Technical University of Denmark

Written examination date: May 23, 2023

Course title: Computer Science Modelling

Course number: 02141

Aids allowed: All aid

Exam duration: 4 hours

Weighting: 7-step scale

NOTES:

- You can submit your answers digitally or as a paper submission.
- Digital submissions must be done as a single pdf-file.
- You must clearly indicate on each page your student number, and which task you attempt to solve.
- We recommend that you first skim all tasks before deciding in which order you want to solve them.

Exercise 1 (15%) Semantics

Consider the program graphs `PG1` - `PG3` in Figures 1 - 3 on page 3.

Question 1a: *For each program graph* `PGi` *(*$\mathtt{i} \in \{1, 2, 3\}$*) on page 3, determine whether there exists a program* C_i *in Guarded Command Language such that* $\mathbf{edges}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![C_i]\!]$ *generates it. If a suitable program exists, provide it. Otherwise, argue (in 1-2 sentences) why no such program exists.*

Question 1b: *For each of the program graphs* `PG1` *-* `PG3` *in Figures 1 - 3 on page 3, determine whether it is a deterministic system.*
Justify your answer in one sentence.

Question 1c: *Consider the program graph* `PG2` *in Figure 2 on page 3. How many different execution sequences of the form*

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

exist if the set of variables is $\mathbf{Var} = \{x, y, z\}$*?*
Justify your answer in 1-3 sentences.

Question 1d: *Prove or disprove (in 1-5 lines): every program graph that constitutes a deterministic and evolving system has at least one complete execution sequence.*

Question 1e: *Suppose we extend the Guarded Command Language by a new command for Java-style for loops*

$$\mathtt{for}(i \mathrel{\mathtt{:=}} a_1;\ b;\ i \mathrel{\mathtt{:=}} a_2)\ \{\ C\ \}$$

where i is an (integer) variable, a_1, a_2 are arithmetic expressions, b is a Boolean expression, and C is a command. Informally, the above loop

1. *sets variable i to the value of arithmetic expression a_1;*
2. *executes the loop body C if the loop guard b holds (otherwise, the loop terminates);*
3. *updates i to the value of a_2 after termination of loop body C; and*
4. *continues with step (2).*

Formalise the above informal semantics by defining the program graph of for-loops. That is, give a formal definition of

$$\mathbf{edges}(q_\triangleright \rightsquigarrow q_\blacktriangleleft)[\![\mathtt{for}(i \mathrel{\mathtt{:=}} a_1;\ b;\ i \mathrel{\mathtt{:=}} a_2)\ \{\ C\ \}]\!].$$

Briefly explain your definition in 1-3 sentences.

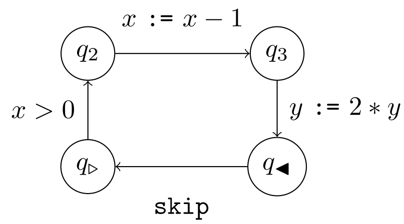


Figure 1: Program graph PG1

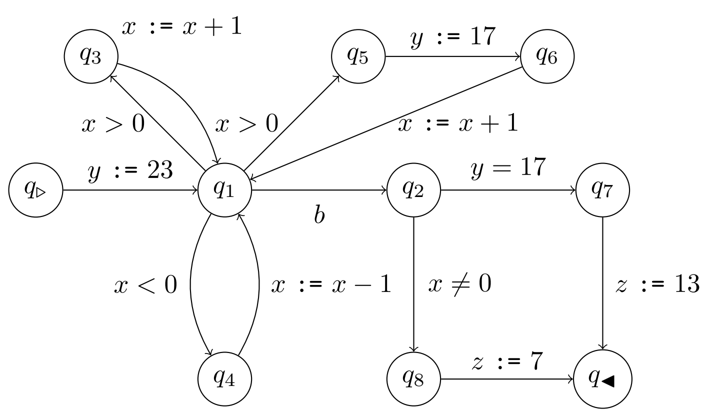


Figure 2: Program graph PG2 with $b = \neg(x > 0) \land \neg(x < 0) \land \neg(x > 0)$.

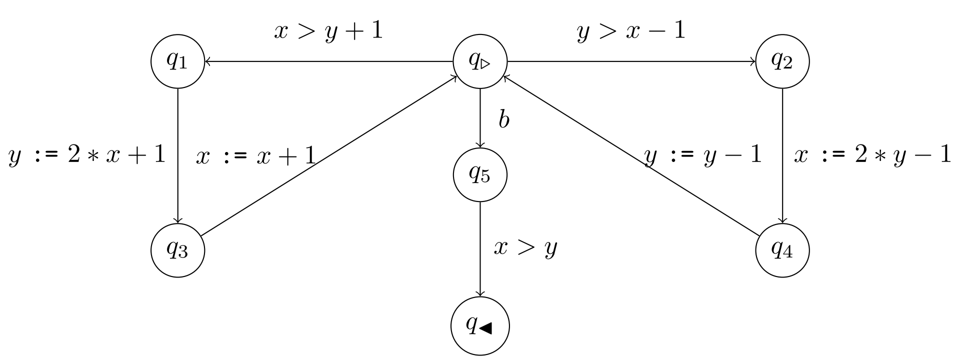
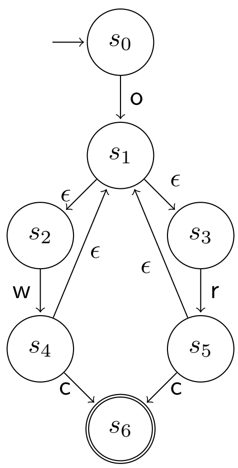


Figure 3: Program graph PG3 with $b = \neg(x > y + 1) \land \neg(y > x - 1)$.

Exercise 2 (20%) Formal Languages

Alice and Bob want to formalise correct usages of an API with endpoints o (open), w (write), r (read), and c (close).

Alice proposes automaton A:



Bob proposes regular expression B:

$$\mathsf{o}\mathsf{r}^*\mathsf{w}^*\mathsf{c}$$

where the set $\{\mathsf{o}, \mathsf{r}, \mathsf{w}, \mathsf{c}\}$ is the input alphabet for A and B.

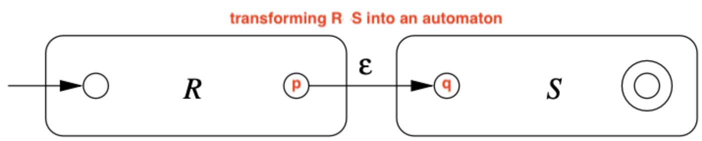
(a) Fill the table below. In the empty cell corresponding to language L provide a short as possible word that belongs to L. Write "———" if no such word exists.

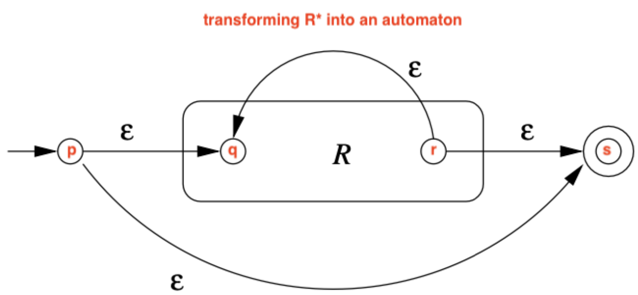
Language	Shortest word
$\mathcal{L}(\mathsf{A}) \cup \mathcal{L}(\mathsf{B})$	
$\mathcal{L}(\mathsf{A}) \cap \mathcal{L}(\mathsf{B})$	
$\mathcal{L}(\mathsf{A}) \setminus \mathcal{L}(\mathsf{B})$	
$\mathcal{L}(\mathsf{B}) \setminus \mathcal{L}(\mathsf{A})$	
$\overline{\mathcal{L}(\mathsf{A})} \cap \overline{\mathcal{L}(\mathsf{B})}$	
$\mathcal{L}(\mathsf{A})^* \cup \mathcal{L}(\mathsf{B})^*$	

(b) Alice and Bob decide to transform the regular expression B into a finite state automaton to easily compare their solutions. Alice correctly applies the transformations seen in class to obtain a minimized DFA A_B.

Provide $\mathsf{A_B}$ as transition diagram. Do *not* include the so-called "dead"/"error" state and its transitions. Explain at a high-level which transformations you have applied (1 sentence per transformation). You do not need to provide the details of each transformation.

(c) Bob does not like to have too many ϵ transitions and decides to modify the algorithm for transforming regular expressions into finite automata by changing the rules below:





The modifications of Bob are:

- In the transformation of RS, states p and q are merged.
- In the transformation of R^*, p is merged with q, and r is merged with s.

After applying the new algorithm, Bob obtains the automaton $\mathsf{B_B}$. Provide $\mathsf{B_B}$ as minimized DFA (again as a transition diagram, with no "error"/"dead" state). You do not need to explain how you obtained it.

(d) Answer the following questions with a short sentence (as a rule of thumb, your answer should fit in the table). The sentence must start with "YES" or "NO".

Question	Short answer
$\mathcal{L}(\mathsf{A}_{\mathsf{B}}) = \mathcal{L}(\mathsf{B}_{\mathsf{B}})$?	
$\mathcal{L}(\mathsf{A}_{\mathsf{B}}) = \mathcal{L}(\mathsf{B})$?	
$\mathcal{L}(\mathsf{B}_{\mathsf{B}}) = \mathcal{L}(\mathsf{B})$?	
$\mathcal{L}(\mathsf{A}_{\mathsf{B}}) = \mathcal{L}(\mathsf{A})$?	
$\mathcal{L}(\mathsf{B}_{\mathsf{B}}) = \mathcal{L}(\mathsf{A})$?	

Exercise 3 (20%) Program Verification

Question 3a: *Complete the following annotated GCL program to a formal program proof (that is, all annotations must result from applying the proof rules of Floyd-Hoare logic introduced in the course).*

```
{ n ≥ 0 }
r := 12 * n;
k := 0;
do { k ≤ 6 * n ∧ r = 12 * n − 2 * k }
        k < 6 * n ->
            r := r − 2;
            k := k + 1
    []
        k < 6 * n ->
            k := k + 3;
            k := k − 3
od
{ r = 0 }
```

Question 3b: *Show that the GCL program below computes n^3 by constructing a formal program proof for the given contract (that is, all annotations must result from applying the proof rules of Floyd-Hoare logic introduced in the course).*

```
{ true }
p := n * n;
r := 0;
do
    p > 0 ->
        r := r + n;
        p := p − 1
od
{ r = n^3 }
```

Question 3c: *The course material does not consider a proof rule for verifying assignments of the form $A[x]$:= a, where A is an array, x is a variable and a is an arithmetic expression. Suppose we treat an assignment to an array element like any other assignment, that is, we construct the triple*

$$\{\ P\ \} A[x]\ \texttt{:=}\ a \{\ \exists \underline{y} \colon P[\underline{y} \,/\, A[x]] \ \land\ A[x] = a[\underline{y} \,/\, A[x]]\ \}\ ,$$

where $P[\underline{y} \,/\, A[x]]$ and $a[\underline{y} \,/\, A[x]]$ denote the substitution (i.e. syntactic replacement) of every occurrence of $A[x]$ by $\underline{y}$ in P and a, respectively.

*Show that the above rule is **not** correct. To this end, find a GCL program C, predicates P and Q, and memories σ and σ' such that*

- *one can construct a program proof $\{\ P\ \}\ C\ \{\ Q\ \}$ using our existing rules and the above rule for assigning to array elements, and*
- *there is a complete execution sequence of C with initial memory σ and final memory σ' such that $\sigma \models P$ and $\sigma' \not\models Q$.*

Give both the program proof and the complete execution sequence for your example to demonstrate that you have indeed chosen adequate C, P, Q, σ, σ'.

Hint: Notice that $A[i]$ and $A[j]$ are syntactically different array elements but refer to the same array element if variables i and j store the same value.

Exercise 4 (10%) Program Analysis

Question 4a: *Explain whether the following statements about the program graphs* `PG4` *and* `PG5` *in Figures 5 - 7 (on page 10) are* `true`, `false`, *or* `unknown` *(that is, no conclusive answer is possible). Your answers must be* based on the results of the detection of signs analysis *in [FM, chapter 4]. You can safely assume that formalmethods.dk implements this analysis. Explain your answer to each question in 1-3 sentences.*

(a) `PG4` has no complete execution sequences.

(b) For every execution sequence of `PG4`, the sign of `y` will eventually become equal to the sign of `z`.

(c) For every complete execution sequence of `PG5`, no element of array A is 0 whenever x is positive upon termination.

(d) There exist executions of `PG5` that can get stuck.

Question 4b: *Consider the abstract memory* $\hat{\sigma}[x \mapsto +][y \mapsto +]$ *and the following guarded command:*

$$\texttt{if } x > y \texttt{ -> } x \texttt{ := } x - y \texttt{ [] } x \leq y \texttt{ -> } x \texttt{ := } 17 \texttt{ fi}$$

What are the possible signs of x *upon termination according to a detection of signs analysis (as defined in [FM, chapter 4]) for the above initial abstract memory? For each of those signs, explain in 1-3 sentences whether it is realistic (that is if there exists a concrete execution sequence in which we can observe variables whose values have those signs).*

```
do x<0 -> y:=(-1*z)*z
[] x=0 -> y:=0
[] x>0 -> y:=z*z
od
```

Figure 4: Code for PG4

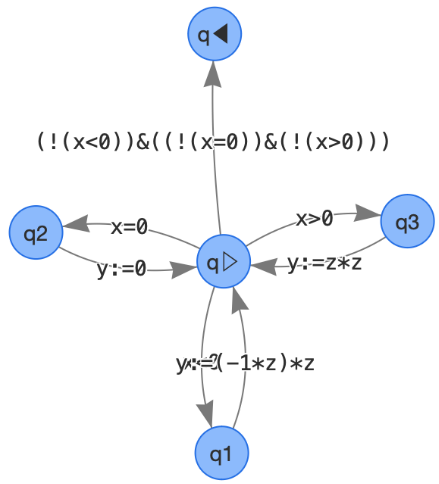


Figure 5: Program graph PG4

```
if
   x >= 0 -> A[x] := A[x+1]
[] x > 1 -> A[x+1] := A[x]-1
fi;
if
   A[x] > 0 -> x := 0
[] A[x] < 0 -> x := 1
fi
```

Figure 6: Code for PG5

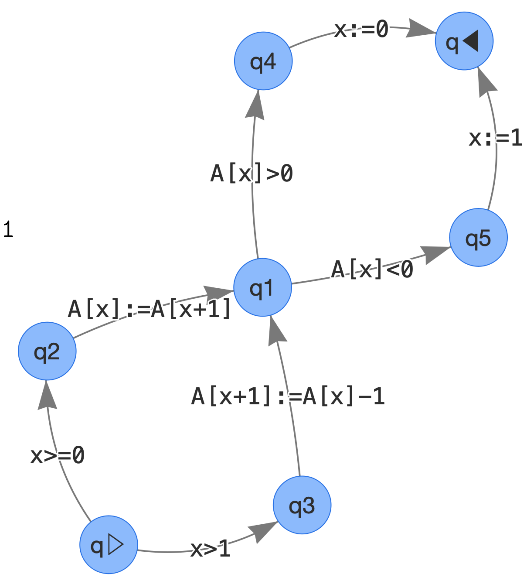


Figure 7: Program graph PG5

Exercise 5 (10%) Language-based Security

In this exercise we focus on the classical confidentiality policy with public and private variables. Let `h` be a private variable and `l` be a public variable. Alice does not like that the program

```
l := h ; l := 0
```

is considered as *not secure* with the security analysis seen in the course, while it is considered as *non-interferent*. To remedy this, she proposes a new security analysis based on the following (underlined) modification to the function actualFlows (see essential exercises in the lecture on language-based security):

$$\begin{aligned}&\mathsf{actualFlows}(c_1;c_2, X)\\&\quad = (\mathsf{actualFlows}(c_1, X)\underline{\setminus\{\{y\}\rightrightarrows\mathsf{updates}(c_2) \mid y \in V\}}) \cup \mathsf{actualFlows}(c_2, X)\end{aligned}$$

where V is the set of all variables, and function updates is intended to determine the set of variables that are *necessarily* updated in a program (i.e. updated in all possible executions):

$$\begin{array}{rcl}\mathsf{updates}(\texttt{skip}) & = & \emptyset\\ \mathsf{updates}(x{:=}e) & = & \{x\}\\ \mathsf{updates}(A[e_1]{:=}e_2) & = & \{A\}\\ \mathsf{updates}(c_1;c_2) & = & \mathsf{updates}(c_1) \cup \mathsf{updates}(c_2)\\ \mathsf{updates}(\texttt{if}\ gc\ \texttt{fi}) & = & \mathsf{updates}(gc)\\ \mathsf{updates}(\texttt{do}\ gc\ \texttt{od}) & = & \emptyset\\ \mathsf{updates}(b\ \texttt{->}\ c) & = & \mathsf{updates}(c)\\ \mathsf{updates}(gc_1[]gc_2) & = & \mathsf{updates}(gc_1) \cap \mathsf{updates}(gc_2)\end{array}$$

Question 5a: *Would the new analysis determine that the above program* `l := h ; l := 0` *is secure? Write "YES" or "NO" and explain your answer in 1 sentence.*

Question 5b: *Is there a program that has the non-interference property but that is deemed as not secure by the new analysis function? If the answer is "YES" provide the smallest example you can find and explain in 2-3 lines why it is secure but it does not have the non-interference property. If the answer is "NO", provide your argument in 2-3 lines.*

Question 5c: *Bob implements the new security analysis of Alice but does a mistake when computing the* updates *function: He writes* $\cup$ *instead of* $\cap$ *in this case of the function:*

$$\mathsf{updates}(gc_1 [] gc_2) \quad = \quad \mathsf{updates}(gc_1) \;\cup\; \mathsf{updates}(gc_2)$$

Provide an example of a program that exploits Bob's bug to deem the program as secure but such that the program does not have the non-interference property. Explain your example in 2-3 lines.

Question 5d: *Now, forget about Bob's bug. Are we guaranteed that if a program is deemed as secure with the new definition of Alice, it will be non-interferent? If the answer is "NO" provide the smallest example you can find and explain in 2-3 lines why it is secure but it does not have the non-interference property. If the answer is "YES", provide your argument in 2-3 lines.*

Question 5e: *Are the following statements correct or incorrect? Provide your argument in 1 sentence.*

(a) *According to the lecture's security analysis, if* $c_1\,\texttt{;}c_2$ *is secure then* $c_2\,\texttt{;}c_1$ *is secure.*

(b) *According to Alice's new security analysis, if* $c_1\,\texttt{;}c_2$ *is secure then* $c_2\,\texttt{;}c_1$ *is secure.*

(c) *If* $c_1\,\texttt{;}c_2$ *is non-interferent then* $c_2\,\texttt{;}c_1$ *is non-interferent.*

Exercise 6 (15%) Context-free Languages

The following context-free grammar provides a syntax for positive modal logic formulas:

$$\begin{array}{rcll} F & \rightarrow & P & \text{(proposition...)} \\ & \mid & F \wedge F & \text{(... and ...)} \\ & \mid & \Diamond F & \text{(possibly...)} \\ & \mid & \Box F & \text{(necessarily ...)} \\ P & \rightarrow & \mathsf{a} \mid \mathsf{b} \mid \mathsf{c} & \text{(some atomic propositions)} \end{array}$$

where the set of non-terminal symbols is $N = \{F, P\}$, the set of terminal symbols is $T = \{\Box, \Diamond, \mathsf{a}, \mathsf{b}, \mathsf{c},\}$ and the initial symbol is F. We call a string accepted by the grammar a "formula".

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

$$\mathsf{a} \wedge \Diamond\Box\mathsf{b}$$

would be represented as a value of your datatype.

(b) Consider the following operators for composing models: $\_\wedge\_$, $\Diamond\_$, and $\Box\_$. Fill the table below as follows: in the cell corresponding to row (horizontal) x and column (vertical) y write YES if, according to the above grammar (without any additional operator precedence), the operators x and y can yield an ambiguity. Otherwise, write "———".

	$\_\wedge\_$	$\Diamond\_$	$\Box\_$
$\_\wedge\_$			
$\Diamond\_$			
$\Box\_$			

(c) If you have discovered at least one ambiguity in (b) that involves at least one modal operator ($\Box$ or $\Diamond$), showcase it by providing an example of a formula that has at least two distinct parse trees. Otherwise, explain in 2-3 lines why no formula exists that can have two parse trees.

(d) If you have discovered at least one ambiguity in (b) that involves at least one modal operator ($\Box$ or $\Diamond$) 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.

(e) Consider the following PDA which accepts by final state and is defined by the tuple $(\{q0, q_1, q_2\}, T, T \cup N, \delta, F, \{q_2\})$ where the transition function δ is defined as follows:

$$\begin{array}{rcll} \delta(q_0, u, F) & = & \{(q_0, F)\} & \text{if } u \in \{\Diamond, \Box\} \\ \delta(q_0, v, F) & = & \{(q_1, P)\} & \text{if } v \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \\ \delta(q_1, \epsilon, P) & = & \{(q_2, F)\} & \\ \delta(q_2, \wedge, F) & = & \{(q_0, F)\} & \end{array}$$

The PDA accepts the same language as the grammar. Check this by showing how the PDA can accept the string $\mathsf{a} \wedge \Diamond\Box\mathsf{b}$.

(f) Can we build a DFA that recognizes the set of formulas accepted by the grammar described at the beginning of the exercise? Provide a short answer in 1-2 sentences. If your answer is positive, provide the DFA.

Exercise 7 (5%) Model Checking

Fill the following table according to the rules explained below:

ϕ	$TS1 \models \phi$	$TS2 \not\models \phi$
$\mathsf{AF}p$		
$\mathsf{AG}p$		
A(pUq)		
AG(p $\rightarrow$ AF q)		
$\neg$p $\wedge$ AFp		

In column $TS1 \models \phi$ draw the smallest transition system $TS1$ that you can find that satisfies the formula ϕ. In column $TS2 \not\models \phi$ provide the mimimal extension you can do to TS1 (i.e. TS2 has the same states, transitions, initial states, labels, etc. but possibly more) that does *not* satisfy the formula. If you cannot find a transition system, write "——".

Here, "smallest" refers to the size of a transition system, counted as the number of states and transitions. As in the book, the transition systems cannot have stuck states.

Exercise 8 (5%) Induction Proofs

Consider the function $\mathsf{dup}\colon \Sigma^* \to \Sigma^*$, given by the following recursive definition:

$$\mathsf{dup}(w) \;=\; \begin{cases} \varepsilon, & \text{if } w = \varepsilon \\ \mathsf{dup}(w')\ a\ a, & \text{if } w = w'a, a \in \Sigma, w' \in \Sigma^* \end{cases}$$

Furthermore, let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We transform A into another DFA A_2 as follows:

- $A_2 = (Q_2, \Sigma, \delta_2, q_0, F)$,
- $Q_2 = \{q_e\} \cup Q \cup \bigcup_{q \in Q}\{q_a \mid a \in \Sigma\}$ (where the error state q_e is fresh and the state q_a is fresh for every $q \in Q$ and $a \in \Sigma$),
- $\delta_2(q, a) = \begin{cases} q_a & \text{if } q \in Q \\ \delta(q', a) & \text{if } q \text{ is of the form } q'_a \text{ for some } q' \in Q \text{ and } a \in A \\ q_e & \text{otherwise} \end{cases}$

Complete the proof below to show that A_2 accepts $\mathsf{dup}(w)$ if and only if A accepts w by filling the blank lines.

We first show by induction on the structure of words that, for every word $w \in \Sigma^*$ and all states $q \in Q$ (but not necessarily the states in $Q_2 \setminus Q$), we have

$$\delta^*(q, w) = \delta_2^*(q, \mathsf{dup}(w)).$$

Induction base: For $w = \varepsilon$, we have **. . . (1-2 lines suffice)**

Induction hypothesis: For every word $w \in \Sigma^*$ and all states $q \in Q$,

$$\delta^*(q, w) = \delta_2^*(q, \mathsf{dup}(w)).$$

Induction step: Let $w = w'a$, where $a \in \Sigma$ and $w' \in \Sigma^*$,
... (5-6 lines suffice)

Hence, $\delta^*(q, w) = \delta_2^*(q, \mathsf{dup}(w))$, which finishes the proof by induction.

Now, using the above property for $q = q_0$, we conclude (for all words $w \in \Sigma^*$) that $\delta^*(q_0, w) \in F$ if and only if $\delta^*(q_0, \mathsf{dup}(w)) \in F$. Thus, A_2 accepts $\mathsf{dup}(w)$ if and only if A accepts w.