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

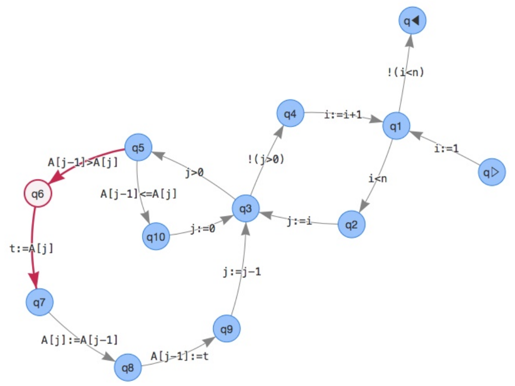
Exercises on Formal Methods

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.)

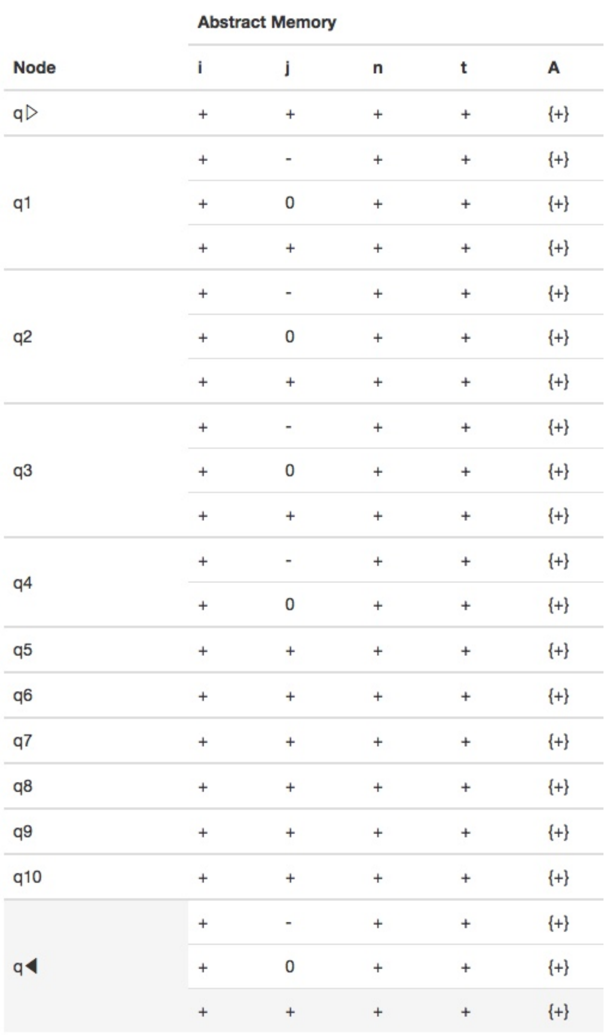
Solution: $\langle q_{\blacktriangleleft}; [\mathtt{i} = 2, \mathtt{j} = 0, \mathtt{n} = 2, \mathtt{t} = 3, \mathtt{A} = [3, 4, 2, 1]]\rangle$.

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 **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
```

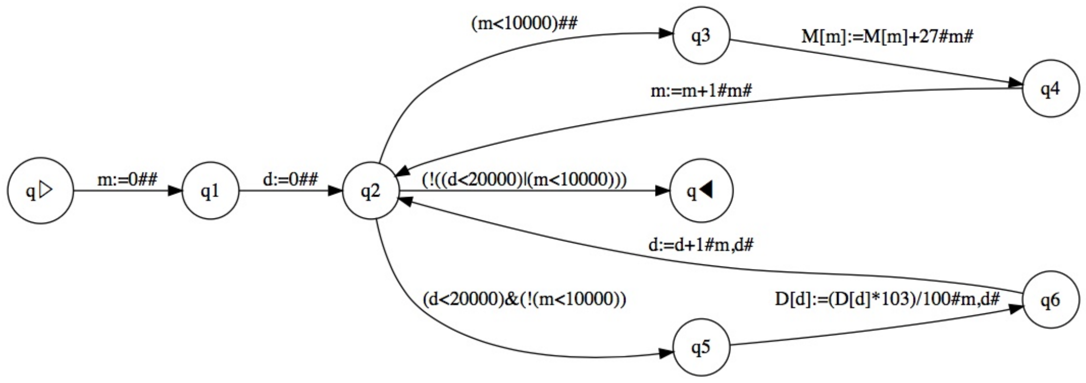
Here M is intended to be an array of length 10000 and D is intended to be an array of length 20000. (This may be seen as a cloud provider updating costs with a fixed value for one company and with a percentage for another company.)

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 $\mathsf{sec}[\![C]\!](\{\,\})$ 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`$\to$`d` and `m`$\to$`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 $\mathsf{sec}[\![C]\!](\{\,\})$ 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 $\mathsf{sec}[\![C]\!](\{\,\})$ 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 $\mathsf{sec}[\![C]\!](\{\,\})$ 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`$\to$`d` and `m`$\to$`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

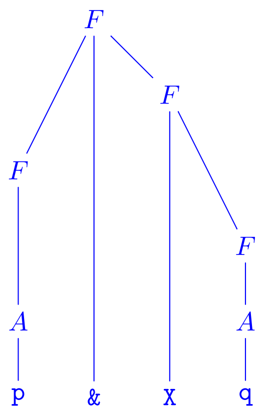
F	$\to$	`tt`	(true)
	$\mid$	A	(atomic predicates)
	$\mid$	`!`F	(negation)
	$\mid$	F `&` F	(conjunction)
	$\mid$	`X`F	(next state)
	$\mid$	`G`F	(globally)
	$\mid$	F `U` F	(until)
	$\mid$	`[`F`]`	(grouping of a formula)
A	$\to$	`p` $\mid$ `q` $\mid$ `r`	(atomic predicates)

the set of non-terminal symbols is $N = \{F, A\}$, the set of terminal symbols is $T = \{$`tt`, `!`, `&`, `X`, `G`, `U`, `[`, `]`, `p`, `q`, `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

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

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

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

(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:

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

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 $(q, \texttt{X tt U tt}, F)$, 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:

$$\begin{array}{rl}
 & (\text{q}, \texttt{X tt U tt}, \text{F}) \\
\vdash & (\text{q}, \texttt{X tt U tt}, \texttt{X}\text{F}) \\
\vdash & (\text{q}, \texttt{tt U tt}, \text{F}) \\
\vdash & (\text{q}, \texttt{tt U tt}, \text{F U F}) \\
\vdash & (\text{q}, \texttt{tt U tt}, \texttt{tt U} \text{ F}) \\
\vdash & (\text{q}, \texttt{U tt}, \texttt{U} \text{ F}) \\
\vdash & (\text{q}, \texttt{tt}, \text{F}) \\
\vdash & (\text{q}, \texttt{tt}, \texttt{tt}) \\
\vdash & (\text{q}, ,)
\end{array}$$

(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.