Technical University of Denmark

Written examination, May 26, 2020

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

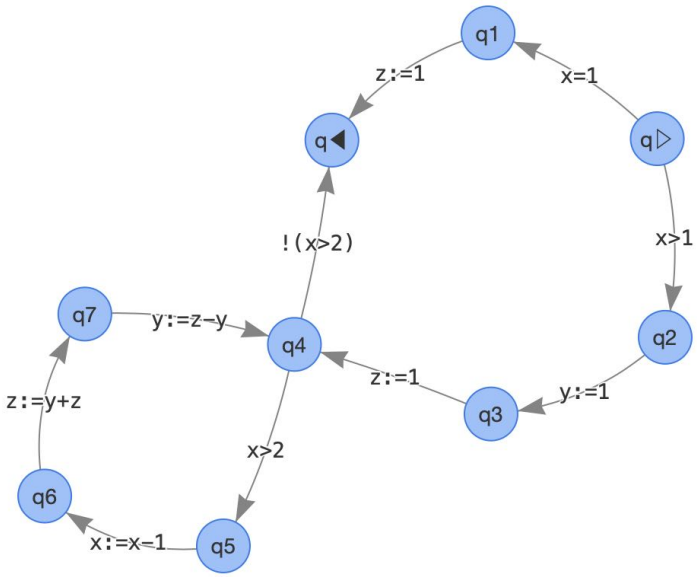
Weighting: 7-step scale

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.)*

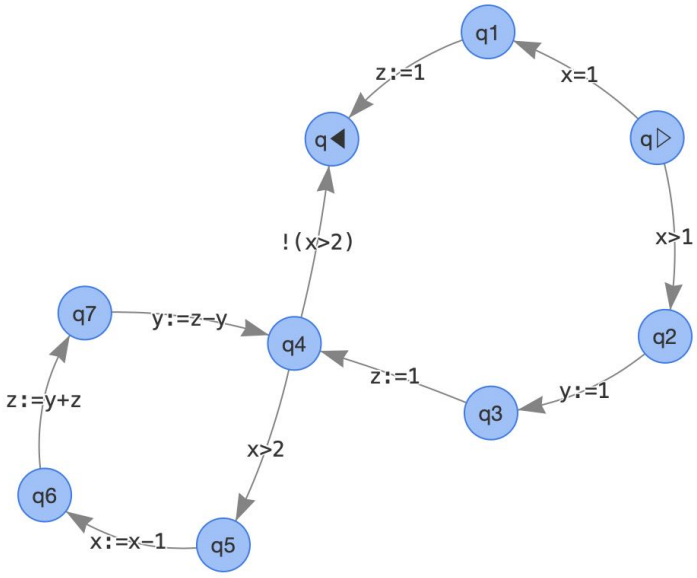
Solution : Yes, because the conditions of Proposition 1.22 are easily checked for the nodes $q_{\triangleright}$ and q_4.

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 $\mathtt{x} > 2$ because the test $\mathtt{x} > 2$ 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 $\mathtt{x} > 1$ because 1 is subtracted from $\mathtt{x}$.

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

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

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:

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

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 $\mathbf{A}(q_5) = \mathbf{A}(q_6) = \mathbf{A}(q_7) = \{[\mathtt{x} \mapsto +, \mathtt{y} \mapsto +, \mathtt{z} \mapsto +]\}$ 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 $\mathbf{A}(q_5) = \{[\mathtt{x} \mapsto +, \mathtt{y} \mapsto +, \mathtt{z} \mapsto +]\}$ and $\mathbf{A}(q_6) = \mathbf{A}(q_7) = \{[\mathtt{x} \mapsto +, \mathtt{y} \mapsto +, \mathtt{z} \mapsto +], [\mathtt{x} \mapsto 0, \mathtt{y} \mapsto +, \mathtt{z} \mapsto +], [\mathtt{x} \mapsto -, \mathtt{y} \mapsto +, \mathtt{z} \mapsto +]\}$.

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

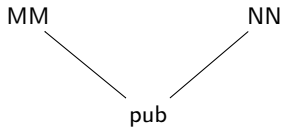
In short, the detection of signs analysis is not able to express the condition $\mathtt{z} > \mathtt{y} > 0$ any more precisely than $\mathtt{z} > 0, \mathtt{y} > 0$ and therefore the answer is no.

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 $\mathbf{edges}_{\mathbf{s}}(q_{\triangleright} \rightsquigarrow q_{\blacktriangleleft})[\![\cdots]\!](\{\})$ 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 $\mathbf{edges}_{\mathbf{s}}(q_{\triangleright} \rightsquigarrow q_{\blacktriangleleft})[\![\cdots]\!](\{\})$ 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 $\mathbf{edges}_{\mathbf{s}}(q_{\triangleright} \rightsquigarrow q_{\blacktriangleleft})[\![\cdots]\!](\{\,\})$ and observe that the actions on the edges for assignments within the loops will be

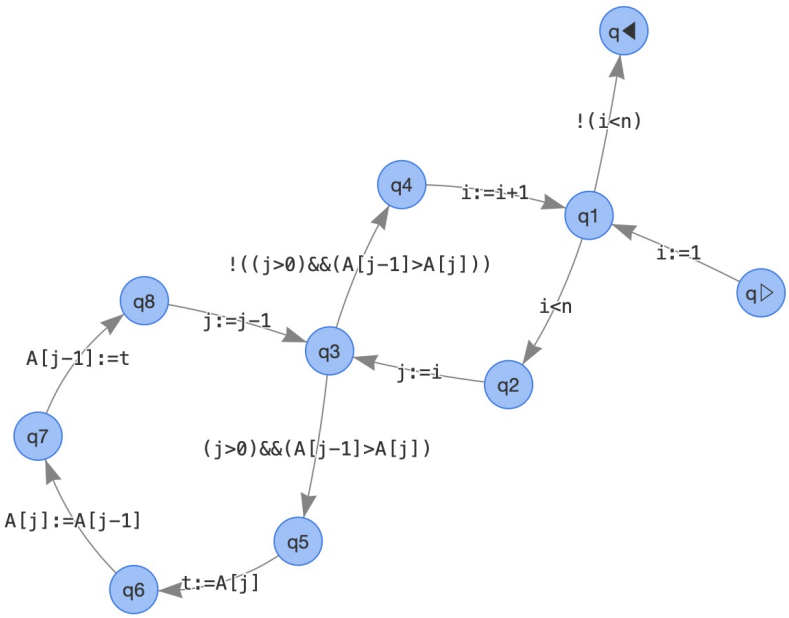
```
N[i]:=N[i]+27{i,n,j,m}
i:=i+1{i,n,j,m}
M[j]:=M[j]+33{j,m}
j:=j+1{j,m}.
```

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

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:

F	$\rightarrow$	C	(composed type)
	|	F`->`F	(function type)
C	$\rightarrow$	B	(basic type)
	|	C `*` C	(tuple type)
	|	C `list`	(list type)
	|	`[`F`]`	(brackets for grouping types)
B	$\rightarrow$	`int`	(integer type)
	|	`string`	(string type)

the set of non-terminal symbols is $N = \{F, C, B\}$, the set of terminal symbols is $T = \{$`->`, `*`, `list`, `int`, `string`, `[`, `]`$\}$ 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:

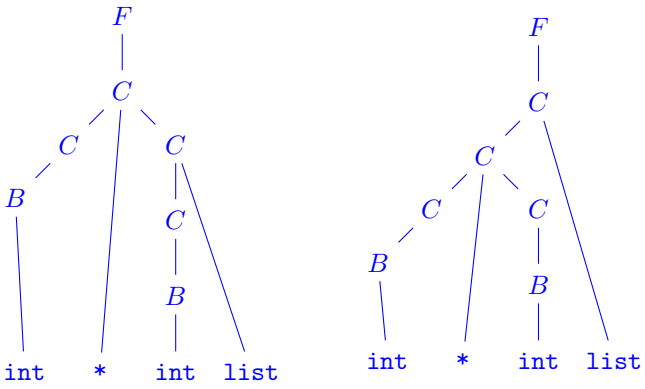
	`->`	`*`	`list`
`->`	X	no	no
`*`	yes	X	no
`list`	yes	no	X

(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:



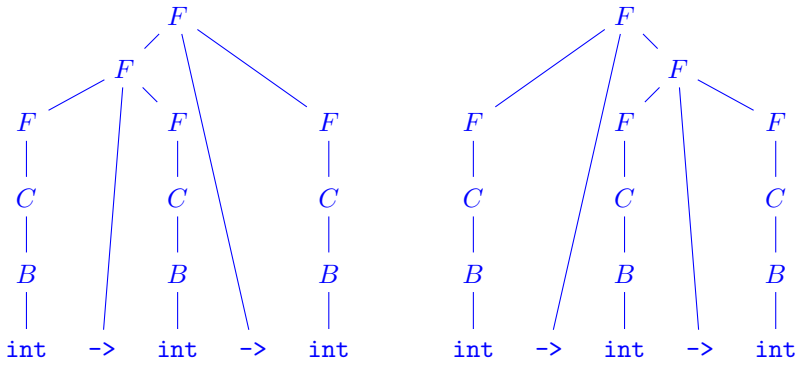
(d) In F# operator `->` is right-associative. Is it also like this in the provided grammar? If the answer is yes, provide a brief explanation. If the answer is no, explain why and provide a new grammar that enforces associativity of `->` as in F# and that accepts the same language of the original grammar.

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 `->`.

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

(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

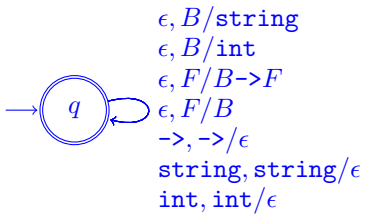
$$
\begin{array}{lcl}
F & \rightarrow & B \mid B\texttt{->}F \\
B & \rightarrow & \texttt{int} \mid \texttt{string}
\end{array}
$$

(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:

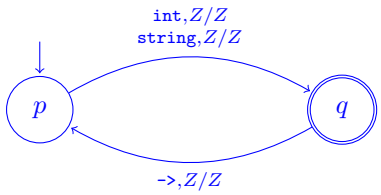
$$
\begin{array}{rcl}
\delta(q, \epsilon, F) & = & \{(q, B), (q, B\texttt{->}F)\} \\
\delta(q, \epsilon, B) & = & \{(q, \texttt{int}), (q, \texttt{string})\} \\
\delta(q, \texttt{int}, \texttt{int}) & = & \{(q, \epsilon)\} \\
\delta(q, \texttt{string}, \texttt{string}) & = & \{(q, \epsilon)\} \\
\delta(q,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

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



(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

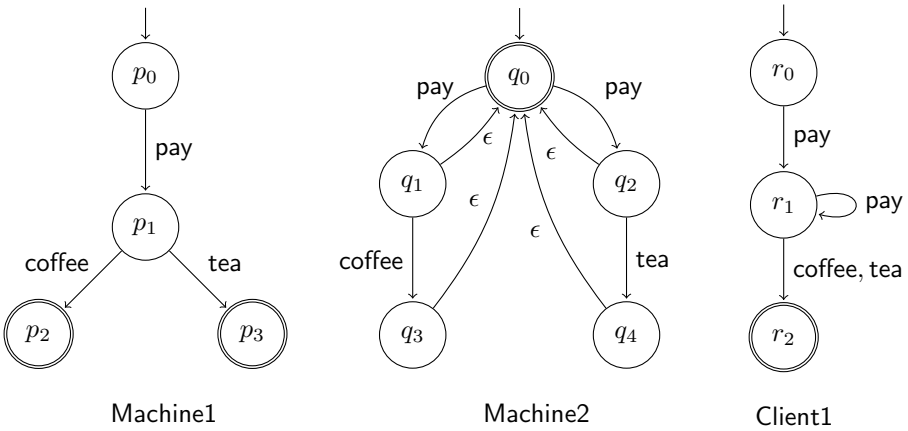


It accepts by final state. Note that the stack plays no role and remains constant (always containing the initial stack symbol Z).

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 $\mathsf{pay}^n\mathsf{coffee}$ or $\mathsf{pay}^n\mathsf{tea}$ 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 $\mathsf{pay}^n\mathsf{tea}$ 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

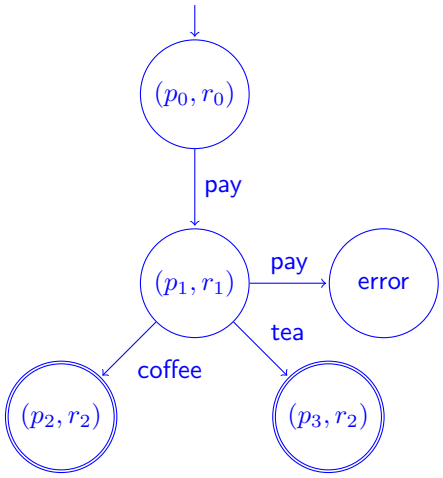
$$\delta((s_A, s_B), a) = \begin{cases} \left(\begin{array}{l} \{(s'_A, s_B) \mid s'_A \in \delta_A(s_A, a)\} \cup \\ \{(s_A, s'_B) \mid s'_B \in \delta_B(s_B, a)\} \end{array} \right) & \text{if } a = \epsilon \\ \{\mathsf{error}\} & \text{if } a \in (\Sigma \setminus \{\epsilon\}) \text{ , } \delta_A(s_A,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

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

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

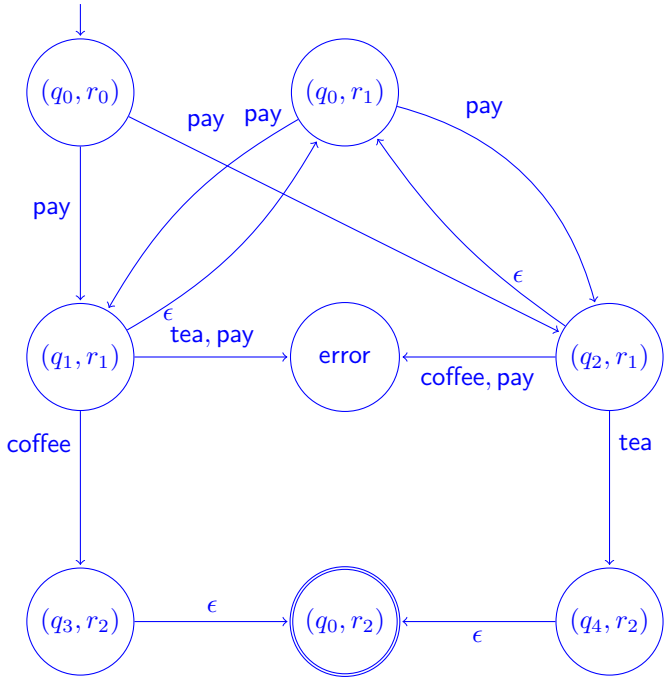
Construct the automaton Client1 $\triangleright$ Machine1 and provide it as a transition diagram. Depict only the reachable states.

Solution:



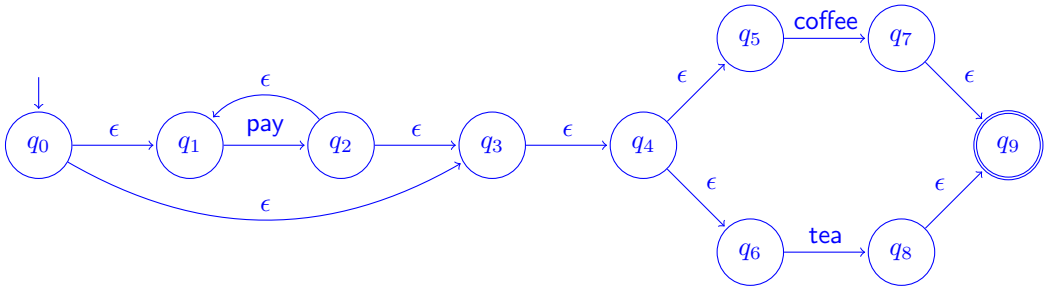
(c) Construct the automaton Client1 ▷ Machine2 and provide it as a transition diagram.

Solution:



(d) Convert 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: