numbers (hence not doable with finite states).

Exercise 3 (15%) Program Verification

Question 3a: *For each contract below (using notation as on formalmethods.dk), argue whether it is correct or not (you do not have to construct proof trees). Here,* `x`, `y`, `z` *are program variables and* `A` *and* `B` *are arrays of length* `n`.

(a)
```
{ x > 3 }
  if x >= 0 -> x := x - 3
  [] x >= 5 -> x := x + y*y
  [] x >= 7 -> x := x + y
  fi
{ x >= 0 }
```

Solution : The contract is *not* correct. For example, assume $\sigma(x) = 8$ and $\sigma(y) = -10$. Then the third branch of the conditional is enabled, resulting in a final memory σ' with $\sigma'(x) = -2 < 0$.

(b)
```
{ x = 23 }
  do x > 0 -> x := x - 2
  [] x < 0 -> x := x + 4
  od
{ x = 421723 }
```

Solution : Since x is initially odd and positive, the loop never terminates: it eventually ends up in a loop $x = 1, -1, 3, 1, \ldots$ Consequently, there exist no terminating execution sequences starting with $x = 23$ and the contract is trivially correct.

(c)
```
{0 <= x < n && 0 <= y < n && A[x] == 42 && A[y] == 17 }
  z := A[x]; A[x] := A[y]; A[y] := z
{0 <= x < n && 0 <= y < n && A[x] == 17 && A[y] == 42 }
```

Solution : The contract is correct. The conditions on x and y ensure that all array accesses are within the array bounds. The program then swaps two array elements as stated by the contract.

(d)
```
{x = max {A[i] | 0 <= i < n } }
  x := A[y]
{A[y] = max {A[i] | 0 <= i < n } }
```

Solution : The contract is not correct. For example, if $A = [1, 2]$ and $x = 2$, $y = 0$, then $x = 2$ holds in the precondition, but $A[0] = 2$ does not hold in the postcondition.

Question 3b: *Show that the contract in Figure* **??** *is valid with respect to partial correctness. You may either construct a proof tree (an annotated program) as in the lecture or derive and analyse shortest path fragments as in [FM].*

```
{ n >= 0 }
i := 0;
res := 0;
{ n >= 0 && i <= n && res = n * i }
do i < n ->
    j := 0;
    do j < n ->
        res := res + 1;
        j := j + 1
    od;
    i := i + 1
od
{ res = n^2 }
```

Figure 4: Contract for program C

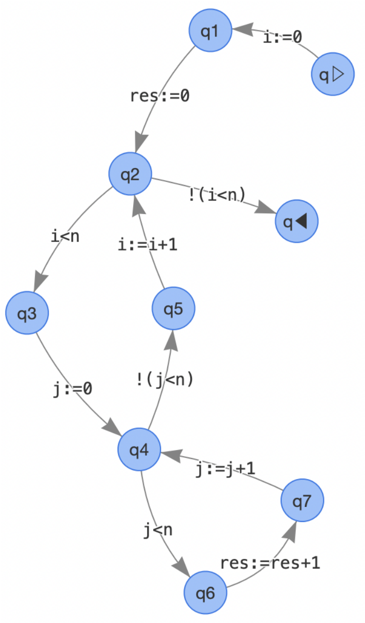


Figure 5: Program graph for C

Solution : We construct a proof tree:

```
{ n >= 0 }
{ n >= 0 && 0 <= n && 0 = n * 0 }
i := 0;
{ n >= 0 && i <= n && 0 = n * i }
res := 0;
{ n >= 0 && i <= n && res = n * i }
do i < n ->
    { n >= 0 && i <= n && res = n * i && i < n }
    { n >= 0 && i <= n && res = n * i }
    { n >= 0 && i < n && res = n * i + 0 && 0 <= n }
    j := 0;
    { n >= 0 && i < n && res = n * i + j && j <= n }
    do j < n ->
        { n >= 0 && i < n && res = n * i + j && j <= n && j < n }
        { n >= 0 && i < n && res + 1 = n * i + j + 1 && j + 1 <= n }
        res := res + 1;
        { n >= 0 && i < n && res = n * i + j + 1 && j + 1 <= n }
        j := j + 1
        { n >= 0 && i < n && res = n * i + j && j <= n }
    od;
    { n >= 0 && i < n && res = n * i + j && j <= n && !(j < n) }
    { n >= 0 && i+1 <= n && res = n * (i+1) }
    i := i + 1
    { n >= 0 && i <= n && res = n * i }
od
{ n >= 0 && i <= n && res = n * i && !(i < n) }
{ res = n^2 }
```

Exercise 4 (13%) Program Analysis

Question 4a: *Explain whether the following statements about the program graphs* `PG4` *and* `PG5` *in Figures* **??** - **??** *(on page* **??***) are* `true`, `false`, *or whether no conclusive answer is possible (*`unknown`*). Your answers must be* based on the results of the detection of signs analysis *in [FM] and implemented on formalmethods.dk (that is, you may only inspect individual actions and the analysis information obtained for adjacent nodes). Explain your answer to each question.*

(a) Upon termination of `PG4`, x will be non-negative.

Solution : `unknown`; the detection of sign analysis produces the abstract memory $[x \mapsto -, y \mapsto +]$ for the final node, which might be reachable.

(b) `PG4` never divides by zero.

Solution : `true`; in nodes q_2 and q_3, the detection of sign analysis rules out that $x = 0$.

(c) There is at least one execution of `PG5` that terminates.

Solution : `true`; the detection of sign analysis produces a concrete execution for $\mathtt{x} = 0 = \mathtt{y}$.

(d) In `PG5`, the sign of variable `x` in node q_1 coincides with the sign of `x` in the initial node $q_\triangleright$.

Solution : `unknown`; this is obviously true by inspecting the given program. However, our detection of signs analysis is not precise enough to guarantee this. Since $+1$ and -1 get abstracted to $+$ and $-$, respectively, the analysis yields an arbitrary sign after executing $x := x + 1; x := x - 1$.

The detection of signs analysis in [FM] often loses precisions for updates involving small integers. We will thus extend it such that the numbers $1, 2$ and their negative counterparts are treated more precisely, similar to 0. That is, our refined analysis should detect whether a variable (or array element) has value $-2, -1, 0, 1, 2$, a value smaller than -2 (indicated by < -2) or a value greater than 2 (indicated by > 2).

Question 4b: *Give a formal definition of the set* $\widehat{\textbf{Mem}}$ *of abstract memories that can be used for the refined detection of signs analysis.*

$$(\hat{\sigma}_1, \hat{\sigma}_2) \in \widehat{\textbf{Mem}} \ =$$

```
x := 2;
if
   true -> y := -2/x
[] true -> y := x/3
fi;
x := y / -x
```

Figure 6: Code for `PG4`

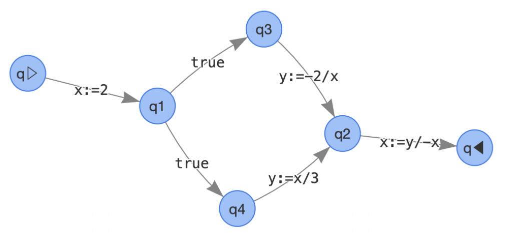


Figure 7: Program graph `PG4`

```
y := x;
do
  y > 0 -> x := x - 1;
           y := y - x;
           x := x +1
[]
  y < 0 -> x := x + 1;
           y := y + x;
           x := x - 1
od
```

Figure 8: Code for `PG5`

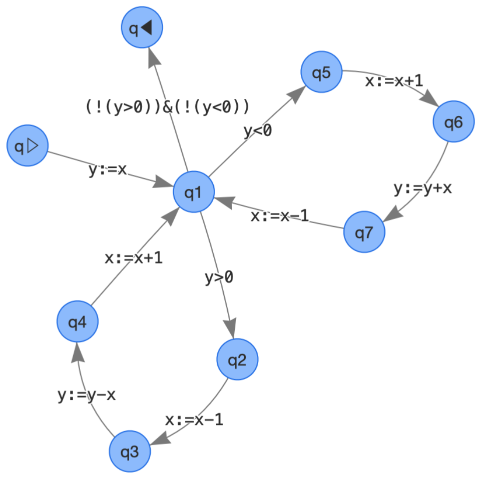


Figure 9: Program graph `PG5`

Solution :

$$
\begin{aligned}
(\hat{\sigma}_1, \hat{\sigma}_2) \in \widehat{\mathbf{Mem}} \;=\; \{(\hat{\sigma}_1, \hat{\sigma}_2) \,|\hat{\sigma}_1 \colon \texttt{Var} \to \{< -2, -2, -1, 0, 1, 2, > 2\} \\
\hat{\sigma}_2 \colon \texttt{Arr} \to \{< -2 - 2, -1, 0, 1, 2, > 2\}\}
\end{aligned}
$$

Question 4c: *Apply the refined detection of signs analysis to compute an analysis assignment for the program graph in Figure* **??** *on page* **??**. *You may assume that all program variables are initially larger than 2. Briefly explain your answer, particularly when performing a division operation.*

$$
\begin{aligned}
\mathbf{A}(q_{\triangleright}) &= \{\,[\,\texttt{x} \mapsto > 2,\ \texttt{y} \mapsto > 2\,]\,\} \\
\mathbf{A}(q_1) &= \\
\mathbf{A}(q_3) &= \\
\mathbf{A}(q_4) &= \\
\mathbf{A}(q_2) &= \\
\mathbf{A}(q_{\blacktriangleleft}) &=
\end{aligned}
$$

Solution :

$$
\begin{aligned}
\mathbf{A}(q_{\triangleright}) &= \{\,[\,\texttt{x} \mapsto > 2,\ \texttt{y} \mapsto > 2\,]\,\} \\
\mathbf{A}(q_1) &= \{\,[\,\texttt{x} \mapsto\ 2,\ \texttt{y} \mapsto > 2\,]\,\} \\
\mathbf{A}(q_3) &= \{\,[\,\texttt{x} \mapsto\ 2,\ \texttt{y} \mapsto > 2\,]\,\} \\
\mathbf{A}(q_4) &= \{\,[\,\texttt{x} \mapsto\ 2,\ \texttt{y} \mapsto > 2\,]\,\} \\
\mathbf{A}(q_2) &= \{\,[\,\texttt{x} \mapsto\ 2,\ \texttt{y} \mapsto\ -1\,]\,\} \cup \{\,[\,\texttt{x} \mapsto\ 2,\ \texttt{y} \mapsto\ 0\,]\,\} \\
\mathbf{A}(q_{\blacktriangleleft}) &= \{\,[\,\texttt{x} \mapsto\ 0,\ \texttt{y} \mapsto\ -1\,]\,\} \cup \{\,[\,\texttt{x} \mapsto\ 0,\ \texttt{y} \mapsto\ 0\,]\,\}
\end{aligned}
$$

$\tilde{/}$	<-2	-2	-1	0	1	2	>2
<-2	$\{0, 1, 2, > 2\}$	$\{1, 2, > 2\}$	$\{> 2\}$	$\{\}$	$\{< -2\}$	$\{-1, -2, < -2\}$	$\{0, -1, -2, < -2\}$
-2	$\{0\}$	$\{1\}$	$\{2\}$	$\{\}$	$\{-2\}$	$\{-1\}$	$\{0\}$
-1	$\{0\}$	$\{0\}$	$\{1\}$	$\{\}$	$\{-1\}$	$\{0\}$	$\{0\}$
0	$\{0\}$	$\{0\}$	$\{0\}$	$\{\}$	$\{0\}$	$\{0\}$	$\{0\}$
1	$\{0\}$	$\{0\}$	$\{-1\}$	$\{\}$	$\{1\}$	$\{0\}$	$\{0\}$
2	$\{0\}$	$\{-1\}$	$\{-2\}$	$\{\}$	$\{2\}$	$\{1\}$	$\{0\}$
>2	$\{0, -1, -2, < -2\}$	$\{-1, -2, < -2\}$	$\{< -2\}$	$\{\}$	$\{> 2\}$	$\{1, 2, > 2\}$	$\{0, 1, 2, > 2\}$

Exercise 5 (7%) Information Flow

Let u and v be the only two variables in a GCL program and let us consider its deterministic program graph with initial state $q_{\triangleright}$ and final state $q_{\blacktriangleleft}$. We say that *"u interferes with v"* if we can find two memories σ_1 and σ_2 such that:

(i) $\sigma_1(v) = \sigma_2(v)$

(ii) $\langle q_{\triangleright}, \sigma_1 \rangle \Rightarrow^* \langle q_{\blacktriangleleft}, \sigma'_1 \rangle$

(iii) $\langle q_{\triangleright}, \sigma_2 \rangle \Rightarrow^* \langle q_{\blacktriangleleft}, \sigma'_2 \rangle$

(iv) $\sigma'_1(v) \neq \sigma'_2(v)$

NOTE: although u does not explicitly appear in the definition above, it is of course in the domain of the memories $\sigma_1, \sigma_2, \ldots$.

Consider the following GCL programs:

(a) `x := y`

(b) `y := x ; x := 0`

(c) `if y>0 -> y:=y-1 [] true -> x:=1 fi`

(d) `if x=0 -> y:=x [] true -> y:=0 fi`

For each program

- Consider a confidentiality policy public < private and specify a security assignment for `x` and `y` such that the program is not secure w.r.t to the notion of security in [FM]. Explain why the program is not secure in 2-3 lines.
- Given the security assignment, determine if the private variable (if any) interferes with the public variable (if any). If you state that there is an interference, then provide memories σ_1 and σ_2 with the above stated properties (i–iv). Otherwise, explain why such pair of memories do not exist.

Solution :

(a) `x := y`. With assignment $\texttt{x} \mapsto \textsf{public}, \texttt{y} \mapsto \textsf{private}$ the program is not secure since there is an forbidden flow $\texttt{y} \rightarrow \texttt{x}$. Variable `y` interferes with `x`. This is witnessed with the pair of executions

$$\langle q_{\triangleright}, (\texttt{x} \mapsto 0, \texttt{y} \mapsto 0) \rangle \Rightarrow^* \langle q_{\blacktriangleleft}, (\texttt{x} \mapsto 0, \texttt{y} \mapsto 0) \rangle$$
$$\langle q_{\triangleright}, (\texttt{x} \mapsto 0, \texttt{y} \mapsto 1) \rangle \Rightarrow^* \langle q_{\blacktriangleleft}, (\texttt{x} \mapsto 1, \texttt{y} \mapsto 1) \rangle$$

(b) `y := x ; x := 0`. With assignment $\mathtt{x} \mapsto \mathsf{private}, \mathtt{y} \mapsto \mathsf{public}$ the program is not secure since there is a forbidden flow $\mathtt{x} \to \mathtt{y}$.

Variable `x` interferes with `y`. This is witnessed with the pair of executions

$$\langle q_{\triangleright}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 0)\rangle \Rightarrow^{*} \langle q_{\blacktriangleleft}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 0)\rangle$$
$$\langle q_{\triangleright}, (\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 0)\rangle \Rightarrow^{*} \langle q_{\blacktriangleleft}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 1)\rangle$$

(c) `if y>0 -> y:=y=-1 [] true -> x:=1 fi`. With assignment $\mathtt{x} \mapsto \mathsf{public}, \mathtt{y} \mapsto \mathsf{private}$ the program is not secure since there is an implicit forbidden flow $\mathtt{y} \to \mathtt{x}$. Variable `y` interferes with `x`. This is witnessed with the pair of executions

$$\langle q_{\triangleright}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 0)\rangle \Rightarrow^{*} \langle q_{\blacktriangleleft}, (\mathtt{x} \mapsto 1, \mathtt{y} \mapsto 0)\rangle$$
$$\langle q_{\triangleright}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 1)\rangle \Rightarrow^{*} \langle q_{\blacktriangleleft}, (\mathtt{x} \mapsto 0, \mathtt{y} \mapsto 0)\rangle$$

(d) `if x=0 -> y:=x [] true -> y:=0 fi`. With assignment $\mathtt{y} \mapsto \mathsf{public}, \mathtt{x} \mapsto \mathsf{private}$ the program is not secure since there is a forbidden flow $\mathtt{x} \to \mathtt{y}$. However, `x` does not interfere with `y` since in the final state, `y` is always 0.

Exercise 6 (20%) Context-free Languages

The following context-free grammar provides a syntax for a new language that we call *Guarded Command Modelling Language (GCML)*:

$$\begin{array}{lcll} C & \to & A & \text{(actions)} \\ & \mid & C\,\texttt{;}\,C & \text{(sequencing)} \\ & \mid & C\,\texttt{[]}\,C & \text{(branching)} \\ & \mid & C\,\texttt{!} & \text{(looping)} \\ & \mid & \texttt{[}C\texttt{]} & \text{(grouping)} \\ A & \to & \texttt{skip} \mid \texttt{assign} & \end{array}$$

where the set of non-terminal symbols is $N = \{C, A\}$, the set of terminal symbols is $T = \{\texttt{skip}, \texttt{assign}, \texttt{;}, \texttt{[}, \texttt{]}, \texttt{!}\}$ and the initial symbol is C. We call a string accepted by the grammar a "model".

(a) Design a set of datatypes that are suitable to store abstract representations (ASTs) of models and show how the model

```
[ assign ; [assign [] skip ] ] !
```

would be represented as a value of your datatype.

Solution: In F# we could define the following types:

```
type C =
       | Action of A
       | Seq of C * C
       | Branch of C * C
       | Loop of C
and A =
       | Skip
       | Assign
```

The string/model would be represented by

```
Loop(Seq(Assign,Branch(Skip,Assign)))
```

As usual explicit operators "grouping" is optional.

(b) Consider the following operators for composing models: `_;_`, `_[]_`, `_!` and `[_]`. Fill the table below as follows: in the cell corresponding to row x and column y write YES if, according to the above grammar (without any additional operator precedence), the operators x and y can yield an ambiguity. Otherwise, leave the cell blank.

	`_;_`	`_[]_`	`_!`	`[_]`
`_;_`				
`_[]_`				
`_!`				
`[_]`				

Solution:

	`_;_`	`_[]_`	`_!`	`[_]`
`_;_`	YES	YES	YES	
`_[]_`	YES	YES	YES	
`_!`	YES	YES		
`[_]`				

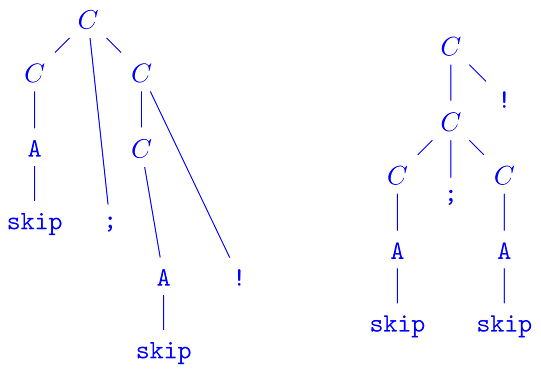
(c) If you have discovered at least one ambiguity in (b), showcase it by providing an example of a model that has at least two distinct parse trees. Otherwise, explain in 2-3 lines why no string/model exists that can have two parse trees.

Solution:

The ambiguity between `;` and `!` is due to undefined precedence and can be showcased by the model

`skip ; skip !`

and the two distinct parse trees:



The rest of the ambiguities can be similarly be showcased.

(d) If you have discovered at least one ambiguity in (b), provide change the grammar to remove the ambiguity. You do not need to solve all ambiguities you have discovered. Explain in 2-3 lines the change that you have done and why it solves the ambiguity.

Solution:

A possible way to solve *all* ambiguities is to make the two model composition operators `_;_` and `_ []_` left-associative, and to impose the following precedence for the operators: `_;_` < `_ []_` < `_ !` and `[_ ]`.

We can impose those rules by using the techniques seen in class (allowing left-recursion only and applying stratification):

$$\begin{array}{rcl} C_0 & \rightarrow & C_0\,\texttt{;}\,C_1 \mid C_1 \\ C_1 & \rightarrow & C_1\,\texttt{[]}\,C_2 \mid C_2 \\ C_2 & \rightarrow & C_2\,\texttt{!} \mid \texttt{[}C_0\texttt{]} \mid A \\ A & \rightarrow & \texttt{skip} \mid \texttt{assign} \end{array}$$

(e) Construct a Pushdown Automaton (PDA) that accepts the same language as the grammar described at the beginning of the exercise, and show how it runs on the string `[skip]` by providing an accepting sequence of configurations (aka instantaneous descriptions). How many steps does your PDA take to recognize the string?

Solution: If we apply the standard construction seen in class (and described in book [HMU]) to translate a context-free grammar into an equivalent, non-deterministic PDA we obtain the following PDA (which accepts by empty stack):
$(\{q\}, T, T \cup N, \delta, P, \{q\})$ where the transition function δ is defined as follows:

$$\begin{array}{rcll}
\delta(q,\epsilon,C) & = & \{(q,A),(q,C\,\texttt{;}\,C),(q,C\,\texttt{[]}\,C),(q,C\,\texttt{!}),(q,\texttt{[}C\texttt{]})\} & \\
\delta(q,\epsilon,A) & = & \{(q,\texttt{skip}),(q,\texttt{assign})\} & \\
\delta(q,t,t) & = & \{(q,\epsilon)\} & \text{if } t \in T.
\end{array}$$

It takes 6 transitions to recognize the string:

$$\begin{array}{ll}
 & (\text{q}, \texttt{[skip]}, \text{C}) \\
\vdash & (\text{q}, \texttt{[skip]}, \texttt{[}\ \text{C}\ \texttt{]}) \\
\vdash & (\text{q}, \texttt{skip]}, \text{C}\ \texttt{]}) \\
\vdash & (\text{q}, \texttt{skip]}, \text{A}\ \texttt{]}) \\
\vdash & (\text{q}, \texttt{skip]}, \texttt{skip]}) \\
\vdash & (\text{q}, \texttt{]}, \texttt{]})
\end{array}$$

(f) Can the string discussed in (e) be recognized in less steps by another (equivalent) PDA? If no, explain why in 2-3 lines. If yes, provide the new PDA, show how it would recognize the string in less steps, and explain in 2-3 lines the idea behind the new PDA.

Solution:
We can reduce the number of transitions it takes to recognize strings by observing that strings to be parsed as C which start by `[` can only be generated by the production with body `[`C`]`, and, similarly, strings starting with `skip` and `assign` to be parsed as A can only be obtained with one production respectively. We can implement this idea in the following PDA:

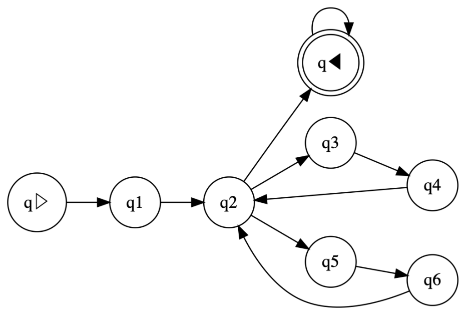
$$\begin{array}{rcll}
\delta(q,\epsilon,C) & = & \{(q,A),(q,C\,\texttt{;}\,C),(q,C\,\texttt{[]}\,C),(q,C\,\texttt{!})\} & \\
\delta(q,\texttt{[},C) & = & \{(q,C\texttt{]})\} & \\
\delta(q,\epsilon,C) & = & \{(q,A),(q,C\,\texttt{;}\,C),(q,C\,\texttt{[]}\,C),(q,C\,\texttt{!}),(q,\texttt{[}C\texttt{]})\} & \\
\delta(q,\texttt{skip},A) & = & \{(q,\epsilon)\} & \\
\delta(q,\texttt{assign},A) & = & \{(q,\epsilon)\} & \\
\delta(q,t,t) & = & \{(q,\epsilon)\} & \text{if } t \in T.
\end{array}$$

we can now recognize the string in 4 transitions:

$$\begin{array}{ll}
 & (\mathrm{q},\ \texttt{[skip]},\ \mathrm{C}) \\
\vdash & (\mathrm{q},\ \texttt{skip]},\ \mathrm{C}\ \texttt{]}) \\
\vdash & (\mathrm{q},\ \texttt{skip]},\ \mathrm{A}\ \texttt{]}) \\
\vdash & (\mathrm{q},\ \texttt{]},\ \texttt{]}) \\
\vdash & (q,\ \epsilon,\ \epsilon)
\end{array}$$

Exercise 7 (10%) Model Checking

Consider the following transition system:



where

- the set of initial states contains only the state $q_\triangleright$;
- the set of atomic propositions are the names of the states, and
- the labelling function for states therefore trivially maps a state to the set consisting of just that state.

Recall that a state formula Φ of CTL holds on a transition system if and only if it holds on all initial states, which in this case is merely the initial node $q_\triangleright$.

Next consider the following CTL formulae :

1 $\mathbf{EF}\, q_\blacktriangleleft$

2 $\mathbf{AF}\, q_\blacktriangleleft$

3 $\mathbf{AGEF}\, q_\blacktriangleleft$

4 $\mathbf{EF}\,\mathbf{AG} q_\blacktriangleleft$

5 $\mathbf{E}(\neg q_2)\mathbf{U} q_\blacktriangleleft$

6 $\mathbf{A}(\neg q_2)\mathbf{U} q_\blacktriangleleft$

7 $\mathbf{E}(\neg q_4)\mathbf{U} q_\blacktriangleleft$

8 $\mathbf{A}(\neg q_4)\mathbf{U} q_\blacktriangleleft$

Which of the above formulae hold on the transition system? Explain your answer in 1-2 lines.

Solution :

1 $\mathbf{EF}\, q_\blacktriangleleft$ holds. A witness is the path $q_\triangleright, q_1, q_2, q_\blacktriangleleft$.

2 $\mathbf{AF}\, q_\blacktriangleleft$ does not hold. A counterexample is the path $q_\triangleright, q_1, q_2, q_3, q_4, q_2, q_3, \ldots$.

3 $\mathbf{AGEF}q_{\blacktriangleleft}$ holds since it is possible to reach $q_{\blacktriangleleft}$ from all states.

4 $\mathbf{EF}\,\mathbf{AG}q_{\blacktriangleleft}$ holds. A witness is the path $q_{\triangleright}, q_1, q_2, q_{\blacktriangleleft}, q_{\blacktriangleleft}, \ldots$.

5 $\mathbf{E}(\neg q_2)\mathbf{U}q_{\blacktriangleleft}$ does not hold since it is not possible to reach $q_{\blacktriangleleft}$ without passing through q_2.

6 $\mathbf{A}(\neg q_2)\mathbf{U}q_{\blacktriangleleft}$ does not hold for the same reason.

7 $\mathbf{E}(\neg q_4)\mathbf{U}q_{\blacktriangleleft}$ holds. A witness is $q_{\triangleright}, q_1, q_2, q_{\blacktriangleleft}$.

8 $\mathbf{A}(\neg q_4)\mathbf{U}q_{\blacktriangleleft}$ does not hold for the same reason as (6).