

Answers to  
Theory of Computation

Jim Hefferon  
<https://hefferon.net/computation>



This is a draft (date: February 5, 2022). I'd be glad for bug reports. Send to the contact information given at [hefferon.net](http://hefferon.net).

*Cover:* Flauros, the sixty-fourth demon and the Duke of Hell, who gives true answers of all things past, present, and future. Although, you should verify what he says for yourself. (Unless you can get him inside a triangle, and then you're good.) From *Dictionnaire Infernal*, Jacques Albin Simon Collin de Plancy, 1863.

## Chapter I: Mechanical Computation

**I.1.10** It is like a program in that it is single-purpose, that is, just as we may have a program that doubles its input and does nothing else, so also we may have such a Turing machine. It is unlike a program in that it is hardware.

It is like the computers we have on our desks today in that it mechanically converts the input into the output. A Turing machine is unlike such a computer in that it is purpose-built, for a single job.

**I.1.11** The definition describes the machine. The description of the machine's action, involving the definitions of configurations and the transitions, follows Definition 1.4.

Restated, as A Bauer says in a comment to the question, Why is the road not part of the car?

**I.1.12** It is not a question of determining whether a particular mathematical statement is true. Obviously, if  $x = 2$  then we can determine whether  $x > 3$ . Similarly, we can determine the truth or falsity of  $\forall x \in \mathbb{N} [x^2 > 3]$ . Rather, the Entscheidungsproblem asks for an algorithm that inputs any mathematical statement at all, and outputs 'T' or 'F'.

**I.1.13**

(A) Here is the sequence of tapes traced out in the course of the computation of the predecessor of 2.

Step	Configuration	Step	Configuration	Step	Configuration
0	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1 1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	3	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1 1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>1</sub></div>	6	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>
1	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1 1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	4	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>1</sub></div>	7	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>3</sub></div>
2	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1 1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	5	<div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">1</div> <div style="border: 1px solid black; width: 40px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>		

(B) This is the tape diagram sequence for the sum of 2 and 2.

Step	Configuration	Step	Configuration	Step	Configuration
0	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	5	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>	10	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>3</sub></div>
1	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	6	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>	11	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>4</sub></div>
2	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>0</sub></div>	7	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>	12	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>5</sub></div>
3	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>1</sub></div>	8	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>2</sub></div>		
4	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>1</sub></div>	9	<div style="border: 1px solid black; width: 100px; margin: 0 auto; text-align: center;">1 1 1 1 1</div> <div style="border: 1px solid black; width: 60px; margin: 0 auto; text-align: center;">q<sub>3</sub></div>		

(C) This is the predecessor computation.

$$\begin{aligned}
 &\langle q_0, 1, \varepsilon, 1 \rangle \vdash \langle q_0, 1, 1, \varepsilon \rangle \vdash \langle q_0, B, 11, \varepsilon \rangle \vdash \langle q_1, 1, 1, \varepsilon \rangle \vdash \langle q_1, B, 1, \varepsilon \rangle \vdash \langle q_2, 1, \varepsilon, \varepsilon \rangle \\
 &\vdash \langle q_2, B, \varepsilon, 1 \rangle \vdash \langle q_3, 1, \varepsilon, \varepsilon \rangle
 \end{aligned}$$

There is no instruction for state  $q_3$  so  $\langle q_3, 1, \varepsilon, \varepsilon \rangle$  is a halting configuration.

The sum of 2 and 2 is a little longer.

$$\begin{aligned} \langle q_0, 1, \varepsilon, 1B11 \rangle &\vdash \langle q_0, 1, 1, B11 \rangle \vdash \langle q_0, B, 11, 11 \rangle \vdash \langle q_1, B, 11, 11 \rangle \vdash \langle q_1, 1, 11, 11 \rangle \\ &\vdash \langle q_2, 1, 11, 11 \rangle \vdash \langle q_2, 1, 1, 111 \rangle \vdash \langle q_2, 1, \varepsilon, 1111 \rangle \vdash \langle q_2, B, \varepsilon, 11111 \rangle \\ &\vdash \langle q_3, B, \varepsilon, 11111 \rangle \vdash \langle q_3, 1, \varepsilon, 1111 \rangle \vdash \langle q_4, B, \varepsilon, 1111 \rangle \vdash \langle q_5, 1, \varepsilon, 111 \rangle \end{aligned}$$

**I.1.14**

- (A) To show that some operation on negative numbers (or any mathematical object, for that matter) can be computed, name a representation for them with strings over a finite alphabet and then show how to use a Turing machine to do mechanical computations on those strings. For instance, to show that multiplication of two integers is mechanical, we could use the alphabet  $\Sigma = \{B, 0, \dots, 9, +, -\}$  and represent each integer as a pair  $\langle \text{sign}, \text{natural number} \rangle$ . Making a Turing machine that has four cases, for the four possible pairs of signs, is tedious but perfectly possible.
- (B) If by “problem” we mean that the machine fails to halt, this is not the source of the problem. Rather, this machine is a straightforward infinite loop.
- (C) We could give a machine states that are not sequential. The only limitation is that we use the convention that the initial state is  $q_0$ . For instance, this machine fits the definition  $\mathcal{P} = \{q_0Bq_{50}, q_01q_{50}, q_{50}1q_1, q_{50}1q_1\}$ , and on any input reaches state  $q_{50}$  after one step.

**I.1.15** The halting state is  $q_4$  and the working states are  $q_0, q_1, q_2$  and  $q_3$ .

**I.1.16** The trace is not too interesting.

Step	Configuration	Step	Configuration	Step	Configuration
0		4		8	
1		5		9	
2		6		10	
3		7			

**I.1.17**

- (A) The result of running the mystery machine on input 11 is this.

Step	Configuration	Step	Configuration
0		2	
1		3	

- (B) On input 1011 it gives this.

Step	Configuration	Step	Configuration
0		2	
1		3	

- (C) The input 110 gives this.

Step	Configuration	Step	Configuration
0	$\begin{array}{c} \boxed{110} \\ q_0 \end{array}$	3	$\begin{array}{c} \boxed{110} \\ q_0 \end{array}$
1	$\begin{array}{c} \boxed{110} \\ q_1 \end{array}$	4	$\begin{array}{c} \boxed{110} \\ q_4 \end{array}$
2	$\begin{array}{c} \boxed{110} \\ q_0 \end{array}$		

(D) With the input 1101 the machine gives this sequence of tapes.

Step	Configuration	Step	Configuration
0	$\begin{array}{c} \boxed{1101} \\ q_0 \end{array}$	4	$\begin{array}{c} \boxed{1101} \\ q_1 \end{array}$
1	$\begin{array}{c} \boxed{1101} \\ q_1 \end{array}$	5	$\begin{array}{c} \boxed{1101} \\ q_4 \end{array}$
2	$\begin{array}{c} \boxed{1101} \\ q_0 \end{array}$		
3	$\begin{array}{c} \boxed{1101} \\ q_0 \end{array}$		

(E) This is what happens when the initial tape is empty.

Step	Configuration
0	$\begin{array}{c} \boxed{\phantom{1101}} \\ q_0 \end{array}$
1	$\begin{array}{c} \boxed{\phantom{1101}} \\ q_4 \end{array}$

I.1.18 This is the transition table.

$\Delta$	B	$\emptyset$	1
$q_0$	$Bq_4$	$Rq_0$	$Rq_1$
$q_1$	$Bq_4$	$Rq_2$	$Rq_0$
$q_2$	$Bq_4$	$Rq_0$	$Rq_3$
$q_3$	-	-	-

I.1.19 The machine  $\mathcal{P}_{\text{blankones}} = \{q_0BBq_2, q_01Bq_1, q_1BRq_0, q_111q_0\}$  does this on a tape with two 1's.

Step	Configuration	Step	Configuration
0	$\begin{array}{c} \boxed{11} \\ q_0 \end{array}$	3	$\begin{array}{c} \boxed{\phantom{11}} \\ q_1 \end{array}$
1	$\begin{array}{c} \boxed{1} \\ q_1 \end{array}$	4	$\begin{array}{c} \boxed{\phantom{11}} \\ q_0 \end{array}$
2	$\begin{array}{c} \boxed{1} \\ q_0 \end{array}$	5	$\begin{array}{c} \boxed{\phantom{11}} \\ q_2 \end{array}$

I.1.20

(A) This machine does the job; note that the head never moves.

$$\mathcal{P}_{\text{singlebitnot}} = \{q_0BBq_1, q_0\emptyset 1q_1, q_01\emptyset q_1\}$$

This shows input  $\sigma = \emptyset$ .

Step	Configuration
0	$\emptyset$ $q_0$
1	1 $q_1$

(B) This machine does ‘and’. Note that the states are not numbered sequentially; sometimes making a gap is convenient when you are writing the machine. For instance, here state  $q_{100}$  is used as the halting state, since it was clear there would not be a hundred states and so this one would be available. Similarly,  $q_{10}$  is for what the machine does if the first bit is 0 while  $q_{20}$  is for the case where the first bit is 1.

$$\mathcal{P}_{\text{singlebitand}} = \{q_0BBq_{100}, q_0\theta Bq_{10}, q_01Bq_{20}, q_{10}BRq_{11}, q_{10}\theta\theta q_{100}, q_{10}11q_{100}, q_{11}BBq_{100}, q_{11}\theta\theta q_{100}, q_{11}1\theta q_{100}, q_{20}BRq_{21}, q_{20}\theta\theta q_{100}, q_{20}1\theta q_{100}, q_{21}BBq_{100}, q_{21}\theta\theta q_{100}, q_{21}11q_{100}\}$$

This shows what the machine does on input  $\sigma = \theta 1$ .

Step	Configuration	Step	Configuration
0	$\emptyset 1$ $q_0$	2	1 $q_{11}$
1	1 $q_{10}$	3	$\emptyset$ $q_{100}$

(c) This is the ‘or’ machine. As in the prior item, the states are not numbered sequentially. Note also that this machine is basically the same as the ‘or’ machine.

$$\mathcal{P}_{\text{singlebitor}} = \{q_0BBq_{100}, q_0\theta Bq_{10}, q_01Bq_{20}, q_{10}BRq_{11}, q_{10}\theta\theta q_{100}, q_{10}11q_{100}, q_{11}BBq_{100}, q_{11}\theta\theta q_{100}, q_{11}11q_{100}, q_{20}BRq_{21}, q_{20}\theta\theta q_{100}, q_{20}1\theta q_{100}, q_{21}BBq_{100}, q_{21}\theta 1q_{100}, q_{21}11q_{100}\}$$

Here is what it does on input  $\sigma = \theta 1$ .

Step	Configuration	Step	Configuration
0	$\emptyset 1$ $q_0$	2	1 $q_{11}$
1	1 $q_{10}$	3	1 $q_{100}$

**I.1.21** This machine uses  $q_0$  to slide to the right end, appends the  $\theta 1$ , and uses  $q_2$  to slide back to the left.

$$\mathcal{P}_{\text{append}\theta 1} = \{q_0B\theta q_1, q_0\theta Rq_0, q_01Rq_0, q_1B1q_2, q_1\theta Rq_1, q_11Lq_2, q_2BRq_3, q_2\theta Lq_2, q_21Lq_2\}$$




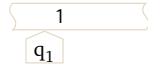
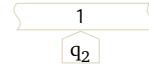

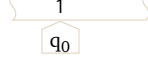
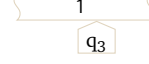
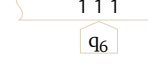

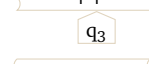
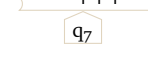


This example shows what the machine does, on input  $\sigma = 11$ .

Step	Configuration	Step	Configuration	Step	Configuration
0	1 1 $q_0$	4	1 1 $\theta$ $q_1$	8	1 1 $\theta$ 1 $q_2$
1	1 1 $q_0$	5	1 1 $\theta$ 1 $q_2$	9	1 1 $\theta$ 1 $q_2$
2	1 1 $q_0$	6	1 1 $\theta$ 1 $q_2$	10	1 1 $\theta$ 1 $q_3$
3	1 1 $\theta$ $q_1$	7	1 1 $\theta$ 1 $q_2$		

**I.1.22** This machine works.

$$\mathcal{P}_{\text{constantthree}} = \{q_0BBq_2, q_01Bq_1, q_1BRq_0, q_111q_0, q_2B1q_2, q_21Rq_3, q_3B1q_3, q_31Rq_4, q_4B1q_4, q_411q_5, q_5BLq_6, q_51Lq_6, q_6BBq_7, q_61Lq_7\}$$

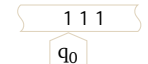
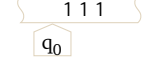
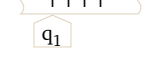
Here is the machine acting on an input of 2.

<i>Step</i>	<i>Configuration</i>	<i>Step</i>	<i>Configuration</i>	<i>Step</i>	<i>Configuration</i>
0		5		10	
1		6		11	
2		7		12	
3		8		13	
4		9			

**I.1.23** This machine computes successor.

$$\mathcal{P}_{\text{successor}} = \{q_0B1q_1, q_01Lq_0\}$$

An example is this tape sequence for input 3.

<i>Step</i>	<i>Configuration</i>
0	
1	
2	

**I.1.24**

- (A) We begin with the head at the start of a sequence of 1's. Erase that first 1, then slide to the end of the sequence, and past a blank. Put two 1's, and then slide back to the start of the initial sequence. Iterate that. The machine has nine instructions and alphabet  $\Sigma = \{1, B\}$ .

$$\mathcal{P}_{\text{doubler}} = \{q_0BBq_9, q_01Bq_1, q_1BRq_2, q_11Rq_2, q_2BRq_3, q_21Rq_2, q_3B1q_4, q_31Rq_3, q_4BBq_4, q_41Rq_5, q_5B1q_6, q_511q_6, q_6BLq_7, q_61Lq_6, q_7BRq_8, q_71Lq_7, q_8BRq_9, q_811q_0\}$$

Here is the sequence of tapes for the input 2. It shows the blanks because there can be two of them, which can be confusing. It is in halves just to allow a page break in the middle.

Step	Configuration	Step	Configuration	Step	Configuration
0	1 1 q <sub>0</sub>	5	B 1 B 1 q <sub>4</sub>	10	B 1 B 1 1 q <sub>7</sub>
1	B 1 q <sub>1</sub>	6	B 1 B 1 B q <sub>5</sub>	11	B 1 B 1 1 q <sub>7</sub>
2	B 1 q <sub>2</sub>	7	B 1 B 1 1 q <sub>6</sub>	12	B 1 B 1 1 q <sub>8</sub>
3	B 1 B q <sub>2</sub>	8	B 1 B 1 1 q <sub>6</sub>	13	B 1 B 1 1 q <sub>0</sub>
4	B 1 B B q <sub>3</sub>	9	B 1 B 1 1 q <sub>6</sub>	14	B B B 1 1 q <sub>1</sub>

This is the second half.

Step	Configuration	Step	Configuration	Step	Configuration
15	B B B 1 1 q <sub>2</sub>	20	B B B 1 1 1 B q <sub>5</sub>	25	B B B 1 1 1 1 q <sub>6</sub>
16	B B B 1 1 q <sub>3</sub>	21	B B B 1 1 1 1 q <sub>6</sub>	26	B B B 1 1 1 1 q <sub>7</sub>
17	B B B 1 1 q <sub>3</sub>	22	B B B 1 1 1 1 q <sub>6</sub>	27	B B B 1 1 1 1 q <sub>8</sub>
18	B B B 1 1 B q <sub>3</sub>	23	B B B 1 1 1 1 q <sub>6</sub>	28	B B B 1 1 1 1 q <sub>9</sub>
19	B B B 1 1 1 q <sub>4</sub>	24	B B B 1 1 1 1 q <sub>6</sub>		

(B) To double a number represented in binary just append a 0 to the right side. The machine has three instructions and  $\Sigma = \{0, 1, B\}$ .

$$\mathcal{P}_{\text{doublerbinary}} = \{q_0BBq_3, q_000q_3, q_01Rq_1, q_1B0q_2, q_10Rq_1, q_11Rq_1, q_2BRq_3, q_20Lq_2, q_21Lq_2\}$$

On input 6 it gives this tape sequence.

Step	Configuration	Step	Configuration	Step	Configuration
0	1 1 0 q <sub>0</sub>	4	1 1 0 0 q <sub>2</sub>	8	1 1 0 0 q <sub>2</sub>
1	1 1 0 q <sub>1</sub>	5	1 1 0 0 q <sub>2</sub>	9	1 1 0 0 q <sub>3</sub>
2	1 1 0 q <sub>1</sub>	6	1 1 0 0 q <sub>2</sub>		
3	1 1 0 q <sub>1</sub>	7	1 1 0 0 q <sub>2</sub>		

I.1.25 This Turing machine works.

$$\mathcal{P}_{\text{odd}} = \{q_0BBq_{100}, q_01Rq_1, q_1BLq_{100}, q_11Bq_2, q_2BLq_3, q_21Lq_3, q_3BBq_4, q_31Bq_4, q_4BRq_5, q_411q_5, q_5BRq_0, q_511q_0\}$$

For instance, on input 3 it does this

Step	Configuration	Step	Configuration	Step	Configuration
0	$\overbrace{111}^{q_0}$	3	$\overbrace{11}^{q_3}$	6	$\overbrace{1}^{q_0}$
1	$\overbrace{111}^{q_1}$	4	$\overbrace{1}^{q_4}$	7	$\overbrace{1}^{q_1}$
2	$\overbrace{11}^{q_2}$	5	$\overbrace{1}^{q_5}$	8	$\overbrace{1}^{q_{100}}$

and on 4 it does this.

Step	Configuration	Step	Configuration	Step	Configuration
0	$\overbrace{1111}^{q_0}$	5	$\overbrace{11}^{q_5}$	10	$\overbrace{\phantom{1111}}^{q_4}$
1	$\overbrace{1111}^{q_1}$	6	$\overbrace{11}^{q_0}$	11	$\overbrace{\phantom{1111}}^{q_5}$
2	$\overbrace{111}^{q_2}$	7	$\overbrace{11}^{q_1}$	12	$\overbrace{\phantom{1111}}^{q_0}$
3	$\overbrace{111}^{q_3}$	8	$\overbrace{1}^{q_2}$	13	$\overbrace{\phantom{1111}}^{q_{100}}$
4	$\overbrace{11}^{q_4}$	9	$\overbrace{1}^{q_3}$		

**I.1.26** While this Turing machine is moving right, if it hits a comma then it replaces it with a 1 and closes up to the left. That is, it scans left to the start of the string, erases the first 1, and then starts moving right again.

$$\mathcal{P}_{\text{addany}} = \{q_0BLq_{10}, q_01Rq_0, q_0,1q_1, q_1BRq_2, q_11Lq_1, q_1,Lq_1, q_2BRq_0, q_21Bq_2, \\ q_{10}BRq_{100}, q_{10}1Lq_{10}, q_{10},Lq_{10}\}$$

For instance, on input 11,1,11 it does this.

Step	Configuration	Step	Configuration	Step	Configuration
0	$\overbrace{11,1,11}^{q_0}$	6	$\overbrace{1111,11}^{q_1}$	12	$\overbrace{111,11}^{q_0}$
1	$\overbrace{11,1,11}^{q_0}$	7	$\overbrace{1111,11}^{q_2}$	13	$\overbrace{111111}^{q_1}$
2	$\overbrace{11,1,11}^{q_0}$	8	$\overbrace{111,11}^{q_2}$	14	$\overbrace{111111}^{q_1}$
3	$\overbrace{1111,11}^{q_1}$	9	$\overbrace{111,11}^{q_0}$	15	$\overbrace{111111}^{q_1}$
4	$\overbrace{1111,11}^{q_1}$	10	$\overbrace{111,11}^{q_0}$	16	$\overbrace{111111}^{q_1}$
5	$\overbrace{1111,11}^{q_1}$	11	$\overbrace{111,11}^{q_0}$	17	$\overbrace{111111}^{q_1}$

(It is split into two sets of tables to give the formatter a place to put a page break.)

Step	Configuration	Step	Configuration	Step	Configuration
18	1 1 1 1 1 1 q <sub>2</sub>	23	1 1 1 1 1 q <sub>0</sub>	28	1 1 1 1 1 q <sub>10</sub>
19	1 1 1 1 1 q <sub>2</sub>	24	1 1 1 1 1 q <sub>0</sub>	29	1 1 1 1 1 q <sub>10</sub>
20	1 1 1 1 1 q <sub>0</sub>	25	1 1 1 1 1 q <sub>0</sub>	30	1 1 1 1 1 q <sub>10</sub>
21	1 1 1 1 1 q <sub>0</sub>	26	1 1 1 1 1 q <sub>10</sub>	31	1 1 1 1 1 q <sub>10</sub>
22	1 1 1 1 1 q <sub>0</sub>	27	1 1 1 1 1 q <sub>10</sub>	32	1 1 1 1 1 q <sub>100</sub>

**I.1.27** No. Given a configuration  $C = \langle q, s, \tau_L, \tau_R \rangle$ , one preceding configuration is  $\hat{C} = C$  with  $\mathcal{I} = q s s q$ .

**I.1.28**

(A) The machine can flip the first bit, slide right and flip the second bit, etc. The fact that we know the number of bits in advance means that we can just have four groups of states: first flip and slide, second flip and slide, etc. (We could do an arbitrary number of bits but it would be a little harder.) At the end we slide left to position the head at the left end of the non-blank characters.

Although the question says that you need not produce a machine, here is one. In this machine, state  $q_0$  flips the first bit and then  $q_1$  slides right. The next flip and slide combination is  $q_2$  and  $q_3$ , etc., until the machine gets to the fourth bit. Then it moves the head back to the start with  $q_7$ .

$$\mathcal{P}_{\text{bitwiseNOT}} = \{q_0 B B q_1, q_0 0 1 q_1, q_0 1 0 q_1, q_1 B R q_2, q_1 0 R q_2, q_1 1 R q_2, q_2 B B q_3, q_2 0 1 q_3, q_2 1 0 q_3, q_3 B R q_4, q_3 0 R q_4, q_3 1 R q_4, q_4 B B q_5, q_4 0 1 q_5, q_4 1 0 q_5, q_5 B R q_6, q_5 0 R q_6, q_5 1 R q_6, q_6 B B q_7, q_6 0 1 q_7, q_6 1 0 q_7, q_7 B R q_8, q_7 0 L q_7, q_7 1 L q_7\}$$

For instance, on input 0110 it does this.

Step	Configuration	Step	Configuration	Step	Configuration
0	0 1 1 0 q <sub>0</sub>	5	1 0 0 0 q <sub>5</sub>	9	1 0 0 1 q <sub>7</sub>
1	1 1 1 0 q <sub>1</sub>	6	1 0 0 0 q <sub>6</sub>	10	1 0 0 1 q <sub>7</sub>
2	1 1 1 0 q <sub>2</sub>	7	1 0 0 1 q <sub>7</sub>	11	1 0 0 1 q <sub>7</sub>
3	1 0 1 0 q <sub>3</sub>	8	1 0 0 1 q <sub>7</sub>	12	1 0 0 1 q <sub>8</sub>
4	1 0 1 0 q <sub>4</sub>				

(B) The machine has two cases, depending on what it finds for  $b_3$ . If  $b_3 = 0$  then it moves right, copying the next bit into the current one, and then at the end it appends a 0. Likewise, if  $b_3 = 1$  then it moves right, copying, and then at the end it appends a 1. As with the prior item, because we know in advance that there are four bits we can just have three groups of states.

(C) One implementation is to expect two four-bit strings,  $a_3 a_2 a_1 a_0$  and  $b_3 b_2 b_1 b_0$ , separated by a blank. The machine reads  $a_3$  and goes into one of two states,  $r_0$  or  $r_1$ , depending on whether  $a_3 = 0$  or  $a_3 = 1$ . From state  $r_0$  it slides past the blank separator, then reads  $b_3$ , and changes it to the logical 'and' of  $b_3$  and 0. Similarly, from state  $r_1$  it slides past the blank separator, reads  $b_3$ , and changes it to the logical 'and' of  $b_3$  and 1. Repeat that for a total of four iterations.

## I.1.29

- (A) Under the convention that the machine starts with its head under the first non-blank character if there are any,  $\mathcal{P} = \{q_0BBq_1, q_011q_0\}$  will halt only if the input is blank.
- (B) The machine  $\mathcal{P} = \{q_0BBq_0, q_011q_1\}$  will fail to halt only if the input is blank.
- (C) The machine  $\mathcal{P} = \{q_0BRq_1, q_01Rq_1, q_1BBq_1, q_111q_2\}$  will halt if and only if the second character is 1. (Trying the same idea for the first character won't work because of our convention that the first character is blank only if all characters are blank, so there is only one input string with a blank first character.)

**I.1.30** The strategy here is to scan right. If the machine sees a b then it goes into a state,  $q_{20}$  that signifies it has seen one b and if the next character is also a b then it will halt and go to the accepting state. If it sees an a then it goes into state  $q_{10}$ , signifying that the number of b's in the consecutive count is zero.

$$\mathcal{P}_{\text{twobs}} = \{q_0BBq_4, q_0aRq_{10}, q_0bRq_{20}, q_{10}BBq_4, q_{10}aRq_{10}, q_{10}bRq_{20}, q_{20}BBq_4, q_{20}aRq_0, q_{20}bbq_3\}$$

As an example, on input abba it does this.

Step	Configuration	Step	Configuration
0	$\begin{array}{c} \text{a b b a} \\ \text{q}_0 \end{array}$	2	$\begin{array}{c} \text{a b b a} \\ \text{q}_{20} \end{array}$
1	$\begin{array}{c} \text{a b b a} \\ \text{q}_{10} \end{array}$	3	$\begin{array}{c} \text{a b b a} \\ \text{q}_3 \end{array}$

Note that it ends in the accept state. In contrast, on input aba the machine ends in the reject state.

Step	Configuration	Step	Configuration
0	$\begin{array}{c} \text{a b a} \\ \text{q}_0 \end{array}$	3	$\begin{array}{c} \text{a b a} \\ \text{q}_0 \end{array}$
1	$\begin{array}{c} \text{a b a} \\ \text{q}_{10} \end{array}$	4	$\begin{array}{c} \text{a b a} \\ \text{q}_4 \end{array}$
2	$\begin{array}{c} \text{a b a} \\ \text{q}_{20} \end{array}$		

**I.1.31** For this machine, if the first character is 0 then it puts a 1 in its place, marking that the string is in the language. Otherwise, it puts a 0. Then it blanks the remaining characters, and returns to the first character.

$$\mathcal{P}_{\text{starts with zero}} = \{q_001q_1, q_010q_1, q_0B0q_{10}, q_10Rq_2, q_11Rq_2, q_20Bq_3, q_21Bq_3, q_2BBq_4, q_3BRq_2, q_4BLq_4, q_409q_{10}, q_411q_{10}\}$$

This is the action of the machine on 01101.

Step	Configuration	Step	Configuration	Step	Configuration
0	$\begin{array}{c} 01101 \\ \text{q}_0 \end{array}$	6	$\begin{array}{c} 1BB01 \\ \text{q}_2 \end{array}$	12	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$
1	$\begin{array}{c} 11101 \\ \text{q}_1 \end{array}$	7	$\begin{array}{c} 1BBB1 \\ \text{q}_3 \end{array}$	13	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$
2	$\begin{array}{c} 11101 \\ \text{q}_2 \end{array}$	8	$\begin{array}{c} 1BBB1 \\ \text{q}_2 \end{array}$	14	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$
3	$\begin{array}{c} 1B101 \\ \text{q}_3 \end{array}$	9	$\begin{array}{c} 1BBBBB \\ \text{q}_3 \end{array}$	15	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$
4	$\begin{array}{c} 1B101 \\ \text{q}_2 \end{array}$	10	$\begin{array}{c} 1BBBBBB \\ \text{q}_2 \end{array}$	16	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$
5	$\begin{array}{c} 1BB01 \\ \text{q}_3 \end{array}$	11	$\begin{array}{c} 1BBBBBB \\ \text{q}_4 \end{array}$	17	$\begin{array}{c} 1BBBBBB \\ \text{q}_{10} \end{array}$

**I.1.32** For this machine, if the first character is not a 0 then it puts a 0 in its place, marking that the string as not in the language. If the first character is 0, then it checks whether the second character is 1. If so, it leaves it alone, otherwise it puts 0, marking that the input is not in the language. Then it blanks the remaining characters, and returns to the single character marking whether the input is a member.

$$\mathcal{P}_{\text{starts with } 01} = \{q_0 0 B q_1, q_0 1 0 q_3, q_0 B 0 q_{10}, q_1 B R q_2, q_2 0 R q_4, q_2 1 R q_4, q_2 B 0 q_{10}, q_3 0 R q_4, q_4 0 B q_5, q_4 1 B q_5, q_4 B B q_6, q_5 B R q_4, q_6 0 0 q_{10}, q_6 1 1 q_{10}, q_6 B L q_6\}$$

This is the machine on 01101.

Step	Configuration	Step	Configuration	Step	Configuration
0	0 1 1 0 1 q <sub>0</sub>	6	B 1 B B 1 q <sub>5</sub>	11	B 1 B B B B q <sub>6</sub>
1	B 1 1 0 1 q <sub>1</sub>	7	B 1 B B 1 q <sub>4</sub>	12	B 1 B B B B q <sub>6</sub>
2	B 1 1 0 1 q <sub>2</sub>	8	B 1 B B B q <sub>5</sub>	13	B 1 B B B B q <sub>6</sub>
3	B 1 1 0 1 q <sub>4</sub>	9	B 1 B B B B q <sub>4</sub>	14	B 1 B B B B q <sub>6</sub>
4	B 1 B 0 1 q <sub>5</sub>	10	B 1 B B B B q <sub>6</sub>	15	B 1 B B B B q <sub>10</sub>
5	B 1 B 0 1 q <sub>4</sub>				

**I.1.33** The main thing with this machine is that there could be zero-many 0's. That is, the machine should accept a string that starts with 1. On the other hand, if the input string is 000, without a 1, then it is not in the language.

With that, the machine reads past initial 0's, if any, blanking them out. If it find a 1 then it leaves it alone, marking that the input is in the language. Then it blanks the remaining characters, and returns to the single character marking whether the input is a member. Otherwise it puts 0, marking that the input is not in the language.

$$\mathcal{P}_{\text{contains } 1} = \{q_0 0 B q_1, q_0 1 R q_2, q_0 B 0 q_{10}, q_1 B R q_0, q_2 0 B q_3, q_2 1 B q_3, q_2 B L q_4, q_3 B R q_2, q_4 0 0 q_{10}, q_4 1 1 q_{10}, q_4 B L q_4\}$$

This is the machine with input 00110.

Step	Configuration	Step	Configuration	Step	Configuration
0	0 0 1 1 0 q <sub>0</sub>	5	B B 1 1 0 q <sub>2</sub>	10	B B 1 B B B q <sub>4</sub>
1	B 0 1 1 0 q <sub>1</sub>	6	B B 1 B 0 q <sub>3</sub>	11	B B 1 B B B q <sub>4</sub>
2	B 0 1 1 0 q <sub>0</sub>	7	B B 1 B 0 q <sub>2</sub>	12	B B 1 B B B q <sub>4</sub>
3	B B 1 1 0 q <sub>1</sub>	8	B B 1 B B q <sub>3</sub>	13	B B 1 B B B q <sub>10</sub>
4	B B 1 1 0 q <sub>0</sub>	9	B B 1 B B B q <sub>2</sub>		

**I.1.34** The idea of this machine is that it starts with two groups of 1's. The head blanks the first 1 in the first group, then moves over and blanks the last 1 in the second group. At that point the relation of

less-than-or-equal exists between the original two groups of 1's if and only if it exists between what remains of the two groups. Iterating the machine ends with one of two cases: the first string is empty (in which case we must blank out what's left of the second string, if any, and print a 1), and the second string is empty but the first is not (in this case we blank out what's left of the first string). The first case is handled by instructions starting with  $q_{50}$  and the second with instructions starting with  $q_{75}$ .

$$\begin{aligned} P_{\text{leq}} = \{ & q_0BRq_{50}, q_01Bq_1, q_1BRq_2, q_111q_2, q_2BRq_3, q_21Rq_2, q_3BLq_{75}, q_31Rq_4, q_4BLq_5, q_41Rq_4, \\ & q_5BBq_6, q_51Bq_6, q_6BLq_7, q_611q_7, q_7BLq_8, q_71Lq_7, q_8BRq_0, q_81Lq_8, \\ & q_{50}BRq_{51}, q_{50}11q_{51}, q_{51}B1q_{100}, q_{51}1Bq_{52}, q_{52}BRq_{51}, q_{52}11q_{51}, \\ & q_{75}BLq_{76}, q_{75}11q_{76}, q_{76}BLq_{100}, q_{76}1Bq_{77}, q_{77}BLq_{76}, q_{77}1Lq_{76} \} \end{aligned}$$

For instance, on inputs 2 and 3 it does this.

Step	Configuration	Step	Configuration	Step	Configuration
0	1 1 1 1 1 $q_0$	5	B 1 B 1 1 1 $q_3$	10	B 1 B 1 1 1 1 1 $q_6$
1	B 1 1 1 1 $q_1$	6	B 1 B 1 1 1 $q_3$	11	B 1 B 1 1 1 1 1 $q_6$
2	B 1 1 1 1 $q_2$	7	B 1 B 1 1 1 B $q_3$	12	B 1 B 1 1 1 1 1 $q_6$
3	B 1 B 1 1 1 $q_2$	8	B 1 B 1 1 1 1 $q_4$	13	B 1 B 1 1 1 1 1 $q_6$
4	B 1 B 1 1 1 $q_3$	9	B 1 B 1 1 1 1 B $q_5$	14	B 1 B 1 1 1 1 1 $q_6$

(This tape sequence is split in two to leave a place for a line break.)

Step	Configuration	Step	Configuration	Step	Configuration
15	B 1 B 1 1 1 1 1 $q_6$	21	B B B 1 1 1 1 1 $q_2$	27	B B B 1 1 1 1 1 B $q_3$
16	B 1 B 1 1 1 1 1 $q_7$	22	B B B 1 1 1 1 1 $q_3$	28	B B B 1 1 1 1 1 1 $q_4$
17	B 1 B 1 1 1 1 1 $q_7$	23	B B B 1 1 1 1 1 $q_3$	29	B B B 1 1 1 1 1 1 B $q_5$
18	B 1 B 1 1 1 1 1 $q_8$	24	B B B 1 1 1 1 1 $q_3$	30	B B B 1 1 1 1 1 1 1 $q_6$
19	B 1 B 1 1 1 1 1 $q_0$	25	B B B 1 1 1 1 1 $q_3$	31	B B B 1 1 1 1 1 1 1 $q_6$
20	B B B 1 1 1 1 1 $q_1$	26	B B B 1 1 1 1 1 $q_3$		

**I.1.35** The strategy is to blank out the first character, slide to the end, and then blank out the final character (if it matches the first), and then repeat. When the string is empty, the machine puts a 1 on the tape, showing success.

The machine below remembers that the first character was an a by moving to state  $q_{20}$ . It remembers a b by moving to state  $q_{30}$ . Starting in  $q_{40}$  it moves back to the start of the remaining input string. State  $q_{60}$  is for the cases where the machine detects that the input was not a palindrome, while state  $q_{80}$  is for when the machine finds it is one. Finally, states  $q_1, \dots, q_4$  handle the case that the input string (either the initial string or

what is left of the initial string after some end-trimming) has one character.

$$\mathcal{P}_{\text{pal}} = \{q_0\text{BB}q_{80}, q_0\text{aB}q_1, q_0\text{bB}q_3, q_1\text{BR}q_2, q_1\text{aR}q_2, q_1\text{bR}q_2, q_2\text{BL}q_{80}, q_2\text{aa}q_{20}, q_2\text{bb}q_{20}, q_3\text{BR}q_4, q_3\text{aR}q_4, q_3\text{bR}q_4, q_4\text{BL}q_{80}, q_4\text{aa}q_{30}, q_4\text{bb}q_{30}, q_{20}\text{BL}q_{21}, q_{20}\text{aR}q_{20}, q_{20}\text{bR}q_{20}, q_{21}\text{BB}q_{60}, q_{21}\text{aB}q_{40}, q_{21}\text{bb}q_{60}, q_{30}\text{BL}q_{31}, q_{30}\text{aR}q_{30}, q_{30}\text{bR}q_{30}, q_{31}\text{BB}q_{60}, q_{31}\text{aa}q_{60}, q_{31}\text{bB}q_{40}, q_{40}\text{BL}q_{41}, q_{40}\text{aL}q_{41}, q_{40}\text{bL}q_{41}, q_{41}\text{BR}q_0, q_{41}\text{aL}q_{41}, q_{41}\text{bL}q_{41}, q_{60}\text{BB}q_{100}, q_{60}\text{aB}q_{61}, q_{60}\text{bB}q_{61}, q_{61}\text{BL}q_{60}, q_{61}\text{aa}q_{60}, q_{61}\text{bb}q_{60}, q_{80}\text{B1}q_{100}, q_{80}\text{a1}q_{100}, q_{80}\text{b1}q_{100}\}$$

As an example, this is the action of the machine on abba.

Step	Configuration	Step	Configuration	Step	Configuration
0		4		8	
1		5		9	
2		6		10	
3		7		11	

(The tape sequence is in two parts to leave room for a page break.)

Step	Configuration	Step	Configuration	Step	Configuration
12		16		20	
13		17		21	
14		18		22	
15		19			

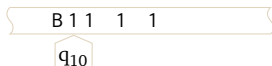
A second example is the action of the machine on aba.

Step	Configuration	Step	Configuration	Step	Configuration
0		5		10	
1		6		11	
2		7		12	
3		8		13	
4		9		14	

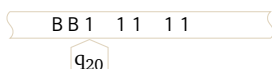
**I.1.36** The strategy is to make two copies of the starting block of 1's, then move the head to the start of the first of the two copies. This is an example initial configuration.



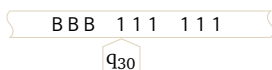
The machine strips the leading 1, moves right, puts in the 1 that will start the first copy block, and then puts in the 1 to start the second block. With that, the head moves back to the start.



Now comes the complication. The machine strips the leading 1, then moves right and puts in a 1 at the end of the first copy block. But the place where it must put this 1 is the blank separating the two copy blocks. So it must then move right and put in a blank before moving to the end of the second copy block. At that end it puts two 1's, the first to mark the stripped leading 1 and the second to mark the inserted blank. With that, the head moves back to the start.



The machine iterates the prior sequence of steps until it finds that the leading sequence of 1's is now blank. To finish, it moves the head to the start of the first copy block.



**I.1.37** Let the machine have the same configuration at two different steps  $\mathcal{C}(t) = \mathcal{C}(\hat{t})$  with  $t < \hat{t}$ . By determinism,  $\mathcal{C}(t + 1) = \mathcal{C}(\hat{t} + 1)$ . Likewise,  $\mathcal{C}(t + i) = \mathcal{C}(\hat{t} + i)$  for  $i = 2, \dots$ . This means that the machine cycles every  $\hat{t} - t$  steps, because  $i = \hat{t} - t$  gives that  $\mathcal{C}(t + (\hat{t} - t)) = \mathcal{C}(\hat{t} + (\hat{t} - t))$  but  $\mathcal{C}(t + (\hat{t} - t)) = \mathcal{C}(\hat{t})$ .

This sufficient condition is not necessary, since we can make a machine that never repeats its configuration but also never halts. One way is to make a machine that writes a 1 to the tape, then moves right, and then repeats.

**I.1.38** Consider this configuration, with the machine in state  $q_5$ .



It could have arrived at that configuration after this configuration and instruction.



or this



or for that matter, this.



**I.2.2** Church's Thesis is an assertion that 'mechanically computable' (by a discrete and deterministic machine) is equivalent to 'computable by a Turing machine'. It is not a theorem because we cannot derive it from the traditional axioms for mathematics.

In some ways it is more like a definition than a theorem. But it is not a usual definition, where as a convenience we take a word so that we don't have to keep saying a phrase over and over, as where we define 'bachelor' as 'unmarried man' or 'linearly dependent' as ' $x_0 = a_1 x_1 + \dots + a_n x_n$ '. Rather, it is more like Calculus's definition of the derivative in that it expresses the introduction of a fundamental concept.

**I.2.3**

- (A) As with a program, a Turing Machine can implement an algorithm. But an algorithm may be implemented by more than one Turing Machine. For instance, we can do a shell sort of a list of five numbers with two different Turing Machines, if only by renaming some of the states. In the words of M Davis (Davis 2016), “[Church’s Thesis] says nothing about what algorithms ARE. It is about what algorithms can and can not DO.” *Comment:* there is no widely agreed-upon definition of algorithm, which is odd as algorithms are a central study in Computer Science.
- (B) A Turing Machine is a single-purpose device. For instance, we may write a Turing Machine to multiply two of its inputs. But today we use “computer” to mean a general-purpose, that is, programmable, device. *Comment:* we will later see that there are general-purpose Turing Machines, so the distinction is blurry, one of connotation rather than of precise definition.
- I.2.4** Anything that can be done on the modern computers that we have in our pockets can be done, in principle, on an old-timey-style mechanical device. For more detail, see ?? B; the low level construction details covered there can be done with gears and levers. So the phrase ‘mechanically computable’ is not about the exact implementation, it covers any way of computing that is discrete and deterministic, and physically realizable.
- I.2.5**
- (A) The tape, as given in the definition, is not infinite. It is unbounded.  
For an analogy, consider the dictionary definition of a book. In that definition there is no upper limit on the number of pages. But no book, no actual bound collection of pages, has infinitely many. So also, no halting Turing machine computation uses infinitely much tape.  
While it is true that you can write a Turing machine that does not halt and that just keeps using more and more tape, at no step of the calculation is the amount of tape infinite. It never runs out — it is unbounded — but it is never infinite.
- (B) It is a model. A model necessarily discards some aspects of the situation it models. See (Wikipedia 2016). (Besides, just to be argumentative for the sheer pleasure of it, whether the universe is finite — either just currently finite, or finite into any unbounded future — is not clear. In any event it is not a mathematical question.)
- I.2.6** Church’s Thesis is the third.  
The first is mistaken because Church’s Thesis speaks only to discrete and deterministic mechanisms. The second is mistaken because Church’s Thesis does not speak to the capabilities, or limitations, of human computers. There are people who posit that humans can do more than mechanisms can do, and people who speculate that there may be machines that that can do more than people can, even in principle, but Church’s Thesis speaks to an equivalence between what can be done by *any* discrete and deterministic mechanism and what can be done by this one kind, Turing machines. The last misses the mark for much the same reason as the second.
- I.2.7** (From (Andreas Blass 2015).) The first benefit that we get from this thesis is that it lets us connect formal mathematical theorems to real-world issues of computability. For example, the theorem that the word problem for groups is Turing-undecidable has the real-world interpretation that no algorithm can solve all instance of the word problem.  
The second benefit is in mathematics itself, specifically in computability theory. Published proofs that something is Turing-computable almost never proceed by exhibiting a Turing-machine program, or indeed a program in any actual computing language. Sometimes, if the matter is simple enough, they provide some sort of pseudo-code. Most often, though, they merely give an informal description of an algorithm. It is left to the reader to see that this actually does give an algorithm (in the intuitive sense) and therefore, by the Church-Turing thesis, could be simulated by a Turing machine. The usual situation is that, although experts in Turing-machine programming would (if they existed) be able to routinely convert the intuitive algorithm into a Turing machine program, the program would be too large and complicated to be worth writing down.
- I.2.8** Lance Fortnow says it perfectly, “Computation is about process, about the transitions made from one state of the machine to another. Computation is not about the input and the output, point A and point B, but the journey. . . . You can feed a Turing machine an infinite digits of a real number (Siegelmann), have computers interact with each other (Wegner-Goldin), or have a computer that perform an infinite series of tasks (Denning) but in all these cases the process remains the same, each step following Turing’s model.”

(Fortnow 2010).

### I.2.9

- (A) Yes, this is clear. To compute  $f \circ g(x)$  we compute  $y = g(x)$  and then compute  $f(y)$ .  
 (B) The composition  $f \circ g$  may be computable, or may be not computable. On the one hand, the constant function  $g(x) = 0$  is computable, and the composition  $f \circ g(x)$  equals  $f(0)$ . Because  $f(0)$  is fixed, not variable, this result is computable.

On the other hand, if  $g(x) = x$  then the composition is  $f \circ g(x) = f(x)$  is not computable.

**I.2.10** We can simulate a ternary machine using an ordinary programming language, that is, on a binary machine. We can simulate a binary machine on a Turing Machine. Thus we can simulate a ternary machine on a Turing machine. So, anything that can be computed on a ternary machine can be computed on a Turing machine.

The converse is even easier: to simulate a binary machine on a ternary machine just pick a trit and avoid using it.

**I.2.11** The role of Church's Thesis here is that it allows us to avoid exhibiting Turing machines as sets of four-tuples to do these computations. Instead we can describe how to do them inside a usual programming environment.

- (A) Let  $f_0$  be computed by the Turing machine  $\mathcal{P}_0$  and let  $f_1$  be computed by  $TM_1$ . For  $x \in \mathbb{N}$  this will compute  $h(x)$ : run  $\mathcal{P}_0$  on input  $x$  for one step, then  $\mathcal{P}_1$  on  $x$  for a step, then  $\mathcal{P}_0$  for another step, etc., until each halts. When that happens, if it ever does, output 1.  
 (B) As with the prior item, for  $x \in \mathbb{N}$  compute the function on input  $x$  by: run  $\mathcal{P}_0$  on  $x$  for one step, then  $\mathcal{P}_1$  on  $x$  for a step, then  $\mathcal{P}_0$  for another step, etc. The difference here is that if either halts then output 1.  
 (C) Suppose  $f$  is computed by  $\mathcal{P}$ . Fix  $x \in \mathbb{N}$ . Run  $\mathcal{P}$  on input 0 for a step. Then run  $\mathcal{P}$  on input 0 for an additional step and run  $\mathcal{P}$  on 1 for a step. After that, run  $\mathcal{P}$  on input 0 for another step, run  $\mathcal{P}$  on 1 for another step, and start a run of  $\mathcal{P}$  on input 2. In this way we cycle among the machine running on different inputs.

Whenever a computation of  $\mathcal{P}$  on some input  $i$  halts, check whether what is left on its tape equals  $x$ . If so, output 1.

- (D) Let  $f_0$  be computed by the Turing machine  $\mathcal{P}_0$  and let  $f_1$  be computed by  $TM_1$ . For  $x \in \mathbb{N}$ , run  $\mathcal{P}_0$  on input  $x$  until it halts. Suppose the output is  $y$ . Run  $\mathcal{P}_1$  on  $y$  until it halts, and then the output is  $h(x)$ . Since each function  $f_0$  and  $f_1$  was given as total, that output will appear eventually, and so  $h$  is also total.  
 (E) This is the same as the prior item except that the function  $h$  will be partial since there is some input for which  $\mathcal{P}_0$  does not halt.

**I.2.12** Here is a procedure that is intuitively mechanically computable, and so by Church's Thesis is computable.

Let the input be  $n$ . For  $i = 0$  compute  $f(i)$  and see if it equals  $n$ . If so, output 0. If not we iterate: take  $i = 1$  and compute  $f(1)$ . If so, output 0. Otherwise, repeat until the procedure outputs some value; if it never does then we are in the second case of  $h$ 's specification.

**I.2.13** This procedure is intuitively mechanically computable and so by Church's Thesis is computable. Let the input be  $n$ . Run the Turing machine computing  $f$  on input  $n$  until it halts, if it ever does. If so then run the Turing machine computing  $g(n)$  until it halts, if ever. After that, if after that ever comes, output 1. Otherwise we are in the second case of  $h$ 's specification.

**I.2.14** If there is a root then this procedure will find it eventually. If there is no root then this procedure will just run forever, in an unbounded search.

**I.2.15** Perhaps predictably, you end with nothing since there is no bill still in your possession.

**I.2.16** The strategy is to move right on tape 0, writing the complementary bit onto tape 1 until the tape 0 head hits a blank. Then the machine moves left on tape 1. (Below, rather than write the pair of characters as  $\langle x, y \rangle$  we just have  $xy$ .)

$\Delta$	00	01	0B	10	11	1B	B0	B1	BB
$q_0$	00, $q_1$	01, $q_1$	01, $q_1$	10, $q_1$	11, $q_1$	10, $q_1$	B0, $q_1$	B1, $q_1$	LL, $q_2$
$q_1$	RR, $q_0$	RR, $q_0$	RR, $q_0$	RR, $q_0$	RR, $q_0$	RR, $q_0$	RR, $q_0$	RR, $q_0$	LL, $q_2$
$q_2$	0L, $q_2$	0L, $q_2$	0R, $q_3$	1L, $q_2$	1L, $q_2$	1R, $q_3$	BL, $q_2$	BL, $q_2$	BR, $q_3$

(Many of the input pairs cannot happen. For example, if the machine is in state  $q_0$  then the input  $00$  cannot happen.)

**I.2.17** The strategy is to move right on the two tapes, writing the logical and onto tape 1 until on both tapes the heads hit blanks. Then the machine moves left on tape 1. (Below, instead of writing the character pair as  $\langle x, y \rangle$ , we just write  $xy$ .)

$\Delta$	$00$	$01$	$0B$	$10$	$11$	$1B$	$B0$	$B1$	$BB$
$q_0$	$00, q_1$	$00, q_1$	$0B, q_1$	$10, q_1$	$11, q_1$	$1B, q_1$	$BB, q_1$	$BB, q_1$	$LL, q_2$
$q_1$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$RR, q_0$	$LL, q_2$
$q_2$	$LL, q_2$	$LL, q_2$	$0R, q_3$	$LL, q_2$	$LL, q_2$	$1R, q_3$	$BL, q_2$	$BL, q_2$	$BR, q_3$

(Many of these input pairs cannot happen. For example, if the machine is in state  $q_1$  then the input  $B0$  cannot happen.)

**I.3.8** In short, primitive recursion is one way to prove that a function is primitive recursive.

A longer version is that ‘primitive recursive’ is a collection of functions. To show that a function is primitive recursive, you construct it by starting with the zero function, the successor function, and the projection functions. Put these together with function composition, or with the technique of primitive recursion.

Another point: this is a constant function that takes two inputs and outputs 3:  $f(x_0, x_1) = S(S(S(Z(x_0, x_1))))$ . The construction shows that  $f(x_0, x_1) = 3$  is primitive recursive, but that construction does not use the schema of primitive recursion. So primitive recursion is one of the ways to build up functions that are primitive recursive, but composition is another allowed constructor.

**I.3.9** A total recursive function is any computable function  $\phi$  that terminates on all inputs. A primitive recursive function is one that can be derived as in Definition 3.7.

Every primitive recursive function is a total recursive function. The converse does not hold — there are total recursive functions that are not primitive recursive — but at this point in the book you have not yet seen that. It is covered in the next section.

**I.3.10** There the definition gives 1.

**I.3.11**

(A)  $f(0) = 0, f(1) = f(2 \cdot 1 - 2) = f(0) = 0$

(B) By definition  $f(2) = f(2 \cdot 2 - 2) = f(2)$  gives what amounts to an infinite loop.

**I.3.12**

(A)

$n$	0	1	2	3	4	5	6	7	8	9	10
$t(n)$	42	42	42	42	42	42	42	42	42	42	42

(B) This

$$F(y) = \begin{cases} 0 & \text{– if } y = 0 \\ h(F(z), z) & \text{– if } y = S(z) \end{cases}$$

works where  $g$  is the constant  $g = 24$  and  $h(F(z), z) = F(z)$ .

By the way, since  $F$  is constant you can construct it as a primitive recursive function without using the schema of primitive recursion. Use composition:  $F(y) = S(S(\dots Z(y))) = S^{42} \circ Z(y)$ .

**I.3.13** One way to show that is to note that it is the composition  $\text{plus\_two} = S \circ S(x)$ . Another way is to use the addition function with a second argument of  $(S \circ S) \circ Z$ , where  $Z$  is the zero function that takes no inputs.

**I.3.14** Starting with this

$$\text{is\_zero}(y) = \begin{cases} 1 & \text{– if } y = 0 \\ 0 & \text{– if } y = S(z) \end{cases}$$

we get it to fit the schema by taking  $g(x_0, \dots, x_{k-1})$  to be the function of no arguments  $S(Z())$  and taking  $h(\text{is\_zero}(z), x_0, \dots, x_{k-1}, z)$  to be the two-argument zero function.

**I.3.15**

$$(A) \quad \begin{array}{c|cccccccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ t(n) & 0 & 1 & 3 & 6 & 10 & 15 & 21 & 28 & 36 & 45 & 55 \end{array}$$

(B) Writing

$$t(y) = \begin{cases} 0 & \text{-- if } y = 0 \\ h(t(z), z) & \text{-- if } y = S(z) \end{cases}$$

gives that  $g$  is the constant  $g = 0$  and that  $h(t(z), z) = S(\text{plus}(t(z), z))$ .

**I.3.16**

$$(A) \quad \begin{array}{c|cccccccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ t(n) & 0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 & 100 \end{array}$$

(B) Writing

$$s(y) = \begin{cases} g() & \text{-- if } y = 0 \\ h(s(z), z) & \text{-- if } y = S(z) \end{cases}$$

Here,  $g$  is the constant function, the function of no arguments,  $g() = 0$ . And,  $h(s(z), z) = \text{plus}(s(z), \text{pred}(\text{product}(2, S(z))))$ .

**I.3.17**

$$(A) \quad \begin{array}{c|ccccc} n & 0 & 1 & 2 & 3 & 4 & 5 \\ t(n) & 0 & 1 & 8 & 27 & 64 & 125 \end{array}$$

(B) Write

$$d(y) = \begin{cases} g() & \text{-- if } y = 0 \\ h(d(z), z) & \text{-- if } y = S(z) \end{cases}$$

where  $g$  is the constant  $g() = 0$  and  $h(d(z), z) = S(\text{plus}(d(z), \text{plus}(\text{product}(3, S(z)), \text{product}(3, \text{product}(S(z), S(z)))))$ ).

**I.3.18**

$$(A) \quad \begin{array}{c|cccccccccc} n & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ h(n) & 1 & 3 & 7 & 15 & 31 & 63 & 127 & 255 & 511 & 1023 \end{array}$$

(B) Write

$$H(y) = \begin{cases} g() & \text{-- if } y = 0 \\ h(H(z), z) & \text{-- if } y = S(z) \end{cases}$$

where  $g$  is the constant  $g() = 1$  and  $h(H(z), z) = S(\text{product}(H(z), 2))$ .

**I.3.19** The derivation for factorial is this.

$$\text{fact}(y) = \begin{cases} 1 & \text{if } y = 0 \\ \text{product}(\text{fact}(z), S(z)) & \text{if } y = S(z) \end{cases}$$

Here  $g$  is a function of no inputs  $g() = 1$  and  $h(a, b) = \text{product}(a, S(b))$ .

**I.3.20** These are easy to compute by hand. But for fun we can write a program.

```
(define (gcd-euclid n m)
  (if (= m 0)
      n
      (gcd-euclid m (remainder n m))))
```

This version of the same program will mechanically give us the recursive calls.

```
(define (gcd-euclid-verbose n m)
  (if (= m 0)
      (begin
        (printf " return ~a\n" n)
        n)
      (begin
        (printf " call (gcd-euclid ~a ~a)\n" m (remainder n m))
        (gcd-euclid-verbose m (remainder n m)))))
```

(A) This only takes two steps.

```
> (gcd-euclid-verbose 28 12)
call (gcd-euclid 12 4)
call (gcd-euclid 4 0)
return 4
4
```

(B) This also happens quickly.

```
> (gcd-euclid-verbose 104 20)
call (gcd-euclid 20 4)
call (gcd-euclid 4 0)
return 4
4
```

(c) These two are relatively prime.

```
> (gcd-euclid-verbose 309 25)
call (gcd-euclid 25 9)
call (gcd-euclid 9 7)
call (gcd-euclid 7 2)
call (gcd-euclid 2 1)
call (gcd-euclid 1 0)
return 1
1
```

**I.3.21** The properties give this recursion to compute the greatest common divisor.

$$\gcd(a, b) = \begin{cases} a & \text{-- if } b = a \\ 1 & \text{-- if } b = 1 \\ \gcd(b, a) & \text{-- if } b > a \\ \gcd(a - b, b) & \text{-- otherwise} \end{cases}$$

These are easy to compute by hand. For fun we can write a program.

```
(define (gcd a b)
  (cond
    [(= a b)
     a]
    [(= b 1)
     1]
    [(> b a)
     (gcd b a)]
    [else
     (gcd (- a b) b)]))
```

(A) this is easy to check by hand.

```
> (gcd 28 12)
4
```

(B) Here also.

```
> (gcd 104 20)
4
```

(c) These two are relatively prime.

```
> (gcd 309 25)
1
```

**I.3.22**

- (A) The constant function that always returns zero  $C_0(\vec{x})$  is included the definition of primitive recursion, Definition 3.7, as  $Z(\vec{x}) = 0$ . The function that always returns 1 is then  $C_1(\vec{x}) = \rightarrow (C_0(\vec{x}))$ , the function that returns 2 is  $C_2(\vec{x}) = \rightarrow (\rightarrow (C_0(\vec{x})))$ , etc.
- (B) Since the parts of the right-hand side of  $\max(\{x, y\}) = y + (x \div y)$  are all primitive recursive, then the max function is also primitive recursive. Similarly for  $\min(\{x, y\}) = x + y - \max(\{x, y\})$ .
- (c) For any  $x, y \in \mathbb{N}$  either  $x - y \geq 0$  or  $y - x \geq 0$ . Thus  $\text{absdiff}(x, y) = \text{propersub}(x, y) + \text{propersub}(y, x)$  and since addition is primitive recursive,  $\text{absdiff}(x, y)$  is therefore primitive recursive.

**I.3.23**

(A) This primitive recursion defines sign.

$$\text{sign}(y) = \begin{cases} 0 & \text{- if } y = 0 \\ 1 & \text{- if } y => (z) \end{cases}$$

(B) A primitive recursion will do

$$\text{negsign}(y) = \begin{cases} 1 & \text{- if } y = 0 \\ 0 & \text{- if } y => (z) \end{cases}$$

but we can also observe that  $\text{negsign}(y) = 1 - \text{sign}(y)$ , that is,  $\text{negsign}(y) = \text{probersub}(1, \text{sign}(y))$ .

(C) The one is  $\text{lessthan}(x, y) = \text{sign}(\text{probersub}(y, x))$  while the other is  $\text{greaterthan}(x, y) = \text{sign}(\text{probersub}(x, y))$ .

**I.3.24**

(A) We have  $\text{not}(x) = 1 - x = \text{probersub}(1, x)$ . For the two-input ones,  $\text{and}(x, y) = x \cdot y = \text{product}(x, y)$  and  $\text{or}(x, y) = \text{sign}(x + y)$  (an alternative is  $\text{or}(x, y) = 1 - ((1 - x) \cdot (1 - y))$ ).

(B)  $\text{equal}(x, y) = \text{negsign}(\text{lessthan}(x, y) + \text{greaterthan}(x, y))$

**I.3.25**

(A)  $\text{notequal}(x, y) = \text{sign}(\text{lessthan}(x, y) + \text{greaterthan}(x, y))$

(B) The first is  $m(x) = 7 \cdot \text{equal}(x, 1) + 9 \cdot \text{equal}(x, 5)$  (note the use of addition and multiplication, which were shown to be primitive recursive in the subsection body). The second is similar  $n(x, y) = 7 \cdot \text{equal}(x, 1) \cdot \text{equal}(y, 2) + 9 \cdot \text{equal}(x, 5) \cdot \text{equal}(y, 5)$ .

**I.3.26**

(A) Since  $g$  is given as primitive recursive, this function is also primitive recursive  $h(a, b) = \text{plus}(a, g(b))$  because it uses composition. With that, this primitive recursion will do the bounded sum.

$$f(y) = \begin{cases} 0 & \text{- if } y = 0 \\ h(f(z), z) & \text{- if } y => (z) \end{cases}$$

(B) This is similar to bounded sum.

(C) Consider the bounded product function.

$$P_p(\vec{x}, m) = \prod_{0 \leq i < m} p(\vec{x}, i)$$

Note that (i) if  $P_p(\vec{x}, m) = 1$  then  $P_p(\vec{x}, i) = 1$  for all  $i < m$ , and (ii) if  $P_p(\vec{x}, m) = 0$  then  $P_p(\vec{x}, i) = 0$  for all  $i \geq m$ .

Next consider the bounded sum function.

$$S_{P_p}(\vec{x}, m) = \sum_{0 \leq i < m} P_p(\vec{x}, i)$$

Because of (i) and (ii),  $S_{P_p}(\vec{x}, m)$  is the least number  $i$  such that  $p(\vec{x}, i) = 0$ , as desired.

Now just define  $M(\vec{x}, m)$  by cases.

$$M(\vec{x}, m) = \begin{cases} m & \text{- if } S_{P_p}(\vec{x}, m) = 0 \\ S_{P_p}(\vec{x}, m) & \text{- otherwise} \end{cases}$$

**I.3.27**

(A) This is the idea.

$$U(\vec{x}, m) = \begin{cases} 1 & \text{- if } m = 0 \\ p(\vec{x}, m) \cdot U(\vec{x}, z) & \text{- if } m => (z) \end{cases}$$

To be formal we have to do some more work such as expressing the multiplication  $p(\vec{x}, m) \cdot U(\vec{x}, z)$  as  $\text{product}(p(\vec{x}, m), U(\vec{x}, z))$ .

(B) Take  $E(\vec{x}, m) = 1 \dot{-} \hat{E}(\vec{x}, m)$ , where the latter function is below.

$$\hat{E}(\vec{x}, m) = \begin{cases} 0 & \text{-- if } m = 0 \\ (1 \dot{-} p(\vec{x}, m)) \cdot \hat{E}(\vec{x}, z) & \text{-- if } m = \succ(z) \end{cases}$$

(C) This follows straight from the definition on noting that one suitable bound on the quotient  $k$  is the dividend  $y$ , that is,  $\text{divides}(x, y) = \exists k \leq y [y = x \cdot k]$ .

(D) As in the prior item, we need only find a suitable bound. Here,  $\text{prime}(y) = 1$  if and only if both  $1 < y$  and there is no  $x < y$  with  $x > 1$  and  $\text{divides}(x, y)$ . All of those components are primitive recursive.

### I.3.28

(A) The table is straightforward.

$a$	0	1	2	3	4	5	6	7
$\text{rem}(a, 3)$	0	1	2	0	1	2	0	1

(B) The relationship is not true when  $\text{rem}(a, 3) = 2$ .

(C) This follows from the prior item: item (1) is 0, item (2) is also 0, and item (3) is  $\text{rem}(z, 3) + 1$ .

$$\text{rem}(a, 3) = \begin{cases} 0 & \text{-- if } a = 0 \\ 0 & \text{-- if } a = S(z) \text{ and } \text{rem}(z, 3) + 1 = 3 \\ \text{rem}(z, 3) + 1 & \text{-- if } a = S(z) \text{ and } \text{rem}(z, 3) + 1 \neq 3 \end{cases}$$

(D) Here is the explicit definition given in the form for primitive recursion.

$$\text{rem}(a, 3) = \begin{cases} 0 & \text{-- if } a = 0 \\ \text{product}(S(\text{rem}(z, 3)), \text{notequal}(S \text{rem}(z, 3), 3), 3) & \text{-- if } a = S(z) \end{cases}$$

(E) Change the 3's to  $b$ 's.

$$\text{rem}(a, b) = \begin{cases} 0 & \text{-- if } a = 0 \\ \text{product}(S(\text{rem}(z, b)), \text{notequal}(S \text{rem}(z, b), b), b) & \text{-- if } a = S(z) \end{cases}$$

### I.3.29

(A) The division is easy.

$a$	0	1	2	3	4	5	6	7	8	9	10
$\text{div}(a, 3)$	0	0	0	1	1	1	2	2	2	3	3

(B) One way to say it is that  $\text{div}(a + 1, 3) = \text{div}(a, 3)$  except when  $\text{rem}(a + 1, 3) = 0$ .

(C) Based on the prior item, this is the explicit definition given in the form for primitive recursion (again, the order of the two inputs is not important).

$$\text{div}(a, 3) = \begin{cases} 0 & \text{-- if } a = 0 \\ \text{plus}(\text{div}(z, 3), \text{equal}(\text{rem}(S(z), 3), 0)) & \text{-- if } a = S(z) \end{cases}$$

(D) Replace the 3's with  $b$ 's.

$$\text{div}(a, b) = \begin{cases} 0 & \text{-- if } a = 0 \\ \text{plus}(\text{div}(z, b), \text{equal}(\text{rem}(S(z), b), 0)) & \text{-- if } a = S(z) \end{cases}$$

### I.3.30 This starts with bounded minimization.

$$\lfloor x/y \rfloor = \min_{t \leq x} [(t + 1) \cdot y > x]$$

For instance,  $\lfloor 9/2 \rfloor = \min_{t \leq 9} [(t + 1) \cdot 2 > 9]$  gives  $t + 1 = 5$ , so  $t = 4$ . Putting it in terms of previously defined functions, for instance changing the multiplication dot to a call to the plus function, is routine.

### I.3.31

(A) Here are the first eleven values of  $F$ .

$n$	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

### I.3.32

(A) The arithmetic is straightforward. Note the anti-diagonals.

		$x$				
		0	1	2	3	4
	0	0	1	3	6	10
	1	2	4	7	11	16
$y$	2	5	8	12	17	23
	3	9	13	18	24	31
	4	14	19	25	32	40

(B) In  $C(x, y) = 0 + 1 + 2 + \dots + (x + y) + y$ , the sum  $x + y$  is primitive recursive from the section body. It then follows that the function  $0 + 1 + \dots + (x + y)$  is primitive recursive, since it is a bounded sum. With that, adding  $y$  shows that  $C$  is primitive recursive.

### I.3.33

(A)  $P(3, 2) = P(2, 2) + P(2, 1) = P(1, 2) + P(1, 1) + P(1, 1) + P(0, 0) = 0 + 1 + 1 + 1 = 3$ .

(B)  $P(3, 0) = 1$ ,  $P(3, 1) = 3$ ,  $P(3, 3) = 1$

(C) The row is  $\langle 1, 4, 6, 4, 1 \rangle$ .

**I.3.34** This is an implementation.

```
(define (M x)
  (if (<= x 100)
      (M (M (+ x 11)))
      (- x 10)))
```

These are the output values.

$$M(x) = \begin{cases} 91 & \text{-- if } x \in \{0, \dots, 101\} \\ x - 10 & \text{-- otherwise} \end{cases}$$

That this function is primitive recursive follows from Exercise 3.22.

**I.3.35** We use induction. Every initial function in Definition 3.7 is total. Every primitive recursive function is derived from the initial functions using a finite number of applications of function composition and primitive recursion. Both of these combinators yield total function outputs from total function inputs.

**I.3.36** (Davis 1982) Suppose that if two functions  $f, \hat{f}: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  both satisfy Definition 3.3. We will show that for all  $\langle x_0, \dots, x_{n-1}, y \rangle \in \mathbb{N}^{n+1}$  the two are equal  $f(x_0, \dots, x_{n-1}, y) = \hat{f}(x_0, \dots, x_{n-1}, y)$ , by induction on  $y$ .

For the base step, they are equal when  $y = 0$  because they both equal  $g(x_0, \dots, x_{n-1})$ . For the inductive step, suppose that they are equal for  $y = 0, \dots, y = k$  and consider the  $y = k + 1$  case.

$$\begin{aligned} f(x_0, \dots, x_{n-1}, k + 1) &= g(f(x_0, \dots, x_{n-1}, k), x_0, \dots, x_{n-1}, k) \\ &= g(\hat{f}(x_0, \dots, x_{n-1}, k), x_0, \dots, x_{n-1}, k) = \hat{f}(x_0, \dots, x_{n-1}, k + 1) \end{aligned}$$

The second equality follows from the induction hypothesis.

**I.4.9** The value  $\mathcal{H}_4(2, 0) = 1$  is immediate from the definition. Next,  $\mathcal{H}_4(2, 1) = \mathcal{H}_3(2, \mathcal{H}_4(2, 0))$ , giving  $\mathcal{H}_2(2, \mathcal{H}_3(2, 0)) = \mathcal{H}_1(2, \mathcal{H}_2(2, 0)) = 2$ .

The last three are similar; a script is the best way to proceed. This is a copy of the straightforward transcription given in the section body, for convenience.

```
(define (H n x y)
  (cond
    [(= n 0) (+ y 1)]
    [(and (= n 1) (= y 0)) x]
    [(and (= n 2) (= y 0)) 0]
    [(and (> n 2) (= y 0)) 1]
    [else (H (- n 1) x (H n x (- y 1)))])
```

These calls give the answers.

```
> (H 4 2 0)
1
> (H 4 2 1)
2
> (H 4 2 2)
4
> (H 4 2 3)
16
> (H 4 2 4)
65536
```

The last one takes some time; below, the first number is milliseconds of CPU time.

```
> (time (H 4 2 4))
cpu time: 193921 real time: 193804 gc time: 25086
65536
```

Here is the summary.

$y$	0	1	2	3	4
$\mathcal{H}(4, 2, y)$	1	2	4	16	65 536

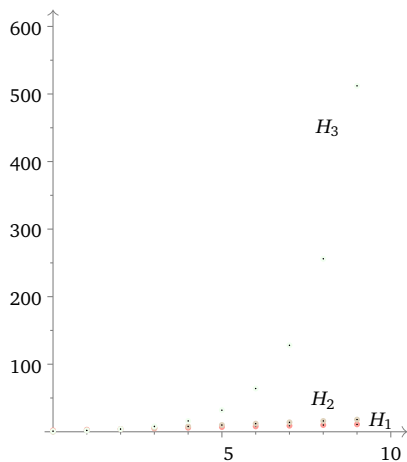
**I.4.10** We can get the values with Lemma 4.2, or the Racket procedure given in the section body.

```
> (for ([y (in-range 10)])
      (printf "~a,~a,\n" y (H 1 2 y)))
(0,2),
(1,3),
(2,4),
(3,5),
(4,6),
(5,7),
(6,8),
(7,9),
(8,10),
(9,11),
```

This summarizes the output.

$y$	0	1	2	3	4	5	6	7	8	9
$\mathcal{H}_1(2, y)$	2	3	4	5	6	7	8	9	10	11
$\mathcal{H}_2(2, y)$	0	2	4	6	8	10	12	14	16	18
$\mathcal{H}_3(2, y)$	1	2	4	8	16	32	64	128	256	512

Here is a graph.



**I.4.11** A script yields that  $\mathcal{H}_4(3, 3) = 7\,625\,597\,484\,987$ . If that is seconds then the number of years is about 241 645.76.

**I.4.12** We have that  $\mathcal{H}_3(3, 3) = 27$  and  $\mathcal{H}_2(2, 2) = 4$  so the ratio is 6.75.

**I.4.13** The proof of Lemma 4.2 shows that  $\mathcal{H}_1(x, y) = x + y$ . The definition of  $\mathcal{H}$  gives

$$\mathcal{H}_2 = \begin{cases} 0 & \text{-- if } y = 0 \\ \mathcal{H}_2(x, \mathcal{H}_1(x, y - 1)) & \text{-- otherwise} \end{cases}$$

and we will use induction to show  $\mathcal{H}_2(x, y) = x \cdot y$ . The base step is part of that definition. So suppose that  $\mathcal{H}_2(x, y) = x \cdot y$  for  $y = 0, y = 1, \dots, y = k$ . For  $y = k + 1$ ,

$$\mathcal{H}_2(x, y) = \mathcal{H}_1(x, x \cdot k) = x + x \cdot k = x \cdot (1 + k) = x \cdot y$$

(because  $y = k + 1$  the ‘otherwise’ branch applies).

The other equality is similar. The key step is  $\mathcal{H}_3(x, y) = \mathcal{H}_2(x, \mathcal{H}_3(x, k)) = \mathcal{H}_2(x, x^k) = x \cdot x^k = x^{k+1} = x^y$ .

**I.4.14** Writing a small script makes the job of computing easier. Here is the table of values.

	$y = 0$	$y = 1$	$y = 2$	$y = 3$	$y = 4$	$y = 5$	
$k = 0$	1	2	3	4	5	6	...
$k = 1$	2	3	4	5	6	7	...
$k = 2$	3	5	7	9	11	13	...
$k = 3$	5	13	29	61	125	253	...
$k = 4$	13	65	533	...			

**I.4.15** In each application of the recursion, either the first argument decreases or else the first argument remains the same and the third argument decreases. Further, each time that the third argument reaches zero, the first argument decreases, so it eventually reaches zero as well. If the first argument is zero then the computation terminates.

**I.4.16** No. All such functions are total, they halt on all inputs, while some computable functions are not total.

**I.4.17** This Racket program illustrates the definition.

```
(define (g x y)
  (if (= 100 (+ x y))
      0
      1))

(define (f x)
  (define (f-helper x y)
    (if (= 0 (g x y))
        y
        (f-helper x (+ 1 y))))
  (let ([y 0])
    (f-helper x 0)))
```

- (A)  $f(0) = 100$
- (B)  $f(1) = 99$
- (C)  $f(50) = 50$
- (D)  $f(100) = 0$
- (E) This is not defined.

This is another description of  $f$ .

$$f(x) = \begin{cases} 100 - x & \text{-- if } x \leq 100 \\ \text{undefined} & \text{-- otherwise} \end{cases}$$

**I.4.18**

- (A) The value of  $f(0)$  is undefined because there is no  $y$  where  $0 \cdot y$  equals 100.
- (B)  $f(1) = 100$
- (C)  $f(50) = 2$
- (D)  $f(100) = 1$
- (E) This is not defined because there is no  $y$  where  $101 \cdot y$  equals 100.

**I.4.19**

- (A)  $f(0) = 3$   
 (B)  $f(1) = 3$   
 (C)  $f(50) = 257$   
 (D)  $f(100) = 257$   
 (E) We don't know if this is defined.

**I.4.20**

- (A)  $f(0) = 0$   
 (B)  $f(1) = 1$   
 (C)  $f(50) = 50^2$   
 (D)  $f(100) = 100^2$   
 (E)  $f(x) = x^2$

**I.4.21**

- (A)  $f(0) = 1$   
 (B)  $f(1) = 1$   
 (C)  $f(2) = 2$   
 (D)  $f(3) = 3$   
 (E)  $f(4) = 2$   
 (F)  $f(42) = 2$   
 (G) In general, if  $x = 0$  or  $x = 1$  then  $f(x) = 1$ . For  $x > 1$ , the value of  $f(x)$  is the smallest prime dividing  $x$ .

**I.4.22**

- (A) This Racket script helps with the calculations.

```
;; g Function given in exercise
(define (g x y)
  (ceiling (- (/ (add1 x) (add1 y))
             1)))

;; f Compute from g by mu-recursion
(define (f x)
  (define (f-helper y)
    (if (= 0 (g x y))
        y
        (f-helper (add1 y))))
  (let ([y 0])
    (f-helper y)))

;; f Compute from g by mu-recursion
```

This table gives  $f(x)$  for  $0 \leq x < 6$  and includes the relevant values of  $g(x, y)$ .

$g(x, y)$	$y = 0$	$y = 1$	$y = 2$	$y = 3$	$y = 4$	$y = 5$	$f(x)$
$x = 0$	0						0
$x = 1$	1	0					1
$x = 2$	2	1	0				2
$x = 3$	3	1	1	0			3
$x = 4$	4	2	1	1	0		4
$x = 5$	5	2	1	1	1	0	5

- (B) We will argue that  $f(x) = x$ . Fix  $x$ . To have  $g(x, y) = 0$  we need that  $(x + 1)/(y + 1)$  is less than or equal to 1. The first  $y$  giving that is  $x$ .

**I.4.23**

- (A)

$$\text{remtwo}(y) = \begin{cases} 0 & \text{-- if } y = 0 \\ 1 \div \text{remtwo}(z) & \text{-- if } y = S(z) \end{cases}$$

- (B) Let  $f(n) = \mu x [x + \text{remtwo}(n) = 0]$ . If  $n$  is even then  $\text{remtwo}(n) = 0$  and  $x = 0$  suffices. If  $n$  is odd then the  $\text{remtwo}(n) = 1$  and there is no  $x$  such that  $x + 1 = 0$ .

I.4.24 We can compute  $f(x)$  by running the machine until it halts. This is the computation with  $x = 0$ .

Step	Configuration
0	
1	
2	
3	

Here is the computation with input  $x = 1$ .

Step	Configuration	Step	Configuration
0		3	
1		4	
2		5	

The same for  $x = 2$ .

Step	Configuration	Step	Configuration	Step	Configuration
0		3		6	
1		4		7	
2		5			

This is 3.

Step	Configuration	Step	Configuration	Step	Configuration
0		4		7	
1		5		8	
2		6		9	
3					

This is 4.

Step	Configuration	Step	Configuration	Step	Configuration
0	$\overbrace{1111}^{q_0}$	4	$\overbrace{1111}^{q_0}$	8	$\overbrace{11111}^{q_1}$
1	$\overbrace{1111}^{q_0}$	5	$\overbrace{11111}^{q_1}$	9	$\overbrace{11111}^{q_1}$
2	$\overbrace{1111}^{q_0}$	6	$\overbrace{11111}^{q_1}$	10	$\overbrace{11111}^{q_1}$
3	$\overbrace{1111}^{q_0}$	7	$\overbrace{11111}^{q_1}$	11	$\overbrace{11111}^{q_2}$

Finally, this is the computation for  $f(5)$ .

Step	Configuration	Step	Configuration	Step	Configuration
0	$\overbrace{11111}^{q_0}$	5	$\overbrace{11111}^{q_0}$	10	$\overbrace{111111}^{q_1}$
1	$\overbrace{11111}^{q_0}$	6	$\overbrace{111111}^{q_1}$	11	$\overbrace{111111}^{q_1}$
2	$\overbrace{11111}^{q_0}$	7	$\overbrace{111111}^{q_1}$	12	$\overbrace{111111}^{q_1}$
3	$\overbrace{11111}^{q_0}$	8	$\overbrace{111111}^{q_1}$	13	$\overbrace{111111}^{q_2}$
4	$\overbrace{11111}^{q_0}$	9	$\overbrace{111111}^{q_1}$		

This table summarizes.

$x$	0	1	2	3	4	5
$f(x)$	3	5	7	9	11	13

**I.4.25** To compute  $f(x)$  we can run the machine until it halts. This is the computation with  $x = 0$ ,

Step	Configuration
0	$\overbrace{\quad}^{q_0}$
1	$\overbrace{1}^{q_2}$

and this is the computation with input  $x = 1$ .

Step	Configuration
0	$\overbrace{1}^{q_0}$
1	$\overbrace{1}^{q_1}$
2	$\overbrace{11}^{q_2}$

This is the same for  $x = 2$ .

Step	Configuration
0	
1	
2	

The  $x = 3$  computation does this.

Step	Configuration
0	
1	
2	

This is  $x = 4$ .

Step	Configuration
0	
1	
2	

Finally, this is the computation for  $f(5)$ .

Step	Configuration
0	
1	
2	

The table below summarizes.

$x$	0	1	2	3	4	5
$f(x)$	2	3	3	3	3	3

For this machine the running time is constant. The machine in the prior exercise does the same job but its running time is a linear function of the input size.

**I.4.26** One way to compute  $f(x)$  is to run the machine until it halts.

(A) This is the computation with  $x = 0$ .

Step	Configuration	Step	Configuration
0		3	
1		4	
2			

(B) This is  $x = 1$ .

Step	Configuration	Step	Configuration
0		3	
1		4	
2			

(C) This is the same for  $x = 2$ .

Step	Configuration	Step	Configuration
0		3	
1		4	
2			

(D) The table summarizes.

$x$	0	1	2
$f(x)$	4	4	4

This machine ignores its input, moves right two squares and back left two, and then halts. So its running time is constant,  $f(x) = 4$ .

**I.2** These are not too hard to compute by hand, but a small program is also handy.

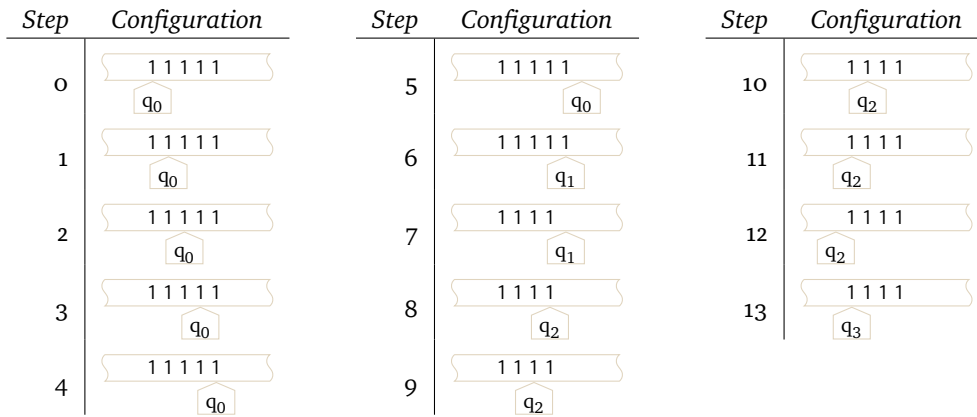
```
;; 3n+1 function
(define (H n)
  (if (even? n)
      (/ n 2)
      (+ (* 3 n) 1)))

;; Collatz number calculator: use unbounded search to find it
(define (C n)
  (define (C-helper n k)
    (if (= 1 n)
        k
        (C-helper (H n) (add1 k))))
  (C-helper n 0))
```

(A) The script gives this.

```
> (H 4)
2
> (H (H 4))
1
> (H (H (H 4)))
4
```



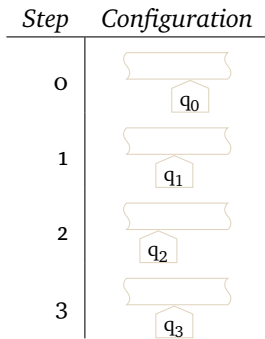


This is the command line invocation for an empty tape,

```

$ ./turing-machine.rkt -f machines/pred.tm -c " "
step 0: q0: *B*
step 1: q1: *B*B
step 2: q2: *B*BB
step 3: q3: B*B*B
step 4: HALT
    
```

and the matching sequence of tape diagrams.



I.A.2 This simulates 1 + 2.

```

$ ./turing-machine.rkt -f machines/pred.tm -c "1" -r " 11"
step 0: q0: *1* 11
step 1: q0: 1*B*11
step 2: q1: 1*B*11
step 3: q1: 1*1*11
step 4: q2: 1*1*11
step 5: q2: *1*111
step 6: q2: *B*1111
step 7: q3: *B*1111
step 8: q3: B*1*111
step 9: q4: B*B*111
step 10: q5: BB*1*11
step 11: HALT
    
```

and this is the matching tape diagram.

Step	Configuration	Step	Configuration	Step	Configuration
0		4		8	
1		5		9	
2		6		10	
3		7			

The command line for  $0 + 2$

```
$ ./turing-machine.rkt -f machines/pred.tm -c " " -r "11"
```

gives these tape diagrams.

Step	Configuration	Step	Configuration	Step	Configuration
0		3		6	
1		4		7	
2		5		8	

The command line for  $0 + 0$

```
$ ./turing-machine.rkt -f machines/pred.tm -c " "
```

gives this sequence of tape diagrams.

Step	Configuration	Step	Configuration	Step	Configuration
0		3		6	
1		4		7	
2		5		8	

**I.A.3** This ten-instruction machine

$$\mathcal{P}_{\text{addthree}} = \{q_0 1 R q_0, q_0 B B q_1, q_1 B 1 q_1, q_1 1 R q_2, q_2 B 1 q_2, q_2 1 R q_3, q_3 B 1 q_3, q_3 1 1 q_4, q_4 1 L q_4, q_4 B R q_5\}$$

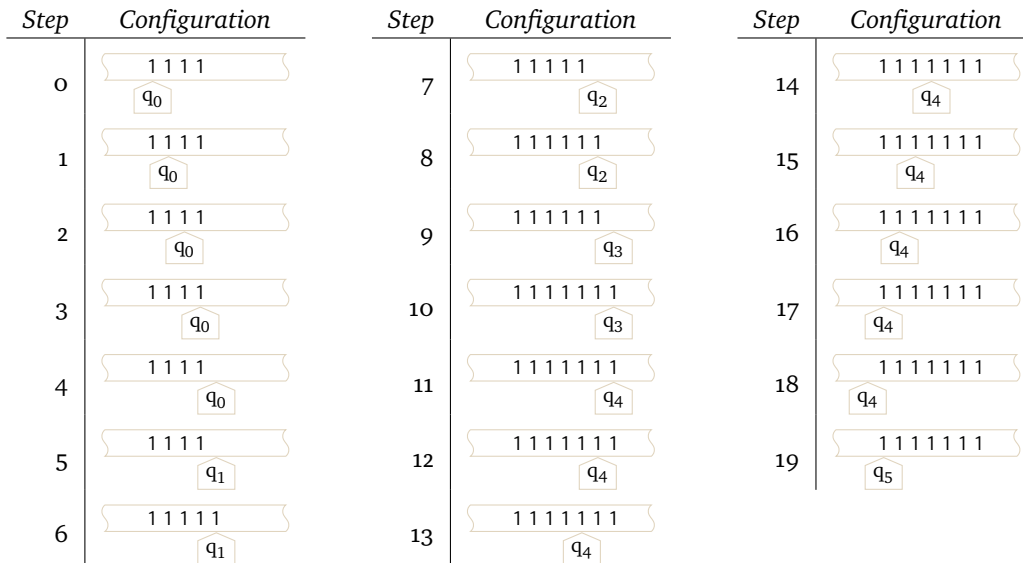
matches the source for the simulator.

```
0 1 R 0
0 B B 1
1 B 1 1
1 1 R 2
2 B 1 2
2 1 R 3
3 B 1 3
3 1 1 4
4 B R 5
4 1 L 4
```

This command line for input 4

```
$. /turing-machine.rkt -f machines/addthree.tm -c "1" -r "111"
```

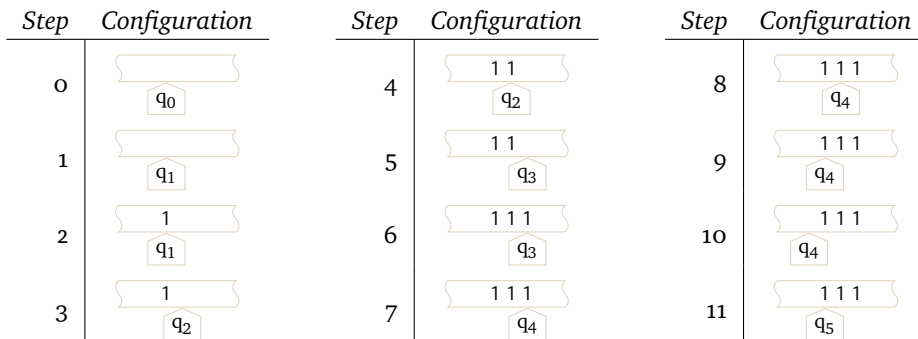
gives these tape diagrams.



The command line for 0

```
$. /turing-machine.rkt -f machines/addthree.tm -c ""
```

gives this sequence of tape diagrams.



I.A.4

(A) Something like this will do.

$$\mathcal{P}_0 = \{q_{10}\theta Rq_{10}, q_{10}1Rq_{10}, q_{10}BBq_{100}\}$$

(B) This will back up, blanking out cells.

$$\mathcal{P}_1 = \{q_{20}\theta Bq_{21}, q_{20}1Bq_{21}, q_{20}BBq_{100}, q_{21}\theta Lq_{20}, q_{21}1Lq_{20}, q_{21}BLq_{20}\}$$

(c) Here, roughly  $q_0$  is where none of the substring is matched,  $q_1$  is where  $\theta$  has been seen and the machine is looking for the 1, and  $q_2$  is where  $\theta 1$  has been seen and the machine is looking for the  $\theta$ . Then the four states  $q_3, q_4, q_5$ , and  $q_6$  mark a failure (they end with  $q_6$  writing  $\theta$  on a blank tape). The four states  $q_7, q_8,$

$q_9$ , and  $q_{10}$  mark a success.

$$\mathcal{P}_{\text{decide}_{010}} = \{q_0\emptyset Rq_1, q_01Rq_0, q_0BBq_3, q_1\emptyset Rq_1, q_11Rq_2, q_1BBq_3, q_2\emptyset Rq_7, q_211q_0, q_2BBq_3, \\ q_3\emptyset Rq_3, q_31Rq_3, q_3BLq_4, q_4\emptyset Bq_5, q_41Bq_5, q_4BBq_6, \\ q_5\emptyset Lq_4, q_51Lq_4, q_5BLq_4, q_6\emptyset\emptyset q_{100}, q_61\emptyset q_{100}, q_6B\emptyset q_{100}, \\ q_7\emptyset Rq_7, q_71Rq_7, q_7BLq_8, q_8\emptyset Bq_9, q_81Bq_9, q_8BBq_{10}, \\ q_9\emptyset Lq_8, q_91Lq_8, q_9BLq_8, q_{10}\emptyset 1q_{100}, q_{10}11q_{100}, q_{10}B1q_{100}\}$$

### I.B.1

(A) This is a table for  $(P \wedge Q) \wedge R$ .

$P$	$Q$	$R$	$P \wedge Q$	$(P \wedge Q) \wedge R$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

(B) And this is a table for  $P \wedge (Q \wedge R)$ . It has the same final column as the one from the prior item.

$P$	$Q$	$R$	$Q \wedge R$	$P \wedge (Q \wedge R)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(C) This is a table for  $P \wedge (Q \vee R)$ .

$P$	$Q$	$R$	$Q \vee R$	$P \wedge (Q \vee R)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

(D) And this is a table for  $(P \wedge Q) \vee (P \wedge R)$ . It has the same final column as the prior item.

$P$	$Q$	$R$	$P \wedge Q$	$P \wedge R$	$(P \wedge Q) \vee (P \wedge R)$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

**I.B.2**(A) This is a table for  $\neg(P \vee Q)$ .

$P$	$Q$	$P \vee Q$	$\neg(P \vee Q)$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(B) The same for  $\neg P \wedge \neg Q$ .

$P$	$Q$	$\neg P$	$\neg Q$	$\neg P \wedge \neg Q$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

It has the same final column as the prior item.

(C) This is a table for  $\neg(P \wedge Q)$ .

$P$	$Q$	$P \wedge Q$	$\neg(P \wedge Q)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

(D) This is the one for  $\neg P \vee \neg Q$ .

$P$	$Q$	$\neg P$	$\neg Q$	$\neg P \vee \neg Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

The final column is the same as the final column of the prior item.

**I.C.4** It takes 17331 turns to stabilize.**I.D.8** From two arguments  $k$  and  $y$ , either  $k$  decreases or  $k$  is fixed and  $y$  decreases. When  $y$  reaches zero then  $k$  decreases, so it eventually reaches zero as well, and  $\mathcal{A}(0, y)$  obviously terminates.**I.D.11** All the parts of  $\mathcal{A}$  are primitive recursive but as a whole it is not? Yes. It is a question of uniformity. Imagine that we have an enumeration of the primitive recursive functions  $f_0, f_1, \dots$ . Perhaps  $\mathcal{A}_0 = f_9$ , and  $\mathcal{A}_1 = f_{121}$ , and  $\mathcal{A}_2 = f_{800}$ . But the function associating the subscript on the  $\mathcal{A}$  with the subscript on the  $f$  is not primitive recursive.

## Chapter II: Background

**II.1.19** To show that the map is one-to-one we will show that  $g(n) = g(\hat{n})$  can only happen if  $n = \hat{n}$ . So let  $g(n) = g(\hat{n})$ , giving  $3n = 3\hat{n}$ . Divide by three to get  $n = \hat{n}$ .

To show the map is onto we will show that every member of the codomain  $y \in \{3k \mid k \in \mathbb{N}\}$  is mapped-to by some member of the domain. Because  $y$  is a multiple of three,  $y/3$  is a natural number. Of course,  $y = g(y/3)$ , so  $y$  is indeed the image of some member of the domain.

**II.1.20** This is a type error. It is not the sets that are one-to-one or onto; rather, it is a function  $f$  between the sets. (For  $D = \{n^2 \mid n \in \mathbb{N}\}$  and  $C = \{n^3 \mid n \in \mathbb{N}\}$  the natural such function  $f: D \rightarrow C$  is  $f(n) = n^{3/2}$ .)

### II.1.21

- (A) The function  $f: \mathbb{Z} \rightarrow \mathbb{Z}$  given by  $f(x) = 2x$  is one-to-one but not onto. Verifying that it is one-to-one is routine: assume that  $f(x) = f(\hat{x})$  for  $x, \hat{x} \in \mathbb{Z}$ , so that  $2x = 2\hat{x}$ , and then cancel the 2's to get that  $x = \hat{x}$ . To show that it is not onto we need only name one codomain element  $y \in \mathbb{Z}$  that is not the image of any domain element. One such is  $y = 1$ , since the image of every domain element is even.
- (B) The function  $g: \mathbb{Z} \rightarrow \mathbb{Z}$  given by  $g(x) = 2x - 1$  is also one-to-one but not onto. The one-to-one argument works just as it did in the prior item: assume that  $g(x) = g(\hat{x})$  for  $x, \hat{x} \in \mathbb{Z}$  so that  $2x - 1 = 2\hat{x} - 1$ . Add 1 to both sides and cancel the 2's to get  $x = \hat{x}$ . This function is not onto because the codomain element  $y = 2$  is not the image of any domain element, since it is even but the number  $2x - 1$  is odd for any input integer  $x$ .
- (C) They are not inverse. For instance,  $f \circ g(1) = f(g(1)) = f(1) = 2$ .

### II.1.22

- (A) The function  $f: \mathbb{N} \rightarrow \mathbb{N}$  given by  $f(n) = n + 1$  is one-to-one but not onto. To check that it is one-to-one suppose that  $f(n_0) = f(n_1)$  for  $n_0, n_1 \in \mathbb{N}$ , so that  $n_0 + 1 = n_1 + 1$ , and then subtract 1 from both sides to get  $n_0 = n_1$ . It is not onto because the codomain element  $0 \in \mathbb{N}$  is not the image of any domain element.
- (B) The map  $f: \mathbb{Z} \rightarrow \mathbb{Z}$  given by  $f(n) = n + 1$  is a correspondence—it is both one-to-one and onto. The verification that it is one-to-one goes: assume that  $f(n_0) = f(n_1)$  so that  $n_0 + 1 = n_1 + 1$ , and subtract 1 to conclude that  $n_0 = n_1$ . The verification that it is onto goes: fix some codomain element  $y \in \mathbb{Z}$  and note that  $x = y - 1$  is a domain element which satisfies that  $f(x) = y$ .
- (C) The function  $f: \mathbb{N} \rightarrow \mathbb{N}$  defined by  $f(n) = 2n$  is one-to-one but not onto. For one-to-one, assume that  $n_0, n_1 \in \mathbb{N}$  are such that  $f(n_0) = f(n_1)$ . Then  $2n_0 = 2n_1$  and cancelling the 2's gives that  $n_0 = n_1$ . It is not onto because the codomain element  $y = 1$  is not the image of any domain element, since all such images are even.
- (D) The function  $f: \mathbb{Z} \rightarrow \mathbb{Z}$  defined by  $f(n) = 2n$  is one-to-one but not onto. The one-to-one verification is as in the prior item: suppose that  $f(n_0) = f(n_1)$  for some  $n_0, n_1 \in \mathbb{Z}$ , so that  $2n_0 = 2n_1$ , and cancel the 2's to conclude that  $n_0 = n_1$ . Also as in the prior item, this map is not onto because the codomain element  $y = 1$  is not in the range of the function, as all of the range elements are even.
- (E) The function  $f: \mathbb{Z} \rightarrow \mathbb{N}$  given by  $f(n) = |n|$  is not one-to-one but it is onto. It is not one-to-one because the two domain elements  $n_0 = -1$  and  $n_1 = 1$  map to the same output,  $f(n_0) = 2 = f(n_1)$ . To show that  $f$  is an onto map consider a codomain element  $y \in \mathbb{N}$ , and observe that if we take the domain element  $x \in \mathbb{Z}$  such that  $x = y$  then  $f(x) = y$ .

### II.1.23

- (A) This function is a correspondence. To verify that it is one-to-one suppose that  $f(q_0) = f(q_1)$  for  $q_0, q_1 \in \mathbb{Q}$ , so that  $q_0 + 3 = q_1 + 3$ , and subtract 3 from both sides to get that  $q_0 = q_1$ . For onto, consider a codomain element  $y \in \mathbb{Q}$  and observe that the domain element  $q \in \mathbb{Q}$  given by  $q = y - 3$  satisfies that  $f(q) = y$ .
- (B) This function is not a correspondence. It is not onto because the codomain element  $1/2 \in \mathbb{Q}$  is not the image of any domain element  $z \in \mathbb{Z}$ , since each such image  $f(z) = z + 3$  is an integer. (This function is

one-to-one: suppose that  $f(z_0) = f(z_1)$ , so that  $z_0 + 3 = z_1 + 3$ , and subtract 3 from both sides to conclude that  $z_0 = z_1$ . But because it is not onto, it is not a correspondence.)

- (C) This is not a correspondence. It is not even a function, because it is not well-defined—the two  $q_0 = 1/2$  and  $q_1 = 2/4$  are equal,  $q_0 = q_1$ , but they are not associated with the same  $|a \cdot b|$ 's since  $|1 \cdot 2| = 2$  and  $|2 \cdot 4| = 8$ .

### II.1.24

- (A) This set is finite. It has the same cardinality as  $I = \{0, 1, 2\}$ , as witnessed by the function  $f: I \rightarrow \{1, 2, 3\}$  given by  $f(x) = x + 1$ , which is one-to-one and onto by inspection.
- (B) The set  $S = \{0, 1, 4, 9, 16, \dots\}$  of perfect squares is infinite. There is no correspondence between some  $I = \{0, 1, \dots, n-1\}$  and  $S$  since there is no onto function—for any  $f: I \rightarrow S$  the set  $\{f(0), f(1), \dots, f(n-1)\}$  cannot equal  $S$  because it has a largest element but  $S$  has no largest element.
- (C) The set of primes is infinite. The argument from the prior item applies.
- (D) By the Fundamental Theorem of Arithmetic, a fifth degree polynomial has at most five real number roots. Thus this set has cardinality at most five, and so is finite.

### II.1.25

- (A) The map  $f: \{0, 1, 2\} \rightarrow \{3, 4, 5\}$  given by  $f(0) = 3$ ,  $f(1) = 4$ , and  $f(2) = 5$ , that is,  $f(x) = x + 3$ , is a correspondence. By inspection each element of the codomain is associated with one and only one element of the domain.
- (B) The function  $f(x) = x^3$  is a correspondence between these sets. For each perfect cube there is one and only one associated integer cube root.

### II.1.26

- (A) The function  $f$  given by  $0 \mapsto \pi$ ,  $1 \mapsto \pi + 1$ ,  $3 \mapsto \pi + 2$ , and  $7 \mapsto \pi + 3$  is both one-to-one and onto, by inspection.
- (B) Let  $E = \{2n \mid n \in \mathbb{N}\}$  and  $S = \{n^2 \mid n \in \mathbb{N}\}$ . Consider the map  $f: E \rightarrow S$  given by  $f(x) = (x/2)^2$ . To show that  $f$  is one-to-one, suppose that  $f(x_0) = f(x_1)$ . Then  $(x_0/2)^2 = (x_1/2)^2$  and because these are natural numbers their square roots are equal,  $x_0/2 = x_1/2$ . Thus  $x_0 = x_1$  and the map is one-to-one.  
For onto, consider  $y \in S$ . Because  $y$  is a perfect square there is a natural number  $n \in \mathbb{N}$  so that  $y = n^2$ . Let  $x = 2n$ . Clearly  $x$  is even and  $f(x) = y$ .
- (C) The second interval is  $2/3$ -rds the width of the first, so consider the function  $f(x) = (2/3) \cdot x - (5/3)$ . It is one-to-one because  $f(x_0) = f(x_1)$  implies that  $(2/3) \cdot x_0 - (5/3) = (2/3) \cdot x_1 - (5/3)$ , and straightforward algebra gives that  $x_0 = x_1$ . To show that  $f$  is onto, fix  $y \in (-1..1)$ . Let  $x = (y + (5/3)) / (2/3)$  (this came from solving  $y = (2/3)x - (5/3)$  for  $x$ ). When  $y$  is an element of the codomain  $(-1..1)$  then this  $x$  is an element of the domain  $(1..4)$ , and more algebra shows that  $f(x) = y$ .

**II.1.27** The problem statement is ambiguous as to which set is the domain and which is the codomain. We will prove that  $f: (0..1) \rightarrow (1..\infty)$  given by  $f(x) = 1/x$  is one-to-one and onto.

For one-to-one assume that  $f(x_0) = f(x_1)$ , where  $x_0, x_1 \in (0..1)$ . Then  $1/x_0 = 1/x_1$  and cross-multiplication gives  $x_1 = x_0$ . For onto suppose that  $y \in (1..\infty)$  and note that if  $x = 1/y$  then  $f(x) = y$  and also  $x \in (0..1)$ .

**II.1.28** Call the sets  $D = \{1, 2, 3, 4, \dots\}$  and  $C = \{7, 10, 13, 16 \dots\}$ . We will show that the function  $f: D \rightarrow C$  described by the formula  $f(x) = 3(x - 1) + 7$  gives a correspondence between the sets. This function is one-to-one since  $f(x_0) = f(x_1)$  implies that  $3(x_0 - 1) + 7 = 3(x_1 - 1) + 7$ , and subtracting 7, dividing by 3, and adding 1 gives that  $x_0 = x_1$ . It is onto because if  $y$  is an element of the codomain  $C$  then it has the form  $y = 3n + 4$  for  $n \in \mathbb{N}^+$ . That equation gives  $y = 3(n - 1) + 7$ . Some algebra gives  $(y - 7)/3 = n - 1$  for  $n \in \mathbb{N}^+$  and we have that  $n = ((y - 7)/3) + 1$  is an element of  $\mathbb{N}^+ = D$  with  $f(n) = y$ .

### II.1.29

- (A) The obvious map is this.

$x$	0	1	2	3	4	5	6	7	8	9
$f(x)$	48	49	50	51	52	53	54	55	56	57

It is one-to-one by inspection, meaning that looking over the association shows that never do two different domain elements map to the same codomain element. Similarly, it is also onto by inspection.

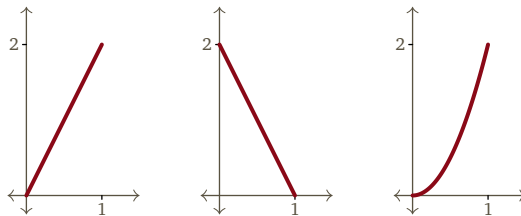


FIGURE 2, FOR QUESTION II.1.31: The correspondences  $f_0, f_1, f_2: (0..1) \rightarrow (0..2)$  given by  $f_0(x) = 2x$ ,  $f_1(x) = 2 - 2x$ , and  $f_2(x) = 2x^2$ .

- (B) The inverse map associates the same pairs of elements in the prior item's table, but the domain and codomain swap places so here  $A$  is the domain and  $C$  is the codomain.

$y$	48	49	50	51	52	53	54	55	56	57
$x$	0	1	2	3	4	5	6	7	8	9

As in the prior item, this map is one-to-one and onto by inspection.

**II.1.30** For each pair of sets we could define a function whose domain is the first set, or a function whose domain is the second set. We choose whichever is convenient.

- (A) Write the even numbers as  $E = \{2n \mid n \in \mathbb{N}\}$ . Consider the function  $f: \mathbb{N} \rightarrow E$  defined by  $f(m) = 2m$ . To verify that  $f$  is one-to-one, assume that two inputs yield the same output  $f(m) = f(\hat{m})$ , giving  $2m = 2\hat{m}$ , and divide by 2 to get that therefore the two inputs are the same,  $m = \hat{m}$ .

To verify that  $f$  is onto, consider a member of the codomain  $y \in E$ , so that  $y = 2n$  for some  $n \in \mathbb{N}$ . Now,  $n = y/2$  has the property that  $f(n) = y$ . Note that  $n$  is an element of the domain  $\mathbb{N}$  because as a member of the codomain,  $y$  is even, and thus dividing by two yields a natural number.

- (B) Write the odd numbers as  $M = \{2n + 1 \mid n \in \mathbb{N}\}$ . Consider  $g: \mathbb{N} \rightarrow M$  given by  $n \mapsto 2n + 1$ . To verify that  $g$  is one-to-one, assume that  $g(m_0) = g(m_1)$ , so that  $2m_0 + 1 = 2m_1 + 1$ , subtract 1, and divide by 2, to conclude that  $m_0 = m_1$ .

To verify that  $g$  is onto, start with an element of the codomain  $y \in M$ . Then it has the form  $y = 2k + 1$  for some natural number  $k$ . That means there is a natural number  $k$  that maps under  $g$  to  $y$ , namely  $k = (y-1)/2$ .

- (C) One answer is to cite the two prior exercise parts and that the inverse of a correspondence is also a correspondence, to get that the composition  $g \circ f^{-1}$  is a correspondence from the even numbers to the odds.

A direct answer is to consider the map  $p: E \rightarrow M$  given by  $p(n) = n + 1$ . Verify that  $p$  is one-to-one and onto as in the prior two items.

**II.1.31** The natural correspondence  $f_0: (0..1) \rightarrow (0..2)$  is  $f_0(x) = 2x$ . For the other two there are many choices but two more correspondences  $f_0, f_1, f_2: (0..1) \rightarrow (0..2)$  are  $f_1(x) = 2 - 2x$  and  $f_2(x) = 2x^2$ . Figure 2 on page 39 shows the three graphs. They are different functions because they return different values on the input  $x = 1/3$ .

Each function is one-to-one and onto because the figure shows that every horizontal line at  $y \in (0..2)$  intercepts its graph in one and only one point.

For an algebraic verification that they are correspondences we can use  $f_1$  as a model. It is one-to-one because if  $f(x_0) = f(x_1)$  then  $2 - 2x_0 = 2 - 2x_1$ , and then subtracting 2 and dividing by  $-2$  yields that  $x_0 = x_1$ . It is onto because if  $y \in (0..2)$  then the number  $x = (y - 2)/(-2)$  is an element of  $(0..1)$  and has the property that  $f(x) = 2 - 2 \cdot ((y - 2)/(-2)) = y$ .

**II.1.32** The function

$$\hat{f}(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if } n = 1 \\ n^2 & \text{if } n > 1 \end{cases}$$

is different than Example 1.8's function  $f$  because they give different outputs for the input 0. That  $\hat{f}$  is onto is clear, as every perfect square is the output associated with some input. That  $\hat{f}$  is a one-to-one function is also

clear, although it is a little messier to write down. Assume  $\hat{f}(x_0) = \hat{f}(x_1)$  and then there are four cases: both  $x_0$  and  $x_1$  are members of the set  $\{0, 1\}$ , only the first one is a member, only the second is a member, or neither is a member. All four cases are easy.

**II.1.33** To check that  $f$  is one-to-one suppose that  $f(x_0) = f(x_1)$ , so that  $c^{x_0} = c^{x_1}$ . Take the logarithm base  $c$  of both sides, giving  $x_0 = x_1$ .

For onto, consider a codomain element  $y \in (0 .. \infty)$ . The number  $x = \log_c(y)$  is defined and is an element of  $\mathbb{R}$ , the domain of  $f$ . Of course,  $f(x) = f(\log_c(y)) = c^{\log_c(y)} = y$ .

**II.1.34** The two  $P_2 = \{2^k \mid k \in \mathbb{N}\} = \{1, 2, 4, 8, \dots\}$  and  $P_3 = \{3^k \mid k \in \mathbb{N}\} = \{1, 3, 9, \dots\}$  are subsets of  $\mathbb{N}$ . The natural correspondence  $g: P_2 \rightarrow P_3$  is to associate elements having the same exponent,  $g(2^k) = 3^k$ . This function is one-to-one because if  $g(2^{k_0}) = g(2^{k_1})$  then  $3^{k_0} = 3^{k_1}$ , and taking the logarithm base 3 of both sides shows that the inputs are the same,  $k_0 = k_1$ . This function is onto because if  $y \in P_3$  then it has the form  $y = 3^k$ , and is the image of  $2^k \in P_2$ .

The obvious generalization is that if two natural numbers  $a, b \in \mathbb{N}$  are greater than 1 then  $P_a = \{a^k \mid k \in \mathbb{N}\}$  and  $P_b = \{b^k \mid k \in \mathbb{N}\}$  are sets of natural numbers that have the same cardinality. The argument is as in the prior paragraph.

**II.1.35** Of course, for each item there are many possible answers.

(A) One is  $f_0: \mathbb{N} \rightarrow \mathbb{N}$  given by  $f_0(n) = n + 1$ . It is one-to-one because if  $f_0(n) = f_0(\hat{n})$  then  $n + 1 = \hat{n} + 1$  and so  $n = \hat{n}$ . It is not onto because no  $n \in \mathbb{N}$  maps to 0.

A second one is  $f_1: \mathbb{N} \rightarrow \mathbb{N}$  given by  $f_1(n) = n^2$ . It is one-to-one because if  $f_1(n) = f_1(\hat{n})$  then  $n^2 = \hat{n}^2$  and because these are natural numbers they are nonnegative, so  $n = \hat{n}$ . This function is not onto because no natural number maps to 2.

(B) This function

$$f_0(x) = \begin{cases} x/2 & \text{-- if } x \text{ is even} \\ (x-1)/2 & \text{-- if } x \text{ is odd} \end{cases} \quad \begin{array}{c|cccccc} x & 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ \hline f_0(x) & 0 & 0 & 1 & 1 & 2 & 2 & \dots \end{array}$$

is onto because every  $y \in \mathbb{N}$  is the image of  $x = 2y$ . It is not one-to-one because  $f_0(1) = f_0(0)$ .

A second one is  $f_1(x) = \lfloor \sqrt{x} \rfloor$  (recall that the floor operation,  $\lfloor n \rfloor$ , gives the largest integer less than or equal to  $n$ ).

$$\begin{array}{c|cccccccccccc} x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \dots \\ \hline f_1(x) & 0 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & \dots \end{array}$$

It is onto because every  $y \in \mathbb{N}$  is the image of  $x = y^2$ . It is not one-to-one because  $f_1(1) = f_1(2)$ .

(C) One such map is  $f_0(x) = 0$ . It is not one-to-one because  $f_0(0) = f_0(1)$ . It is not onto because no natural number input maps to 1.

A non-constant such function is the square of the prior item's second one,  $f_1(x) = (\lfloor \sqrt{x} \rfloor)^2$ .

$$\begin{array}{c|cccccccccccc} x & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \dots \\ \hline f_1(x) & 0 & 1 & 1 & 1 & 4 & 4 & 4 & 4 & 4 & 9 & 9 & \dots \end{array}$$

It is not one-to-one because  $f_1(1) = f_1(2)$ . It is not onto because no number maps to 2, as 2 is not a square.

(D) The natural function that is both one-to-one and onto is the identity function,  $f_0(x) = x$ . It is one-to-one because if  $f_0(n) = f_0(\hat{n})$  then  $n = \hat{n}$ , just straight from the definition of  $f_0$ . It is onto because any codomain element  $y \in \mathbb{N}$  is the image under  $f_0$  of itself,  $f_0(y) = y$ .

Another correspondence swaps even and odd numbers.

$$f_1(x) = \begin{cases} x+1 & \text{-- if } x \text{ is even} \\ x-1 & \text{-- if } x \text{ is odd} \end{cases} \quad \begin{array}{c|cccccc} x & 0 & 1 & 2 & 3 & 4 & 5 & \dots \\ \hline f_1(x) & 1 & 0 & 3 & 2 & 5 & 4 & \dots \end{array}$$

Observe first that  $f_1: \mathbb{N} \rightarrow \mathbb{N}$ ; that is, if  $x \in \mathbb{N}$  then  $f_1(x) \in \mathbb{N}$ . To check that  $f_1$  is one-to-one suppose that  $f_1(n) = f_1(\hat{n})$ . There are two cases. If  $n$  is odd then  $f_1(n) = n - 1$  is even, and so  $f_1(\hat{n})$  is even also, implying that  $\hat{n}$  is odd, and therefore  $n - 1 = \hat{n} - 1$ , giving that  $n = \hat{n}$ . The even case is similar. To check

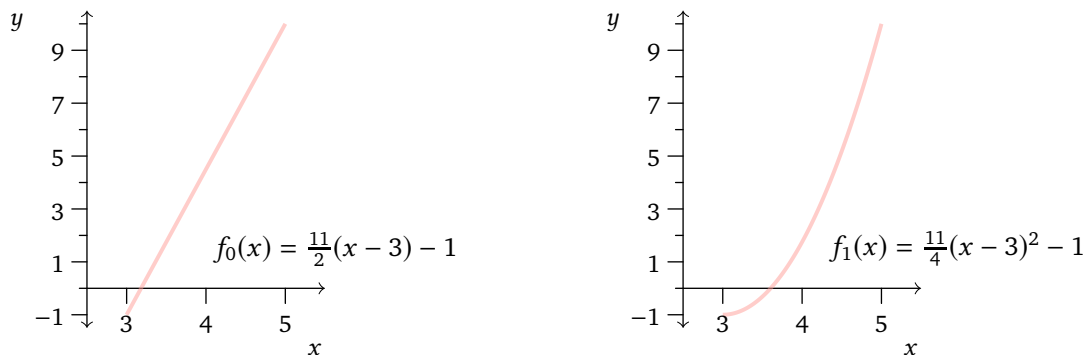


FIGURE 3, FOR QUESTION II.1.36: Two correspondences between  $(3..5)$  and  $(-1..11)$ .

that  $f_1$  is onto consider a codomain point  $y \in \mathbb{N}$ . Here also there are two cases and we will only do the odd one, so suppose that  $y$  is odd. Then  $y = f_1(x)$  where  $x = y - 1$ , and  $x$  is an element of the domain  $\mathbb{N}$  because if  $y$  is odd then  $y - 1 \geq 0$ .

**II.1.36** Let  $D = (3..5)$  and  $C = (-1..10)$ .

One correspondence is  $f_0(x) = (11/2)(x-3) - 1$ . Whether  $f_0: D \rightarrow C$  is not obvious so we start with that: if  $3 < x < 5$  then subtracting 3, multiplying by  $11/2$ , and subtracting 1 from all terms gives  $-1 < f_0(x) < 10$ . Next we show that this map is onto. If  $y \in C$  then  $y = f_0(x)$  where  $x = (2/11)(y + 1) + 3$ . To verify that this input  $x$  is an element of the domain  $D$ , start with  $-1 < y < 10$ , add 1 to all terms, multiply by  $2/11$ , and add 3, ending with  $3 < x < 5$ . Finally, we show that the map  $f_0$  is one-to-one. If  $f_0(x) = f_0(\hat{x})$  then  $(11/2)(x - 3) - 1 = (11/2)(\hat{x} - 3) - 1$  and adding 1 to both sides, multiplying by  $2/11$ , and finally adding 3, gives that  $x = \hat{x}$ .

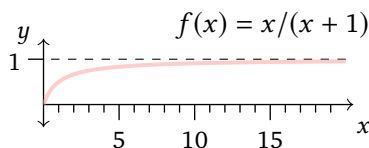
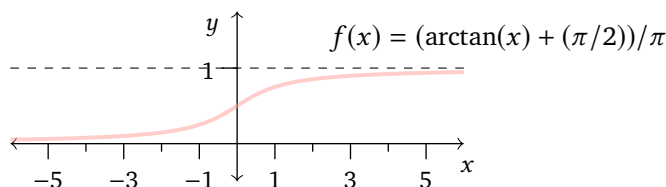
Another correspondence is  $f_1(x) = (11/4)(x - 3)^2 - 1$ . As with  $f_0$  we start by verifying that  $f_1: D \rightarrow C$ : if  $3 < x < 5$  then subtracting 3, squaring, multiplying by  $11/4$ , and subtracting 1 from all terms gives  $-1 < f_1(x) < 10$ . To show that  $f_1$  is onto assume that  $y \in C$ . Then  $y = f_1(x)$  where  $x = \sqrt{(4/11)(y + 1)} + 3$ . To check that  $x \in D$ , start with  $-1 < y < 10$  and add 1 to all terms, multiply by  $4/11$ , square (this doesn't change the direction of the  $<$ 's because all numbers are nonnegative), and add 3, ending with  $3 < x < 5$ . To show that this map is one-to-one, assume  $f_1(x) = f_1(\hat{x})$  so that  $(11/4)(x - 3)^2 - 1 = (11/4)(\hat{x} - 3)^2 - 1$ . To both sides add 1, multiply by  $4/11$ , take the square root (which is unambiguous because the  $x$ 's are all greater than zero) and then add 3, giving that  $x = \hat{x}$ .

Figure 3 on page 41 shows that the two functions are unequal. It also gives an alternative demonstration that each is a correspondence because for each, every horizontal line at  $y \in (-1..10)$  intersects the graph one and only one point.

**II.1.37** In each case we produce a function from one set to the other and verify that it is both one-to-one and onto. For the third item, note that just because a set appears first in the question does not mean that it must be the domain of the correspondence.

- (A) Where  $S_4 = \{4k \mid k \in \mathbb{N}\}$  and  $S_5 = \{5k \mid k \in \mathbb{N}\}$ , let  $f: S_4 \rightarrow S_5$  be  $f(x) = (5/4)x$ . It is one-to-one because  $f(x_0) = f(x_1)$  implies that  $(5/4)x_0 = (5/4)x_1$ , so  $x_0 = x_1$ . It is onto because if we consider a codomain element  $y = 5k$  for some  $k \in \mathbb{N}$  then clearly  $y = f(4k)$ .
- (B) These two sets are the same set, that is, they contain the same numbers. They correspond via the identity function.
- (C) Write  $T = \{0, 1, 3, 6, \dots\} = \{n(n+1)/2 \mid n \in \mathbb{N}\}$  (these are called the triangular numbers). Consider the function  $f: \mathbb{N} \rightarrow T$  defined by  $f(n) = n(n+1)/2$ . This map is one-to-one because the continuous function  $\hat{f}: \mathbb{R} \rightarrow \mathbb{R}$  given by  $\hat{f}(x) = x(x+1)/2$  is a parabola with roots at 0 and  $-1$  and so the part in the positive numbers, with  $x \geq 0$ , has that a horizontal line at  $y$  will intersect the graph in at most one point. The fact that  $\hat{f}$  is one-to-one gives that its restriction  $f$  is also one-to-one. The function  $f$  is onto by definition, since  $T$  is defined as the range of that function.

**II.1.38**

FIGURE 4, FOR QUESTION II.1.38: A correspondence between  $(0.. \infty)$  and  $(0.. 1)$ .FIGURE 5, FOR QUESTION II.1.41: A correspondence between  $\mathbb{R}$  and  $(0.. 1)$ .

- (A) One is  $f(x) = 2x$ . This map is one-to-one because if  $f(x) = f(\hat{x})$  then  $2x = 2\hat{x}$  and so  $x = \hat{x}$ . This map is onto because where  $0 < y < 2$ , the number  $x = y/2$  has the properties that  $f(x) = y$  and  $0 < x < 1$ .
- (B) One is  $f(x) = (b - a) \cdot x + a$ . Note that  $f: (0.. 1) \rightarrow (a.. b)$  since if  $0 < x < 1$  then multiplying by  $b - a$  gives  $0 < x < (b - a)$  (the  $<$ 's maintain their direction because  $a < b$ ), and adding  $a$  gives  $a < x < b$ . Further, this function is one-to-one: because  $b - a \neq 0$  this line does not have slope 0 and so the function passes the horizontal line test (for an algebraic argument, start with  $f(x) = f(\hat{x})$ , subtract  $a$  and divide by  $b - a$ , and conclude that  $x = \hat{x}$ ). Finally, to show that this function is onto, fix  $y \in (a.. b)$ . Note that  $f(x) = y$  where  $x = (y - a)/(b - a)$ . What remains is to check that  $x \in (0.. 1)$ : starting from  $a < y < b$ , subtract  $a$  and divide by  $b - a$  to get  $0 < (y - a)/(b - a) < 1$  (as earlier, note that the  $<$ 's maintain their direction because  $b - a > 0$ ).
- (C) One is  $f(x) = x + a$ . This function is one-to-one because  $f(x) = f(\hat{x})$  implies that  $x + a = \hat{x} + a$  and so  $x = \hat{x}$ . It is onto because if  $y \in (a.. \infty)$  then  $y = f(x)$  where  $x = y - a$  (and  $x \in (0.. \infty)$ ).
- (D) Similar triangles gives  $x/(x + 1) = f(x)/1$ . Figure 4 on page 42 shows that it is a correspondence because for each  $y \in (0.. 1)$ , the horizontal line at that height intercepts the function's graph in exactly one point.
- II.1.39** Consider the function  $f: \mathbb{N} \rightarrow S$  given by  $f(m) = m + \sqrt[2]{2} = 2^{1/(m+2)}$ . It is one-to-one because if  $f(m) = f(\hat{m})$  then  $2^{1/(m+2)} = 2^{1/(\hat{m}+2)}$ . Take the logarithm of both sides  $\lg(2^{1/(m+2)}) = \lg(2^{1/(\hat{m}+2)})$  to get  $(1/(m + 2)) \cdot \lg(2) = (1/(\hat{m} + 2)) \cdot \lg(2)$ , and so  $1/(m + 2) = 1/(\hat{m} + 2)$ . Thus  $m = \hat{m}$ .
- This function is onto because if  $y \in S$  then  $y = 2^{1/n}$  for some  $n \in \mathbb{N}$  with  $n \geq 2$ . Thus,  $y = 2^{1/(x+2)} = f(x)$  for  $x \in \mathbb{N}$ .
- II.1.40** Recall that just as every natural number has a finite decimal representation, and that this representation is unique, so also every natural number has a unique binary representation.
- (A) View each string of bits  $\sigma$  starting with 1 as the binary representation of a natural number  $n$ . Then  $f: B \rightarrow \mathbb{N}$  given by  $\sigma \mapsto n - 1$  is clearly a correspondence.
- (B) Prefix each string  $\tau \in \mathbb{B}^*$  with 1, so  $\sigma = 1 \hat{\ } \tau$ . Now the prior argument applies.
- II.1.41** The tangent function is a correspondence from  $(-\pi/2.. \pi/2)$  to  $\mathbb{R}$  so its inverse, arctangent, is a correspondence in the other direction. Thus,  $g(x) = (\arctan(x) + (\pi/2))/\pi$  gives a correspondence from  $\mathbb{R}$  to  $(0.. 1)$ . See Figure 5 on page 42, which shows that  $g$  is one-to-one and onto because for each element  $y$  of the codomain  $(0.. 1)$ , the horizontal line at  $y$  intercepts the function's graph in exactly one point.
- II.1.42**
- (A) The map  $g: I_0 \rightarrow I_1$  given by  $g(x) = x \cdot (r_1/r_0)$  is onto because if  $y \in I_1 = [0.. 2\pi r_1)$  then  $y = g(x)$  where  $x = y \cdot (r_0/r_1)$ , and  $0 \leq y < 2\pi r_1$  implies that  $0 \leq y \cdot (r_0/r_1) < 2\pi r_0$ . To show that this map is one-to-one suppose  $g(x) = g(\hat{x})$ . Then  $x \cdot (r_1/r_0) = \hat{x} \cdot (r_1/r_0)$  and so  $x = \hat{x}$ .
- (B) The map  $g(x) = a + x \cdot (b - a)$  is a correspondence. To verify that it is one-to-one, suppose that  $g(x) = g(\hat{x})$  so that  $a + x \cdot (b - a) = a + \hat{x} \cdot (b - a)$ . Subtract  $a$  and divide by  $b - a$  to conclude that  $x = \hat{x}$ . This map is also

onto because  $y \in [a..b]$  is the image of  $x = (y - a)/(b - a)$ , and  $a \leq y < b$  implies that  $0 \leq y - a < b - a$ , and so  $0 \leq (y - a)/(b - a) < 1$ .

- (c) By the prior item there are correspondences  $f: [0..1) \rightarrow [a..b]$  and  $g: [0..1) \rightarrow [c..d]$ . Then the composition  $g \circ f^{-1}$  has domain  $[a..b)$ , codomain  $[c..d)$ , and is a correspondence.

(This is also easy to check by considering the map  $f: [a..b) \rightarrow [c..d)$  given by  $f(x) = c + (x - a) \cdot (d - c)/(b - a)$ .)

**II.1.43** Suppose for contradiction that  $f$  is not one-to-one. Then there are  $x, \hat{x} \in D$  such that  $f(x) = f(\hat{x})$  but  $x \neq \hat{x}$ . But one or the other of these two is smaller; without loss of generality suppose that  $x < \hat{x}$ . Then  $x < \hat{x}$  but  $f(x) = f(\hat{x})$ , which contradicts that the function is strictly increasing.

Yes, the same argument works for  $D \subseteq \mathbb{N}$ .

**II.1.44**

- (A) Let  $\mathcal{E} = \{2n \mid n \in \mathbb{N}\}$ . The natural correspondence is  $f: \mathbb{N} \rightarrow \mathcal{E}$  given by  $n \mapsto 2n$ . This map is one-to-one because if  $f(n) = f(\hat{n})$  then  $2n = 2\hat{n}$  and so  $n = \hat{n}$ . This map is onto because if  $y \in \mathcal{E}$  then  $y = 2k$  for some  $k \in \mathbb{N}$  and setting  $n = k$  gives that  $y = f(n)$ .
- (B) Let  $T = \{n \in \mathbb{N} \mid n > 4\}$ . Consider  $f: \mathbb{N} \rightarrow T$  given by  $n \mapsto n + 5$ . This is one-to-one because if  $f(n) = f(\hat{n})$  then  $n + 5 = \hat{n} + 5$  and  $n = \hat{n}$ . It is onto because if  $y \in T$  then  $y = f(x)$  where  $x = y - 5$ , and observe that  $y \in T$  implies that  $y - 5 \in \mathbb{N}$ . So  $f$  is a correspondence.

**II.1.45** We will show that no matter what is the cardinality of the finite set  $S \subset \mathbb{N}$ , there is a correspondence  $f: \mathbb{N} \rightarrow \mathbb{N} - S$ . We will use induction, at each step taking the finite set to have one element more than at the prior step.

Before the proof we give an illustration. Suppose that we have a five-element set  $S_5 = \{2, 5, 8, 10, 11\}$  and this correspondence  $f_5: \mathbb{N} \rightarrow \mathbb{N} - S_5$ .

$$f_5(n) \begin{array}{c|cccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \\ \hline & 0 & 1 & 3 & 4 & 6 & 7 & 9 & 12 & \dots \end{array}$$

Now we add a sixth element, so let  $S = S_5 \cup \{9\}$ . We want a correspondence  $f: \mathbb{N} \rightarrow \mathbb{N} - S$ . The natural approach is find that  $9 = f_5(6)$  and define  $f$  in this way.

$$f(n) = \begin{cases} f_5(n) & \text{-- if } n < 6 \\ f_5(n + 1) & \text{-- if } n > 6 \end{cases} \quad f(n) \begin{array}{c|cccccccc} n & 0 & 1 & 2 & 3 & 4 & 5 & 6 & \dots \\ \hline & 0 & 1 & 3 & 4 & 6 & 7 & 12 & \dots \end{array}$$

The induction argument's base case is  $S = \emptyset$  and here the identity function  $f(n) = n$  is clearly a one-to-one and onto map from  $\mathbb{N}$  to  $\mathbb{N} - S$ .

For the inductive step take  $k \in \mathbb{N}$ , assume that the statement is true for any subset of  $\mathbb{N}$  whose cardinality is less than or equal to  $k$ , and consider  $S \subset \mathbb{N}$  with  $|S| = k + 1$ . Fix any  $s \in S$  and let  $S_k = S - \{s\}$ . By the inductive hypothesis there is a correspondence  $f_k: \mathbb{N} \rightarrow \mathbb{N} - S_k$ . Set  $\hat{n}$  to be such that  $f_k(\hat{n}) = s$  and define  $f: \mathbb{N} \rightarrow \mathbb{N} - S$  in this way.

$$f(n) = \begin{cases} f_k(n) & \text{-- if } n < \hat{n} \\ f_k(n + 1) & \text{-- if } n > \hat{n} \end{cases}$$

The function  $f: \mathbb{N} \rightarrow \mathbb{N} - S$  is one-to-one because it is a restriction of  $f_k$ , which is a one-to-one map from  $\mathbb{N}$  to  $\mathbb{N} - \hat{S}$ . Similarly,  $f$  is onto because  $f_k$  is onto, from  $\mathbb{N}$  to  $\mathbb{N} - S_k$ . Thus  $f$  is a correspondence between  $\mathbb{N}$  and  $\mathbb{N} - S$ , and so the two have the same cardinality.

**II.1.46**

- (A) The inverse function  $f^{-1}: C \rightarrow D$  is given by  $f^{-1}(\text{Spades}) = 0$ ,  $f^{-1}(\text{Hearts}) = 1$ ,  $f^{-1}(\text{Clubs}) = 2$ , and  $f^{-1}(\text{Diamonds}) = 3$ . By inspection it is both one-to-one and onto.
- (B) Suppose that  $f: D \rightarrow C$  is a correspondence. Consider the binary relation, the set of ordered pairs,  $f^{-1} = \{\langle y, x \rangle \mid f(x) = y\}$ . We will show that  $f^{-1}$  is a function, which means that we will show that it is well-defined, that each input  $y \in C$  is associated with one and only one output  $x \in D$ .

Each  $y \in C$  is associated with at least one  $x$  because the function  $f$  is onto. Each  $y \in C$  is associated with at most one  $x$  because  $f$  is one-to-one.

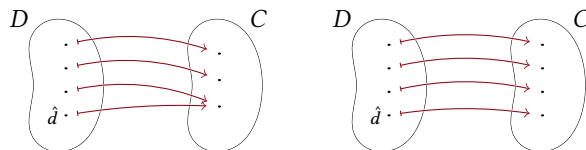


FIGURE 6, FOR QUESTION II.1.48: A finite function's domain has at least as many elements as its range.

(c) To show that  $f^{-1} : C \rightarrow D$  is one-to-one suppose that  $f^{-1}(y) = f^{-1}(\hat{y})$ . Then there is a  $x \in D$  such that  $f(x) = y$  and  $f(x) = \hat{y}$ . This contradicts that  $f$  is a function, because a function must be well-defined.

To show that  $f^{-1}$  is onto consider a member of its codomain,  $\hat{x} \in D$ . By the definition of a function there is a  $\hat{y} \in C$  with  $f(\hat{x}) = \hat{y}$ , and then  $\hat{x} = f^{-1}(\hat{y})$ .

**II.1.47** One direction is straightforward: no finite set has the same cardinality as a proper subset of itself, by Lemma 1.5.

For the other direction, fix an infinite set  $S$ , to show that it has the same cardinality as a proper subset of itself. (*Remark:* as elsewhere, we feel free here to use the Axiom of Choice.) Choose any element  $s_0 \in S$ . Then  $S - \{s_0\}$  is not empty (or else  $S$  is not infinite) so choose  $s_1 \in S - \{s_0\}$ . Continuing in this way gives  $\{s_0, s_1, s_2, \dots\}$ . Then this is a correspondence from  $S$  to  $S - \{s_0\}$ .

$$f(x) = \begin{cases} s_{i+1} & \text{if } x = s_i \text{ for some } i \in \mathbb{N} \\ x & \text{otherwise} \end{cases}$$

**II.1.48** Denote the function  $f : D \rightarrow C$ , and assume that  $D$  is finite.

(A) We will argue that the statement  $|\text{dom}(f)| \geq |\text{ran}(f)|$  holds for all functions on finite domains by induction on the number of elements in  $D$ .

The base step is that the domain has no elements at all,  $\text{dom}(f) = \emptyset$ . The only function with an empty domain is the empty function. The empty function's range is empty,  $\text{ran}(f) = \emptyset$ , and thus  $|\text{dom}(f)| \geq |\text{ran}(f)|$ .

For the inductive step, fix  $k \in \mathbb{N}$ , assume that the statement is true for any function with a domain having 0 elements, or 1 elements,  $\dots$  or  $k$  elements, and consider the  $|D| = k + 1$  case.

Because  $k + 1 > 0$  there is a  $\hat{d} \in D$ . Let  $\hat{D} = D - \{\hat{d}\}$  and consider the restriction  $f|_{\hat{D}} : \hat{D} \rightarrow C$ . Its domain has  $k$  many elements, so the induction hypothesis applies, giving  $|\text{dom}(f|_{\hat{D}})| \geq |\text{ran}(f|_{\hat{D}})|$ .

To add back the element  $\hat{d}$  there are two cases; see Figure 6 on page 44. The first case, on the left in the figure, is that the range of  $f$  is the same as the range of the restriction  $f|_{\hat{D}}$ . The other case is that the range of  $f$  has one additional element. In either case, adding 1 to both sides ends the argument.

$$|\text{dom}(f)| = |\text{dom}(f|_{\hat{D}})| + 1 \geq |\text{ran}(f|_{\hat{D}})| + 1 \geq |\text{ran}(f)|$$

(B) Assume that the function is one-to-one. We will show that the domain and range have the same size by induction on the number of elements in the domain. This argument is like the prior item's except that it has only the single case shown on the right of Figure 6 on page 44.

The base step is that the function's domain has no elements at all,  $\text{dom}(f) = \emptyset$ . This function's range is empty and  $|\text{dom}(f)| = |\text{ran}(f)|$ .

For the inductive step, fix  $k \in \mathbb{N}$ , assume that the domain and range have the same size for any one-to-one function where the domain  $D$  has 0 elements, or 1 element,  $\dots$  or  $k$  elements, and suppose that  $|D| = k + 1$ .

Because  $k + 1 > 0$  there is a  $\hat{d} \in D$ . Let  $\hat{D} = D - \{\hat{d}\}$ . Consider the restriction function  $f|_{\hat{D}} : \hat{D} \rightarrow C$ . The function  $f$  is one-to-one so the restriction is also one-to-one. The induction hypothesis applies, giving  $|\text{dom}(f|_{\hat{D}})| = |\text{ran}(f|_{\hat{D}})|$ . Adding 1 to both sides gives  $|\text{dom}(f)| = |\text{dom}(f|_{\hat{D}})| + 1 = |\text{ran}(f|_{\hat{D}})| + 1 = |\text{ran}(f)|$ .

(c) Figure 7 on page 45 illustrates the argument.

For the proof, let  $f$  be a function with a finite domain that is not one-to-one. Fix some numbering of its

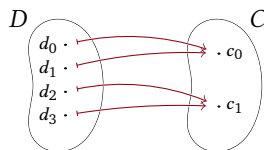


FIGURE 7, FOR QUESTION II.1.48: A finite function that is not one-to-one has more elements in its domain than its range.

domain,  $D = \{d_0, d_1, \dots, d_n\}$ . Consider

$$\hat{D} = \{d_i \in D \mid i \text{ is minimal among the indices } j \text{ where } f(d_j) = f(d_i)\}$$

(in the illustration,  $f(d_0) = f(d_1)$  and  $f(d_2) = f(d_3)$ ) so this is the set  $\{d_0, d_2\}$ .

Because  $f$  is not one-to-one,  $\hat{D}$  is a proper subset of  $D$ . The restriction  $f \upharpoonright_{\hat{D}}: \hat{D} \rightarrow C$  is one-to-one. By the prior item, its range has the same number of elements as its domain. The range of  $f$  is the same as the range of  $f \upharpoonright_{\hat{D}}$ . But the domain of  $f$  has more elements than the domain of  $f \upharpoonright_{\hat{D}}$ . Thus the domain of  $f$  contains more elements than its range.

- (D) Let  $D$  and  $C$  be finite sets. If there is a correspondence  $f: D \rightarrow C$  then the prior items show that the sets have the same number of elements.

For the other direction assume that the two have the same number of elements. Fix an ordering of each,  $D = \{d_0, \dots, d_k\}$  and  $C = \{c_0, \dots, c_k\}$ . Then the map  $f: D \rightarrow C$  defined by  $f(d_i) = c_i$  is clearly a correspondence.

**II.2.17** The table alternates between column 0 and column 1.

$n \in \mathbb{N}$	6	7	8	9	10	11	12
$f(n) \in \{0, 1\} \times \mathbb{N}$	$\langle 0, 3 \rangle$	$\langle 1, 3 \rangle$	$\langle 0, 4 \rangle$	$\langle 1, 4 \rangle$	$\langle 0, 5 \rangle$	$\langle 1, 5 \rangle$	$\langle 0, 6 \rangle$

The formulas are  $x(n) = (1 + (-1)^{n+1})/2$  and  $y(n) = \lfloor n/2 \rfloor$ .

**II.2.18**

- (A) The pair before  $\langle 50, 50 \rangle$  is  $\langle 49, 51 \rangle$  and the pair after is  $\langle 51, 49 \rangle$ . As a check they correspond to  $\text{cantor}(50, 50) = 5\,100$ ,  $\text{cantor}(49, 51) = 5\,099$ , and  $\text{cantor}(51, 49) = 5\,101$ .
- (B) The pair before  $\langle 100, 4 \rangle$  is  $\langle 99, 5 \rangle$  and the pair after is  $\langle 101, 3 \rangle$ . As a check, the three correspond to  $\text{cantor}(100, 4) = 5\,560$ ,  $\text{cantor}(99, 5) = 5\,559$ , and  $\text{cantor}(101, 3) = 5\,561$ .
- (C) The pair before  $\langle 4, 100 \rangle$  is  $\langle 3, 101 \rangle$  and the pair after is  $\langle 5, 99 \rangle$ . The three give  $\text{cantor}(100, 4) = 5\,465$ ,  $\text{cantor}(99, 5) = 5\,463$ , and  $\text{cantor}(5, 99) = 5\,465$ .
- (D) Before  $\langle 0, 200 \rangle$  is  $\langle 199, 0 \rangle$  while after is  $\langle 1, 199 \rangle$ . These three correspond to  $\text{cantor}(0, 200) = 20\,100$ ,  $\text{cantor}(199, 0) = 20\,099$ , and  $\text{cantor}(1, 199) = 20\,101$ .
- (E) Before  $\langle 200, 0 \rangle$  is  $\langle 199, 1 \rangle$  while after is  $\langle 0, 201 \rangle$ . They correspond to  $\text{cantor}(200, 0) = 20\,300$ ,  $\text{cantor}(199, 1) = 20\,299$ , and  $\text{cantor}(0, 201) = 20\,301$ .

**II.2.19**

- (A) The correspondence  $f_T: \mathbb{N} \rightarrow T$  is given by  $f_T(n) = 2n$ . The correspondence  $f_F: \mathbb{N} \rightarrow T$  is given by  $f_F(m) = 5m$ . Checking that each is one-to-one and onto is easy.

$n \in \mathbb{N}$	0	1	2	3	4	5	6	7	8	9
$f_T(n) \in T$	0	2	4	6	8	10	12	14	16	18
$f_F(n) \in F$	0	5	10	15	20	25	30	35	40	45

- (B) This correspondence gives outputs in ascending order.

$n \in \mathbb{N}$	0	1	2	3	4	5	6	7	8	9
$f(n) \in T \cup F$	0	2	4	5	6	8	10	12	14	15

**II.2.20** This gives the first few associations for a function  $f: \mathbb{N} \rightarrow \mathbb{N} \times \{0, 1\}$ .

$n$	0	1	2	3	4	5	...
$f(n)$	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 1, 0 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 0 \rangle$	$\langle 2, 1 \rangle$	...

```
(define (nextpair p)
  (let ((x (first p))
        (y (second p)))
    (if (= y 0)
        (list 0 (+ x 1))
        (list (+ x 1) (- y 1)))))
```

FIGURE 8, FOR QUESTION II.2.24: Racket code that inputs a pair, as a list, and outputs the next one.

From that table a description sufficient for computing  $f$  is easy.

$$f(n) = \begin{cases} \langle \lfloor n/2 \rfloor, 0 \rangle & \text{if } n \text{ is even} \\ \langle \lfloor n/2 \rfloor, 1 \rangle & \text{if } n \text{ is odd} \end{cases}$$

That gives  $f(0) = \langle 0, 0 \rangle$ ,  $f(10) = \langle 5, 0 \rangle$ ,  $f(100) = \langle 50, 0 \rangle$ , and  $f(101) = \langle 50, 1 \rangle$ .

To go from the pair to the number we need the inverse,  $f^{-1}(i, j) = 2i + j$ . Then  $f^{-1}(2, 1) = 5$ ,  $f^{-1}(20, 1) = 41$ , and  $f^{-1}(200, 1) = 401$ .

**II.2.21** This gives the first few of the associations.

Number	0	1	2	3	4	5	6	7	8	...
Pair	$\langle 0, 0 \rangle$	$\langle 1, 0 \rangle$	$\langle 2, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 1 \rangle$	$\langle 0, 2 \rangle$	$\langle 1, 2 \rangle$	$\langle 2, 2 \rangle$	...

The formula for the top to bottom association is

$$f(n) = \begin{cases} \langle 0, \lfloor n/3 \rfloor \rangle & \text{if } 3 \text{ divides } n \\ \langle 1, \lfloor n/3 \rfloor \rangle & \text{if } n \bmod 3 \text{ is } 1 \\ \langle 2, \lfloor n/3 \rfloor \rangle & \text{if } n \bmod 3 \text{ is } 2 \end{cases}$$

(in short:  $f(n) = \langle n \bmod 3, \lfloor n/3 \rfloor \rangle$ ). Thus  $f(0) = \langle 0, 0 \rangle$ ,  $f(10) = \langle 1, 3 \rangle$ ,  $f(100) = \langle 1, 33 \rangle$ , and  $f(1000) = \langle 1, 333 \rangle$ . The formula for the bottom to top function  $f^{-1}: \{0, 1, 2\} \times \mathbb{N} \rightarrow \mathbb{N}$  is also easy,  $f^{-1}(x, y) = x + 3y$ . With that,  $f^{-1}(1, 2) = 7$ ,  $f^{-1}(1, 20) = 61$ , and  $f^{-1}(1, 200) = 601$ .

**II.2.22** This is the initial part of an enumeration  $f$  of  $\{0, 1, 2, 3\} \times \mathbb{N}$ .

Number	0	1	2	3	4	5	6	7	8	...
Pair	$\langle 0, 0 \rangle$	$\langle 1, 0 \rangle$	$\langle 2, 0 \rangle$	$\langle 3, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 1 \rangle$	$\langle 3, 1 \rangle$	$\langle 0, 2 \rangle$	...

The formula for the top to bottom association is  $f(n) = \langle n \bmod 4, \lfloor n/4 \rfloor \rangle$ , so  $x(n) = n \bmod 4$  and  $y(n) = \lfloor n/4 \rfloor$ .

In general, for  $k \in \mathbb{N}$  an enumeration  $f$  of  $\{0, 1, 2, \dots, k\} \times \mathbb{N}$  is  $f(n) = \langle n \bmod (k + 1), \lfloor n/(k + 1) \rfloor \rangle$ .

**II.2.23** Example 2.4's table shows up to  $n = 6$ .

Number	7	8	9	10	11	12	13	14	15	16
Pair	$\langle 1, 2 \rangle$	$\langle 2, 1 \rangle$	$\langle 3, 0 \rangle$	$\langle 0, 4 \rangle$	$\langle 1, 3 \rangle$	$\langle 2, 2 \rangle$	$\langle 3, 1 \rangle$	$\langle 4, 0 \rangle$	$\langle 0, 5 \rangle$	$\langle 1, 4 \rangle$

**II.2.24** One way to compute this function is by brute force. Given  $n$ , a program can generate the pairs in ascending order  $\langle 0, 0 \rangle$ ,  $\langle 0, 1 \rangle$ ,  $\langle 1, 0 \rangle$ ,  $\langle 0, 2 \rangle$ ,  $\dots$ , keeping count, until it generates the matching one. Figure 8 on page 46 has a routine that from an input pair, gives the next one.

**II.2.25** Corollary 2.12 gives that, because the set  $A - B$  is a subset of  $A$ , it is countable. Similarly  $B - A \subseteq B$  is countable. Then, by the same result, their union is also countable.

**II.2.26** Where  $S = \{a + bi \mid a, b \in \mathbb{Z}\}$ , obviously the map  $g: \mathbb{Z} \times \mathbb{Z} \rightarrow S$  given by  $g(a, b) = a + bi$  is a correspondence. Lemma 2.8 says that  $\mathbb{Z} \times \mathbb{Z}$  is countable so there is a correspondence  $f: \mathbb{N} \rightarrow \mathbb{Z} \times \mathbb{Z}$ . Then the composition  $g \circ f: \mathbb{N} \rightarrow S$  is a correspondence.

**II.2.27** Below are the pairs in ascending order along with the resulting associated rational number.

$n$	pairs	$f(n) \in \mathbb{Q}$
0	$\langle 0, 0 \rangle, \langle 0, 1 \rangle$	0
1	$\langle 1, 0 \rangle, \langle 0, 2 \rangle, \langle 1, 1 \rangle$	1
2	$\langle 2, 0 \rangle, \langle 0, 3 \rangle, \langle 1, 2 \rangle$	1/2
3	$\langle 2, 1 \rangle$	2
4	$\langle 3, 0 \rangle, \langle 0, 4 \rangle, \langle 1, 3 \rangle$	1/3
5	$\langle 2, 2 \rangle, \langle 3, 1 \rangle$	3
6	$\langle 4, 0 \rangle, \langle 0, 5 \rangle, \langle 1, 4 \rangle$	1/4
7	$\langle 2, 3 \rangle$	2/3
8	$\langle 3, 2 \rangle$	3/2
9	$\langle 4, 1 \rangle$	4
10	$\langle 5, 0 \rangle, \langle 0, 6 \rangle, \langle 1, 5 \rangle$	1/5
11	$\langle 2, 4 \rangle, \langle 3, 3 \rangle, \langle 4, 2 \rangle, \langle 5, 1 \rangle$	5

**II.2.28**

- (A) The map  $f: \mathbb{Z}^{n+1} \rightarrow \mathbb{Z}_n[x]$  given by  $f(a_0, \dots, a_n) = a_n x^n + \dots + a_1 x + a_0$  is clearly a correspondence. Since Corollary 2.9 says that there is a correspondence  $g: \mathbb{N} \rightarrow \mathbb{Z}^{n+1}$ , we have a correspondence  $f \circ g$  from  $\mathbb{N}$  to  $\mathbb{Z}_n[x]$ .
- (B) The set  $\mathbb{Z}[x]$  is the union of the  $\mathbb{Z}_i[x]$  over all  $i \in \mathbb{N}$ , and the union of countably many countable sets is countable.

**II.2.29** We will use that the function cantor:  $\mathbb{N}^2 \rightarrow \mathbb{N}$  is a correspondence.

To verify that cantor<sub>3</sub> is onto, fix a codomain element  $n \in \mathbb{N}$ . Because cantor is onto there is a pair  $\langle w, z \rangle \in \mathbb{N}^2$  such that  $\text{cantor}(w, z) = n$ . Also because cantor is onto, there is a pair  $\langle x, y \rangle \in \mathbb{N}^2$  so that  $\text{cantor}(x, y) = w$ . Then  $\text{cantor}_3(x, y, z) = \text{cantor}(\text{cantor}(x, y), z) = n$ .

What's left is to verify that cantor<sub>3</sub> is one-to-one. Suppose that  $\text{cantor}_3(a_0, b_0, c_0) = \text{cantor}_3(a_1, b_1, c_1)$ . Then  $\text{cantor}(\text{cantor}(a_0, b_0), c_0) = \text{cantor}(\text{cantor}(a_1, b_1), c_1)$ . The function cantor is one-to-one so the prior sentence implies that  $\text{cantor}(a_0, b_0) = \text{cantor}(a_1, b_1)$  and  $c_0 = c_1$ . Again applying that cantor is one-to-one gives that  $a_0 = a_1$ ,  $b_0 = b_1$ , along with the  $c_0 = c_1$ .

**II.2.30** A way to be confident about these is to write a small script.

- (A)  $c_3(0, 0, 0) = 0$ ,  $c_3(1, 2, 3) = 172$ ,  $c_3(3, 3, 3) = 381$   
 (B)  $c_3(0, 0, 0) = 0$ ,  $c_3(0, 0, 1) = 1$ ,  $c_3(1, 0, 0) = 2$ ,  $c_3(0, 1, 0) = 3$ ,  $c_3(1, 0, 1) = 4$   
 (C)  $c_3(x, y, z) = x + [(y + z + 1)(y + z) + 2x + 2y + 2] \cdot [(y + z + 1)(y + z) + 2x + 2y] / 8$

**II.2.31** Plug  $x = y = i$  into  $\text{cantor}(x, y) = x + (x + y)(x + y + 1) / 2$  to get  $\text{cantor}(i, i) = i + (2i)(2i + 1) / 2 = 2i^2 + 2i$ . That's  $2i(i + 1)$ . It is obviously a multiple of 2 but because the numbers  $i$  and  $i + 1$  are consecutive, one of them contributes another factor of 2. Hence the entire expression is a multiple of 4.

**II.2.32** Let  $S_0, S_1$ , and  $S_2$  be countably infinite and suppose that  $f_0: \mathbb{N} \rightarrow S_0$ ,  $f_1: \mathbb{N} \rightarrow S_1$  and  $f_2: \mathbb{N} \rightarrow S_2$  are onto. Then  $\hat{f}: \mathbb{N} \rightarrow S_0 \cup S_1 \cup S_2$  given by

$$\hat{f}(n) = \begin{cases} f_0(n/3) & \text{-- if } n \text{ is a multiple of 3, that is, } n \bmod 3 = 0 \\ f_1((n-1)/3) & \text{-- if } n \text{ leaves a remainder of 1 on division by 3, so } n \bmod 3 = 1 \\ f_2((n-2)/3) & \text{-- if } n \text{ leaves a remainder of 2 on division by 3, so } n \bmod 3 = 2 \end{cases}$$

(briefly,  $\hat{f}(n) = f_{n \bmod 3}(\lfloor n/3 \rfloor)$ ) and illustrated by

$n$	0	1	2	3	4	5	6	...
$\hat{f}(n)$	$f_0(0)$	$f_1(0)$	$f_2(0)$	$f_0(1)$	$f_1(1)$	$f_2(1)$	$f_3(0)$	...

is clearly onto. Lemma 2.11 applies, and so the union is countable.

**II.2.33** Think of a function  $f: \{0, 1\} \rightarrow \mathbb{N}$  as a pair  $\langle f(0), f(1) \rangle$ . That is, there is a natural correspondence between  $\{0, 1\}$  and  $\mathbb{N} \times \mathbb{N}$ , which is countable.

**II.2.34** Because  $S$  is countable, Lemma 2.11 says that either it is empty or there is an onto function  $g: \mathbb{N} \rightarrow S$ . If  $S$  is empty then the range set is empty, so it is countable. Otherwise, the function  $f \circ g: \mathbb{N} \rightarrow \text{ran}(f)$  is onto.

**II.2.35** The cross product of two finite sets is finite because the number of elements in the cross product equals the product of the number of elements in the sets.

Next we do a finite set,  $S = \{s_0, \dots, s_{m-1}\}$ , and a countably infinite set,  $T = \{t_0, t_1, \dots\}$ . Consider  $S \times T$  (the  $T \times S$  argument is similar). Enumerate it row-wise, so that where  $n = d \cdot m + r$  is the usual quotient-remainder relationship then  $f(n) = \langle s_r, t_d \rangle$ . (This initial portion of the  $m = 3$  case

$n$	0	1	2	3	4	5	6	...
$f(n)$	$\langle 0, 0 \rangle$	$\langle 1, 0 \rangle$	$\langle 2, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 1 \rangle$	$\langle 0, 2 \rangle$	...

illustrates that an equivalent formula is  $f(n) = \langle s_{n \bmod m}, t_{\lfloor n/m \rfloor} \rangle$ .) Clearly this function is a correspondence.

**II.2.36**

- (A) The set of length 5 strings  $\Sigma^5 = \{\langle s_0, \dots, s_4 \rangle \mid s_i \in \Sigma\}$  is the cross product of five finite sets, each with 40 characters. So  $\Sigma^5$  has  $40^5$  many members. (By the way,  $40^5 = 102\,400\,000$ .) Thus it is countable.
- (B) The set  $\Sigma^0 \cup \Sigma^1 \cup \dots \cup \Sigma^5$  is the union of finitely many finite sets. So it is finite and therefore countable.
- (C) The set  $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \dots$  is the union of countably many finite sets. By Corollary 2.12 it is countable.
- (D) A program is a finite string. So the set of all programs is a subset of  $\Sigma^*$ . Corollary 2.12 says that this set is countable.

**II.2.37**

(A) These are easy to calculate by hand or with a script.

	$m = 0$	$m = 1$	$m = 2$	$m = 3$
$n = 0$	0	2	4	6
$n = 1$	1	5	9	13
$n = 2$	3	11	19	27
$n = 3$	7	23	39	55

- (B) Every natural number greater than zero is a unique product of primes,  $2^{e_2}3^{e_3} \dots p^{e_p}$ . So every such number is the product of a power of two,  $2^n$ , with an odd number,  $2m + 1$ , and so corresponds to the pair  $\langle n, m \rangle$ .
- (C) This table shows that under the box enumeration  $\langle 3, 4 \rangle$  corresponds with 19.

$y = 4$	16	17	18	19	20
$y = 3$	9	10	11	12	21
$y = 2$	4	5	6	13	22
$y = 1$	1	2	7	14	23
$y = 0$	0	3	8	15	24
	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$

**II.2.38**

- (A)  $\text{cantor}(2, 1) = 8$
- (B) Solving  $8 = 1 + [(1 + y)(1 + y + 1)]/2$  leads to the quadratic equation  $0 = y^2 - 3y - 12$ . The quadratic formula gives  $(-3 \pm \sqrt{57})/2$ , two different solutions,  $y \approx 2.27$  and  $y \approx -5.27$ . Figure 9 on page 49 shows the graph.

**II.2.39**

(A) Let  $f_0: \mathbb{N} \rightarrow S_0$  and  $f_1: \mathbb{N} \rightarrow S_1$  be correspondences. Then this map is onto the set  $S_0 \cup S_1$

$$\hat{f}(n) = \begin{cases} f_0(g_0^{-1}(n)) & \text{if } n \in C_0 \\ f_1(g_1^{-1}(n)) & \text{if } n \in C_1 \end{cases}$$

because the range of  $g_0^{-1}$  is  $\mathbb{N}$  and the image of  $\mathbb{N}$  under  $f_0$  is  $S_0$ , and likewise for  $S_1$ .

(B) Let  $f_0: \mathbb{N} \rightarrow S_0$ ,  $f_1: \mathbb{N} \rightarrow S_1$ , and  $f_2: \mathbb{N} \rightarrow S_2$  be correspondences. Then this map is onto.

$$\hat{f}(n) = \begin{cases} f_0(g_0^{-1}(n)) & \text{if } n \in C_0 \\ f_1(g_1^{-1}(n)) & \text{if } n \in C_1 \\ f_2(g_2^{-1}(n)) & \text{if } n \in C_2 \end{cases}$$

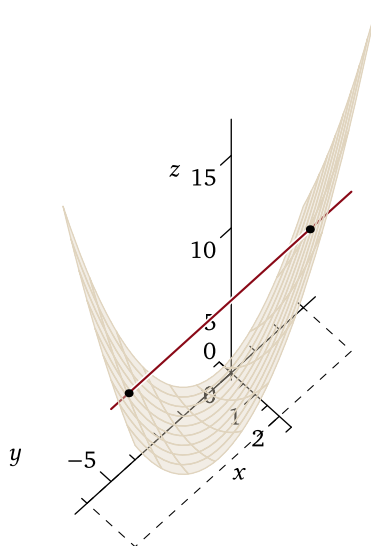


FIGURE 9, FOR QUESTION II.2.38: Cantor's unpairing function, along with the line where  $x = 1$  and  $z = 8$ .

- (c) Begin with countably infinitely many sets  $S_j$ , each of which is countable. Let  $f_j: \mathbb{N} \rightarrow S_j$  be correspondences. Write  $C_k$  for the set of natural numbers whose binary representation ends in  $k$ -many 0's but not  $k+1$ -many. Each  $C_k$  is countably infinite so there are correspondences  $g_k: \mathbb{N} \rightarrow C_k$  for all  $k \in \mathbb{N}$ . Then where  $n \in C_j$  the map given by  $\hat{f}(n) = f_j(g_j^{-1}(n))$  has domain  $\mathbb{N}$  and codomain  $\cup_j S_j$ , and is clearly onto.

**II.3.9** Infinite is different than exhaustive. The list  $0, 2, 4, \dots$  is infinite but does not exhaust all of the natural numbers.

**II.3.10** The union is not the whole real numbers. It does not include irrational numbers. For instance,  $\pi = 3.14\dots$  is not in any of the sets and so is not in the union.

**II.3.11** Cantor's Theorem says that the power set has more elements than the set.

- (A) The set  $\{0, 1, 2\}$  has three elements. Its power set has eight.

$$\mathcal{P}(\{0, 1, 2\}) = \{\{0, 1, 2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0\}, \{1\}, \{2\}, \emptyset\}$$

- (B) The set  $\{0, 1\}$  has two elements while its power set  $\mathcal{P}(\{0, 1\}) = \{\{0, 1\}, \{0\}, \{1\}, \emptyset\}$  has four.  
 (C) The set  $\{0\}$  only has one element but its power set  $\mathcal{P}(\{0\}) = \{\{0\}, \emptyset\}$  has two.  
 (D) The empty set has zero elements, while its power set  $\mathcal{P}\emptyset = \{\emptyset\}$  has one.

**II.3.12** The definition of ' $\leq$ ' for cardinality is that  $|S| \leq |T|$  if there is a one-to-one function from  $S$  to  $T$ .

- (A) The function  $f: S \rightarrow \hat{S}$  given by  $f(1) = 11$ ,  $f(2) = 12$ , and  $f(3) = 13$  is one-to-one by inspection.  
 (B) The function  $f: T \rightarrow \hat{T}$  given by  $f(1) = 11$ ,  $f(2) = 12$ , and  $f(3) = 13$  is one-to-one by inspection.  
 (C) Again by inspection, the function  $f: S \rightarrow \hat{S}$  given by  $f(1) = 1$ ,  $f(2) = 3$ , and  $f(3) = 5$  is one-to-one.  
 (D) The function  $f: \mathbb{E} \rightarrow \mathbb{O}$  given by  $f(x) = x+1$  is a one-to-one function. The verification is easy:  $f(x_0) = f(x_1)$  implies that  $x_0 + 1 = x_1 + 1$ , and so  $x_0 = x_1$ .

**II.3.13** The two differ on the set from which the elements are drawn.

- (A) Countable; this is the set  $\{\dots, -2, -1, 0, 1\}$ , which is a subset of the natural numbers.  
 (B) Uncountable; it is the set of real numbers  $(-\infty..2)$ . (This set corresponds to  $\mathbb{R}$  under the map  $x \mapsto \ln(-1 \cdot (x - 2))$ .)

**II.3.14**

- (A) This set is uncountable. (It has at least the same cardinality as  $(1..4) \subset \mathbb{R}$  since the embedding map is one-to-one. This interval has the same cardinality as  $\mathbb{R}$  because  $x \mapsto \tan((\pi/3) \cdot (x - 1) - (\pi/2))$  is a correspondence.)  
 (B) Countable. (Because it is given as a subset of the naturals, this set is  $\{1, 2, 3\}$ , which is finite.)  
 (C) Uncountable. (The function  $x \mapsto \ln(x - 5)$  is a correspondence with  $\mathbb{R}$ .)

(D) Countable. (Any subset of  $\mathbb{N}$  is countable.)

**II.3.15** For the two element set  $A_2 = \{0, 1\}$ , the power set,  $\mathcal{P}(A_2) = \{\{0, 1\}, \{0\}, \{1\}, \{\}\}$ , has four elements. Making a function from  $A_2$  to  $\mathcal{P}(A_2)$  involves picking where to send 0 and where to send 1. So there are  $4 \cdot 4 = 16$  different functions.

<i>function</i> $f_i$	$f_i(0)$	$f_i(1)$	<i>function</i> $f_i$	$f_i(0)$	$f_i(1)$
$f_0$	{ }	{ }	$f_8$	{ 1 }	{ }
$f_1$	{ }	{ 0 }	$f_9$	{ 1 }	{ 0 }
$f_2$	{ }	{ 1 }	$f_{10}$	{ 1 }	{ 1 }
$f_3$	{ }	{ 0, 1 }	$f_{11}$	{ 1 }	{ 0, 1 }
$f_4$	{ 0 }	{ }	$f_{12}$	{ 0, 1 }	{ }
$f_5$	{ 0 }	{ 0 }	$f_{13}$	{ 0, 1 }	{ 0 }
$f_6$	{ 0 }	{ 1 }	$f_{14}$	{ 0, 1 }	{ 1 }
$f_7$	{ 0 }	{ 0, 1 }	$f_{15}$	{ 0, 1 }	{ 0, 1 }

The three element set  $A_3$  has  $|\mathcal{P}(A_3)| = 8$ . The number of functions from  $A_3$  to  $\mathcal{P}(A_3)$  is  $8^3 = 512$ . In general, where  $|A| = n$  the power set has  $|\mathcal{P}(A_3)| = 2^n$ -many elements. The number of functions is  $(2^n)^n = 2^{(n^2)}$ . (As a check,  $n = 2$  gives  $2^4 = 16$  and  $n = 3$  gives  $2^9 = 512$ .)

**II.3.16**

(A) These are the functions from  $S$  to  $T$ .

<i>function</i> $f_i$	$f_i(0)$	$f_i(1)$
$f_0$	10	10
$f_1$	10	11
$f_2$	11	10
$f_3$	11	11

The functions that are one-to-one are  $f_1$  and  $f_2$ .

(B) These are the functions from  $S$  to  $T$ .

<i>function</i> $f_i$	$f_i(0)$	$f_i(1)$	<i>function</i> $f_i$	$f_i(0)$	$f_i(1)$
$f_0$	10	10	$f_5$	11	12
$f_1$	10	11	$f_6$	12	10
$f_2$	10	12	$f_7$	12	11
$f_3$	11	10	$f_8$	12	12
$f_4$	11	11			

The functions that are one-to-one are  $f_1, f_2, f_3, f_5, f_6,$  and  $f_7$ .

**II.3.17**

(A) The answer is (iii) since the union of two finite sets is finite. (All of these apply: (ii) countable or uncountable, (iii) finite, (iv) countable, and (v) finite, countably infinite, or uncountable. The sharpest one, the one that is most specific, is (iii).)

(B) The sharpest is (iv).

(C) The sharpest is (i).

(D) The sharpest is (iv). (The intersection could be finite but there are cases where it is countable, such as  $A = \mathbb{N}$  and  $B = \mathbb{R}$ . So that is the sharpest statement.)

**II.3.18** The question doesn't require a proof but here the parenthetical comments give a justification.

(A) They are all possible except for the last one. (An example of (i) and (ii) is  $\text{id}: \{0, 1\} \rightarrow \{0, 1\}$ . An example of (iii) and (iv) is  $\text{id}: \mathbb{N} \rightarrow \mathbb{N}$ . Lemma 2.11 rules out (v).)

(B) They are all possible. (An example of (i) and (ii) is  $\text{id}: \{0, 1\} \rightarrow \{0, 1\}$ . An example of (iii) and (iv) is  $\text{id}: \mathbb{N} \rightarrow \mathbb{N}$ . An example of (v) is  $\iota: \mathbb{N} \rightarrow \mathbb{R}$  given by  $\iota(x) = x$ .)

**II.3.19** Theorem 3.6 says that a set's cardinality is strictly less than that of its power set. So one set with a cardinality larger than  $\mathbb{R}$  is  $\mathcal{P}(\mathbb{R})$ .

**II.3.20**

- (A) The set of finite bit strings,  $\mathbb{B}^*$ , is the union of the sets  $\mathbb{B}^k = \{ \langle b_0 b_1 \dots b_{k-1} \rangle \mid k \in \mathbb{N} \}$ . The set  $\mathbb{B}^k$  is finite, it has  $2^k$  many elements, and so this is a countable union of countable sets.
- (B) Suppose that the set of infinite bit strings has an exhaustive enumeration  $f_0, f_1, \dots$ . Let  $g: \mathbb{B} \rightarrow \mathbb{B}$  be given by  $g(0) = 1$  and  $g(1) = 0$ . Then the infinite bit string  $F: \mathbb{N} \rightarrow \mathbb{B}$  defined by  $F(i) = g(f_i(i))$  is not equal to any of the  $f_i$ , which contradicts that the enumeration is exhaustive.

**II.3.21** Let  $S$  and  $T$  be such that  $S \subseteq T$ . Then the identity function, the map  $\text{id}: S \rightarrow T$  given by  $\text{id}(s) = s$ , is clearly one to one. Then  $|S| \leq |T|$  by Definition 3.3.

**II.3.22** Let the collection of all such functions be  $S = \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \text{ran}(f) \text{ is finite} \}$ . Assume that it is countable so that there is a correspondence  $h: \mathbb{N} \rightarrow S$ . For all natural numbers  $i$ , the value of  $h(i)$  is a function, which we will denote  $f_i$ .

Define  $g: \mathbb{N} \rightarrow \mathbb{N}$  by  $g(x) = 0$  if  $x \neq 0$  and  $g(x) = 1$  otherwise. Consider the function  $k: \mathbb{N} \rightarrow \mathbb{N}$  given by  $k(i) = g(f_i(i))$ . This map has a finite range, namely  $\text{ran}(k) = \{0, 1\}$ , but  $k$  is not equal to any  $f_i \in S$  because they differ on the input  $i$ .

**II.3.24** As stated in the question, the set of rational numbers is countable. Suppose that the irrational numbers were also countable. Then, because the union of two countable sets is countable, the real numbers would be countable. But the reals are not countable and so the irrationals are not countable either.

**II.3.25**

- (A) Write the decimal expansion of  $z$  as  $z = 0.z_0z_1z_2\dots$ . It does not end in 9's as the output of  $g$  is never a 9. So  $z$  has a unique decimal expansion. For each  $q_i$  the decimal expansion of  $z$  differs from  $q_i$ 's in that  $z_i \cdot 10^{-(i+1)} \neq q_{i,i} \cdot 10^{-(i+1)}$ , by the definition of  $g$ . Because their decimal expansions differ, and because neither ends in 9's, the two are different numbers.

Since  $z$  does not equal any rational number, it is irrational.

- (B) If  $d = \sum_{n \in \mathbb{N}} d_{n,n} \cdot 10^{-(n+1)}$  is rational then its decimal expansion repeats. That is, there is a decimal place such that past that place the expansion consists of a sequence of digits  $d_j d_{j+1} \dots d_{j+k}$  repeated again and again. But then the prior item's  $z$  would also repeat, in that past that same decimal place its expansion would consist of a sequence of digits  $g(d_j)g(d_{j+1}) \dots g(d_{j+k})$  repeated again and again. That would make  $z$  rational, which it isn't.
- (C) Why should it be a contradiction? In Theorem 3.1 we start with what purports to be a list of all reals and then produce a real that is not on that list. Here we start with what is a list of all rationals and produce something that is not a rational. Entirely different.

**II.3.26** Before we do the proof, consider  $S = \{0, 1, 2, 3\}$ . These are the elements of  $\mathcal{P}(S)$ .

$$\begin{aligned} & \{ \}, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}, \{1, 2\}, \{0, 1, 2\} \\ & \{3\}, \{0, 3\}, \{1, 3\}, \{2, 3\}, \{0, 1, 3\}, \{0, 2, 3\}, \{1, 2, 3\}, \{0, 1, 2, 3\} \end{aligned}$$

So, fixing  $\hat{s} = 3 \in S$  gives two classes of subsets, those sets containing  $\hat{s}$  and those not containing  $\hat{s}$ , and the two have the same number of members.

Our proof uses induction on  $|S|$ . The base case is that  $|S| = 0$ , so  $S$  is empty. Then  $\mathcal{P}(S) = \{\emptyset\}$  and  $|\mathcal{P}(S)| = 1$ , as required.

For the inductive step fix  $k \in \mathbb{N}^+$ , assume that the statement is true for  $n = 0, n = 1, \dots, n = k$ , and consider a set  $S$  where  $|S| = k + 1$ . Because  $|S| = k + 1$  the set has an element,  $\hat{s} \in S$ . The inductive hypothesis applies to  $S - \{\hat{s}\}$ , so  $\mathcal{P}(S - \{\hat{s}\})$  has  $2^k$ -many elements. As illustrated in the example above, the elements of  $\mathcal{P}(S - \{\hat{s}\})$ , which do not contain  $\hat{s}$ , are in correspondence with the elements of  $\mathcal{P}(S)$  that do contain  $\hat{s}$ . Thus the total number of elements of  $\mathcal{P}(S)$  is  $2 \cdot 2^k$ , which equals  $2^{k+1}$ .

**II.3.27** Suppose that  $f(s_i) = H$ . There are two possibilities, either  $s_i \in H$  or else  $s_i \notin H$ . If  $s_i \in H$  then student  $s_i$  is a member of their own list, so  $s_i$  is not humble, so this violates its case assumption. On the other hand, if  $s_i \notin H$  then  $s_i$  is not a member of their own list, so  $s_i$  is humble, which also violates this case assumption.

**II.3.28** One way is to take the diagonal bits two at a time; for instance changing 01 to 10, and changing any other two-bit sequence to 01.

**II.3.29**

- (A) In  $a + ar + ar^2 + ar^3 + \dots = a \cdot 1/(1-r)$ , take  $a = 9$  and  $r = 0.1$ . The left side is  $9 + 0.9 + 0.09 + 0.009 + \dots =$

9.999 99 ... The right side is  $9 \cdot 1/(1 - (0.1)) = 9/0.9 = 1/0.1 = 1/(1/10) = 10$ . Subtract 9 from both sides to get  $0.999 99 \dots = 1$ .

- (B) Consider a real number with two representations. Write the representations as  $\vec{x} = \langle x_N, x_{N-1}, \dots \rangle$  and  $\vec{y} = \langle y_N, y_{N-1}, \dots \rangle$ , subject to  $x_i, y_i \in \{0, 1, \dots, 9\}$  and to this condition.

$$\sum_{N \geq i} x_i \cdot 10^i = \sum_{N \geq i} y_i \cdot 10^i \tag{*}$$

(The number  $N \in \mathbb{Z}$  is the furthest left nonzero decimal place among the two representations.) For instance, for  $1.000 \dots = 0.999 \dots$  we have  $x_0 = 1, x_{-1} = 0, x_{-2} = 0, \dots$ , giving  $1 \cdot 10^0 + 0 \cdot 10^{-1} + 0 \cdot 10^{-2} + \dots$ , and  $y_0 = 0, y_{-1} = 9, y_{-2} = 9, \dots$ , giving  $0 \cdot 10^0 + 9 \cdot 10^{-1} + 9 \cdot 10^{-2} + \dots$ .

Let the leftmost decimal place where the two differ be  $M$ , so that  $x_M \neq y_M$  and  $x_i = y_i$  for  $i \in \{N, N - 1, \dots, M\}$ . One of  $x_M$  or  $y_M$  is larger; without loss of generality suppose that  $x_M > y_M$ . We must show that  $x_M = y_M + 1$  (as it does for the example  $\langle 1, 0, 0 \dots \rangle$  and  $\langle 0, 9, 9 \dots \rangle$ ).

The reason is that the remaining digits can add to a number no larger than  $\sum_{M > i} 9 \cdot 10^i$ . This adds to  $10^M$  by the geometric series formula. If  $x_M - y_M > 1$  then  $(\sum_{N \leq i \leq M} x_i 10^i) - (\sum_{N \leq i \leq M} y_i 10^i) = (x_M - y_M) \cdot 10^M$  is at least  $2 \cdot 10^M$ , and the remaining digits cannot make up the difference to satisfy condition (\*).

- (c) Continuing with the notation from the prior item, we want to show that  $x_j = 0$  and  $y_j = 9$  for  $j \in \{M - 1, M - 2, \dots\}$ . The argument is similar to the prior one and we will do only  $j = M - 1$ . We have that the representations differ by one in decimal place  $M$ .

$$\left( \sum_{N \leq i \leq M} x_i 10^i \right) - \left( \sum_{N \leq i \leq M} y_i 10^i \right) = 10^M$$

Suppose that  $y_{M-1} - x_{M-1} \leq 8$ . Then

$$\left( \sum_{N \leq i \leq M-1} x_i 10^i \right) - \left( \sum_{N \leq i \leq M-1} y_i 10^i \right) \geq 10^M - 8 \cdot 10^{M-1}$$

and the remaining digits can add to a number no larger than  $\sum_{M-1 > i} 9 \cdot 10^i = 1 \cdot 10^{M-1}$ , again by the geometric series formula. Adding that into the prior equation is not enough to satisfy (\*). (The cases of  $j = M - 2, j = M - 3$ , etc. are similar.)

**II.3.30** Suppose otherwise, that  $\mathcal{C}$  is the set of all sets. Then  $\mathcal{P}(\mathcal{C})$  would be a collection of sets so we would have  $\mathcal{P}(\mathcal{C}) \subseteq \mathcal{C}$ . That would give  $|\mathcal{P}(\mathcal{C})| \leq |\mathcal{C}|$ , which would contradict Cantor's Theorem.

**II.3.31**

- (A) These are the chains for  $S$ .

$s \in S$	Chain
0	... 0, a, 0, a ...
1	... 1, b, 1, b ...
2	... 2, d, 3, c, 2, d ...
3	- same as the chain for 2 -

These are the chains for  $T$ .

$t \in T$	Chain
a	... a, 0, a, 0 ...
b	... b, 1, b, 1 ...
c	... c, 2, d, 3, c, 2 ...
d	- same as the chain for c -

- (B) Let  $S = \{0, 2, 4, 6, \dots\}$ , let  $T = \{1, 3, 5, \dots\}$ , and take  $f: S \rightarrow T$  to be  $f(s) = s + 1$ , and  $g: T \rightarrow S$  to be  $g(t) = t + 1$ . The function  $f$  is one-to-one because if  $f(s) = f(\hat{s})$  then  $s + 1 = \hat{s} + 1$ , and so  $s = \hat{s}$ . The function  $g$  is one-to-one in the same way.

The chain associated with  $0 \in S$  is  $0, 1, 2, 3 \dots$ . It contains all the elements of both sets, and so the chain is unique. It also has a first element, that is, there is no element of  $T$  that is  $g^{-1}(0)$ .

- (C) If they are not already disjoint, if  $S \cap T \neq \emptyset$ , then we can just color all the elements of  $S$  purple and all the elements of  $T$  gold and that will make them disjoint for sure. More precisely, consider the set  $\{0\} \times S$  of pairs  $\langle 0, s \rangle$  along with the set  $\{1\} \times T$  of pairs  $\langle 1, t \rangle$ . These are disjoint sets because any two elements differ in the first entry. Obviously they correspond to the original sets  $S$  and  $T$ .
- (D) First we show that every element is in a unique chain. Fix some  $x \in S \cup T$ . Because we assume that the two sets are disjoint, the function operation that applies to  $x$  is determined, meaning that if  $x \in S$  then for the chain we next consider  $f(x)$  while if  $x \in T$  then we are next doing  $g$ . With that, all the elements of the chain to the right of  $x$  are determined. And, because the two functions are one-to-one, all of the element to the left are likewise determined also.

As to the four possibilities, either a chain is finite, that is, it repeats, or it is infinite. If it repeats then it is case (i). If it is infinite then it either does not have an initial element, and so is case (ii), or else it does. If it does have an initial element then either that element is from the set  $S$ , as in case (iii), or else that element is from the set  $T$ , as in case (iv).

- (E) We will consider the correspondence for each chain type below. But first note that because each case is a restriction of a function, each is well-defined. That is, never does any of these maps associate two different outputs with the same input.

A type (i) chain repeats after some number of elements  $s_0, t_0, s_1, t_1, \dots, s_{n-1}, t_{n-1}, s_n = s_0$  (because the sets are disjoint it cannot repeat after a single element, and we can thus take the first element to be a member of  $S$ ). The function  $f$  makes these associations.

$$s_0 \mapsto t_0 \quad s_1 \mapsto t_1, \quad \dots \quad s_{n-1} \mapsto t_{n-1}$$

This is clearly a correspondence among the elements of the chain. It is manifestly onto the set  $\{t_0, t_1, \dots, t_{n-1}\}$  and it is one-to-one because the  $t_j$ 's are distinct, or else the chain would be shorter.

For a type (ii) chain

$$\dots f^{-1}(g^{-1}(s)), g^{-1}(s), s, f(s), g(f(s)), f(g(f(s))), \dots$$

the function  $f$  associates these.

$$\dots f^{-1}(g^{-1}(s)) \mapsto g^{-1}(s) \quad s \mapsto f(s) \quad g(f(s)) \mapsto f(g(f(s))), \dots$$

This also is clearly a correspondence.

A type (iii) chain

$$s, f(s), g(f(s)), f(g(f(s))), \dots$$

has these associations.

$$s \mapsto f(s) \quad g(f(s)) \mapsto f(g(f(s))), \dots$$

Again, clearly a correspondence.

The one that takes some thought is a chain of type (iv).

$$t, g(t), f(g(t)), g(f(g(t))), \dots$$

We want a correspondence function from  $S$  to  $T$ . Consider these associations.

$$g(t) \mapsto t \quad g(f(t)) \mapsto f(t), \dots$$

Because the function  $g$  is one-to-one its inverse is defined, over its range. So this association is a well-defined function. Like the chain types above, this also is clearly a correspondence.

**II.4.5** A Universal Turing machine is a regular machine, in that it is just another machine in the list  $\mathcal{P}_0, \mathcal{P}_1, \dots$ . Perhaps your friend meant something more like, "What does a Universal Turing machine do that a non-universal machine does not do?" Every machine does a job. Some machines expect a single input and

then double it, some interpret their input as two numbers and then add them. A Universal Turing machine expects two inputs,  $e$  and  $x$ , and the output of this machine is the same as the output of the machine  $\mathcal{P}_e$  on input  $x$ , including failing to halt if that machine fails to halt. It is universal in that any input/output behavior that can be mechanically produced can be produced by this one device.

**II.4.6** Every general purpose computer, including any standard laptop or cell phone, is equivalent in computable power to a Universal Turing machine.

A person can object that a standard laptop for instance does not come from the store with unboundedly much memory. That's true. But such a machine can store things in the Cloud and so is not restricted to the installed memory, and in this sense its memory is unbounded.

A person can go on to worry that there are only so many bits available in the physical universe and thus memory is not in the end unbounded. Perhaps that is true — although we could worry back that we have not got enough time to exhaust what bits that there are before the Big Bang becomes a Big Crunch, or whatever it will become — but at the least we can take general purpose devices to be so very close to Universal Turing machines that it is quite hard to tell them apart.

**II.4.7** Yes. If  $\mathcal{P}_{e_0}$  and  $\mathcal{P}_{e_1}$  are universal then giving the first one the inputs  $e_1$  and  $x$  will result in an input/output behavior exactly like the second. Likewise, giving  $\mathcal{P}_{e_0}$  the inputs  $e_0$  and  $x$  will result in the machine behaving just like itself.

**II.4.8** The answer is that it does those things in software. The universal machine does not use its states to simulate the states of the simulated machine. Rather, it has codes for those states that it manipulates on the tape.

Chucklehead.

**II.4.9** Yes, every computable function has infinitely many indices by Lemma 2.15. So if  $\mathcal{P}_e$  is universal, then the associated function  $\phi_e$  has infinitely many unequal indices so that  $\phi_e = \phi_{e_0} = \phi_{e_1} = \dots$ . The machines  $\mathcal{P}_{e_0}, \mathcal{P}_{e_1}, \dots$  are all universal.

**II.4.10** We know that 5 equals  $\text{cantor}(2, 0)$ , that is,  $\text{pair}(5) = \langle 2, 0 \rangle$ . So in  $\phi_{e_0}(e_0, 5)$  the argument to  $\phi_{e_0}$  is the value of  $\phi_2(0)$ . So, where  $\text{pair}(\phi_2(0)) = \langle a, b \rangle$ , the value of  $\phi_{e_0}(e_0, 5)$  is  $\phi_a(b)$ .

**II.4.11** We write  $f_{i,a}$  for the function produced by freezing  $x_i$  to the value  $a$ .

(A) The resulting one-variable function is  $f_{0,4}(x_1) = 12 + 4x_1$ .

(B) The one-variable function is  $f_{0,5}(x_1) = 15 + 5x_1$ .

(C) The one-variable function is  $f_{1,0}(x_0) = 3x_0$ .

**II.4.12** We use  $\hat{f}$  for the name of each function.

(A) The resulting two-variable function is  $\hat{f}(x_1, x_2) = 1 + 2x_1 + 3x_2$ .

(B) The result is the two-variable function  $\hat{f}(x_1, x_2) = 2 + 2x_1 + 3x_2$ .

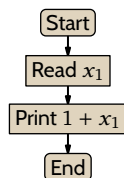
(C) This gives  $\hat{f}(x_1, x_2) = a + 2x_1 + 3x_2$ .

(D) This results in the one-variable function  $\hat{f}(x_2) = 11 + 3x_2$ .

(E) The result is  $\hat{f}(x_2) = a + 2b + 3x_2$ .

**II.4.13**

(A) The two subscripts on  $s_{1,1}$  mean that we will freeze the first input and leave the second one as a variable. So  $\phi_{s_{1,1}(e,1)}$  will be a one-input function. The arguments  $e$  and 1 indicate that we are working on Turing machine  $e$ , the one whose flow chart is given in the problem statement, and that the input is frozen to be  $x_0 = 1$ . This flowchart gives a sense of the result.

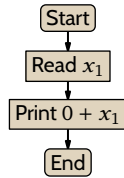


The function is  $\phi_{s_{1,1}(e,1)}(x_1) = 1 + x_1$ .

(B) The values are  $\phi_{s_{1,1}(e,1)}(0) = 1$ ,  $\phi_{s_{1,1}(e,1)}(1) = 2$ , and  $\phi_{s_{1,1}(e,1)}(2) = 3$ .

(C) The notation  $s_{1,1}(e, 0)$  indicates that we work on Turing machine  $e$ , that we freeze the first input to  $x_0 = 0$ ,

and that we leave the second input as a variable. This flowchart gives the idea.

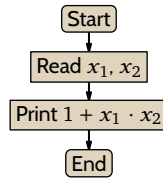


The function is  $\phi_{s_{1,1}(e,1)}(x_1) = 0 + x_1 = x_1$ .

(D) We have  $\phi_{s_{1,1}(e,0)}(0) = 0$ ,  $\phi_{s_{1,1}(e,0)}(1) = 1$ , and  $\phi_{s_{1,1}(e,0)}(2) = 2$ .

**II.4.14**

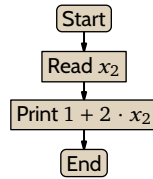
(A) The notation  $s_{1,2}(e, 1)$  indicates that we work on  $\mathcal{P}_e$ , that we freeze the first input to  $x_0 = 1$ , and that we leave the second and third inputs as variables. This flowchart sketches the behavior.



The function is  $\phi_{s_{1,2}(e,1)}(x_1, x_2) = 1 + x_1 \cdot x_2$ .

(B) We have  $\phi_{s_{1,2}(e,1)}(0, 1) = 1$ ,  $\phi_{s_{1,2}(e,1)}(1, 0) = 1$ , and  $\phi_{s_{1,2}(e,1)}(2, 3) = 7$ .

(C) The  $s_{2,1}(e, 1, 2)$  says that we start with  $\mathcal{P}_e$ , that we freeze the first input to  $x_0 = 1$  and the second input to  $x_1 = 2$ , and leave the third input as a variable. This sketches the result.



The function is  $\phi_{s_{2,1}(e,1,2)}(x_2) = 1 + 2 \cdot x_2$ .

(D) The values are  $\phi_{s_{2,1}(e,1,2)}(0) = 1$ ,  $\phi_{s_{2,1}(e,1,2)}(1) = 3$ , and  $\phi_{s_{2,1}(e,1,2)}(2) = 5$ .

**II.4.15**

(A)  $\phi_{s_{1,1}(e,0)}(x_1) = x_1$

(B)  $\phi_{s_{1,1}(e,0)}(5) = 5$

(C)  $\phi_{s_{1,1}(e,1)}(x_1) = x_1$

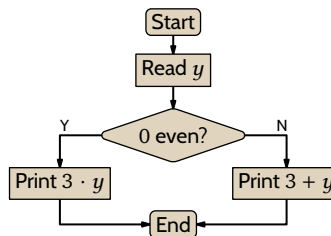
(D)  $\phi_{s_{1,1}(e,1)}(5) = 5$

(E)  $\phi_{s_{1,1}(e,2)}(x_1) \uparrow$

(F)  $\phi_{s_{1,1}(e,2)}(5) \uparrow$

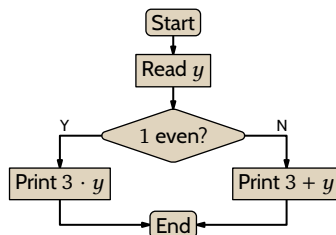
**II.4.16**

(A) This flowchart sketches  $\mathcal{P}_{s(e,0,3)}$ .



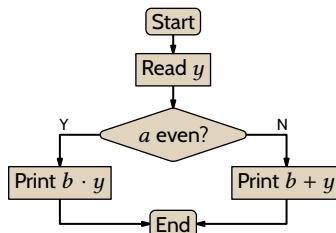
The function is  $\phi_{s(e,0,3)}(y) = 3y$ . Consequently,  $\phi_{s(e,0,3)}(5) = 15$ .

(B) This is the flowchart sketching  $\mathcal{P}_{s(e,1,3)}$ .



The computed function is  $\phi_{s(e,1,3)}(y) = 3 + y$ , and consequently  $\phi_{s(e,1,3)}(5) = 8$ .

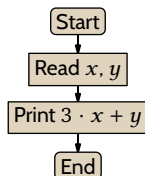
(c) This is the flowchart sketching  $\mathcal{P}_{s(e,a,b)}$ .



The distinction between this and the flowchart in the question statement is that here the machine does not read in  $x_0$  or  $x_1$ . Those two are hard-coded, as  $a$  and  $b$ , into the program body. This is a family of functions parametrized by  $a$  and  $b$ , and the indices of these functions are uniformly computable from  $e$ ,  $a$ , and  $b$ , using the  $s$ - $m$ - $n$  function  $s$ .

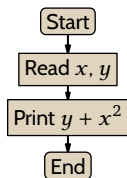
**II.4.17**

(A) The function  $\psi(x, y) = 3x + y$  is computable by Church's Thesis, as sketched in this flowchart.



(B) By the prior item there is a Turing machine that computes the function  $\psi$ . Suppose that it is  $\mathcal{P}_e$ . The  $s$ - $m$ - $n$  theorem gives that  $\mathcal{P}_{s_{1,1}(e,n)}$  computes the function  $y \mapsto 3n + y$ . Take  $\psi_n = \phi_{s_{1,1}(e,n)}$ .

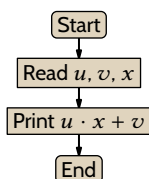
**II.4.18** Start with the function  $\psi: \mathbb{N}^2 \rightarrow \mathbb{N}$  defined by  $\psi(x, y) = y + x^2$ . By Church's Thesis it is computable, since it is computed by the machine sketched by this flowchart.



Suppose that machine has index  $e$ , that is, suppose that  $\psi(x, y) = \phi_e(x, y)$ .

By the  $s$ - $m$ - $n$  theorem there is a family of computable functions parametrized by  $n$ ,  $\phi_{s_{1,1}(n)}$ , with the desired behavior,  $\phi_{s_{1,1}(n)}(y) = y + n^2$ . Take  $g(n)$  to be  $s_{1,1}(n)$ .

**II.4.19** Start with the function  $\psi(u, v, x) = ux + v$ . Church's Thesis gives that it is computable, by the machine sketched by this flowchart.



Let that machine have index  $e$ , that is, suppose that  $\psi(u, v, x) = \phi_e(u, v, x)$ .

The  $s$ - $m$ - $n$  theorem gives a family of computable functions  $\phi_{s_{2,1}(m,b)}$  parametrized by  $m$  and  $b$  with the desired behavior,  $\phi_{s_{2,1}(m,b)}(x) = m \cdot x + b$ . Take  $g(m, b)$  to be  $s_{2,1}(m, b)$ .

#### II.4.20

- (A) This gives the same result as  $\phi_{e_1}(5)$ , that is, it converges and gives 20.
- (B) This returns 4 times the the value of  $\text{cantor}(e_0, 5)$ .
- (C) This returns  $\phi_{e_1}(5) = 20$ .

#### II.4.21

- (A) This gives the same result as  $\phi_{e_1}(4)$ , that is, it gives 6.
- (B) This returns the value of  $\phi_4(e_1)$ , whatever that is.
- (C) This returns  $\phi_{e_2}(3)$  plus 2, which is 11.
- (D) Because  $\text{pair}(4) = \langle 1, 1 \rangle$ , this returns the same as  $\phi_1(1)$ , whatever that is.

**II.5.8** We have shown that there are tasks that no computer can do. The Halting problem is a concrete example of such a task. We will see many, many more.

Besides, there are many programs that we want to run without a time bound. One is an operating system.

**II.5.9** False. There is such a function. But it is not computable; there is no Turing machine whose behavior is that function.

**II.5.10** The unsolvability results do not say that you cannot prove that a particular machine does a particular thing. The given machine does indeed fail to halt on all inputs and it is indeed obvious. Instead the results say that no machine can take in the index  $x$  and correctly determine whether  $\mathcal{P}_x$  does that thing.

**II.5.11** First, to detect a loop we must look for a repeated configuration, not just a repeated state-character pair. Machines can run forever without repeating a configuration, for instance by just writing more characters to the tape, as in code that keeps incrementing a variable.

Further, the Halting problem is not to detect an infinite loop. Rather, the Halting problem is to detect whether the machine will halt. Here also, code that just keeps incrementing a variable will not loop, but also not halt.

**II.5.12** A machine could do that. It would not violate the unsolvability of the Halting problem.

However: a machine can fail to halt for reasons other than that it is in an infinite loop. The code below would not be flagged as being in an infinite loop by the runtime environment described, but it does not halt.

```
i=0
while True:
    i=i+1
```

**II.5.13** No. Either  $f$  does halt on all inputs, or it doesn't. So there are two possibilities, both of which are computable, it is just that we don't know which one is true.

#### II.5.14

- (A) False. The function exists. We say that the problem is unsolvable because that function is not mechanically computable.
- (B) False. Church's Thesis asserts that Turing machines are a maximally-powerful model of computation. The existence of unsolvable problems is inherent in effective computation.

**II.5.15** Any finite set is computable, so yes. Thus, we should be cautious about conflating 'computable' with knowable.

#### II.5.16

- (A) The function  $f(x, y) = 3x + y$  is intuitively computable, as sketched on the left of Figure 10 on page 58. Church's Thesis gives that there is a Turing machine with that behavior; let that machine have index  $e$ , so that  $f(x, y) = \phi_e(x, y)$  for all  $x, y \in \mathbb{N}$ . Applying the  $s$ - $m$ - $n$  Theorem to parametrize  $x$  gives this computable family of one-input computable functions:  $\phi_{s(e,x)}(y) = 3x + y$ . Flowcharts sketching some associated machines are on the right side of Figure 10 on page 58.
- (B) The function  $f(x, y) = xy^2$  is intuitively computable, as in Figure 11 on page 58. Church's Thesis gives that there is a Turing machine  $\mathcal{P}_e$  with that behavior, so that  $f(x, y) = \phi_e(x, y)$  for all  $x, y \in \mathbb{N}$ . Applying the  $s$ - $m$ - $n$  Theorem to parametrize  $x$  gives  $\phi_{s(e,x)}(y) = xy^2$ , which is a family of one-input functions. Flowcharts sketching some associated machines are on the right side of Figure 11 on page 58.

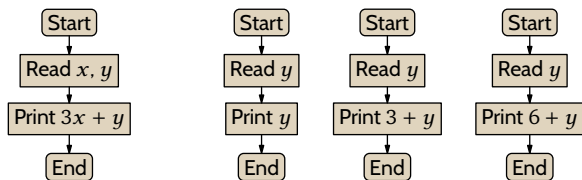


FIGURE 10, FOR QUESTION II.5.16: The machine associated with the function  $f = \phi_e$  and the machines associated with  $\phi_{s(e,x)}$  for  $x = 0$ ,  $x = 1$ , and  $x = 2$ .

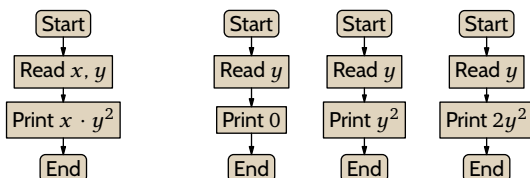


FIGURE 11, FOR QUESTION II.5.16: The machine associated with the function  $f = \phi_e$  and machines associated with  $\phi_{s(e,x)}$  for  $x = 0$ ,  $x = 1$ , and  $x = 2$ .

- (c) The function  $f$  is intuitively computable, as sketched on the left of Figure 12 on page 59. Church's Thesis gives that there is a Turing machine  $\mathcal{P}_e$  with that behavior, so that  $f(x, y) = \phi_e(x, y)$  for all  $x, y \in \mathbb{N}$ . Apply the  $s$ - $m$ - $n$  Theorem to parametrize  $x$ . The resulting family of one-input functions is this.

$$\phi_{s(e,x)}(y) = \begin{cases} x & \text{-- if } x \text{ is odd} \\ 0 & \text{-- otherwise} \end{cases}$$

Flowcharts sketching associated machines are on the right side of Figure 12 on page 59.

- II.5.17** The answer to all three is the same. Running a Turing machine for a fixed number of steps is not a problem (and in the third item, the number of steps is fixed for a given input).  
**II.5.18** We will show that this function is not mechanically computable.

$$\text{total\_decider}(e) = \begin{cases} 1 & \text{-- if } \phi_e(y) \downarrow \text{ for all } y \\ 0 & \text{-- otherwise} \end{cases}$$

The function

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } \phi_x(x) \downarrow \\ \uparrow & \text{-- otherwise} \end{cases}$$

is intuitively computable by the flowchart on the left of Figure 13 on page 59. Thus Church's Thesis says that there is a Turing machine that computes it; let that machine have index  $e$ , so that  $\psi(x, y) = \phi_e(x, y)$ . The  $s$ - $m$ - $n$  Theorem gives a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 13 on page 59. Notice that  $\phi_x(x) \downarrow$  if and only if  $\text{total\_decider}(s(e, x)) = 1$ .

Since  $s$  is computable, if we can mechanically compute  $\text{total\_decider}$  then we can mechanically solve the Halting problem. We cannot mechanically solve the Halting problem, so we cannot mechanically compute  $\text{total\_decider}$ .

- II.5.19** We will show that this function is not mechanically computable.

$$\text{square\_decider}(e) = \begin{cases} 1 & \text{-- if } \phi_e(y) = y^2 \text{ for all } y \\ 0 & \text{-- otherwise} \end{cases}$$

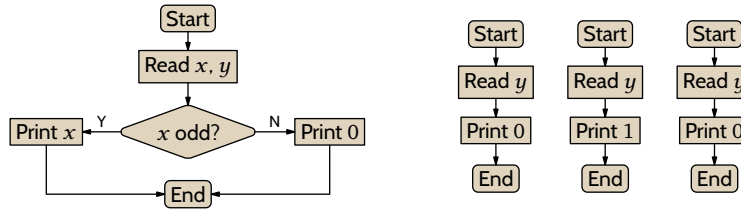


FIGURE 12, FOR QUESTION II.5.16: A sketch of the machine for the function  $f = \phi_e$  and of the machines associated with  $\phi_{s(e,x)}$  when  $x = 0$ ,  $x = 1$ , and  $x = 2$ .

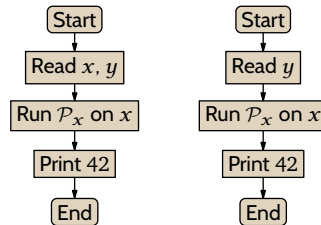


FIGURE 13, FOR QUESTION II.5.18: The machine associated with the function  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

Consider this function.

$$\psi(x, y) = \begin{cases} y^2 & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

The flowchart on the left of Figure 14 on page 60 shows that  $\psi$  is intuitively computable so by Church's Thesis there is a Turing machine that computes it. Let that machine have index  $e$ . Apply the  $s$ - $m$ - $n$  Theorem to get a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 14 on page 60. Then  $\phi_x(x) \downarrow$  if and only if  $\text{square\_decider}(s(e, x)) = 1$ .

So, since the function  $s$  is computable, if we can mechanically compute  $\text{square\_decider}$  then we can mechanically solve the Halting problem. But we can't mechanically solve the Halting problem so we cannot mechanically compute  $\text{square\_decider}$ .

**II.5.20** We will show that this function is not computable.

$$\text{same\_value\_decider}(e) = \begin{cases} 1 & \text{if } \phi_e(y) = \phi_e(y + 1) \text{ for some } y \\ 0 & \text{otherwise} \end{cases}$$

This function

$$\psi(x, y) = \begin{cases} 42 & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

is intuitively computable by the flowchart on the left of Figure 15 on page 60. So, by Church's Thesis there is a Turing machine that computes it; let that machine have index  $e$ . Apply the  $s$ - $m$ - $n$  Theorem to get a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 15 on page 60. Observe that these machines produce the function with constant output 42 if and only if the machine gets through the middle box. More precisely stated,  $\phi_x(x) \downarrow$  if and only if  $\text{same\_value\_decider}(s(e, x)) = 1$ .

Since the function  $s$  is computable, if we can mechanically compute  $\text{same\_value\_decider}$  then we can mechanically solve the Halting problem, which we cannot do.

**II.5.21** We will show that this function is not computable; no Turing machine has this input-output behavior.

$$\text{diverges\_on\_five\_decider}(e) = \begin{cases} 1 & \text{if } \phi_e(5) \uparrow \\ 0 & \text{otherwise} \end{cases}$$

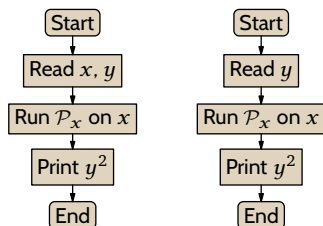


FIGURE 14, FOR QUESTION II.5.19: The machine associated with the function  $\psi = \phi_e$  and the machine associated with  $\phi_{s(e,x)}$ .

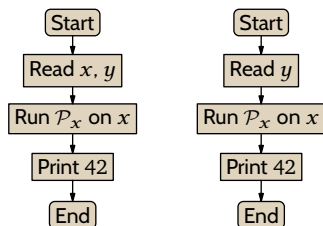


FIGURE 15, FOR QUESTION II.5.20: The machine associated with the function  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

The function

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } \phi_x(x) \downarrow \\ \uparrow & \text{-- otherwise} \end{cases}$$

is intuitively computable by the flowchart on the left of Figure 16 on page 61. So Church's Thesis says that there is a Turing machine that computes it. Let that machine have index  $e$ , so that  $\psi(x, y) = \phi_e(x, y)$ . The  $s$ - $m$ - $n$  Theorem gives a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 16 on page 61. Then  $\phi_x(x) \downarrow$  if and only if  $\text{diverges\_on\_five\_decider}(s(e, x)) = 1$ . Because the function  $s$  is mechanically computable, this means that if we could mechanically compute  $\text{diverges\_on\_five\_decider}$  then we could mechanically solve the Halting problem. But we cannot mechanically solve the Halting problem, so we cannot mechanically compute  $\text{diverges\_on\_five\_decider}$ .

**II.5.22** We want to show that this function is not computable.

$$\text{diverge\_on\_odds\_decider}(e) = \begin{cases} 1 & \text{-- if } \phi_e(y) \uparrow \text{ for all odd } y \in \mathbb{N} \\ 0 & \text{-- otherwise} \end{cases}$$

The argument given in the prior item will do again here.

**II.5.23** We will show that no Turing machine has this input-output behavior.

$$\text{successor\_decider}(e) = \begin{cases} 1 & \text{-- if } \phi_e(x) = x + 1 \text{ for all } x \in \mathbb{N} \\ 0 & \text{-- otherwise} \end{cases}$$

The function

$$\psi(x, y) = \begin{cases} y + 1 & \text{-- if } \phi_x(x) \downarrow \\ \uparrow & \text{-- otherwise} \end{cases}$$

is intuitively computable by the flowchart on the left of Figure 17 on page 61. So Church's Thesis says that there is a Turing machine that computes it. Let that machine have index  $e$ , so that  $\psi(x, y) = \phi_e(x, y)$ . The  $s$ - $m$ - $n$  Theorem gives a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 17

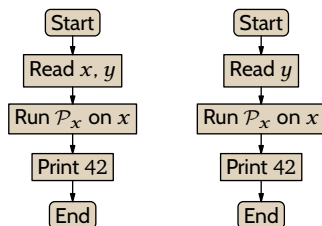


FIGURE 16, FOR QUESTION II.5.21: The machine associated with  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

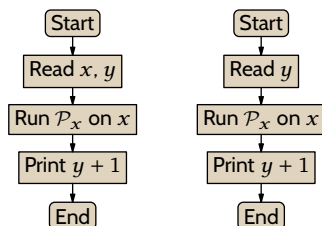


FIGURE 17, FOR QUESTION II.5.23: Sketches of the machine associated with  $\psi = \phi_e$  and the machine associated with  $\phi_{s(e,x)}$ .

on page 61. Then  $\phi_x(x) \downarrow$  if and only if  $\text{successor\_decider}(s(e, x)) = 1$ . The function  $s$  is mechanically computable, so if we could mechanically compute  $\text{successor\_decider}$  then we could mechanically solve the Halting problem. But we can't do that, so we cannot mechanically compute  $\text{successor\_decider}$ .

**II.5.24** We will show that this function is not mechanically computable.

$$\text{converges\_on\_twice\_input\_decider}(e) = \begin{cases} 1 & \text{-- if there is } x \in \mathbb{N} \text{ so that } \phi_e(x) \downarrow \text{ and } \phi_e(2x) \downarrow \\ 0 & \text{-- otherwise} \end{cases}$$

The function

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } \phi_x(x) \downarrow \\ \uparrow & \text{-- otherwise} \end{cases}$$

is intuitively computable by the flowchart on the left of Figure 18 on page 62. So Church's Thesis says that there is a Turing machine that computes it. Let that machine have index  $e$ . Apply the  $s$ - $m$ - $n$  Theorem to get a family of functions parametrized by  $x$ , whose machines are sketched on the right of Figure 18 on page 62. Then  $\phi_x(x) \downarrow$  if and only if  $\text{converges\_on\_twice\_input\_decider}(s(e, x)) = 1$ . So if we could mechanically compute  $\text{converges\_on\_twice\_input\_decider}$  then we could mechanically solve the Halting problem. But we cannot mechanically solve the Halting problem, so we cannot mechanically compute  $\text{converges\_on\_twice\_input\_decider}$ .

**II.5.25** This problem is solvable. We can write a program that inputs  $\text{cantor}(x, y)$ , applies the unpairing function to get  $x$  and  $y$ , and then plugs them into  $ax + by$ , to see if the result equals  $c$ .

**II.5.26** Suppose that we could compute  $K_0$ . That is, suppose that there is a Turing machine  $\mathcal{P}$  such that  $\phi_{\mathcal{P}}(\langle e, x \rangle) = 1$  if  $\phi_e(x) \downarrow$  and  $\phi_{\mathcal{P}}(\langle e, x \rangle) = 0$  if  $\phi_e(x) \uparrow$ . Then we could solve the Halting problem by just applying  $\mathcal{P}$  to the input  $\langle x, x \rangle$ .

**II.5.27**

(A) One such machine takes input  $x$ , uses a Universal Turing machine to run  $\mathcal{P}_x$  on input  $x$ , and then exits.

Figure 19 on page 62 has a flowchart.

(B) For a fixed input  $y$ , one of these two machines works: (1) for all input, print 1, or (2) for all inputs, print 0.

**II.5.28**

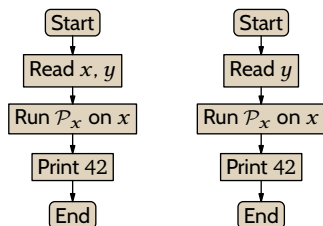


FIGURE 18, FOR QUESTION II.5.24: Sketch of the machine for  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

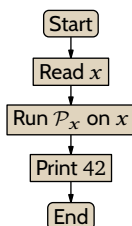


FIGURE 19, FOR QUESTION II.5.27: A single machine that reflects the Halting problem.

- (A) This is solvable. Given the index  $e$ , start by translating it to the set of Turing machine instructions,  $\mathcal{P}_e$  (remember that we are using numberings that are acceptable in the sense defined on page 73). Now count the number of states that appear in that Turing machine.
- (B) This is not solvable; `halts_on_empty` is not mechanically computable. For, consider this function.

$$\psi(x) = \begin{cases} 42 & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{otherwise} \end{cases}$$

It is intuitively mechanically computable, and so by Church's Thesis there is a Turing machine that computes it. Let the index of that machine be  $e$ . Apply the  $s$ - $m$ - $n$  theorem to parametrize  $x$ , giving a uniformly computable family of functions  $\phi_{s(e,x)}$ . (The  $\phi_{s(e,x)}$ 's are functions of no input; that's fine as the associated Turing machine simply does not read any input from the tape.)

Then  $\phi_x(x) \downarrow$  if and only if `halts_on_empty`( $s(e,x)$ ) = 1. If `halts_on_empty` were mechanically computable then we could mechanically solve the Halting problem, so `halts_on_empty` is not mechanically computable.

- (C) This is solvable. From the index  $e$ , use a Universal Turing machine to run  $\mathcal{P}_e$  on input  $e$  for one hundred steps. By the end either it has halted or it hasn't.

**II.5.29** Yes, if  $K$  were finite then it would be computable, since any finite set is computable. For example, we could compute membership in a finite set with a sequence of if-then-else statements.

**II.5.30** Either  $f$  does halt on all inputs, or it doesn't. So there are two possibilities, both of which are computable (either the answer is 'yes' or the answer is 'no'), although we may well not know which one is true.

**II.5.31** Consider a Turing machine that, for any input, just searches for an even number greater than two that is not the sum of two primes. If, on any input, that Turing machine halts then the conjecture is false. Otherwise it is true. So if we determined whether that Turing machine halts then we would have settled the question. See Figure 20 on page 63.

**II.5.32** No. The set of Turing machines is countable. So the set of solvable problems is (a most) countable. Adding one more, by adding the Halting problem, would still mean that we have uncountably many problems but only countably many are solvable. So there would still be problems that are not solvable.

**II.5.33** The set of functions  $f: \mathbb{N} \rightarrow \mathbb{N}$  is the disjoint union of two subsets: the functions that are computable and the functions that are not. We know that the set of computable functions is countable. Because the union

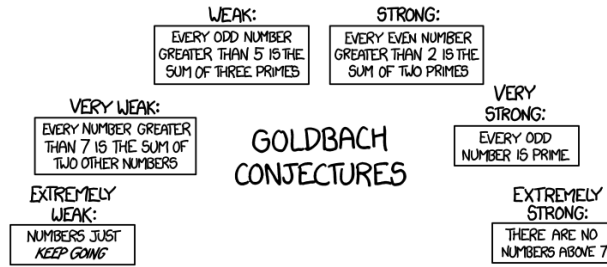


FIGURE 20, FOR QUESTION II.5.31: Goldbach’s conjectures. (Courtesy xkcd.com)

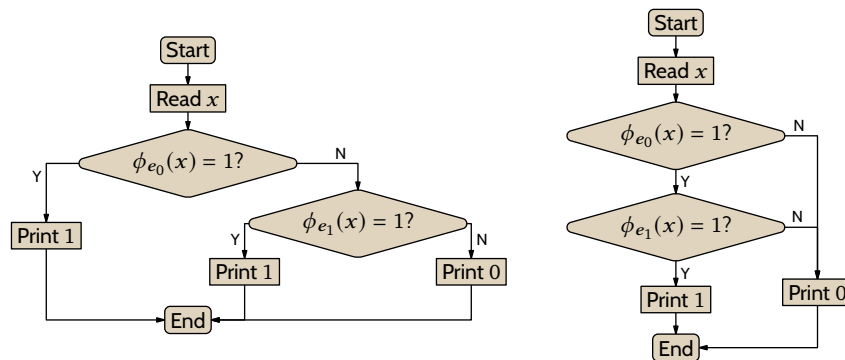


FIGURE 21, FOR QUESTION II.5.35: Sketches of machines to decide union and intersection of decidable languages.

of two countable sets is countable, if the set of non-computable functions were also countable then there would be countably many functions overall, which there are not. Thus the set of non-computable functions is not countable.

II.5.34 Fix an element  $k \in K$ . The range of this function is  $K$ .

$$f(x, n) = \begin{cases} x & \text{– if } \mathcal{P}_x \text{ with input } x \text{ halts in } n \text{ steps} \\ k & \text{– otherwise} \end{cases}$$

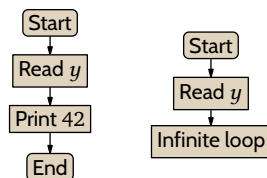
Clearly the function is computable and converges on all inputs.

II.5.35 Start with decidable languages  $\mathcal{L}_0$  and  $\mathcal{L}_1$  and consider the union  $\mathcal{L} = \mathcal{L}_0 \cup \mathcal{L}_1$ . Because they are decidable, there are Turing machines  $\mathcal{P}_{e_0}$  and  $\mathcal{P}_{e_1}$  that decide membership.

$$\phi_{e_0}(x) = \begin{cases} 1 & \text{– if } x \in \mathcal{L}_0 \\ 0 & \text{– if } x \notin \mathcal{L}_0 \end{cases} \quad \phi_{e_1}(x) = \begin{cases} 1 & \text{– if } x \in \mathcal{L}_1 \\ 0 & \text{– if } x \notin \mathcal{L}_1 \end{cases}$$

Note that because they decide membership, these machines halt on all inputs.

- (A) A sketch of the machine that decides membership in  $\mathcal{L}_0 \cup \mathcal{L}_1$  is in Figure 21 on page 63. A number is an element of the union if it is a member of either set. So, given  $x$ , this machine first runs  $\mathcal{P}_{e_0}$  on  $x$ . If that returns 1 then  $x$  is a member of the union, so the new machine outputs 1. Otherwise the new machine runs  $\mathcal{P}_{e_1}$  on  $x$  and returns the result.
- (B) A sketch of the machine that decides membership in  $\mathcal{L}_0 \cap \mathcal{L}_1$  is in Figure 21 on page 63. A number is an element of the intersection if it is a member of both sets. So, given  $x$ , this machine first runs  $\mathcal{P}_{e_0}$  on  $x$ . If that returns 0 then  $x$  is not a member of the intersection, so the new machine outputs 0. Otherwise the new machine runs  $\mathcal{P}_{e_1}$  on  $x$  and returns the result.
- (C) Complement is straightforward. For the complement of  $\mathcal{L}_0$  the new machine just runs  $\mathcal{P}_{e_0}$  and subtracts the result from 1.

FIGURE 22, FOR QUESTION II.6.13: The machines used to show that the set  $\mathcal{I}$  is not trivial.

**II.6.10** Briefly, we have to distinguish between a machine and the function that it computes. Many different machines compute the same function. Rice’s theorem is about those properties of the machine that extend to be properties of the computed functions. For instance, “Does this code compute the squaring function?” is unsolvable, but “Does this code contain the letter K?” is not.

A more sophisticated answer is that two machines ‘extensionally equal’ if they compute the same function, so that for every input either both machines run forever on that input, or they both terminate and output the same value. An ‘extensional property’ of programs is a property that respects extensional equality, i.e., if two machines are extensionally equal then they either both have the property or both not have it. Then Rice’s Theorem says that a computable extensional property of machines either holds of all machines or of none.

Here are some non-extensional properties, to which Rice’s Theorem does not apply: (i) the machine halts within 100 steps (we can always modify a machine to an extensionally equal one that runs longer), (ii) the machine uses fewer than  $n$  memory cells within the first  $m$  steps of execution (we can always modify a machine to an extensionally equal one so that it uses extra memory), and (iii) the machine source contains state  $q_{10}$  (we can add extra states).

**II.6.11** No, it is not an index set. We can write two Turing machines with the same behavior, say, on input  $x$  they return output  $x$ . But one of them takes only a few steps to do this while the other takes 101 steps. The index of the second set is a member of  $\mathcal{I}$  but the index of the first set is not a member.

**II.6.12** The set of indices  $e \in \mathbb{N}$  such that  $\emptyset \subseteq \{x \mid \phi_e(x) \downarrow\}$  is all of  $\mathbb{N}$ . In fact, this problem is solvable, although trivially in the sense that for any  $e$ , the answer is ‘yes’.

**II.6.13** Let  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x \text{ is total}\}$ . We will show that it is a nontrivial index set.

We first show that  $\mathcal{I}$  is not the empty set. Consider the program sketched on the left of Figure 22 on page 64. Clearly this sketch outlines a computation. By Church’s Thesis, there is a Turing machine fitting this sketch. It has an index, and that index is an element of  $\mathcal{I}$ . So  $\mathcal{I}$  is not empty.

Next we show that  $\mathcal{I}$  is not all of  $\mathbb{N}$ . Consider the program sketched on the right of Figure 22 on page 64. By Church’s Thesis, there is a Turing machine fitting this sketch. Its index is not an element of  $\mathcal{I}$ , and so  $\mathcal{I}$  is not all of  $\mathbb{N}$ .

To finish, we show that  $\mathcal{I}$  is an index set. So suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$ , we know that  $\phi_x(a) \downarrow$  for all  $a \in \mathbb{N}$ . Because  $\phi_x \simeq \phi_{\hat{x}}$  we therefore know that  $\phi_{\hat{x}}(a) \downarrow$  for all  $a \in \mathbb{N}$ . Thus,  $\hat{x} \in \mathcal{I}$ . Consequently,  $\mathcal{I}$  is an index set.

By the prior three paragraphs, Rice’s theorem shows that the problem of determining totality, the problem of computing whether, given  $x$ , the Turing machine  $\mathcal{P}_x$  halts on all inputs, is unsolvable.

**II.6.14** Let  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(y) = y^2 \text{ for all } y\}$ . We will show that it is a nontrivial index set.

We first show that  $\mathcal{I}$  is not the empty set. Consider the program sketched on the left of Figure 23 on page 65. Clearly this sketch outlines a computable routine. By Church’s Thesis, there is a Turing machine fitting this sketch. It has an index, and that index is an element of  $\mathcal{I}$ . So  $\mathcal{I}$  is not empty.

Next we show that  $\mathcal{I}$  is not all of  $\mathbb{N}$ . Consider the program sketched on the right of Figure 23 on page 65. By Church’s Thesis, there is a Turing machine fitting this sketch. Its index is not an element of  $\mathcal{I}$ , and so  $\mathcal{I}$  is not all of  $\mathbb{N}$ .

To finish, we show that  $\mathcal{I}$  is an index set. So suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$ , we know that  $\phi_x(y) = y^2$  for all  $y \in \mathbb{N}$ . Because  $\phi_x \simeq \phi_{\hat{x}}$  we therefore know that  $\phi_{\hat{x}}(y) = y^2$  for all  $y \in \mathbb{N}$ . Thus,  $\hat{x} \in \mathcal{I}$ . Consequently,  $\mathcal{I}$  is an index set.

By the prior three paragraphs, Rice’s theorem shows that the problem of determining totality, the problem

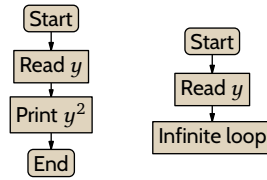


FIGURE 23, FOR QUESTION II.6.14: The machines used to show that the set  $\mathcal{I}$  of indices of the squarer function is not trivial.

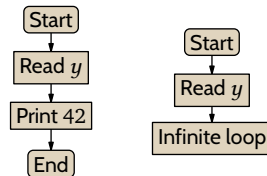


FIGURE 24, FOR QUESTION II.6.15: The machines used to show that the set  $\mathcal{I}$  is not trivial.

of computing whether, given  $x$ , the Turing machine  $\mathcal{P}_x$  halts on all inputs, is unsolvable.

**II.6.15** We will show that  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(y) = \phi_x(y+1) \text{ for some } y \in \mathbb{N}\}$  is a nontrivial index set.

We first show that  $\mathcal{I}$  is not empty. Consider the program sketched on the left of Figure 24 on page 65. It is intuitively computable, so by Church's Thesis there is a Turing machine fitting this sketch. That Turing machine's index is an element of  $\mathcal{I}$ . So  $\mathcal{I}$  is not empty.

Next we argue that  $\mathcal{I} \neq \mathbb{N}$ . Consider the program outlined on the right of Figure 24 on page 65. By Church's Thesis, there is a Turing machine fitting this. Its index is not an element of  $\mathcal{I}$ , and so  $\mathcal{I}$  is not all of  $\mathbb{N}$ .

To finish, we show that  $\mathcal{I}$  is an index set. Suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$ , there exists  $y \in \mathbb{N}$  so that  $\phi_x(y) = \phi_x(y+1)$ . Because  $\phi_x \simeq \phi_{\hat{x}}$ , we know that  $\phi_{\hat{x}}(y) = \phi_{\hat{x}}(y+1)$  for the same  $y$ . Thus,  $\hat{x} \in \mathcal{I}$  also. Therefore  $\mathcal{I}$  is an index set.

Consequently, by Rice's theorem, the problem of determining membership in  $\mathcal{I}$  is unsolvable.

**II.6.16** We will show that  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(5) \uparrow\}$  is a nontrivial index set.

First we show that  $\mathcal{I} \neq \emptyset$ . Consider the program sketched on the left of Figure 25 on page 66. It is intuitively computable, so by Church's Thesis there is a Turing machine fitting this sketch. That machine's index is an element of  $\mathcal{I}$ .

Next we argue that  $\mathcal{I} \neq \mathbb{N}$ . Consider the program outlined on the right of Figure 25 on page 66. By Church's Thesis, there is a Turing machine fitting this. Its index is not an element of  $\mathcal{I}$ .

To finish, we show that  $\mathcal{I}$  is an index set. Suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$ , we know that  $\phi_x(5) \uparrow$ . Because  $\phi_x \simeq \phi_{\hat{x}}$  we have that  $\phi_{\hat{x}}(5) \uparrow$  also. Thus,  $\hat{x} \in \mathcal{I}$ . Therefore  $\mathcal{I}$  is an index set.

Consequently, by Rice's theorem, the problem of determining membership in  $\mathcal{I}$  is unsolvable.

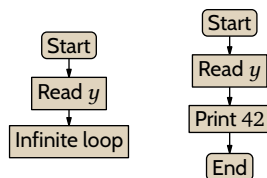
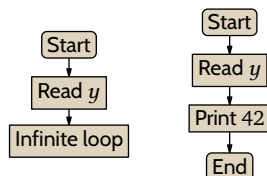
**II.6.17** We will show that  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(y) \uparrow \text{ for all odd } y\}$  is a nontrivial index set. Note that this answer is essentially a copy of the answer to the prior exercise.

First we show that  $\mathcal{I} \neq \emptyset$ . Consider the program sketched on the left of Figure 26 on page 66. It is intuitively computable, so by Church's Thesis there is a Turing machine fitting this sketch. That machine's index is an element of  $\mathcal{I}$ .

Next we argue that  $\mathcal{I} \neq \mathbb{N}$ . Consider the program outlined on the right of Figure 26 on page 66. By Church's Thesis, there is a Turing machine fitting this. Its index is not an element of  $\mathcal{I}$ .

To finish, we show that  $\mathcal{I}$  is an index set. Suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$ , we know that  $\phi_x(y) \uparrow$  for all odd inputs  $y$ . Because  $\phi_x \simeq \phi_{\hat{x}}$  we also know that  $\phi_{\hat{x}}(y) \uparrow$  for all odd  $y$ . Thus,  $\hat{x} \in \mathcal{I}$ . Therefore  $\mathcal{I}$  is an index set.

Consequently, by Rice's theorem, the problem of determining membership in  $\mathcal{I}$  is unsolvable.

FIGURE 25, FOR QUESTION II.6.16: Outlines of the Turing machines used to show that  $\mathcal{I}$  is not trivial.FIGURE 26, FOR QUESTION II.6.17: The machines used to show that  $\mathcal{I}$  is not trivial.

**II.6.18** We will show that  $\mathcal{I} = \{e \in \mathbb{N} \mid \phi_e(x) = x + 1\}$  is a nontrivial index set.

First we show that  $\mathcal{I} \neq \emptyset$ . Consider the program sketched on the left of Figure 27 on page 67. It is intuitively computable, so by Church's Thesis there is a Turing machine fitting this sketch. That machine's index is an element of  $\mathcal{I}$ .

Next we argue that  $\mathcal{I} \neq \mathbb{N}$ . Consider the program outlined on the right of Figure 27 on page 67. By Church's Thesis, there is a Turing machine matching this. Its index is not an element of  $\mathcal{I}$ .

We finish by showing that  $\mathcal{I}$  is an index set. Suppose that  $e \in \mathcal{I}$  and that  $\hat{e}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$ , we know that  $\phi_e(x) = x + 1$  for all inputs  $x$ . Because  $\phi_e \simeq \phi_{\hat{e}}$  we also know that  $\phi_{\hat{e}}(x) = x + 1$  for all  $x$ . Thus,  $\hat{e} \in \mathcal{I}$ . Therefore  $\mathcal{I}$  is an index set.

Consequently, by Rice's theorem, the problem of determining membership in  $\mathcal{I}$  is mechanically unsolvable.

**II.6.19** We will show that  $\mathcal{I} = \{e \in \mathbb{N} \mid \text{both } \phi_e(x) \uparrow \text{ and } \phi_e(2x) \uparrow \text{ for some } x\}$  is a nontrivial index set.

First we show that  $\mathcal{I} \neq \emptyset$ . Consider the program sketched on the left of Figure 28 on page 67. It is intuitively computable, so by Church's Thesis there is a Turing machine fitting this sketch. That machine's index is an element of  $\mathcal{I}$ .

Next we argue that  $\mathcal{I} \neq \mathbb{N}$ . Consider the program outlined on the right of Figure 28 on page 67. By Church's Thesis, there is a Turing machine fitting this. Its index is not an element of  $\mathcal{I}$ .

To finish, we show that  $\mathcal{I}$  is an index set. Suppose that  $e \in \mathcal{I}$  and that  $\hat{e}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$ , we know that for some input  $x$  we have both  $\phi_e(x) \uparrow$  and  $\phi_e(2x) \uparrow$ . Because  $\phi_e \simeq \phi_{\hat{e}}$  we know that  $\phi_{\hat{e}}(x) \uparrow$  and  $\phi_{\hat{e}}(2x) \uparrow$ , for the same  $x$ . Thus,  $\hat{e} \in \mathcal{I}$ . Therefore  $\mathcal{I}$  is an index set.

Consequently, by Rice's theorem, the problem of determining membership in  $\mathcal{I}$  is unsolvable.

### II.6.20

(A) We first state the problem precisely: given an index  $x$ , decide if  $\phi_x(y) \downarrow$  for all  $y \in \mathbb{N}$ . To show that this problem is unsolvable we will verify that  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(y) \downarrow \text{ for all } y \in \mathbb{N}\}$  is a nontrivial index set.

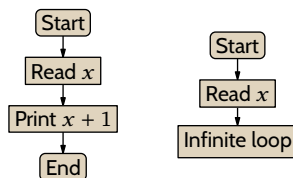
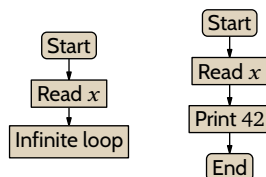
It is not empty because we can write a program that returns its input, and then by Church's Thesis there is a Turing machine with that behavior. That machine's index is a member of  $\mathcal{I}$ .

It is not equal to  $\mathbb{N}$  because we can write a program that does not halt on any input. By Church's Thesis there is a Turing machine with that behavior and the machine's index is not a member of  $\mathcal{I}$ .

We finish verifying that  $\mathcal{I}$  is an index set. Suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$  we know that  $\phi_x(y) \downarrow$  for all  $y \in \mathbb{N}$ . Because the two have the same behavior,  $\phi_{\hat{x}}(y) \downarrow$  for all  $y$  also. Therefore  $\hat{x} \in \mathcal{I}$  and  $\mathcal{I}$  is an index set.

(B) This set is the complement of the one in the prior item. Exercise 6.29 shows that the complement of an index set is also an index set. But we don't have that result in the main section body so we will verify that the given problem is unsolvable without reference to it.

The problem is: given an index  $x$ , decide if  $\phi_x(y) \uparrow$  for some  $y \in \mathbb{N}$ . To show that this is unsolvable we

FIGURE 27, FOR QUESTION II.6.18: The machines used to show that  $\mathcal{I}$  is not trivial.FIGURE 28, FOR QUESTION II.6.19: The machines used to show that  $\mathcal{I}$  is not trivial.

will verify that  $\mathcal{I} = \{x \in \mathbb{N} \mid \phi_x(y) \uparrow \text{ for some } y \in \mathbb{N}\}$  is a nontrivial index set.

The set  $\mathcal{I}$  is not empty because we can write a program that does not halt on any input and then by Church's Thesis there is a Turing machine with that behavior. That machine's index is a member of  $\mathcal{I}$ .

The set is not equal to  $\mathbb{N}$  because we can write a program that returns its input and then by Church's Thesis there is a Turing machine with that behavior. That machine's index is not a member of  $\mathcal{I}$ .

We finish verifying that  $\mathcal{I}$  is an index set. Suppose that  $x \in \mathcal{I}$  and that  $\hat{x}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$  we know that  $\phi_x(y) \uparrow$  for at least one  $y \in \mathbb{N}$ . Because the two have the same behavior,  $\phi_{\hat{x}}(y) \uparrow$  for at least one  $y$  also. Therefore  $\hat{x} \in \mathcal{I}$  and  $\mathcal{I}$  is an index set.

**II.6.21** Each answer just gives the numbered fill-ins.

- (A) (1)  $\phi_e(x) \downarrow$  for all inputs  $x$  that are a multiple of 5. (2) and (3) See Figure 29 on page 68. (4)  $\phi_e(5k) \downarrow$  for all  $k \in \mathbb{N}$ . (4)  $\phi_e(5k) \downarrow$  for all  $k \in \mathbb{N}$  also.
- (B) (1)  $\phi_e(x) = 7$  for some input  $x$ . (2) and (3) See Figure 30 on page 68. (4)  $\phi_e(x) = 7$  for some input  $x \in \mathbb{N}$ . (4)  $\phi_e(x) = 7$  for the same  $x$ .

**II.6.22** For both items we will use Rice's Theorem.

- (A) We will show that this is a nontrivial index set:  $\mathcal{I} = \{e \in \mathbb{N} \mid \mathcal{P}_e \text{ accepts an infinite language}\}$ .

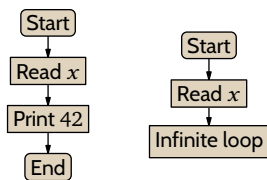
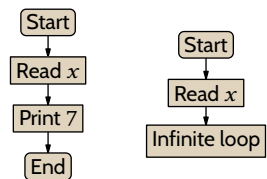
First,  $\mathcal{I}$  is nonempty because the set  $\mathcal{L} = \mathbb{B}^*$  is infinite and we can easily write a Turing machine to output 1 on all inputs, which accepts this subset of  $\mathbb{B}$ . The index of that Turing machine is an element of  $\mathcal{I}$ .

Second,  $\mathcal{I}$  does not equal  $\mathbb{N}$  because we can easily write a Turing machine to output 0 on all inputs, which accepts the empty subset of  $\mathbb{B}$ , which is not infinite. The index of that Turing machine is not an element of  $\mathcal{I}$ .

Finally, to verify that  $\mathcal{I}$  is an index set, suppose that  $x \in \mathcal{I}$  and that  $\hat{x} \in \mathbb{N}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$  there is an infinite  $\mathcal{L} \subseteq \mathbb{B}^*$  so that  $\mathcal{P}_x$  accepts  $\mathcal{L}$ . Because  $\phi_x \simeq \phi_{\hat{x}}$  the set of bit strings for which  $\mathcal{P}_{\hat{x}}$  outputs 1 equals the set for which  $\mathcal{P}_x$  outputs 1, and so  $\mathcal{P}_{\hat{x}}$  also accepts an infinite set of bit strings. Thus  $\hat{x} \in \mathcal{I}$ , and  $\mathcal{I}$  is an index set.

- (B) We will show that  $\mathcal{I} = \{e \in \mathbb{N} \mid \mathcal{P}_e \text{ accepts } 101\}$  is a nontrivial index set. First,  $\mathcal{I}$  is nonempty because we can write a Turing machine to output 1 on all inputs, which accepts the string 101. The index of that Turing machine is an element of  $\mathcal{I}$ . Second,  $\mathcal{I}$  does not equal  $\mathbb{N}$  because we can write a Turing machine to output 0 on all inputs, which does not accept the string 101. The index of that Turing machine is not an element of  $\mathcal{I}$ .
- Finally, to verify that  $\mathcal{I}$  is an index set, suppose that  $x \in \mathcal{I}$  and that  $\hat{x} \in \mathbb{N}$  is such that  $\phi_x \simeq \phi_{\hat{x}}$ . Because  $x \in \mathcal{I}$  the machine  $\mathcal{P}_x$  accepts 101. Because  $\phi_x \simeq \phi_{\hat{x}}$  the machine  $\mathcal{P}_{\hat{x}}$  also accepts 101. Thus  $\hat{x} \in \mathcal{I}$ , and  $\mathcal{I}$  is an index set.

**II.6.23** We will use Rice's Theorem so consider  $\mathcal{I} = \{e \mid \phi_e(x) \downarrow \text{ for some } x \in \mathbb{N}\}$ . This set is not empty because we can write a Turing machine that halts on all inputs. This set is not equal to  $\mathbb{N}$  because we can write a Turing machine that fails to halt for any input.

FIGURE 29, FOR QUESTION II.6.21: The machines used to argue that  $\mathcal{I}$  is not trivial.FIGURE 30, FOR QUESTION II.6.21: The machines used to argue that  $\mathcal{I}$  is not trivial.

To show this is an index set suppose that  $e \in \mathcal{I}$  and that the number  $\hat{e}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$  there is a number  $x_0$  such that  $\phi_e(x_0) \downarrow$ . Then  $\phi_{\hat{e}}(x_0) \downarrow$  also, because  $\phi_e \simeq \phi_{\hat{e}}$ . Thus  $\hat{e} \in \mathcal{I}$  and this is an index set.

**II.6.24** We can produce two Turing machines,  $\mathcal{P}_e$  and  $\mathcal{P}_{\hat{e}}$  that have the same behavior,  $\phi_e \simeq \phi_{\hat{e}}$ , but  $e \in \mathcal{J}$  and  $\hat{e} \notin \mathcal{J}$ . Here are two such machines.

$$\mathcal{P}_e = \{q_0 \text{BB}q_0, q_0 11q_0, q_1 \text{BB}q_0\}$$

$$\mathcal{P}_{\hat{e}} = \{q_0 \text{BB}q_0, q_0 11q_0\}$$

**II.6.25** Briefly, we cannot tell anything by whether the machine ‘goes on’. It can go on to do quite complex things.

For more, we can reduce this problem to the Halting problem. Consider this  $\psi: \mathbb{N}^2 \rightarrow \mathbb{N}$ .

$$\psi(x, y) = \begin{cases} 42 & \text{– if machine } \mathcal{P}_x \text{ halts on } x \text{ and } y = \varepsilon \\ \uparrow & \text{– otherwise} \end{cases}$$

(We can take this machine to have tape alphabet  $\Sigma = \{B, 1\}$ .) This is intuitively computable so Church’s Thesis gives that there is a Turing machine with this behavior, and we can assume it has index  $e$ .

Apply the  $s$ - $m$ - $n$  Theorem to parametrize  $x$ , giving a family of machines and associated computable functions as here.

$$\phi_{s(e,x)}(y) = \begin{cases} 42 & \text{– if machine } \mathcal{P}_x \text{ halts on } x \text{ and } y = \varepsilon \\ \uparrow & \text{– otherwise} \end{cases}$$

Then  $\phi_x(x) \downarrow$  if and only if the function  $\phi_{s(e,x)}$  halts only on  $y = \varepsilon$ . So if we could check the latter mechanically then we could mechanically solve the Halting problem, which of course we cannot.

**II.6.26** There are of course many answers; one is  $\mathcal{I}_0 = \{e \mid \mathcal{P}_e \text{ solves the Halting problem}\}$ .

**II.6.27** We will call each set  $\mathcal{I}$ .

- (A) Suppose that  $e \in \mathcal{I}$  and suppose that  $\hat{e}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$  the machine  $\mathcal{P}_e$  halts on at least five inputs, that is, the function  $\phi_e$  converges on at least five inputs. Because  $\phi_e \simeq \phi_{\hat{e}}$ , the function  $\phi_{\hat{e}}$  also halts on at least five inputs, namely, the same five. Thus  $\hat{e} \in \mathcal{I}$  also.
- (B) Suppose that  $e \in \mathcal{I}$  and suppose also that  $\hat{e} \in \mathbb{N}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$  the function  $\phi_e$  is one-to-one. Because  $\phi_e \simeq \phi_{\hat{e}}$ , the function  $\phi_{\hat{e}}$  is also one-to-one. Thus  $\hat{e} \in \mathcal{I}$ .

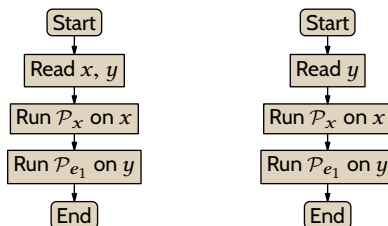


FIGURE 31, FOR QUESTION II.6.30: Sketches of machines for the proof of half of Rice's Theorem.

- (c) Suppose that  $e \in \mathcal{I}$  and also that  $\hat{e} \in \mathbb{N}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$  the function  $\phi_e$  is either total or else  $\phi_e(3) \uparrow$ . In the first case, that  $\phi_e$  is total, because  $\phi_e \simeq \phi_{\hat{e}}$ , the function  $\phi_{\hat{e}}$  is also total. In the case that  $\phi_e(3) \uparrow$ , because they have the same behavior,  $\phi_{\hat{e}}(3) \uparrow$  also. In either case,  $\hat{e} \in \mathcal{I}$ .

**II.6.28**

- (A) The relation  $\simeq$  is reflexive because a Turing machine  $\mathcal{P}_e$  has the same behavior as itself:  $\phi_e \simeq \phi_e$ . It is symmetric because if  $\mathcal{P}_e$  has the same behavior as  $\mathcal{P}_{\hat{e}}$  then the converse also holds. Transitivity is just as clear.
- (B) Each equivalence class is a set of integers  $e \in \mathbb{N}$ . Two integers are together in a class,  $e, \hat{e} \in \mathcal{C}$  if the input-output behaviors of the associated Turing machines  $\mathcal{P}_e$  and  $\mathcal{P}_{\hat{e}}$  are the same,  $\phi_e \simeq \phi_{\hat{e}}$ . (As an example, one class contains the indices of all Turing machines that double the input,  $\mathcal{C} = \{e \in \mathbb{N} \mid \phi_e(x) = 2x \text{ for all } x\}$ .)
- (c) We follow the hint. Suppose that  $\mathcal{I}$  is an index set and that  $e \in \mathcal{C}$  is an element of  $\mathcal{I}$ . If  $\hat{e} \in \mathcal{C}$  then  $\phi_{\hat{e}} \simeq \phi_e$  because  $\mathcal{C}$  is an equivalence class for the relation  $\simeq$ . But  $\mathcal{I}$  is an index set so  $\hat{e} \in \mathcal{I}$  also.

**II.6.29**

- (A) Let  $\mathcal{I}$  be an index set and consider the complement,  $\mathcal{I}^c$ . Suppose that  $e \in \mathcal{I}^c$  and suppose also that  $\hat{e} \in \mathbb{N}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . We will show that  $\hat{e} \in \mathcal{I}^c$ .  
By the definition of set complement, either  $\hat{e} \in \mathcal{I}$  or  $\hat{e} \in \mathcal{I}^c$ . If  $\hat{e} \in \mathcal{I}$  then because  $\phi_e \simeq \phi_{\hat{e}}$  and because  $\mathcal{I}$  is an index set, we have that  $e \in \mathcal{I}$  also. This contradicts that  $e \in \mathcal{I}^c$  and therefore  $\hat{e} \in \mathcal{I}^c$ .
- (B) Fix a collection  $\mathcal{I}_j$  of index sets and consider the union  $\mathcal{I} = \cup_j \mathcal{I}_j$ . Suppose that  $e \in \mathcal{I}$  and suppose also that  $\hat{e} \in \mathbb{N}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . We will show that  $\hat{e} \in \mathcal{I}$ .  
The number  $e$  is an element of the union  $\cup_j \mathcal{I}_j = \mathcal{I}$  so it is an element of some  $\mathcal{I}_{j_0}$ . Because  $\phi_e \simeq \phi_{\hat{e}}$  and because  $\mathcal{I}_{j_0}$  is an index set, the number  $\hat{e}$  is an element of  $\mathcal{I}_{j_0}$ , and is therefore an element of the union  $\mathcal{I}$ . Thus  $\mathcal{I}$  is an index set.
- (c) Let  $\mathcal{I}_j$  be a collection of index sets and let  $\mathcal{I} = \cap_j \mathcal{I}_j$ . Suppose that  $e \in \mathcal{I}$ , and also that  $\hat{e} \in \mathbb{N}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . We will show that  $\hat{e} \in \mathcal{I}$ .  
Since  $e$  is an element of the intersection  $\cap_j \mathcal{I}_j = \mathcal{I}$ , it is an element of each set  $\mathcal{I}_j$ . Because  $\phi_e \simeq \phi_{\hat{e}}$  and because  $\mathcal{I}_j$  is an index set, the number  $\hat{e}$  is an element of  $\mathcal{I}_j$  also. As  $\hat{e}$  is an element of each  $\mathcal{I}_j$ , it is an element of the intersection  $\mathcal{I}$ . Thus  $\mathcal{I}$  is an index set.

**II.6.30**

The differences between this case and the one in the section are very small. The proof there starts by considering an index set  $\mathcal{I}$ . It fixes an index  $e_0 \in \mathbb{N}$  so that  $\phi_{e_0}(y) \uparrow$  for all inputs  $y \in \mathbb{N}$ . It then does the  $e_0 \notin \mathcal{I}$  case. So what's left is to do  $e_0 \in \mathcal{I}$ . Here's a difference: because  $\mathcal{I}$  is nontrivial, it does not equal  $\mathbb{N}$  so there is some  $e_1 \notin \mathcal{I}$ . Because  $\mathcal{I}$  is an index set,  $\phi_{e_0} \neq \phi_{e_1}$ . Thus there is an input  $y$  such that  $\phi_{e_1}(y) \downarrow$ .

Consider the flowchart on the left of Figure 31 on page 69. By Church's Thesis there is a Turing machine with that behavior; call it  $\mathcal{P}_e$ . Apply the  $s$ - $m$ - $n$  theorem to parametrize  $x$ , resulting in the uniformly computable family of functions  $\phi_{s(e,x)}$ , whose computation is outlined on the right of Figure 31 on page 69. We've constructed the machines so that if  $\phi_x(x) \uparrow$  then  $\phi_{s(e,x)} \simeq \phi_{e_0}$  and thus  $s(e,x) \in \mathcal{I}$ . Further, if  $\phi_x(x) \downarrow$  then  $\phi_{s(e,x)} \simeq \phi_{e_1}$  and thus  $s(e,x) \notin \mathcal{I}$ . Therefore if  $\mathcal{I}$  were mechanically computable, so that we could effectively check whether  $s(e,x) \in \mathcal{I}$ , then we could mechanically solve the Halting problem. (This part of the argument is like the one in the section, but with the cases of  $s(e,x) \in \mathcal{I}$  and  $s(e,x) \notin \mathcal{I}$  reversed.)

**II.7.7**

Not completely right. It is right that a set is computably enumerable if it can be enumerated by a Turing machine. That is, a Turing machine exists that, when started on a blank tape, will print all of the set members on the tape. (This machine just outputs  $f(0), f(1), \dots$ . It may run forever if there are infinitely

many elements in the set.) However, the “but is not computable” is wrong. Any computable set is computably enumerable (although there are computably enumerable sets that are not computable). For instance, the set of even numbers is computably enumerable and also computable.

**II.7.8** For each part we will produce a function  $f: \mathbb{N} \rightarrow \mathbb{N}$  that is effective.

- (A) The identity function,  $f(x) = x$ , works.
- (B) The function  $f(x) = 2x$  comes to mind.
- (C)  $f(x) = x^2$
- (D) This will do.

$$f(x) = \begin{cases} 5 & \text{– if } x = 0 \\ 7 & \text{– if } x = 1 \\ 11 & \text{– otherwise} \end{cases}$$

**II.7.9** For each item we produce an effective function  $f: \mathbb{N} \rightarrow \mathbb{N}$ .

- (A) The association  $n \mapsto n$ -th prime is a function. Clearly we can write a program that computes it, so by Church’s Thesis it is effective.
- (B) To compute this function one approach is brute force: generate the  $n$ -th one by searching through the natural numbers,  $0, 1, \dots$ , checking each to see if its digits appear in non-increasing order. Output the  $n$ -th such number.

**II.7.10** The first is computable, since to decide if a number  $e$  is a member of this set we can run a simulation for twenty steps. The second is computably enumerable but not computable. It is computably enumerable because we can dovetail computations for the indices  $e \in \mathbb{N}$ . It is not computable because clearly if we could solve it then we could solve the problem of deciding membership in  $\{e \mid \phi_e(y) \downarrow\}$ , but we cannot solve that problem by Rice’s Theorem.

**II.7.11**

- (A) Decidable. Indices are source-equivalent: there is a program that takes in the index  $e$  and puts out the set of instructions for  $\mathcal{P}_e$ . With that set of instructions, just scan it for  $q_4$ .
- (B) Semidecidable but not decidable. To show it is computably enumerable, use dovetailing on the indices  $e \in \mathbb{N}$  to run  $\mathcal{P}_e$  on input 3. When a the machines halts, enumerate its  $e$  into the set. As to being not computable, we know from the prior section that this is not a computable set.
- (C) Computable. Given  $e$ , decide if it is in the set by running  $\mathcal{P}_e$  on 3 for the hundred steps.

**II.7.12** The function is an enumeration but unless it is computable, it is not a computable enumeration. This can happen; an example of a countable set that is not computably enumerable is  $K^c$ .

**II.7.13** Briefly, we dovetail. First run  $\mathcal{P}_0$  on input 5 for one step. Then run  $\mathcal{P}_0$  on input 5 for a second step and  $\mathcal{P}_1$  on input 5 for one step. Then run  $\mathcal{P}_0$  for a third step, along with  $\mathcal{P}_1$  for its second step and  $\mathcal{P}_2$  on input 5 for one step. Continue in this way. The enumerating function  $f: \mathbb{N} \rightarrow \mathbb{N}$  is:  $f(0)$  is the index of the first such computation to halt,  $f(1)$  is the index of the second, etc. (Note that  $A_5$  is infinite since any finite set is computable, so computation of any  $f(i)$  will halt in a finite number of steps.)

**II.7.14** Dovetail. Start by running  $\mathcal{P}_0$  on input 2 for one step. Then run  $\mathcal{P}_0$  on input 2 for a second step and  $\mathcal{P}_1$  on input 2 for one step. Next, run  $\mathcal{P}_0$  for a third step, along with  $\mathcal{P}_1$  for its second step and  $\mathcal{P}_2$  on input 2 for one step. Continue in this way. Then  $f(0)$  is the index of the first such computation to halt and output 4,  $f(1)$  is the index of the second, etc. (This set is infinite since any finite set is computable and this set is clearly not computable. Thus computation of any  $f(i)$  will halt in a finite number of steps.)

**II.7.15** The complement of the Halting problem set,  $K^c$ , is countable since it is a subset of  $\mathbb{N}$ . Thus it has an enumeration, a function with domain  $\mathbb{N}$  whose range is this set. But Corollary 7.6 shows that it has no computable enumeration.

**II.7.16** There are of course many correct answers. One answer is to take  $K$  as the first set. For the second, take  $K$  with its smallest element omitted and for the third, take  $K$  with the smallest two elements omitted.

A perhaps less wiseguy-ish answer is to take  $K$  along with some sets that we used as examples in the prior section, such as  $\{e \in \mathbb{N} \mid \phi_e(3) \downarrow\}$  and  $\{e \in \mathbb{N} \mid \text{there is } x \in \mathbb{N} \text{ so that } \phi_e(x) = 7\}$ .

**II.7.17**

- (A) Enumerate it by dovetailing. The difference from some of the dovetail examples is that here there are two numbers,  $e$  and  $x$ . So recall Cantor's pairing function.

Number	0	1	2	3	4	5	6	...
Pair	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 1, 0 \rangle$	$\langle 0, 2 \rangle$	$\langle 1, 1 \rangle$	$\langle 2, 0 \rangle$	$\langle 0, 3 \rangle$	...

To dovetail, first run  $\mathcal{P}_0$  on input 0 for one step. Then run  $\mathcal{P}_0$  on input 0 for a second step along with  $\mathcal{P}_0$  on input 1 for one step. Next, run  $\mathcal{P}_0$  on input 0 for its third step, and  $\mathcal{P}_0$  on input 1 for its second step, and  $\mathcal{P}_1$  on input 0 for one step. The enumerating function outputs  $\langle e, x \rangle$  when  $\mathcal{P}_e$  halts on  $x$ .

- (B) This is immediate from the second item of Lemma 7.3.

**II.7.18** The two collections are not disjoint. Rather, the collection of computable sets is a subset of the collection of computably enumerable sets.

Further, clearly the collection  $\{W_e \mid e \in \mathbb{N}\}$  is countable. Since there are uncountably many subsets of  $\mathbb{N}$  (because the power set of  $\mathbb{N}$  has cardinality greater than  $\mathbb{N}$ ), the computably enumerable sets do not include every subset of  $\mathbb{N}$ .

**II.7.19** The computable sets are also computably enumerable so we will have shown that Tot is neither computable nor computably enumerable if we have shown that it is not computably enumerable.

So, assume that  $\text{Tot} = \{e \mid \phi_e(y) \downarrow \text{ for all } y\}$  is computably enumerable, enumerated by  $f: \mathbb{N} \rightarrow \mathbb{N}$ . Following the hint, that gives a table like the one starting Section 1 on Unsolvability, because the  $i, j$  entry  $\phi_i(j)$  is sure to halt. Diagonalize: consider  $h: \mathbb{N} \rightarrow \mathbb{N}$  given by  $h(e) = 1 + \phi_e(e)$ . It is clearly effective, total, and unequal to any member of Tot, which is a contradiction.

**II.7.20** We can easily show that the set  $S = \{e \mid \phi_e(3) \uparrow\}$  is unsolvable, for instance with Rice's theorem. Clearly the complement,  $S^c = \{e \mid \phi_e(3) \downarrow\}$ , is computably enumerable. If  $S$  were also computably enumerable then by Lemma 7.5 that would contradict that it is not computable.

**II.7.21** Yes, the Halting problem set  $K$  fits both criteria. Three other examples are the set  $\{e \in \mathbb{N} \mid \phi_e(3) \downarrow\}$ , the set  $\{e \in \mathbb{N} \mid \text{there is } x \in \mathbb{N} \text{ so that } \phi_e(x) = 7\}$ , and  $\{e \in \mathbb{N} \mid \phi_e(x) = 2x \text{ for all } x \in \mathbb{N}\}$ .

**II.7.22** There are only countably many Turing machines to do the enumeration.

**II.7.23** The answer to the second effectively answers the first, but we will do both items anyway.

- (A) Let  $S = \{s_0, \dots, s_{k-1}\}$  for  $k \in \mathbb{N}$ . Then this function enumerates  $S$ .

$$f(x) = \begin{cases} 1 & \text{if } x = s_0 \text{ or } \dots \text{ or } x = s_{k-1} \\ 0 & \text{otherwise} \end{cases}$$

It is clearly computable.

- (B) The algorithm inputs a set of numbers  $S = \{s_0, \dots, s_{k-1}\}$ . It returns a function, the one in the prior item. This function is clearly computable.

Another approach is to start with the number  $k = |S|$ . Define a function with  $k + 1$  inputs,  $x_0, \dots, x_{k-1}, y$  that tests whether  $y$  equals any of the  $x_i$ 's, and returns 1 if it does or 0 if it does not. Where  $S = \{s_0, \dots, s_{k-1}\}$ , apply the  $s$ - $m$ - $n$  Theorem to freeze the value of each  $x_i$  to be  $s_i$ . That's the desired function.

**II.7.24** Let the computably enumerable set be  $W$  and fix a computable enumeration  $\phi(0), \phi(1), \dots$ . We will define an  $S \subseteq W$  that is computable. The first element of  $S$  is  $s_0 = \phi(0)$ . The second element  $s_1$  is the first member of the enumeration that is larger than  $s_0$ ; such a number must eventually appear because  $W$  is infinite. In general  $s_i$  is the first element of the enumeration that is larger than  $s_{i-1}$  (as with  $s_1$ , one must eventually appear).

The set  $S$  is computably enumerable because we have just described a mechanical procedure to enumerate it. It is computable because given a number  $n \in \mathbb{N}$ , to determine whether it is an element of  $S$  just wait until either it is enumerated into  $S$  or a number larger than it is enumerated into  $S$ , which shows  $n \notin S$ .

**II.7.25**

- (A) Arguing by Church's Thesis: compute  $\text{steps}(e)$  by running  $\mathcal{P}_e$  on input  $e$  and if it ever halts, outputting the number of steps.

- (B) If steps were total in addition to being computable then we could solve the Halting problem. To decide whether  $e \in K$ , first compute  $\text{steps}(e)$ . Then run  $\mathcal{P}_e$  on input  $e$  for  $\text{steps}(e)$ -many steps. If this computation has not halted by then, it will never halt.

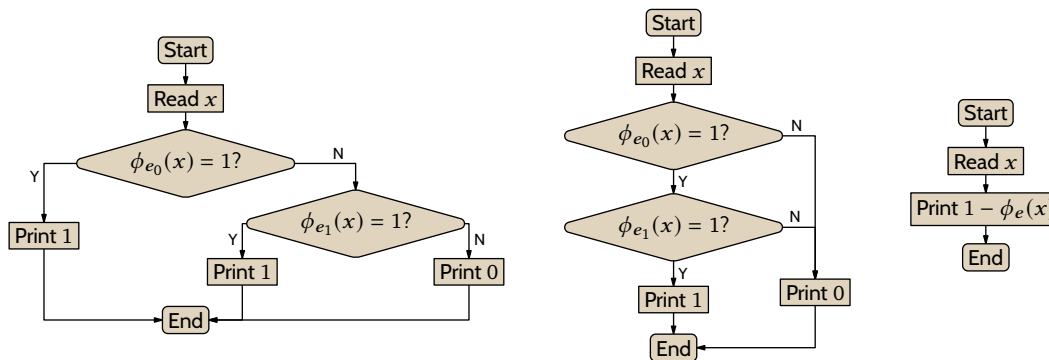


FIGURE 32, FOR QUESTION II.7.30: Machines to decide union, intersection, and complement of computable sets.

(c) This is the same as the prior argument, but with  $f$  in place of steps.

**II.7.26** Fix an  $\hat{r} \in R$ . Then this function

$$g(i) = \begin{cases} f(i) & \text{if } f(i) \downarrow \\ \hat{r} & \text{otherwise} \end{cases}$$

is total, computable, and enumerates  $R$ .

**II.7.27** Suppose that  $S$  is computable, so that its characteristic function

$$\mathbb{1}_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if not} \end{cases}$$

is a computable function. To enumerate  $S$  in increasing order the idea is to run  $\mathbb{1}_S(0), \mathbb{1}_S(1), \dots$ , each of which must converge because the function is computable, and then the first element of  $S$  is the smallest, the second element is the second smallest, etc.

More formally, we define the enumerating function by: where  $s_0$  is the smallest element of  $S$  then  $f(0) = s_0$ , and where we have values for  $f(0), \dots, f(k)$  then the value for  $f(k+1)$  is the smallest number  $x$  such that  $x > f(k)$  and  $\mathbb{1}_S(x) = 1$ . Because  $S$  is infinite, there is always such an  $x$ .

For the other direction, assume that  $S$  is an infinite set that is enumerable in increasing order, by the computable function  $h$ . Given  $x$ , to decide whether  $x \in S$ , compute  $h(0), h(1), \dots$  until one of the outputs is  $x$  (in which case  $x \in S$ ) or one of the outputs is a number larger than  $x$  (in which case  $x \notin S$ ).

**II.7.28** One direction is easy. If a set is computably enumerable without repetition then it must be infinite, simply because if it were finite then the enumeration would run out of new elements to output.

The other direction is about memoizing. Let the set  $S$  be infinite and computably enumerable with enumerating function  $f$ . We will produce an enumerating function  $g$  that is one-to-one. Define  $g(0) = f(0)$ . For each  $i \in \mathbb{N}$  define  $g(i+1)$  to be the first element of  $S$  enumerated by  $f$  after it enumerates  $g(i)$  that is not an element of  $\{g(0), \dots, g(i)\}$ .

**II.7.29** The set of ordered pairs  $\{\langle a, b \rangle \mid a \in K \text{ and } b \in K^c\}$  is not computably enumerable, or else  $K^c$  would be computably enumerable. Similarly it is not co-computably enumerable.

**II.7.30**

(A) No. The set  $\mathbb{N}$  is obviously computable. Its subset  $K$  is not.

(B) Yes. Let  $S_0 \subseteq \mathbb{N}$  and  $S_1 \subseteq \mathbb{N}$  be computable via the functions  $\phi_{e_0}$  and  $\phi_{e_1}$ . A sketch of the machine to compute the union is in Figure 32 on page 72.

(C) Yes. A sketch of the machine to compute the intersection is in Figure 32 on page 72.

(D) Yes. A sketch of the machine to compute the complement is in Figure 32 on page 72.

**II.7.31**

(A) Yes. Let  $W_{e_0}, W_{e_1} \subseteq \mathbb{N}$  be computably enumerable, via  $\phi_{e_0}, \phi_{e_1} : \mathbb{N} \rightarrow \mathbb{N}$ . Dovetail enumerations of the two sets. That is, run  $\mathcal{P}_{e_0}$  and  $\mathcal{P}_{e_1}$  on input 0 for a step each. Then, run  $\mathcal{P}_{e_0}$  and  $\mathcal{P}_{e_1}$  on input 0 for a second

step, and also run  $\mathcal{P}_{e_0}$  and  $\mathcal{P}_{e_1}$  on input 1 for a step each. Continue in this way, and when a number is enumerated into either of  $W_{e_0}$  or  $W_{e_1}$ , enumerate it into their union.

- (B) Yes. As in the prior item, dovetail enumerations of the two sets. When a number has been enumerated into both of  $W_{e_0}$  or  $W_{e_1}$ , enumerate it into their intersection.
- (C) No. The Halting problem set  $K$  is computably enumerable but its complement is not.

**II.7.32** Let  $S$  be the domain of a partial computable function  $f$ . We will produce a partial computable function  $g$  whose range is  $S$  by dovetailing.

First run the computation of  $f(0)$  for a step. Then run the computation of  $f(0)$  for a second step and the computation of  $f(1)$  for a step. Next run the computation of  $f(0)$  for its third step, run the computation of  $f(1)$  for its second step, and run the computation of  $f(2)$  for a step. Continue in this way. Define  $g(0)$  to be the first number  $i$  where the computation of  $f(i)$  halts, define  $g(1)$  to be the next number  $j$  where the computation of  $f(j)$  halts, etc. Clearly  $g$  is partial computable and has range  $S$ .

The other direction works much the same way. We are given a partial computable  $g$  whose range is  $S$  and we will produce a partial computable  $f$  whose domain is  $S$  by dovetailing computations of  $g$ . First run  $g(0)$  for a step. Then run the computation of  $g(0)$  for a second step and the computation of  $g(1)$  for a step. Next run  $g(0)$  for its third step,  $g(1)$  for its second step, and  $g(2)$  for a step. Continue in this way. When a computation halts,  $g(i) = k$ , define  $f(k)$  to have some nominal value, such as 42. Clearly  $f$  is partial computable and has domain  $S$ .

**II.8.12**

- (A) This is wrong. For instance, the empty set is reducible to the Halting problem set,  $\emptyset \leq_T K$ , but while a decider for  $\emptyset$  is trivial, there is no computable decider for  $K$ .

The correct statement is the opposite: if  $A \leq_T B$  then a decider for  $B$  can be used to decide the set  $A$ .

- (B) This is wrong. As in the prior item, if  $A$  is the empty set then it is computable. Then  $A \leq_T B$  where  $B$  equals the Halting problem set  $K$ , but  $B$  is not decidable. The other way around is right: if  $B$  is decidable then  $A$  is decidable also.
- (C) True. If  $A \leq_T B$  then  $\phi_e^B = \mathbb{1}_A$  for some index  $e \in \mathbb{N}$ . If we could compute  $B$  then we could use that ability to compute  $A$ . So if  $A$  is uncomputable then  $B$  is also uncomputable.

**II.8.13** A decider is a Turing machine (whose computed function is the characteristic function of some set). An oracle is an addition to a Turing machine, a set that the machine can query.

**II.8.14** First of all, they are fascinating and fun. Is that not enough?

Beyond that, they help us understand the relationships among problems. In the Complexity chapter we will expand on the idea to help understand how many everyday, practical, problems are related. This will include trying to understand which are easy and which are hard.

**II.8.15** Their answer captures the spirit of the reason for considering oracle computations. But it is lacking in two ways. The first is a technicality: there are oracles such as the empty set, where computations relative to that oracle can only solve problems that you could solve already.

The second lack is that their answer doesn't refer to the mechanism at all. That is, the oracle set doesn't solve the problems. We might informally speak of an oracle as solving problems but for a definition we should say that we give a machine access to answers about membership in the set and then the input-output behavior of the machine is the solution to those problems.

**II.8.16** We take the first question to ask: is there a most-powerful oracle, one that solves every problem? The answer is no. Given an oracle  $X \subseteq \mathbb{N}$ , the problem of deciding membership in  $K^X$ , the Halting problem in that oracle, is not solvable from  $X$ .

The second question asks whether, given a set  $S \subseteq \mathbb{N}$ , there is an oracle that suffices to determine membership in  $S$ . The answer is yes; the oracle  $S$  will do.

**II.8.17** Each oracle can only compute answers to some problems. But for each oracle there are problems that it cannot solve.

**II.8.18** In an oracle computation that halts, there are indeed only finitely many calls to the oracle. But the calls when the computation's input is 0 may well differ from the calls when the computation's input is 1.

**II.8.19** See Figure 33 on page 74.

- (A) The flowchart on the left is straightforward.

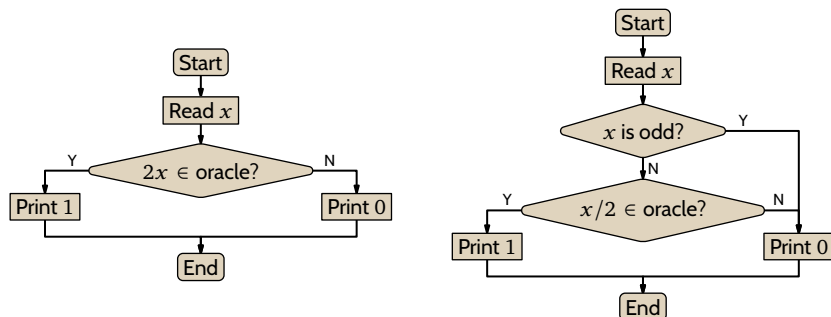


FIGURE 33, FOR QUESTION II.8.19: Showing that  $B \leq_T 2B$  and  $2B \leq_T B$ .

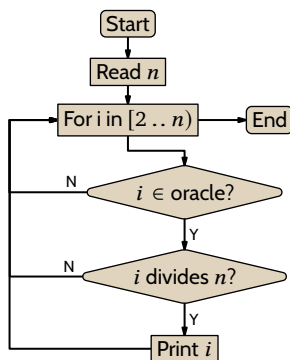


FIGURE 34, FOR QUESTION II.8.20: Oracle machine that can use the set of primes to find prime in the input.

(B) For the flowchart on the right, notice that if  $x$  is odd then  $x/2$  is a natural number.

**II.8.20** See Figure 34 on page 74.

**II.8.21** Mimicing Example 8.3, we go in two steps. First, on the left of Figure 35 on page 75 is a flowchart sketching a program. By Church's Thesis there is a Turing machine that computes it. Let the index of that machine be  $e$ . Apply the  $s$ - $m$ - $n$  theorem to get a family of machines parametrized by  $x$ . The  $x$ -th member of that family is shown in the middle. Clearly it halts for all input if and only if  $\mathcal{P}_x$  halts on inputs 3 and 4. Thus,  $\mathcal{P}_x$  halts on inputs 3 and 4 if and only if  $s(e, x) \in K$ . Restated, there is a computable function, namely  $x \mapsto s(e, x)$ , so that  $x \in \{x \mid \phi_x(3) \downarrow\}$  if and only if  $s(e, x) \in K$ , and therefore  $\{x \mid \phi_x(3) \downarrow\} \leq_T K$ .

The second step uses that computable function  $x \mapsto s(e, x)$  for the oracle machine. That machine is on the right of the figure.

**II.8.22** By definition,  $K_0 = \{\langle e, x \rangle \mid \phi_e(x) \downarrow\}$ . So  $e \in S$  if and only if  $\langle e, 3 \rangle \in K_0$ .

**II.8.23** Let  $D = \{x \mid \phi_x(y) = 2y \text{ for all inputs } y\}$ . Consider this function.

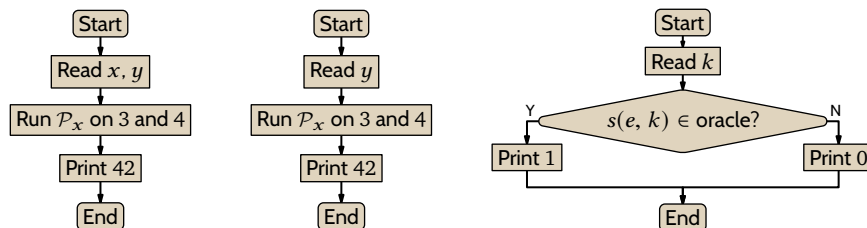
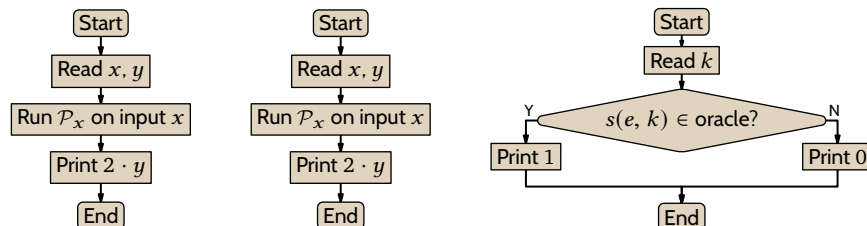
$$\psi(x, y) = \begin{cases} 2y & \text{if } \mathcal{P}_x \text{ halts on input } x \\ \uparrow & \text{otherwise} \end{cases}$$

We can produce this behavior with a program that takes two inputs,  $x, y \in \mathbb{N}$ , simulates running  $\mathcal{P}_x$  on input  $x$ , prints  $2y$ , and then stops. See the flowchart on the left of Figure 36 on page 75. Church's Thesis gives that there is a Turing machine, with an index  $e$ , so that  $\phi_e = \psi$ .

Apply the  $s$ - $m$ - $n$  theorem to get a family of functions  $\phi_{s(e,x)}$  that take in one input  $y$  and output  $2y$  if  $\phi_x(x) \downarrow$ , or diverges otherwise. These are sketched in the middle of the figure. Then  $x \in K$  if and only if  $s(e, x) \in D$ . On the right of the figure is an oracle machine giving that  $K \leq_T D$ , as required.

**II.8.24**

(A) Let  $\mathcal{I} = \{e \mid \phi_e(j) = 7 \text{ for some } j\}$ . This set is not empty because one element is an index of a Turing

FIGURE 35, FOR QUESTION II.8.21: The set  $\{x \mid \phi_x(3) \downarrow \text{ and } \phi_x(4) \downarrow\}$  reduces to  $K$ .FIGURE 36, FOR QUESTION II.8.23: Showing that  $K \leq_T \{x \mid \phi_x(y) = 2y \text{ for all input } y\}$ .

machine that outputs 7 for all inputs. This set is not all of  $\mathbb{N}$  because one non-element is an index of a Turing machine that fails to halt on all inputs.

To see that  $\mathcal{I}$  is an index set, suppose that  $e \in \mathcal{I}$  and that  $\hat{e}$  is such that  $\phi_e \simeq \phi_{\hat{e}}$ . Because  $e \in \mathcal{I}$ , there is an input  $j$  so that  $\phi_e(j) = 7$ . Because  $\phi_e \simeq \phi_{\hat{e}}$ , for the same  $j$  we have  $\phi_{\hat{e}}(j) = 7$ , and therefore  $\hat{e} \in \mathcal{I}$  also.

- (B) As in Example 8.3, we go in two steps. The first is that on the left of Figure 37 on page 76 is a flowchart sketching a program. The second chart expands on the nested loop involved in ‘dovetail’. By Church’s Thesis there is a Turing machine that computes this flowchart. Let the index of that machine be  $e$ . Apply the  $s$ - $m$ - $n$  theorem to get a family of machines parametrized by  $x$ , shown in the third flowchart. Clearly it halts for all inputs  $y$  if and only if  $\mathcal{P}_x$  outputs a 7 for any input. Thus,  $\mathcal{P}_x$  ever outputs a 7 if and only if  $s(e, x) \in K$ . That means there is a computable function,  $x \mapsto s(e, x)$ , where  $x \in \{x \mid \phi_x(j) = 7 \text{ for some } j\}$  if and only if  $s(e, x) \in K$ .

The second step is to use the computable function  $x \mapsto s(e, x)$  to get the oracle machine in Figure 38 on page 76.

**II.8.25** On the left of Figure 39 on page 76 is a sketch of a program. By Church’s Thesis there is a Turing machine with the same behavior. Let the index of that machine be  $e$ . Apply the  $s$ - $m$ - $n$  theorem to get a family of machines parametrized by  $x$ , whose  $x$ -th member,  $\mathcal{P}_{s(e,x)}$ , is shown in the middle. Clearly it halts for all inputs  $y$  if and only if  $x \in S$ . In particular, it halts for  $y = s(e, x)$  if and only if  $x \in S$ . Thus, we have a computable function  $x \mapsto s(e, x)$  so that  $x \in S$  if and only if  $s(e, x) \in K$ . With that, the oracle machine on the right gives that  $S \leq_T K$ .

**II.8.26** Consider the routine sketched on the left of Figure 40 on page 76. By Church’s Thesis there is a Turing machine with that behavior. Let its index be  $e$ . Apply the  $s$ - $m$ - $n$  theorem to get, as shown in the middle, the machine  $\mathcal{P}_{s(e,x)}$  that computes the partial function  $\phi_{s(e,x)}$ . This shows that there is a computable function, namely  $x \mapsto s(e, x)$ , so that  $k \in K$  if and only if  $s(e, k) \in \text{Tot}$ . On the right is an oracle machine using that computable function to show that  $K \leq \text{Tot}$ .

### II.8.27

- (A) If this function had a computable extension  $g$  then we could solve the Halting problem, by running  $\mathcal{P}_x$  on input  $x$  for  $g(x)$ -many steps. If it hasn’t halted by then, it will never halt.
- (B) Consider the routine sketched on the left of Figure 41 on page 77. By Church’s Thesis there is a Turing machine with that behavior. Let its index be  $e$ . In the middle is  $\mathcal{P}_{s(e,x)}$ , computing the partial function  $\phi_{s(e,x)}$ . If  $x \notin K$  then  $\phi_{s(e,x)}(y)$  is undefined for all inputs  $y$ , and this function is a member of  $\text{Ext}$ , since for instance it is extended by the function that always returns 0, which is obviously computable. If  $x \in K$

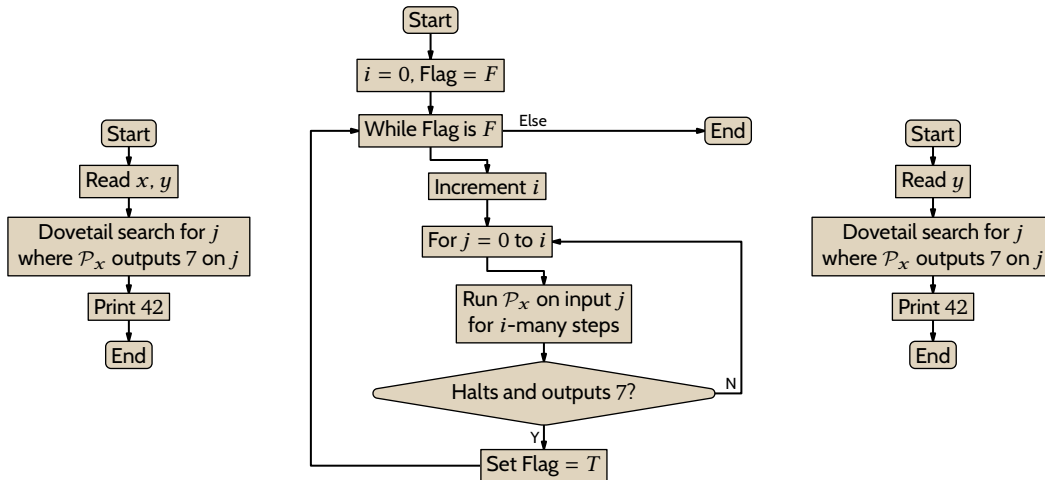


FIGURE 37, FOR QUESTION II.8.24: Computing  $\{x \mid \phi_x(j) = 7 \text{ for some } j\}$  with a  $K$  oracle.

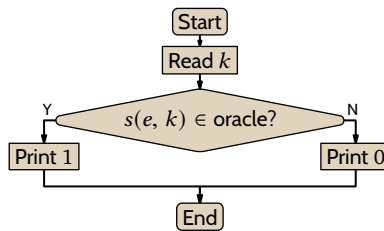


FIGURE 38, FOR QUESTION II.8.24: Oracle machine for computing  $\{x \mid \phi_x(j) = 7 \text{ for some } j\}$  from  $K$ .

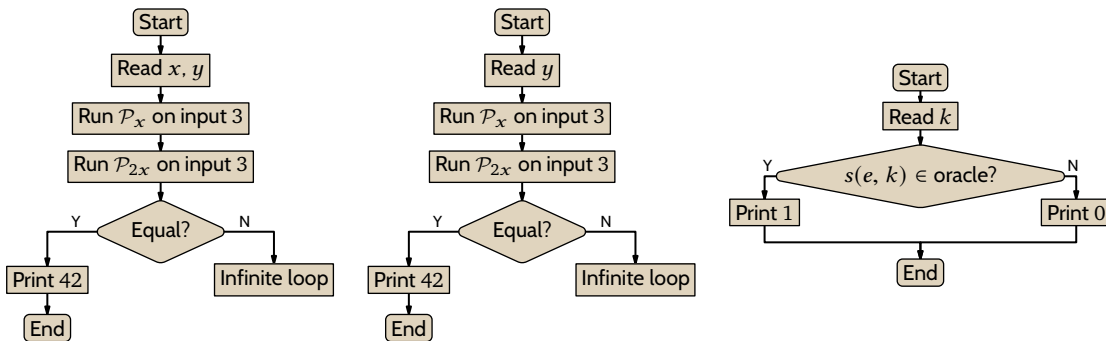


FIGURE 39, FOR QUESTION II.8.25: The set  $\{x \mid \phi_x(3) \downarrow = \phi_{2x}(3) \downarrow\}$  is  $T$ -equivalent to  $K$ .

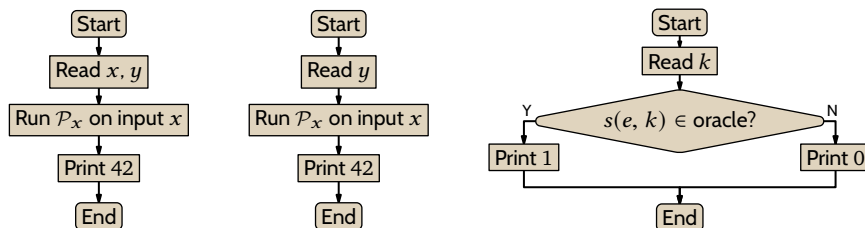
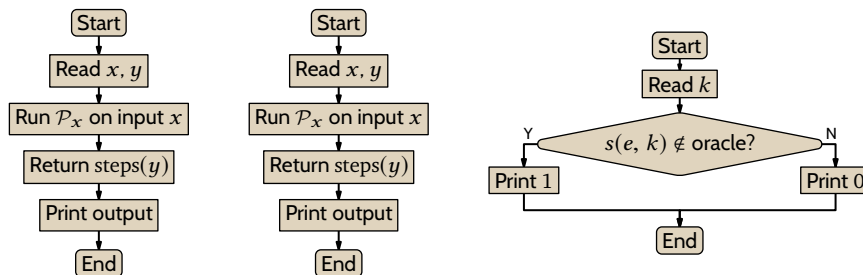
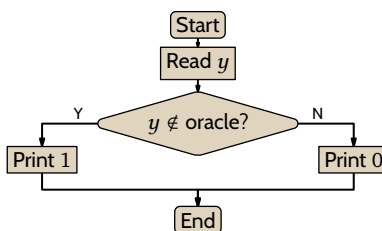


FIGURE 40, FOR QUESTION II.8.26: Computing  $K$  with a Tot oracle.

FIGURE 41, FOR QUESTION II.8.27: Computing  $K$  with a Ext oracle.FIGURE 42, FOR QUESTION II.8.29: Computing  $A$  from an  $A^c$  oracle.

then  $\phi_{s(e,x)}(y) = \text{steps}(y)$  for all inputs  $y$ , and by the first item this is not a member of Ext. This gives a computable function, namely  $x \mapsto s(e, x)$ , so that, by the first part of this question,  $k \in K$  if and only if  $s(e, k) \notin \text{Ext}$ . The oracle machine on the right shows  $K \leq \text{Ext}$ .

**II.8.28** The computation will proceed in exactly the same way, so it will come to the same result.

**II.8.29** See Figure 42 on page 77.

**II.8.30** If  $K \leq_T \emptyset$  were true then there would be an oracle machine whose computed function  $\phi^\emptyset$  is the characteristic function of  $K$ . But because membership in  $\emptyset$  is itself computable we could replace oracle calls with the computation itself, so that any time the machine asks if  $x$  is an element of the oracle, we replace that with 'no'. That mechanically computes  $K$ , which is impossible.

**II.8.31** Refer to Figure 43 on page 78. The flowchart give in the question statement is on the left.

(A) True. This is shown in the flowchart in the middle. Basically, interchange the outputs of 1 and 0.

(B) True. This is shown in the right-hand flowchart. It is the same as the left-hand one but it tests whether  $f(x)$  is not a member of the oracle.

(C) True. This is the left-hand flowchart again.

**II.8.32** Any subset of  $\mathbb{N}$  can be an oracle, so there are uncountably many oracles.

**II.8.33** One is  $C = \{\text{cantor}(0, a) \mid a \in A\} \cup \{\text{cantor}(1, b) \mid b \in B\}$ . Clearly from this set we can answer questions about membership in each of  $A$  and  $B$ .

**II.8.34** There are only countably many programs that can involve an oracle call. That is, there are only countably many oracle-enhanced Turing machines.

**II.8.35**

(A) No. The Halting problem set  $K$  is a subset of the natural numbers  $\mathbb{N}$  but  $K \not\leq_T \mathbb{N}$ , because the Halting problem is unsolvable.

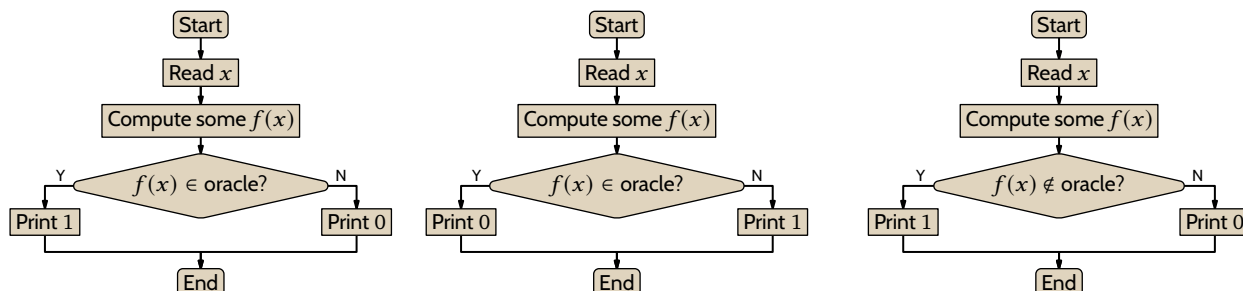
(B) No. The Halting problem set  $K$  is not Turing reducible to  $\mathbb{N}$ , but  $\mathbb{N} = K \cup K^c$ .

(C) No. The Halting problem set  $K$  is not Turing reducible to  $\emptyset$ , but  $\emptyset = K \cap K^c$ .

(D) Yes. We can write this oracle-using program: read the input  $x$  and if  $\text{oracle}(x)$  then output 0, otherwise output 1.

**II.8.36**

(A) The natural definition is that a set  $B$  is computably enumerable in the set  $A$  if  $B$  is the range of a total  $A$ -computable function, or if  $B$  is empty. That is,  $B$  is c.e. in  $A$  if  $B = \{\phi_e^A(x) \mid x \in \mathbb{N}\}$  for some index  $e$

FIGURE 43, FOR QUESTION II.8.31: Oracle machines for what  $A \leq_T B$  implies.

where the function is total, or if  $B = \emptyset$ .

- (B) As in Lemma 8.4, we can output the elements of  $\mathbb{N}$  without even referring to the oracle. The function  $\phi^A(x) = x$  is computable, and does not need to ever refer to the oracle.
- (C) Dovetail. First run the computation of  $\mathcal{P}_0^A$  on input 0 for one step. Then run the computation of  $\mathcal{P}_0^A$  on input 0 for a second step, and the computation of  $\mathcal{P}_1^A$  on input 1 for one step. Next run the computation of  $\mathcal{P}_0^A$  on input 0 for its third step, the computation of  $\mathcal{P}_1^A$  on input 1 for its second step, and the computation of  $\mathcal{P}_2^A$  on input 2 for one step. As computations halt, enumerate the index into  $K^A$ .

**II.9.7** Saying that the two are the same function, that  $\phi_{g(v)} = \phi_{\phi_v(v)}$ , is very different than saying that the indices are the same, that  $g(v) = \phi_v(v)$ . (That is, we can easily write two different programs to compute the same function.) In particular, the Padding Lemma, Lemma 2.15 on page 74, says that every computable function (including the partial functions) has many indices.

### II.9.8

- (A) Apply the Fixed Point Theorem to the computable function  $f(x) = x + 7$ .
- (B) Apply the Fixed Point Theorem to  $f(x) = 2x$ .

### II.9.9

- (A) No matter what numbering scheme we use for Turing machines, if it is acceptable then there is a Turing machine that computes the same function as the machine with twice its index; that is, there exists a  $k \in \mathbb{N}$  so that  $\phi_k = \phi_{3k}$ .
- (B) For any acceptable numbering scheme, there is a Turing machine that computes the same function as the machine whose index is the square. That is, there exists a  $k \in \mathbb{N}$  so that  $\phi_k = \phi_{k^2}$ .
- (C) Consider this  $f: \mathbb{N} \rightarrow \mathbb{N}$ .

$$f(x) = \begin{cases} 1 & \text{if } x = 5 \\ 0 & \text{otherwise} \end{cases}$$

For any acceptable numbering for Turing machines there will be a  $k \in \mathbb{N}$  so that  $\phi_k = \phi_{f(k)}$ .

- (D) If the function is  $f(x) = 42$  then the Fixed Point Theorem gives that for any acceptable numbering there is a  $k \in \mathbb{N}$  so that  $\phi_k = \phi_{f(k)} = \phi_{42}$ . The number  $k = 42$  comes to mind, but the Padding Lemma shows that there are infinitely many such indices  $k$ .

### II.9.10

- (A) Consider the function on the left and observe that it is computed by the program sketched on the left of Figure 44 on page 79. Church's Thesis gives there is a Turing machine with this behavior. Let that machine's index be  $e$ .

$$\psi(x, y) = \begin{cases} 42 & \text{if } y = x^2 \\ \uparrow & \text{otherwise} \end{cases} \quad \phi_{s(e, x)}(y) = \begin{cases} 42 & \text{if } y = x^2 \\ \uparrow & \text{otherwise} \end{cases}$$

Apply the  $s$ - $m$ - $n$  Theorem to get the family of functions on the right.

- (B) The number  $e$  is fixed because it is the index of the Turing machine that we chose to compute  $\psi$ . So, define  $g: \mathbb{N} \rightarrow \mathbb{N}$  by  $g(x) = s(e, x)$ .

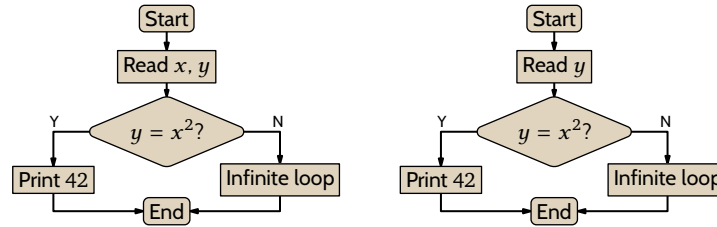


FIGURE 44, FOR QUESTION II.9.10: Flowcharts sketching the machine for  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

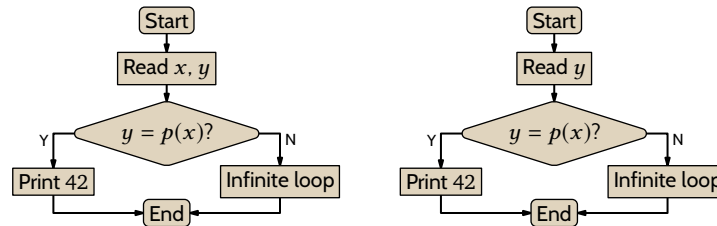


FIGURE 45, FOR QUESTION II.9.11: Flowcharts outlining the machine for  $\psi = \phi_e$  and the machine for  $\phi_{s(e,x)}$ .

(c) The Fixed Point Theorem gives a  $m \in \mathbb{N}$  with  $\phi_m = \phi_{g(m)} = \phi_{s(e,m)}$ . Thus  $W_m = \{m^2\}$ .

**II.9.11**

(A) The function  $p: \mathbb{N} \rightarrow \mathbb{N}$  such that  $p(x)$  is the  $x$ -th prime number is computable by Church's Thesis.

(B) We want to get the family of functions on the right side below, computed by the machines sketched on the right of Figure 45 on page 79, as a uniformly computable family. By now this is a familiar situation. Consider the function on the left, observe that it is computed by the machine sketched on the left of the figure, cite Church's Thesis to get that it has an index, and let that index be  $e$ .

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } y = p(x) \\ \uparrow & \text{-- otherwise} \end{cases} \quad \phi_{s(e,x)}(y) = \begin{cases} 42 & \text{-- if } y = p(x) \\ \uparrow & \text{-- otherwise} \end{cases}$$

Apply the  $s$ - $m$ - $n$  Theorem to get the uniformly computable family of functions on the right.

(c) The number  $e$  is fixed so  $s(e, x)$  is a function only of the variable  $x$ . To emphasize that, define  $g: \mathbb{N} \rightarrow \mathbb{N}$  by  $g(x) = s(e, x)$ . The Fixed Point Theorem gives a  $m \in \mathbb{N}$  with  $\phi_m = \phi_{g(m)} = \phi_{s(e,m)}$ , which halts only when the input is the  $m$ -th prime.

**II.9.12** Consider the function on the left. By Church's Thesis it is computable and so has an index; let that index be  $e$ .

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } y = 10^x \\ \uparrow & \text{-- otherwise} \end{cases} \quad \phi_{s(e,x)}(y) = \begin{cases} 42 & \text{-- if } y = 10^x \\ \uparrow & \text{-- otherwise} \end{cases}$$

Apply the  $s$ - $m$ - $n$  Theorem to get the uniformly computable family of functions on the right. The number  $e$  is fixed so define  $g: \mathbb{N} \rightarrow \mathbb{N}$  by  $g(x) = s(e, x)$ . The Fixed Point Theorem gives  $k \in \mathbb{N}$  with  $\phi_k = \phi_{g(k)} = \phi_{s(e,k)}$ , and so  $W_k = \{10^x\}$ .

**II.9.13** We use Corollary 9.3 as a guide. Consider the function on the left below. It is computed by the machine sketched on the left in Figure 46 on page 80, so Church's Thesis gives that there is a Turing machine index for this machine; let it be  $e$ .

$$\psi(x, y) = \begin{cases} 42 & \text{-- if } y \leq x \\ \uparrow & \text{-- otherwise} \end{cases} \quad \phi_{s(e,x)}(y) = \begin{cases} 42 & \text{-- if } y \leq x \\ \uparrow & \text{-- otherwise} \end{cases} \quad (*)$$

Apply the  $s$ - $m$ - $n$  Theorem to get the family of machines sketched on the right, which compute the family of

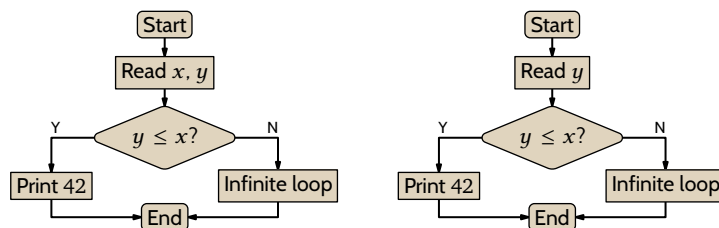


FIGURE 46, FOR QUESTION II.9.13: A sketch of the machine for  $\psi = \phi_e$ , and the machine for  $\phi_{s(e,x)}$ .

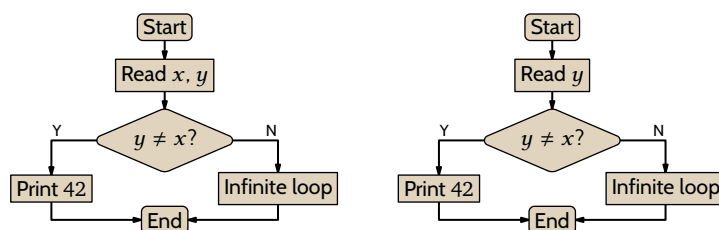


FIGURE 47, FOR QUESTION II.9.15: Sketches for  $\psi = \phi_e$  and for  $\phi_{s(e,x)}$ .

functions given on the right of (\*).

Because  $e$  is the index of the machine on the left, it is fixed. So  $s(e, x)$  is a function of the single variable  $x$ . Let  $g: \mathbb{N} \rightarrow \mathbb{N}$  be defined by  $g(x) = s(e, x)$ . By the Fixed Point Theorem Theorem 9.1, the function  $g$  has a fixed point so that there is a  $k \in \mathbb{N}$  with  $\phi_k = \phi_{s(e,k)}$ . Observe that  $W_k = \{0, 1, \dots, k\}$ .

**II.9.14** Yes, we can take  $f(x) = x$ . Or, if we want  $f(n) \neq n$  then this is the Padding Lemma.

**II.9.15**

(A) The function on the left below is computed by the machine sketched on the left of Figure 47 on page 80. Church's Thesis gives that it has an index; let that index be  $e$ .

$$\psi(x, y) = \begin{cases} 42 & \text{if } y \neq x \\ \uparrow & \text{otherwise} \end{cases} \quad \phi_{s(e,x)}(y) = \begin{cases} 42 & \text{if } y \neq x \\ \uparrow & \text{otherwise} \end{cases}$$

The  $s$ - $m$ - $n$  Theorem parametrizes  $x$  to get the family of functions on the right. The number  $e$  is fixed so define  $g: \mathbb{N} \rightarrow \mathbb{N}$  by  $g(x) = s(e, x)$ . Apply the Fixed Point Theorem to get a  $k \in \mathbb{N}$  with  $\phi_k = \phi_{g(k)} = \phi_{s(e,k)}$ , and so  $W_k = \mathbb{N} - \{k\}$ .

(B) No such computably enumerable  $W_m$  exists. The set  $M = \{m \mid \phi_m(x) \text{ converges}\}$  is computably enumerable, by dovetailing. It is the complement of the given set, so if the given set were computably enumerable then the two would be computable. But showing that they are not computable is routine, for instance with Rice's Theorem.

**II.9.16**

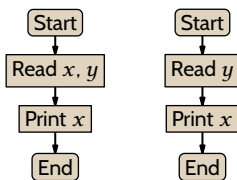
(A) Consider the function  $\psi(x, y) = x$ . By Church's Thesis there is a Turing machine with that behavior — it is computed by the machine on the left of Figure 48 on page 81 — so let that be machine  $\mathcal{P}_e$ . Apply the  $s$ - $m$ - $n$  Theorem to get the machine on the right, which computes the family of functions  $\phi_{s(e,x)}(y) = x$ . There,  $e$  is fixed since it is just the index of the Turing machine on the left, so let  $g: \mathbb{N} \rightarrow \mathbb{N}$  be given by  $g(x) = s(e, x)$ . By the Fixed Point Theorem there is an  $m \in \mathbb{N}$  where  $\phi_m = \phi_{g(m)} = \phi_{s(e,m)}$ , which has range  $\{m\}$ .

(B) This is the same as the prior answer, but starting with  $\psi(x, y) = 2x$ .

**II.9.17** By Corollary 9.3 there is an index  $e$  satisfying this.

$$\phi_e(y) = \begin{cases} \downarrow & \text{if } y = e \\ \uparrow & \text{otherwise} \end{cases}$$

Then  $e \in K$ .

FIGURE 48, FOR QUESTION II.9.16: Flowcharts for  $\psi = \phi_e$  and for  $\phi_{s(e,x)}$ .

By the Padding Lemma there exists  $\hat{e} \neq e$  such that  $\phi_{\hat{e}} \simeq \phi_e$ . Then  $\phi_{\hat{e}}(\hat{e}) \uparrow$ , and so  $\hat{e} \neq K$ . Thus  $K$  is not an index set.

**II.A.1** Put the passengers from bus  $B_0$  in rooms 0, 100, 200, etc. Put the passengers from  $B_1$  in rooms 1, 101, 201, etc. In general, put  $b_{i,j}$  in room  $i + 100j$ .

**II.A.2** Person  $j$  from bus  $B_i$  goes into  $f(b_{i,j}) = 2 \cdot \text{cantor}(i, j) + 1$ .

**II.A.3** Each person is a triple.  $\langle i, j, k \rangle = \langle \text{floor number, space number for the bus, person number on that bus} \rangle$ . One way to assign them rooms is to use  $\text{cantor}(\text{cantor}(i, j), k)$ .

**II.A.4** No, the power set  $\mathcal{P}(\mathbb{R})$  has more members than does  $\mathbb{R}$ .

**II.C.2** Here is a classic.

```
((lambda (x) `(,x ',x))
 '(lambda (x) `(,x ',x)))
```

```
(define self
  (lambda (w)
    ((lambda (w) (list (quote lambda)
      (quote (w))
      (list w ((lambda (w) (list (quote quote) w)) w))))
      (quote
        (lambda (w) (list (quote lambda)
          (quote (w))
          (list w ((lambda (w) (list (quote quote) w)) w))))))))
    ;;Explanation...
    ;;;<PRINT_a <=> f(w) = a
    ;;;<PRINT_a> <=> (quote a)
    ;;;<TM_q <=> q(w) = <PRINT_w>
    ;;;<TM_q> <=> (lambda (w) (list (quote quote) w))
    ;;;<TM_p <=> w(q(w))
    ;;;<TM_p> <=> (lambda (w) (list (quote lambda)
    ;;; (quote (w))
    ;;; (list w (<TM_q> w))))
    ;;;<SELF <=> TM_p(PRINT_<TM_p>(w))
    ;;;<SELF> <=> (lambda (w) (<TMP> (quote <TMP>)))
```

**II.C.3** The code

```
# quine.py
s = r"print 's = r\" + s + '\" + '\nexec(s)'"
exec(s)
```

gives this command line output.

```
$ python quine.py
s = r"print 's = r\" + s + '\" + '\nexec(s)'"
exec(s)
```

Here is another, using Python's string substitution operator %.

```
s = 's = %r\nprint(s%%s)'
print(s%s)
```

**II.C.4**

```
(define (diag s)
  (regexp-replace #rx"x" s s))
```

**II.D.5**  $(2(c+1)(n+2))^{(c+1)n}$

**II.E.1** I got these. 

	0	100	200	300	400	500	600	700
	0.000	0.577	3.178	9.366	19.569	36.004	59.118	91.427

 (If you get different numbers because your machine is much faster than mine, don't email me. I don't give a damn.)

**II.E.2** Turing machine number 666 proves to be  $((0 \ 0 \ 1 \ 0) (2 \ 0 \ 1 \ 0))$ .

**II.E.3**

(A) The new way gives an index without having to enumerate all Turing machines with intermediate Cantor numbers. The new way has the disadvantage that there are natural numbers for which there is no associated Turing machine (such as the number  $2^1$ ).

(B) Sage gives this.

```
sage: 2*3*5^5*7*11*13^4*17*5*19^2
1265294248968750
```

## Chapter III: Languages

### III.1.17

(A) Five are 000, 001, 010, 011, and 100.

(B) These five work:  $\emptyset$ , 1, 00, 11, and 000. (Is the empty string in that language? It is an arguable point.)

**III.1.18** No. By definition, a language is a set of strings and also by definition, a string is of finite length. For some real numbers the decimal representation does not make a finite string; one is  $\pi = 3.14 \dots$  and another is  $1/3 = 0.33 \dots$

**III.1.19** It is both.

**III.1.20** If  $\beta = \langle b_0, \dots, b_{i-1} \rangle$  is a string then  $\beta \wedge \beta^R = \langle b_0, \dots, b_{i-1} \rangle \wedge \langle b_{i-1}, \dots, b_0 \rangle$  is clearly a palindrome.

There are palindromes not of that form. One is  $\gamma = 010$ , which cannot be decomposed into  $\gamma = \beta \wedge \beta^R$ .

### III.1.21

(A) These are the members of  $\mathcal{L}_0 \wedge \mathcal{L}_1$ .

$\varepsilon$ , b, bb, bbb,

a, ab, abb, abbb

aa, aab, aabb, aabbb

aaa, aaab, aaabb, aaabbb

(B) These are the members of  $\mathcal{L}_1 \wedge \mathcal{L}_0$ .

$\varepsilon$ , a, aa, aaa,

b, ba, baa, baaa

bb, bba, bbaa, bbaaa

bbb, bbba, bbbba, bbbbaa

(C) We have this.

$$\mathcal{L}_0^2 = \mathcal{L}_0 \wedge \mathcal{L}_0 = \{\emptyset, a, aa, aaa, aaaa, aaaaa, aaaaaa\}$$

(D) There are many different correct answers, but a natural ten are  $\varepsilon$ , a, aa,  $\dots$ ,  $a^9$ .

### III.1.22

(A) The language consists of any number of a's followed by a single b. Thus five members are b, ab, aab, aaab, and aaaab.

(B) The language consists of a number of a's followed by the same number of b's. Thus five members are  $\varepsilon$ , ab, aabb, aaabbb, and aaaabbbb.

(C) The language is a set of strings of 1's followed by 0's, where there is one more 0 than 1. Five members are 0, 100, 11000, 1110000, and 111100000.

(D) The language consists of strings from  $\mathbb{B}^*$  where there is a string of 1's, then a string of 0's, then a single 1. The number of 0's is twice the number of leading 1's. Five members are 1, 1001, 1100001,  $1^3 0^6 1$ , and  $1^4 0^8 1$ .

### III.1.23

(A)  $\mathcal{L}^2 = \{aa, aab, aba, abab\}$

(B)  $\mathcal{L}^3 = \{aaa, aaab, aaba, aabab, abaa, abaab, ababa, ababab\}$

(C)  $\mathcal{L}^1 = \mathcal{L} = \{a, ab\}$

(D)  $\mathcal{L}^0 = \{\varepsilon\}$

**III.1.24**

- (A)  $\mathcal{L}_0 \hat{\ } \mathcal{L}_1 = \{ab, abb, abb, abbb\}$
- (B)  $\mathcal{L}_1 \hat{\ } \mathcal{L}_0 = \{ba, bab, bba, bbab\}$
- (C)  $\mathcal{L}_0^2 = \{aa, aab, aba, abab\}$
- (D)  $\mathcal{L}_1^2 = \{bb, bbb, bbbb\}$
- (E)  $\mathcal{L}_0^2 \hat{\ } \mathcal{L}_1^2 = \{aabb, aabbb, aabbbb, aabbbbb, ababb, ababbb, ababbbb, ababbbbb\}$

**III.1.25**

- (A) The minimum is three, which happens when the second is a subset of the first. The maximum is five, which happens when the two are disjoint.
- (B) The minimum is zero, which happens when the two are disjoint. The maximum is two, which happens when the second is a subset of the first.
- (C) The maximum is six, as with  $\mathcal{L}_0 = \{a, b, c\}$  and  $\mathcal{L}_1 = \{x, y\}$ , which gives  $\mathcal{L}_0 \hat{\ } \mathcal{L}_1 = \{ax, ay, bx, by, cx, cy\}$ . The minimum is four, as with  $\mathcal{L}_0 = \{\varepsilon, a, aa\}$  and  $\mathcal{L}_1 = \{\varepsilon, a\}$ , which gives  $\mathcal{L}_0 \hat{\ } \mathcal{L}_1 = \{\varepsilon, a, aa, aaa\}$ .
- (D) The minimum is three, as when  $\mathcal{L}_1 = \{\varepsilon, a\}$  gives  $\mathcal{L}_1^2 = \{\varepsilon, a, aa\}$ . The maximum is four, an example of which is  $\mathcal{L}_1 = \{a, b\}$  giving  $\mathcal{L}_1^2 = \{aa, ab, ba, bb\}$ .
- (E) The intersection can be infinite, as with  $\mathcal{L}_0 = \{\varepsilon, a, b\}$  and  $\mathcal{L}_1 = \{\varepsilon, a\}$ , where  $a^n \in \mathcal{L}_0^* \cap \mathcal{L}_1^*$  for all  $n \in \mathbb{N}$ . The smallest the intersection can be is size one, as when  $\mathcal{L}_0 = \{a, b, c\}$  and  $\mathcal{L}_1 = \{x, y\}$  and the only element of  $\mathcal{L}_0^* \cap \mathcal{L}_1^*$  is the empty string.

**III.1.26** The concatenation of 0-many strings is  $\varepsilon$ . That is, if  $\sigma \in \mathcal{L}$  then  $\sigma^0$  is defined to be the empty string.

**III.1.27** It is the empty set,  $\emptyset^* = \emptyset$ . By definition,  $\mathcal{L}^* = \{\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1} \mid k \in \mathbb{N} \text{ and } \sigma_0, \dots, \sigma_{k-1} \in \mathcal{L}\}$ . If the language is empty then the condition ' $\sigma_0, \dots, \sigma_{k-1} \in \mathcal{L}$ ' is never met.

**III.1.28** No. Let  $\mathcal{L} = \{a, b\}$  and let  $k = 2$ . Then the  $k$ -th power of the language is  $\mathcal{L}^k = \{aa, ab, ba, bb\}$ , while the language of  $k$ -th powers is  $\{\sigma^2 \mid \sigma \in \mathcal{L}\} = \{aa, bb\}$ .

**III.1.29** Yes, but only in an edge case. If  $\mathcal{L} = \emptyset$  then  $\mathcal{L}^* = \emptyset$  while  $(\mathcal{L} \cup \{\varepsilon\})^* = \{\varepsilon\}$ .

We will show that if  $\mathcal{L} \neq \emptyset$  then the two sets are equal,  $\mathcal{L}^* = (\mathcal{L} \cup \{\varepsilon\})^*$ . The containment  $\mathcal{L}^* \subseteq (\mathcal{L} \cup \{\varepsilon\})^*$  is clear.

For the other direction,  $(\mathcal{L} \cup \{\varepsilon\})^* \subseteq \mathcal{L}^*$ , assume that  $\sigma \in (\mathcal{L} \cup \{\varepsilon\})^*$ . Then  $\sigma = \sigma_0 \hat{\ } \dots \hat{\ } \sigma_{n-1}$  for some  $n \in \mathbb{N}$  and  $\sigma_i \in \mathcal{L} \cup \{\varepsilon\}$ . If for all  $i \in \{0, \dots, n-1\}$  the string  $\sigma_i$  equals  $\varepsilon$  then  $\sigma = \varepsilon$ , and is an element of  $\mathcal{L}^*$  because  $\mathcal{L} \neq \emptyset$  and  $\varepsilon = \rho^0$  for any  $\rho \in \mathcal{L}$ . Otherwise there is at least one  $i \in \{0, \dots, n-1\}$  such that  $\sigma_i \neq \varepsilon$ . In this case, omitting from the string  $\sigma$  those  $\sigma_j$  that equal the empty string gives that  $\sigma \in \mathcal{L}^k$  for some  $k \leq n$ , and so  $\sigma \in \mathcal{L}^*$ .

**III.1.30** Let the alphabet be  $\Sigma$ .

- (A) Consider unequal strings  $\sigma_0 \neq \sigma_1$ . They can't both be the empty string, and if one of them is the empty string but the other is not then clearly  $\sigma_0^2 \neq \sigma_1^2$  because one of those two is empty while the other is not. So now suppose that neither of the two strings  $\sigma_0$  and  $\sigma_1$  is empty. If they are different lengths then their squares are also different lengths. The case remaining is that they are nonempty equal-length strings,  $\sigma_0 = \langle s_{0,0}, s_{0,1}, \dots, s_{0,n} \rangle$  and  $\sigma_1 = \langle s_{1,0}, s_{1,1}, \dots, s_{1,n} \rangle$  for some  $n \in \mathbb{N}$ . Fix the smallest index  $i$  where  $s_{0,i} \neq s_{1,i}$ . For that same  $i$  we have that  $\sigma_0^2$  and  $\sigma_1^2$  differ on index  $i$ , and are therefore unequal.

The prior paragraph implies that if  $\mathcal{L}$  has  $k$ -many different elements  $\mathcal{L} = \{\sigma_0, \dots, \sigma_{k-1}\}$  then the elements of the set  $\{\sigma_0^2, \dots, \sigma_{k-1}^2\} \subseteq \mathcal{L}^2$  are different.

- (B) The language  $\mathcal{L} = \{\varepsilon\}$  has  $\mathcal{L}^2 = \{\varepsilon\}$ .
- (C) Let  $\mathcal{L} = \{\sigma_0, \sigma_1, \dots, \sigma_{k-1}\}$  for  $k \in \mathbb{N}^+$  (we take these strings to all be different from each other). Then the list  $\sigma_0 \hat{\ } \sigma_0, \sigma_0 \hat{\ } \sigma_1, \dots, \sigma_{k-1} \hat{\ } \sigma_{k-1}$  has length  $k^2$ . Hence the largest number of elements that  $\mathcal{L}^2$  can have is  $k^2$ .
- (D) Consider the language  $\mathcal{L} = \{\sigma_0, \sigma_1, \dots, \sigma_{k-1}\}$  where the  $k$ -many strings have length one and are unequal, so each consists of a single symbol that is different from the symbols used in any of the others. Clearly all of the members of the list  $\sigma_0 \hat{\ } \sigma_0, \sigma_0 \hat{\ } \sigma_1, \dots, \sigma_{k-1} \hat{\ } \sigma_{k-1}$  are unequal, and so  $\mathcal{L}^2$  has  $k^2$ -many elements.

**III.1.31** Recall that  $\mathcal{L}^k = \{\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1} \mid \sigma_i \in \mathcal{L}\}$  and that  $\mathcal{L}^* = \{\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1} \mid k \in \mathbb{N} \text{ and } \sigma_0, \dots, \sigma_{k-1} \in \mathcal{L}\}$ . If  $\mathcal{L} = \emptyset$  then both sets are empty, so we can assume  $\mathcal{L} \neq \emptyset$ .

We first show that  $\mathcal{L}^k \subseteq \mathcal{L}^*$ . If  $k = 0$  then because we are assuming  $\mathcal{L} \neq \emptyset$ , we can take some  $\sigma \in \mathcal{L}$  and find  $\sigma^0 = \varepsilon$ . Thus  $\mathcal{L}^0 = \{\varepsilon\}$  and  $\varepsilon \in \mathcal{L}^*$ . Now suppose that  $k > 0$  and fix  $\sigma \in \mathcal{L}^k$ , so that  $\sigma = \sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1}$ . Then the definition of  $\mathcal{L}^*$  gives that  $\sigma \in \mathcal{L}^*$ .

We finish by showing that  $\mathcal{L}^* \subseteq \mathcal{L}^0 \cup \mathcal{L}^1 \cup \dots$ . Fix  $\sigma \in \mathcal{L}^*$  (recall that  $\mathcal{L} \neq \emptyset$ ), so that  $\sigma = \sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1}$  for some  $k \in \mathbb{N}$ . Note that  $\sigma \in \mathcal{L}^k$  (even if  $k = 0$ ). Thus  $\mathcal{L}^* \subseteq \mathcal{L}^0 \cup \mathcal{L}^1 \cup \dots$ .

**III.1.32** The set of concatenations  $\sigma_1 \hat{\ } \sigma_0$  where  $\sigma_1 \in \mathcal{L}_1$  and  $\sigma_0 \in \mathcal{L}_0$  is empty, because there are no  $\sigma_0$  members of  $\mathcal{L}_0$ .

**III.1.33**

- (A) An alphabet is finite by definition. (Appendix A has a review.)
- (B) Because the language is finite, there is a longest string and for  $B$  we can take the length of that string (or else the language is empty and we can take  $B = 0$ ).
- (C) Let  $\mathcal{L}_0, \dots, \mathcal{L}_k$  be finite languages over  $\Sigma^*$  for some  $k \in \mathbb{N}$ . Then the union is also finite because it contains at most  $|\mathcal{L}_0| + \dots + |\mathcal{L}_k|$  elements.
- (D) Let  $\mathcal{L}_0, \dots, \mathcal{L}_k$  be finite languages over  $\Sigma^*$ . Then the intersection contains at most  $\min(|\mathcal{L}_0|, \dots, |\mathcal{L}_k|)$ -many elements, so it is finite also. The number of strings in the concatenation  $\mathcal{L}_0 \hat{\ } \mathcal{L}_1 \hat{\ } \dots \hat{\ } \mathcal{L}_k$  is at most the product  $|\mathcal{L}_0| \cdot |\mathcal{L}_1| \cdot \dots \cdot |\mathcal{L}_k|$ .
- (E) If  $\Sigma = \{a, b\}$  then the complement of the finite language  $\mathcal{L} = \{\}$  is  $\mathcal{L}^c = \Sigma^*$ , which is infinite since it contains  $a^n$  for all  $n$ . For the same alphabet,  $\mathcal{L} = \{a, b\}$  is finite but  $\mathcal{L}^*$  is infinite.

**III.1.34** In  $\hat{\ } \mathcal{L}$  all strings are of even length. The language  $\mathcal{L}$  contains all palindromes, of any length.

**III.1.35** The prefixes are  $\varepsilon, a, b, ab, bb, aba, bba, abaa, abaab, \text{ and } abaaba$ .

**III.1.36**

- (A) Concatenating  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{m-1} \in \mathcal{L}^m$  with  $\tau_0 \hat{\ } \dots \hat{\ } \tau_{n-1} \in \mathcal{L}^n$  gives  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{m-1} \hat{\ } \tau_0 \hat{\ } \dots \hat{\ } \tau_{n-1} \in \mathcal{L}^{m+n}$ .
- (B) By definition  $\mathcal{L}_0 \hat{\ } \mathcal{L}_1 = \{\sigma_0 \hat{\ } \sigma_1 \mid \sigma_0 \in \mathcal{L}_0 \text{ and } \sigma_1 \in \mathcal{L}_1\}$ . If one of the two is the empty set then the condition is never satisfied, so the result is the empty set.
- (C) For any string  $\sigma \in \mathcal{L}^*$  we have that  $\sigma^0 = \varepsilon$ . Thus, if there are any strings in the language then when raised to the zero power they all give the empty string, so the result is  $\mathcal{L}^0 = \{\varepsilon\}$ .
- (D) By the second item, if  $\mathcal{L} = \emptyset$  gave that  $\mathcal{L}^0 = \emptyset$  for  $\mathcal{L} = \emptyset$ , then for any language  $\hat{\ } \mathcal{L}$ , extending the formula for the prior item results in  $\hat{\ } \mathcal{L} = \hat{\ } \mathcal{L}^1 = \hat{\ } \mathcal{L}^{1+0} = \hat{\ } \mathcal{L} \hat{\ } \emptyset = \emptyset$ , which is nonsense.

**III.1.37**

- (A) Consider the set  $S = \Sigma \times \dots \times \Sigma$  (we are avoiding calling this  $\Sigma^n$  because the two differ, in that the cross product is set of  $n$ -tuples while  $\Sigma^n$  is a subset of  $\Sigma^*$ ). It is the cross product of countably many countable sets and so is countable by Chapter Two's Corollary 2.9. So there is a one-to-one and onto function  $g: \mathbb{N} \rightarrow S$ . The function  $\text{concat}: S \rightarrow \Sigma^n$  that concatenates its arguments together  $f(\sigma_0, \dots, \sigma_{n-1}) = \sigma_0 \hat{\ } \dots \hat{\ } \sigma_{n-1}$  is clearly onto. So the composition  $f \circ g: \mathbb{N} \rightarrow \Sigma^n$  is onto. Then by Chapter Two's Lemma 2.11,  $\Sigma^n$  is countable.
- (B) The set  $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \dots$  is the countable union of countable sets and so by Chapter Two's Corollary 2.12 is a countable set.

**III.1.38** To prevent confusion between the two forms, denote  $\{\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{k-1} \mid \sigma_i \in \mathcal{L}\}$  as  ${}^k\mathcal{L}$ , for the purpose of this argument.

We first show that any element of  ${}^k\mathcal{L}$  is an element of  $\mathcal{L}^k$ . We do a simple induction argument. The base cases are that  ${}^0\mathcal{L}$  and  $\mathcal{L}^0$  both equal to  $\{\varepsilon\}$  by definition, and that  $\sigma_0 \in \mathcal{L}^1$  and  $\sigma_0 \in {}^1\mathcal{L}$  for any  $\sigma_0 \in \mathcal{L}$ . The inductive step is that from the hypothesis that  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1} \in {}^i\mathcal{L}$  implies that  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1} \in \mathcal{L}^i$  we get that  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1} \hat{\ } \sigma_i \in \mathcal{L}^{i+1}$ , because it is  $(\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1}) \hat{\ } \sigma_i$ .

The other direction is that any element of any  $\mathcal{L}^k$  is an element of  ${}^k\mathcal{L}$ . Again we do this by a simple induction. Again the base cases are that  ${}^0\mathcal{L}$  and  $\mathcal{L}^0$  both equal to  $\{\varepsilon\}$  by definition, and that  $\sigma_0 \in \mathcal{L}^1$  and  $\sigma_0 \in {}^1\mathcal{L}$  for any  $\sigma_0 \in \mathcal{L}$ . The inductive step is that if  $\sigma \in \mathcal{L}^i$  then  $\sigma \in {}^i\mathcal{L}$  and so  $\sigma = \sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1}$ . Then  $\sigma \hat{\ } \tau$  has the form  $\sigma_0 \hat{\ } \dots \hat{\ } \sigma_{i-1} \hat{\ } \tau$  and so is an element of  ${}^{i+1}\mathcal{L}$ .

**III.1.39** The statement is true. As motivation for the argument, the natural language to try for a counterexample is  $\mathcal{L} = \{a, aa, \dots\}$ . However,  $\mathcal{L} \hat{\ } \mathcal{L}$  is a proper subset of  $\mathcal{L}$  since it does not contain  $a$ .

With that, suppose  $\mathcal{L} \neq \emptyset$ . Then there is a string  $\sigma \in \Sigma^*$  such that  $\sigma \in \mathcal{L}$ . From among all language members, take  $\sigma$  to have minimal length. If  $\varepsilon \neq \mathcal{L}$  then  $\sigma \hat{\ } \tau \in \mathcal{L} \hat{\ } \mathcal{L}$  is strictly longer than  $\sigma$  for any  $\tau \in \mathcal{L}$ , and by the minimality of  $\sigma$ , it is not an element of  $\mathcal{L} \hat{\ } \mathcal{L}$ . This means that  $\mathcal{L} \hat{\ } \mathcal{L}$  is not equal to  $\mathcal{L}$ .

**III.1.40** Every language is countable, since it is a subset of  $\Sigma^*$ , which is countable. But there are uncountably many reals.

**III.1.41**

- (A) For any sets, languages or not, union and intersection are commutative.
- (B) We have  $\mathcal{L} \cap \{\varepsilon\} = \{\sigma \cap \varepsilon \mid \sigma \in \mathcal{L}\} = \{\sigma \mid \sigma \in \mathcal{L}\} = \mathcal{L}$ .
- (C) Where  $\Sigma = \{a, b\}$  take  $\mathcal{L}_0 = \{a\}$  and  $\mathcal{L}_1 = \{b\}$ . Then  $\mathcal{L}_0 \cap \mathcal{L}_1 = \{ab\}$  while  $\mathcal{L}_1 \cap \mathcal{L}_0 = \{ba\}$ .
- (D) This is immediate from the fact that string concatenation is associative: the  $i$ -th entry of  $(\sigma_0 \cap \sigma_1) \cap \sigma_2$  is the same as the  $i$ -th entry of  $\sigma_0 \cap (\sigma_1 \cap \sigma_2)$ .
- (E) This is immediate from the fact that for strings  $(\sigma_0 \cap 1)^R$  equals  $\sigma_1^R \cap \sigma_0^R$ .
- (F) A string is an element of  $(\mathcal{L}_0 \cup \mathcal{L}_1) \cap \mathcal{L}_2$  if and only if it has the form  $\sigma \cap \tau$  where  $\tau \in \mathcal{L}_2$ , and  $\sigma \in \mathcal{L}_0$  or  $\sigma \in \mathcal{L}_1$ . That's true if and only if both  $\tau \in \mathcal{L}_2$  and  $\sigma \in \mathcal{L}_0$ , or both  $\tau \in \mathcal{L}_2$  and  $\sigma \in \mathcal{L}_1$ . In turn that holds if and only if  $\sigma \in (\mathcal{L}_0 \cap \mathcal{L}_2) \cup (\mathcal{L}_1 \cap \mathcal{L}_2)$ . The right-distributive argument goes the same way.
- (G) For any language  $\mathcal{L}$  the definition gives  $\emptyset \cap \mathcal{L} = \{\sigma_0 \cap \sigma_1 \mid \sigma_0 \in \emptyset \text{ and } \sigma_1 \in \mathcal{L}\}$ . The condition ' $\sigma_0 \in \emptyset$ ' is never satisfied so this set is empty. The same goes for  $\mathcal{L} \cap \emptyset$ .
- (H) For any strings, no matter how the concatenation is parenthesized we can remove the parentheses. For instance,  $((\sigma_0 \cap \sigma_1) \cap \sigma_2) \cap \sigma_3$  equals  $(\sigma_0 \cap (\sigma_1 \cap \sigma_2)) \cap \sigma_3$  because the  $i$ -th elements of the two are equal. (We can easily show this statement by induction.) From this, the result is immediate.

**III.2.12**

- (A) We've adopted the convention that the start symbol is the head of the first listed rule so it is  $\langle expr \rangle$ .
- (B) The terminals are a, b, etc.
- (C) The nonterminals are  $\langle expr \rangle$ ,  $\langle term \rangle$ , and  $\langle factor \rangle$ .
- (D) There are twenty seven rewrite rules on the final line. The other two lines have a total of four, so that makes thirty one.
- (E) Two are a+b and j\*h+k\*m.
- (F) Three are a+, and +, and \*\*.

**III.2.13**

- (A) The start symbol, by our convention that it is the head of the first listed rule, is  $\langle sentence \rangle$ .
- (B) The terminals are the, young, caught, man, and ball.
- (C) The nonterminals are  $\langle sentence \rangle$ ,  $\langle noun phrase \rangle$ ,  $\langle verb phrase \rangle$ ,  $\langle article \rangle$ ,  $\langle noun \rangle$ ,  $\langle adjective \rangle$ , and  $\langle verb \rangle$ .
- (D) Three are the ball caught the young man, the ball caught the man, and the man caught the man.
- (E) Three are ball, man man man, and  $\varepsilon$ .

**III.2.14**

- (A) The alphabet is  $\Sigma = \{0, \dots, 9\}$ . The terminals are 0, ..., 9. The nonterminals are  $\langle natural \rangle$  and  $\langle digit \rangle$ . The start symbol is  $\langle natural \rangle$ .
- (B) For the production

$$\langle natural \rangle \rightarrow \langle digit \rangle$$

the head is  $\langle natural \rangle$  while the body is  $\langle digit \rangle$ . For the production

$$\langle natural \rangle \rightarrow \langle digit \rangle \langle natural \rangle$$

the head is  $\langle natural \rangle$  while the body is  $\langle digit \rangle \langle digit \rangle$ . For the production

$$\langle digit \rangle \rightarrow \emptyset$$

the head is  $\langle digit \rangle$  and the body is  $\emptyset$ . The remaining eight productions are similar.

- (C) The two metacharacters are  $\rightarrow$  and  $|$ .

- (D) Here is a derivation.

$$\begin{aligned} \langle natural \rangle &\Rightarrow \langle digit \rangle \langle natural \rangle \\ &\Rightarrow 4 \langle natural \rangle \\ &\Rightarrow 4 \langle digit \rangle \\ &\Rightarrow 42 \end{aligned}$$

For the associated parse tree, see Figure 49 on page 87.

- (E) Here is a derivation.

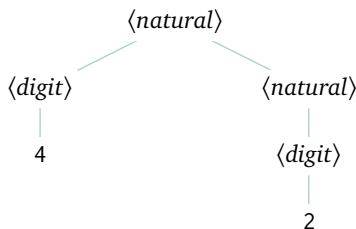


FIGURE 49, FOR QUESTION III.2.14: Parse tree for 42.

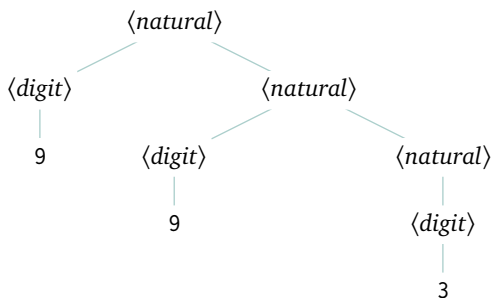


FIGURE 50, FOR QUESTION III.2.14: Parse tree associated with the derivation of 993.

$$\begin{aligned}
 \langle \text{natural} \rangle &\Rightarrow \langle \text{digit} \rangle \langle \text{natural} \rangle \\
 &\Rightarrow \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{natural} \rangle \\
 &\Rightarrow \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \\
 &\Rightarrow \langle \text{digit} \rangle 9 \langle \text{digit} \rangle \\
 &\Rightarrow \langle \text{digit} \rangle 93 \\
 &\Rightarrow 993
 \end{aligned}$$

The parse tree is Figure 50 on page 87.

(F) We want to start at the start symbol and end at the desired string. Just willy-nilly expanding is not the game.

(G) Here is one answer

$$\begin{aligned}
 \langle \text{integer} \rangle &\rightarrow +\langle \text{natural} \rangle \mid -\langle \text{natural} \rangle \mid \langle \text{natural} \rangle \\
 \langle \text{natural} \rangle &\rightarrow \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{natural} \rangle \\
 \langle \text{digit} \rangle &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
 \end{aligned}$$

(H) Yes, we can derive both  $+0$  and  $-0$ .

### III.2.15

(A) Here is a derivation of ‘the car hit a wall’.

$$\begin{aligned}
 \langle \text{sentence} \rangle &\Rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{predicate} \rangle \\
 &\Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{direct object} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle \\
 &\Rightarrow \text{the} \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle \Rightarrow \text{the car} \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle \\
 &\Rightarrow \text{the car hit} \langle \text{article} \rangle \langle \text{noun} \rangle \Rightarrow \text{the car hit a} \langle \text{noun} \rangle \\
 &\Rightarrow \text{the car hit a wall}
 \end{aligned}$$

(B) Here is a derivation of ‘the car hit the wall’.

$$\begin{aligned}
\langle \text{sentence} \rangle &\Rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{predicate} \rangle \\
&\Rightarrow \text{the} \langle \text{noun} \rangle \langle \text{predicate} \rangle \Rightarrow \text{the car} \langle \text{predicate} \rangle \\
&\Rightarrow \text{the car} \langle \text{verb} \rangle \langle \text{direct object} \rangle \Rightarrow \text{the car hit} \langle \text{direct object} \rangle \\
&\Rightarrow \text{the car hit} \langle \text{article} \rangle \langle \text{noun} \rangle \Rightarrow \text{the car hit the} \langle \text{noun} \rangle \\
&\Rightarrow \text{the car hit the wall}
\end{aligned}$$

(c) This is one for ‘the wall hit a car’.

$$\begin{aligned}
\langle \text{sentence} \rangle &\Rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{predicate} \rangle \\
&\Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{direct object} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun} \rangle \\
&\Rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \text{hit} \langle \text{article} \rangle \langle \text{noun} \rangle \Rightarrow \langle \text{article} \rangle \text{wall hit} \langle \text{article} \rangle \langle \text{noun} \rangle \\
&\Rightarrow \langle \text{article} \rangle \text{wall hit} \langle \text{article} \rangle \text{car} \Rightarrow \langle \text{article} \rangle \text{wall hit a car} \\
&\Rightarrow \text{the wall hit a car}
\end{aligned}$$

### III.2.16

(A) This is a derivation for ‘dog bites man’.

$$\begin{aligned}
\langle \text{sentence} \rangle &\Rightarrow \langle \text{subject} \rangle \langle \text{predicate} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun}_1 \rangle \langle \text{predicate} \rangle \\
&\Rightarrow \langle \text{article} \rangle \langle \text{noun}_1 \rangle \langle \text{verb} \rangle \langle \text{direct object} \rangle \Rightarrow \langle \text{article} \rangle \langle \text{noun}_1 \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun}_2 \rangle \\
&\Rightarrow \varepsilon \langle \text{noun}_1 \rangle \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun}_2 \rangle \Rightarrow \varepsilon \text{dog} \langle \text{verb} \rangle \langle \text{article} \rangle \langle \text{noun}_2 \rangle \\
&\Rightarrow \varepsilon \text{dog bites} \langle \text{article} \rangle \langle \text{noun}_2 \rangle \Rightarrow \varepsilon \text{dog bites} \varepsilon \langle \text{noun}_2 \rangle \\
&\Rightarrow \varepsilon \text{dog bites} \varepsilon \text{man} = \text{dog bites man}
\end{aligned}$$

(B) The start symbol  $\langle \text{subject} \rangle$  expands to  $\langle \text{article} \rangle \langle \text{noun}_1 \rangle$ , but man is derived only from  $\langle \text{noun}_2 \rangle$ . So man cannot be the first word of the sentence.

III.2.17 There is no nonterminal  $\langle \text{dog} \mid \text{flea} \rangle$  or  $\langle \text{man} \mid \text{dog} \rangle$ . This is confusing nonterminals with terminals.

III.2.18 The outermost operation is the  $*$ , so we go for that first. Next we get the parentheses and the  $+$ . Here is a derivation.

$$\begin{aligned}
\langle \text{expr} \rangle &\Rightarrow \langle \text{term} \rangle \\
&\Rightarrow \langle \text{term} \rangle * \langle \text{factor} \rangle \\
&\Rightarrow \langle \text{factor} \rangle * \langle \text{factor} \rangle \\
&\Rightarrow ( \langle \text{expr} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( \langle \text{term} \rangle + \langle \text{expr} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( \langle \text{term} \rangle + \langle \text{term} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( \langle \text{factor} \rangle + \langle \text{term} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( a + \langle \text{term} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( a + \langle \text{factor} \rangle ) * \langle \text{factor} \rangle \\
&\Rightarrow ( a + b ) * \langle \text{factor} \rangle \\
&\Rightarrow ( a + b ) * c
\end{aligned}$$

### III.2.19

(A) First the leftmost derivation.

$$\begin{aligned}
S &\Rightarrow T b U \Rightarrow a T b U \Rightarrow a a T b U \Rightarrow a a \varepsilon b U = a a b U \\
&\Rightarrow a a b a U \Rightarrow a a b a b U \Rightarrow a a b a b \varepsilon = a a b a b
\end{aligned}$$

Next the rightmost derivation.

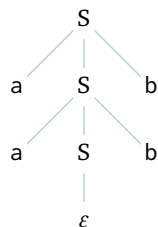
$$\begin{aligned}
S &\Rightarrow T b U \Rightarrow T b a U \Rightarrow T b a b U \Rightarrow T b a b \varepsilon = T b a b \\
&\Rightarrow a T b a b \Rightarrow a a T b a b \Rightarrow a a \varepsilon b a b = a a b a b
\end{aligned}$$

(B) This is the leftmost derivation.

$$\begin{aligned}
S &\Rightarrow T b U \Rightarrow \varepsilon b U = b U \Rightarrow b a U \\
&\Rightarrow b a a U \Rightarrow b a a b U \Rightarrow b a a b \varepsilon = b a a b
\end{aligned}$$

Here is the rightmost.

$$\begin{aligned}
S &\Rightarrow T b U \Rightarrow T b a U \Rightarrow T b a a U \Rightarrow T b a a b U \\
&\Rightarrow T b a a b \varepsilon = T b a a b \Rightarrow \varepsilon b a a b = b a a b
\end{aligned}$$

FIGURE 51, FOR QUESTION III.2.22: Parse tree for  $a^2b^2$ .

(c) The start symbol expands to something that includes a b.

**III.2.20**

(A) Here are the derivations of aabb,

$$S \Rightarrow aABb \Rightarrow aaBb \Rightarrow aabb$$

of aaabb,

$$S \Rightarrow aABb \Rightarrow aaABb \Rightarrow aaaBb \Rightarrow aabb$$

and of aabbb

$$S \Rightarrow aABb \Rightarrow aABb \Rightarrow aaBb \Rightarrow aaBbb \Rightarrow aabbb$$

(B) The production starting with the start symbol S shows that every derived string must begin with a and end with b. So three strings that cannot be derived are a, ba, and  $\epsilon$  (the empty string).

(c)  $\mathcal{L} = \{a^n b^m \mid n, m \geq 2\}$

**III.2.21** Here is one.

$$S \rightarrow TU$$

$$T \rightarrow aTb \mid \epsilon$$

$$U \rightarrow bUa \mid \epsilon$$

**III.2.22** Figure 51 on page 89 shows the tree.

**III.2.23** We will use induction to show that in the course of a derivation all of the results are of either the form either  $a^k S b^k$  or of the form  $a^k b^k$ . By the definition of the language generated by a grammar, only strings without any nonterminals are in the language so that will give the verification.

The induction is on the number of derivation steps, that is, on the number of rule applications. The base step is that there are zero-many rule applications. In this case the string is 'S', which is the  $k = 0$  instance of the desired form.

For the inductive step assume that the statement is true where the number of rule applications is  $n = 0, n = 1, \dots, n = k$  and consider the  $n = k + 1$  case. We get the string for the  $k + 1$  case by applying a rule to a string after the  $k$  case. The inductive hypothesis is that such a string has one of two forms,  $a^k S b^k$  or  $a^k b^k$ . The rules don't apply to the latter string so consider the former one.

Applied to  $a^k S b^k$  the first rule gives  $a^{k+1} S b^{k+1}$ , which has the right form. The second rule gives  $a^{k+1} b^{k+1}$ , again of the right form.

**III.2.24** Bit of a trick question: this grammar has no terminating derivations so it generates the empty language  $\mathcal{L} = \emptyset$ .

**III.2.25** Here is one.

$$\langle \text{identifier} \rangle \rightarrow \langle \text{letter} \rangle \mid \langle \text{letter-or-digit-string} \rangle$$

$$\langle \text{letter-or-digit-string} \rangle \rightarrow \langle \text{letter} \rangle \langle \text{letter-or-digit-string} \rangle$$

$$\mid \langle \text{digit} \rangle \langle \text{letter-or-digit-string} \rangle$$

$$\mid \epsilon$$

$$\langle \text{letter} \rangle \rightarrow a \mid \dots \mid z \mid A \mid \dots \mid Z$$

$\langle \text{digit} \rangle \rightarrow 0 \mid \dots \mid 9$

### III.2.26

$\langle \text{identifier} \rangle \rightarrow \langle \text{letter} \rangle \langle \text{second} \rangle$

$\langle \text{second} \rangle \rightarrow \langle \text{letter-or-digit} \rangle \langle \text{third} \rangle \mid \varepsilon$

$\langle \text{third} \rangle \rightarrow \langle \text{letter-or-digit} \rangle \langle \text{fourth} \rangle \mid \varepsilon$

$\langle \text{fourth} \rangle \rightarrow \langle \text{letter-or-digit} \rangle$

$\langle \text{letter-or-digit} \rangle \rightarrow \langle \text{letter} \rangle \mid \langle \text{digit} \rangle$

$\langle \text{letter} \rangle \rightarrow a \mid \dots \mid z \mid A \mid \dots \mid Z$

$\langle \text{digit} \rangle \rightarrow 0 \mid \dots \mid 9$

III.2.27 It is the empty language,  $\mathcal{L} = \emptyset$ .

### III.2.28

(A)  $\langle \text{string} \rangle \rightarrow \langle \text{letter} \rangle \langle \text{sring} \rangle \mid \varepsilon$

$\langle \text{letter} \rangle \rightarrow a \mid \dots \mid z$

(B)  $\langle \text{string}_1 \rangle \rightarrow \langle \text{digit} \rangle \langle \text{sring}_2 \rangle$

$\langle \text{string}_2 \rangle \rightarrow \langle \text{digit} \rangle \langle \text{sring}_2 \rangle \mid \varepsilon$

$\langle \text{digit} \rangle \rightarrow 0 \mid \dots \mid 9$

III.2.29 Here is the start of a derivation.

$\langle \text{postal address} \rangle \Rightarrow \langle \text{name} \rangle \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{personal part} \rangle \langle \text{last name} \rangle \langle \text{opt suffix} \rangle \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{first name} \rangle \langle \text{last name} \rangle \langle \text{opt suffix} \rangle \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{last name} \rangle \langle \text{opt suffix} \rangle \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{opt suffix} \rangle \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \varepsilon \langle \text{EOL} \rangle \langle \text{street address} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $= \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{house num} \rangle \langle \text{street name} \rangle \langle \text{apt num} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{street name} \rangle \langle \text{apt num} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \langle \text{apt num} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \varepsilon \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $= \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{town} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{town name} \rangle, \langle \text{state or region} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{char string} \rangle, \langle \text{state or region} \rangle$   
 $\Rightarrow \langle \text{char string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{digit string} \rangle \langle \text{char string} \rangle \langle \text{EOL} \rangle \langle \text{char string} \rangle, \langle \text{char string} \rangle$

(A) From the above derivation steps we can easily get to the address.

(B) The above derivation steps get us most of the way there. The problem is the 'B' in 221B; it is not part of a digit string.

(C) Some are that you may need more than one line for the address, that some addresses do not have a house number, and that some names or addresses require non-ASCII letters.

### III.2.30

(A) These two rules suffice:  $S \rightarrow 11S \mid \varepsilon$ .

(B) As with the prior item, this is straightforward:  $S \rightarrow 111S \mid \varepsilon$ .

### III.2.31

(A) This is a sentence of length one.

$\langle \text{sentence} \rangle \Rightarrow \text{buffalo} \langle \text{sentence} \rangle \Rightarrow \text{buffalo} \varepsilon = \text{buffalo}$

Here is a sentence of length one.

$\langle \text{sentence} \rangle \Rightarrow \text{buffalo} \langle \text{sentence} \rangle \Rightarrow \text{buffalo buffalo} \langle \text{sentence} \rangle$   
 $\Rightarrow \text{buffalo buffalo} \varepsilon = \text{buffalo buffalo}$

And, a sentence of length three.

$$\begin{aligned}
\langle \text{sentence} \rangle &\Rightarrow \text{buffalo} \langle \text{sentence} \rangle \Rightarrow \text{buffalo buffalo} \langle \text{sentence} \rangle \\
&\Rightarrow \text{buffalo buffalo buffalo} \langle \text{sentence} \rangle \Rightarrow \text{buffalo buffalo buffalo } \varepsilon \\
&= \text{buffalo buffalo buffalo}
\end{aligned}$$

- (B) In English 'buffalo' is a noun referring to the American bison, or to a city in New York State, and it is a verb referring to fooling or confusing someone. So, the sentence of length one could be a command instructing the listener to fool someone. The sentence of length two could be a command instructing that same listener to fool all of a city in upstate New York. The length three sentence could exhort American bison in particular to confuse that entire city.

### III.2.32

- (A) This is a derivation of  $(a \cdot b)$ .

$$\begin{aligned}
\langle s \text{ expression} \rangle &\Rightarrow (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle) \Rightarrow (\langle \text{atomic symbol} \rangle \cdot \langle s \text{ expression} \rangle) \\
&\Rightarrow (\langle \text{atomic symbol} \rangle \cdot \langle \text{atomic symbol} \rangle) \Rightarrow (\langle \text{letter} \rangle \langle \text{atom part} \rangle \cdot \langle \text{atomic symbol} \rangle) \\
&\Rightarrow (\langle \text{letter} \rangle \varepsilon \cdot \langle \text{atomic symbol} \rangle) = (\langle \text{letter} \rangle \cdot \langle \text{atomic symbol} \rangle) \\
&\Rightarrow (a \cdot \langle \text{atomic symbol} \rangle) \Rightarrow (a \cdot \langle \text{letter} \rangle \langle \text{atom part} \rangle) \\
&\Rightarrow (a \cdot \langle \text{letter} \rangle \varepsilon) = (a \cdot \langle \text{letter} \rangle) \Rightarrow (a \cdot b)
\end{aligned}$$

- (B) This is a derivation of  $(a \cdot (b \cdot c))$ .

$$\begin{aligned}
\langle s \text{ expression} \rangle &\Rightarrow (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle) \\
&\Rightarrow (\langle s \text{ expression} \rangle \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (\langle s \text{ expression} \rangle \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (\langle \text{atomic symbol} \rangle \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (\langle \text{letter} \rangle \langle \text{atomic part} \rangle \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (a \langle \text{atomic part} \rangle \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (a \varepsilon \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) = (a \cdot (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle)) \\
&\Rightarrow (a \cdot (\langle \text{atomic symbol} \rangle \cdot \langle s \text{ expression} \rangle)) \Rightarrow (a \cdot (\langle \text{atomic symbol} \rangle \cdot \langle \text{atomic symbol} \rangle)) \\
&\Rightarrow (a \cdot (\langle \text{letter} \rangle \langle \text{atomic part} \rangle \cdot \langle \text{atomic symbol} \rangle)) \\
&\Rightarrow (a \cdot (\langle \text{letter} \rangle \langle \text{atomic part} \rangle \cdot \langle \text{letter} \rangle \langle \text{atomic part} \rangle)) \\
&\Rightarrow (a \cdot (\langle \text{letter} \rangle \varepsilon \cdot \langle \text{letter} \rangle \langle \text{atomic part} \rangle)) = (a \cdot (\langle \text{letter} \rangle \cdot \langle \text{letter} \rangle \langle \text{atomic part} \rangle)) \\
&\Rightarrow (a \cdot (\langle \text{letter} \rangle \cdot \langle \text{letter} \rangle \varepsilon)) = (a \cdot (\langle \text{letter} \rangle \cdot \langle \text{letter} \rangle)) \\
&\Rightarrow (a \cdot (b \cdot \langle \text{letter} \rangle)) \Rightarrow (a \cdot (b \cdot c))
\end{aligned}$$

- (C) This is the first few steps of a derivation of  $((a \cdot b) \cdot c)$ .

$$\begin{aligned}
\langle s \text{ expression} \rangle &\Rightarrow (\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle) \\
&\Rightarrow ((\langle s \text{ expression} \rangle \cdot \langle s \text{ expression} \rangle) \cdot \langle s \text{ expression} \rangle)
\end{aligned}$$

The rest proceeds as does the prior answer.

### III.2.33 The steps are tedious but routine.

$$\begin{aligned}
\langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \Rightarrow \langle \text{expr} \rangle + \langle \text{factor} \rangle \\
&\Rightarrow \langle \text{expr} \rangle + (\langle \text{expr} \rangle) \Rightarrow \langle \text{expr} \rangle + (\langle \text{term} \rangle) \\
&\Rightarrow \langle \text{expr} \rangle + (\langle \text{term} \rangle * \langle \text{factor} \rangle) \Rightarrow \langle \text{expr} \rangle + (\langle \text{factor} \rangle * \langle \text{factor} \rangle) \\
&\Rightarrow \langle \text{term} \rangle + (\langle \text{factor} \rangle * \langle \text{factor} \rangle) \Rightarrow \langle \text{factor} \rangle + (\langle \text{factor} \rangle * \langle \text{factor} \rangle) \\
&\Rightarrow a + (\langle \text{factor} \rangle * \langle \text{factor} \rangle) \Rightarrow a + (b * \langle \text{factor} \rangle) \\
&\Rightarrow a + (b * c)
\end{aligned}$$

### III.2.34

- (A) It is  $\mathcal{L} = \{\varepsilon\}$ .  
(B) This is one leftmost derivation.

$$S \Rightarrow \varepsilon$$

and here is another.

$$S \Rightarrow S \Rightarrow \varepsilon$$

### III.2.35 One leftmost derivation is this.

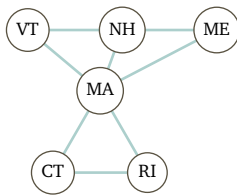


FIGURE 52, FOR QUESTION III.3.15: Adjacency relation among the New England states.

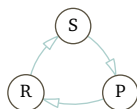


FIGURE 53, FOR QUESTION III.3.15: Relation of 'beats' among Rock, Paper, and Scissors.

$$\begin{aligned} \langle \text{bit-string} \rangle &\Rightarrow \langle \text{bit-string} \rangle \langle \text{bit-string} \rangle \Rightarrow \emptyset \langle \text{bit-string} \rangle \\ &\Rightarrow \emptyset \langle \text{bit-string} \rangle \langle \text{bit-string} \rangle \Rightarrow \emptyset \emptyset \langle \text{bit-string} \rangle \\ &\Rightarrow \emptyset \emptyset \emptyset \end{aligned}$$

Here is another.

$$\begin{aligned} \langle \text{bit-string} \rangle &\Rightarrow \langle \text{bit-string} \rangle \langle \text{bit-string} \rangle \Rightarrow \langle \text{bit-string} \rangle \emptyset \\ &\Rightarrow \langle \text{bit-string} \rangle \langle \text{bit-string} \rangle \emptyset \Rightarrow \emptyset \langle \text{bit-string} \rangle \emptyset \\ &\Rightarrow \emptyset \emptyset \emptyset \end{aligned}$$

**III.2.36**

- (A) One leftmost derivation is  $E \Rightarrow E - E \Rightarrow a - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a$ . A second one is  $E \Rightarrow E - E \Rightarrow E - E - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a$ .
- (B) Here we go:  $E \Rightarrow E - T \Rightarrow E - T - T \Rightarrow T - T - T \Rightarrow a - T - T \Rightarrow a - b - T \Rightarrow a - b - a$ .

$$\text{III.2.37} \quad S \Rightarrow aBSc \Rightarrow aBaBScc \Rightarrow aBaBabccc \Rightarrow aaBBabccc \Rightarrow aaBaBbccc \Rightarrow aaaBBbccc \Rightarrow aaaBbbccc \Rightarrow aaabbbccc$$

**III.3.15**

- (A) Figure 52 on page 92 shows the graph.
- (B) The graph is a loop, Figure 53 on page 92.
- (C) The graph is Figure 54 on page 93.
- (D) This graph is Figure 55 on page 93.
- (E) This prerequisite structure is Figure 56 on page 93.

**III.3.16**

- (A) See Figure 57 on page 93.
- (B) See Figure 58 on page 93.
- (C) It is  $\binom{n}{2} = n(n+1)/2$ .

**III.3.17**

- (A) There are ten vertices,  $\mathcal{N} = \{v_0, \dots, v_9\}$ . There are fifteen edges.

$$\mathcal{E} = \{v_0v_1, v_0v_4, v_0v_5, v_1v_2, v_1v_6, v_2v_3, v_2v_7, v_3v_4, v_3v_8, v_4v_9, v_5v_7, v_5v_8, v_6v_8, v_6v_9, v_7v_9\}$$

- (B) The walk  $w_1 = \langle v_0v_5, v_5v_7 \rangle$  has length two. The walk  $w_2 = \langle v_0v_1, v_1v_2, v_2v_7 \rangle$  has length three.
- (C) A closed walk is  $\langle v_4v_0, v_0v_1, v_1v_2, v_2v_3, v_3v_4 \rangle$ . An open one is  $\langle v_4v_9, v_9v_6, v_6v_8, v_8v_3, v_3v_2 \rangle$ .
- (D) One cycle is  $\langle v_5v_0, v_0v_1, v_1v_6, v_6v_8, v_8v_5 \rangle$ .
- (E) Yes, this graph is connected since we can find a path between any two vertices.

**III.3.18**

- (A) One possible picture is Figure 59 on page 93.

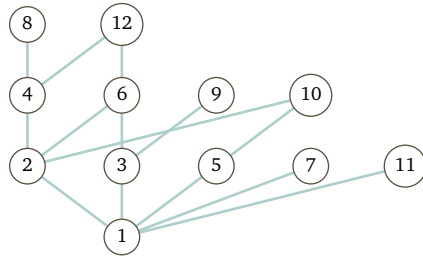


FIGURE 54, FOR QUESTION III.3.15: Divisibility relation among small numbers.

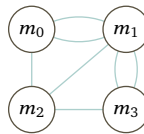


FIGURE 55, FOR QUESTION III.3.15: Bridge connection relation between land masses.

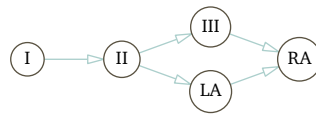


FIGURE 56, FOR QUESTION III.3.15: Prerequisite structure among courses.

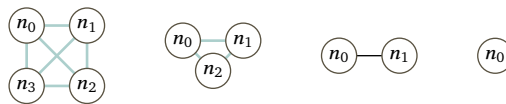


FIGURE 57, FOR QUESTION III.3.16: Complete graphs  $K_4$ ,  $K_3$ ,  $K_2$ , and  $K_1$ .

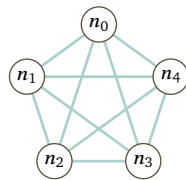


FIGURE 58, FOR QUESTION III.3.16: Complete graph  $K_5$ .

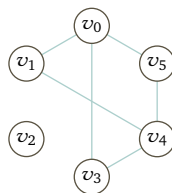


FIGURE 59, FOR QUESTION III.3.18: A graph with some edges.

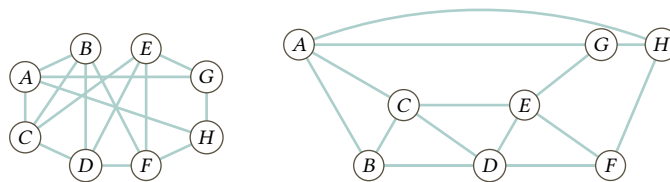


FIGURE 60, FOR QUESTION III.3.19: Drawings of the same graph, with edge crossings and without.

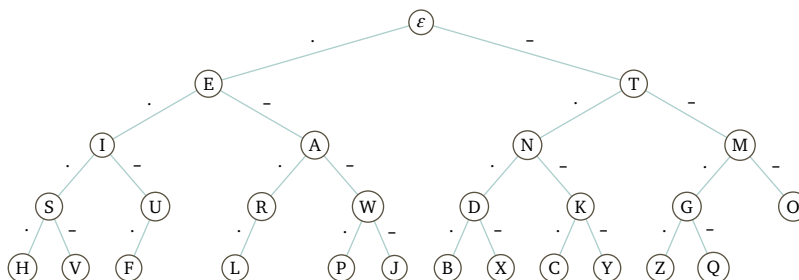


FIGURE 61, FOR QUESTION III.3.21: Prefix relation for Morse code representations of ASCII letters.

(B) Here is the matrix.

$$\mathcal{M}(\mathcal{G}) = \begin{matrix} & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \\ \begin{matrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

(c) From the picture, Figure 59 on page 93, there are clearly only two:  $G_0 = \{v_0, v_1, v_4, v_5\}$  and  $G_1 = \{v_0, v_3, v_4, v_5\}$ .

(d) It is the same two as in the prior item.

**III.3.19**

(A) On the left of Figure 60 on page 94 is the drawing with the vertices connected by the given edges.

(B) See the right side of Figure 60 on page 94 for the planar drawing.

**III.3.20**

	<i>Closed or open?</i>	<i>Vertices can repeat?</i>	<i>Edges can repeat?</i>
Walk	Either	Yes	Yes
Trail	Open	Yes	No
Circuit	Closed	Yes	No
Path	Open	No	No
Cycle	Closed	No	No

**III.3.21** See Figure 61 on page 94.

**III.3.22** Fix a vertex (if the tree is empty then the problem is trivial). Give it one color, and give its neighbors the second color. Assign the first color to all the vertices that are two away from the initial vertex. In general, where a vertex is  $k$  away from the initial vertex, give it the first color if  $k$  is even and the second color if  $k$  is odd. Trees don't have loops, so this assignment is well-defined.

**III.3.23**

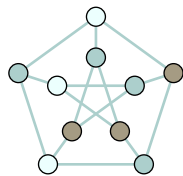


FIGURE 62, FOR QUESTION III.3.23: A 3-coloring of the Petersen graph.

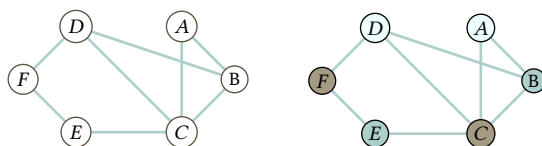


FIGURE 63, FOR QUESTION III.3.24: Relation of incompatibility between fish species.

- (A) The outside is a ring of five nodes. With only two colors, in a path around it we must alternate colors but then because it has an odd number of vertices, the fourth node and the fifth are then the same color.  
 (B) See Figure 62 on page 95.

**III.3.24**

- (A) See Figure 63 on page 95.  
 (B) The chromatic number is three; see the right-hand graph in the same figure.  
 (C) It is the number of fish tanks needed to safely keep all the fish species.

**III.3.25** The minimal number of frequencies is three. Figure 64 on page 96 gives a three-coloring. Starting with  $v_6$  and tracing around shows that no two-coloring will suffice.

**III.3.26** The directed graph is Figure 65 on page 96. Every type is compatible with itself so every vertex is shown with a loop. The type  $O^-$  can donate to everyone, as we see from its row below, so it is a universal donor. The type  $AB^+$  is a universal receptor, as we see from its column.

Here is the graph's adjacency matrix.

$$\begin{array}{c}
 \begin{array}{cccccccc}
 & O^- & O^+ & A^- & A^+ & B^- & B^+ & AB^- & AB^+ \\
 O^- & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 O^+ & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
 A^- & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
 A^+ & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
 B^- & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
 B^+ & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
 AB^- & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
 AB^+ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
 \end{array}
 \end{array}$$

**III.3.27** The graph in Example 3.2 has this degree sequence:  $\langle 4, 4, 3, 3, 2 \rangle$ . The vertices  $v_1$  and  $v_2$  have degree 4, the vertices  $v_3$  and  $v_4$  have degree 3, and the vertex  $v_0$  has degree 2.

In Example 3.4 the graph on the left has degree sequence  $\langle 3, 3, 3, 3 \rangle$ . The graph on the right has  $\langle 3, 3, 3, 3, 3, 3, 3, 3 \rangle$  (there are eight 3's, one for each corner of the cube).

**III.3.28** Both of these graphs are complete in the sense that every possible node-to-node connection is there

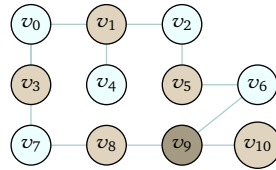


FIGURE 64, FOR QUESTION III.3.25: Cell tower three-coloring.

---

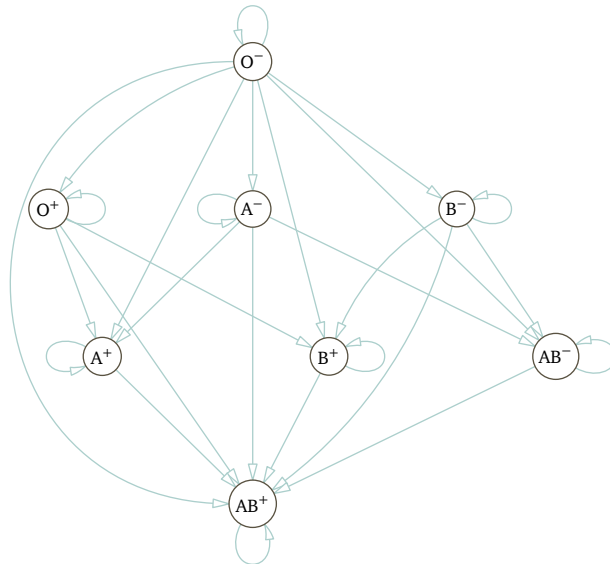


FIGURE 65, FOR QUESTION III.3.26: Compatibility for blood types.

---

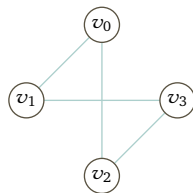


FIGURE 66, FOR QUESTION III.3.29: Graph associated with the given matrix.

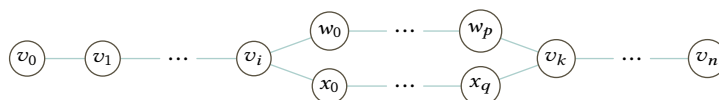


FIGURE 67, FOR QUESTION III.3.31: Two distinct paths makes a cycle.

(except that there are no loops). Here is the adjacency matrix for the graph on the left.

$$\begin{array}{c}
 u_0 \quad u_1 \quad u_2 \quad u_3 \\
 u_0 \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \\
 u_1 \\
 u_2 \\
 u_3
 \end{array}$$

And this is the adjacency matrix for the cube graph on the right.

$$\begin{array}{c}
 v_0 \quad v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7 \\
 v_0 \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix} \\
 v_1 \\
 v_2 \\
 v_3 \\
 v_4 \\
 v_5 \\
 v_6 \\
 v_7
 \end{array}$$

**III.3.29** The simple graph is Figure 66 on page 97.

**III.3.30**

- (A) The function is suggested by the names of the vertices:  $a \mapsto A$ ,  $b \mapsto B$ , etc. It is one-to-one and onto by inspection so it is a correspondence.
- (B) The edges on the left are:  $ax$ ,  $ay$ ,  $az$ ,  $bx$ ,  $by$ ,  $bz$ ,  $cx$ ,  $cy$ , and  $cz$ . The edges on the right are:  $AX$ ,  $AY$ ,  $AZ$ ,  $BX$ ,  $BY$ ,  $BZ$ ,  $CX$ ,  $CY$ , and  $CZ$ . Under the mapping the edge  $ax$  is associated with  $AX$ , the edge  $ay$  is associated with  $AY$ , etc., and this exhausts the edges on the right.

**III.3.31**

- (A) A path is a trail with distinct vertices (except that possibly the starting vertex equals the ending vertex). This trail meets that criteria.
- (B) The vertex  $B$  appears twice, not both at the start and end.
- (C) Figure 67 on page 97 illustrates. By definition every tree is connected, so there is a path. Suppose for contradiction that there are two unequal paths from  $v_0$  to  $v_n$ . Then there is some vertex  $v_i$  where the paths diverge and also a vertex  $v_k$  where they come together again. But that makes a cycle from  $v_i$ , around the one path to  $v_k$ , and back by the other path to return to  $v_i$ . The contradiction is that trees do not have cycles.

**III.3.32**

- (A) For each edge we choose two vertices. Thus the number of potential edges is  $\binom{n}{2} = n \cdot (n - 1)/2$ .
- (B) For each edge, we either include it or leave it out. So with  $n$  vertices there are  $2^{n \cdot (n-1)/2}$  graphs. (Some of these may be isomorphic, but they are different in that they involve different vertices connected.)

**III.3.33**

- (A) By the definition of graph isomorphism, Definition 3.14, if two graphs  $\mathcal{G}$  and  $\hat{\mathcal{G}}$  are isomorphic then there is a correspondence between their vertices. They therefore have the same number of vertices.
- (B) As in the prior item, the definition of graph isomorphism states that the edges correspond.
- (C) Suppose that  $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$  and  $\hat{\mathcal{G}} = \langle \hat{\mathcal{N}}, \hat{\mathcal{E}} \rangle$  are isomorphic via the function  $f$ . Let vertex  $v \in \mathcal{N}$  have degree  $k$ . Then in the set  $\mathcal{E}$  the vertex  $v$  occurs  $k$  times (counting once for edges in which the vertex occurs once, and twice for loops on  $v$ ).

Apply the correspondence. That is, consider the vertex  $f(v)$  and the set  $\hat{\mathcal{E}}$ . Because  $f$  gives a correspondence among edges, the vertex must occur  $k$  times in the set. So  $f(v)$  has degree  $k$ .

- (D) This is the same as the prior item, but instead of a single vertex  $v$  there is a sequence of vertices  $v_0, \dots, v_n$ .
- (E) Clearly they have the same degree sequence,  $\langle 3, 1, 1, 1, 1, 1 \rangle$ . Suppose that the two are isomorphic via the correspondence  $f$ . Then  $f$  must associate the two degree-3 vertices. The graph on the left has two unequal length 2 paths that start with its degree 3 vertex. The graph on the right does not have two.
- (F) The degree sequences differ. On the left the degree sequence is  $\langle 3, 3, 3, 3 \rangle$  while on the right it is  $\langle 4, 4, 4, 4, 4, 4, 4 \rangle$ .

**III.3.34**

- (A) There is a length 2 walk from  $v_i$  to  $v_j$  if and only if there is some  $v_k$  where both  $v_i v_k$  and  $v_k v_j$  are edges. Thus the number of such walks is  $m_{i,1}m_{1,j} + m_{i,2}m_{2,j} + \dots + m_{i,n}m_{n,j}$ , where each  $m_{a,b}$  is an entry from  $\mathcal{M}(\mathcal{G})$ . But that is the  $i, j$  entry of  $(\mathcal{M}(\mathcal{G}))^2$ .
- (B) For induction assume the hypothesis that the statement is true for  $n = 2, \dots, n = p$  and consider  $n = p + 1$ . There is a length  $p + 1$  walk from  $v_i$  to  $v_j$  if and only if there is some  $v_k$  where there is a length  $p$  walk from  $v_i$  to  $v_k$ , and also  $v_k v_j$  is an edge. By the inductive hypothesis the number of such walks is  $q_{i,1}m_{1,j} + q_{i,2}m_{2,j} + \dots + q_{i,n}m_{n,j}$ , where each  $q_{i,b}$  is an entry from  $(\mathcal{M}(\mathcal{G}))^p$  and  $m_{b,j}$  is an entry from  $\mathcal{M}(\mathcal{G})$ . We recognize that as the  $i, j$  entry of  $(\mathcal{M}(\mathcal{G}))^{p+1}$ .

**III.3.35**

- (A) The vertices are  $\mathcal{N}_0 = \{v_0, \dots, v_7\}$ . These are the edges.

$$\mathcal{E}_0 = \{v_0v_1, v_0v_2, v_0v_4, v_1v_3, v_2v_3, v_2v_6, v_4v_5, v_4v_6, v_5v_7, v_6v_7\}$$

- (B) The vertices are  $\mathcal{N}_1 = \{n_0, \dots, n_7\}$ . These are the edges.

$$\mathcal{E}_1 = \{n_0n_4, n_0n_5, n_1n_4, n_1n_5, n_2n_3, n_2n_6, n_3n_4, n_3n_7, n_5n_6, n_6n_7\}$$

- (C) The degree sequences are equal. Each is  $3, 3, 3, 3, 2, 2, 2, 2$ .

- (D) These are the images of edges from the left-hand graph.

edge of $\mathcal{G}_0$	$v_0v_1$	$v_0v_2$	$v_0v_4$	$v_1v_3$	$v_2v_3$	$v_2v_6$	$v_4v_5$	$v_4v_6$	$v_5v_7$	$v_6v_7$
image edge	$n_6n_2$	$n_6n_7$	$n_6n_5$	$n_2n_3$	$n_7n_3$	$n_7n_0$	$n_5n_1$	$n_5n_0$	$n_1n_4$	$n_0n_4$

That list has no  $n_3n_4$ , although  $\mathcal{G}_1$  has such an edge. Further, that list shows  $n_7n_0$ , although  $\mathcal{G}_1$  has no such edge. So the given correspondence between vertices does not induce a correspondence between edges. This map is not a graph isomorphism.

- (E) The graph  $\mathcal{G}_1$  has a pair of degree 3 vertices,  $n_6$  and  $n_3$ , that are separated by a degree 2 vertex,  $n_2$ . But  $\mathcal{G}_0$  has no such pair (there, the degree 3 vertices are separated by two degree 2 vertices).

**III.3.36**

- (A) We will keep track of a set  $R$  of reachable vertices. To start, put  $q_0$  into a set  $R_0$ . For the first step find all vertices connected to  $q_0$ , and enter them into the set  $R_1 \supseteq R_0$ . At step  $k + 1$  put all vertices connected to any vertex in  $R_k$  into a set  $R_{k+1} \supseteq R_k$ . When there are no new vertices then we are done (this must happen after finitely many steps because there are finitely many vertices, by our definition of a graph). The result is the set  $R$ . The unreachable vertices are those that are not members of  $R$ .

(B) For the graph on the left these are the steps.

<i>step</i> $k$	$R_k$
0	$\{w_0\}$
1	$\{w_0, w_1\}$
2	$\{w_0, w_1, w_2\} = R$

Here are the steps for the graph on the right.

<i>step</i> $k$	$R_k$
0	$\{w_0\}$
1	$\{w_0, w_1\}$
2	$\{w_0, w_1, w_3\} = R$

### III.A.5

$\langle \text{zip} \rangle ::= \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle [- \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle]$

$\langle \text{digit} \rangle ::= \emptyset \mid 1 \dots \mid 9$

### III.A.6

Here we stick to lower-case ASCII letters.

$\langle \text{palindrome} \rangle ::= a \langle \text{palindrome} \rangle a \mid b \langle \text{palindrome} \rangle b \mid \dots \mid z \langle \text{palindrome} \rangle z \mid \langle \text{letter} \rangle \mid \varepsilon$

$\langle \text{letter} \rangle ::= a \mid b \mid \dots \mid z$

You could omit  $\varepsilon$  if you don't like it as a palindrome.

### III.A.7

Here is one way to do it.

$\langle \text{course code} \rangle ::= \langle \text{dept} \rangle \langle \text{space} \rangle \langle \text{course-number} \rangle$

$\langle \text{dept} \rangle ::= \langle \text{two-letter} \rangle \mid \langle \text{three-letter} \rangle$

$\langle \text{two-letter} \rangle ::= \langle \text{letter} \rangle \langle \text{letter} \rangle$

$\langle \text{three-letter} \rangle ::= \langle \text{two-letter} \rangle \langle \text{letter} \rangle$

$\langle \text{course-number} \rangle ::= \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle$

$\langle \text{digit} \rangle ::= \emptyset \mid \dots \mid 9$

The  $\langle \text{space} \rangle$  produces a hard-to-show blank space.

You could also put the department names in on a case-by-case basis, as here.

$\langle \text{dept} \rangle ::= \text{MA} \mid \text{PSY} \mid \dots$

This has the disadvantage that if a department code changes then this part of the description also changes. It has the advantage of being more correct, of not allowing department names that are nonsensical.

### III.A.8

(A) This will do.

$\langle \text{pointfloat} \rangle ::= \langle \text{intpart} \rangle \langle \text{fraction} \rangle \mid \langle \text{intpart} \rangle . \mid \langle \text{fraction} \rangle$

(B)

$\langle \text{intpart} \rangle ::= \langle \text{intpart} \rangle \langle \text{digit} \rangle \mid \langle \text{digit} \rangle$

(C) This is just a list of cases

$\langle \text{exponent} \rangle ::= e \langle \text{intpart} \rangle \mid e+ \langle \text{intpart} \rangle \mid e- \langle \text{intpart} \rangle \mid E \langle \text{intpart} \rangle \mid E+ \langle \text{intpart} \rangle \mid E- \langle \text{intpart} \rangle$

### III.A.9

(A) This is a grammar for standard notation.

$\langle \text{start} \rangle ::= \langle \text{thousands} \rangle \langle \text{hundreds} \rangle \langle \text{tens} \rangle \langle \text{ones} \rangle$

$\langle \text{thousands} \rangle ::= M^*$

$\langle \text{hundreds} \rangle ::= C \mid CC \mid CCC \mid CCCC \mid D \mid DC \mid DCC \mid DCCC \mid DCCCC \mid \varepsilon$

$\langle \text{tens} \rangle ::= X \mid XX \mid XXX \mid XXXX \mid L \mid LX \mid LXX \mid LXXX \mid LXXXX \mid \varepsilon$

$\langle \text{ones} \rangle ::= I \mid II \mid III \mid IIII \mid V \mid VII \mid VIII \mid VIIII \mid \varepsilon$

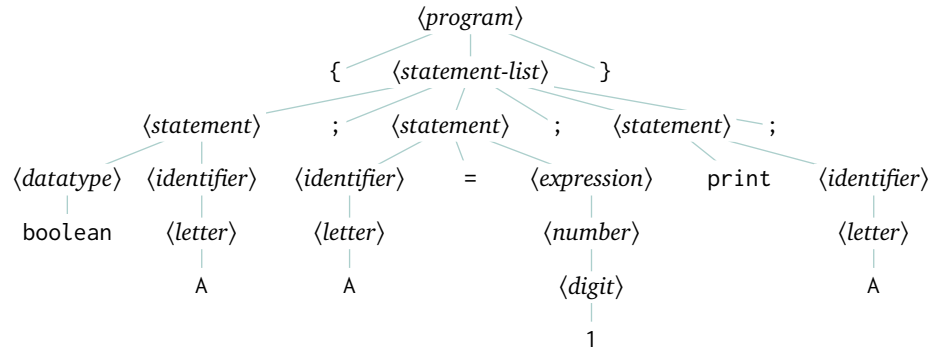


FIGURE 68, FOR QUESTION III.A.10: Parse tree for C-like language.

(B) This is a grammar for subtractive notation.

$$\begin{aligned} \langle \text{start} \rangle &::= \langle \text{thousands} \rangle \langle \text{hundreds} \rangle \langle \text{tens} \rangle \langle \text{ones} \rangle \\ \langle \text{thousands} \rangle &::= M^* \\ \langle \text{hundreds} \rangle &::= CM \mid CD \mid (D \mid \varepsilon) (\varepsilon \mid C \mid CC \mid CCC) \\ \langle \text{tens} \rangle &::= XC \mid XL \mid (L \mid \varepsilon) (\varepsilon \mid X \mid XX \mid XXX) \\ \langle \text{ones} \rangle &::= IX \mid IV \mid (V \mid \varepsilon) (\varepsilon \mid I \mid II \mid III) \end{aligned}$$

### III.A.10

(A) Here is a derivation. We feel free to use the BNF notation to, for instance, put in multiple  $\langle \text{statement} \rangle$ 's on the third line instead of putting them in one at a time.

$$\begin{aligned} \langle \text{program} \rangle &\Rightarrow \{ \langle \text{statement-list} \rangle \} \\ &\Rightarrow \{ \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \langle \text{data-type} \rangle \langle \text{identifier} \rangle ; \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean} \langle \text{identifier} \rangle ; \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean} \langle \text{letter} \rangle ; \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; \langle \text{statement} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; \langle \text{identifier} \rangle = \langle \text{expression} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; \langle \text{letter} \rangle = \langle \text{expression} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = \langle \text{expression} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = \langle \text{number} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = \langle \text{digit} \rangle ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = 1 ; \langle \text{statement} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = 1 ; \text{print} \langle \text{identifier} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = 1 ; \text{print} \langle \text{letter} \rangle ; \} \\ &\Rightarrow \{ \text{boolean } A ; A = 1 ; \text{print } A ; \} \end{aligned}$$

Figure 68 on page 100 shows the tree.

(B) Yes, because all derivations must go from  $\langle \text{program} \rangle$  through  $\langle \text{statement-list} \rangle$ , which forces them to have curly braces.

**III.A.11** In this partial derivation we feel free to leverage the BNF notation to, for instance, put in multiple  $\langle \text{s-expression} \rangle$ 's for a  $\langle \text{list} \rangle$  at one time, instead of putting them in one at a time.

$$\begin{aligned} \langle \text{s-expression} \rangle &\Rightarrow \langle \text{list} \rangle \\ &\Rightarrow ( \langle \text{s-expression} \rangle \langle \text{s-expression} \rangle \langle \text{s-expression} \rangle ) \\ &\Rightarrow ( \langle \text{s-expression} \rangle \langle \text{list} \rangle \langle \text{s-expression} \rangle ) \\ &\Rightarrow ( \langle \text{s-expression} \rangle ( \langle \text{s-expression} \rangle \langle \text{s-expression} \rangle ) \langle \text{s-expression} \rangle ) \\ &\Rightarrow ( \langle \text{atomic-symbol} \rangle ( \langle \text{s-expression} \rangle \langle \text{s-expression} \rangle ) \langle \text{s-expression} \rangle ) \end{aligned}$$

$\Rightarrow ( \langle \text{letter} \rangle \langle \text{atomic-part} \rangle ( \langle s\text{-expression} \rangle \langle s\text{-expression} \rangle ) \langle s\text{-expression} \rangle )$   
 $\Rightarrow ( \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{atomic-part} \rangle ( \langle s\text{-expression} \rangle \langle s\text{-expression} \rangle ) \langle s\text{-expression} \rangle )$   
 $\Rightarrow ( \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{atomic-part} \rangle ( \langle s\text{-expression} \rangle \langle s\text{-expression} \rangle ) \langle s\text{-expression} \rangle )$   
 $\Rightarrow ( \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle ( \langle s\text{-expression} \rangle \langle s\text{-expression} \rangle ) \langle s\text{-expression} \rangle )$   
 $\Rightarrow ( c \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle ( \langle s\text{-expression} \rangle \langle s\text{-expression} \rangle ) \langle s\text{-expression} \rangle )$

Expanding the remaining nonterminals to letters is tedious, so we will stop here.

### III.A.12

(A) Here we will leverage the BNF notation to, for instance, substitute for multiple nonterminals, instead of putting them in one at a time.

$\langle \text{format-spec} \rangle \Rightarrow \emptyset \langle \text{width} \rangle \langle \text{type} \rangle$   
 $\Rightarrow \emptyset \langle \text{integer} \rangle \langle \text{type} \rangle$   
 $\Rightarrow \emptyset \langle \text{digit} \rangle \langle \text{type} \rangle$   
 $\Rightarrow \emptyset 3 \langle \text{type} \rangle$   
 $\Rightarrow \emptyset 3 f$

(B)  $\langle \text{format-spec} \rangle \Rightarrow \langle \text{sign} \rangle \# \emptyset \langle \text{width} \rangle \langle \text{type} \rangle$   
 $\Rightarrow + \# \emptyset \langle \text{width} \rangle \langle \text{type} \rangle$   
 $\Rightarrow + \# \emptyset \langle \text{integer} \rangle \langle \text{type} \rangle$   
 $\Rightarrow + \# \emptyset \langle \text{digit} \rangle \langle \text{type} \rangle$   
 $\Rightarrow + \# \emptyset 2 \langle \text{type} \rangle$   
 $\Rightarrow + \# \emptyset 2 X$



## Chapter IV: Automata

### IV.1.16

	<i>Step</i>	<i>Configuration</i>		<i>Step</i>	<i>Configuration</i>
(A)	0	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a b b a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		3	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>2</sub></div>
	1	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b b a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>		4	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> </div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>
	2	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>2</sub></div>			

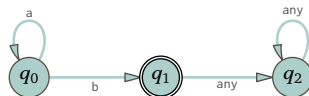
	<i>Step</i>	<i>Configuration</i>		<i>Step</i>	<i>Configuration</i>
(B)	0	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b a b</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		2	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>
	1	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a b</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		3	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> </div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>2</sub></div>

	<i>Step</i>	<i>Configuration</i>		<i>Step</i>	<i>Configuration</i>		<i>Step</i>	<i>Configuration</i>
(C)	0	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b b a a b b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		3	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a b b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>		6	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>2</sub></div>
	1	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b a a b b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		4	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>		7	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>
	2	<div style="border: 1px solid black; padding: 2px; display: inline-block;">a a b b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>0</sub></div>		5	<div style="border: 1px solid black; padding: 2px; display: inline-block;">b a a</div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>2</sub></div>		8	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> </div> <div style="border: 1px solid black; padding: 2px; display: inline-block; margin-left: 20px;">q<sub>1</sub></div>

IV.1.17 False. An example is that Example 1.7's machine recognizes the set containing each of these: 0, 1, 2, ...

IV.1.18 First, "n is infinite" is wrong. That set is a language {b, ab, a<sup>2</sup>b, ...} and in each string from that language the number of a's is finite.

Second, this Finite State machine recognizes the language.



IV.1.19 A language can contain the empty string. A machine can recognize the empty string. But we have no definition for a language recognizing a string.

IV.1.20 Each input character consumed causes one transition. So an input string of length  $n$  causes the machine to undergo  $n$  transitions. (This includes that the empty string  $\epsilon$  causes no transitions.) With  $n$  transitions, the machine will have visited  $n + 1$  many not necessarily distinct states, including the start state.

### IV.1.21

(A) A string is in this language if and only if it has some number of a's, followed by a single b, followed by the same number of a's. That is, the two strings of a's have the same length. Five elements of the language are aba, aabaa, b, aaabaaa, and aaaabaaaa. Five non-elements are ba, aaba, a, abba, and bbabb.

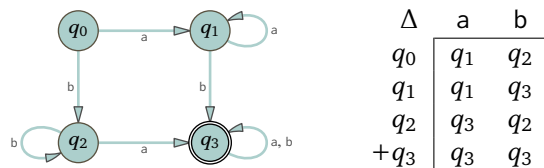


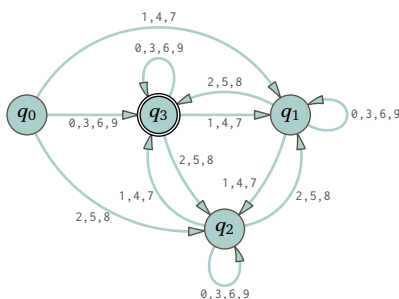
FIGURE 69, FOR QUESTION IV.1.24: Finite State machine accepting strings with at least one a and b.

- (B) The strings in this language have some number of a's followed by a single b, followed by a number of a's. The two numbers of a's need not be the same, and may be zero. Five language elements are abaa, aabaa, ab, baaaa, and b. Five non-elements are bab, aabb, bb, aabaab, and a.
- (C) The criteria for a string to be in this language is that it must start with b and be followed by some number of a's. That could be zero-many a's. Five elements of the language are baa, ba, b, ba<sup>3</sup>, and ba<sup>4</sup>. Five non-elements are a,  $\epsilon$ , abaaa, baaab, and bb.
- (D) A string is in this language if and only if it consists of some number of a's followed by a b, followed by some number of a's, where the second string of a's has two more than the first string. Five members of the language are abaaa, aabaaaa, a<sup>3</sup>ba<sup>5</sup>, a<sup>4</sup>ba<sup>6</sup>, and baa. Five non-members are aba, abaaaa, ba, b, and bab.
- (E) This language has members that are doubles in that a member consists of the concatenation of two copies of some string. Five members of the language are aaabaab, babbab, aa, bbbabbbba, and abbaabba. Five non-members are aba, b, aabaab, babaa, and bbb.

**IV.1.22**

- (A) For Example 1.6 the only accepting state is  $q_2$ . In Example 1.7 also, the only accepting state is  $q_2$ . For Example 1.8 the accepting states are  $q_3$ ,  $q_6$ , and  $q_8$ . For Example 1.9 it is  $q_0$ .
- (B) Only Example 1.9 accepts the empty string  $\epsilon$  because only in this machine is  $q_0$  an accepting state.
- (C) For Example 1.6 the shortest accepted string is the length two string  $\emptyset\emptyset$ . For Example 1.7 it is any single-digit length one string, such as 9. For Example 1.8 any of the length three strings jpg, png, and pdf. For Example 1.9 it is the empty string  $\epsilon$ .

**IV.1.23** Modify the machine so that its initial state is not an accepting state. Here there are two states whose intuitive meaning is that the sum of the digits seen so far is congruent to 0 modulo 3.



**IV.1.24**

- (A) Five members are ab, baab, ba, b<sup>7</sup>a<sup>3</sup>, and ab<sup>6</sup>a. Five nonmembers are aaaa, bbbb, a,  $\epsilon$ , and b. We take the meaning of the states to be as here.

State	Description
$q_0$	start state
$q_1$	have seen at least one a, no b's
$q_2$	have seen at least one b, no a's
$q_3$	at least one of each

The circle drawing and transition table are Figure 69 on page 104.

- (B) Five members are: bb,  $\epsilon$ , aba, a, and bbbaab. Five nonmembers are aaa, baaa, a<sup>5</sup>, bababa, and a<sup>10</sup>. We take the states with this meaning.

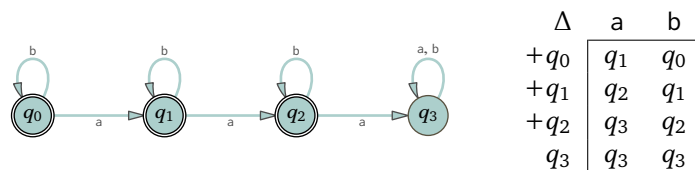


FIGURE 70, FOR QUESTION IV.1.24: Finite State machine accepting strings with fewer than three a's.

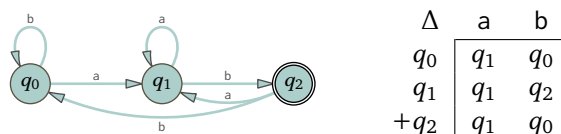


FIGURE 71, FOR QUESTION IV.1.24: Finite State machine accepting strings that end in ab.

State	Description
$q_0$	start state, no a's seen yet
$q_1$	have seen one a, and maybe some number of b's
$q_2$	have seen two a's
$q_3$	have seen three or more a's

For the circle diagram and transition table, see Figure 70 on page 105.

- (c) Five members are ab, bab, aab, baab, and abab. Five nonmembers: a, bbb,  $\epsilon$ , b, and abababb. We use the states with this meaning.

State	Description
$q_0$	start state, have not just seen an a
$q_1$	have seen an a, waiting for a b's
$q_2$	have just seen ab

The transition diagram and table are Figure 71 on page 105.

- (d) Five members are aabb, aaabb, aabbb,  $a^7b^9$ , and  $a^2b^{200}$ . Five nonmembers are  $\epsilon$ , a, ab, ba, and b. We use the states with this meaning.

State	Description
$q_0$	start state, have not just seen anything
$q_1$	have seen one character, an a
$q_2$	have seen at least two a's
$q_3$	have seen at least two a's and one b
$q_4$	have seen at least two a's and at least two b's
$q_5 = e$	error

See Figure 72 on page 106 for the circle diagram and the table.

- (e) Five members are aabba, aabb, abb, bb, and  $a^5bb^7$ . Five nonmembers are  $\epsilon$ , bab, aabaa, aabbbbaa, and aabbaab. We take the states with this meaning.

State	Description
$q_0$	have so far seen some number of a's, including zero-many
$q_1$	have seen some a's followed by one b
$q_2$	seen some a's, two b's, and some a's
$e$	error

The two representations of the transition function are Figure 73 on page 106.

IV.1.25 This is the graph picture.

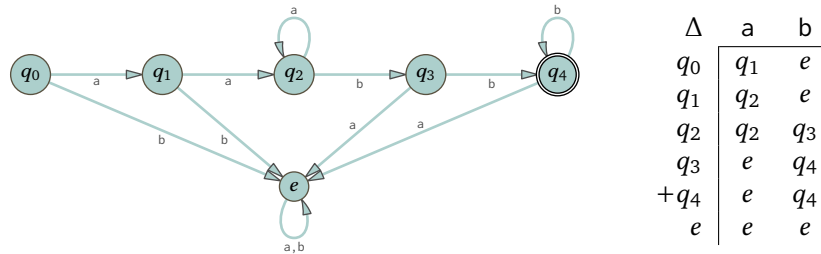


FIGURE 72, FOR QUESTION IV.1.24: Finite State machine for strings of at least two a's and then at least two b's.

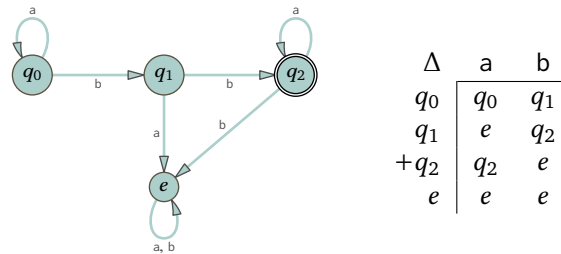
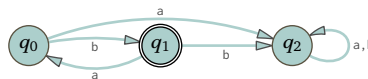


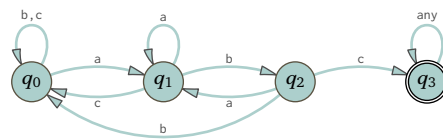
FIGURE 73, FOR QUESTION IV.1.24: Finite State machine for strings with some a's, then two b's, then some a's.



The language is  $\{\sigma \in \{a, b\}^* \mid \sigma = b(ab)^n \text{ for } n \in \mathbb{N}\}$ .

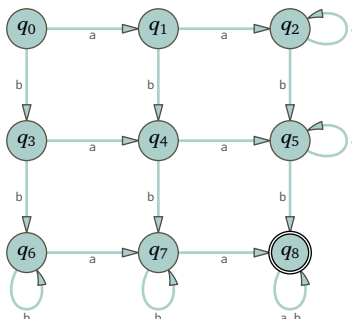
**IV.1.26** The language is the set of strings over  $\Sigma = \{a, b\}$  the contain exactly one b.

**IV.1.27** To accept strings containing the substring abc the machine must have a state whose intuitive meaning is “have just seen a, next looking for bc.” That state is  $q_1$ . The state  $q_2$  means that the machine has just processed a substring ab and is looking for c. And  $q_3$  means the machine has seen the substring abc. Finally,  $q_0$  is the state the machine is in if it has so far not seen even the first character in a potential substring abc. Here is the machine.



Note that if the machine processes ab followed by b then it returns to  $q_0$ , but if it processes ab followed by a then it does not return to  $q_0$ , it passes to  $q_1$ . This follows from the intuitive meaning of  $q_1$ .

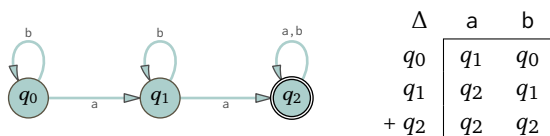
**IV.1.28** Five elements of the language are aabb, bbaa, abab, aababb, and  $a^3b^4$ . Five non-elements are  $\epsilon$ , a, b, aab, and bbbbabbbb. Here is the machine.



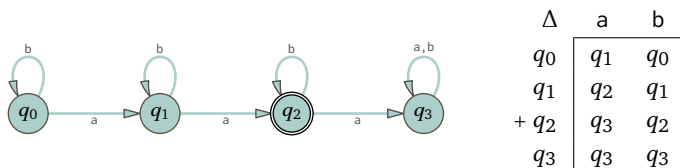
The intuitive meaning of the states in the first row,  $q_0$ ,  $q_1$ , and  $q_2$ , is that when the machine is in those states it has so far seen zero-many b's. States in the second row are where the machine has so far seen one b. The third row contains the states where the machine has seen two b's. Similarly, the first column contains the states where the machine has so far seen no a's, in the second column are states where the machine has seen a single a, and in the final column the machine has seen two a's.

**IV.1.29**

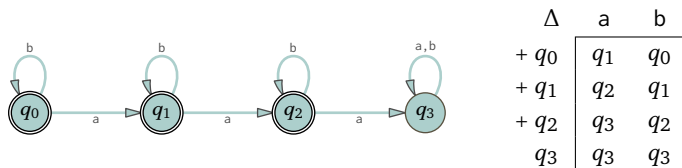
- (A) Five strings that are elements of the language are aa, aaa, aab, aba, and baa. Five that are not are  $\epsilon$ , a, b, ab, and ba. This machine accepts only strings in that language.



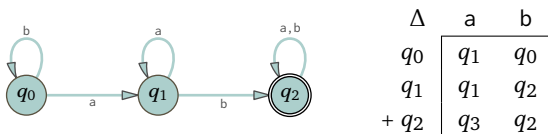
- (B) Five elements of the language are aa, aab, aba, baa, and aabb. Five that are not in the language are  $\epsilon$ , a, ba, aaa, and baaa.



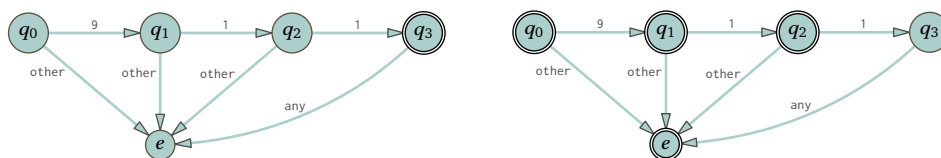
- (C) Five strings that are elements of the language are  $\epsilon$ , a, b, aa, and ab. Five that are not are aaa, aaab, aaba, abaa, and baaa.



- (D) Five from the language are ab, aab, aba, abb, and bab. Five that are not in the language are  $\epsilon$ , a, b, ba, and bbbbaaaa.



**IV.1.30** The first machine accepts only 911 while the second accepts anything but 911. Note that in the two machines the sets of accepting states are complementary.



**IV.1.31** Here are the length three strings.

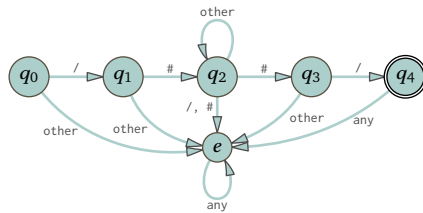
Input $\sigma$	aaa	aab	aba	abb	baa	bab	bba	bbb
$\hat{\Delta}(\sigma)$	$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_0$

And here are the length four strings.

Input $\sigma$	aaaa	aaab	aaba	aabb	abaa	abab	abba	abbb
$\hat{\Delta}(\sigma)$	$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_2$

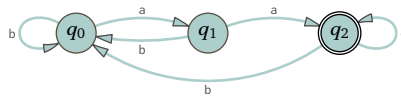
baaa	baab	baba	babb	bbaa	bbab	bbba	bbbb
$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_2$	$q_1$	$q_0$

IV.1.32

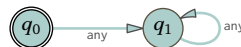


IV.1.33

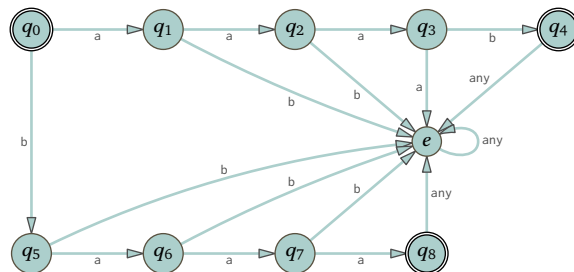
(A) Five that are in the language are aa, aaa, baa, aaaa, and abaa. Five that are not are  $\epsilon$ , a, b, ab ba, and aba. Here is the machine.



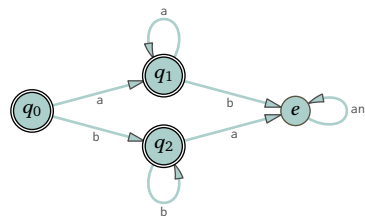
(B) The only string in the language is  $\epsilon$ . That is,  $\mathcal{L} = \{\epsilon\}$ . Five not in the language are a, b, aa, ab, and ba. The machine is this.



(C) There are only two strings in the language:  $\mathcal{L} = \{aaab, baaa\}$ . Five strings not in the language are  $\epsilon$ , a, b, aa, and ab. The machine is this.



(D) Five strings in the language are  $\epsilon$ , a, b, aa, and bb. Five not in the language are ab, ba, aab, aba, and baa. This is the machine.



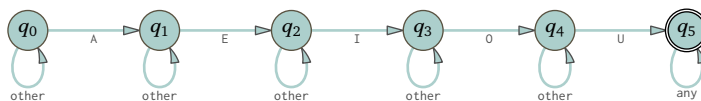
Note that  $q_0$  is an accepting state.

IV.1.34 It outputs the length one string containing the start state,  $\hat{\Delta}(\epsilon) = q_0$ .

IV.1.35

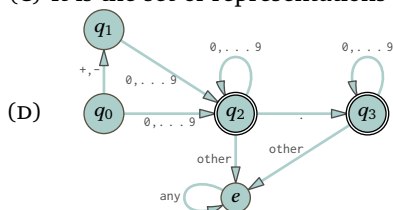
- (A) 1, 3, 4, 5, 6, 7, 8, 9
- (B) 1, 2, 3, 4, 6, 7, 8, 9

IV.1.36 Note that although the string  $\sigma = \text{AEAI} \text{AOAUA}$  has an A that occurs after an E, nonetheless the machine should accept it because there exists a subsequence of string elements consisting of the ordered vowels.



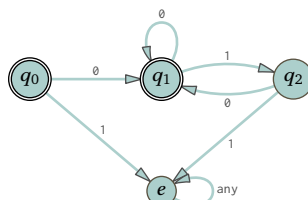
**IV.1.37**

- (A) Five that are: +1, 123, -123.45, -0.0, and 0. Five that are not: +, +-, . . . , 12.3.4, and 2+3
- (B) No, before the period must come at least one digit.
- (C) It is the set of representations of real numbers with integer part, a decimal point, and a decimal part.

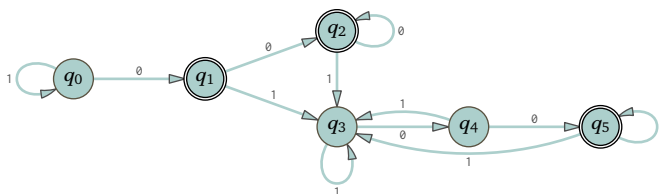


**IV.1.38**

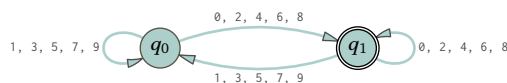
- (A) Five that are in the language are 010, 0010, 0, 0101000, and  $\epsilon$ . Five that are not are 10, 0110, 0101, 01001, and 1.



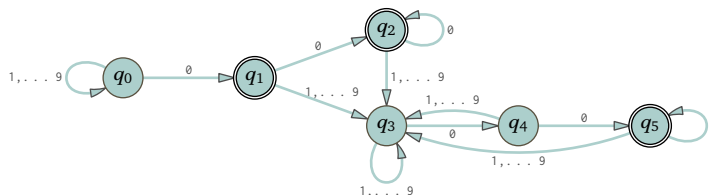
- (B) A string is a binary representation of a multiple of 4 if and only if it ends in two zero's, 00, or it is 0. Five in the language are 100, 0100, 1000, 10100, and 0. Five not in the language are 10, 1, 1001, 001, and  $\epsilon$ .



- (C) Five strings in the language are 0, 20, 12, 88, and 08. Five that are not are 1, 01, 2221, 17, and  $\epsilon$ .



- (D) This is basically the same as the second item. A string is a decimal representation of a multiple of 100 if and only if it ends in two zero's, 00, or it is 0. Five in the language are 100, 0100, 2000, 60300, and 0. Five not in the language are 10, 8, 1009, 001, and  $\epsilon$ .



**IV.1.39** A number is divisible by four if and only if its representation ends in 00, 04, . . . 96, or the single-digit cases where the representation is 0, 4, or 8. That's 28 cases. A picture would be cluttered but this is a finite language and for any finite language there is a Finite State machines that recognizes that language. (We shall see an alternative approach that makes a clearer picture in the section on nondeterminism.)

**IV.1.40** This is the computation.

$$\langle q_0, 2332 \rangle \vdash \langle q_2, 332 \rangle \vdash \langle q_2, 32 \rangle \vdash \langle q_2, 2 \rangle \vdash \langle 1_0, \epsilon \rangle$$

The state  $q_1$  is not an accepting state so the machine rejects 2332.

IV.1.41 Recall the transition function for that example.

$\Delta$	$\emptyset$	$1$
$q_0$	$q_1$	$q_3$
$q_1$	$q_2$	$q_4$
$+ q_2$	$q_2$	$q_5$
$q_3$	$q_4$	$q_0$
$q_4$	$q_5$	$q_1$
$q_5$	$q_5$	$q_2$

(A) If the input string is  $\sigma = \emptyset$  then peel off the final character with  $\sigma = \sigma_0 \hat{\ } \emptyset = \varepsilon \hat{\ } \emptyset$  and the definition gives this.

$$\hat{\Delta}(\emptyset) = \Delta(\hat{\Delta}(\varepsilon), \emptyset) = \Delta(q_0, \emptyset) = q_1$$

(Note that in the left-most expression  $\emptyset$  is a length one string, while in the rest of the above equation  $\emptyset$  is a character, and we are ignoring this uninteresting point.) Similarly,  $\hat{\Delta}(1) = q_3$ .

(B) For  $\sigma = \emptyset\emptyset$  use the prior item's computation of  $\hat{\Delta}(\emptyset)$ .

$$\hat{\Delta}(\emptyset\emptyset) = \Delta(\hat{\Delta}(\emptyset), \emptyset) = \Delta(q_1, \emptyset) = q_2$$

The other three are similar.

$$\hat{\Delta}(\emptyset 1) = q_4 \quad \hat{\Delta}(1\emptyset) = q_4 \quad \hat{\Delta}(11) = q_0$$

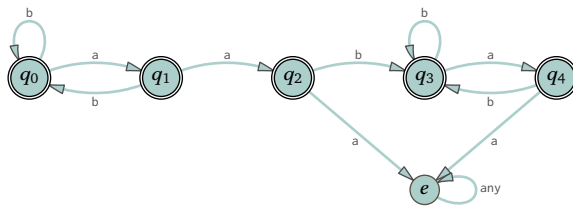
(C) For each of these use the prior item's computation. Here is the first.

$$\hat{\Delta}(\emptyset\emptyset\emptyset) = \Delta(\hat{\Delta}(\emptyset\emptyset), \emptyset) = \Delta(q_2, \emptyset) = q_2$$

The others are computed in the same way.

$\hat{\Delta}(\emptyset\emptyset\emptyset)$	$\hat{\Delta}(\emptyset\emptyset 1)$	$\hat{\Delta}(\emptyset 1\emptyset)$	$\hat{\Delta}(\emptyset 11)$	$\hat{\Delta}(1\emptyset\emptyset)$	$\hat{\Delta}(1\emptyset 1)$	$\hat{\Delta}(11\emptyset)$	$\hat{\Delta}(111)$
$q_2$	$q_5$	$q_5$	$q_1$	$q_5$	$q_1$	$q_1$	$q_3$

IV.1.42



IV.1.43

(A) There is one such machine.

$\Delta$	$\emptyset$	$1$
$q_0$	$q_0$	$q_0$

(B) This list has 16 machines.

$\Delta_0$	$\emptyset$	$1$	$\Delta_1$	$\emptyset$	$1$	$\Delta_2$	$\emptyset$	$1$	$\dots$	$\Delta_{15}$	$\emptyset$	$1$
$q_0$	$q_0$	$q_0$	$q_0$	$q_0$	$q_0$	$q_0$	$q_0$	$q_0$		$q_0$	$q_1$	$q_1$
$q_1$	$q_0$	$q_0$	$q_1$	$q_0$	$q_1$	$q_1$	$q_1$	$q_0$		$q_1$	$q_1$	$q_1$

(C) Fix  $n$  states,  $Q = \{q_0, \dots, q_{n-1}\}$ . Each line of the transition table has two columns so the number of distinct lines is  $n \cdot n = n^2$ . An overall table chooses  $n$  such lines. There are  $n^2 \cdot n^2 \cdot \dots \cdot n^2 = n^{2n}$  many ways to do that.

(D) For each of the  $n$  states, either it is final or it is not. The additional factor of  $2^n$  makes the total number of distinct tables equal to  $2^n n^{2n}$ .

state	meaning	with wolf	with goat	with cabbage	with nothing
$s_0$	n: mwg, f: -	$s_{12}$	$s_{10}$	$s_9$	$s_8$
$s_1$	n: mwg, f: c	$s_{13}$	$s_{11}$	$e$	$s_9$
$s_2$	n: mwc, f: g	$s_{14}$	$e$	$s_{11}$	$s_{10}$
$s_3$	n: mw, f: gc	$e$	$e$	$e$	$e$
$s_4$	n: mgc, f: w	$e$	$s_{14}$	$s_{13}$	$s_{12}$
$s_5$	n: mg, f: wc	$e$	$s_{15}$	$e$	$s_{13}$
$s_6$	n: mc, f: wg	$e$	$e$	$e$	$e$
$s_7$	n: m, f: wgc	$e$	$e$	$e$	$e$
$s_8$	n: wgc, f: m-	$e$	$e$	$e$	$e$
$s_9$	n: wg, f: mc	$e$	$e$	$e$	$e$
$s_{10}$	n: wc, f: mg	$e$	$s_0$	$e$	$s_2$
$s_{11}$	n: w, f: mgc	$e$	$s_1$	$s_{10}$	$s_3$
$s_{12}$	n: gc, f: mw	$e$	$e$	$e$	$e$
$s_{13}$	n: g, f: mwc	$s_1$	$e$	$s_4$	$s_5$
$s_{14}$	n: c, f: mwg	$s_2$	$s_4$	$e$	$s_6$
$s_{15}$	n: -, f: mwg	$s_{15}$	$s_{15}$	$s_{15}$	$s_{15}$
$e$	error	$e$	$e$	$e$	$e$

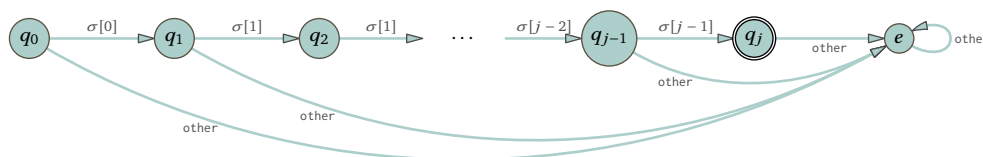
FIGURE 74, FOR QUESTION IV.1.44: Finite state machine for Wolf-Goat-Cabbage.

**IV.1.44** We can have the man, the wolf, the goat, and the cabbage, on the near or far side. The allowed transitions are: man crosses with wolf, man crosses with goat, man crosses with cabbage, and man crosses by himself. Here are the states; some of them lead to an error state, as readers of the 700's would have understood.

In the Finite State machine shown in Figure 74 on page 111 the start state is  $s_0$  and the only accepting state is  $s_{15}$ . In words: *The man first brings the goat to the far side and leaves it there. He goes back and brings the wolf with him, leaving it on the far side, and brings back the goat. He leaves the goat on the initial side, takes the cabbage and brings it to the far side. Finally, he goes back to the original shore, and takes the goat to bring it to the far side.*

**IV.1.45** This is based on Example 1.8.

We will prove this by induction on the number of strings in the language. If the language is empty then we use any Finite State machine with the right alphabet and no accepting states. If the language has one string  $\sigma$  then we write a Finite State machine whose state transitions track the characters of  $\sigma$  as does the top line in the example. That is, we have this.

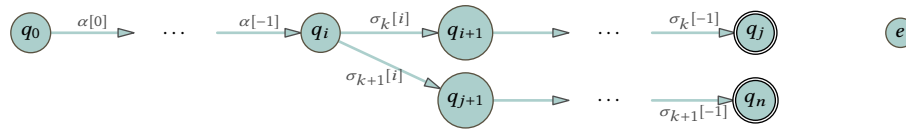


As the diagram shows, the last state in that sequence is an accepting state. Also as shown, all transitions for characters not in the sequence go to  $e$ . (If  $\sigma = \epsilon$  then there are two states, with the initial state  $q_0$  an accepting state and all transitions going to the error state  $e$  that is not an accepting state.)

For the inductive case, assume that there is a machine for any language of size  $n = 1, \dots, n = k$  and consider a language  $\mathcal{L}$  with  $k + 1$ -many strings. Order the alphabet  $\Sigma$ ; in the example we use the grade school alphabetical ordering. This character ordering gives rise to an ordering of the strings, called lexicographic ordering. List  $\mathcal{L}$ 's strings lexicographically:  $\sigma_0, \dots, \sigma_k, \sigma_{k+1}$ .

The induction assumption gives a machine for the language consisting of the strings  $\sigma_0, \dots, \sigma_k$ . The example shows what to do next; it has the two final strings png and pdf and the machine splits where the two strings split, after the p. Following that model, find the longest common substring for  $\sigma_k$  and  $\sigma_{k+1}$ , so that

$\sigma_k = \alpha \hat{\ } \beta$ , and  $\sigma_{k+1} = \alpha \hat{\ } \gamma$ , and the initial character of  $\beta$  is not equal to the initial character of  $\gamma$ . (It could be that  $\beta = \epsilon$ . It could also be that  $\alpha = \epsilon$ .) As in the example, split the machine after it has traversed the common string.



(To be more readable this figure omits all the transitions involving  $e$ .)

**IV.1.46** Recall that an alphabet is finite, by definition.

- (A) We will show that there are infinitely many different Finite State machines. Fix any  $\Sigma$  and suppose  $s \in \Sigma$ . The machines below differ in the number of states, and in that they recognize different languages. (If  $\Sigma$  contains characters other than  $s$  then use the second column.)

$\Delta$	$s$	$other$	$\Delta$	$s$	$other$	$\Delta$	$s$	$other$	
+ $q_0$	$q_0$	$q_0$	+ $q_0$	$q_1$	$q_0$	+ $q_0$	$q_1$	$q_0$	...
				$q_1$	$q_0$		$q_2$	$q_0$	

That shows, under a reasonable interpretation of what it means for machines to differ, that there are at least countably many Finite State machines. (The technical wording is that these machines are not isomorphic.)

- (B) The argument that there are no more than countably many machines also requires an interpretation. Assume that the set of possible states is countable,  $Q = \{q_0, q_1, \dots\}$ , so as to not get uncountably many machines by virtue of there being uncountably many possible states. With that, the argument that we can count the Finite State machines is straightforward since for any collection of finitely many states  $q_0, \dots, q_k$  there are finitely many transition tables. Thus the number of Finite State machines using those states is countable. And then the union of countably many countable sets is countable, so the number of machines over all collections of finitely many states is countable.
- (c) Because  $\Sigma$  has at least two members, the usual diagonalization construction shows that the power set of  $\Sigma$  is not countable.

**IV.2.23** The main point is that entries in the table are sets of states. This is the machine in Example 2.7.

$\Delta$	$a$	$b$
$q_0$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$q_1$	$\{q_3\}$	$\{\}$
$q_2$	$\{\}$	$\{q_3\}$
+ $q_3$	$\{q_3\}$	$\{q_3\}$

This is Example 2.8's machine.

$\Delta$	$a$	$b$	$c$
+ $q_0$	$\{q_1\}$	$\{\}$	$\{\}$
$q_1$	$\{\}$	$\{\}$	$\{q_0\}$

**IV.2.24**

- (A) Yes. From  $q_0$  a  $\epsilon$  transition goes to  $q_2$ , which is an accepting state.
- (B) No. From  $q_0$  the machine can either read the  $\emptyset$  and go to  $q_1$ , or take the  $\epsilon$  transition to  $q_2$  and then read the  $\emptyset$ , which sends the machine to no-state. Neither  $q_2$  nor no-state is an accepting state. Said more formally, the  $\epsilon$  closure of  $q_0$  is  $\{q_0, q_2\}$ . Neither state, on reading the character  $\emptyset$ , transitions to a state whose  $\epsilon$  closure includes an accepting state.
- (c) Yes. With that input the machine can go from  $q_0$  to  $q_1$ , then to  $q_2$ , and then around the loop to  $q_2$  again. More formally, this is a sequence of allowed transitions that ends in an accepting state.

$$\langle q_0, \emptyset 11 \rangle \vdash \langle q_1, 11 \rangle \vdash \langle q_2, 1 \rangle \vdash \langle q_2, \epsilon \rangle$$

(D) No. Starting at  $q_0$  with the character  $\emptyset$  first on the tape the machine can do two things. If it takes the  $\epsilon$  transition then when it reads the  $\emptyset$  it goes to no-state, which is not an accepting state.

Otherwise, the  $\emptyset$  moves the machine to  $q_1$ . Next, on the following tape character 1, the machine moves to  $\{q_2\}$ . Finally, on the third character  $\emptyset$ , the machine moves to no-state.

(E) There are two:  $\emptyset 1111$  and  $11111$ .

**IV.2.25** In the first place, someone in this class saying something is “just mathematical abstraction” is like someone objecting in an electrodynamics class that this is all “just electricity.” That’s exactly what we are doing here.

Said another way, if someone compares what we are thinking about here with a course on data structures, and holds the latter up as a model of “real” then that is a stretch. “Real” is changing diapers or grafting fruit trees. All of what we are doing, in the entire programs of computer science and mathematics, is abstraction of one kind of another.

If that person responds, “OK, but abstraction can go too far, and get away from what brought us into the program” then that is more measured. One answer is that people in a program need to take as the default assumption that the people who put together this program, and such programs across the world, put in steps that lead somewhere. Nondeterminism is here because it is necessary to understand the material of the fifth chapter, which describes the most important problem in the field today, one that will directly impact all practitioners, the P versus NP question.

Besides, it is interesting, and fun. Those are respectable criteria for at least some consideration.

**IV.2.26** The machine accepts the input string if there is a sequence of legal transitions that allows the machine to process the string and that ends in an accepting state. It does not say there must not be any wrong ways to go, only that there must be a right way.

For instance, the given machine can accept  $aab$  by starting in  $q_0$ , transitioning to  $q_1$  on the first  $a$ , taking the  $\epsilon$  transition to  $q_0$ , then going to  $q_1$  on the second  $a$ , then going to  $q_2$ , which is accepting, with the  $b$ .

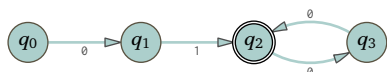
$$\langle q_0, aab \rangle \vdash \langle q_1, ab \rangle \vdash \langle q_0, ab \rangle \vdash \langle q_1, b \rangle \vdash \langle q_2, \epsilon \rangle$$

This machine recognizes the language  $\mathcal{L} = \{a^n b \mid n \in \mathbb{N}\}$ .

**IV.2.27** This is the graph of the nondeterministic machine. (It is nondeterministic in that the edges don’t list every element of the alphabet  $\{-, \emptyset, \dots, 9\}$ .)



**IV.2.28**



**IV.2.29**

(A) Here are the  $\epsilon$  closures.

state	$q_0$	$q_1$	$q_2$
$\epsilon$ closure	$\{q_0\}$	$\{q_1, q_2\}$	$\{q_2\}$

(B) It does not accept the empty string because the  $\epsilon$  closure of  $q_0$  does not contain any final states.

(C) It accepts both of the one-character strings  $a$  and  $b$ . For instance, on the string  $a$  the machine can transition from  $q_0$  to  $q_1$ , and then it can make an  $\epsilon$  transition to  $q_2$ , which is an accepting state.

(D) The given machine can accept  $aab$  by starting in  $q_0$ , transitioning to  $q_0$  on the first  $a$ , then going to  $q_1$  on the second  $a$ , taking the  $\epsilon$  transition to  $q_2$ , then going to  $q_2$  with the  $b$ , thereby ending in an accepting state.

$$\langle q_0, aab \rangle \vdash \langle q_0, ab \rangle \vdash \langle q_1, b \rangle \vdash \langle q_2, b \rangle \vdash \langle q_2, \epsilon \rangle$$

(E) Five of the shortest strings are:  $a$ ,  $b$ ,  $aa$ ,  $ab$ , and  $aab$ .

(F) Five that it does not accept are: the empty string  $\epsilon$ ,  $ba$ ,  $baa$ ,  $bab$ , and  $baaa$ .

**IV.2.30**

$\Delta$	$\epsilon$	a	b
$q_0$	$\{q_1\}$	$\{q_1\}$	$\{\}$
$q_1$	$\{q_0\}$	$\{\}$	$\{q_2\}$
$+ q_2$	$\{\}$	$\{\}$	$\{\}$

**IV.2.31**

- (A)  $\langle q_0, 011 \rangle \vdash \langle q_2, 11 \rangle \vdash \langle q_2, 1 \rangle \vdash \langle q_2, \epsilon \rangle$
- (B)  $\langle q_0, 00011 \rangle \vdash \langle q_1, 0011 \rangle \vdash \langle q_1, 011 \rangle \vdash \langle q_1, 11 \rangle \vdash \langle q_2, 1 \rangle \vdash \langle q_2, \epsilon \rangle$
- (C) Yes, because the state  $q_0$  is accepting.
- (D) It accepts  $\emptyset$  because it can go through these steps that end in an accepting state.

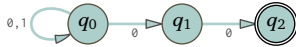
$$\langle q_0, \emptyset \rangle \vdash \langle q_2, \epsilon \rangle$$

It does not accept 1 because there is no such sequence of steps.

- (E) Five are  $\epsilon, \emptyset, 01, 011$ , and  $0111$ .
- (F) Here are five:  $1, 10, 11, 010$ , and  $110$ .
- (G) One description is  $\mathcal{L} = \{\sigma \in \mathbb{B}^* \mid \sigma = \epsilon \text{ or } \sigma = 0^i 1^j \text{ for } i \geq 1 \text{ and } j \geq 0\}$ .

**IV.2.32**

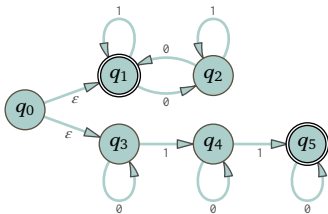
(A)



(B)



(C)

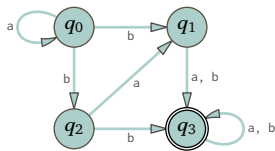


(D)



**IV.2.33**

(A)



- (B) A string is in the language if it consists of, first, some number of repetitions of: either a number of repetitions of a or ba. Then it has either bb, or a b followed by an a or b. Finally, the suffix consists of some number of characters, either a or b.
- (C) This is the transition function for the associated deterministic machine.

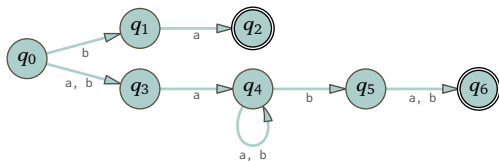
$\Delta_D$	a	b
$s_0 = \{ \}$	$s_0$	$s_0$
$s_1 = \{ q_0 \}$	$s_1$	$s_8$
$s_2 = \{ q_1 \}$	$s_4$	$s_4$
$+ s_3 = \{ q_2 \}$	$s_2$	$s_4$
$s_4 = \{ q_3 \}$	$s_4$	$s_4$
$s_5 = \{ q_0, q_1 \}$	$s_7$	$s_{14}$
$+ s_6 = \{ q_0, q_2 \}$	$s_5$	$s_{14}$
$s_7 = \{ q_0, q_3 \}$	$s_7$	$s_{14}$
$+ s_8 = \{ q_1, q_2 \}$	$s_9$	$s_4$
$s_9 = \{ q_1, q_3 \}$	$s_4$	$s_4$
$+ s_{10} = \{ q_2, q_3 \}$	$s_9$	$s_4$
$+ s_{11} = \{ q_0, q_1, q_2 \}$	$s_{12}$	$s_{14}$
$s_{12} = \{ q_0, q_1, q_3 \}$	$s_7$	$s_{14}$
$+ s_{13} = \{ q_0, q_2, q_3 \}$	$s_{12}$	$s_{14}$
$+ s_{14} = \{ q_1, q_2, q_3 \}$	$s_9$	$s_4$
$+ s_{15} = \{ q_0, q_1, q_2, q_3 \}$	$s_{12}$	$s_{14}$

IV.2.34

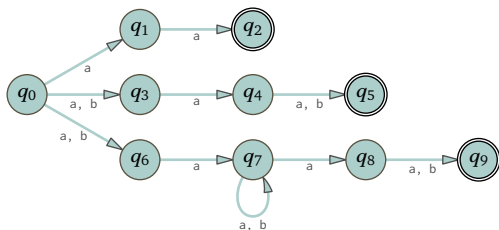


IV.2.35

(A)



(B)



IV.2.36 That machine has seven states.

state $q$	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$
$\epsilon$ closure $\hat{E}(q)$	$\{q_0, q_2, q_3, q_5\}$	$\{q_1\}$	$\{q_2, q_3, q_5\}$	$\{q_3\}$	$\{q_4\}$	$\{q_5\}$	$\{q_6\}$

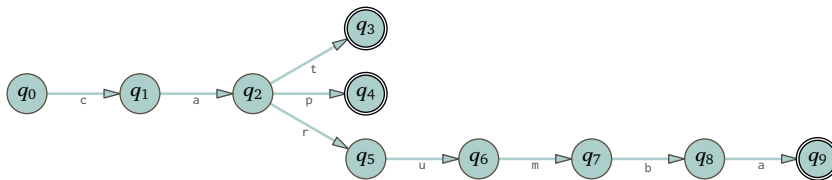
IV.2.37



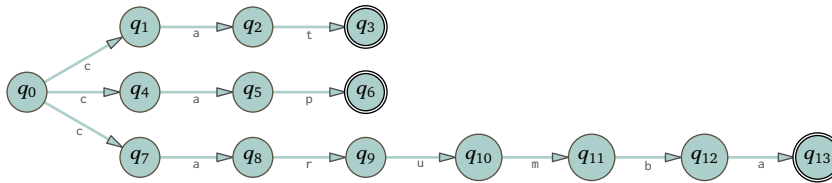
The associated deterministic machine has eight states. Its accepting states are the ones containing  $q_2$ .

$\Delta_D$	$\emptyset$	1
$s_0 = \{ \}$	$s_0$	$s_0$
$s_1 = \{ q_0 \}$	$s_0$	$s_2$
$s_2 = \{ q_1 \}$	$s_3$	$s_0$
$+ s_3 = \{ q_2 \}$	$s_3$	$s_3$
$s_4 = \{ q_0, q_1 \}$	$s_3$	$s_2$
$+ s_5 = \{ q_0, q_2 \}$	$s_3$	$s_6$
$+ s_6 = \{ q_1, q_2 \}$	$s_3$	$s_3$
$+ s_7 = \{ q_0, q_1, q_2 \}$	$s_3$	$s_6$

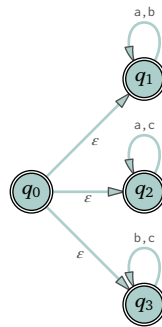
IV.2.38 There are a number of ways to go. This is one



and this is another.



IV.2.39

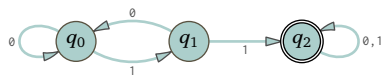


IV.2.40 It is  $\mathcal{L} = \{ \sigma \in \{ a, b \}^* \mid \sigma \text{ has no substring } bba \}$ .

IV.2.41 Here is the nondeterministic machine



and this is a deterministic version.



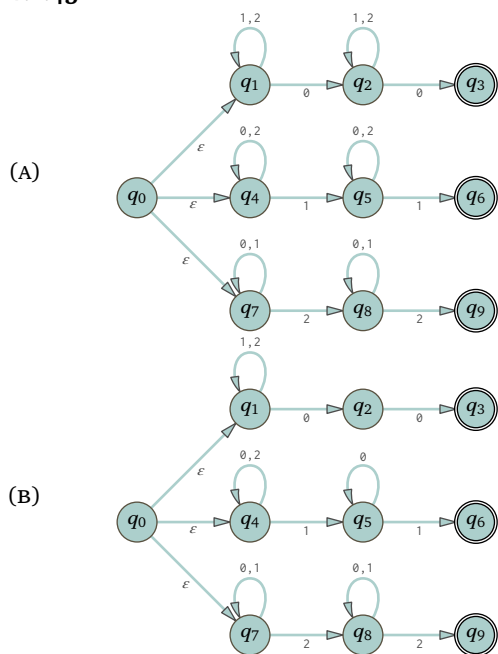
IV.2.42 This is for the machine on the left.

$\Delta_D$	$\emptyset$	1
$s_0 = \{ \}$	$s_0$	$s_0$
$s_1 = \{ q_0 \}$	$s_4$	$s_0$
$s_2 = \{ q_1 \}$	$s_3$	$s_3$
$+ s_3 = \{ q_2 \}$	$s_0$	$s_1$
$s_4 = \{ q_0, q_1 \}$	$s_7$	$s_3$
$+ s_5 = \{ q_0, q_2 \}$	$s_4$	$s_1$
$+ s_6 = \{ q_1, q_2 \}$	$s_3$	$s_5$
$+ s_7 = \{ q_0, q_1, q_2 \}$	$s_7$	$s_5$

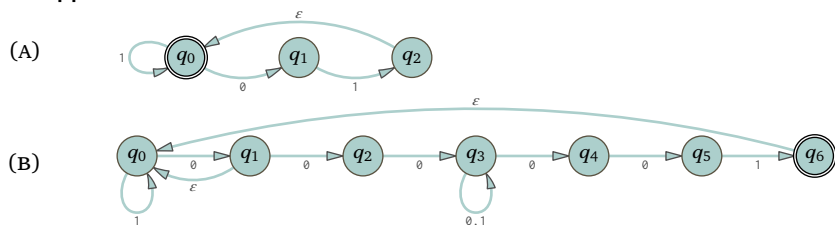
And, this is for the machine on the right.

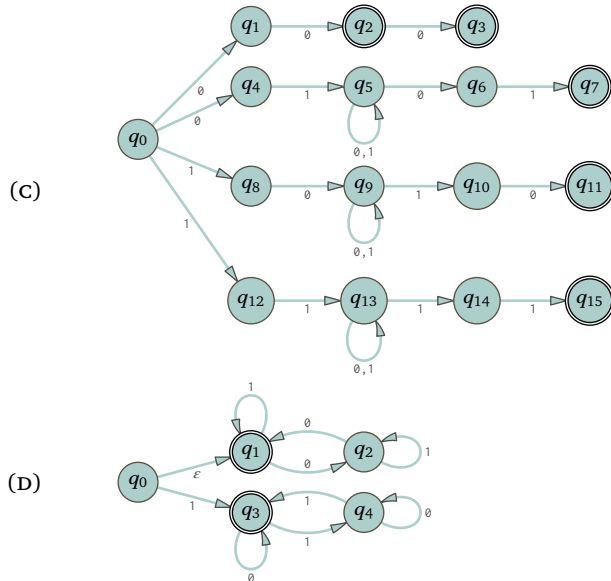
$\Delta_D$	$\emptyset$	1
$s_0 = \{ \}$	$s_0$	$s_0$
$s_1 = \{ q_0 \}$	$s_4$	$s_1$
$s_2 = \{ q_1 \}$	$s_3$	$s_3$
$+ s_3 = \{ q_2 \}$	$s_0$	$s_5$
$s_4 = \{ q_0, q_1 \}$	$s_7$	$s_1$
$+ s_5 = \{ q_0, q_2 \}$	$s_4$	$s_1$
$+ s_6 = \{ q_1, q_2 \}$	$s_3$	$s_5$
$+ s_7 = \{ q_0, q_1, q_2 \}$	$s_7$	$s_5$

IV.2.43



IV.2.44





IV.2.45 Here are the graphs.



The machine on the left, the one that accepts only the empty string, has this table.

$\Delta$	a	b	c
+ q0	{ }	{ }	{ }

The machine on the right has this.

$\Delta$	a	b	c
q0	{ q1 }	{ q1 }	{ q1 }
+ q1	{ q1 }	{ q1 }	{ q1 }

IV.2.46

(A) This is a derivation of abb.

$$S \Rightarrow aA \Rightarrow abB \Rightarrow abb$$

This is a derivation of aabb.

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabB \Rightarrow aabb$$

And, this is a derivation of abbb.

$$S \Rightarrow aA \Rightarrow abB \Rightarrow abbB \Rightarrow abbb$$

Verifying that they are accepted by the machine is routine.

(B) The language of the machine and the grammar is  $\mathcal{L} = \{ a^i b^j \mid i > 0, j > 1 \}$ .

IV.2.47 Each of these is solvable. For each we will sketch an algorithm.

(A) Clearly we can write a simulator for deterministic Finite State machines. By Church's Thesis since we can write it on a modern computer, we can write one for a Turing machine. When given an input  $\sigma$ , the machine will consume it in no more than  $|\sigma|$  steps, and so the simulation is sure to halt. Then just check whether the ending state is an accepting state.

(B) The section body discusses how to simulate a nondeterministic computation, by dovetailing (that is, time-slicing). With that we can answer questions about whether this machine accepts a given input  $\sigma$  just as in the prior item.

IV.2.48

(A) Here is the list.

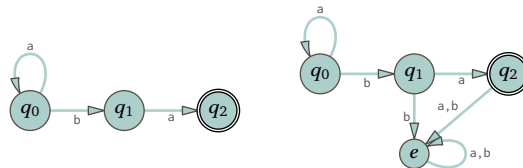


	a*b	a*	$\emptyset$	$\epsilon$	b(a b)a	(a b)( $\epsilon$  a)a
$\epsilon$	F	T	F	T	F	F
a	F	T	F	F	F	F
b	T	F	F	F	F	F
aa	F	T	F	F	F	T
ab	T	F	F	F	F	F
ba	F	F	F	F	F	T
bb	F	F	F	F	F	F
aaa	F	T	F	F	F	T
aab	T	F	F	F	F	F
aba	F	F	F	F	F	F
abb	F	F	F	F	F	F
baa	F	F	F	F	T	T
bab	F	F	F	F	F	F
bba	F	F	F	F	T	F
bbb	F	F	F	F	F	F

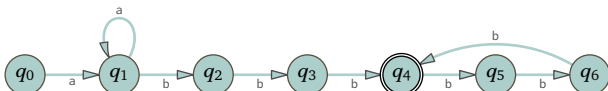
FIGURE 75, FOR QUESTION IV.3.18: String matching test.

	0*1	1*0	$\emptyset$	$\epsilon$	0(0 1)*	(100)( $\epsilon$  1)0*
$\epsilon$	F	F	F	T	F	F
0	F	T	F	F	T	F
1	T	F	F	F	F	F
00	F	F	F	F	T	F
01	T	F	F	F	T	F
10	F	T	F	F	F	F
11	F	F	F	F	F	F
000	F	F	F	F	T	F
001	T	F	F	F	T	F
010	F	F	F	F	T	F
011	F	F	F	F	T	F
100	F	F	F	F	F	T
101	F	F	F	F	F	F
110	F	T	F	F	F	F
111	F	F	F	F	F	F

FIGURE 76, FOR QUESTION IV.3.19: String matching test.

FIGURE 77, FOR QUESTION IV.3.32: Finite State machine for  $a^*b^*$ .

- (B) Five members of  $\mathcal{L}_1$  are:  $abbb$ ,  $aabbb$ ,  $ab^6$ ,  $aab^6$ ,  $ab^9$ . Five that are not are:  $\epsilon$ ,  $a$ ,  $b$ ,  $abbb$ ,  $abbbba$ ,  $ba$ . A regular expression is  $aa^*bbb(bbb)^*$ . This is a nondeterministic Finite State machine that accepts  $\mathcal{L}_1$ .



IV.3.24  $((a|b)^*)|((a|c)^*)|((b|c)^*)$

IV.3.25

- (A)  $b(a|b)^*$   
 (B)  $(a|b)^*a(a|b)$   
 (C) The two characters could come in the order  $\dots a \dots b \dots$  or the order  $\dots a \dots b \dots$ . So this works:  
 $(a|b)^*((a(a|b)^*b)|(b(a|b)^*a))(a|b)^*$ .  
 (D) The  $a$ 's must come in triplets:  $(b^*ab^*ab^*ab^*)^*b^*$ .

IV.3.26

- (A)  $aba(a|b)^*$   
 (B)  $(a|b)^*aba$   
 (C)  $(a|b)^*aba(a|b)^*$

IV.3.27

- (A) There must be at least one 1, and more 1's must come in pairs:  $(0^*10^*1)^*0^*10^*$ .  
 (B) The regular expression  $(\epsilon|1)(01)^*(\epsilon|0)$  will do. Note that it matches the empty string, as well as the string  $0$  and the string  $1$ .  
 (C) A multiple of eight in binary ends in three  $0$ 's:  $(0|1)^*000$ . That expression matches binary numbers with leading  $0$ 's and won't match  $0$ . If you don't like that then instead use  $(1(0|1)^*000)|0$ .

IV.3.28

- (A) This is an answer:  $((ba)^*bb^*)^*$ . Notice that it matches the empty string, and that it matches  $babab$ .  
 (B)  $(a|b)^*((bba^*bb)|(bbb))(a|b)^*$

IV.3.29

- (A) Here the  $0$ 's come in pairs:  $(1^*01^*01^*)^*1^*$ .  
 (B) Here two of the  $1$ 's have no Kleene star's:  $0^*10^*10^*1^*(0|1)^*$ .  
 (C) One that will do is to start with the expression from the prior item and replace all the  $0$ 's with the expression from the first item.

$$((1^*01^*01^*)^*1^*)^*1((1^*01^*01^*)^*1^*)^*1((1^*01^*01^*)^*1^*)^*1^*((1^*01^*01^*)^*1^*)^*|1)^*$$

IV.3.30

- (A)  $(a(a|b)^*a)|(b(a|b)^*b)$   
 (B)  $((aaa)^*b(aaa)^*)|(a(aaa)^*ba(aaa)^*)|(aa(aaa)^*baa(aaa)^*)$

IV.3.31

- (A)  $1(0|1)^*000$   
 (B)  $1(01)^*0|0(10)^*1$   
 (C)  $\epsilon|0|1|0(10^*)(1|\epsilon)|1(01)^*(0|\epsilon)$

IV.3.32

- (A) A nondeterministic Finite State machine is on the left in Figure 77 on page 121. If you prefer a deterministic machine, use the one on the right.

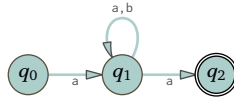


FIGURE 78, FOR QUESTION IV.3.32: Finite State machine for  $ab^*(a|b)^*a$ .

(B) A nondeterministic machine is in Figure 78 on page 122. Note that  $b^*(a|b)^*$  collapses to  $(a|b)^*$ , so that is what the machine handles. We have given a way to convert a nondeterministic Finite State machine to a deterministic one, so if you prefer a deterministic one then take the above nondeterministic machine as an abbreviation.

**IV.3.33**

- (A) Transform the machine  $\mathcal{M}$  to a new machine  $\hat{\mathcal{M}}$  by making the set of final states of  $\hat{\mathcal{M}}$  be  $S$ .
- (B) Transform the machine  $\mathcal{M}$  to  $\hat{\mathcal{M}}$  by making the start state be the given first single state, and also make the set  $S$  of the prior item contain only the ending state. Then apply the argument given for the prior item.

**IV.3.34** We will construct the reachable states, and then the unreachable ones are the others.

- (A) The set of reachable states is  $S_2 = S_3 = \dots = \{q_0, q_1, q_2\}$ . So the set of unreachable states is  $\{q_3\}$ .

$i$	$S_i$
0	$\{q_0\}$
1	$\{q_0, q_1\}$
2	$\{q_0, q_1, q_2\}$
3	$\{q_0, q_1, q_2\}$

- (B) The set of reachable states is  $S_2 = S_3 = \dots = \{q_0, q_1, q_3\}$ . Thus the set of unreachable states is  $\{q_2, q_4\}$ .

$i$	$S_i$
0	$\{q_0\}$
1	$\{q_0, q_1\}$
2	$\{q_0, q_1, q_3\}$
3	$\{q_0, q_1, q_3\}$

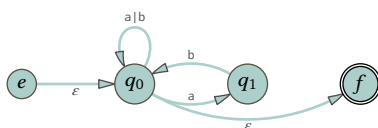
**IV.3.35** Alphabets are finite, as defined in Appendix A. So the set of length  $n$  strings over  $\Sigma$  is finite, and therefore countable. Chapter II's Corollary 2.12 shows that the countable union of countable sets is countable, and so the union over all  $n$  of length  $n$  strings is countable.

**IV.3.36** Figure 79 on page 123 shows the two trees.

**IV.3.37** Figure 80 on page 123 shows the two trees.

**IV.3.38**

- (A) This is  $\hat{\mathcal{M}}$ .



- (B) This is the before half of the diagram from Lemma 3.13's proof, with  $\hat{\mathcal{M}}$  repeated for convenience.

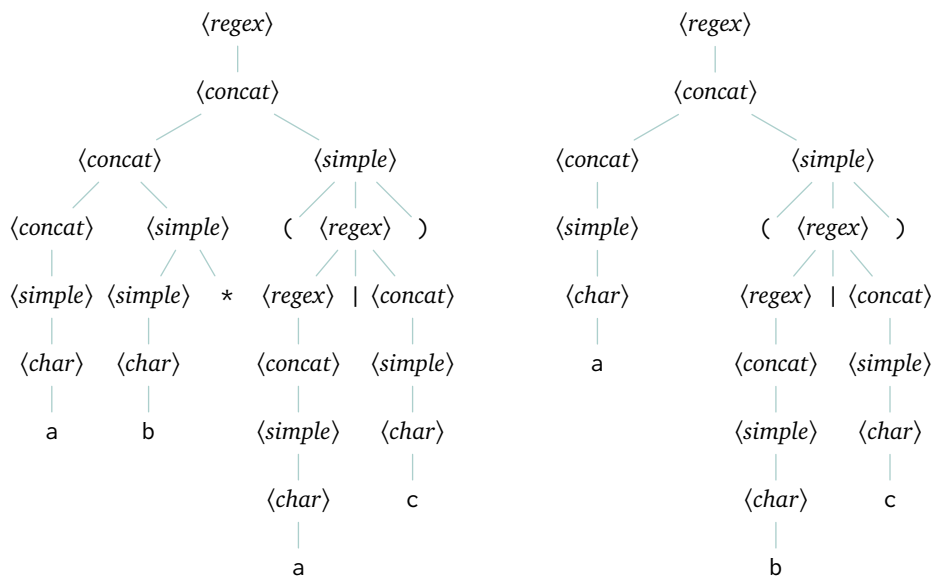


FIGURE 79, FOR QUESTION IV.3.36: Parse trees for  $a(b|c)$  and  $ab^*(a|c)$ .

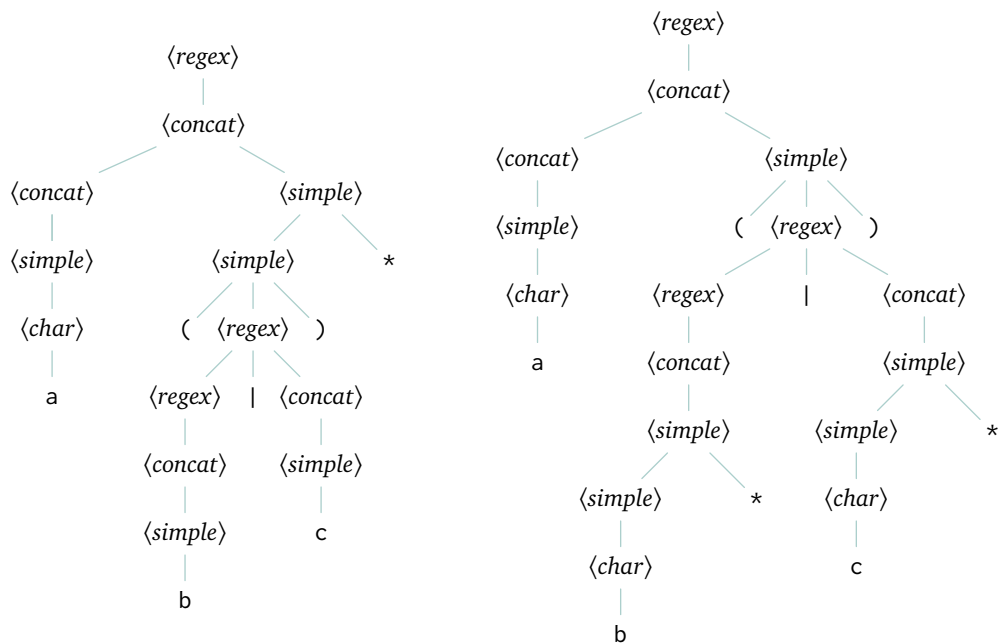
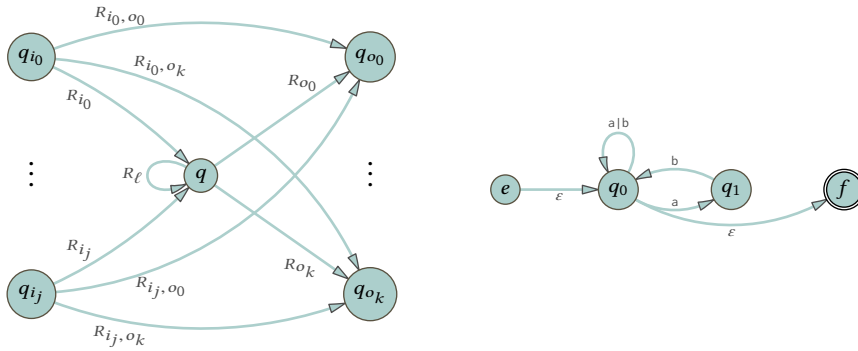


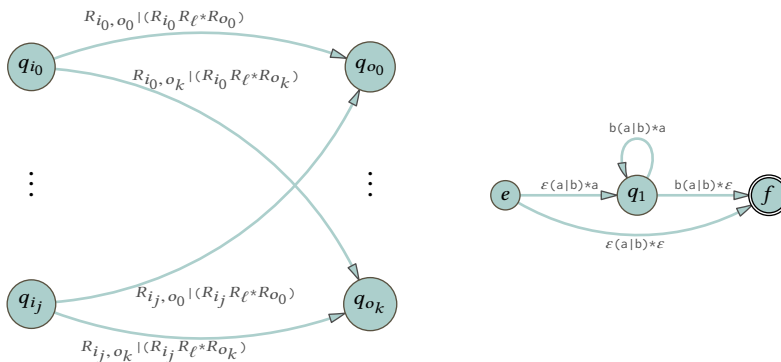
FIGURE 80, FOR QUESTION IV.3.37: Parse trees for  $a(b|c)^*$  and  $a(b^*|c)^*$ .



The state  $q_0$  has two states that originate arrows into it, and two that receive arrows from it. So we get this translation of the diagram's notation.

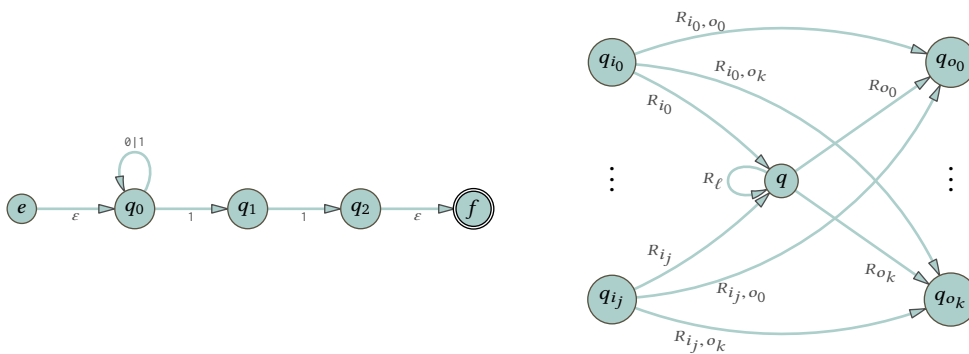
$q$	$q_{i_0}$	$q_{i_1}$	$q_{o_0}$	$q_{o_1}$	$R_{i_0}$	$R_{i_1}$	$R_{i_0, o_0}$	$R_{i_0, o_1}$	$R_{i_1, o_0}$	$R_{i_1, o_1}$	$R_{\ell}$	$R_{o_0}$	$R_{o_1}$
$q_0$	$e$	$q_1$	$q_1$	$f$	$\epsilon$	$b$	-	-	-	-	$a b$	$a$	$\epsilon$

(c) This is the after half diagram of Lemma 3.13's proof, along with the machine  $\hat{\mathcal{M}}$  after  $q_0$  is eliminated.



**IV.3.39**

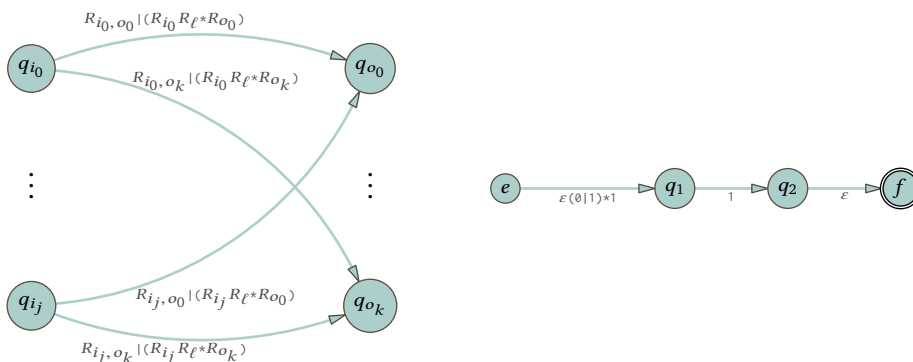
(A) This is  $\hat{\mathcal{M}}$  along with the before half of the diagram from Lemma 3.13's proof.



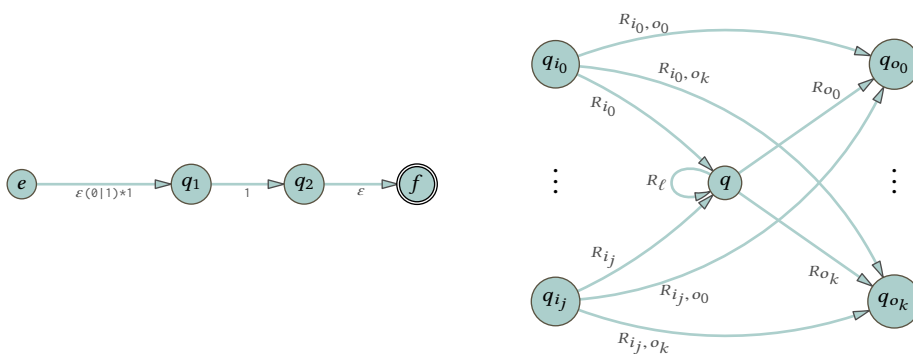
The state  $q_0$  has one state that originates arrows into it, and one that receives arrows from it. So we get this translation of the diagram's notation.

$q$	$q_{i_0}$	$q_{o_0}$	$R_{i_0}$	$R_{i_0, o_0}$	$R_{\ell}$	$R_{o_0}$
$q_0$	$e$	$q_1$	$\epsilon$	-	$\emptyset 1$	$1$

This is the after half diagram of Lemma 3.13's proof, along with the machine  $\hat{\mathcal{M}}$  after  $q_0$  is eliminated.



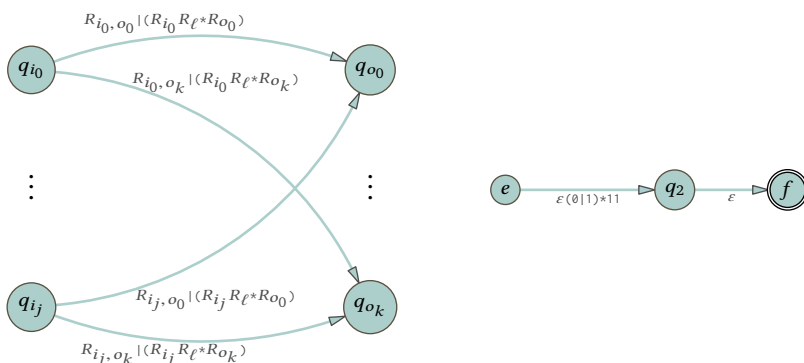
(B) This is the machine from the prior answer, along with the before half of the diagram from Lemma 3.13's proof.



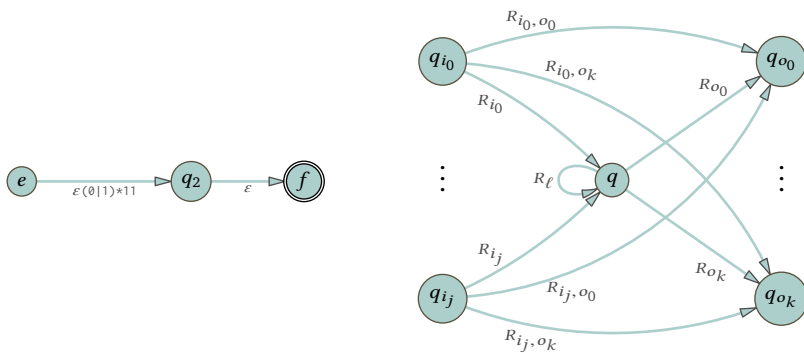
The state  $q_1$  has one state that originates arrows into it, and one that receives arrows from it. So we get this translation of the diagram's notation.

$$\begin{array}{c|ccc} q & q_{i_0} & q_{o_0} \\ \hline q_1 & e & q_2 \end{array} \quad \begin{array}{c|cccc} R_{i_0} & R_{i_0, o_0} & R_\ell & R_{o_0} \\ \hline \epsilon(\emptyset|1)^*1 & - & - & 1 \end{array}$$

This is the after half diagram of Lemma 3.13's proof, along with the machine  $\hat{M}$  after  $q_1$  is eliminated.



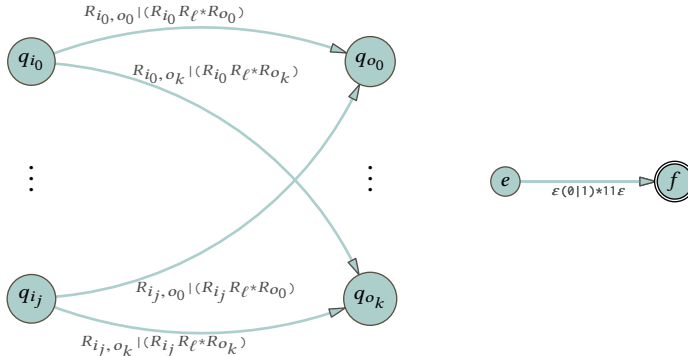
(C) This is the machine from the prior answer, along with the before half diagram.



The state  $q_2$  has one state that originates arrows into it, and one that receives arrows from it. So we get this translation of the diagram's notation.

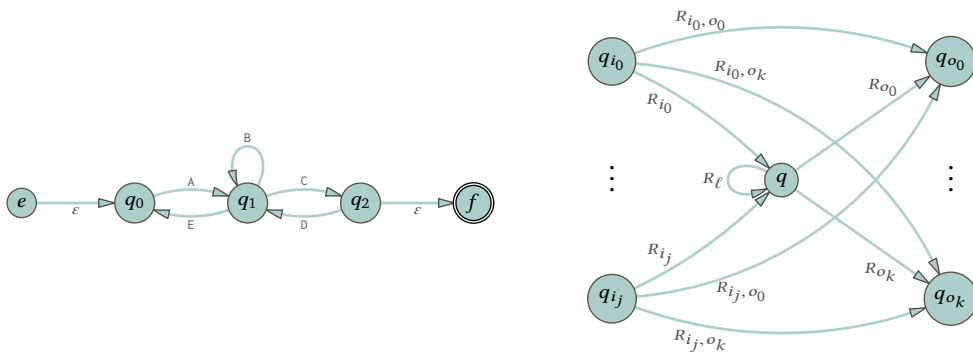
$$\frac{q \quad q_{i_0} \quad q_{o_0}}{q_2 \quad e \quad f} \quad \frac{R_{i_0} \quad R_{i_0, o_0} \quad R_\ell \quad R_{o_0}}{\varepsilon(\emptyset|1)^*11} \quad \frac{}{-} \quad \frac{}{-} \quad \frac{}{\varepsilon}$$

This is the after half diagram along with the machine after  $q_2$  is gone.



(D) The regular expression is  $\varepsilon(\emptyset|1)^*11\varepsilon$ .

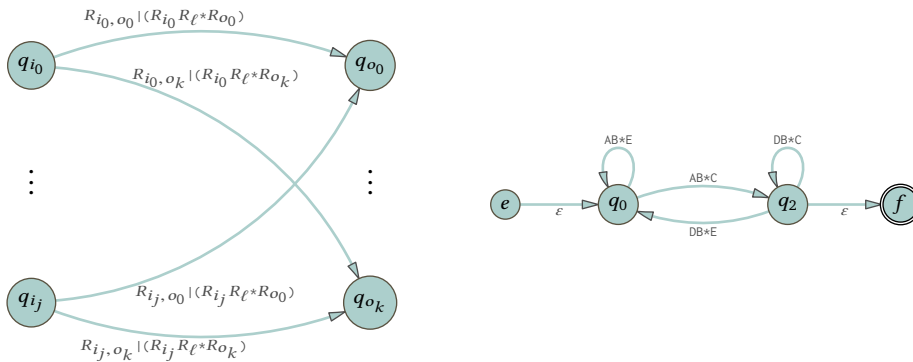
IV.3.40 This is  $\hat{\mathcal{M}}$  along with the before half of the diagram from Lemma 3.13's proof.



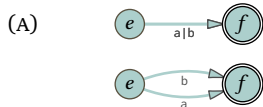
The state  $q_1$  has two states that originate arrows into it, and two that receive arrows from it. So we get this translation of the diagram's notation.

$$\frac{q \quad q_{i_0} \quad q_{i_1} \quad q_{o_0} \quad q_{o_1}}{q_1 \quad q_0 \quad q_2 \quad q_0 \quad q_2} \quad \frac{R_{i_0} \quad R_{i_1} \quad R_{i_0, o_0} \quad R_{i_0, o_1} \quad R_{i_1, o_0} \quad R_{i_1, o_1} \quad R_\ell \quad R_{o_0} \quad R_{o_1}}{A \quad D \quad - \quad - \quad - \quad - \quad B \quad E \quad C}$$

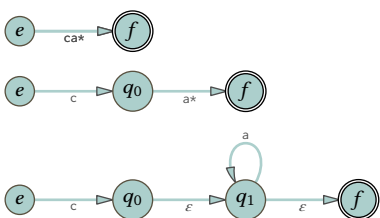
Here is the after diagram, along with the machine  $\hat{\mathcal{M}}$  after the elimination of  $q_1$ .



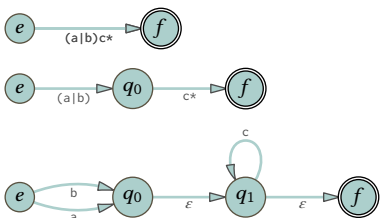
IV.3.41 Each of these gives the progression of steps.



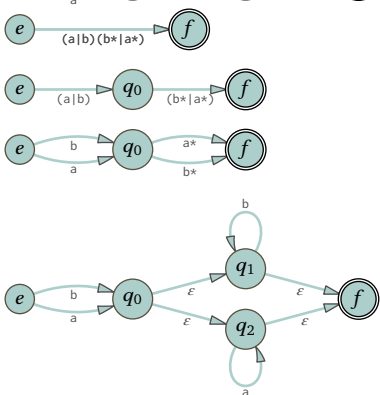
(B) First concatenation and then Kleene star.



(C)



(D)



IV.4.7

- (A) False. For example, the language  $\{a^n \mid n \in \mathbb{N}\} = \{\epsilon, a, aa, \dots\}$  is regular and infinite.
- (B) False, the empty language is recognized by any Finite State machine that has no accepting states. It is also the language described by the regular expression  $\emptyset$ .
- (C) False. Rather, the intersection of two regular languages is regular. For example, where  $\mathcal{L}$  is not regular, the intersection  $\mathcal{L} \cap \mathbb{B}^* = \mathcal{L}$  is not regular.
- (D) False. The language  $\mathbb{B}^*$  is the set of all strings is recognized by any Finite State machine whose states are all accepting states. It is also the language described by the regular expression  $(\emptyset|1)^*$ .
- (E) False, a Finite State machine with no accepting states does not accept any strings.
- (F) False, a minimal machine cannot be further reduced. For instance, a machine with only one state, and that state accepting, is the smallest machine that recognizes  $\mathcal{L} = \mathbb{B}^*$ .

IV.4.8 The first is true and the second is false.

- (A) If the language consists of the strings  $\sigma_0, \dots, \sigma_{n-1}$  then a regular expression is  $\sigma_0 | \dots | \sigma_{n-1}$ .
- (B) The language over  $\Sigma = \{a, b\}$  described by the regular expression  $a^*$  is  $\{\epsilon, a, aa, \dots\}$ , and is infinite.

IV.4.9 Yes. To show this we need to show that this is the language of some Finite State machine. In binary, powers of 2 are represented by strings of bits starting with 1 and ending in 0's. So the regular expression  $10^*$  describes the language. (If you want to allow leading 0's then the regular expression is  $0^*10^*$ .)

IV.4.10 This is not really a sensible question. If we assert that no English sentence has more than a million words (see (Wikipedia contributors 2020)) then we might claim that English is a regular language because we can list the sentences that we declare allowed, and there are finitely many of them.

Otherwise we must choose a grammar. But which one? Is "Colorless green ideas sleep furiously" a legal sentence even though it is not sensible? Is "My god, it is her" allowed even though some would say it should be "My god, it is she"?

Really, it is not a sensible question. (See <https://cs.stackexchange.com/a/116178/67754>.)

IV.4.11 One way to show this is to give a regular expression for each.

- (A)  $a(a|b)^*a$
- (B)  $b^*(ab^*a)^*b^*$

**IV.4.12** Each is false. The same example suffices for all three. For  $\mathcal{L}_0$  use the set of all bitstrings,  $\mathbb{B}^*$ . For  $\mathcal{L}_1$  use any non-regular subset of it, the existence of which is guaranteed by Lemma 4.2.

**IV.4.13** Because the language is regular it is described by a regular expression,  $R$ . Then the regular expression  $1R$  describes  $\hat{\mathcal{L}}$ .

**IV.4.14** The size of the set  $Q_0 \times Q_1$  is  $n_0 \cdot n_1$ .

**IV.4.15** These are the tables for the machines.

$\Delta_0$	a	b	$\Delta_1$	a	b
+ $q_0$	$q_0$	$q_1$	+ $s_0$	$s_1$	$s_0$
+ $q_1$	$q_2$	$q_1$	+ $s_1$	$s_0$	$s_0$
	$q_2$	$q_0$			

Here is the cross product machine, without specifying any accepting states.

$\Delta$	a	b
$(q_0, s_0)$	$(q_0, s_1)$	$(q_1, s_0)$
$(q_0, s_1)$	$(q_0, s_0)$	$(q_1, s_0)$
$(q_1, s_0)$	$(q_2, s_1)$	$(q_1, s_0)$
$(q_1, s_1)$	$(q_2, s_0)$	$(q_1, s_0)$
$(q_2, s_0)$	$(q_2, s_1)$	$(q_0, s_0)$
$(q_2, s_1)$	$(q_2, s_0)$	$(q_0, s_0)$

- (A) To have this machine accept the intersection of the two languages, take the accepting states to be those in  $F = \{(q_0, s_1), (q_1, s_1)\}$ .
- (B)  $F = \{(q_0, s_0), (q_0, s_1), (q_1, s_0), (q_1, s_1)\}$

**IV.4.16** Here is the transition table for  $\mathcal{M}_1$ .

$\Delta_1$	a	b
$s_0$	$s_0$	$s_1$
+ $s_1$	$s_1$	$s_0$

The product of this machine with itself has four states.

$\Delta_1$	a	b
$(s_0, s_0)$	$(s_0, s_0)$	$(s_1, s_1)$
$(s_0, s_1)$	$(s_0, s_1)$	$(s_1, s_0)$
$(s_1, s_0)$	$(s_1, s_0)$	$(s_0, s_1)$
+ $(s_1, s_1)$	$(s_1, s_1)$	$(s_0, s_0)$

Here the accepting state is set for the intersection, to only  $(s_1, s_1)$ . We could include the states  $(s_0, s_1)$  and  $(s_1, s_0)$  for the union construction, but it wouldn't matter because those are unreachable states.

**IV.4.17** The machine on the left below accepts a string  $\sigma \in \mathbb{B}^*$  if it contains at least two 0's. The one on the right accepts a string if it contains an even number of 1's.

$\Delta_0$	0	1	$\Delta_1$	0	1
$q_0$	$q_1$	$q_0$	+ $s_0$	$s_0$	$s_1$
$q_1$	$q_2$	$q_1$	+ $s_1$	$s_1$	$s_0$
+ $q_2$	$q_2$	$q_2$			

The product construction gives this. We want the intersection of the two languages.

	$\Delta$	$0$	$1$
$(q_0, s_0)$	$(q_1, s_0)$	$(q_0, s_1)$	
$(q_0, s_1)$	$(q_1, s_1)$	$(q_0, s_0)$	
$(q_1, s_0)$	$(q_2, s_0)$	$(q_1, s_1)$	
$(q_1, s_1)$	$(q_2, s_1)$	$(q_1, s_0)$	
+ $(q_2, s_0)$	$(q_2, s_0)$	$(q_2, s_1)$	
$(q_2, s_1)$	$(q_2, s_1)$	$(q_2, s_0)$	

**IV.4.18**

- (A) True. Every language is a subset of the language  $\Sigma^*$ .  
 (B) False. Take  $\mathcal{L}_0$  to be not regular, and take  $\mathcal{L}_1$  to be the empty language, which is regular. Their union is  $\mathcal{L}_0$ .  
 (C) False. A language with one element is regular, and its only subset is the empty language, which is also regular.  
 (D) True. This is part of Lemma 4.3.

**IV.4.19** It might be regular, or might be not regular. First note that the language of all strings  $\mathcal{L} = \Sigma^*$  is regular. Now assume that  $\mathcal{L}_1$  is not regular. The concatenation  $\Sigma^* \hat{\ } \mathcal{L}_1 = \Sigma^*$  gives a language that is regular. Also, where the alphabet is  $\Sigma$  and  $s \in \Sigma$  then the concatenation  $\{s\} \hat{\ } \mathcal{L}_1$  gives a language that is not regular.

**IV.4.20** By Lemma 4.3, regular languages are closed under Kleene star and concatenation.

**IV.4.21** The statement is false. There are three premisses, that all finite languages are regular, that there are countably many finite languages, and that there are countably many regular sets. All three are true. However, the conclusion that all regular languages are finite does not follow. In addition, it is false: the language  $\mathcal{L} \subseteq \mathbb{B}^*$  described by the regular expression  $0^*$  is regular and infinite.

**IV.4.22** Let  $\Sigma$  be the alphabet of the language  $\mathcal{L}$ . The collection of even length strings over  $\Sigma$  is regular since there is clearly a Finite State machine that accepts only those strings. The intersection of  $\mathcal{L}$  with this collection is an intersection of two regular languages, so it is regular.

**IV.4.23** Let  $\{a^n b^n \in \{a, b\}^* \mid n \in \mathbb{N}\}$  be  $\mathcal{L}_0$ . Let the language described by the regular expression  $a^* b^*$  be  $\mathcal{L}_1$ , and let the language  $\{\sigma \in \{a, b\}^* \mid \sigma \text{ contains the same number of } a\text{'s as } b\text{'s}\}$  be  $\mathcal{L}_2$ . If  $\mathcal{L}_2$  were regular then because the intersection of regular languages is a regular language,  $\mathcal{L}_2 \cap \mathcal{L}_1$  would be regular. But that set is  $\mathcal{L}_0$ , which was given as not regular.

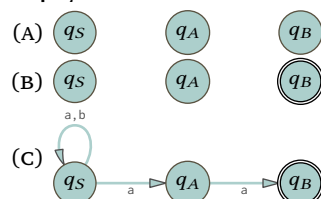
**IV.4.24** The graphical representation of a Finite State machine is a directed graph. So we can naturally extend the definition of one vertex being reachable from another to one state  $q$  being reachable from another state  $\hat{q}$ , if there is a sequence of directed edges that go from the first node to the second.

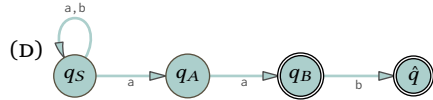
- (A) Fix a Finite State machine  $\mathcal{M}$  that recognizes  $\mathcal{L}$ . From it derive a new Finite State machine by making an accepting state every state  $q$  from which an accepting state of  $\mathcal{M}$  is reachable. Clearly that machine recognizes  $\text{pref}(\mathcal{L})$ .  
 (B) Again, fix a Finite State machine  $\mathcal{M}$  that recognizes  $\mathcal{L}$ . Let  $S$  be the set of states reachable from  $q_0$ . Get  $\hat{\mathcal{M}}$  by introducing a new state  $r$  and defining an  $\varepsilon$  transition to each state in  $S$ . Here also, clearly this machine recognizes  $\text{suff}(\mathcal{L})$ .  
 (C) We have  $\text{allpref}(\mathcal{L}) = (\mathcal{L}^c \hat{\ } \Sigma^*)^c$ , so it is regular.

**IV.4.25** If there was a Finite State machine  $\mathcal{M}$  that accepted this language then to solve the Halting problem, write  $\mathcal{M}$  as a subroutine. Then, given  $e \in \mathbb{N}$ , feed  $1^e$  to the subroutine. If it accepts, then  $e \in K$  and otherwise  $e \notin K$ .

**IV.4.26** Over  $\mathbb{B}$  the language of all strings  $\mathcal{L} = \mathbb{B}^*$  is regular, and is uncountable.

**IV.4.27** The result is a nondeterministic Finite State machine.





**IV.4.28**

- (A) Lemma 4.3 shows that the collection of regular languages is closed under union.
- (B) This identity shows that closure under complement will also demonstrate closure under set difference.
- (C) Fix a regular language  $\mathcal{L}$  over some alphabet  $\Sigma$ . It is recognized by some deterministic Finite State machine  $\mathcal{M}$  with input alphabet  $\Sigma$ .

Define a new machine  $\hat{\mathcal{M}}$  with the same states and transition function as  $\mathcal{M}$  but whose accepting states are the complement,  $F_{\hat{\mathcal{M}}} = Q_{\mathcal{M}} - F_{\mathcal{M}}$ . We will show that the language of this machine is  $\Sigma^* - \mathcal{L}$ .

Because  $\mathcal{M}$  is deterministic, each input string  $\tau$  is associated with a unique last state, the state that the machine is in after it consumes  $\tau$ 's last character, namely  $\hat{\Delta}(\tau)$ . Observe that  $\hat{\Delta}(\tau) \in F_{\hat{\mathcal{M}}}$  if and only if  $\hat{\Delta}(\tau) \notin F_{\mathcal{M}}$ . Thus the language of  $\hat{\mathcal{M}}$  is the complement of the language of  $\mathcal{M}$  and since this language is recognized by a Finite State machine, it too is regular.

**IV.4.29** For one direction suppose that the machine accepts at least one string,  $\sigma$ , of length  $k$  where  $n \leq k < 2n$ . In processing  $\sigma$  the machine must go through at least  $n$ -many transitions. But the machine has  $n$  states and starting at  $q_0$  and visiting all the other states exactly once would require only  $n - 1$  many transitions. Thus, by the Pigeonhole principle, in processing the string  $\sigma$  the machine visits some state  $q$  twice. That is, in processing  $\sigma$  the machine makes a loop.

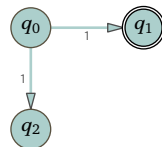
Said another way, the accepted string  $\sigma$  decomposes as  $\sigma = \alpha \wedge \beta \wedge \gamma$  where processing the  $\alpha$  portion takes the machine from the start to the loop (from state  $q_0$  to  $q$ ), the  $\beta$  portion takes the machine around the nontrivial loop (so that  $\beta \neq \epsilon$  brings the machine from  $q$  to  $q$  again) and the  $\gamma$  portion brings the machine to some final state. But then the recognized language is infinite, since clearly the machine also accepts  $\alpha \wedge \beta \wedge \beta \wedge \gamma$ , etc. (Note that we can take  $\beta$  to be of length less than  $n$  because otherwise the size of the machine would imply it has a subloop.)

Conversely, suppose the language of accepted strings is infinite. At least one of them is longer than  $n$ ; fix one that is longer but of minimal length. As in the prior paragraph, because the machine only has  $n$  states, by the Pigeonhole principle the accepted string  $\sigma$  decomposes as  $\sigma = \alpha \wedge \beta \wedge \gamma$  where the  $\beta$  portion is a nontrivial loop so processing  $\alpha$  brings the machine from  $q_0$  to some  $q$ , then processing  $\beta \neq \epsilon$  brings the machine from  $q$  to  $q$  again, and then processing the  $\gamma$  portion brings the machine to some final state. Because the string is of minimal length,  $|\beta| < n$ . In total then,  $|\sigma| < 2n$ .

**IV.4.30**

- (A)  $\hat{h}(01) = aba$ ,  $\hat{h}(10) = baa$ ,  $\hat{h}(101) = baaba$
- (B) A regular expression to describe  $\hat{\mathcal{L}} = \{\sigma \wedge 1 \mid \sigma \in \mathbb{B}^*\}$  is  $(0|1)^*1$  A regular expression to describe  $\hat{h}(\hat{\mathcal{L}})$  is  $(a|ba)^*ba$ .
- (C) Let  $\mathcal{L}$  be described by the regular expression  $R$ . Apply the function  $h$  to each non-meta symbol in  $R$ . The result is a regular expression that describes  $h(\mathcal{L})$ .

**IV.4.31** For this machine the language of accepted strings contains 1.



If we complement the set of accepting states then the language of the resulting machine also contains 1.

**IV.4.32**

- (A) We prove this by induction on the length of  $\sigma_1$ . For the base case consider both  $\sigma_1 = \epsilon$  and  $\sigma_1 = a$  for some  $a \in \Sigma$ . With both, the statement obviously holds.

For the inductive case assume that the statement is true when  $|\sigma_1| = 0$ ,  $|\sigma_1| = 1, \dots, |\sigma_1| = k$  and consider the  $|\sigma_1| = k + 1$  case. In this case  $\sigma_1 = \alpha \wedge a$  for some character  $a \in \Sigma$  and string  $\alpha \in \Sigma^*$  of length  $k$ . Then  $(\sigma_0 \wedge \sigma_1)^R = (\sigma_0 \wedge \alpha \wedge a)^R$ . Apply the base case to get  $= a^R \wedge (\sigma_0 \wedge \alpha)^R$ . The inductive hypothesis gives  $= a \wedge \alpha^R \wedge \sigma_0^R$ , and finish by recognizing that as  $\sigma_1^R \wedge \sigma_0^R$ .

- (B) We prove this by induction on the length of the regular expression. The base case is that  $|R| = 1$ . If  $R = \emptyset$  then by condition (i)  $R^R$  is the regular expression  $\emptyset$ , and  $\mathcal{L}(R) = \mathcal{L}(R^R)$  because both equal the empty set, as required. If  $R = \varepsilon$  then by condition (ii)  $R^R = \varepsilon$ , and  $\mathcal{L}(R) = \{\varepsilon\} = \mathcal{L}(R^R)$ , again as required. And, similarly, if  $R = x$  for some  $x \in \Sigma$  then by condition (iii)  $R^R = x$  and  $\mathcal{L}(R) = \{x\} = \mathcal{L}(R^R)$ .

Now for the inductive step. Assume that the statement is true for regular expressions  $R$  of length  $n = 1, n = 2, \dots, n = k$ , and consider an expression of length  $n = k + 1$ . We handled the case that it is a concatenation,  $R = R_0 \wedge R_1$ , in this question's first item.

If  $R = R_0 | R_1$  then  $\mathcal{L}(R^R) = \mathcal{L}((R_0 | R_1)^R) = \mathcal{L}(R_0^R | R_1^R)$  by condition (v). By the definition of the language associated with a pipe regular expression following Definition 3.4 that gives  $= \mathcal{L}(R_0^R) \cup \mathcal{L}(R_1^R)$ . The inductive hypothesis gives  $= \mathcal{L}(R_0)^R \cup \mathcal{L}(R_1)^R$ , which equals  $\{\sigma_0^R \mid \sigma_0 \in \mathcal{L}(R_0)\} \cup \{\sigma_1^R \mid \sigma_1 \in \mathcal{L}(R_1)\}$  and again applying the material following Definition 3.4 that equals  $\mathcal{L}(R)^R$ .

The final form is similar. If  $R = R_0^*$  then  $\mathcal{L}(R^R) = \mathcal{L}((R_0^*)^R)$ . By the definition following Definition 3.4 that equals  $\{\sigma_0 \wedge \dots \wedge \sigma_m \mid \sigma_i \in \mathcal{L}(R_0^R)\}$ . Rewriting gives  $\{\tau_0^R \wedge \dots \wedge \tau_m^R \mid \tau_j \in \mathcal{L}(R_0)\}$ . By this exercise's first item that equals  $\{(\tau_m \wedge \dots \wedge \tau_0)^R \mid \tau_j \in \mathcal{L}(R_0)\}$ , which is the set  $(\mathcal{L}(R_0)^*)^R$ .

- IV.5.8** There is no such regular expression. Definition 3.4 just does not have the capacity for that construct.
- IV.5.9** Compared with  $\sigma = \alpha\beta\gamma$ , the string  $\alpha\gamma$  has had  $\beta$  omitted. By definition  $\sigma$  has balanced parentheses. Because  $\beta$  consists of at least one open parentheses,  $($ , its omission of  $\beta$  means that in  $\alpha\gamma$  the number of open parentheses is at least one fewer than the number of closed parentheses.
- IV.5.10** If a language is finite,  $\mathcal{L} = \{\sigma_0, \dots, \sigma_n\}$ , then the conclusion of the Pumping Lemma holds by taking the pumping length to be  $p > \max(|\sigma_0|, \dots, |\sigma_n|)$ .
- IV.5.11** It is not correct.

The first problem is that it is not sensible. Strings don't have states.

Even if you figure that they meant to introduce a machine and say something like, "if a language has a string where the Finite State machine that recognizes the language has fewer states than the length of the string then it cannot be a regular language" then it still isn't sensible because if there is a Finite State machine then the language is regular.

And, even if we stretch figuring out what they meant all the way to making things up for them and assume they meant something like this, "If in an argument where we assumed the language is recognized by a Finite State machine we show that the language has a string that is longer than the number of states in the machine, then we have the desired contradiction" then the statement, while now technically sensible, is wrong. For example, there is a Finite State machine that accepts strings described by  $a^*$ , and there certainly are strings of that form whose length is greater than the number of states in the machine.

#### IV.5.12

- (A) Five elements of  $\mathcal{L}_0$  are  $bb, abbb, aabbbb, a^3b^5$ , and  $a^4b^6$ . Five strings that are not elements are  $\varepsilon, a, ab, aabbb, \text{ and } baabbbb$ .

Assume for contradiction that  $\mathcal{L}_0$  is regular. By the Pumping Lemma it has a pumping length,  $p$ . Consider  $\sigma = a^p b^{p+2}$ . This string is a member of the language that is of length greater than or equal to  $p$  so the Pumping Lemma gives a decomposition  $\sigma = \alpha \wedge \beta \wedge \gamma$  subject to the three conditions. Condition (1) is that  $|\alpha\beta| \leq p$ , so these two substrings contain only a's. Condition (2) is that  $\beta$  is nonempty, so it consists of at least one a.

Consider the list  $\alpha\gamma, \alpha\beta^2\gamma, \dots$ . Its second element  $\alpha\beta^2\gamma$  has more a's, but no more b's, than does  $\sigma = \alpha\beta\gamma$ . That means  $\alpha\beta^2\gamma$  is not an element of the language  $\mathcal{L}_0$ , a contradiction to the Pumping Lemma's condition (3).

- (B) In prose, this language consists of a's followed by b's and then followed by c's, with only the restriction that the number of a's equals the number of c's. Thus five elements of  $\mathcal{L}_1$  are:  $\varepsilon, ac, b, aabcc, \text{ and } bb$ . Five that are not are:  $aac, acc, ab, bbc, \text{ and } ca$ .

Assume for contradiction that  $\mathcal{L}_1$  is regular. Then it has a pumping length  $p$ . Consider  $\sigma = a^p c^p$ . Because  $\sigma \in \mathcal{L}_1$  and  $|\sigma| \geq p$ , the Pumping Lemma says that it decomposes,  $\sigma = \alpha \wedge \beta \wedge \gamma$ , in a way that satisfies the three conditions. By condition (1) the first two substrings together are short,  $|\alpha\beta| \leq p$ . Thus these two consist entirely of a's. By condition (2) the second substring  $\beta$  is nonempty, so it consists of at least one a. Finally, condition (3) gives that the strings  $\alpha\gamma, \alpha\beta^2\gamma, \dots$  are all members of the language  $\mathcal{L}_1$ . But the

string  $\alpha\gamma$ , when compared with  $\sigma = \alpha\beta\gamma$ , has at least one fewer a but the same number of c's. That means it does not have the same number of the two characters, so it is not a member of the language, which is a contradiction.

- (c) A prose description is that in  $\mathcal{L}_2$  the strings have a's followed by b's, with the number of a's less than the number of b's. So five strings that are members of  $\mathcal{L}_2$  are abb, abbb, abbbb,  $ab^5$ , and  $a^4b^5$ . Five strings that are not elements are  $\epsilon$ , a, ab, aab, and baabbb.

For contradiction suppose that  $\mathcal{L}_2$  is regular and fix a pumping length,  $p$ . The string  $\sigma = a^p b^{p+1}$  is an element of the language whose length is greater than or equal to the pumping length. So the Pumping Lemma gives a decomposition,  $\sigma = \alpha\beta\gamma$ , subject to the three conditions. Condition (1), that  $|\alpha\beta| \leq p$ , says that these two substrings contain only a's. Condition (2), that  $\beta$  is nonempty, says that it consists of at least one a.

Consider the strings  $\alpha\gamma$ ,  $\alpha\beta^2\gamma$ ,  $\dots$ . Compared with  $\sigma = \alpha\beta\gamma$ , the string  $\alpha\beta^2\gamma$  has more a's, but no more b's. Thus the number of a's is at least  $p + 1$  while the number of b's is fixed at  $p + 1$ . That means that  $\alpha\beta^2\gamma$  is not a member of the language  $\mathcal{L}_2$ , which contradicts the Pumping Lemma's condition (3).

- IV.5.13** The Pumping Lemma only allows us to conclude that  $\alpha\beta$  consists of ('s, not that this substring got all the ('s. The substring  $\gamma$  indeed got all of the close parentheses, the )'s, but that we know, it may have gotten some of the open parentheses as well.

**IV.5.14** Yes, that is right.

- IV.5.15** Let  $\mathcal{L} = \{\sigma \in \{a, b\}^* \mid \text{the number of a's is greater than the number of b's}\}$ . Assume for contradiction that it is regular. Then the Pumping Lemma applies and this language has a pumping length,  $p$ . The string  $\sigma = a^{p+1}b^p$  is an element of the language whose length is greater than or equal to  $p$ . So there is a decomposition  $\sigma = \alpha\beta\gamma$  that satisfies the three conditions. Condition (1) gives that  $|\alpha\beta| \leq p$  and so these two substrings contain only a's. Condition (2) gives that  $\beta$  is nonempty and thus it consists of at least one a.

The contradiction comes on considering condition (3)'s list  $\alpha\gamma$ ,  $\alpha\beta^2\gamma$ ,  $\dots$ . The first entry in that list,  $\alpha\gamma$ , when compared with  $\sigma = \alpha\beta\gamma$  will have at least one a fewer, and the same number of b's. It is therefore not a member of the language  $\mathcal{L}$ , contradicting condition (3). This contradiction shows that  $\mathcal{L}$  is not regular.

**IV.5.16**

- (A) Five strings that are members are aaa, aaab, aaabb, aaabbb, and aaabbbb. This language is regular, and a regular expression that describes it is  $aaab^*$ .
- (B) Five strings in this language are bbb, abbbb, aabbbbb,  $a^3b^6$ , and  $a^4b^7$ . This language is not regular.

To get a contradiction assume that it is regular. The Pumping Lemma says that the language has a pumping length,  $p$ . The string  $\sigma = a^p b^{p+3}$  is a member of the language and has length greater than or equal to the pumping length. So it decomposes,  $\sigma = \alpha\beta\gamma$ , in a way that is subject to the three conditions. The first condition is that  $|\alpha\beta| \leq p$  and so the two substrings  $\alpha$  and  $\beta$  consist only of a's. The second condition is that  $|\beta| > 0$  and so the substring  $\beta$  consists of at least one a.

Finally, consider the list  $\alpha\gamma$ ,  $\alpha\beta^2\gamma$ ,  $\dots$ . Condition (3) says that any element of that list is a member of the language. However, the list's first element has fewer a's than  $\sigma$  but the same number of b's, and therefore the number of a's is not three less than the number of b's. This is a contradiction.

- (c) Five members of this language are  $\epsilon$ , abab, babbab, aaaa, and aaabaaab. It is not regular.

Suppose otherwise, that it is regular. By the Pumping Lemma the language has a pumping length,  $p$ . Consider  $\sigma = a^p b^p$ . It is a member of the language whose length is greater than or equal to  $p$  and so it has a decomposition  $\sigma = \alpha\beta\gamma$  that satisfies the three conditions. The first condition,  $|\alpha\beta| \leq p$ , gives that the two substrings contain only a's. The second condition,  $|\beta| > 0$ , gives that the second substring consists of at least one a.

Now consider the list from the third condition,  $\alpha\gamma$ ,  $\alpha\beta^2\gamma$ ,  $\dots$ . Compare  $\alpha\gamma$  to  $\sigma = \alpha\beta\gamma$ . Because the  $\beta$  is omitted in  $\alpha\gamma$ , before its first b it has fewer a's than  $\sigma$  has before its first b. But before the second b the two strings have the same number of a's. So  $\alpha\gamma$  is not a member of the language, which contradicts the third condition.

- (D) Five strings in this language are  $\epsilon$ , aab, b, aaaabbbb, and abbbb. This language is regular. A regular expression that describes it is  $a^*b^*$ .
- (E) Five strings in this language are  $b^{13}$ ,  $ab^{14}$ ,  $b^{14}$ ,  $a^2b^{15}$ , and  $ab^{15}$ . This language is not regular.

Assume that it is regular, for contradiction. Then the language has a pumping length,  $p$ . Let  $\sigma = a^p b^{p+13}$ . By the Pumping Lemma it has a decomposition into  $\sigma = \alpha\beta\gamma$  that satisfies the three conditions. As a consequence of the first condition the first two substrings consist only of a's. A consequence of the second condition is that  $\beta$  consists of at least one a.

Consider  $\alpha\gamma, \alpha\beta^2\gamma, \dots$ . Its second element,  $\alpha\beta^2\gamma$ , has at least one more a than does  $\sigma$ . Thus its number of a's is not more than 12 less than its number of b's, which means that it is not a member of the language. That contradicts the Pumping Lemma's third condition.

**IV.5.17** The first is not regular while the second is regular.

(A) For contradiction, suppose that this language is regular. Then it has a pumping length,  $p$ . Consider the string  $\sigma = a^{p^2} b^p$ . It is a member of the language whose length is greater than or equal to  $p$  so it decomposes into  $\sigma = \alpha\beta\gamma$  in a way that satisfies the three conditions. Condition (1) implies that both  $\alpha$  and  $\beta$  consist only of a's. Condition (2) implies that  $\beta$  consists of at least one a.

Now consider condition (3)'s list,  $\alpha\gamma, \alpha\beta^2\gamma, \dots$ . Compared to  $\sigma = \alpha\beta\gamma$ , the entry  $\alpha\gamma$  has at least one fewer a but the same number of b's. Therefore the number of a's is not the square of the number of b's, and so  $\alpha\gamma$  is not in the language. That contradicts condition (3).

(B) A regular expression that describes this language is  $aaaa^*bbbb^*$ .

**IV.5.18** Five strings from the language are  $cb, acbb, aacbbb, aaacbbbb, \text{ and } aaaacbbbb$ .

To show that this language is not regular, assume for contradiction that it is. Then the language has a pumping length,  $p$ . Consider  $\sigma = a^p cb^{p+1}$ . Because this string is a member of the language and has length at least  $p$ , the Pumping Lemma gives that it decomposes into  $\sigma = \alpha\beta\gamma$ , subject to the three conditions.

The first condition,  $|\alpha\beta| \leq p$ , implies that the two substrings  $\alpha$  and  $\beta$  contain only a's. The second condition,  $|\beta| > 0$ , implies that  $\beta$  consists of at least one a.

Now, in the list  $\alpha\gamma, \alpha\beta^2\gamma, \dots$  consider the first one,  $\alpha\gamma$ . By the prior paragraph, compared with  $\sigma = \alpha\beta\gamma$  this string has at least one fewer a but the same number of b's. So the number of a's is not one less than the number of b's, and this string is not in the language, which is a contradiction to the third condition.

**IV.5.19** The language is regular. It is a trick question: take  $\alpha = \epsilon$  to get that the set equals  $\mathbb{B}^*$ .

**IV.5.20** Assume that it is regular and let the pumping length be  $p$ . Take  $\sigma = 1^{p!}$ . This string is a member of  $\mathcal{L}$  and has length at least  $p$ , so the Pumping Lemma says it decomposes as  $\sigma = \alpha\beta\gamma$  subject to the three conditions. The second condition implies that  $|\beta| > 0$ . Consider the list  $\alpha\gamma, \alpha\beta^2\gamma, \alpha\beta^3\gamma, \dots$ . After the first, the difference between the lengths of successive list elements is always  $|\beta|$ . But, as the hint notes, the difference between successive factorials is not fixed, instead it grows without bound. So there is a list element that is not in the language, which contradicts the third condition. So the language is not regular.

**IV.5.21** The first is regular, and is described by the regular expression  $\emptyset^*1\emptyset^*$ . The second is not regular. To prove that assume for contradiction that it is regular. Then it has a pumping length,  $p$ . Consider  $\sigma = \emptyset^p 1 \emptyset^p$ , which is a member of the language of length at least  $p$ , so the Pumping Lemma says it decomposes into  $\sigma = \alpha\beta\gamma$  in a way that satisfies the three conditions. The first condition implies that the  $\alpha$  and  $\beta$  substrings consist only of  $\emptyset$ 's. The second condition says that  $\beta$  consists of at least one  $\emptyset$ .

Consider the strings on the third condition's list  $\alpha\gamma, \alpha\beta^2\gamma, \dots$ . By the prior paragraph the first,  $\alpha\gamma$ , has fewer  $\emptyset$ 's before the 1 than does  $\sigma$ , but the same number of  $\emptyset$ 's after the 1. Hence it is not a member of the language, contradicting the Pumping Lemma's third condition.

**IV.5.22** The language  $\mathcal{L}_4 = \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \text{ and } i + j = k \text{ and } k < 4.\}$  is finite, and any finite language is regular.

As to the language if all sums,  $\mathcal{L} = \{a^i b^j c^k \mid i, j, k \in \mathbb{N} \text{ and } i + j = k.\}$ , suppose that it is regular and fix a pumping length  $p$ . Consider  $\sigma = a^p b^0 c^p$ . Because  $|\sigma| \geq p$  it has a decomposition  $\sigma = \alpha\beta\gamma$  subject to the three conditions of the Pumping Lemma. By the first condition the length  $|\alpha\beta|$  is less than or equal to  $p$ , and so consists entirely of a's. By the second condition the string  $\beta$  is nonempty and so consists of at least one a. The third condition says that  $\alpha\gamma$  is also a member of the language, but that is a contradiction since this string has fewer a's than c's, and no b's, and thus is not a member of the language.

**IV.5.23**

(A) It is regular because it is finite,  $\{00000, 00001, 00010, \dots, 11111\}$ .

(B) The language  $\mathcal{L} = \{\sigma \in \mathbb{B}^* \mid \text{the number of } \emptyset\text{'s minus the number of } 1\text{'s is five}\}$  is not regular. For, suppose

otherwise. Fix a pumping length  $p$  and consider  $\sigma = \theta^{p+5}1^p$ . It is a member of  $\mathcal{L}$  with length that is at least  $p$ , so the Pumping Lemma applies. Decompose it into  $\sigma = \alpha\beta\gamma$  subject to the three conditions. Condition (1), that  $|\alpha\beta| \leq p$ , implies that these two substrings contain only  $\theta$ 's. Condition (2) says that  $\beta$  is not empty so it consists of at least one  $\theta$ .

Condition (3) says that every string on the list  $\alpha\gamma, \alpha\beta^2\gamma, \dots$  is a member of  $\mathcal{L}$  but that is not true. By the prior paragraph the string  $\alpha\gamma$  has at least one  $\theta$  less than the string  $\sigma = \alpha\beta\gamma$  but the same number of 1's, and is therefore not an element of  $\mathcal{L}$ .

**IV.5.24** Call this language  $\mathcal{L}$ . Where  $\hat{\mathcal{L}} = \{\theta^m 1^n \in \mathbb{B}^* \mid m = n\}$ , we have  $\hat{\mathcal{L}} = \mathbb{B}^* - \mathcal{L}$ . By Theorem 4.5 regular languages are closed under set difference. Clearly  $\mathbb{B}^*$  is regular—it is described by the regular expression  $(\theta|1)^*$ —so if  $\mathcal{L}$  were regular then it would imply that  $\hat{\mathcal{L}}$  is regular also. But  $\hat{\mathcal{L}}$  is not regular, since it is the language of balanced parentheses, Example 5.2 with  $\theta$ 's substituted for open parentheses and 1's for closed parentheses.

**IV.5.25** Following the hint, the language  $\mathcal{L}$  is the set of strings that start and end with the same symbol. For instance, starting with a, each ab marks a transition to a block of b's (possibly only one), and each ba marks a transition back to a block of a's. Thus the regular expression is  $(a(a|b)^*a) \mid (b(a|b)^*b)$ .

**IV.5.26** The proof of the Pumping Lemma uses the Pigeonhole Principle to show that there is at least one loop and it then reasons with that loop. It is unaffected by there being a later loop.

More explicitly, the Pumping Lemma says that we can take the pumping length to be the number of states in any machine that recognizes the language. So set  $p = 9$ . If  $\sigma = aabbabbaba$  then because  $|\sigma| \geq p$  and  $\sigma \in \mathcal{L}$ , we know there is a decomposition satisfying the three conditions. One such is  $\alpha = a$ ,  $\beta = abbabb$ , and  $\gamma = aba$ . Checking that (1)  $|\alpha\beta| \leq p$ , that (2)  $\beta \neq \varepsilon$ , and (3) the strings  $\alpha\gamma, \alpha\beta^2\gamma, \dots$  are all members of  $\mathcal{L}$ .

If  $\sigma = a(abb)^{200}aba$  then again because  $|\sigma| \geq p$  and  $\sigma \in \mathcal{L}$ , we know there is a decomposition satisfying the three conditions. One such is  $\alpha = a$ ,  $\beta = abb$ , and  $\gamma = (abb)^{199}aba$ . Checking the three conditions is routine.

Finally, if  $\sigma = aabbbabbaa$  then we can take  $\alpha = aab$ ,  $\beta = bbabba$ , and  $\gamma = a$ . Or, we could instead take  $\alpha = aab$ ,  $\beta = bba$ , and  $\gamma = bbaa$ . Again, checking the three conditions is routine.

**IV.5.27** Let  $\mathcal{L} = \{\sigma \in \mathbb{B}^* \mid \sigma = 1^n \text{ where } n \text{ is prime}\}$ . Assume for contradiction that it is regular and fix a pumping length  $p$ . Consider condition (3)'s list.

$$\alpha\gamma, \alpha\beta^2\gamma, \alpha\beta^3\gamma, \dots, \alpha\beta^n\gamma, \dots$$

Except for the first one, all of the gaps between list items have the same width,  $|\beta|$ . So the length of the general item  $\alpha\beta^n\gamma$  is  $|\alpha\beta^2\gamma| + (n-2) \cdot |\beta|$ , for  $n \geq 2$ . Take  $n$  to be  $|\alpha\beta^2\gamma| + 2$  and that length is not prime. So not every string in condition (3)'s list is a member of  $\mathcal{L}$ , which is the desired contradiction.

**IV.5.28**

(A) The number of c's is the product of the number of a's and the number of b's. Five such strings are abc, abbcc, abbbccc, aabcc, and aabbbccc.

(B) For contradiction assume that this language,  $\mathcal{L}$ , is regular. The Pumping Lemma says that  $\mathcal{L}$  has a pumping length,  $p$ . Consider  $\sigma = a^p b^p c^{(p^2)}$ . That string has more than  $p$  characters so it decomposes as  $\sigma = \alpha\beta\gamma$ , subject to the three conditions. Condition (1) is that  $|\alpha\beta| \leq p$  and so both substrings  $\alpha$  and  $\beta$  are composed entirely of a's. Condition (2) is that  $\beta$  is not the empty string and so  $\beta$  consists of at least one a.

Condition (3) is that the strings in the list  $\alpha\gamma, \alpha\beta^2\gamma, \alpha\beta^3\gamma, \dots$  are also members of the language. We will get the contradiction from the first one,  $\alpha\gamma$ . Compared to  $\sigma = \alpha\beta\gamma$ , in the first one the  $\beta$  is gone. So the number of a's in  $\alpha\gamma$  is less than the number in  $\sigma = \alpha\beta\gamma$ . But the number of b's and the number of c's is the same. Thus in  $\alpha\gamma$  the number of a's times the number of b's does not equal the number of c's.

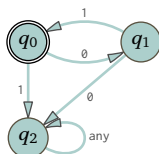
**IV.5.29** We need that  $|\alpha\beta| \leq 4$ , that  $|\beta| > 0$ , and that the strings  $\alpha\gamma, \alpha\beta^2\gamma, \alpha\beta^3\gamma, \dots$  are members of  $\mathcal{L}$ .

(A)  $\alpha = \varepsilon$ ,  $\beta = a$ , and  $\gamma = bbb$

(B)  $\alpha = \varepsilon$ ,  $\beta = b$ , and  $\gamma = b^{14}$

**IV.5.30**

(A) This works.



(B) The prior item shows that there is a Finite State machine recognizing this language that has three states. So, any non-empty word of the language must cause it to repeat a state, and therefore can be pumped. The minimum length of a non-empty word of the language is 2 so the minimum pumping length is at most 2. However the language contains no word of length 1 so if  $\sigma \in \mathcal{L}$  and  $|\sigma| \geq 1$  then automatically  $|\sigma| \geq 2$ , and so  $\sigma$  can be pumped. Therefore the minimum pumping length for the language is 1.

**IV.5.31**

- (A) Because it accepts the input  $\varepsilon$  the initial state  $q_0$  must be an accepting state. If  $q_0$  were the only accepting state then this machine accepts  $a$  by virtue of a loop from  $q_0$  to itself, which would make the machine accept  $aa$ . That is not in the language  $\mathcal{L}_1$ , so there must be at least one other accepting state.
- (B) Because it accepts the input  $\varepsilon$  the initial state  $q_0$  must be an accepting state. If  $q_0$  were the only accepting state then this machine accepts  $a$  by virtue of a loop from  $q_0$  to itself, which would make the machine accept  $aaa$ . That string is not a member of the language  $\mathcal{L}_2$  so there must be at least one accepting state other than  $q_0$ .

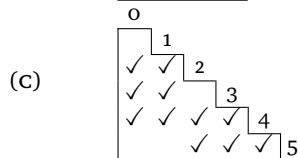
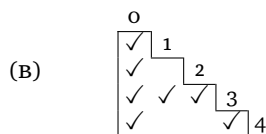
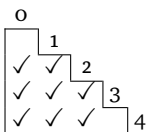
If the machine accepts  $aa$  by being in state  $q_0$  then that involves a loop, so the string  $aaaa$  would also be accepted by this machine. But that string is not in the language  $\mathcal{L}_2$ , so the extended transition function on this input,  $\hat{E}(aa)$ , is not equal to the state  $q_0$ . Similarly, if  $\hat{E}(aa)$  were equal to  $\hat{E}(a)$  then there would be a loop from this state to itself on input  $a$ , and so the machine would also accept  $aaa$ . But the machine does not accept  $aaa$  because that string is not in the language  $\mathcal{L}_2$ , so there must be a third accepting state.

- (C) This is the natural extension of the prior two items. Consider  $\mathcal{L}_n = \{\varepsilon, a, \dots, a^n\}$ . If  $\hat{E}(a^n)$  were equal to  $\hat{E}(a^i)$  for any  $i < n$  then there would be a loop and so the machine would accept additional strings.

**IV.6.8** Pairs that are not 1-distinguishable are:  $q_0, q_1$ , and  $q_2, q_2$ , and  $q_1, q_2$ , so one class is  $\mathcal{E}_{i,0} = \{q_0, q_1, q_2\}$ . Besides that, another pair of states that are not 1-distinguishable is  $q_3, q_5$  so another class is  $\mathcal{E}_{i,1} = \{q_3, q_5\}$ . The last class is  $\mathcal{E}_{i,2} = \{q_4\}$ .

**IV.6.9**

- (A) The given classes say that we can  $i$ -distinguish  $q_0$  from  $q_2, q_3$ , and  $q_4$ . So looking down the column headed by 0 we see checkmarks in rows labeled 2, 3, and 4. The other columns work the same way.



**IV.6.10**

- (A) These are the checkmarks.

	a	b	a	b
✓ $q_0, q_1$	$q_1, q_1$	$q_2, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_0, q_2$	$q_1, q_2$	$q_2, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
$q_0, q_5$	$q_1, q_5$	$q_2, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_1, q_2$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_1, q_5$	$q_1, q_5$	$q_3, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
✓ $q_2, q_5$	$q_2, q_5$	$q_4, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
$q_3, q_4$	$q_3, q_4$	$q_5, q_5$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$

For example, the first row is checkmarked because on the right side the second entry, the b entry, has two classes that differ,  $\mathcal{E}_{0,0} \neq \mathcal{E}_{0,1}$ .

- (B) The checkmarks show that we can 1-distinguish  $q_0$  from  $q_1$  and  $q_2$  but not from  $q_5$ . And, we can tell  $q_1$  and  $q_2$  from  $q_5$  but not from each other. The matching  $\sim_1$  classes are  $\mathcal{E}_{1,0} = \{q_0, q_5\}$ ,  $\mathcal{E}_{1,1} = \{q_1, q_2\}$ , and  $\mathcal{E}_{1,2} = \{q_3, q_4\}$

**IV.6.11** First, all the states are reachable, by inspection.

Start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.



The checkmark denote a pair of states that are 0-distinguishable and the blanks denote pairs of states that are 0-indistinguishable. Here are the two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_2\} \quad \mathcal{E}_{0,1} = \{q_1\}$$

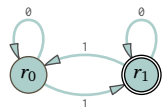
Next we see if any of these classes split.

$q_0, q_2$	0	1	0	1
	$q_0, q_2$	$q_1, q_1$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$

The answer is that they don't split. The process stops and here are the two  $\sim$ -equivalence classes.

$$\mathcal{E}_{1,0} = \{q_0, q_2\} = r_0 \quad \mathcal{E}_{1,1} = \{q_1\} = r_1$$

Incorporating the arrow information from the original machine gives this minimized one.

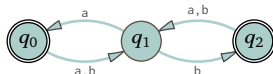


**IV.6.12**

- (A) These are the sets.

Step $n$	$R_n$
0	$\{q_0\}$
1	$\{q_0, q_1\}$
2	$\{q_0, q_1, q_2\}$
3	$\{q_0, q_1, q_2\}$

Thus the set of reachable states is  $R = \{q_0, q_1, q_2\}$ . The set of unreachable states is  $Q - R = \{q_3\}$ . Here is the machine with only the reachable states.

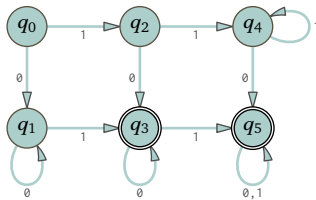


- (B) These are the sets.

Step $n$	$R_n$
0	$\{q_0\}$
1	$\{q_0, q_1, q_3\}$
2	$\{q_0, q_1, q_3, q_4\}$
3	$\{q_0, q_1, q_3, q_4\}$

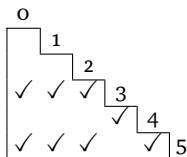
The set of reachable states is  $R = R_2 = \{q_0, q_1, q_3, q_4\}$ . The set of unreachable states is  $Q - R = \{q_2, q_5\}$ .

**IV.6.13** For convenience, here is that machine.



Every state is reachable, by inspection.

Start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.



There is a checkmark in box  $i, j$  if the states  $q_i$  and  $q_j$  are 0-distinguishable. The blanks denote pairs that are 0-indistinguishable. There are two  $\sim_0$  equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_1, q_2, q_4\} \quad \mathcal{E}_{0,1} = \{q_3, q_5\}$$

The next step is to see if any of these classes split.

	0	1	0	1
✓ $q_0, q_1$	$q_1, q_1$	$q_2, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_0, q_2$	$q_1, q_3$	$q_2, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_0, q_4$	$q_1, q_5$	$q_2, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_1, q_2$	$q_1, q_3$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
✓ $q_1, q_4$	$q_1, q_5$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
$q_2, q_4$	$q_3, q_5$	$q_4, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_3, q_5$	$q_3, q_5$	$q_5, q_5$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$

The only pairs of states that are 1-distinguishable are  $q_2, q_4$  and  $q_3, q_5$ . These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0, q_2\} \quad \mathcal{E}_{1,1} = \{q_1\} \quad \mathcal{E}_{1,2} = \{q_2, q_4\} \quad \mathcal{E}_{1,3} = \{q_3, q_5\}$$

Now try it again to see if any of these classes split.

	0	1	0	1
$q_2, q_4$	$q_3, q_5$	$q_4, q_4$	$\mathcal{E}_{1,3}, \mathcal{E}_{1,3}$	$\mathcal{E}_{1,2}, \mathcal{E}_{1,2}$
$q_3, q_5$	$q_3, q_5$	$q_5, q_5$	$\mathcal{E}_{1,3}, \mathcal{E}_{1,3}$	$\mathcal{E}_{1,3}, \mathcal{E}_{1,3}$

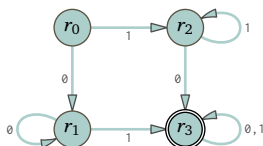
The only pairs of states that are 1-distinguishable are  $q_2, q_4$  and  $q_3, q_5$ . These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0, q_2\} \quad \mathcal{E}_{1,1} = \{q_1\} \quad \mathcal{E}_{1,2} = \{q_2, q_4\} \quad \mathcal{E}_{1,3} = \{q_3, q_5\}$$

There is no splitting; these are the  $\sim$  classes.

$$r_0 = \mathcal{E}_{2,0} = \{q_0, q_2\} \quad r_1 = \mathcal{E}_{2,1} = \{q_1\} \quad r_2 = \mathcal{E}_{2,2} = \{q_2, q_4\} \quad r_3 = \mathcal{E}_{2,3} = \{q_3, q_5\}$$

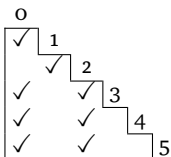
Bringing in the transition information from the original machine gives this minimized version.



**IV.6.14** Not a hell of a lot. You get back the original machine in that you get  $r_0 = \mathcal{E}_{n,0} = \{q_0\}$ ,  $r_1 = \mathcal{E}_{n,1} = \{q_1\}$ , etc.

**IV.6.15** By inspection the states  $q_6$  and  $q_7$  are unreachable, so drop them.

With that, start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.

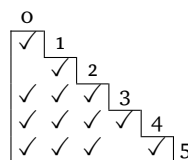


The checkmarks are for pairs of states that are 0-distinguishable, and the blanks denote pairs that are 0-indistinguishable. There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_2\} \quad \mathcal{E}_{0,1} = \{q_1, q_3, q_4, q_5\}$$

Next we see if any of these classes split.

	a	b	a	b
✓ $q_1, q_3$	$q_4, q_0$	$q_2, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_1, q_4$	$q_4, q_4$	$q_2, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_1, q_5$	$q_4, q_2$	$q_2, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_3, q_4$	$q_0, q_4$	$q_4, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_3, q_5$	$q_0, q_2$	$q_4, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_4, q_5$	$q_4, q_2$	$q_4, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
$q_0, q_2$	$q_1, q_1$	$q_3, q_5$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$

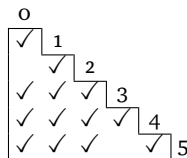


So,  $q_0$  and  $q_2$  are not 1-distinguishable. On the other hand, as shown in the triangular table,  $q_1$  is 1-distinguishable from  $q_3, q_4$ , and  $q_5$ . And,  $q_3$  is 1-distinguishable from  $q_4$  but not from  $q_5$ . Finally,  $q_4$  is 1-distinguishable from  $q_5$ . These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0, q_2\} \quad \mathcal{E}_{1,1} = \{q_1\} \quad \mathcal{E}_{1,2} = \{q_3, q_5\} \quad \mathcal{E}_{1,3} = \{q_4\}$$

Determine which states are 2-distinguishable by computing whether any of those classes split.

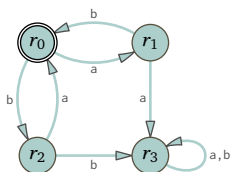
	a	b	a	b
$q_3, q_5$	$q_0, q_2$	$q_4, q_4$	$\mathcal{E}_{1,0}, \mathcal{E}_{1,0}$	$\mathcal{E}_{1,3}, \mathcal{E}_{1,3}$
$q_0, q_2$	$q_1, q_1$	$q_3, q_5$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,1}$	$\mathcal{E}_{1,2}, \mathcal{E}_{1,2}$



There is no splitting so the minimal machine has four states.

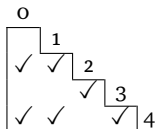
$$r_0 = \mathcal{E}_{2,0} = \{q_0, q_2\} \quad r_1 = \mathcal{E}_{2,1} = \{q_1\} \quad r_2 = \mathcal{E}_{2,2} = \{q_3, q_5\} \quad r_3 = \mathcal{E}_{2,3} = \{q_4\}$$

The transitions from the input machine give this minimized one.



**IV.6.16** The answer is “yes.” The initial state of the minimal machine,  $r_0$ , is the one that contains  $q_0$ . In the minimized machine a state is accepting if and only if it contains any final states in the original machine. So if  $q_0$  is accepting then  $r_0$  is also accepting.

**IV.6.17** By inspection all of the states are reachable. So start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.

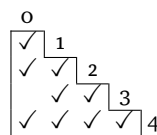


There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_1, q_3\} \quad \mathcal{E}_{0,1} = \{q_2, q_4\}$$

Next we see if any of these classes split.

	0	1	0	1
✓ $q_0, q_1$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_0, q_3$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
$q_1, q_3$	$q_2, q_2$	$q_4, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_2, q_4$	$q_1, q_4$	$q_4, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$

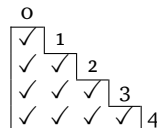


So,  $q_1$  and  $q_3$  are not 1-distinguishable but all the others are 1-distinguishable. These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0\} \quad \mathcal{E}_{1,1} = \{q_1, q_3\} \quad \mathcal{E}_{1,2} = \{q_2\} \quad \mathcal{E}_{1,3} = \{q_4\}$$

Step 2 is to decide of any of the  $\sim_1$  classes split.

	0	1	0	1
✓ $q_1, q_3$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,2}$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,3}$

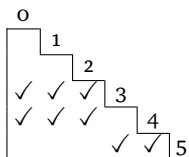


So,  $q_1$  and  $q_3$  are 2-distinguishable. These are the  $\sim_2$  classes.

$$\mathcal{E}_{1,0} = \{q_0\} \quad \mathcal{E}_{1,1} = \{q_1\} \quad \mathcal{E}_{1,2} = \{q_2\} \quad \mathcal{E}_{1,3} = \{q_3\} \quad \mathcal{E}_{1,4} = \{q_4\}$$

The starting machine is minimal.

**IV.6.18** By eye, all of the states are reachable. So start with the triangular table, checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.

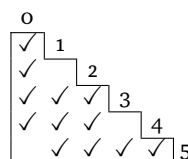


There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_1, q_2, q_5\} \quad \mathcal{E}_{0,1} = \{q_3, q_4\}$$

Next we see if we can 1-distinguish any of these states that cannot be 0-distinguished.

	a	b	a	b
✓ $q_0, q_1$	$q_1, q_1$	$q_2, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_0, q_2$	$q_1, q_2$	$q_2, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
$q_0, q_5$	$q_1, q_5$	$q_2, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_1, q_2$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_1, q_5$	$q_1, q_5$	$q_3, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
✓ $q_2, q_5$	$q_2, q_5$	$q_4, q_5$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
$q_3, q_4$	$q_3, q_4$	$q_5, q_5$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$

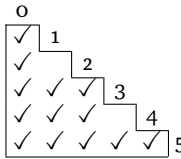


These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0, q_5\} \quad \mathcal{E}_{1,1} = \{q_1, q_2\} \quad \mathcal{E}_{1,2} = \{q_3, q_4\}$$

Iterate. See if any of these classes split.

	a	b	a	b
✓ $q_0, q_5$	$q_1, q_5$	$q_2, q_5$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,0}$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,0}$
$q_1, q_2$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,1}$	$\mathcal{E}_{1,2}, \mathcal{E}_{1,2}$
$q_3, q_4$	$q_3, q_4$	$q_5, q_5$	$\mathcal{E}_{1,2}, \mathcal{E}_{1,2}$	$\mathcal{E}_{1,0}, \mathcal{E}_{1,0}$

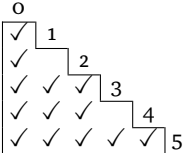


The class  $\mathcal{E}_{1,0}$  splits in two. The pairs that are not 2-distinguishable are:  $q_1, q_2$ , and  $q_3, q_4$ . These are the  $\sim_2$  classes.

$$\mathcal{E}_{2,0} = \{q_0\} \quad \mathcal{E}_{2,1} = \{q_1, q_2\} \quad \mathcal{E}_{2,2} = \{q_3, q_4\} \quad \mathcal{E}_{2,3} = \{q_5\}$$

Again.

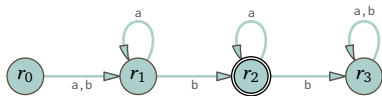
	a	b	a	b
$q_1, q_2$	$q_1, q_2$	$q_3, q_4$	$\mathcal{E}_{2,1}, \mathcal{E}_{2,1}$	$\mathcal{E}_{2,2}, \mathcal{E}_{2,2}$
$q_3, q_4$	$q_3, q_4$	$q_5, q_5$	$\mathcal{E}_{2,2}, \mathcal{E}_{2,2}$	$\mathcal{E}_{2,3}, \mathcal{E}_{2,3}$



There is no splitting. These are the  $\sim$  classes.

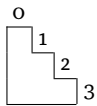
$$r_0 = \mathcal{E}_{3,0} = \{q_0\} \quad r_1 = \mathcal{E}_{3,1} = \{q_1, q_2\} \quad r_2 = \mathcal{E}_{3,2} = \{q_3, q_4\} \quad r_3 = \mathcal{E}_{3,3} = \{q_5\}$$

Here is the minimized machine.



**IV.6.19** We will go through the algorithm but before we do, observe that the answer must be that a machine with a minimal number of states and that accepts no string has one state, namely the initial state, and all arrows are loops.

By inspection all of the states are reachable. So start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.

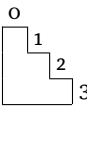


There is one  $\sim_0$ -equivalence class.

$$\mathcal{E}_{0,0} = \{q_0, q_1, q_2, q_3\}$$

Next we see if any of these classes split.

	0	1	0	1
$q_0, q_1$	$q_1, q_1$	$q_2, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_0, q_2$	$q_1, q_0$	$q_2, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_0, q_3$	$q_1, q_0$	$q_2, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_1, q_2$	$q_1, q_0$	$q_3, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_1, q_3$	$q_1, q_0$	$q_3, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_2, q_3$	$q_0, q_0$	$q_2, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$



Of course, if there is only one class then it cannot split, so there is only one  $\sim_1$  class.

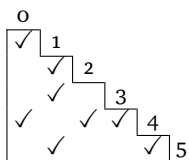
$$\mathcal{E}_{0,0} = \{q_0, q_1, q_2, q_3\}$$

This is the resulting minimized machine, as predicted.



If a machine has that all states are accepting then a similar thing happens. It recognizes the language  $\Sigma^*$  and the minimal such machine has one state, which is both initial and accepting.

**IV.6.20** By inspection all of the states are reachable. So start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.

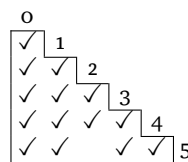


There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_2, q_3, q_5\} \quad \mathcal{E}_{0,1} = \{q_1, q_4\}$$

Next we see if any of these classes split.

	a	b	a	b
✓ $q_0, q_2$	$q_1, q_1$	$q_0, q_4$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_0, q_3$	$q_1, q_5$	$q_0, q_2$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_0, q_5$	$q_1, q_4$	$q_0, q_1$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_2, q_3$	$q_1, q_5$	$q_4, q_2$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,0}$
✓ $q_2, q_5$	$q_1, q_4$	$q_4, q_1$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$
✓ $q_3, q_5$	$q_5, q_4$	$q_2, q_1$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$
✓ $q_1, q_4$	$q_3, q_3$	$q_0, q_4$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$

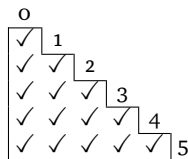


So,  $q_2$  and  $q_5$  are not 1-distinguishable but all the other pairs are 1-distinguishable. These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0\} \quad \mathcal{E}_{1,1} = \{q_1\} \quad \mathcal{E}_{1,2} = \{q_2, q_5\} \quad \mathcal{E}_{1,3} = \{q_3\} \quad \mathcal{E}_{1,4} = \{q_4\}$$

Finally we see if the one remaining multi-state classes splits.

	a	b	a	b
$q_2, q_5$	$q_1, q_4$	$q_4, q_1$	$\mathcal{E}_{1,1}, \mathcal{E}_{1,4}$	$\mathcal{E}_{1,4}, \mathcal{E}_{1,1}$



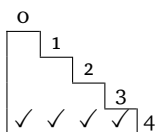
Thus all of the states are not 2-distinguishable. These are the  $\sim_2$  classes.

$$\mathcal{E}_{2,0} = \{q_0\} \quad \mathcal{E}_{2,1} = \{q_1\} \quad \mathcal{E}_{2,2} = \{q_2\} \quad \mathcal{E}_{2,3} = \{q_3\} \quad \mathcal{E}_{2,4} = \{q_4\} \quad \mathcal{E}_{2,5} = \{q_5\}$$

The original machine is minimal.

**IV.6.21** You will get a machine that recognizes the same language. The reachable states of this machine will form a minimal set. But there will still be unreachable states (although perhaps fewer of them.)

**IV.6.22** By inspection all of the states are reachable. So start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.



There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{0, 1, 2, 3\} \quad \mathcal{E}_{0,1} = \{4\}$$

Next we see if any of these classes split.

	a	b	a	b
$q_0, q_1$	$q_1, q_2$	$q_0, q_1$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_0, q_2$	$q_1, q_3$	$q_0, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_0, q_3$	$q_1, q_4$	$q_0, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
$q_1, q_2$	$q_2, q_3$	$q_1, q_2$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_1, q_3$	$q_2, q_4$	$q_1, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$
✓ $q_2, q_3$	$q_3, q_4$	$q_2, q_3$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$

So,  $q_3$  is 1-distinguishable from all the other states, but no other pairs are 1-distinguishable. These are the  $\sim_1$  classes.

$$\mathcal{E}_{1,0} = \{q_0, q_1, q_2\} \quad \mathcal{E}_{1,1} = \{q_3\} \quad \mathcal{E}_{1,2} = \{q_4\}$$

Next time through.

	a	b	a	b
$q_0, q_1$	$q_1, q_2$	$q_0, q_1$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,0}$
✓ $q_0, q_2$	$q_1, q_3$	$q_0, q_2$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,0}$
✓ $q_1, q_2$	$q_2, q_3$	$q_1, q_2$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,1}$	$\mathcal{E}_{1,0}, \mathcal{E}_{0,0}$

So,  $q_2$  is 2-distinguishable from  $q_0$  and  $q_1$ , giving these  $\sim_2$  classes.

$$\mathcal{E}_{2,0} = \{q_0, q_1\} \quad \mathcal{E}_{2,1} = \{q_2\} \quad \mathcal{E}_{2,2} = \{q_3\} \quad \mathcal{E}_{2,3} = \{q_4\}$$

Once more through.

	a	b	a	b
✓ $q_0, q_1$	$q_1, q_2$	$q_0, q_1$	$\mathcal{E}_{2,0}, \mathcal{E}_{2,1}$	$\mathcal{E}_{2,0}, \mathcal{E}_{2,0}$

So,  $q_1$  is 3-distinguishable from  $q_0$  and these are the giving these  $\sim_3$  classes.

$$\mathcal{E}_{3,0} = \{q_0\} \quad \mathcal{E}_{3,1} = \{q_1\} \quad \mathcal{E}_{3,2} = \{q_2\} \quad \mathcal{E}_{3,3} = \{q_3\} \quad \mathcal{E}_{3,4} = \{q_4\}$$

No additional splitting happens on the next iteration; indeed, there is nothing left to split. The process stops. The minimized machine has five states; it is the machine we started with.

**IV.6.23**

- (A) We must show that it is reflexive, symmetric, and transitive. Fix some  $n$ . Obviously any state is  $n$ -indistinguishable from itself, for any  $n$ . Symmetry is also clear. So suppose  $q_0 \sim_n q_1$  and  $q_1 \sim_n q_2$ . Let  $\sigma$  be a string of length less than or equal to  $n$ . If  $\sigma$  takes the machine from  $q_0$  to an accepting state then because of  $q_0 \sim_n q_1$  it also takes the machine from  $q_1$  to an accepting state. Because of  $q_1 \sim_n q_2$  it also takes the machine from  $q_2$  to an accepting state. The same holds for non-accepting states.
- (B) It is an equivalence because it is  $\sim_k$  for some  $k$ .

**IV.6.24**

- (A) By inspection all of the states are reachable. So start by checkmarking the  $i, j$  entries where one of  $q_i$  and  $q_j$  is accepting while the other is not.



There are two  $\sim_0$ -equivalence classes.

$$\mathcal{E}_{0,0} = \{q_0, q_2\} \quad \mathcal{E}_{0,1} = \{q_1\}$$

Next we see if any of these classes split.

$q_0, q_2$	a	b	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	a	b
$q_0, q_2$	$q_1, q_1$	$q_2, q_2$	$\mathcal{E}_{0,1}, \mathcal{E}_{0,1}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$	$\mathcal{E}_{0,0}, \mathcal{E}_{0,0}$



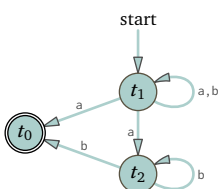
So the class  $\mathcal{E}_{0,0}$  does not split. These are the  $\sim_1$  classes.

$$r_0 = \mathcal{E}_{0,0} = \{q_0, q_2\} \quad r_1 = \mathcal{E}_{0,1} = \{q_1\}$$

Here is the minimized machine.



(B) Here is the machine with the arrows reversed, the nodes renamed, and the start node made final, and the final one made a start.

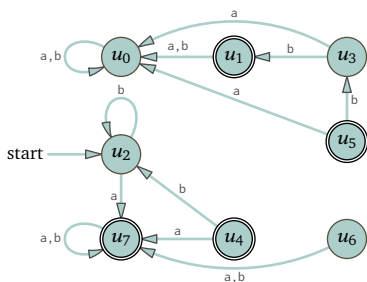


It is nondeterministic; for instance, two edges labeled a leave node  $t_1$ .

(C) This is the table to convert that nondeterministic Finite State machine to a deterministic one.

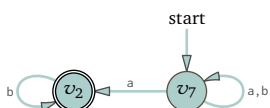
Node name	Set of nodes	a	b
$u_0$	$\emptyset$	$u_0$	$u_0$
+ $u_1$	$\{t_0\}$	$u_0$	$u_0$
$u_2$	$\{t_1\}$	$u_7$	$u_2$
$u_3$	$\{t_2\}$	$u_0$	$u_1$
+ $u_4$	$\{t_0, t_1\}$	$u_7$	$u_2$
+ $u_5$	$\{t_0, t_2\}$	$u_0$	$u_3$
$u_6$	$\{t_1, t_2\}$	$u_7$	$u_7$
+ $u_7$	$\{t_0, t_1, t_2\}$	$u_7$	$u_7$

Here is the arrow diagram.



The only reachable states are  $u_2$  and  $u_7$ .

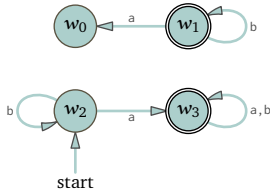
(D) Here is the picture.



(E) Convert that nondeterministic Finite State machine to a deterministic one.

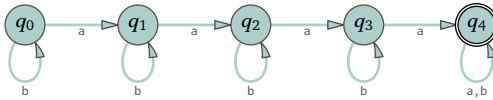
Node name	Set of nodes	a	b
$w_0$	$\emptyset$	$w_0$	$w_0$
+ $w_1$	$\{v_2\}$	$w_0$	$w_1$
$w_2$	$\{v_7\}$	$w_3$	$w_2$
+ $w_3$	$\{v_2, v_7\}$	$w_3$	$w_3$

Here is the arrow diagram.



Once we omit unreachable states, this machine is the same (except for state renaming) as the one in the first item.

**IV.6.25** The machine of Exercise 6.22



has five states and is minimal for the language described by  $b^*ab^*ab^*ab^*a(a|b)^*$ . Using induction to generalize this observation is straightforward.

**IV.7.8** This is a regular language so you don't need the stack.

**IV.7.12** It is the language of even-length palindromes,  $\{\tau \hat{\ } \tau^R \mid \tau \in \mathbb{B}^*\}$ .

**IV.7.13**

$$X \rightarrow aXc \mid b$$

**IV.7.14**

$$T \rightarrow aTa \mid bTb \mid b$$

**IV.7.16**

$$S \rightarrow aSa \mid bSb \mid cTc \mid a \mid b \mid c \mid \varepsilon$$

$$T \rightarrow aTa \mid bTb \mid a \mid b \mid c \mid \varepsilon$$

**IV.7.18** One direction is that if a string  $\sigma \in \mathbb{B}^*$  is generated by the grammar then it is a palindrome  $\sigma = \sigma^R$ . We do induction on the grammar's structure. The basis step is that the string is either empty  $\sigma = \varepsilon$ , or of length one  $\sigma = \emptyset$  or  $\sigma = 1$ . All three cases are evidently palindromes. For the inductive step we assume the length of the string  $|\sigma|$  is greater than one and that the statement is true for all shorter strings, all  $\tau \in \mathbb{B}^*$  with  $|\tau| < |\sigma|$ . In the case that  $\sigma = \emptyset\tau\emptyset$ , the string  $\sigma$  is a palindrome by the inductive hypothesis that  $\tau$  is a palindrome. The case that  $\sigma = 1\tau 1$  is similar.

The other direction is that if the string is a palindrome  $\sigma = \sigma^R$  then it is generated by the grammar. We can use induction on the length  $|\sigma|$ . If the length of  $\sigma \in \mathbb{B}^*$  is 0 then the string is empty,  $\sigma = \varepsilon$ . In this case it is a palindrome and is also derived from the grammar since  $P \rightarrow \varepsilon$  is one of the listed production rules. If the length of  $\sigma$  is 1 then either  $\sigma = \emptyset$  or  $\sigma = 1$ , and these two are both palindromes and also derived from the grammar. Finally, assume that  $|\sigma| > 1$  and that the statement is true for all strings  $\tau \in \mathbb{B}^*$  of length less than  $\sigma$ . There are two cases. If  $\sigma$ 's first character is 0 then because  $\sigma$  is a palindrome it has the form  $\sigma = \emptyset\tau\emptyset$ , where  $\tau$  is a palindrome. The induction hypothesis applies to give that  $\tau$  is derived from the grammar. But then so is  $\sigma$ , using the production  $P \rightarrow \emptyset P \emptyset$ . The case that  $\sigma$ 's first character is 1 works the same way.

**IV.A.20**

- (A)  $.^*abc.^*$
- (B)  $.^*abc+.^*$
- (C)  $.^*abc2.^*$

- (D) `.*abc2,5.*`  
 (E) `.*abc2,.*`  
 (F) `.*a(b|c).*` or `.*a[bc].*`

**IV.A.21**

- (A) `.*abe.*` Note that in practice you often can just write `abe`; see your language's regular expression documentation.  
 (B) `[a-zA-Z0-9]*`  
 (C) `[^]`

**IV.A.22**

- (A) `(a|e|i|o|u).*bc.*`  
 (B) `(a|e|i|o|u)(bc|.*bc).*`  
 (C) `.*a.*e.*i.*o.*u.*`  
 (D) `(.*a.*e.*i.*o.*u.*)|(.*a.*e.*i.*u.*o.*)|...|(.*u.*o.*i.*e.*a.*)` There are  $5! = 120$  clauses.  
 You might use a program to write this regex.

**IV.A.23** `.*([.*|{.*}].*)`**IV.A.24** `\d{10}`**IV.A.25**

- (A) `(\d{3}|\(\d{3}\)) \d{3}-\d{4}`  
 (B) We must factor out the space as well as the area code: `(\d{3} |\(\d{3}\) )?\d{3}-\d{4}`.

**IV.A.27** `\d{2}:\d{2}:\d{2}(\. \d +)|\d{2}(:\d{2})?`**IV.A.28** One is `[-\+]?(\d)+\.\?|\d)*\.\d+`**IV.A.29**

- (A) `4(\d15|\d12)`  
 (B) `(5[1-5]\d2|222[1-9]|22[3-9]\d|2[3-6]\d2|27[01]\d|2720)\d12`  
 (C) `3[47][0-9]13`

**IV.A.30**

- (A) We can do the six as alternatives.

$$[A-Z]\d2 \d[A-Z]2|[A-Z]\d \d[A-Z]2|[A-Z]\d[A-Z] \d[A-Z]2|[A-Z]2\d2 \d[A-Z]2|[A-Z]2\d \d[A-Z]2|[A-Z]2\d[A-Z] \d[A-Z]2$$

- (B) We can factor out the ending of `[A-Z]2`.

$$[A-Z]\d2|[A-Z]\d|[A-Z]\d[A-Z]|[A-Z]2\d2|[A-Z]2\d|[A-Z]2\d[A-Z]) \d[A-Z]2$$
**IV.A.31** A suitable regex is `textasciicircum g.n.{3}i.$` and a match is 'gynaecia', meaning the aggregate of carpels or pistils in a flower.**IV.A.32** Factoring out the whitespace would mean that times without `am` or `pm` must have a trailing whitespace.**IV.A.33** This tries to match a character after the end of the string: `$`. (In practice, in a regular expression multiline mode the dollar sign matches a newline character and so this can match. But that lies outside our scope.)**IV.A.34** `^M0,4(CM|CD|D?C0,3)(XC|XL|L?X0,3)(IX|IV|V?I0,3)$`**IV.A.35** Assume it is a regular language and let its pumping length be  $p$ . Consider  $\sigma = a^p b a^p b$ . It is an element of  $\mathcal{L}$  whose length is greater than  $p$  so By the Pumping Lemma it decomposes as  $\sigma = \alpha \beta \gamma$  subject to the three conditions. By the first condition the length of  $\alpha \beta$  is less than  $p$ , and so these two substrings consist solely of a's. The second condition is that  $\beta$  is not the empty string. The third condition says that the strings  $\alpha \gamma, \alpha \beta^2 \gamma, \dots$  are all members of  $\mathcal{L}$ . Note that the first is not a square because in omitting  $\beta$  it omits at least one a from the prefix but not from the suffix. That's a contradiction.**IV.A.36**

- (A) We take the alphabet to be  $\Sigma = \mathbb{B}$ . Assume that it is regular and let its pumping length be  $p$ . Consider  $\sigma = \theta^p 1 \theta^p$ , which is a member of the language. By the Pumping Lemma it therefore decomposes into  $\sigma = \alpha \beta \gamma$  subject to the three conditions. By the first and second conditions  $\alpha \beta$  consists solely of  $\theta$ 's, and is

therefore part of the prefix before the 1, and  $\beta$  is nonempty. By the third condition the string  $\alpha\gamma$  is also a member of  $\mathcal{L}$ . But this is a contradiction because the omission of  $\beta$  means that  $\alpha\gamma$  has fewer 0's before the 1 than after.

(B)  $(a^*)b\backslash 1$

**IV.A.37**

(A)  $.^*r$

(B)  $.^*i.^*t.^*$

(C)  $.^*foo.^*$

**IV.A**

(A) HELP

(B)

$(A|B|C)\backslash 1$   
 $(AB|OE|SK)$

$.^*M?0.^*$	B	O
$(AN FE BE)$	B	E

**IV.B.15** If  $k \neq n$  then  $a^k b^k \in \mathcal{L}$  but  $a^m b^k \notin \mathcal{L}$ , so there are infinitely many classes.

## Chapter V: Computational Complexity

**V.1.27** True. Remember that Big  $\mathcal{O}$  means something like “grows at a rate less than or equal to.” Growing at a rate less than or equal to  $n^2$  implies growing at a rate less than or equal to  $n^3$ .

**V.1.28** First, the assertion, “I have an algorithm with running time  $\mathcal{O}(x^2)$ ” does not mean the time is quadratic, since if  $f(x) = x$  then  $f$  is  $\mathcal{O}(n^2)$ .

Also, even if the running time is quadratic, it may involve constants and lower order terms. Thus,  $1000 \cdot x^2$  is  $\mathcal{O}(n^2)$ , as is  $x^2 + 3x + 42$ . All of the things that impact performance on a particular platform are not part of the algorithm analysis, which only applies to a computing model.

(Another issue is that the runtime behavior is quadratic only for sufficiently large inputs. So unless you know that 5 is greater than the  $N$  of the definition, there is no reason to think quadratic at all.)

**V.1.29** No doubt they mean that for their problem they have an algorithm that is  $\mathcal{O}(n^3)$ .

**V.1.30**

- (A) Because 5 in binary is 101, it takes three bits.
- (B) The number 50 in binary is 11 0010, so it takes six bits.
- (C) In binary the number 500 is 1 1111 0100, so it takes nine bits.
- (D) The binary equivalent of 5 000 is 1 0011 1000 1000, which is thirteen bits.

This table verifies these using the formula  $\text{bits}(n) = 1 + \lceil \lg(n) \rceil$ .

<i>integer n</i>	5	50	500	5 000
<i>binary</i>	101	110 010	1 1111 0100	1 0011 1000 1000
$\lg(n)$	2.322	5.644	8.966	12.288
$1 + \lg(n)$	3	6	9	13

**V.1.31** The second is true, the first is not.

Recall the intuition that ‘ $f$  is  $\mathcal{O}(g)$ ’ means something like “ $f$ ’s growth rate is less than or equal to  $g$ ’s,” while ‘ $f$  is  $\Theta(g)$ ’ means something like “ $f$ ’s growth rate is roughly the same as  $g$ ’s.” With this in mind, the second one makes sense since if  $f$  and  $g$  grow at roughly the same rate then that certainly implies  $f$ ’s that rate is less than or equal to  $g$ ’s. More precisely, by definition, if  $f$  is  $\Theta(g)$  then both  $f$  is  $\mathcal{O}(g)$  and  $g$  is  $\mathcal{O}(f)$ .

Again with the intuition in mind, assuming that  $f$ ’s growth is less than or equal to  $g$ ’s does not give that the two rates are equal. That is, there are functions  $f, g$  so that  $f$  is  $\mathcal{O}(g)$  but  $f$  is not  $\Theta(g)$ . One such pair is  $f(n) = n$  and  $g(n) = n^2$ .

**V.1.32** We use the principles for simplifying Big  $\mathcal{O}$  expressions from page 267.

- (A)  $\mathcal{O}(n^2)$
- (B)  $\mathcal{O}(2^n)$
- (C)  $\mathcal{O}(n^4)$
- (D)  $\mathcal{O}(\lg n)$

**V.1.33**

- (A) For sufficiently large input arguments  $n$ , this function is  $f(n) = 0$ . So it is  $\mathcal{O}(1)$ .
- (B) For sufficiently large arguments  $n$ , this function is  $f(n) = n^2$ . So it is  $\mathcal{O}(n^2)$ .
- (C) For sufficiently large  $n$ , this function is  $f(n) = \lg(n)$ . So it is  $\mathcal{O}(\lg(n))$ .

**V.1.34** Because we will apply L’Hôpital’s Rule, to signal that these are functions on  $\mathbb{R}$  we convert from  $n$ ’s to  $x$ ’s.

- (A) The limit of the ratios is this.

$$\lim_{x \rightarrow \infty} \frac{3x^3 + 2x + 4}{\ln(x) + 6}$$

This is an “ $\infty/\infty$ ” limit. Apply L'Hôpital's Rule.

$$\lim_{x \rightarrow \infty} \frac{3x^3 + 2x + 4}{\ln(x) + 6} = \lim_{x \rightarrow \infty} \frac{9x^2 + 2}{1/x} = \lim_{x \rightarrow \infty} \frac{x \cdot (9x^2 + 2)}{1} = \infty$$

So  $g$  is  $\mathcal{O}(f)$  but  $f$  is not  $\mathcal{O}(g)$ . In short,  $f$ 's growth rate is strictly bigger than  $g$ 's.

(B) To apply the theorem, consider the limit of the ratio. Use L'Hôpital's Rule.

$$\lim_{x \rightarrow \infty} \frac{3x^2 + 2x + 4}{x + 5x^3} = \lim_{x \rightarrow \infty} \frac{6x + 2}{1 + 15x^2} = \lim_{x \rightarrow \infty} \frac{6}{30x} = 0$$

Conclude that  $f$  is  $\mathcal{O}(g)$  but it is not the case that  $g$  is  $\mathcal{O}(f)$ . Briefly,  $f$ 's growth rate is strictly less than  $g$ 's.

(C) This is the limit of the ratios.

$$\lim_{x \rightarrow \infty} \frac{(1/2)x^3 + 12x^2}{x^2 \ln(x)}$$

We could start by simplifying the fraction by cancelling  $x^2$  from the numerator and the denominator. However, we instead choose to apply L'Hôpital's Rule. Note that we need to use the Product rule on the denominator.

$$\lim_{x \rightarrow \infty} \frac{(1/2)x^3 + 12x^2}{x^2 \ln(x)} = \lim_{x \rightarrow \infty} \frac{(3/2)x^2 + 24x}{2x \ln(x) + x} = \lim_{x \rightarrow \infty} \frac{3x + 24}{2 \ln(x) + 3} = \lim_{x \rightarrow \infty} \frac{3}{2(1/x)} = \lim_{x \rightarrow \infty} \frac{3x}{2} = \infty$$

So  $g$  is  $\mathcal{O}(f)$  but  $f$  is not  $\mathcal{O}(g)$ .

(D) L'Hôpital's Rule gives this.

$$\lim_{x \rightarrow \infty} \frac{\lg(x)}{\ln(x)} = \lim_{x \rightarrow \infty} \frac{(1/x) \cdot (1/\ln(2))}{(1/x)} = \lim_{x \rightarrow \infty} \frac{1}{\ln(2)} \cdot \frac{(1/x)}{(1/x)} = \lim_{x \rightarrow \infty} \frac{1}{\ln(2)}$$

So both  $g$  is  $\mathcal{O}(f)$  and  $f$  is  $\mathcal{O}(g)$ , that is,  $g$  is  $\Theta(f)$ . In short, they grow at equivalent rates.

(E) We have this.

$$\lim_{x \rightarrow \infty} \frac{x^2 + \lg(x)}{x^4 - x^3} = \lim_{x \rightarrow \infty} \frac{2x + (1/x)}{4x^3 - 3x^2} = \lim_{x \rightarrow \infty} \frac{2x^2 + 1}{4x^4 - 3x^3} = \lim_{x \rightarrow \infty} \frac{4x}{16x^3 - 9x^2} = \lim_{x \rightarrow \infty} \frac{4}{48x^2 - 18x} = 0$$

So  $f$  is  $\mathcal{O}(g)$  but  $g$  is not  $\mathcal{O}(f)$ .

(F) We consider the ratio of the functions.

$$\lim_{x \rightarrow \infty} \frac{55}{x^2 + x}$$

This is not an “ $\infty/\infty$ ” limit, so L'Hôpital's Rule doesn't apply. But it doesn't matter because it is easier than that. The denominator grows at a faster rate than the numerator because the numerator doesn't grow at all. That is,  $f$  is  $\mathcal{O}(g)$  but  $g$  is not  $\mathcal{O}(f)$ .

**V.1.35** We follow the principles for simplifying Big  $\mathcal{O}$  expressions.

- (A) Both of these are  $\mathcal{O}(n^2)$ . So both  $f$  is  $\mathcal{O}(g)$  and also  $g$  is  $\mathcal{O}(f)$ , that is,  $f$  is  $\Theta(g)$ .
- (B) Here,  $g$  is a logarithmic function and  $f$  is a cubic, so  $g$  is  $\mathcal{O}(f)$  but it is not the case that  $f$  is  $\mathcal{O}(g)$ .
- (C) The function  $f$  is quadratic, while  $g$  is a square root, so  $g$  is  $\mathcal{O}(f)$  but  $f$  is not  $\mathcal{O}(g)$ .
- (D) The function  $f$  is  $\mathcal{O}(n^{1.2})$  while the function  $g$  is  $\mathcal{O}(n^{\sqrt{2}})$ , which is approximately  $\mathcal{O}(n^{1.414})$ . Consequently,  $f$  is  $\mathcal{O}(g)$  but  $g$  is not  $\mathcal{O}(f)$ .
- (E) The exponential function  $2^{(1/6)n}$  grows faster than the polynomial function  $n^6$ . So  $f$  is  $\mathcal{O}(g)$  but it is not the case the  $g$  is  $\mathcal{O}(f)$ .
- (F) As is shown in ?? 1.24, the function  $2^n$  is  $\mathcal{O}(3^n)$ , while  $3^n$  is not  $\mathcal{O}(2^n)$ . Thus,  $g$  is  $\mathcal{O}(f)$  but  $f$  is not  $\mathcal{O}(g)$ .
- (G) Recall the logarithm rule that  $\lg(3n) = \lg(3) + \lg(n)$ . From this it follows that the two have equivalent growth rates, so  $f$  is  $\Theta(g)$ .

**V.1.36** Observe that  $\cos(t)$  varies between  $-1$  and  $1$ . Observe also that by the rules of logarithms  $\lg(5^n) = n \cdot \lg(5)$ . Thus, the ones with a growth rate that is less than or equal to  $n^2$ 's are the first, second, fourth, and fifth. The third one has a growth rate that is  $\Theta(n^3)$ .

**V.1.37**

- (A) True. By the simplifying principles from page 267, its growth rate is dominated by the quadratic term, so  $5n^2 + 2n$  is  $\mathcal{O}(n^2)$ , which on the Hardy hierarchy lies below  $n^3$ .  
 (B) True. As discussed in the section, all logarithmic functions grow at the same rate. So  $\ln n$  is  $\Theta(\lg n)$ , and hence  $\ln n$  is  $\mathcal{O}(\lg n)$ .  
 (C) True. Where  $f(n) = n^3 + n^2 + n$ , by the simplifying principles from page 267,  $f$  is  $\mathcal{O}(n^3)$ .  
 (D) True. For any polynomial function  $p$ , we have that  $p$  is  $\mathcal{O}(2^n)$ , by Lemma 1.22.

**V.1.38** For each we use the simplifying principles.

- (A)  $k = 4$   
 (B)  $k = 4$   
 (C)  $k = 3$  (The cosine function is bounded between  $-1$  and  $1$ .)  
 (D)  $k = 12$   
 (E) This is  $\mathcal{O}(n^{7/2})$  but since the question specifies that  $k \in \mathbb{N}$ , the answer is  $k = 4$ .

**V.1.39**

		$n = 1$	$n = 10$	$n = 50$	$n = 100$	$n = 200$
(A)	$\lg n$	–	$1.05 \times 10^{-17}$	$1.79 \times 10^{-17}$	$2.11 \times 10^{-17}$	$2.42 \times 10^{-17}$
	$n$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-17}$	$1.58 \times 10^{-16}$	$6.34 \times 10^{-16}$	$6.34 \times 10^{-16}$
	$n \lg n$	–	$1.05 \times 10^{-16}$	$8.94 \times 10^{-16}$	$2.11 \times 10^{-15}$	$4.85 \times 10^{-15}$
	$n^2$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-16}$	$7.92 \times 10^{-15}$	$3.17 \times 10^{-14}$	$1.27 \times 10^{-13}$
	$n^3$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-15}$	$3.96 \times 10^{-13}$	$3.17 \times 10^{-12}$	$2.54 \times 10^{-11}$
	$2^n$	$6.34 \times 10^{-18}$	$3.24 \times 10^{-15}$	$3.57 \times 10^{-3}$	$4.02 \times 10^{12}$	$5.09 \times 10^{42}$
(B)		$n = 1$	$n = 10$	$n = 50$	$n = 100$	
	$\lg n$	–	$1.05 \times 10^{-17}$	$1.79 \times 10^{-17}$	$2.11 \times 10^{-17}$	
	$n$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-17}$	$1.58 \times 10^{-16}$	$6.34 \times 10^{-16}$	
	$n \lg n$	–	$1.05 \times 10^{-16}$	$8.94 \times 10^{-16}$	$2.11 \times 10^{-15}$	
	$n^2$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-16}$	$7.92 \times 10^{-15}$	$3.17 \times 10^{-14}$	
	$n^3$	$3.17 \times 10^{-18}$	$3.17 \times 10^{-15}$	$3.96 \times 10^{-13}$	$3.17 \times 10^{-12}$	
	$2^n$	$6.34 \times 10^{-18}$	$3.24 \times 10^{-15}$	$3.57 \times 10^{-3}$	$4.02 \times 10^{12}$	
$3^n$	$9.51 \times 10^{-18}$	$1.87 \times 10^{-13}$	$2.76 \times 10^6$	$1.63 \times 10^{30}$		

**V.1.40** The number of seconds in a year is  $60 \cdot 60 \cdot 24 \cdot 365.25 = 31\,557\,600$ . At 10 000 ticks per second, that is about  $3.16 \times 10^{11}$  ticks in a year (as also described in ?? 1.25).

- (A) With  $\lg(n) = 3.16 \times 10^{11}$ , we have  $n = 2^{3.16 \times 10^{11}} = 4.53 \times 10^{94\,997\,841\,911}$ .  
 (B)  $(3.16 \times 10^{11})^2 = 9.96 \times 10^{22}$   
 (C)  $3.16 \times 10^{11}$   
 (D) Because  $(3.16 \times 10^{11})^{1/2} \approx 561\,761.52$ , the answer is 561 761.  
 (E) Because  $(3.16 \times 10^{11})^{1/3} \approx 6808.24$ , the answer is 6 808.  
 (F) We have  $\lg(3.16 \times 10^{11}) \approx 38.20$  so the answer is 38.

**V.1.41** Solving  $100\,000 \cdot n^2 = n^3$  gives  $n = 100\,000$ . So the first number  $n$  where  $f(n) = 100 \cdot n^2$  is less than  $g(n) = n^3$  is  $n = 100\,001$ .

**V.1.42** Linear. Finite State machines consume one character at each step. So they run in time equal to the length of the input. In the notation of Definition 1.26,  $t_M(\sigma) = |\sigma|$ .

**V.1.43** Let  $g$  be the function  $g(x) = 1$ .

- (A) Directly apply the definition of Big  $\mathcal{O}$ , Definition 1.6, by taking  $N = 1$  and  $C = 8$ . If  $x > N = 1$  then  $C \cdot g(x) = 8 \cdot 1 \geq f(x) = 7$ .  
 (B) Recall that  $\sin(x)$  is bounded between  $-1$  and  $1$ . Take  $N = 1$  and  $C = 9$ . If  $x > N$  then  $C \cdot g(x) = 9 \geq 7 + \sin(x) = f(x)$ .  
 (C) Fix  $N = 1$  and  $C = 8$ . If  $x \geq N = 1$  then  $C \cdot g(x) = 2$  is greater than  $f(x) = 7 + (1/x)$ .  
 (D) We first show that if it is bounded then it is  $\mathcal{O}(1)$ . Suppose that  $f$  is bounded by  $K \in \mathbb{R}^+$ . Take  $N = 1$

and  $C = K$ . If  $x \geq N = 1$  then  $C \cdot g(x) = K \geq f(x)$ , so  $f$  is  $\mathcal{O}(g)$ .

For the converse, suppose that  $f$  is  $\mathcal{O}(g)$ . Then there exists constants  $N, C \in \mathbb{R}^+$  so that  $x \geq N$  implies that  $C \cdot g(x) = C \geq f(x)$ . Let  $K = \max(C, f(0), \dots, f(N-1))$ . If  $x \in \mathbb{R}$  then  $f(x) \leq K$ , so  $f$  is bounded by  $K$ .

**V.1.44** A function  $f: \mathbb{R} \rightarrow \mathbb{R}$  is  $\mathcal{O}(1)$  if it is bounded by a constant. So  $g(x) \leq x^{\mathcal{O}(1)}$  says that the function  $g$  has at most a polynomial growth rate.

**V.1.45** The first half, that  $f$  is  $\mathcal{O}(g)$ , is straightforward. Fix  $N = 1$  and  $C = 2$ . If  $x \geq N = 1$  then  $C \cdot g(x) = 2x^2 \geq 2x = f(x)$  (because  $2x^2 - 2x = 0$  gives  $2 \cdot (x(x-1)) = 0$  and so the largest  $x$  for which they are equal is  $x = 1$ ).

For the second half, that  $g$  is not  $\mathcal{O}(f)$ , we will show that for every  $C$  there is no  $N$  that will work. So fix a  $C$ , aiming to prove that for any  $N$  it is not the case that  $x \geq N$  implies that  $C \cdot f(x) \geq g(x)$ . For any  $N$ , consider  $x = \max(N, C+1)$ , which is greater than or equal to  $N$ . Because  $x > C$ , we have that  $g(x) = x^2 > C \cdot x = C \cdot f(x)$ .

**V.1.46** Let  $N = 4$  and  $C = 1$ . If  $n \geq N = 4$  then  $C \cdot n! = n! = 4! \cdot 5 \cdot 6 \cdots (n-1) \cdot n \geq 2^4 \cdot 2^{n-4} = 2^n$  (note that  $4! = 24 > 2^4 = 16$ ).

**V.1.47**

(A) This limit equals 0

$$\lim_{x \rightarrow \infty} \frac{(\log_b(x))^2}{x^d} = \lim_{x \rightarrow \infty} \frac{2 \log_b(x) \cdot \frac{1}{x \ln(b)}}{dx^{d-1}} = \frac{2}{d \ln(b)} \cdot \lim_{x \rightarrow \infty} \frac{\log_b(x)}{x^d}$$

by the calculation in Example 1.20.

(B) This limit equals 0

$$\lim_{x \rightarrow \infty} \frac{(\log_b(x))^3}{x^d} = \lim_{x \rightarrow \infty} \frac{3(\log_b(x))^2 \cdot \frac{1}{x \ln(b)}}{dx^{d-1}} = \frac{3}{d \ln(b)} \cdot \lim_{x \rightarrow \infty} \frac{(\log_b(x))^2}{x^d}$$

by the calculation in the prior item.

(C) The prior item suggests an argument by induction. So assume that the statement holds for the powers  $n = 1, n = 2, \dots, n = k$  and consider the  $n = k + 1$  case.

$$\lim_{x \rightarrow \infty} \frac{(\log_b(x))^{k+1}}{x^d} = \lim_{x \rightarrow \infty} \frac{(k+1)(\log_b(x))^k \cdot \frac{1}{x \ln(b)}}{dx^{d-1}} = \frac{k+1}{d \ln(b)} \cdot \lim_{x \rightarrow \infty} \frac{(\log_b(x))^k}{x^d}$$

Finish by applying the  $n = k$  inductive hypothesis.

**V.1.48** Fix  $N = 1$  and find  $C \in \mathbb{R}$  large enough that  $C \cdot g(1) \geq f(1)$ . Because  $g$  is increasing, if  $x > N = 1$  then  $C \cdot g(x) \geq C \cdot g(1) \geq f(1) = f(x)$ .

**V.1.49**

(A) The function  $f_0(x) = 0$  is clearly computable and has output values that grow at a rate that is  $\mathcal{O}(1)$ . Similarly, the output values of  $f_1(x) = x$  grow at a rate that is  $\mathcal{O}(x)$ , and  $f_2(x) = x^2$  is  $\mathcal{O}(x^2)$ , etc.

(B) The function

$$K(x) = \begin{cases} 0 & \text{if } \phi_x(x) \uparrow \\ 1 & \text{if } \phi_x(x) \downarrow \end{cases}$$

is clearly  $\mathcal{O}(1)$ .

(C) Let  $M(n)$  be the maximum value of  $\phi_i(j)$  where  $0 \leq i, j \leq n$  and  $\phi_i(j) \downarrow$  (if there is no such maximum because none of the  $\phi_i(j)$  converge then define  $M(n) = 0$ ). This is not a computable function, but it is a function. Then the function  $\hat{M}(n) = M(n) + 1$  has the property that for any computable function  $\phi_e$ , for all  $n > e$  if  $\phi_e(n) \downarrow$  then  $\hat{M}(n) > \phi_e(n)$ .

**V.1.50** Consider the number  $n \in \mathbb{N}$ . The hint says that we can check whether a number divides  $n$  in quadratic time. Wrapping a loop around that to do all numbers  $m$  with  $2 < m < n$  adds one to the exponent. So the naive algorithm is  $\mathcal{O}(n^3)$  in the value of  $n$ .

However, in terms of bits, the number  $n$  is represented by about  $\lg(n)$ -many bits. That is, this expresses the runtime as a function of the size of the input.

$$\text{runtime} = n^3 = (2^{\text{size}})^3 = 2^{3 \cdot \text{size}}$$

So the relationship between the size of the input and the runtime is exponential.

**V.1.51** Use Theorem 1.17.

$$\lim_{x \rightarrow \infty} \frac{2^x}{3^x} = \lim_{x \rightarrow \infty} \left(\frac{2}{3}\right)^x = 0$$

**V.1.52**

(A) With an eye toward Theorem 1.17, consider the ratio.

$$\frac{n!}{n^n} = \frac{1}{n} \cdot \frac{2}{n} \cdots \frac{n}{n} \leq \frac{1}{n} \cdot 1 \cdots 1$$

Clearly the limit of the ratio  $n!/n^n$  is 0.

(B) No, don't overlook the  $e^{-n}$ .

**V.1.53**

(A) Apply L'Hôpital's Rule twice.

$$\lim_{x \rightarrow \infty} \frac{x^2 + 5x + 1}{x^2} = \lim_{x \rightarrow \infty} \frac{2x + 5}{2x} = \lim_{x \rightarrow \infty} \frac{2}{2} = 1$$

(B) Apply L'Hôpital's Rule, remembering to use the Chain Rule for the numerator.

$$\lim_{x \rightarrow \infty} \frac{\lg(x+1)}{\lg(x)} = \lim_{x \rightarrow \infty} \frac{(1/x \ln(2)) \cdot 1}{1/x \ln(2)} = 1$$

**V.1.54** No. If  $\mathcal{O}(f)$  is the set of polynomials then  $f$  must be a polynomial. However, then  $f$  has some degree, and no polynomial of higher degree is a member of  $\mathcal{O}(f)$ .

**V.1.55** Recall the logarithm rules that  $\log_b(x^a) = a \cdot \log_b(x)$  and  $b^{\log_b(x)} = x$ .

(A) L'Hôpital's Rule gives

$$\lim_{x \rightarrow \infty} \frac{\lg(x)}{(\lg(x))^2} = \lim_{x \rightarrow \infty} \frac{1/x \ln(2)}{2 \lg(x) \cdot 1/x \ln(2)} = \lim_{x \rightarrow \infty} \frac{1}{2 \lg(x)} = 0$$

and so Theorem 1.17 gives the required conclusion.

(B) By Theorem 1.17, it suffices to show that this is 0.

$$\lim_{x \rightarrow \infty} \frac{x^k}{x^{\lg(x)}} = \lim_{x \rightarrow \infty} x^{k - \lg(x)}$$

Rewrite.

$$x^{k - \lg(x)} = 2^{\lg(x^{k - \lg(x)})} = 2^{(k - \lg(x)) \cdot \lg(x)} = 2^{k \lg(x) - (\lg(x))^2}$$

Apply the prior item to conclude that the limit of the exponent is  $-\infty$ , and so the limit of the entire expression is 0.

(C) Taking  $\lg$  of  $x^{\lg(x)}$  gives  $\lg(x^{\lg(x)}) = \lg(x) \cdot \lg(x)$ . Taking  $\lg$  of the other expression gives  $\lg(2^{(\lg(x))^2}) = (\lg(x))^2 \cdot \lg(2)$ . Since the logarithm and exponential are inverse functions, that is  $(\lg(x))^2$ .

(D) Rewrite  $x^{\lg(x)}$  as  $2^{(\lg(x))^2}$ . Theorem 1.17

$$\lim_{x \rightarrow \infty} \frac{2^{(\lg(x))^2}}{2^x} = \lim_{x \rightarrow \infty} 2^{(\lg(x))^2 - x} = 0$$

gives the desired conclusion.

**V.1.56** We will use the notation from the lemma statement.

- (A) The assumption is that  $f$  is  $\mathcal{O}(g)$ . So there are constants  $N, C \in \mathbb{R}^+$  such that  $x \geq N$  implies that  $C \cdot g(x) \geq f(x)$ . For the function  $a \cdot f$ , use the constants  $\hat{N} = N$  and  $\hat{C} = a \cdot C$ . Then  $x \geq \hat{C}$  implies that  $\hat{C} \cdot g(x) \geq f(x)$ .
- (B) The function  $f_0$  is  $\mathcal{O}(g_0)$  and the function  $f_1$  is  $\mathcal{O}(g_1)$ . So there are constants  $N_0, C_0, N_1, C_1 \in \mathbb{R}^+$  such that  $x \geq N_0$  implies that  $C_0 g_0(x) \geq f_0(x)$ , and  $x \geq N_1$  implies that  $C_1 g_1(x) \geq f_1(x)$ . Take  $\hat{N} = \max(N_0, N_1)$  and  $\hat{C} = \max(C_0, C_1)$ . Then  $x \geq \hat{N}$  implies that  $\hat{C} \cdot (g_0(x) + g_1(x)) = \hat{C} g_0(x) + \hat{C} g_1(x) \geq f_0(x) + f_1(x)$ .
- (C) As in the prior item, there are constants  $N_0, C_0, N_1, C_1 \in \mathbb{R}^+$  such that  $x \geq N_0$  implies that  $C_0(x) \geq f_0(x)$ , and  $x \geq N_1$  implies that  $C_1(x) \geq f_1(x)$ . Take  $\hat{N} = \max(N_0, N_1)$  and  $\hat{C} = C_0 \cdot C_1$ . Then  $x \geq \hat{N}$  implies that  $\hat{C} \cdot (g_0(x) \cdot g_1(x)) = C_0 g_0(x) \cdot C_1 g_1(x) \geq f_0(x) \cdot f_1(x)$ .

**V.1.57**

- (A) We must show that  $f$  is in  $\mathcal{O}(f)$ . Take  $N = 1$  and  $C = 1$ .
- (B) Suppose that  $f_0$  is  $\mathcal{O}(f_1)$  and that  $f_1$  is  $\mathcal{O}(f_2)$ . Then there exists  $N_0, N_1, C_0, C_1 \in \mathbb{R}^+$  such that if  $x \geq N_0$  then  $C_0 \cdot f_1(x) \geq f_0(x)$ , and if  $x \geq N_1$  then  $C_1 \cdot f_2(x) \geq f_1(x)$ . Take  $\hat{N} = \max(N_0, N_1)$  and  $\hat{C} = C_0 \cdot C_1$ . Then  $x \geq \hat{N}$  implies that  $\hat{C} \cdot f_2(x) \geq C_0 \cdot f_1(x) \geq f_0(x)$ .

**V.1.58**

- (A) If  $\lim_{x \rightarrow \infty} f(x)/g(x)$  exists and equals 0 then for any  $\varepsilon > 0$  there is an  $N \in \mathbb{R}$  such that  $x \geq N$  implies that  $f(x)/g(x) < \varepsilon$ . That gives  $f(x) < \varepsilon \cdot g(x)$ . Fix  $\varepsilon = 1/2$  and  $C = 2$ . Take inverses to get that  $x \geq N$  implies  $C \cdot g(x) = 2g(x) \geq f(x)$ .
- (B) To show that  $f$  is  $\mathcal{O}(g)$ , take  $N = 1$  and  $C = 4$ .

On the other hand, the limit doesn't exist, as the ratio oscillates between 1 and 2.

**V.1.59** First use L'Hôpital's Rule to get that if  $g(x) = a_m x^m + \dots + a_0$  then  $g$  is  $\Theta(x^m)$ . Then Example 1.20 shows that any logarithmic function is  $\mathcal{O}(g)$ .

For the other half, fix a base  $b > 1$  and consider the exponential function  $h(x) = b^x$ . L'Hôpital's Rule gives this

$$\lim_{x \rightarrow \infty} \frac{x^m}{b^x} = \frac{m}{\ln(b)} \cdot \lim_{x \rightarrow \infty} \frac{x^{m-1}}{b^x} = \frac{m(m-1)}{(\ln(b))^2} \cdot \lim_{x \rightarrow \infty} \frac{x^{m-2}}{b^x} = \dots = \frac{m!}{(\ln(b))^m} \cdot \lim_{x \rightarrow \infty} \frac{1}{b^x} = 0$$

and Theorem 1.17 gives the desired conclusion.

**V.2.39** There are twenty five: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, and 97.

**V.2.40**

- (A) Yes, it is prime.
- (B) No, it is not prime. Besides the trivial ones, the divisors of 6 165 are 3, 5, 9, 15, 45, 137, 411, 685, 1 233, and 2 055.
- (C) Yes.
- (D) No. Besides itself and 1, this number is also divisible by 7 and 601.
- (E) No. Besides the trivial divisors of itself and 1, this number also has the divisors 3, 11, 33, 233, 699, and 2 563.

**V.2.41**

- (A) One is 3. The entire list is 3, 9, 3469, 10407, and 31221.
- (B) One is 2. The entire list is 2, 4, 8, 6553, 13106, 26212, and 52424.
- (C) One is 5. The entire list is 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 25, 30, 32, 40, 48, 50, 60, 64, 75, 80, 96, 100, 120, 128, 150, 160, 192, 200, 240, 300, 320, 384, 400, 480, 600, 640, 800, 960, 1200, 1600, 1920, 2400, 3200, 4800, and 9600.
- (D) One is 61. The entire list is 61, 71, 4331.
- (E) The only one is 877. It is prime.

**V.2.42**

- (A) The three-atom clause is  $P \vee \neg Q \vee R$ .
- (B) The clause for the  $F$ - $T$ - $T$  line is  $P \vee \neg Q \vee \neg R$ . For the  $T$ - $F$ - $T$  line, use  $\neg P \vee Q \vee \neg R$ . For the  $T$ - $T$ - $T$  line, use  $\neg P \vee \neg Q \vee \neg R$ .

(c) Here is the entire expression in conjunctive normal form.

$$(P \vee \neg Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (\neg P \vee Q \vee \neg R) \wedge (\neg P \vee \neg Q \vee \neg R)$$

Verifying the truth table is routine.

**V.2.43**

(A) This formula is satisfiable.

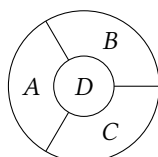
$P$	$Q$	$R$	$P \wedge Q$	$\neg Q$	$\neg Q \wedge R$	$(P \wedge Q) \vee (\neg Q \wedge R)$
F	F	F	F	T	F	F
F	F	T	F	T	T	T
F	T	F	F	F	F	F
F	T	T	F	F	F	F
T	F	F	F	T	F	F
T	F	T	F	T	T	T
T	T	F	T	F	F	T
T	T	T	T	F	F	T

(B) This formula is not satisfiable.

$P$	$Q$	$R$	$P \rightarrow Q$	$P \wedge Q$	$\neg P$	$(P \wedge Q) \vee \neg P$	$\neg(P \wedge Q) \vee \neg P$	$(P \rightarrow Q) \wedge \neg((P \wedge Q) \vee \neg P)$
F	F	F	T	F	T	T	F	F
F	F	T	T	F	T	T	F	F
F	T	F	T	F	T	T	F	F
F	T	T	T	F	T	T	F	F
T	F	F	F	F	F	F	T	F
T	F	T	F	F	F	F	T	F
T	T	F	T	T	F	T	F	F
T	T	T	T	T	F	T	F	F

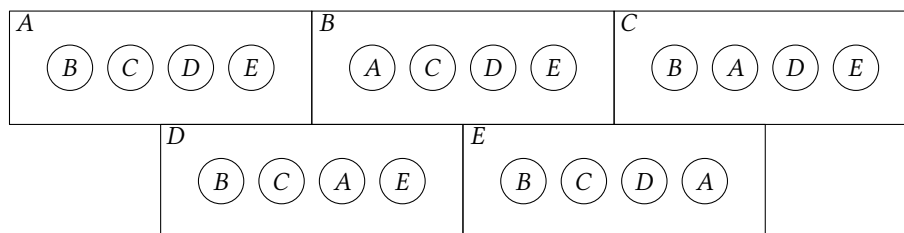
**V.2.44** For the octahedron, the path  $\langle v_0, v_1, v_5, v_4, v_3, v_2, v_0 \rangle$  will do. For the icosahedron, one path is  $\langle v_0, v_{10}, v_{11}, v_8, v_9, v_6, v_7, v_2, v_3, v_4, v_5, v_1, v_0 \rangle$ .

**V.2.45** This is the natural one.



**V.2.46**

(A) This map shows five countries. They are not contiguous; for instance, country A has five separate components.



Because of the portion in the upper left, country A must be a different color than any of the others. The other portions similarly give that all the other countries are different colors than each other. In total there are five countries, each a different color than all the others.

(B) Make a circle and split it into five wedges.

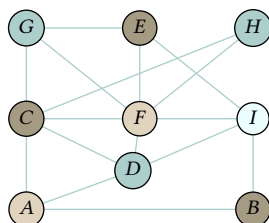
**V.2.47** The answer is 'yes' since  $22 + 986 + 165 = 1173$ .

**V.2.48**

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Weight	Value	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	Weight	Value
N	N	N	N	N	0	0	Y	N	N	N	N	21	50
N	N	N	N	Y	19	40	Y	N	N	N	Y	40	90
N	N	N	Y	N	42	44	Y	N	N	Y	N	63	94
N	N	N	Y	Y	61	84	Y	N	N	Y	Y	82	134
N	N	Y	N	N	49	34	Y	N	Y	N	N	70	84
N	N	Y	N	Y	68	74	Y	N	Y	N	Y	89	124
N	N	Y	Y	N	91	78	Y	N	Y	Y	N	112	128
N	N	Y	Y	Y	110	118	Y	N	Y	Y	Y	131	168
N	Y	N	N	N	33	48	Y	Y	N	N	N	54	98
N	Y	N	N	Y	52	88	Y	Y	N	N	Y	73	138
N	Y	N	Y	N	75	92	Y	Y	N	Y	N	96	142
N	Y	N	Y	Y	94	132	Y	Y	N	Y	Y	115	182
N	Y	Y	N	N	82	82	Y	Y	Y	N	N	103	132
N	Y	Y	N	Y	101	122	Y	Y	Y	N	Y	122	172
N	Y	Y	Y	N	124	126	Y	Y	Y	Y	N	145	176
N	Y	Y	Y	Y	143	166	Y	Y	Y	Y	Y	164	216

FIGURE 81, FOR QUESTION V.2.50: Output of a brute force script to show that there is no solution of an instance of the Knapsack problem.

- (A) AT&T connects to Citigroup, which connects to Ford Motor.  
 (B) Haliburton connects to Citigroup, which connects to Ford Motor.  
 (C) There is no connection between Caterpillar and Ford Motor.  
 (D) Also no connection.
- V.2.49** (A) 2 (B) 2 (C) 0 (D) 4 (I was only in the deleted scenes. But it is all just fun.)
- V.2.50** Figure 81 on page 154 shows the result of writing a small script.
- V.2.51** A 269-269 tie is easy to concoct. For example, the electoral votes of CA, TX, FL, NY, IL, PA, OH, GA, MI, NC, and VA add to 269.
- V.2.52** We can do these by eye.  
 (A) The shortest is  $q_2$  to  $q_1$  to  $q_7$ , which has length 14.  
 (B) The shortest is  $q_0$  to  $q_3$  to  $q_4$  to  $q_8$ , of length 23.  
 (C) There is no path between these two vertices.
- V.2.53** Figure 82 on page 155 is a check of every possibility.
- V.2.54** A 3-clique is a triangle. A 2-clique is an edge.
- V.2.55** A  $k$ -clique has  $k(k-1)/2$  edges, one for each combination of two vertices.
- V.2.56**  
 (A) Suppose that a three-coloring suffices. Then the three vertices  $A$ ,  $C$ , and  $D$  form a triangle, so they must be different colors. Since  $CDF$  is also a triangle, we conclude that  $A$  and  $F$  are the same colors. So far we have:  $\mathcal{C}_0 = \{A, F\}$ ,  $\mathcal{C}_1 = \{C\}$ ,  $\mathcal{C}_2 = \{D\}$ .  
 Next, consider triangle  $DFI$ . It's presence gives that  $C$  and  $I$  are the same colors. So we have  $\mathcal{C}_0 = \{A, F\}$ ,  $\mathcal{C}_1 = \{C, I\}$ ,  $\mathcal{C}_2 = \{D\}$ . Then triangle  $EFI$  leads to  $\mathcal{C}_0 = \{A, F\}$ ,  $\mathcal{C}_1 = \{C, I\}$ ,  $\mathcal{C}_2 = \{D, E\}$ .  
 Finally, triangle  $CFG$  gives that  $D$  and  $G$  are the same color, giving  $\mathcal{C}_0 = \{A, F\}$ ,  $\mathcal{C}_1 = \{C, I\}$ ,  $\mathcal{C}_2 = \{D, E, G\}$ . But this contradicts the presence of triangle  $GEF$ .  
 (B) This is a four-coloring.



21	33	49	42	19	Total	21	33	49	42	19	Total
N	N	N	N	N	0	Y	N	N	N	N	21
N	N	N	N	Y	19	Y	N	N	N	Y	40
N	N	N	Y	N	42	Y	N	N	Y	N	63
N	N	N	Y	Y	61	Y	N	N	Y	Y	82
N	N	Y	N	N	49	Y	N	Y	N	N	70
N	N	Y	N	Y	68	Y	N	Y	N	Y	89
N	N	Y	Y	N	91	Y	N	Y	Y	N	112
N	N	Y	Y	Y	110	Y	N	Y	Y	Y	131
N	Y	N	N	N	33	Y	Y	N	N	N	54
N	Y	N	N	Y	52	Y	Y	N	N	Y	73
N	Y	N	Y	N	75	Y	Y	N	Y	N	96
N	Y	N	Y	Y	94	Y	Y	N	Y	Y	115
N	Y	Y	N	N	82	Y	Y	Y	N	N	103
N	Y	Y	N	Y	101	Y	Y	Y	N	Y	122
N	Y	Y	Y	N	124	Y	Y	Y	Y	N	145
N	Y	Y	Y	Y	143	Y	Y	Y	Y	Y	164

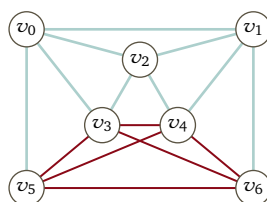
FIGURE 82, FOR QUESTION V.2.53: All possibilities in the Subset Sum instance.

Combination of five vertices					A missing edge	Combination of five vertices					A missing edge
$v_0$	$v_1$	$v_2$	$v_3$	$v_4$	$v_1v_3$	$v_0$	$v_2$	$v_3$	$v_4$	$v_6$	$v_0v_4$
$v_0$	$v_1$	$v_2$	$v_3$	$v_5$	$v_2v_5$	$v_0$	$v_2$	$v_3$	$v_5$	$v_6$	$v_0v_6$
$v_0$	$v_1$	$v_2$	$v_3$	$v_6$	$v_2v_6$	$v_0$	$v_2$	$v_4$	$v_5$	$v_6$	$v_0v_4$
$v_0$	$v_1$	$v_2$	$v_4$	$v_5$	$v_0v_4$	$v_0$	$v_3$	$v_4$	$v_5$	$v_6$	$v_0v_4$
$v_0$	$v_1$	$v_2$	$v_4$	$v_6$	$v_0v_6$	$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_1v_3$
$v_0$	$v_1$	$v_2$	$v_5$	$v_6$	$v_0v_6$	$v_1$	$v_2$	$v_3$	$v_4$	$v_6$	$v_1v_3$
$v_0$	$v_1$	$v_3$	$v_4$	$v_5$	$v_0v_4$	$v_1$	$v_2$	$v_3$	$v_5$	$v_6$	$v_1v_3$
$v_0$	$v_1$	$v_3$	$v_4$	$v_6$	$v_0v_6$	$v_1$	$v_2$	$v_4$	$v_5$	$v_6$	$v_1v_5$
$v_0$	$v_1$	$v_3$	$v_5$	$v_6$	$v_0v_6$	$v_1$	$v_3$	$v_4$	$v_5$	$v_6$	$v_1v_3$
$v_0$	$v_1$	$v_4$	$v_5$	$v_6$	$v_0v_6$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$	$v_2v_5$
$v_0$	$v_2$	$v_3$	$v_4$	$v_5$	$v_0v_4$						

FIGURE 83, FOR QUESTION V.2.59: A check of all possible 5-cliques.

**V.2.57** A finite set of propositional logic statements  $\{S_0, \dots, S_{k-1}\}$  is satisfiable if and only if the single statement  $S_0 \wedge S_1 \dots \wedge S_{k-1}$  is satisfiable.

**V.2.59** It has a 4-clique, and thus it necessarily has a 3-clique.



But it has no 5-clique, as checked by exhaustion in Figure 83 on page 155.

**V.2.60**

- (A) A vertex cover with  $k = 2$  elements is  $S = \{q_1, q_3\}$ . An independent set with  $\hat{k} = 4$  elements is  $\hat{S} = \{q_0, q_2, q_4, q_5\}$ .
- (B) A vertex cover with  $k = 3$  elements is  $S = \{q_1, q_2, q_4\}$ . An independent set with  $\hat{k} = 3$  elements is  $\hat{S} = \{q_0, q_3, q_5\}$ .
- (C) A vertex cover with  $k = 4$  elements is  $S = \{q_2, q_5, q_8, q_9\}$ . An independent set with  $\hat{k} = 6$  elements is

Year	Winner	Other	Tallest?	Year	Winner	Other	Tallest?
1900	W McKinley	W J Bryan	N	1960	J F Kennedy	R Nixon	Y
1904	T Roosevelt	A B Parker	Y	1964	L B Johnson	B Goldwater	Y
1908	W H Taft	W J Bryan	Y	1968	R Nixon	H Humphrey	Y
1912	W Wilson	-2 others-	N	1972	R Nixon	G McGovern	N
1916	W Wilson	C E Hughes	Y	1976	J Carter	G Ford	N
1920	W G Harding	J M Cox	Y	1980	R Reagan	J Carter	Y
1924	C Coolidge	J W Davis	N	1984	R Reagan	W Mondale	Y
1928	H Hoover	A Smith	Y	1990	G H W Bush	M Dukakis	Y
1932	F D Roosevelt	H Hoover	Y	1992	W Clinton	G H W Bush	-same-
1936	F D Roosevelt	A Landon	Y	1996	W Clinton	R Dole	Y
1940	F D Roosevelt	W Willkie	N	2000	G W Bush	A Gore	N
1944	F D Roosevelt	T Dewey	Y	2004	G W Bush	J Kerry	N
1948	H S Truman	T Dewey	Y	2008	B Obama	J McCain	Y
1952	D D Eisenhower	A Stevenson II	Y	2012	B Obama	M Romney	N
1956	D D Eisenhower	A Stevenson II	Y	2016	D Trump	H Clinton	Y

FIGURE 84, FOR QUESTION V.3.9: Height comparison of US Presidential winner and main opponent, since 1900.

$$\hat{S} = \{q_0, q_1, q_3, q_4, q_6, q_7\}.$$

- (D) For any edge, if it has at least one endpoint in  $S$  then it has at most one endpoint in the complement  $\hat{S} = \mathcal{N} - S$ . Conversely, If an edge has at most one endpoint in  $\hat{S}$  then it has at least one endpoint in the complement  $S = \mathcal{N} - \hat{S}$ .

**V.2.61** The Three-dimensional Matching problem requires that the three sets, the instructors, the courses, and the time slots, all have the same size. In this instance, all three have five elements.

A suitable matching is  $\hat{M} = \{ \langle A, 2, \epsilon \rangle, \langle B, 0, \alpha \rangle, \langle C, 3, \gamma \rangle, \langle D, 1, \beta \rangle, \langle E, 4, \delta \rangle \}$ .

**V.2.62**

- (A) The set  $M = X \times Y \times Z$  contains these twenty seven triples  $\langle x, y, z \rangle$ .

$z = a$			$z = d$			$z = e$		
$\langle a, b, a \rangle$	$\langle a, c, a \rangle$	$\langle a, d, a \rangle$	$\langle a, b, d \rangle$	$\langle a, c, d \rangle$	$\langle a, d, d \rangle$	$\langle a, b, e \rangle$	$\langle a, c, e \rangle$	$\langle a, d, e \rangle$
$\langle b, b, a \rangle$	$\langle b, c, a \rangle$	$\langle b, d, a \rangle$	$\langle b, b, d \rangle$	$\langle b, c, d \rangle$	$\langle b, d, d \rangle$	$\langle b, b, e \rangle$	$\langle b, c, e \rangle$	$\langle b, d, e \rangle$
$\langle c, b, a \rangle$	$\langle c, c, a \rangle$	$\langle c, d, a \rangle$	$\langle c, b, d \rangle$	$\langle c, c, d \rangle$	$\langle c, d, d \rangle$	$\langle c, b, e \rangle$	$\langle c, c, e \rangle$	$\langle c, d, e \rangle$

- (B) There are lots of correct answers. One is  $\hat{M} = \{ \langle a, b, a \rangle, \langle b, c, d \rangle, \langle c, d, e \rangle \}$ .

**V.2** No, it cannot be done in three steps. The longest path is to  $v_{10}$ , of three edges. Even if we start on that path by broadcasting to  $v_4$  at the first stage,  $v_0$  still must initiate broadcasts to  $v_6$  and  $v_3$ , each of which will take two stages. But  $v_0$  can only initiate one at a time.

**V.3.9** *Note:* the meanings give here are common usage. But because these terms are not formally defined, different authors may use them in different ways.

- (A) A heuristic is a method that, while it may be used in practice, is not guaranteed to be optimal, perfect, logical, or even rational. An algorithm for a problem is guaranteed to solve the problem.

An example of a heuristic is based on the data in Figure 84 on page 156. In the contests for US President since 1900, the taller candidate has won 20 out of 29 times, for a sample proportion of 0.69. The chance of such a result by happenstance is less than 0.02. You could decide to bet in the next election on the taller candidate. This isn't logical, but it is a method and it is used in practice.

- (B) Pseudocode describes solving a problem at a level of detail that that lies between a program than an algorithm. It typically includes at least a sketch of looping and other control structures, but not syntactical detail such as declaration of variables. This is pseudocode for the recursive depth-first traversal of a binary tree.

```

process_subtree(v):
  process_subtree(v's left child)
  process_vertex(v)
  process_subtree(v's right child)

```

- (C) A Turing machine is more like a program than an algorithm, in that it is an implementation of an algorithm on a specific hardware platform.
- (D) A flowchart is graphical. It may include a range of detail, so that in representing an algorithm, it may contain a sketch of control structures but it may also omit those details.
- (E) A process is an instance of a program that has been loaded into a machine for execution. For instance, if we compile a program to solve the **Vertex to Vertex Path** problem and run it on hardware then we have an executing process. Some processes, such as an operating system shell or a web server, are designed to run forever, while typically the term ‘algorithm’ is used for something that is guaranteed to halt. Thus, a web server may bundle together many routines that implement algorithms, each of which halts, but the larger bundle loops through the subroutines forever.

**V.3.10** A solution is typically an algorithm or computation that acts as the characteristic function of the set.

**V.3.11** A decision problem is any one that ends in a ‘yes’ or ‘no’ answer. A language decision problem is a specialization of that, to ‘yes’ or ‘no’ answers about membership in the given language.

**V.3.12**

- (A) The multiplication is easy:  $(1/0.01)^2 \cdot 21 \cdot 30 = 6.30 \times 10^6$ .
- (B) The answer is  $\lceil \lg(1\,000\,000) \rceil$ . If all you have is a pocket calculator with a natural log function button, then you can get  $6 \ln(10)/\ln(2) \approx 19.93$ , so the answer is 20.
- (C) Multiply  $(6.30 \times 10^6) \cdot 20 = 1.26 \times 10^8$ . That’s less than 16 megabytes, about eight books from Project Gutenberg.

**V.3.13** One is the smell of an baby’s head. They have a great smell. But we don’t compute it.

**V.3.14** Yes, two programs that implement the same algorithm must compute the same function. However, the converse does not hold; two programs computing the same function need not implement the same algorithm. For instance, two programs may both sort input strings but one uses Bubble Sort while the other uses Quick Sort.

**V.3.16** These algorithms mark that input is accepted or not by printing 1 or  $\emptyset$ .

- (A) Input the sequence and unpack it to get  $n, m \in \mathbb{N}$ . Then adds the two numbers. If the sum is both square and one greater than a prime then print 1, and otherwise print  $\emptyset$ .
- (B) If the input is a single  $\emptyset$ , or ends in two  $\emptyset$ ’s, then print 1. Otherwise print  $\emptyset$ .
- (C) Scan the input string, keeping a running count of the number of  $\emptyset$ ’s and 1’s. At the end, if there are more 1’s then print 1, otherwise print  $\emptyset$ .
- (D) Read and save the input string. Then, compare the characters starting from the back with those starting from the front. If each matches (or the string is empty) then print 1. Otherwise print  $\emptyset$ .

**V.3.17**

- (A) The empty language is  $\mathcal{L} = \emptyset = \{ \}$ . Its characteristic function is  $\mathbb{1}_{\mathcal{L}}(n) = 0$ . The algorithm to solve this language decision problem is to, for all input, return 0.
- (B) This language contains two strings,  $\mathbb{B} = \{ \emptyset, 1 \}$  (technically, these are characters, but we don’t distinguish between characters and length-1 strings). The characteristic function is this.

$$\mathbb{1}_{\mathbb{B}}(\sigma) = \begin{cases} 1 & \text{– if } \sigma = \emptyset \text{ or } \sigma = 1 \\ 0 & \text{– otherwise} \end{cases}$$

The algorithm that implements that function is easy.

- (C) The language  $\mathbb{B}^*$  contains all bitstrings. To solve the decision problem for this language, the algorithm returns 1 on all inputs.

**V.3.18**

- (A) For the algorithm, fix a Finite State machine  $\mathcal{M}$  that recognizes the language described by  $a^*ba^*$ . Now, read the input  $\sigma$  and simulate running  $\mathcal{M}$  on that input. This will terminate in a finite number of steps. If

$\mathcal{M}$  ends in an accepting state then print 1 (or yes, or some other value signaling acceptance), otherwise print 0.

- (B) The grammar defines the language  $\mathcal{L} = \{a^n b^m \mid n, m \in \mathbb{N}\}$ . The algorithm for that language is simple. Input the string and if all of the b's come after all of the a's then output 1 (or 'yes', or whatever indicates acceptance). Otherwise, output 0.

**V.3.19** First observe that there is an easy algorithm that can, given a machine  $\mathcal{M}$  and a state  $q$  in that machine, find the set of states reachable from  $q$ . The algorithm proceeds in steps. At step 1, initialize by taking  $R_1$  to be the set of states that are reachable from  $q$  in one transition. Step  $i + 1$  begins with the set of states that are reachable from  $q$  with  $i$ -many transitions or fewer. In that step, for each state in  $R_i$ , find if any states reachable from it in one transition are not in  $R_i$ . Make the set  $R_{i+1}$  by adding them all to  $R_i$ . If there are no such new states then the algorithm stops. Obviously it must stop at some point because the number of states in the machine is finite.

- (A) The language is empty if and only if from the start state, no accepting state is reachable.  
 (B) The language is infinite if and only if there is a state  $q$  that can be reached from the initial state such that  $q$  can be reached from itself by a nonempty string, and some accepting state can be reached from  $q$ .  
 (C) Given  $\mathcal{M}$ , obtain a new machine  $\hat{\mathcal{M}}$  that recognizes the complement of  $\mathcal{L}(\mathcal{M})$  by copying  $\mathcal{M}$ , except setting each accepting state of  $\mathcal{M}$  to be non-accepting in  $\hat{\mathcal{M}}$ , and setting each non-accepting state of  $\mathcal{M}$  to be accepting in  $\hat{\mathcal{M}}$ . Now,  $\mathcal{L}(\mathcal{M})$  is empty if and only if  $\mathcal{L}(\hat{\mathcal{M}})$  is empty.

**V.3.20**

- (A) See if the input string starts with a 1 bit.  
 (B) See if the input string has length less than or equal to ten. If the length equals ten then there are some cases to do but that doesn't change that the algorithm is  $\mathcal{O}(1)$ .

**V.3.21** This is a search problem, since we need only produce a single bridge.

**V.3.22** As remarked in the section, a decision problem involves finding a function—a Boolean function. So in that sense, multiple answers are possible.

- (A) This is a yes-or-no question, so it is a decision problem.  
 (B) Decision problem  
 (C) Function problem.  
 (D) Decision problem.  
 (E) Search problem.  
 (F) Decision problem.  
 (G) Language decision problem, for  $\mathcal{L} = \{\sigma \in \mathbb{B}^* \mid \sigma \text{ represents a number that in decimal has only odd digits}\}$ .

**V.3.23**

- (A) Decision problem  
 (B) Search problem  
 (C) Function problem  
 (D) Function problem  
 (E) Decision problem

**V.3.24**

- (A) A language suited to Turing machines is  $\mathcal{L}_0 = \{1^n \mid n \in \mathbb{N} \text{ is a square}\}$ . A language suited to more everyday machines is  $\mathcal{L}_1 = \{d \in \{0, \dots, 9\}^* \mid d \text{ represents a perfect square in base 10}\}$ .  
 (B) One language is the set of triples  $\langle x, y, z \rangle \in \{0, \dots, 9\}^*$  such that  $x, y, z$  represent integers in base 10 where the squares of the first two add to the square of the third.  
 (C) A language is  $\{\sigma \in \mathbb{B}^* \mid \sigma \text{ represents a graph } \mathcal{G} \text{ with an even number of edges}\}$ .  
 (D) The language  $\{\sigma \in \mathbb{B}^* \mid \sigma \text{ represents a pair } \langle \mathcal{G}, p \rangle \text{ where } p \text{ is a path with a repeated vertex}\}$  will do.

**V.3.25**

- (A)  $\{n \in \mathbb{N} \mid \text{the factors of } n \text{ sum to more than } 2n\}$ .  
 (B)  $\{\langle \mathcal{P}, i \rangle \mid \text{machine } \mathcal{P} \text{ on input } i \text{ halts in less than ten steps}\}$   
 (C)  $\{E \mid \text{the logic expression } E \text{ can be satisfied by three distinct assignments}\}$   
 (D)  $\{\langle \mathcal{G}, B \rangle \mid \text{for all } v_0, v_1 \in \mathcal{G} \text{ there is a path of cost less than } B\}$

**V.3.26**

- (A) The **Graph Colorability** problem is stated as a decision problem. Imagine that we had an algorithm to recognize membership in this language.

$$\mathcal{L} = \{ \sigma \in \mathbb{B}^* \mid \sigma \text{ represents } \langle \mathcal{G}, k \rangle, \text{ where } \mathcal{G} \text{ is } k\text{-colorable} \}$$

Then, given an graph and an integer  $k$ , we could use that algorithm to decide whether the graph is  $k$ -colorable. Convert the graph and integer to a string representing the pair, and use the algorithm to decide if that pair is in the language.

- (B) Consider this.

$$\mathcal{L} = \{ \sigma \in \mathbb{B}^* \mid \sigma \text{ represents a graph with a path that traverses each edge exactly once} \}$$

The **Euler Circuit** problem is given as a function problem. If we had an algorithm to recognize membership in the language then, given an graph, we could use that algorithm to decide whether the graph has a circuit: we just convert the graph to its representation and apply the algorithm. If we know there is no Euler circuit then we are done. If we know there is an Euler circuit then a search will find it.

- (c) The **Shortest Path** problem is given as a function problem. If we had an algorithm to recognize membership in the language

$$\mathcal{L} = \{ \sigma \in \mathbb{B}^* \mid \sigma \text{ represents } \langle \mathcal{G}, v_0, v_1, p \rangle, \text{ where } p \text{ is a minimal-length path between the vertices} \}$$

then, given an graph and two vertices, we could find the minimal path by using that algorithm to test all possible paths from one vertex to the other for minimality by seeing if the representation of the quad is in the language.

**V.3.27** It is a matter of deciding membership in this language.  $\{ \langle \mathcal{P}_e, e \rangle \mid e \in \mathbb{N} \}$ .

**V.3.28** Take  $B \in \mathbb{N}$  as a bound and consider this family.

$$\mathcal{L}_B = \{ \langle \mathcal{G}, v_0, v_1 \rangle \mid \text{There is a path from } v_0 \text{ to } v_1 \text{ of distance less than or equal to } B \}$$

Given a graph and two vertices we can solve the **Shortest Path** problem for it by searching through  $\mathcal{L}_1, \mathcal{L}_2, \dots$  until we find the smallest bound  $B$  for which our triple is a member of the language  $\mathcal{L}_B$ .

**V.3.29**

- (A) Take  $B \in \mathbb{N}$  to be a bound and suppose that we can solve this language decision problem, where  $\mathcal{B}$  is a game board.

$$\mathcal{L}_B = \{ \mathcal{B} \mid \mathcal{B} \text{ can be solved in fewer than } B \text{ slide moves} \}$$

Given a board, to find the minimum number of moves, iterate through  $B = 0, B = 1$ , etc., looking to see if the given board is a member of  $\mathcal{L}_B$ . Stop at the first  $B$  where it is in the language. That is the minimum.

- (B) Take  $B \in \mathbb{N}$  as a bound. Suppose that we have an algorithm to solve this language decision problem, where  $\mathcal{R}$  is a Rubik's cube arrangement.

$$\mathcal{L}_B = \{ \mathcal{R} \mid \mathcal{R} \text{ can be solved in fewer than } B \text{ cube moves} \}$$

Given a cube, to find its minimum number of moves, iterate through  $B = 0, B = 1$ , etc., until a  $B$  appears so that the given cube is in  $\mathcal{L}_B$ . That is the minimum number of moves.

- (c) Take  $B \in \mathbb{N}$ . Suppose that we have an algorithm to solve this language decision problem, where  $S$  is a specification a car's assembly jobs.

$$\mathcal{L}_B = \{ S \mid S \text{ can be accomplished in less time than } B \}$$

Given a specification, find the minimum time by iterating through  $B = 0, B = 1$ , etc., until a  $B$  appears so that the give specification is in  $\mathcal{L}_B$ .

**V.3.30** A function version would input a graph and output a circuit (if there is no circuit then it would output something like the string None).

An optimization version would return a circuit that is best by some criterion. For instance, in a weighted graph it would return the circuit of least weight; this is the **Traveling Salesman** problem.

**V.3.31**

- (A) Search problem
- (B) Decision problem
- (C) Function problem
- (D) Optimization problem

A language suitable for an associated language problem is  $\{M \mid M \text{ has a } 3 \times 3 \text{ invertible submatrix}\}$ .

**V.3.32**

- (A) Given a triple  $\langle n_0, n_1, p \rangle \in \mathbb{N}^3$ , decide if  $p = n_0 \cdot n_1$ .
- (B) Given a triple  $\langle \mathcal{G}, v_0, v_1 \rangle$ , where  $\mathcal{G}$  is a weighted graph and  $v_0 \neq v_1$  are vertices in that graph, decide if  $v_1$  is a vertex that is as near as possible to  $v_0$ ,

**V.3.33**

- (A) Use the language  $\{ \langle s_0, \dots, s_{n-1} \rangle \in \mathbb{Q}^n \mid a_{i,0}s_0 + \dots + a_{i,n-1}s_{n-1} \leq b_i \text{ for all } i \}$ .
- (B) It is a search problem as it is stated. We need only find one feasible sequence, if there is one.
- (C) The function could produce one feasible sequence, if one exists.
- (D) To make it an optimization function, the natural thing is to optimize a linear function  $P(x_0, \dots, x_{n-1}) = c_0x_0 + \dots + c_{n-1}x_{n-1}$ , for some  $c_0, \dots, c_{n-1} \in \mathbb{Q}$ .

**V.3.34** A set with one vertex is independent, as is the empty set. So every graph has a subset of its vertices that is independent. Because of that, these answers at least state a version for some number of vertices  $k \in \mathbb{N}$ .

- (A) Given a graph, decide if it has an independent set with  $k$  vertices.
- (B) Use the language  $\{ \mathcal{G} \mid \mathcal{G} \text{ has } k \text{ vertices that do not share an edge} \}$ .
- (C) Given a graph  $G$ , find an independent set with  $k$ -many vertices. (There may be more than one, or none, but we only look for one.)
- (D) Given a graph, return an independent set of size  $k$  or the string None.
- (E) Given a graph, return an independent set that is maximal, in that it is as large as possible for the graph.

**V.3.35** Every list of numbers has a maximum, but finding it appears to mean at least looking through the list.

**V.3.37** Imagine that we have a routine `shortest_path` that takes in a weighted graph and two vertices, and returns the shortest path between them. That also solves the decision problem of whether there is any path at all.

If the graph is not weighted, convert it to a weighted graph by taking each edge to be of weight 1.

**V.3.38** By taking the numbers in the set  $S = \{s_0, \dots, s_{n-1}\}$  one at a time:  $S_0 = \{s_0\}$ , then  $S_1 = \{s_0, s_1\}, \dots$ , you can figure out from the decision problem solver what is the largest number  $s_j$  in the sum. Subtract that from the goal and then iterate using  $S - \{s_j\}$ .

**V.3.39** Check all pairs of vertices.

**V.4.12** True. Such problems have solution algorithms that are basically a fixed number of if ... then ... branches, so there is a maximum number of steps that any input can trigger. Thus, the solution algorithm is  $\mathcal{O}(1)$ .

**V.4.13** Algorithms are not in P; rather, problems are in P. Specifically, a problem is in P if from among all the algorithms that solves it, at least one runs in polynomial time, is  $\mathcal{O}(n^c)$  for some constant  $c$ . So, better is to say something like, "I've got a problem whose solution algorithm takes polytime." (It is absolutely right that we are not too careful to distinguish between problems, meaning language decision problems, and languages. But problems and algorithms are very different.)

**V.4.14** An order of growth is a set of functions. A complexity class is a set of problems, language decision problems.

**V.4.15** Your friend is observing that the circuit representation may have length exponential in the circuit depth, since an  $r$ -deep circuit may have  $2^r$  many vertices. But the definition of polynomial time is that the number of steps taken is a polynomial function of the size of the input representation. If the representation is very large then that only means that the polynomial receives a large input. That is, we are not comparing the time taken to the depth of the circuit, but rather to the size of the representation.

**V.4.16** The student's machine accepts all strings. The definition requires a machine that only accept strings in the language.

**V.4.17** True.

**V.4.18** True, although not by the same machine. Let  $\mathcal{L}$  be computed by  $\mathcal{P}$ , so that the machine has this

input-output behavior.

$$\phi_{\mathcal{P}}(\sigma) = \mathbb{1}_{\mathcal{L}}(\sigma) = \begin{cases} 1 & \text{- if } \sigma \in \mathcal{L} \\ 0 & \text{- otherwise} \end{cases}$$

Modify the machine by interchanging the output 0's and 1's, to get a new machine  $\hat{\mathcal{P}}$  with the complementary behavior.

$$\phi_{\hat{\mathcal{P}}}(\sigma) = \mathbb{1}_{\mathcal{L}^c}(\sigma) = \begin{cases} 0 & \text{- if } \sigma \in \mathcal{L} \\ 1 & \text{- otherwise} \end{cases}$$

**V.4.19** We must produce an algorithm that runs in polytime. Given the input  $\sigma$ , find its length. If the length is not divisible by three then return 0. Otherwise, find the substring  $\tau$  that is the first third of  $\sigma$ , and check whether  $\tau \wedge \tau \wedge \tau = \sigma$ . If so, return 1, and if not, return 0.

**V.4.20** The algorithm is straightforward. Given  $\sigma$ , for  $i \in \{0, 1, \dots, |\sigma| - 1\}$ , read the characters  $\sigma[i]$  and compare them to the corresponding character read from the back,  $\sigma[|\sigma| - 1 - i]$ . (You can stop once you get through the halfway point,  $i = \lfloor |\sigma|/2 \rfloor$ .) If each comparison gives equality then return 1, else return 0. This algorithm is polynomial since it is basically a loop.

**V.4.21** In each case we must produce an algorithm that runs in polynomial time.

- (A) Given  $\sigma$ , decide if its length is divisible by three. If not, return 0. If it is divisible by three, set  $\tau$  to be the substring that is the first third of  $\sigma$ . Then just check, by moving through the string once, whether  $\sigma = \tau^3$ .
- (B) Stated as a language decision problem, we have  $\{i \in \mathbb{N} \mid \mathcal{P}_i \text{ halts on the empty string in 10 steps}\}$ . There is some overhead in going from the input  $i$  to the Turing machine  $\mathcal{P}_i$ , simulating it with a universal Turing machine, etc. But putting aside that overhead, then no matter what is the input  $i$ , we then just run for ten steps. So it is an element of  $\mathbf{P}$ .

**V.4.22**

- (A) Because the size of the clique is fixed. In the Clique problem, the size of the clique is a parameter, so that we are given a pair  $\langle \mathcal{G}, B \rangle$  and asked if the graph has a  $B$ -sized clique.
- (B) Given  $\mathcal{G}$ , enumerate all triples  $\langle v_0, v_1, v_2 \rangle \in \mathcal{N}^3$  and check whether all three of  $v_0v_1, v_0v_2$ , and  $v_1v_2$  are edges, members of  $\mathcal{E}$ . This takes polynomial time. Specifically, it is  $\mathcal{O}(|\mathcal{N}|^3 \cdot |\mathcal{E}|)$ . Because  $|\mathcal{E}| \leq |\mathcal{N}|^2$ , the algorithm is  $\mathcal{O}(|\mathcal{N}|^5)$ .

**V.4.23**

- (A) To determine whether the string is in alphabetical order, a single pass through it suffices. We can assume that each comparison takes constant time. So, overall this one-pass algorithm takes polytime.
- (B) The elementary school algorithm of adding in columns from the least significant digit (or bit) and then off to the left shows that the runtime of addition is proportional to the number of digits in the larger of the two input numbers (or the number of bits, which is  $\lg(10)$  times the number of digits). We can do addition of individual digits (or bits) with a lookup table, so that step happens in constant time. Therefore, overall, addition takes time proportional to the length of the input representation, which is polytime.
- (C) The naive algorithm for matrix multiplication gives a triply nested loop, and so is  $\mathcal{O}(n^3)$  where  $n$  is the largest of the number of rows or columns in either of  $A$  or  $B$ .
- (D) As mentioned in Problem 2.38, determining whether a number is composite or prime is known to be possible in polytime.
- (E) We can give an algorithm that runs in time polynomial in the length of the graph description. Fix the set  $S_0 = \{v_0\}$ . Next, follow each edge leading out of  $v_0$ , to get the set of vertices that are connected to  $v_0$  in at most one step,  $S_1 = S_0 \cup \{v \mid \langle v_0, v \rangle \in \mathcal{E}\}$ . After that, for all the vertices in  $S_1$ , follow all the edges to get the set  $S_2 = S_1 \cup \{v \mid \langle \hat{v}, v \rangle \in \mathcal{E} \text{ where } \hat{v} \in S_1\}$  of all vertices that are connected to  $v$  in at most two steps. This process must stop at some point, with  $S_{i-1} = S_i$ , because the graph is finite. Then, either  $v_1 \in S_i$  or not, which determines whether there is a path from  $v_0$  to  $v_1$ . The number of steps in this algorithm,  $i$ , is bounded by the number of vertices in the graph, so the algorithm is polynomial in the size of the graph representation.

## V.4.24

- (A) By Dijkstra's algorithm, the Shortest Path problem can be done in polynomial time. So the problem is in P.  
 (B) Knapsack is believed to be quite hard. No one knows of a polynomial algorithm.  
 (C) Euler Path is easy; we just need to check whether each node has even degree. There are algorithms that are linear.  
 (D) No one knows of a polynomial algorithm for Hamiltonian Circuit.

**V.4.25** To show it is in P we must produce a solution algorithm for it that runs in polytime. For that, start by picking a vertex,  $v$ . Let the set  $S_0$  equal  $\{v\}$ . Follow each edge leading out of  $v$  to get all of the vertices that are connected to  $v$  in at most one step, and call that set  $S_1 \supseteq S_0$ . Iterate: for all the vertices in  $S_1$ , follow all the edges to get the set  $S_2$  of all of the vertices that are connected to  $v$  in at most two steps. Clearly this process must stop at some point, with  $S_{i-1} = S_i$ , because the graph is finite.

This algorithm takes a number of steps,  $i$ , that is bounded by the number of vertices in the graph, and so the algorithm is polynomial in the size of the graph.

## V.4.26

- (A) Our definition of complexity class is completely general—it is a collection of languages. So under the definition that we gave, the collection of regular languages is a complexity class.  
 As to the additional condition, a language is regular if it is accepted by a Finite State machine. So the regular languages are computed by a device. Further, a Finite State machine meets the constraint that it runs in a number of steps that is equal to the length of its input,  $\sigma$ .  
 (B) The question notes that we can define regular languages using a type of Turing machine. What remains is to show that each regular language is computable in polytime. As in the prior item, the fact that such a machine consumes its input and never writes to the tape means that it runs in linear time,  $O(|\sigma|)$ .

## V.4.27

- (A) It is a complexity class because it is a collection of languages. In addition, each such language  $\mathcal{L}$  satisfies that there is a computable function  $\phi_e$  with  $\mathcal{L} = \{k \in \mathbb{N} \mid k = \phi_e(i) \text{ for some input } i \in \mathbb{N}\}$ .  
 (B) The resource specification is that there is no finite time bound.

**V.4.28** There are countably many Turing machines to use as deciders, so P is countable.

**V.4.29** No. Take  $\mathcal{L}_0 = \emptyset$  and  $\mathcal{L}_1 = \mathcal{P}(\mathbb{B}^*)$ . Because P is countable, as there are only countably many Turing machines to use as deciders, and the collection of all languages  $\mathcal{P}(\mathbb{B}^*)$  is uncountable, there are uncountably many languages with  $\mathcal{L}_0 \subseteq \mathcal{L} \subseteq \mathcal{L}_1$  that are not in P.

**V.4.30** No. Certainly we can state the Halting problem as a language decision problem, using the language  $\mathcal{L} = \{n \in \mathbb{N} \mid \phi_n(n) \downarrow\}$ . But to be a member of P, there must be a Turing machine that solves the problem (and that runs in polytime). No Turing machine solves the Halting problem, so it is not in P.

**V.4.31** A language is a member of P if there is an algorithm for it that runs in polytime. That is, there is a Turing machine that computes the characteristic function of the language in polytime. Clearly, existence of such a machine is equivalent to the existence of a machine with a designated acceptance state (and that runs in polytime). So the set of languages in P is a subset of the set of all decidable languages.

**V.4.32** Here is the desired input-output behavior for the circuit.

$b_2$	$b_1$	$b_0$	$b_2 + b_1 + b_0$	$b_2 + b_1 + b_0 \pmod{2}$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	2	1
1	0	0	1	0
1	0	1	2	1
1	1	0	2	1
1	1	1	3	0

This propositional logic formula gives the behavior in that table.

$$(\neg b_2 \wedge \neg b_1 \wedge \neg b_0) \vee (\neg b_2 \wedge b_1 \wedge b_0) \vee (b_2 \wedge \neg b_1 \wedge b_0) \vee (b_2 \wedge b_1 \wedge \neg b_0)$$

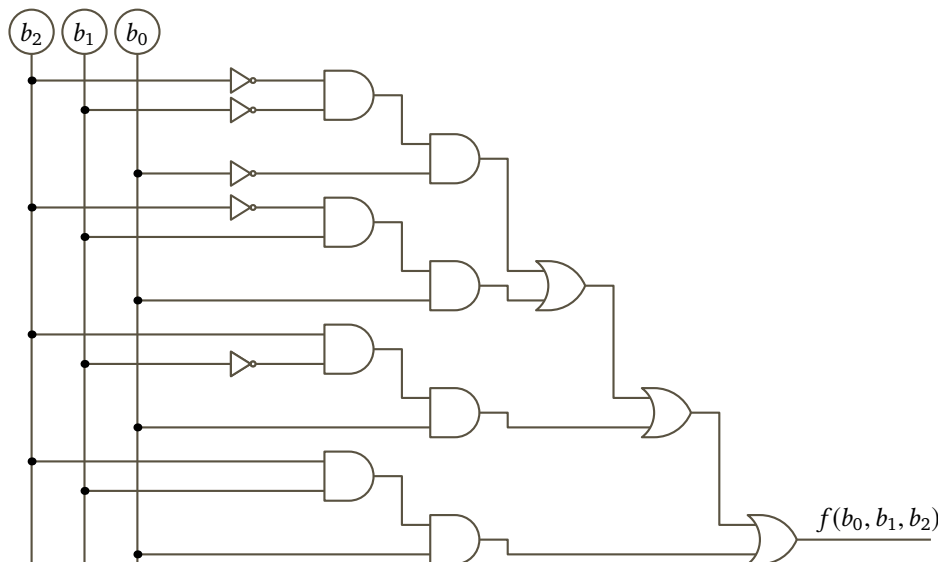


FIGURE 85, FOR QUESTION V.4.32: Initial circuit for the Circuit Evaluation problem.

Figure 85 on page 163 shows the circuit, using the standard notation of  $\sqcap$  for ‘and’ gates,  $\sqcup$  for ‘or’ gates, and  $\triangleright$  for ‘not’ gates. But this circuit does not perfectly match the problem since some gates do not have two inputs and one output. Figure 86 on page 164 modifies the prior diagram to draw the circuit as an acyclic directed graph, by folding the ‘not’ gates into the boolean binary functions. These are the Boolean functions used there.

$P$	$Q$	$\wedge$	$P$	$Q$	$\vee$	$P$	$Q$	$f_0$	$P$	$Q$	$f_1$	$P$	$Q$	$f_2$
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	0	0	1	1	0	1	0	0	1	0	0	1	1
1	0	0	1	0	1	1	0	0	1	0	1	1	0	0
1	1	1	1	1	1	1	1	0	1	1	0	1	1	0

So,  $f_0$  is equivalent to  $\neg P \wedge \neg Q$ , and  $f_1$  is equivalent to  $P \wedge \neg Q$ , and  $f_2$  is equivalent to  $\neg P \wedge Q$ .

**V.4.33** A complexity class is just a set, a collection of languages. So the union of two is also a complexity class. The same holds for the intersection, and for the complement (where the complement is taken inside the universe  $\mathcal{P}(\mathbb{B}^*)$ ).

**V.4.34** Let two languages  $\mathcal{L}_0, \mathcal{L}_1 \in \mathcal{P}(\mathbb{B}^*)$  be elements of  $\mathbf{P}$ . Then there are computable functions to determine membership in the two with runtimes  $\mathcal{O}(x^{n_0})$  and  $\mathcal{O}(x^{n_1})$ . The algorithm to compute membership in the union gets as input a string  $\sigma \in \mathbb{B}^*$ . It first checks whether the string is a member of the first language, and then checks whether it is a member of the second. Together, that takes a running time of  $\mathcal{O}(x^{\max(n_0, n_1)})$ , which means that the algorithm runs in polytime.

The argument for the union of any finite number of languages proceeds by induction and is straightforward.

**V.4.35** Assume that  $\mathcal{L} \in \mathbf{P}$ . Then there is a deterministic Turing machine  $\mathcal{P}$  that runs in polynomial time, and that accepts an input string  $\sigma \in \mathbb{B}^*$  if and only if  $\sigma \in \mathcal{L}$ . Restated, there is a computable function  $f: \mathbb{B}^* \rightarrow \mathbb{B}^*$  such that

$$f(\sigma) = \begin{cases} 0 & \text{if } \sigma \notin \mathcal{L} \\ 1 & \text{if } \sigma \in \mathcal{L} \end{cases}$$

and such that computing  $f$  is  $\mathcal{O}(p)$  for some polynomial  $p$ . (Specifically, what is  $\mathcal{O}(p)$  is the function  $t(n)$ , defined as the maximum runtime of  $f(\tau)$  over all  $\tau \in \mathbb{B}^*$  with  $n = |\tau|$ .)

Then clearly this function is also computable, and also in polytime:  $\bar{f}(\sigma) = 1 - f(\sigma)$ . Consequently,  $\mathcal{L}^c \in \mathbf{P}$ .

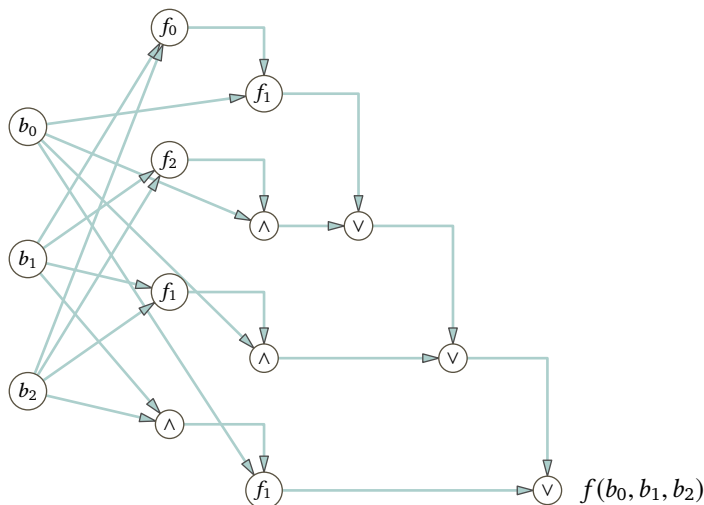


FIGURE 86, FOR QUESTION V.4.32: Directed acyclic graph for the Circuit Evaluation problem.

**V.4.36** Fix a language  $\mathcal{L} \in \mathbf{P}$ . Then there is a Turing machine that computes the characteristic function  $f: \mathbb{B}^* \rightarrow \mathbb{B}^*$  of  $\mathcal{L}$ , with runtime  $\mathcal{O}(x^n)$  for some power  $n \in \mathbb{N}$ . To compute the characteristic function of the reversal,  $\mathcal{L}^R = \{\tau^R \mid \tau \in \mathcal{L}\}$ , the algorithm is given an input string  $\sigma$ . It constructs the reversal,  $\sigma^R$ , and decides whether  $\sigma^R \in \mathcal{L}$ . The runtime is: the algorithm can construct the reversal from the input in one pass through the input, and deciding whether that reversal string is in  $\mathcal{L}$  is  $\mathcal{O}(x^n)$ , to in total the runtime is  $\mathcal{O}(x^{n+1})$ .

**V.4.37** Consider two languages,  $\mathcal{L}_0, \mathcal{L}_1 \in \mathcal{P}(\mathbb{B}^*)$ , that are members of  $\mathbf{P}$ . Then there are powers  $n_0, n_1 \in \mathbb{N}$  so that  $\mathcal{L}_0$  is accepted by a Turing machine with runtime  $\mathcal{O}(x^{n_0})$ , and  $\mathcal{L}_1$  is accepted by a Turing machine with runtime  $\mathcal{O}(x^{n_1})$ .

Here is the algorithm to decide if a given input string  $\sigma \in \mathbb{B}^*$  is a member of  $\mathcal{L}_0 \cap \mathcal{L}_1$ . We will pass through the string, first decomposing it as  $\sigma = \varepsilon \wedge \sigma$ , then as  $\sigma = \sigma[0] \wedge \sigma[1 : ]$ , etc. At each stage of this pass we have  $\sigma = \alpha \wedge \beta$  and we check whether  $\alpha \in \mathcal{L}_0$  and  $\beta \in \mathcal{L}_1$ .

Checking whether  $\alpha \in \mathcal{L}_0$  takes time  $\mathcal{O}(|\alpha|^{n_0})$ , and checking whether  $\beta \in \mathcal{L}_1$  takes time  $\mathcal{O}(|\beta|^{n_1})$ . It simplifies things to overestimate and use  $|\sigma|$ , rather than worry about the length of the substrings. Hence each step of the pass can be done in  $\mathcal{O}(|\sigma|^{\max(n_0, n_1)})$  time. The pass itself takes  $|\sigma| + 1$  many steps, so overall the algorithm runs in polytime,  $\mathcal{O}(|\sigma|^{1+\max(n_0, n_1)})$ .

**V.4.38** We are given a language  $\mathcal{L} \in \mathcal{P}(\mathbb{B}^*)$  such that there is a Turing machine,  $\mathcal{P}_{\mathcal{L}}$ , that decides the language in polytime, in time that is some power,  $n \in \mathbb{N}$ , of the length of the input string.

We will follow the hint that  $\sigma \in \mathcal{L}^*$  if  $\sigma = \varepsilon$ , or  $\sigma \in \mathcal{L}$ , or  $\sigma = \alpha \wedge \beta$  for some  $\alpha, \beta \in \mathcal{L}^*$ . Rather than present a flowchart of pseudocode, we will work through the example of  $\sigma = \langle s_0, s_1, s_2, s_3 \rangle$ . This table lists the substrings  $\sigma[i : j]$ .

	$j = 0$	$j = 1$	$j = 2$	$j = 3$
$i = 0$	$\langle s_0 \rangle$	$\langle s_0 s_1 \rangle$	$\langle s_0 s_1 s_2 \rangle$	$\langle s_0 s_1 s_2 s_3 \rangle = \sigma$
$i = 1$		$\langle s_1 \rangle$	$\langle s_1 s_2 \rangle$	$\langle s_1 s_2 s_3 \rangle$
$i = 2$			$\langle s_2 \rangle$	$\langle s_2 s_3 \rangle$
$i = 3$				$\langle s_3 \rangle$

We start on the main diagonal, which holds all of the length one substrings. We can determine whether each of these is a member of  $\mathcal{L}$  by using  $\mathcal{P}_{\mathcal{L}}$ . If it is a member, put a checkmark next to the substring.

Next we move to the diagonal above, with the length two substrings. We can determine whether each is a member of  $\mathcal{L}$  by using  $\mathcal{P}_{\mathcal{L}}$ , or, taking the hint, by knowing whether both length one substrings are members of  $\mathcal{L}^*$  (which we can do quickly using the checkmarks). Again, when a substring in this diagonal is a member

then checkmark it.

Now iterate. On the diagonal containing the length three substrings, we can determine whether each is a member of  $\mathcal{L}^*$  by first asking  $\mathcal{P}_{\mathcal{L}}$  if it is a member of  $\mathcal{L}$ . If not, we then determine whether it is the concatenation of two substrings. We've already checked or not checked all the members of both of the earlier diagonals, so determining that is very fast.

In this way, we proceed through diagonals until we get to the table's upper right. It holds the given string,  $\sigma$ , so we can determine whether  $\sigma \in \mathcal{L}^*$  in the same way.

Now for the runtime analysis. Let  $m = |\sigma|$ . This is a triply-nested loop. The outer loop moves through the diagonals, from the length one substrings to the length  $m$  substring  $\sigma$ . Inside that is a loop that moves through a particular diagonal, starting at  $i = 0$ , and covering at most  $m$  substrings. Finally, inside of that is a loop that runs the machine  $\mathcal{P}_{\mathcal{L}}$  and then possibly loops through the substring pairs. The runtime on this inner loop is  $\mathcal{O}(m^n) + \mathcal{O}(m)$ , and so in total we have  $\mathcal{O}(m) \cdot \mathcal{O}(m) \cdot (\mathcal{O}(m^n) + \mathcal{O}(m))$ , which is polynomial in  $m = |\sigma|$ .

**V.4.39** Suppose for contradiction that we could solve this problem, that there is a Turing machine that, given  $\mathcal{P}$  as input, decides whether it runs in polytime on the empty input. We will show how to, from  $\mathcal{P}$ , construct a new machine  $\mathcal{N}_{\mathcal{P}}$  that runs in polynomial time if and only if  $\mathcal{P}$  does not halt. That is a contradiction, because then being able to test machines for running in polytime allows us to solve the Halting problem.

The machine  $\mathcal{N}_{\mathcal{P}}$  takes as input bitstrings. For input  $\sigma$ , it uses a universal Turing machine to run  $\mathcal{P}$  on the empty tape. If  $\mathcal{P}$  does not halt within  $|\sigma|$  steps, then  $\mathcal{N}_{\mathcal{P}}$  halts. If  $\mathcal{P}$  does halt within that time then  $\mathcal{N}_{\mathcal{P}}$  will run for  $2^{|\sigma|}$  many steps. Clearly, if  $\mathcal{P}$  halts then  $\mathcal{N}_{\mathcal{P}}$  runs in exponential time, and if  $\mathcal{P}$  does not halt then  $\mathcal{N}_{\mathcal{P}}$  runs in polynomial time, as required.

#### V.4.40

(A) Recall that a function is a kind of relation. Assume that there are computable functions  $f_0, f_1: \mathbb{N} \rightarrow \mathbb{N}$  with associated Turing machines  $\mathcal{P}_0$  and  $\mathcal{P}_1$  that, given  $a \in \mathbb{N}$ , can find  $f_0(a)$  or  $f_1(a)$  in times  $\mathcal{O}(x^{n_0})$  and  $\mathcal{O}(x^{n_1})$ , for some powers  $n_0, n_1 \in \mathbb{N}$ .

For addition the algorithm can, given  $a \in \mathbb{N}$ , compute  $f_0(a)$  and  $f_1(x)$  in time  $\mathcal{O}(x^{\max(n_0, n_1)})$ . The grade school addition algorithm brings the time up to  $\mathcal{O}(x^{1+\max(n_0, n_1)})$ .

The scalar multiplication algorithm can, given  $a \in \mathbb{N}$ , compute  $f_0(a)$  in time  $\mathcal{O}(x^{n_0})$ . The naive multiplication algorithm brings the time up to  $\mathcal{O}(x^{2+n_0})$ .

To subtract we can use a branch that tests if the second number is greater than the first, and if so returns 0. In any event, clearly polytime.

Multiplication is similar, and also in polytime.

For composition, if  $f_0$  is  $\mathcal{O}(x^{n_0})$  and  $f_1$  is  $\mathcal{O}(x^{n_1})$  then the composition is  $\mathcal{O}(x^{n_0 n_1})$ , also polytime.

(B) Here is the naive algorithm. The solution algorithm gets a string  $\sigma$ . it can determine in polytime that  $\sigma = \text{str}(\langle a, y \rangle)$  for some  $a, y \in \mathbb{N}$  (if  $\sigma$  does not have that form then the algorithm rejects it). Now, the algorithm uses  $\mathcal{P}_0$  to determine whether  $f_0(a) = 0, f_0(a) = 1, \dots, f_0(a) = y$ . If none of those relations are true then the algorithm rejects  $\sigma$ . Otherwise, let  $k \in \mathbb{N}$  be such that  $f_0(a) = k$ . The algorithm uses  $\mathcal{P}_1$  to determine whether  $f_1(a) = y - k$ . If so, the algorithm accepts  $\sigma$ , otherwise it rejects.

The reason it is pseudopolynomial and not properly polynomial lies in the testing of the  $y + 1$  many pairs. Inside of the input  $\sigma$ , the number  $y$  is represented in binary. So taking  $y + 1$  steps is taking time that is exponential in the length of the input.

**V.4.41** Because  $\mathcal{L}_0 \in \mathbf{P}$ , there is a Turing machine with runtime  $\mathcal{O}(x^n)$  for some power  $n \in \mathbb{N}$ , and whose input/output behavior is the characteristic function of  $\mathcal{L}_0$ . Because  $\mathcal{L}_1 \leq_p \mathcal{L}_0$ , there is a Turing machine whose runtime is  $\mathcal{O}(x^m)$  for some  $m \in \mathbb{N}$ , whose input/output behavior is the function  $f: \mathbb{N} \rightarrow \mathbb{N}$ , and such that  $\sigma \in \mathcal{L}_1$  if and only if  $f(\sigma) \in \mathcal{L}_0$ .

So, to decide questions about whether  $\sigma \in \mathcal{L}_1$ , first translate them, at a cost of  $\mathcal{O}(|\sigma|^m)$ , to questions about whether  $f(\sigma) \in \mathcal{L}_0$ . Those questions we can settle in  $\mathcal{O}(|\sigma|^n)$ . The result,  $\mathcal{O}(|\sigma|^{\max(n, m)})$ , is polytime.

**V.5.13** That lemma requires that the verifier have the property that if  $\sigma \in \mathcal{L}$  then there is a witness leading to acceptance, while if  $\sigma \notin \mathcal{L}$  then there exists no witness that would cause the verifier to accept.

For instance, consider the problem  $\mathcal{L} = \{n \in \mathbb{N} \mid n \text{ is even}\}$ . Your study partner's proposed verifier inputs a pair  $\langle \sigma, \omega \rangle$ , where  $\sigma$  is a number and  $\omega$  is a bit, and then accepts the pair if  $\omega = 1$ . Absolutely, if you give your verifier the pair  $\langle 101, 0 \rangle$  then it will not accept (here, 101 represents the number 5, and is not an element

of  $\mathcal{L}$ ). However, give it  $\langle 101, 1 \rangle$  and your verifier will accept. So, for  $\sigma = 101$  there exists an  $\omega$  leading to acceptance. Consequently, your verifier is wrong.

**V.5.14**

(A) This formula is satisfiable.

$P$	$Q$	$R$	$P \wedge Q$	$\neg Q$	$\neg Q \wedge R$	$(P \wedge Q) \vee (\neg Q \wedge R)$
F	F	F	F	T	F	F
F	F	T	F	T	T	T
F	T	F	F	F	F	F
F	T	T	F	F	F	F
T	F	F	F	T	F	F
T	F	T	F	T	T	T
T	T	F	T	F	F	T
T	T	T	T	F	F	T

(B) This formula is not satisfiable.

$P$	$Q$	$R$	$P \rightarrow Q$	$P \wedge Q$	$\neg P$	$(P \wedge Q) \vee \neg P$	$\neg((P \wedge Q) \vee \neg P)$	$(P \rightarrow Q) \wedge \neg((P \wedge Q) \vee \neg P)$
F	F	F	T	F	T	T	F	F
F	F	T	T	F	T	T	F	F
F	T	F	T	F	T	T	F	F
F	T	T	T	F	T	T	F	F
T	F	F	F	F	F	F	T	F
T	F	T	F	F	F	F	T	F
T	T	F	T	T	F	T	F	F
T	T	T	T	T	F	T	F	F

**V.5.15** True, by Lemma 5.7.

**V.5.16** The machine accepts  $\sigma$  because there is at least one accepting branch.

**V.5.17**

- (A) Backtracking will solve the problem. But it is a deterministic algorithm. (Technically, a deterministic algorithm is a special case of a nondeterministic one, but when you ask about it, your professor says, “The point of this exercise is to demonstrate understanding of nondeterminism, not to quibble like a sea lawyer.”)
- (B) Your algorithm chooses, but it does so at random. If there are paths through the maze, but the random choices happens not to fall on such a path, then your algorithm will not detect them. A nondeterministic algorithm is required to output 1 if there is a path (and not to output 1 if there is not).
- (C) We will give two answers, one using the unbounded parallelism imagery and one using choosing.

The unbounded parallelism algorithm has a computational history that is a tree. Whenever the algorithm finds itself at a place in the maze where it can do more than one thing, it forks child processes to handle each possibility. So it may have lots of branches, and lots of them may go to dead ends, but if there is a way through the maze then this algorithm will find it. So it reports 1 if there is a path, and never reports 1 when there is not.

The choosing algorithm is to choose. That is, the machine guesses at the correct path (and then verifies it against the input maze). By definition, if there is a way for this guess to be right then the algorithm is right. (Note again that is algorithm is not guessing at random. It is guessing correctly.)

**V.5.18** The algorithm is given an unordered array  $\langle a_0, a_1, \dots, a_{n-1} \rangle$ . It returns Yes if  $k = a_i$  for some index  $i$ .

In the unbounded parallelism model, the computation tree of the algorithm is a single node with  $n$  children, each of which is a process. Child  $i$  checks whether  $a_i = k$ , and if so causes the algorithm to output Yes.

The guessing model is much the same, except that the machine does not compute the tree. It guesses at a child process, and if the array entry equals  $k$  then it causes the algorithm to output Yes. This model, by definition, is the same as the other because it says that an algorithm is right if there is a way for the guess to correctly result in Yes, which will happen if and only if there is an array entry that equals  $k$ .

**V.5.19**

- (A) A deterministic algorithm is to try every permutation, one after the other, until the program recognizes the result as correctly decoding.

- (B) The unbounded parallelism model is to do the same, except that it tries them simultaneously. That is, the computation tree has a single root node with one child process for each of the  $26!$ -many permutations. That child takes its single permutation, substitutes it, and then checks it. This machine has 403 291 461 126 605 635 584 000 000 – 1 child processes that fail the check. But one of the substitutions is right, so overall you get the correctly decoded string.
- (C) The guessing model guesses. It guesses a permutation (or is given it in a hint from the demon), and then it works like any of the prior item's processes: it makes the substitution and then applies the program to check it. This is correct because under this model of nondeterminism it just needs to be possible to have a guess that checks out.

**V.5.20** First the unbounded parallelism description, which works by brute force. Where the number of vertices in the graph is  $k$ , the number of colorings is bounded, by  $4^k$ . The computation tree has a single root node with that many branches. Each branch tries its coloring and then checks to see whether it is correct, that is, whether all connected vertices are different colors. If there is a correct coloring then this algorithm will find it and output Yes.

In the second model, the machine does not spawn  $4^k$ -many branches. The demon gives the machine a branch and the machine just verifies that all connected vertices are different colors. If a correct coloring exists for this graph then there exists a hint that results in the machine verifying a correct coloring and outputting Yes, so this satisfies the definition of a guessing algorithm working.

**V.5.21** We recast this as a sequence of language decision problems for a family of languages parametrized by the bound  $B \in \mathbb{N}$ .

$$\mathcal{L}_B = \{ \langle x_0, x_1, \dots, x_n \rangle \in \mathbb{Z}^{n+1} \mid \text{the } x_j \text{ satisfy all the constraints and } f(x_0, x_1, \dots, x_n) \leq B \}$$

An unboundedly parallel algorithm tries all possible assignments, in some way guaranteed to try every assignment eventually (as with Cantor's enumeration). If there is a satisfying assignment then this algorithm will find it and accept that branch. If there is no satisfying assignment then the computation history tree is infinite, but with no accepting branch.

A guessing algorithm guesses an assignment  $\langle x_0, \dots, x_n \rangle$  and then verifies that the assignment has  $f(x_1, \dots, x_n) \leq B$ . If there is a satisfying assignment then there is a way to guess it so, by definition, this machine recognizes the language  $\mathcal{L}_B$ .

**V.5.22** The language is  $\mathcal{L} = \{ n \in \mathbb{N} \mid n \text{'s prime factorization has exactly two nontrivial primes} \}$ .

One unbounded parallelism algorithm tries each conceivable prime factorization, simultaneously. For instance, given the input 8, it could check the products  $2^0 3^0 5^0 7^0$ ,  $2^1 3^0 5^0 7^0$ ,  $\dots$ ,  $2^7 3^7 5^7 7^7$ , to see whether any of them equals 8 and nontrivially involves exactly two primes. Multiplying is fast and checking whether a number is composite or prime can also be done in polytime. Thus each branch check happens in polytime.

A guessing algorithm is that the machine inputs the number  $n$  and guesses a two-prime factorization (or is given one by a demon). For instance, given 45, the machine might guess  $3^2 5^1$ . (It could also guess  $3^1 5^2$ , but that won't verify.) Again, the verification is polytime because multiplication is polytime, as is checking whether the two numbers are composite or prime. By definition, the machine recognizes the language  $\mathcal{L}$  if, given  $n \in \mathbb{N}$ , when  $n \in \mathcal{L}$  then there exists a guess that verifies, and when  $n \notin \mathcal{L}$  then there is no way for a guess to verify. So this algorithm recognizes  $\mathcal{L}$ .

**V.5.23**

(A) This is the language.

$$\mathcal{L} = \{ M \in X \times Y \times Z \mid M \text{ has an } n\text{-element subset } \hat{M} \text{ where no triples agree on any coordinate} \}$$

A nondeterministic algorithm, framed in terms of guessing, is: given  $M$ , the machine guesses a size  $n$  subset  $\hat{M}$  meeting the conditions. Verifying the conditions clearly takes only polytime. If there is such an  $\hat{M}$  then there is a way for the machine to guess it, and it will verify. If there is no such  $\hat{M}$  then there is no way for the machine to guess one that verifies. So by the definition of this model of nondeterminism, the machine recognizes the language in polytime.

(B) Here is the language.

$$\mathcal{L} = \{ A \mid A \text{ is a multiset of natural numbers with an } \hat{A} \subseteq A \text{ so that } \sum_{a \in \hat{A}} a = \sum_{a \in A - \hat{A}} a \}$$

A nondeterministic algorithm stated in guessing terms is that, when asked to determine if the given  $A$  is a member of  $\mathcal{L}$ , the machine guesses at a  $\hat{A}$ . (That is,  $\hat{A}$  is the witness.) Verifying that  $\sum_{a \in \hat{A}} a = \sum_{a \notin \hat{A}} a$  is clearly polytime. If there is such a partition then for this machine there exists a guess that would pass verification. If there is no such partition then no guess will verify. So by the definition, this machine recognizes the language  $\mathcal{L}$ .

**V.5.24** For unbounded parallelism, we can try every possible  $B$ -coloring of the graph's nodes. There are a large but finite number of those (where the graph has  $k$  vertices, there are  $B^k$ -many colorings). So the computation tree has a single node with that many children. Each coloring is easy to check for validity, since we just have to check every pair of vertices to see if they are the same color. If any of the child nodes is OK, then there is a  $B$ -coloring.

For guessing, the machine nondeterministically selects one coloring (that is, it guesses one or is given one by a demon). It then checks, by iterating through every vertex pair to see if they are the same color. If there is a way to correctly nondeterministically guess then by definition this algorithm succeeds.

**V.5.25**

(A) This is the language  $\mathcal{L}$ .

$$\{ \sigma \mid \sigma \text{ represents a Propositional Logic statement where at least two lines of the truth tables end in } T \}$$

These are suitable fill-ins: (1) a Propositional Logic statement, (2) a pointer to two lines of the truth table, (3) for those two truth table lines, the statement returns  $T$ , (4) two truth table lines that cause the expression  $\sigma$  to evaluate to  $T$ , and (5) pair of truth table lines.

(B) Here is the problem cast as the decision problem for a language.

$$\mathcal{L} = \{ \langle S, T \rangle \mid \text{a subset of } A \subseteq S \text{ has } \sum_{a \in A} a = T \}$$

These are fill-ins: (1) a set and number pair  $\langle S, T \rangle$ , (2) a set of numbers, (3) that is a subset of the given  $S$  and that its elements sum to  $T$ , (4) a subset of  $S$  whose elements sum to  $T$ , and (5) subset of  $S$ .

**V.5.26** Lemma 5.9 requires that we produce a deterministic Turing machine verifier,  $\mathcal{V}$ . It must input pairs of the form  $\langle \sigma, \omega \rangle$ , where  $\sigma = \langle s_0, \dots, s_5, T \rangle$ . It must have the property that if  $\sigma \in \mathcal{CD}$  then there is a  $\omega$  such that  $\mathcal{V}$  accepts the input, while if  $\sigma \notin \mathcal{CD}$  then there is no such witness  $\omega$ . And it must run in time polynomial in  $|\sigma|$ .

The witness  $\omega$  is an arithmetic expression (or, more precisely, a string representation of an arithmetic expression) that evaluates to the target  $T$  and that involves a subset of the six numbers  $s_0, \dots, s_5$ , each used at most once. The verifier checks that the expression does indeed evaluate to the target, and does indeed use each of the six numbers at most once. Clearly those checks can be done in polytime.

If  $\sigma \in \mathcal{CD}$  then by definition there is an arithmetic expression, and so a witness  $\omega$  exists that will cause  $\mathcal{V}$  to accept the input. If  $\sigma \notin \mathcal{CD}$  then there is no such arithmetic expression, and therefore no witness  $\omega$  will cause  $\mathcal{V}$  to accept.

**V.5.27** We recast the Traveling Salesman problem from an optimization problem by using bounds  $B \in \mathbb{N}$  to get this parametrized set of bitstrings.

$$\mathcal{TS}_B = \{ G \in \mathbb{B}^* \mid G \text{ is a weighted graph with a circuit of length less than } B \}$$

To show that this problem is in NP, Lemma 5.9 requires that we produce a deterministic Turing machine verifier,  $\mathcal{V}$ . It must input pairs of the form  $\langle \sigma, \omega \rangle$ , where  $\sigma$  is a graph,  $\mathcal{G}$  (technically, a bitstring representation of a graph). The verifier must have the property that if  $\sigma \in \mathcal{TS}_B$  then there is a  $\omega$  such that  $\mathcal{V}$  accepts the input, while if  $\sigma \notin \mathcal{TS}_B$  then there is no such  $\omega$ . This verifier must run in time polynomial in  $|\sigma|$ .

For a witness we can use a circuit  $\omega = \langle v_{i_0}, v_{i_1}, \dots, v_{i_k} \rangle$  that is of total cost less than the bound. The verifier gets an input pair  $\langle \sigma, \omega \rangle$  where  $\sigma$  is a weighted graph, and checks that  $\omega$  is indeed a Hamiltonian circuit of the graph, and that it costs less than the bound. A verifier can do those checks in time polynomial in  $|\sigma|$ . If the graph  $\sigma$  is indeed an element of  $\mathcal{TS}_B$  then there is such an  $\omega$ , so there is a pair  $\langle \sigma, \omega \rangle$  that the verifier can accept. If  $\sigma$  is not an element of  $\mathcal{TS}_B$  then there is no such  $\omega$ , and so the verifier will never accept such a pair.

**V.5.28** The Independent Sets problem is the decision problem for this language.

$$\mathcal{L} = \{ \langle \mathcal{G}, n \rangle \mid \text{the graph } \mathcal{G} \text{ has at least } n \text{ independent vertices} \}$$

To show that this language decision problem is in NP, Lemma 5.9 requires that we produce a deterministic Turing machine verifier,  $\mathcal{V}$ . This verifier must input pairs of the form  $\langle \sigma, \omega \rangle$ , where  $\sigma$  is a graph-natural number pair,  $\langle \mathcal{G}, n \rangle$ . The verifier must satisfy that if  $\sigma \in \mathcal{L}$  then there is a  $\omega$  such that  $\mathcal{V}$  accepts the input, while if  $\sigma \notin \mathcal{L}$  then there is no such  $\omega$ . This verifier must run in time polynomial in  $|\sigma|$ .

For a witness, take a size- $n$  set of independent vertices from the graph,  $\omega = \langle v_0, \dots, v_{n-1} \rangle$ . The verifier gets an input pair  $\langle \sigma, \omega \rangle$  where  $\sigma$  is a graph, and checks that the vertices are in the graph and that no two of them are connected. Those checks take time polynomial in  $|\sigma|$  (obviously, this depends on the graph being represented with reasonable efficiency, but we always make that assumption). If the pair  $\sigma$  is indeed an element of  $\mathcal{L}$  then there is such a witness  $\omega$ , so there exists a pair  $\langle \sigma, \omega \rangle$  that the verifier can accept. If  $\sigma$  is not an element of  $\mathcal{L}$  then there is no such  $\omega$ , and so there does not exist a pair that the verifier will accept.

**V.5.29** The Knapsack problem is the decision problem for this language,  $\mathcal{L}$ .

$$\{ \langle (w_0, v_0), \dots, (w_{n-1}, v_{n-1}), B, T \rangle \mid \text{there is } I = \{i_0, \dots, i_k\} \subseteq \{0, \dots, n-1\} \text{ so } \sum_{i \in I} w_i \leq B \text{ and } \sum_{i \in I} v_i \geq T \}$$

To show that this language decision problem is in NP, Lemma 5.9 requires that we produce a deterministic Turing machine verifier,  $\mathcal{V}$ . The verifier input pairs  $\langle \sigma, \omega \rangle$ , where  $\sigma = \langle (w_0, v_0), \dots, (w_{n-1}, v_{n-1}), B, T \rangle$ . The verifier must have the property that if  $\sigma \in \mathcal{L}$  then there is a  $\omega$  such that  $\mathcal{V}$  accepts the input, while if  $\sigma \notin \mathcal{L}$  then there is no such  $\omega$ . This verifier must also run in time polynomial in  $|\sigma|$ .

For a witness, take a set  $\omega = \{i_0, \dots, i_k\} \subseteq \{0, \dots, n-1\}$ . The verifier gets an input pair  $\langle \sigma, \omega \rangle$  and checks that the sum of the weights of the indicated pairs,  $\sum_{i \in \omega} w_i$ , is less than or equal to the bound  $B$ , and also that the sum of the values of pairs,  $\sum_{i \in \omega} v_i$ , is greater than or equal to the target  $T$ . Those checks clearly take time polynomial in  $|\sigma|$ .

If  $\sigma$  is indeed an element of  $\mathcal{L}$  then there is such a selection of indices, so there is such a witness  $\omega$ , so there exists a pair  $\langle \sigma, \omega \rangle$  that the verifier can accept. If  $\sigma$  is not an element of  $\mathcal{L}$  then there is no such  $\omega$ , and so there does not exist a pair that the verifier will accept.

**V.5.30** By Lemma 5.7 it suffices to show that the language decision problem is in P. But that's clear: given a triple  $\langle a, b, c \rangle$ , checking whether the first two sum to the third is clearly a polytime task.

**V.5.31** We start by expressing this as a language decision problem.

$$\mathcal{L} = \{ \langle \mathcal{G}, B \rangle \mid \mathcal{G} \text{ has a simple path of length at least } B \}$$

To show that this problem is in NP, Lemma 5.9 wants a deterministic Turing machine verifier,  $\mathcal{V}$ . The verifier input pairs  $\langle \sigma, \omega \rangle$ , where  $\sigma = \langle \mathcal{G}, B \rangle$ . The verifier must satisfy that if  $\sigma \in \mathcal{L}$  then there is a  $\omega$  such that  $\mathcal{V}$  accepts the input pair, while if  $\sigma \notin \mathcal{L}$  then there is no such  $\omega$ . Also, the verifier must run in time polynomial in  $|\sigma|$ .

For a witness, take a path  $\omega = \langle v_0, \dots, v_{n-1} \rangle$ . The verifier gets an input pair  $\langle \sigma, \omega \rangle$  and checks that the path is in  $\mathcal{G}$ , is simple (that no two vertices are equal), and has length at least  $B$ . Those checks clearly can be done in time polynomial in  $|\sigma|$ .

If  $\sigma$  is indeed an element of  $\mathcal{L}$  then there is such a path  $\omega$ , so there exists a pair  $\langle \sigma, \omega \rangle$  that the verifier accepts. If  $\sigma$  is not an element of  $\mathcal{L}$  then there is no such  $\omega$ , and so there does not exist a pair that the verifier accepts.

**V.5.32**

(A) Here is the recasting, as the decision problem for this language.

$$\mathcal{L}_0 = \{ \langle a, b \rangle \in \mathbb{N}^2 \mid \text{there exists } x \in \mathbb{N} \text{ with } ax + 1 = b \}$$

For Lemma 5.9, we will produce a deterministic Turing machine verifier,  $\mathcal{V}$ . It inputs pairs  $\langle \sigma, \omega \rangle$ , where  $\sigma$  is the number pair  $\langle a, b \rangle$ . The verifier must satisfy that if  $\sigma \in \mathcal{L}_0$  then there is a witness  $\omega$  such that  $\mathcal{V}$  accepts the input pair, while if  $\sigma \notin \mathcal{L}_0$  then there is no such  $\omega$ . The verifier must run in time polynomial in  $|\sigma|$ .

For a witness, take a number  $\omega = x$ . The verifier gets an input pair  $\langle \sigma, \omega \rangle$  and checks that  $ax + 1 = b$ . That can be done in time polynomial in  $|\sigma|$ .

If  $\sigma$  is indeed an element of  $\mathcal{L}_0$  then there is such a number  $\omega = x$ , so there exists a pair  $\langle \sigma, \omega \rangle$  that the verifier can accept. If  $\sigma$  is not an element of  $\mathcal{L}_0$  then there is no such  $\omega$ , and so there does not exist a pair that the verifier accepts.

(B) This is the associated language.

$$\mathcal{L}_1 = \{ M \mid \text{there is a set } P \subset \mathbb{Z} \text{ of numbers such that } M \text{ is the multiset of pairwise distances} \}$$

Consider a deterministic Turing machine verifier,  $\mathcal{V}$ , that inputs pairs  $\langle \sigma, \omega \rangle$ , where  $\sigma$  is a multiset  $M$ . For a witness, take a set of positions  $\omega = P$ . The verifier gets an input pair  $\langle \sigma, \omega \rangle$  and checks that the pairwise distances among elements of  $P$  equals  $M$ . That can be done in time polynomial in  $|\sigma|$ .

If  $\sigma \in \mathcal{L}_1$  then there is such a set of positions  $\omega = P$ , so there exists a pair  $\langle \sigma, \omega \rangle$  that the verifier accepts.

If  $\sigma \notin \mathcal{L}_1$  then there is no such set  $\omega$ , and so there does not exist a pair that the verifier accepts.

**V.5.33** Countable. A verifier is a deterministic Turing machine, and there are countably many such machines.

**V.5.34** We use Lemma 5.9. This is a suitable language (where the road map is  $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ ).

$$\mathcal{L} = \{ N \subseteq \mathcal{N} \mid N \text{ is a set of vertices where there are two suitable cycles} \}$$

To use Lemma 5.9, we must produce a verifier and suitable witnesses. The verifier inputs  $\langle \sigma, \omega \rangle$ , where  $\sigma = N$  (technically,  $\sigma$  is a string representation of a set  $N \subseteq \mathcal{N}$ ). A witness is the two cycles,  $\omega = \langle c_0, c_1 \rangle$ . The verifier checks that every vertex  $v \in V$  is on at least one of the two cycles, and that each of the cycles is of length at most  $B$ . Clearly that check can be done in time polynomial in  $|\sigma|$ .

If  $\sigma \in \mathcal{L}$  then there is a pair of cycles, so there exists a suitable witness  $\omega$  such that the verifier accepts the pair  $\langle \sigma, \omega \rangle$ . If  $\sigma \notin \mathcal{L}$  then there are not two such cycles. So there is no such witness  $\omega$  that checks out, and so the verifier will not accept for any such  $\omega$ .

**V.5.35**

(A) The natural language is  $\mathcal{L} = \{ \langle \mathcal{G}_0, \mathcal{G}_1 \rangle \mid \text{they are isomorphic} \}$ .

(B) We use Lemma 5.9, so we must produce a verifier and suitable witnesses. The verifier inputs pairs  $\langle \sigma, \omega \rangle$  where  $\sigma$  is a pair  $\langle \mathcal{G}_0, \mathcal{G}_1 \rangle$ . The witness is a function, a set of ordered pairs  $\langle v, f(v) \rangle$ . The verifier checks that the function satisfies the conditions, that it is one-to-one and onto, and that  $\{v, \hat{v}\}$  is an edge of  $\mathcal{G}_0$  if and only if  $\{f(v), f(\hat{v})\}$  is an edge of  $\mathcal{G}_1$ . That check takes time polynomial in  $|\sigma|$ , because it depends only on the size of the two graphs.

If  $\sigma \in \mathcal{L}$  then there is such a function, so there exists a hint  $\omega$  that will allow the verifier to accept the pair  $\langle \sigma, \omega \rangle$ . If the two graphs are not isomorphic then no witness  $\omega$  will verify.

**V.5.36**

(A) As in the hint, fix a vertex,  $v_0$ . Because the graph is connected, every vertex can be reached by a path that starts with  $v_0$ . For each of its neighbors,  $v$ , consider the set of vertices  $R_v$  that can be reached by a path from  $v$  that does not go through  $v_0$ . At least one of the neighbors must have infinitely many such vertices, or else the graph is a union of the finitely many sets  $R_v$  (and  $v_0$ ), each of which is finite. Pick one such neighbor and call it  $v_1$ . By repeating this process we get an infinite path  $\langle v_0, v_1, \dots \rangle$ .

(B) Suppose that a nondeterministic Turing machine  $\mathcal{P}$  recognizes a language  $\mathcal{L}$ . We will describe how a deterministic machine  $\mathcal{Q}$  can do the same. The nondeterministic machine  $\mathcal{P}$  generates a computation tree and  $\mathcal{Q}$  can do a breadth-first traversal of that tree. Fix an input  $\sigma$ . If the tree triggered by  $\sigma$  has an accepting branch then  $\mathcal{Q}$  will eventually reach it, and will accept that input. If the tree has no accepting branch then  $\mathcal{Q}$  will never find one, and so will not accept  $\sigma$ .

**V.5.37** A configuration of a nondeterministic Turing machine  $\mathcal{P}$  is a set of four-tuples,  $\langle q, s, \tau_L, \tau_R \rangle$ , where  $q \in Q$ ,  $s \in \Sigma$ , and where  $\tau_L$  and  $\tau_R$  are strings of elements from the tape alphabet,  $\tau_L, \tau_R \in \Sigma^*$ . These signify the current state, the character under the read/write head, and the tape contents to the left and right of the head. We write  $\mathcal{C}(t)$  for the machine's configuration after the  $t$ -th transition, and say that this is the configuration at step  $t$ . The initial configuration has the form  $\mathcal{C}(0) = \{ \langle q_0, s, \alpha, \varepsilon \rangle \}$ . We say that  $\alpha \frown \langle s \rangle$  is the machine's input.

We next describe how the machine  $\mathcal{P}$  transitions from being in configuration  $\mathcal{C}(t)$  to being in the next configuration,  $\mathcal{C}(t+1)$ . Start with an empty set and then get  $\mathcal{C}(t+1)$  by adding instructions to it as follows.

For each  $\langle q, s, \tau_L, \tau_R \rangle \in \mathcal{C}(t)$ , compute the set  $\Delta(q, s)$ . That set may be empty, but if not then for each instruction  $I = q_p T_p T_n q_n$  in that set, there are three possibilities. (1) If  $T_n \in \Sigma$  then write that character to the

tape, so that  $\langle q_n, T_n, \tau_L, \tau_R \rangle \in \mathcal{C}(t+1)$ . We say that  $I \vdash \langle q_n, T_n, \tau_L, \tau_R \rangle$ . (2) If  $T_n = L$  then the machine moves the tape head to the left, that is,  $\langle q_n, \hat{s}, \hat{\tau}_L, \hat{\tau}_R \rangle \in \mathcal{C}(t+1)$ , where  $\hat{s}$  is the rightmost character of the string  $\tau_L$  (if  $\tau_L = \varepsilon$  then  $\hat{s}$  is the blank character), where  $\hat{\tau}_L$  is  $\tau_L$  with its rightmost character omitted (if  $\tau_L = \varepsilon$  then  $\hat{\tau}_L = \varepsilon$  also), and where  $\hat{\tau}_R$  is the concatenation of  $\langle s \rangle$  and  $\tau_R$ . We say that  $I \vdash \langle q_n, \hat{s}, \hat{\tau}_L, \hat{\tau}_R \rangle$ . (3) If  $T_n = R$  then the machine moves the tape head to the right. This is like (2) so we omit the details.

A computation tree branch is a sequence of configurations  $I_0 \vdash \mathcal{C}(1) \vdash \dots \vdash \mathcal{C}(h)$ , starting with the initial configuration  $\mathcal{C}(0)$ , and such that there is no instruction  $I$  where  $I_k \vdash I$ . If the instruction  $I_k = \langle \hat{q}, \hat{s}, \beta, \gamma \rangle$  is such that its state is accepting,  $\hat{q} \in A$ , then we say the machine  $\mathcal{P}$  accepts the input.

### V.5.38

- (A) The class **NP** is the set of languages decidable by a nondeterministic machine in polytime. Every problem decidable by a nondeterministic Turing machine is decidable by a deterministic Turing machine. The Halting problem is not decidable by a deterministic Turing machine, so it is not in **NP**.
- (B) There is no polynomial bound on the number of steps, and consequently on the length of  $\omega$ .

**V.6.9** Lemma 6.4 does not say that every  $\mathcal{L} \in \mathbf{P}$  is  $\leq_p$  to every other language. It says that it is related in that way to every other language in the collection **Rec**. Each language in that collection has a Turing machine decider and so there are only countably many such languages.

**V.6.10** The same is true about **NP**. Both are immediate from Lemma 6.4.

**V.6.11** Assume  $\mathcal{L} \leq_p \mathcal{L}^c$ . Then there is a polytime  $f$  so that  $\sigma \in \mathcal{L}$  if and only if  $f(\sigma) \in \mathcal{L}^c$ . That's equivalent to  $\sigma \notin \mathcal{L}$  if and only if  $f(\sigma) \notin \mathcal{L}^c$ , which holds when  $\sigma \in \mathcal{L}^c$  if and only if  $f(\sigma) \in \mathcal{L}$ .

**V.6.12** There is indeed a subset that adds to  $T$ , namely  $A = \{23, 31, 72\}$ .

### V.6.13

- (A) This is wrong. For instance, if  $A = \emptyset$  then  $\emptyset \leq_p B$  for any language  $B$  but while there is a decider for  $\emptyset$  that runs in time  $\mathcal{O}(1)$ , there may be no polynomial time decider for  $B$  (for instance, if  $B = K$  then there is no decider at all). The correct statement is the opposite: a polytime decider for  $B$  can be used to decide the set  $A$  in polytime.
- (B) This is also wrong. As in the prior item, if  $A$  is the empty set, which is decidable, and  $B = K$ , then  $A \leq_p B$  but  $B$  is not decidable in polytime. The reason that  $A \leq_p B$  is that  $A$  can be decided in polytime without any reference to  $B$  at all; in fact, deciding  $A$  is  $\mathcal{O}(1)$ . Instead, the other way around is right: if  $B$  is polytime decidable then  $A$  is polytime decidable also.
- (C) This is true.

### V.6.14

- (A) Three are:  $\beta_2 = 1001001$ ,  $\beta_1 = 1100100$ , and  $\beta_0 = 0110010$ .
- (B) It is a cyclic shift, starting from index  $k = 6$ .
- (C) It is the decision problem for this language.

$$\mathcal{L}_0 = \{ \langle \sigma, \tau \rangle \in \Sigma^2 \mid \tau \text{ is a substring of } \sigma \}$$

- (D) It is the decision problem for this.

$$\mathcal{L}_1 = \{ \langle \alpha, \beta \rangle \in \Sigma^2 \mid \beta \text{ is a cyclic shift of } \alpha \}$$

- (E) The key is the observation that for same length strings,  $\beta$  is a cyclic shift of  $\alpha$  if and only if  $\beta$  is a substring of  $\alpha \hat{\sim} \alpha$ . So the reduction function  $f$  inputs  $\langle \alpha, \beta \rangle$  and if the two are the same length then it outputs  $\langle \alpha \hat{\sim} \alpha, \beta \rangle$  (else it outputs some pair sure to fail, such as  $\langle \varepsilon, \emptyset \rangle$ ). Clearly  $f$  is computable in polytime, and  $\langle \alpha, \beta \rangle \in \mathcal{L}_1$  if and only if  $f(\langle \alpha, \beta \rangle) \in \mathcal{L}_0$ . Therefore  $\mathcal{L}_1 \leq_p \mathcal{L}_0$ .

### V.6.15

- (A) The Independent Set problem is the decision problem for this language.

$$\mathcal{L}_0 = \{ \langle \mathcal{G}, B \rangle \mid \text{there is } I \subseteq \mathcal{N}(\mathcal{G}) \text{ with } |I| \leq B \text{ such that every edge contains a vertex from } I \}$$

Vertex Cover is the decision problem for this language.

$$\mathcal{L}_1 = \{ \langle \mathcal{H}, D \rangle \mid \text{there is } C \subseteq \mathcal{N}(\mathcal{H}) \text{ with } |C| \leq D \text{ such that every edge contains a vertex from } C \}$$

- (B) A vertex cover with  $k = 4$  elements is  $S = \{q_2, q_5, q_8, q_9\}$ .
- (C) An independent set with  $\hat{k} = 6$  elements is  $\hat{S} = \{q_0, q_1, q_3, q_4, q_6, q_7\}$ .
- (D) For any edge, if it has at least one endpoint in  $S$  then it has at most one endpoint in the complement  $\hat{S} = \mathcal{N} - S$ . Conversely, If an edge has at most one endpoint in  $\hat{S}$  then it has at least one endpoint in the complement  $S = \mathcal{N} - \hat{S}$ .
- (E) We will do Independent Set  $\leq_p$  Vertex Cover; the other reduction is similar. The definition requires that we produce a polytime computable function  $f$ . Given an instance of Independent Set  $\sigma = \langle \mathcal{G}, \hat{k} \rangle$ , then we convert that to an instance of Vertex Cover  $f(\sigma) = \langle \mathcal{G}, |\mathcal{G}| - \hat{k} \rangle$ .

**V.6.16**

- (A) The Hamiltonian Circuit problem is the decision problem for this language.

$$\mathcal{L}_0 = \{ \langle \mathcal{H} \rangle \mid \text{the graph contains a circuit that includes each vertex exactly once} \}$$

The Traveling Salesman problem is this decision problem.

$$\mathcal{L}_1 = \{ \langle \mathcal{G}, B \rangle \mid \text{some circuit of } \mathcal{G} \text{ that includes each vertex has total cost less than or equal to } B \}$$

- (B) The reduction function inputs an instance of Hamiltonian Circuit, a graph  $\mathcal{H} = \langle \mathcal{N}, \mathcal{E} \rangle$  whose edges are unweighted. It returns the instance of Traveling Salesman that uses the vertex set  $\mathcal{N}$  as cities and that takes the distances between the cities to be:  $d(v_i, v_j) = 1$  if  $v_i v_j$  is an edge of  $\mathcal{G}$  and  $d(v_i, v_j) = 2$  if  $v_i v_j \notin \mathcal{E}$ , and such that the bound  $B$  is the number of vertices  $|\mathcal{N}|$ .

The bound means that there will be a Traveling Salesman solution if and only if there is a Hamiltonian Circuit solution, namely the Traveling Salesman solution uses the edges of the Hamiltonian circuit.

Clearly  $\mathcal{H} \in \mathcal{L}_0$  if and only if  $f(\mathcal{H}) \in \mathcal{L}_1$ . What remains is to show that  $f$  runs in polytime. The number of edges is less than twice the number of vertices, so that polytime in the input graph size is the same as polytime in the number of vertices. To output the Traveling Salesman instance, the algorithm examines all the pairs of vertices, which is a nested loop and so takes time that is quadratic in the number of vertices. So  $f$  runs in polytime.

**V.6.17**

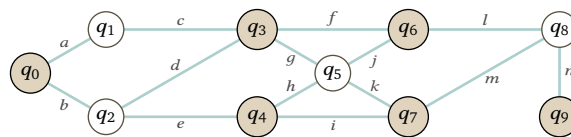
- (A) Vertex Cover is the decision problem for this language.

$$\mathcal{L}_0 = \{ \langle \mathcal{G}, B \rangle \mid \text{there is } C \subseteq \mathcal{N}(\mathcal{G}) \text{ with } |C| \leq B \text{ such that every edge contains a vertex from } C \}$$

Set Cover is the decision problem for this language.

$$\mathcal{L}_1 = \{ \langle S, S_0, \dots, S_{k-1}, D \rangle \mid \text{each } S_j \text{ is a subset of } S, \text{ and } k \leq D, \text{ and } S_0 \cup \dots \cup S_{k-1} = S \}$$

- (B) One vertex cover is  $C = \{q_0, q_3, q_4, q_6, q_7, q_9\}$ .



- (c) The set of edges is  $S = \{a, b, \dots, n\}$ . The subsets are these.

$$S_0 = \{a, b\}, S_1 = \{a, c\}, S_2 = \{b, d, e\}, S_3 = \{c, d, f, g\}, S_4 = \{e, h, i\},$$

$$S_5 = \{g, h, j, k\}, S_6 = \{f, j, l\}, S_7 = \{i, k, m\}, S_8 = \{l, m, n\}, S_9 = \{n\}$$

The union of the subsets associated with vertices in the vertex cover

$$S_{q_0} \cup S_{q_3} \cup S_{q_4} \cup S_{q_6} \cup S_{q_7} \cup S_{q_9} = \{a, b\} \cup \{c, d, f, g\} \cup \{e, h, i\} \cup \{f, j, l\} \cup \{i, k, m\} \cup \{n\}$$

equals  $S$ .

- (D) We must produce a reduction function,  $f$ . It inputs an instance  $\langle \mathcal{G}, B \rangle$  of the **Vertex Cover** problem. The computation is: construct the set  $S$  of all edges, and for each vertex  $v$ , construct  $S_v \subseteq S$  consisting of the edges incident on  $v$ . Then the output of the function is an instance of the **Set Cover** problem,  $f(\langle \mathcal{G}, B \rangle) = \langle S, S_{v_0}, \dots, B \rangle$ . Clearly we can compute this function in polytime, and clearly also  $\langle \mathcal{G}, B \rangle \in \mathcal{L}_0$  if and only if  $\langle S, S_{v_0}, \dots, B \rangle \in \mathcal{L}_1$ .

**V.6.18**

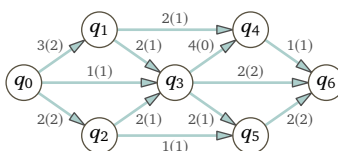
- (A) **Max-Flow** has this language.

$$\mathcal{L}_0 = \{ \langle F, \mathcal{G}, B \rangle \mid \text{the flow } F \text{ meets the constraints of } \mathcal{G} \text{ and the flow into the sink is at least } B \}$$

This is the language for the **Linear Programming** optimization problem.

$$\mathcal{L}_1 = \{ \langle \vec{x}, C, Z, B \rangle \mid \text{the point } \vec{x} \text{ meets the constraints } C \text{ and makes } Z(\vec{x}) \geq B \}$$

- (B) There are algorithms to solve these problems, but for small ones we can do it by eye. The maximum flow is 5, with each edge's numbers given in parentheses.



- (c) The variables are  $x_{0,1}, \dots, x_{5,6}$ . These are the capacity constraints.

$$\begin{aligned} 0 \leq x_{0,1} \leq 3, \quad 0 \leq x_{0,2} \leq 2, \quad 0 \leq x_{0,3} \leq 1, \quad 0 \leq x_{1,3} \leq 2, \quad 0 \leq x_{1,4} \leq 2, \\ 0 \leq x_{2,3} \leq 2, \quad 0 \leq x_{2,5} \leq 1, \quad 0 \leq x_{3,4} \leq 4, \quad 0 \leq x_{3,5} \leq 2, \quad 0 \leq x_{3,6} \leq 2, \\ 0 \leq x_{4,6} \leq 1, \quad \text{and} \quad 0 \leq x_{5,6} \leq 2 \end{aligned}$$

These are the flow constraints.

$$\begin{aligned} x_{0,1} = x_{1,3} + x_{1,4}, \quad x_{0,2} = x_{2,3} + x_{2,5}, \quad x_{0,3} + x_{1,3} + x_{2,3} = x_{3,4} + x_{3,5} + x_{3,6}, \\ x_{1,4} + x_{3,4} = x_{4,6}, \quad \text{and} \quad x_{2,5} + x_{3,5} = x_{5,6} \end{aligned}$$

We want to maximize  $Z = x_{3,6} + x_{4,6} + x_{5,6}$ . Observe that all of these equations and inequalities are linear.

- (D) We must construct a reduction function  $f$ . The input is a triple  $\langle F, \mathcal{G}, B \rangle$ . Use that information to make variables, construct capacity constraints and flow constraints  $C$ , and produce an optimization expression  $Z$ . In particular, the flow  $F$  (shown in parentheses in the network above) gives an associated point  $\vec{x}$  in the **Linear Programming** instance. Clearly that can be done in polytime. Clearly also  $\langle F, \mathcal{G}, B \rangle \in \mathcal{L}_0$  if and only if  $\langle \vec{x}, C, Z, B \rangle \in \mathcal{L}_1$ .

**V.6.19**

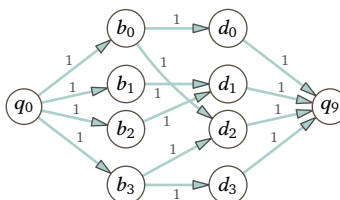
- (A) The maximum number of matches is three, as in  $(b_0, d_0)$ ,  $(b_1, d_1)$ , and  $(b_3, d_2)$ .  
 (B) The **Max-Flow** problem has this language, where  $W$  is a bound.

$$\mathcal{L}_0 = \{ \langle F, \mathcal{G}, W \rangle \mid \text{the flow } F \text{ meets the constraints of } \mathcal{G} \text{ and the flow into the sink is at least } W \}$$

The **Drummer** problem has this, where  $X$  is a bound.

$$\mathcal{L}_1 = \{ \langle B_0, \dots, B_k, (b_0, d_{i_0}), \dots, (b_m, d_{i_m}), X \rangle \mid \text{each } d_{i_j} \in B_j \text{ and } m \geq X \}$$

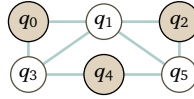
- (c) Here is a network diagram (note the weights of 1 on each edge).



- (D) A reduction function  $f$  is given a sequence  $\langle B_0, \dots, B_k, (b_0, d_{i_0}), \dots, (b_m, d_{i_m}), X \rangle$ . It constructs a network that, from left to right, has a source node, a bank of band nodes, a bank of drummer nodes, and a sink node. Every edge is directed, and every edge has weight 1. The pairs  $(b, d)$  in the input give a flow  $F$ . So the output is the triple  $f(\langle B_0, \dots, B_k, (b_0, d_{i_0}), \dots, (b_m, d_{i_m}), X \rangle) = \langle F, \mathcal{G}, X \rangle$ . Clearly we can compute this  $f$  in polytime. And also clearly,  $\langle B_0, \dots, B_k, (b_0, d_{i_0}), \dots, (b_m, d_{i_m}), X \rangle \in \mathcal{L}_1$  if and only if  $\langle F, \mathcal{G}, X \rangle \in \mathcal{L}_0$ .

### V.6.20

- (A) The natural independent set is this.



### V.6.21

- (A) These constraints

$$0 \leq z_0 \leq 1, \quad 0 \leq z_1 \leq 1, \quad 0 \leq z_2 \leq 1$$

ensure, in an Integer Programming context, that each variable is either 0 or 1. As to the expression, use  $z_0 + (1 - z_1) + (1 - z_2) > 0$ .

- (B) We must produce a reduction function  $f$ . It is given a propositional logic expression. For each variable  $P_i$  in that expression, create a variable  $z_i$  for the Integer Linear Programming problem. To force each  $z_i$  to be either 0 or 1, include the constraint  $0 \leq z_i \leq 1$ . For every clause in the propositional logic expression, add a constraint to the Integer Linear Programming instance as in the prior item.

$$P_i \vee \neg P_j \vee \neg P_k \quad \text{is associated with} \quad z_i + (1 - z_j) + (1 - z_k) > 0$$

Clearly this function can be done in polytime.

### V.6.22

- (A) This is just the definition of 'or'.

- (B) Suppose first that all three polynomials in  $S_{E_0} = \{x_0(1 - x_0), x_1(1 - x_1), x_0(1 - x_1)\}$  have a value of 0. The first polynomial  $x_0(1 - x_0)$  evaluates to 0 if and only if  $x_0$  is either 0 or 1. The second polynomial gives the same for  $x_1$ . The third evaluates to 0 if and only if either  $0 = x_0$  or  $0 = 1 - x_1$ .

- (C) For  $E_1 = (P_0 \vee \neg P_1 \vee \neg P_2) \wedge (P_1 \vee P_2 \vee \neg P_3)$ , the analogous set is this.

$$S_{E_1} = \{x_0(1 - x_0), x_1(1 - x_1), x_2(1 - x_2), x_3(1 - x_3), x_0(1 - x_1)(1 - x_2), x_1x_2(1 - x_3)\}$$

- (D) Take the sum of squares.

$$p(x_0, x_1, x_2, x_3) = [x_0(1 - x_0)]^2 + [x_1(1 - x_1)]^2 + [x_2(1 - x_2)]^2 + [x_3(1 - x_3)]^2 + [x_0(1 - x_1)(1 - x_2)]^2 + [x_1x_2(1 - x_3)]^2$$

- (E) We must produce the reduction function  $f$ , which inputs an instance  $E$  of 3-Satisfiability, a Boolean expression in which all clauses have three or fewer literals, and outputs an instance  $p$  of D. We must verify that  $E$  is satisfiable if and only if  $p$  has zero for some integer  $n$ -tuple. (And we must show it runs in polytime, although that will be clear.)

The function  $f$  constructs the polynomial  $p$  following the pattern outlined in the prior items. For each variable  $P_i$  used in the input expression  $E$ , put  $x_i(1 - x_i)$  into the set  $S_E$ . In addition, for each clause  $L_{i_0} \vee L_{i_1} \vee L_{i_2}$ , where  $L_i$  is a literal, add an at most three-factor polynomial: it has the factor  $x_i$  if  $L_i = P_i$ , and for a negation  $L_i = \neg P_i$  has the factor  $(1 - x_i)$ . Finally, the polynomial  $p$  is the sum of the squares of the polynomials from  $S_E$ .

For one direction of the verification, suppose that  $E$  is satisfiable. For all atoms  $P_i$  that are assigned  $T$ , take  $x_i = 0$ , while if  $P_i$  is assigned  $F$  then take  $x_i = 1$ . Notice that as a result each polynomial member of  $S_E$  has the value of 0. Consequently,  $p$  has a value of 0, and so this set of  $x_i$ 's is a zero.

Now for the other direction. Suppose that we have a tuple of integers  $\vec{c}$  that make the polynomial  $p$  have value 0. Because  $p$  is a sum of squares, each term must have a value of 0. Notice that each term of the form  $x_i(1 - x_i)$  is zero, and therefore  $x_i = 0$  or  $x_i = 1$ . Notice also that each term that matches the form of a clause must have a value of 0, and therefore there is a matching assignment to the atoms  $P_i$  that makes the clause true (namely,  $x_i = 0$  is associated with  $P_i = T$ , and  $x_i = 1$  is associated with  $P_i = F$ ).

V.6.23

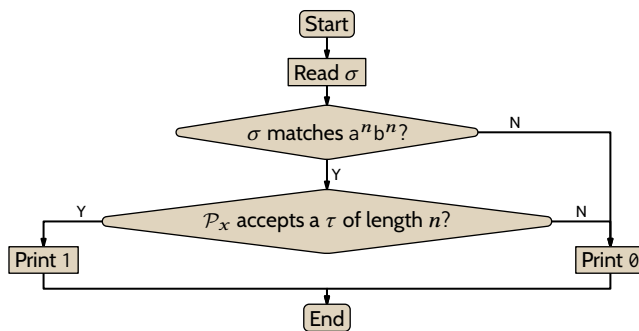
(A) The tweaks are small, but here is the entire argument. Let  $\mathcal{L}$  be an infinite subset of  $\{a^n b^n \mid n \in \mathbb{N}\}$ . For contradiction, assume that it is regular. Then the Pumping Lemma says that  $\mathcal{L}$  has a pumping length,  $p$ .

Consider the list of strings  $a^p b^p, a^{p+1} b^{p+1}, \dots$ . Because  $\mathcal{L}$  is infinite, at least one member of that list is a member of  $\mathcal{L}$ ; let it be  $\sigma$ . It is an element of  $\mathcal{L}$  and its length is greater than or equal to  $p$  so the Pumping Lemma applies. So  $\sigma$  decomposes into substrings  $\sigma = \alpha \hat{\beta} \gamma$  satisfying the three conditions. Condition (1) is that the length of the prefix  $\alpha \hat{\beta}$  is less than or equal to  $p$ . This implies that both  $\alpha$  and  $\beta$  are composed exclusively of a's. Condition (2) is that  $\beta$  is not the empty string, so it contains at least one a.

Condition (3) is that all of the strings  $\alpha \gamma, \alpha \beta^2 \gamma, \alpha \beta^3 \gamma, \dots$  are members of  $\mathcal{L}$ . To get the desired contradiction, consider  $\alpha \beta^2 \gamma$ . Compared with  $\sigma = \alpha \beta \gamma$ , this string has an extra  $\beta$ , which adds at least one a without adding any balancing b's. In short,  $\alpha \beta^2 \gamma$  has more a's than b's and is therefore not a member of  $\mathcal{L}$ . But the Pumping Lemma says that it must be a member of  $\mathcal{L}$ , and therefore the assumption that  $\mathcal{L}$  is regular is incorrect.

(B) The flowchart makes clear that this behavior can be programmed on an everyday computer. Thus Church's Thesis says there is a Turing machine with this behavior. Let that machine have index  $e$ .

Applying the  $s$ - $m$ - $n$  lemma gives a family of machines, of which this is the  $x$ -th.



(C) The reduction function  $f$  must have domain  $\mathbb{N}$  and codomain  $\mathbb{N}$ . It must have the property that  $i \in F$  if and only if  $x \in R$ . And it must run in polynomial time.

The function  $f$  is  $i \mapsto s(e, i)$ . Suppose that  $i \in F$ . Note first that the machine  $\mathcal{P}_i$  is a language decider, and so halts on all inputs  $\tau$ . Because the language of  $\mathcal{P}_i$  is finite, the language of  $\mathcal{P}_{s(e,i)}$  is finite also. Any finite language is regular, so the language of  $\mathcal{P}_{s(e,i)}$  is regular.

Now suppose that  $i \notin F$ . If the machine  $\mathcal{P}_{s(e,x)}$  has even one input  $\sigma$  where the second test box fails to halt on some  $\tau$  then this machine is not a language decider, and so  $s(e, i) \notin R$ . If, on the other hand, for all input  $\sigma$ , the second test box halts on all relevant  $\tau$ 's then because  $i \notin F$ , the language decided by  $\mathcal{P}_{s(e,i)}$  is an infinite subset of  $\{a^n b^n \mid n \in \mathbb{N}\}$ , and so is not regular.

V.6.24

(A) A little time playing with the table numbers

$C(t_i, w_j)$	$w_0$	$w_1$	$w_2$	$w_3$
$t_0$	13	4	7	6
$t_1$	1	11	5	4
$t_2$	6	7	2	8
$t_3$	1	3	5	9

shows that the optimum solution is to pair  $t_0, w_1$ , along with  $t_1, w_3$ , and  $t_2, w_2$ , and finally  $t_3, w_0$ . The total cost is 11.

(B) Because all pairs of  $w$ 's are connected by edges of cost zero, as are all pairs of  $t$ 's, the cost of the circuit is just the cost of passing between the top and bottom. That happens twice in each direction, and making that cost be less than or equal to the bound is the Assignment problem.

To show that  $\text{Assignment} \leq_p \text{Traveling Salesman}$ , we construct a reduction function  $f$ . It inputs a triple  $\langle W, T, C \rangle$  of workers, tasks, and the costs. It constructs the bipartite graph with a bank of  $w$ 's on one side, and a bank of  $t$ 's on the other. Every pair of  $w$ 's is connected by an edge of cost zero, as is every

pair of  $t$ 's. Between each  $t_i$  and  $w_j$  is an edge of cost  $C(t_i, w_j)$ . The output of the reduction function is  $f((W, T, C)) = \mathcal{G}$ . This is computable in polytime, and also an optimum solution for the Assignment problem instance corresponds to an optimum solution to the Traveling Salesman problem instance.

**V.6.25**

(A) We must show that  $\mathcal{L} \leq_p \mathcal{L}$  for any  $\mathcal{L} \in \mathcal{P}(\Sigma^*)$ . Take the reduction function to be the identity,  $f(\sigma) = \sigma$ . Clearly it can be computed in polytime, and just as clearly  $\sigma \in \mathcal{L}$  if and only if  $f(\sigma) \in \mathcal{L}$ .

For transitivity, let  $\mathcal{L}_0 \leq_p \mathcal{L}_1$  via the function  $f$ , and let  $\mathcal{L}_1 \leq_p \mathcal{L}_2$  via the function  $g$ , which are polytime computable. Then for all  $\sigma \in \mathbb{B}^*$  we have  $\sigma \in \mathcal{L}_0$  if and only if  $f(\sigma) \in \mathcal{L}_1$ , and  $f(\sigma) \in \mathcal{L}_1$  if and only if  $g \circ f(\sigma) \in \mathcal{L}_2$ . Because  $f, g$  are polytime,  $g \circ f$  is polytime also.

(B) Suppose that  $\mathcal{L} \leq_p \emptyset$ . Then there is a function  $f$  such that  $\sigma \in \mathcal{L}$  if and only if  $f(\sigma) \in \emptyset$ . Since  $f(\sigma)$  is never an element of the empty set,  $\sigma$  is never an element of  $\mathcal{L}$ . That is,  $\mathcal{L}$  is empty.

The argument that  $\mathcal{L} \leq_p \mathbb{B}^*$  implies that  $\mathcal{L} = \mathbb{B}^*$  is just as easy.

(C) Assume that  $\mathcal{L}_0 \leq_p \mathcal{L}_1$ , so that there is a polytime function  $f: \mathbb{B}^* \rightarrow \mathbb{B}^*$  such that  $\sigma \in \mathcal{L}_0$  if and only if  $f(\sigma) \in \mathcal{L}_1$ . We will use this reduction to show that  $\mathcal{L}_1 \in \mathbf{NP}$  implies that  $\mathcal{L}_0 \in \mathbf{NP}$ .

If  $\mathcal{L}_1 \in \mathbf{NP}$  then there is a nondeterministic Turing machine  $\mathcal{P}_1$  that decides  $\mathcal{L}_1$ . For the nondeterministic machine  $\mathcal{P}_0$  that decided  $\mathcal{L}_0$ , start with an input candidate string  $\tau$ . Compute  $f(\tau)$ . Run  $\mathcal{P}_1$  on input  $f(\tau)$ . If it accepts then  $\mathcal{P}_0$  accepts, and if it rejects then  $\mathcal{P}_0$  rejects.

**V.6.26**

(A) Take  $\mathcal{L}_0$  to be the language of strings whose length is a multiple of four, and  $\mathcal{L}_1$  to be the language of strings whose length is even. Clearly both languages are in  $\mathbf{P}$ , so  $\mathcal{L}_0 \leq_p \mathcal{L}_1$ .

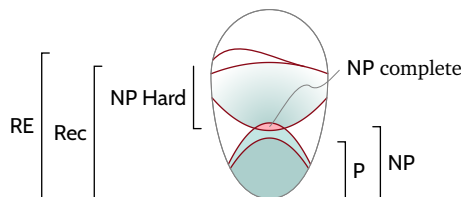
(B) Take  $\mathcal{L}_0$  to be the language of even-length strings, and  $\mathcal{L}_1$  to be the language of odd-length strings. Clearly both languages are in  $\mathbf{P}$ , so  $\mathcal{L}_0 \leq_p \mathcal{L}_1$ .

(C) Take  $\mathcal{L}_0$  to be the language of even-length strings and  $\mathcal{L}_1 = \mathbb{B}^*$ . Because some strings are elements of it and some are not, the set  $\mathcal{L}_0$  does not reduce to  $\mathbb{B}^*$ .

(D) Take  $\mathcal{L}_0$  to be the language of even-length strings and take  $\mathcal{L}_1 = \emptyset$ . Because some strings are elements of it and some are not, the set  $\mathcal{L}_0$  does not reduce to  $\mathbb{B}^*$ .

**V.6.27** It is not possible for  $\mathcal{L}_0 \leq_p \mathcal{L}_2$  or for  $\mathcal{L}_1 \leq_p \mathcal{L}_2$

**V.7.12** Here is the figure again.



(A) The language  $K$  is at the top of RE, in the upper left.

(B) The empty set,  $\emptyset$ , is all the way at the bottom, in the middle.

(C) There are a variety of algorithms for the shortest path problem, but all are fast (linear or not much slower), so the language  $\mathcal{L}_B$  is in  $\mathbf{P}$  but close to its bottom, just above where  $\emptyset$  is.

(D) Because  $SAT$  is NP complete, it is all the way at the top of the NP problems.

**V.7.13** The number one mistake is that “not computable in polynomial time” isn’t the criteria for membership in NP. Instead, the criteria for a problem being a member of NP is that it has a polytime verifier.

Another thing wrong is that the statement seems to confuse NP with the set difference  $\mathbf{NP} - \mathbf{P}$ . That is, the speaker seems to mean something like, “experts suspect that Satisfiability is a problem in NP but not in P.” (Along with that, the speaker likely means NP complete, not just an element of NP.)

One more thing wrong is that the statement seems to say that the problem is in that class because there is not algorithm yet known. It is not a question of whether we know of such an algorithm, it is a question of whether one exists.

**V.7.14** This algorithm is indeed exponential, not polynomial. But how to prove that this is the best possible algorithm? No one knows.

**V.7.15** In saying that the problem has a polytime algorithm, your friend is saying that there is a deterministic

Turing machine that solves this problem and that runs in polytime. But a deterministic Turing machine is a special case of a nondeterministic one. That is,  $P \subseteq NP$ , so this problem is in NP.

**V.7.16** Computational classes such as NP do not contain algorithms, they contain languages. (We don't distinguish too sharply between problems and languages, and so may say that a problem is in a complexity class, but we do distinguish between languages and algorithms to decide those languages.)

**V.7.17**

- (A) The statement “NP is a subset of NP complete” is false. Rather, NP complete is a subset of NP. The other part, that NP complete is a subset of NP hard is true. It is the intersection of NP hard and NP.
- (B) This is a false statement, because nondeterministic machines are not a specialization. Rather, nondeterministic machines are a generalization of deterministic machines, that is, any deterministic machine is a nondeterministic one.

However, the clause about subset is trickier. We know that  $P \subseteq NP$ , which is the opposite direction than the one stated. Most experts judge that NP is not a subset of P, but of course we have no proof.

**V.7.18**

- (A) True. Because Primality  $\in NP$ , it can be solved on a nondeterministic Turing machine. We can simulate such a machine with a deterministic one, and so we can solve the problem on a “regular” computer. (Of course, the solution algorithm may be quite slow.)
- (B) False; we cannot infer this from the given facts. It may be that there is an efficient algorithm, but we don't know. (We don't know from the statement as given, and we also don't know in general.)
- (C) False. The Traveling Salesman problem is NP complete. Thus, from Prime Factorization  $\in NP$ , we can deduce that Prime Factorization  $\leq_p$  Traveling Salesman, and from an efficient algorithm for Traveling Salesman we would get an efficient algorithm for Prime Factorization. But this is the reverse of what the question asks. The question says that Prime Factorization is not known to be NP complete, so we don't know the other direction.

- V.7.19** (A) No, from  $\mathcal{L}_1 \leq_p \mathcal{L}_0$  and  $\mathcal{L}_0$  is NP complete, you cannot conclude that  $\mathcal{L}_1$  is NP complete. An example is where  $\mathcal{L}_0$  is the Satisfiability problem, and  $\mathcal{L}_1$  is the problem of determining whether an input number is even.
- (B) False. It may be that  $\mathcal{L}_0$  is not a member of NP. (C) False. This is a reprise of the first item: from  $\mathcal{L}_1 \leq_p \mathcal{L}_0$  and  $\mathcal{L}_0$  is NP complete, you cannot conclude that  $\mathcal{L}_1$  is NP complete, even knowing that  $\mathcal{L}_1 \in NP$ . An example is where  $\mathcal{L}_0$  is the Satisfiability problem, and  $\mathcal{L}_1$  is the problem of determining whether an input number is even.
- (D) True; this is Lemma 7.5. If  $\mathcal{L}_1$  is NP complete then any  $\mathcal{L} \in NP$  is reducible to  $\mathcal{L}_1$ , so  $\mathcal{L} \leq_p \mathcal{L}_1$ , and transitivity of  $\leq_p$  gives that  $\mathcal{L} \leq_p \mathcal{L}_0$ . That, along with the statement that  $\mathcal{L}_0 \in NP$ , gives that  $\mathcal{L}_0$  is NP complete.
- (E) False. A problem is NP complete if it is in NP and every problem  $\mathcal{L}$  in NP reduces to it. So two NP complete problems reduce to each other. For example, any of the problems on the Basic List are reducible to any of the others, or to Satisfiability. (F) False. An example is that  $\mathcal{L}_1$  is the problem of determining whether a number is even, and  $\mathcal{L}_0$  is a problem whose fastest solution algorithm is exponential (such as chess).
- (G) True. This is Lemma 6.4, that P is closed downward.

**V.7.20**

- (A) Clearly  $\mathcal{L} = \{n \mid n \text{ is even}\}$  is in P. That implies it is in NP. Showing that  $\mathcal{L}$  is NP complete would show that  $P = NP$ .
- (B) This is like the prior item, except that it is not as trivially a member of P. To see that it is a member of P, just note that there is an algorithm to check all 4-tuples with a nested loop.

**V.7.21** Fix a language  $\mathcal{L} \in P$  that is nontrivial, so  $\mathcal{L} \neq \emptyset$  and  $\mathcal{L} \neq \mathbb{N}$ . Because  $P \subseteq NP$  we have  $\mathcal{L} \in NP$ . What remains is to show that  $\mathcal{L}$  is NP hard, that every language  $\hat{\mathcal{L}} \in NP$  satisfies that  $\hat{\mathcal{L}} \leq_p \mathcal{L}$ . But if  $P = NP$  then  $\hat{\mathcal{L}} \in P$ , and Lemma 6.4 says that every nontrivial computable language is P hard, so  $\hat{\mathcal{L}} \leq_p \mathcal{L}$ .

**V.7.22** (A) False; for example, the brute force algorithm solves all instances. (B) True, where we use Cobham's Thesis to take ‘quickly’ to mean polytime. (C) False. If Traveling Salesman  $\in P$  then every NP problem is in P, because every NP problem reduces to Traveling Salesman. That contradicts the  $P \neq NP$  assumption. (D) False. This would imply that Traveling Salesman is in P, which the prior item says is impossible.

**V.7.23** The obvious thing to do is to prove that 3-Satisfiability  $\leq_p$  4-Satisfiability. Briefly, given a propositional logic expression in which the clauses have three literals, such as  $P_i \vee P_j \vee \neg P_k$ , convert it into an expression in which the clauses have four literals by repeating the final literal, as in  $P_i \vee P_j \vee \neg P_k \vee \neg P_k$ .

## V.7.24

(A) In short, the witness  $\omega$  is the path.

The longer version is that, to apply Lemma 5.9, we will produce a deterministic Turing machine verifier,  $\mathcal{V}$ . It inputs pairs  $\langle \mathcal{G}, \omega \rangle$ , and must satisfy that if  $\mathcal{G}$  has a Hamiltonian path then there is a witness  $\omega$  such that  $\mathcal{V}$  accepts the input pair, while if  $\mathcal{G}$  does not have a Hamiltonian path then no witness will result in the pair being accepted. The verifier must run in polytime.

The verifier here interprets the witness  $\omega$  to be a Hamiltonian path through the graph, so it checks that it is a legal path in the given graph and that every vertex is in the path. If there is such a path then there is a witness that will check out. If there is no such path then no witness will suffice. The verifier clearly runs in polytime (the size of a graph is polynomial in the number of vertices since the number of edges is  $\mathcal{O}(|\mathcal{V}|)$ ).

(B) A Hamiltonian path is  $\langle v_0, v_7, v_1, v_2, v_3, v_6, v_5, v_4, v_8 \rangle$ . What we can say about  $v_0$  and  $v_8$  is that, because they have degree 1, they must be the start and end of the Hamiltonian path.

(C) We must produce a reduction function  $f$ . It must input a graph  $\mathcal{G}$ , outputs a graph  $\mathcal{H}$ , and have the property that  $\mathcal{G}$  has a Hamiltonian circuit if and only if  $\mathcal{H}$  has a Hamiltonian path.

Given  $\mathcal{G}$ , we form the graph  $\mathcal{H}$  by adding three vertices. First, fix a vertex  $v \in \mathcal{G}$ . Add a new vertex  $\hat{v} \in \mathcal{H}$  that has the same connections as  $v$ . That is, for every edge  $v_i v$  or  $v v_j$ , add  $\hat{v} v_i$  or  $\hat{v} v_j$ . Also add two vertices of degree one:  $w$  connects only to  $v$ , and  $\hat{w}$  connects only to  $\hat{v}$ .

We must show that there is a Hamiltonian path in  $\mathcal{H}$  if and only if there is a Hamiltonian circuit in  $\mathcal{G}$ . If there is a Hamiltonian circuit in  $\mathcal{G}$  then the Hamiltonian path in  $\mathcal{H}$  is derived by splitting the circuit at  $v$ , and hooking it to a path in  $\mathcal{H}$  that starts at  $w$ , goes to  $v$ , then through the circuit, out to  $\hat{v}$ , and then to  $\hat{w}$ .

For the other direction suppose that there is a Hamiltonian path in  $\mathcal{H}$ . Then, as in the prior item, its start and end must be  $w$  and  $\hat{w}$ . Further, it must then proceed to  $v$  and  $\hat{v}$ . Because  $\hat{v}$  is a copy of  $v$ , a Hamiltonian path in  $\mathcal{H}$  that starts off at these vertices corresponds to a Hamiltonian circuit in  $\mathcal{G}$ .

(D) Because the Hamiltonian Circuit problem is on the list of Basic NP Complete Problems, the fact that it reduces to Hamiltonian Path, and the latter is a member of NP, means that Hamiltonian Path is NP complete.

## V.7.25

(A) There is a Hamiltonian path:  $\langle q_3, q_6, q_4, q_7, q_8, q_5, q_0, q_1, q_2 \rangle$ .

(B) Take  $B \in \mathbb{N}$  to be the parameter and consider this family of languages.

$$\mathcal{L}_B = \{ \langle \mathcal{G}, B \rangle \mid \mathcal{G} \text{ has a simple path of length at least } B \}$$

For the reduction, the witness  $\omega$  is the path. That is, consider a Turing machine verifier  $\mathcal{V}$  that inputs  $\langle \mathcal{G}, B, \omega \rangle$ . It accepts the triple if and only if  $\omega$  represents a path a simple path through the graph, of length at least  $B$ . Clearly that can be done in polytime. If  $\langle \mathcal{G}, B \rangle \in \mathcal{L}_B$  then there is such a witness  $\omega$ , so  $\mathcal{V}$  accepts the input. If  $\langle \mathcal{G}, B \rangle \notin \mathcal{L}_B$  then there is no such witness, and so the verifier  $\mathcal{V}$  does not accept the triple.

(C) Given an instance  $\mathcal{G}$  of the Hamiltonian Path problem the reduction function converts it into the pair  $\langle \mathcal{G}, |\mathcal{G}| - 1 \rangle$ . Obviously that computation can be done in polytime. Further, where a graph has  $n$ -many vertices, a simple path containing  $|\mathcal{G}| - 1$  edges must be Hamiltonian.

(D) The prior exercise shows that the Hamiltonian Path problem is NP complete, and that, along with the other parts of this exercise, gives that Longest Path is NP complete.

## V.7.26

(A) The sum of elements in  $\hat{S} = \{6, 7, 19\}$  equals the sum of elements in  $S - \hat{S} = \{3, 4, 12, 13\}$ .

(B) The sum of the elements of  $\hat{T} = \{4, 6, 7, 13\}$  is  $B = 30$ .

(C) If we can partition a set  $S \subset \mathbb{N}$  into  $\hat{S} \subset S$  and  $S - \hat{S}$  with equal sums, then the total of the elements of  $S$  is twice the sum of the elements of  $\hat{S}$ , and so is even.

(D) This is the Subset Sum problem.

$$\mathcal{L}_0 = \{ \langle T, B \rangle \mid \text{a subset } \hat{T} \subseteq T \text{ has } \sum_{t \in \hat{T}} t = B \}$$

This is the Partition problem.

$$\mathcal{L}_1 = \{ \langle S \rangle \mid \text{there is } \hat{S} \subset S \text{ with } \sum_{s \in \hat{S}} s = \sum_{s \in S - \hat{S}} s \}$$

(E) We must produce a polytime computable function that takes in  $\sigma = \langle S \rangle$  and outputs  $f(\sigma) = \langle T, B \rangle$ , such that  $\sigma \in \mathcal{L}_0$  if and only if  $f(\sigma) \in \mathcal{L}_1$ .

Given  $\sigma = \langle S \rangle$ , consider the sum of its elements  $x = \sum_{s \in S} s$ . Where

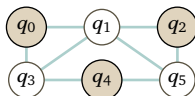
$$f(\sigma) = \begin{cases} \langle S, x/2 \rangle & \text{– if } x \text{ is even} \\ \langle S, x + 1 \rangle & \text{– otherwise} \end{cases}$$

we have that  $\sigma \in \mathcal{L}_1$  if and only if  $f(\sigma) \in \mathcal{L}_0$ , as required (and  $f$  is clearly computable in polytime).

(F) Partition is one of the NP complete sets listed in the section. So Subset Sum is NP hard. But Subset Sum is clearly NP; as a witness  $\omega$  we can use the subset.

### V.7.27

(A) The natural independent set is this.



(B) This is the language.

$$\mathcal{L}_0 = \{ \langle \mathcal{G}, B \rangle \mid \mathcal{G} \text{ has at least } B\text{-many disconnected vertices} \}$$

(C) One way to satisfy it is by taking  $P_0 = T, P_1 = P_2 = P_3 = F$ .

(D) This is the language.

$$\mathcal{L}_1 = \{ E \mid E \text{ is a satisfiable expression with at most 3 literals per clause} \}$$

(E) The graph associated with the expression is this.



(F) We must produce a reduction function,  $f$ . It inputs an instance of the 3-Satisfiability problem,  $E$ . It constructs a graph  $\mathcal{G}$  as in the prior item. That is, for the  $j$ -th clause it adds three vertices, labeling with  $v_{j,i}$  if the literal is  $P_i$ , and with  $\bar{v}_{j,i}$  if the literal is  $\neg P_i$ . It connects these three vertices completely, making a triangle. Further, the reduction function connects any two vertices labeled with the same index  $i$  and where one has an overline and the other does not, that is, it connects all pairs of  $v_{j_0,i}$  and  $\bar{v}_{j_1,i}$ . Clearly this construction can happen in polytime.

The expression  $E$  is satisfiable if and only if in each clause at least one literal evaluates to  $T$ . Let  $S$  be a minimal such set of literals. Consider the associated vertices in the graph  $\mathcal{G}$ . In each triangle there must be at least one vertex associated with a literal from  $S$  because in each clause there must be a literal from  $S$  that evaluates to  $T$ . Further, for each literal in  $S$ , the negation evaluates to  $F$ , and so each associated vertex is not connected to an associated vertex in another triangle. Thus this forms an independent set of vertices. The converse works the same way.

**V.7.28** Move the head left, write a 1, and halt. This is a constant time algorithm.

**V.7.29** No. If we knew of a problem in  $\text{NP} - \text{P}$  then we'd know that  $\text{P} \neq \text{NP}$ . We don't know that. (Experts suspect that some problems are in this area, such as the Vertex-to-Vertex Path problem, but this is guesswork.)

**V.7.30** Let  $\mathcal{L}$  be any NP complete language of bitstrings. Take  $\mathcal{L}_2 = \{ \emptyset \hat{\ } \sigma \mid \sigma \in \mathcal{L} \}$ , and  $\mathcal{L}_1 = \{ \emptyset \hat{\ } \sigma \mid \sigma \in \Sigma^* \}$ , and  $\mathcal{L}_0 = \mathcal{L}_1 \cup \{ 1 \hat{\ } \sigma \mid \sigma \in \mathcal{L} \}$ . (The language  $\mathcal{L}_1$  is an element of  $\text{P}$  because testing membership is as simple as testing whether the first character is  $\emptyset$ .)

**V.7.31** Assume that  $\mathcal{L}_0$  is NP complete. Assume also that  $\mathcal{L}_0 \leq_p \mathcal{L}_1$  and  $\mathcal{L}_1 \in \text{NP}$ . To verify that  $\mathcal{L}_1$  is NP complete we must show that it is NP hard, that for every  $\mathcal{L} \in \text{NP}$  we have  $\mathcal{L} \leq_p \mathcal{L}_1$ .

The assumption that  $\mathcal{L}_0$  is NP complete gives  $\mathcal{L} \leq_p \mathcal{L}_0$ . Thus  $\mathcal{L} \leq_p \mathcal{L}_0 \leq_p \mathcal{L}_1$  and transitivity of polytime reduction from Lemma 6.4 gives that  $\mathcal{L} \leq_p \mathcal{L}_1$ .

### V.7.32

- (A) Fix  $\mathcal{L}, \hat{\mathcal{L}} \in \text{NP}$ . There are associated polytime verifiers  $\mathcal{V}$  and  $\hat{\mathcal{V}}$ . A verifier for the union can just run those two on the input, and accept if either accepts.
- (B) Fix  $\mathcal{L}, \hat{\mathcal{L}} \in \text{NP}$ . There are associated polytime verifiers  $\mathcal{V}$  and  $\hat{\mathcal{V}}$ . A verifier for the concatenation inputs a string  $\sigma$ , and loops over splitting it into substrings  $\sigma = \sigma[0 : i] \hat{\ } \sigma[i : ]$  for  $i \in [0 .. |\sigma|)$ , testing whether both verifiers accept. Each verifier is polytime, and running the loop still keeps the overall algorithm inside of polytime.
- (C) The class **P** is closed under complement. Consequently, if **NP** is not closed under complement then **P**  $\neq$  **NP**. (But it is possible that **NP** is closed under complement but is still different than **P**.)

**V.7.33** Countable. It is a subset of **NP**, which is countable because a verifier is a deterministic Turing machine, and there are countably many such machines.

**V.7.34**

- (A) It is not a member of **NP** because it is not decidable.
- (B) Since  $\mathcal{L} \in \text{NP}$  there is an associated nondeterministic machine  $\mathcal{P}$  that decides membership. Consider the Turing machine  $\hat{\mathcal{P}}_{\mathcal{L}}$  that, given an input  $\tau$ , constructs the computation tree of  $\mathcal{P}_{\mathcal{L}}$  on input  $\tau$  and halts if and only if  $\mathcal{P}_{\mathcal{L}}$  accepts  $\tau$ .  
The reduction  $\mathcal{L} \leq_p \text{HP}$  will be done by a function  $f_{\mathcal{L}}$ . This function will, given  $\sigma$ , return the pair  $\hat{\mathcal{P}}, \sigma$  where  $\hat{\mathcal{P}}$ . (Note that this function is polynomial since, while running  $\hat{\mathcal{P}}_{\mathcal{L}}$  may not be polynomial, constructing it from  $\mathcal{P}$  is polynomial.)

**V.8.14**

- (A) The naive algorithm is to try all subsets. Where the set has  $n$  elements, there are  $2^n$  subsets.
- (B) The naive algorithm is to check each assignment of the  $k$  colors to the  $n$  vertices. There are  $k^n$  many assignments.

**V.8.15** First,  $n! \leq n^n = (2^{\lg n})^n = 2^{n \lg n} \leq 2^{n^2}$ . With that, the naive algorithm is to compute all circuits, and find the one of minimum cost. Where there are  $n$  cities, there are  $n!$  many circuits.

**V.8.16**

- (A) Googling gives  $10^{86}$ .
- (B)  $\lg(10^{86}) = 86 \cdot \lg(10) \approx 86 \cdot 3.32 = 285.52$
- (C) The average chess game is about 40 moves (see (SE author babou and various others 2013)). The longest game in the cited database is 277 moves, so 286 is a long chess game, but not completely out of range.

**V.8.17**

**V.A.11** They are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, and 97.

**V.A.12**

- (A)  $n = pq = 134, (p - 1) \cdot (q - 1) = 120$
- (B) Use  $e = 7$  (the three 2, 3, and 5 are not relatively prime).
- (C) The multiplicative inverse of  $e = 7$  modulo  $n = 134$  is  $d = 103$ .
- (D) The encryption is  $m^e \bmod n = 9^7 \bmod 143 = 48$ . The decryption is  $48^{103} \bmod 143 = 9$ .

**V.A.13**

- (A) With two factors  $n = k_0 \cdot k_1$  if one is greater than  $\sqrt{n}$  then the other must be less.
- (B) If  $n$  is a perfect square  $n = k \cdot k$  then its first nontrivial factor is  $k = \sqrt{n}$ .
- (C) The square root of  $10^{12}$  is  $10^6 = 1\,000\,000$ , a million.
- (D) The input  $n$  has size about  $\lg(n)$  bits. Thus  $\sqrt{n} \approx \sqrt{2^{\text{size of } n}} \approx 2^{(1/2) \cdot \text{size of } n}$ .

## Appendices

### A.1

- (A)  $\sigma \wedge \tau = 10110 \wedge 110111 = 10110110111$ .
- (B)  $\sigma \wedge \tau \wedge \sigma = 10110 \wedge 110111 \wedge 10110 = 101101101110110$
- (C)  $\sigma^R = 10110^R = 01101$
- (D)  $\sigma^3 = 10110 \wedge 10110 \wedge 10110 = 101101011010110$
- (E)  $\emptyset^3 \wedge \sigma = 000 \wedge 10110 = 00010110$

### A.2

- (A) abbca
- (B) ababbcabca
- (C) baacb
- (D) ababab

**A.3** There are  $2^4 = 16$  bit strings of length 4 and half of them start with 0, so there are 8 strings in the language.

### A.4

- (A) Yes.
- (B) Yes.
- (C) No.
- (D) Yes.

**A.5** Show that the length of a concatenation is the sum of the two lengths:  $|\sigma \wedge \tau| = |\langle s_0, \dots, s_{i-1}, t_0, \dots, t_{j-1} \rangle| = i + j = |\sigma| + |\tau|$ . Proving this by induction is routine.

**A.7** One example using  $\mathbb{B}$  is  $\sigma = \langle 0 \rangle$  and  $\tau = \langle 1 \rangle$ . Then  $\sigma \wedge \tau = \langle 0, 1 \rangle$  while  $\tau \wedge \sigma = \langle 1, 0 \rangle$ .

### B.1

- (A) For one-to-one, suppose that  $f(x_0) = f(x_1)$  for  $x_0, x_1 \in \mathbb{R}$ . Then  $3x_0 + 1 = 3x_1 + 1$ . Subtract the 1's and divide by 3 to conclude that  $x_0 = x_1$ .  
For onto, fix a member of the codomain  $c \in \mathbb{R}$ . Observe that  $d = (c - 1)/3$  is a member of the domain and that  $f(d) = 3 \cdot ((c - 1)/3) + 1 = (c - 1) + 1 = c$ . Thus  $f$  is onto.
- (B) The function  $g$  is not one-to-one because  $g(2) = g(-2)$ . It is not onto because no element of the domain  $\mathbb{R}$  is mapped by  $g$  to the element 0 of the codomain.

### B.2

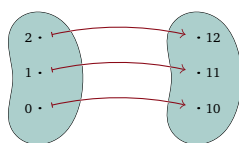
- (A) It is a left inverse because  $g \circ f$  has the action  $(x, y) \mapsto (x, y, 0) \mapsto (x, y)$  for all  $x$  and  $y$ . It is not a right inverse because there is an input on which the composition  $f \circ g$  is not the identity, namely the action is  $(1, 2, 3) \mapsto (1, 2) \mapsto (1, 2, 0)$ .
- (B) Observe that  $f(2) = f(-2) = 4$ . Any left inverse would have to map 4 to 2 and also to map 4 to  $-2$ . That would be not well-defined, so no function is the left inverse.
- (C) One right inverse is  $g_0: C \rightarrow D$  given by  $10 \mapsto 0, 11 \mapsto 1$ . A second is  $g_1: C \rightarrow D$  given by  $10 \mapsto 2, 11 \mapsto 3$ .

### B.3

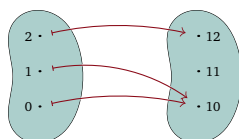
- (A) First, the domain of each equals the codomain of the other so their composition is defined in both ways. Next,  $g \circ f(a) = g(f(a)) = g(a + 3) = (a + 3) - 3 = a$  is the identity map. Similarly,  $f \circ g(a) = (a - 3) + 3$  is also the identity map. Thus they are inverse.
- (B) The inverse of  $h$  is itself. The domain and codomain are equal, as required. If  $n$  is odd then  $h \circ h(n) = h(n - 1) = (n - 1) + 1 = n$  (the second equality holds because  $n - 1$  is even in this case). Similarly if  $n$  is even then  $h \circ h(n) = (n + 1) - 1 = n$ .

- (c) The inverse of  $s$  is the map  $r: \mathbb{R}^+ \rightarrow \mathbb{R}^+$  given by  $r(x) = \sqrt{x}$  (that is, the positive root). The domain of each is the codomain of the other, and each of  $s \circ r$  and  $r \circ s$  is the identity function.

**B.4** This is a bean diagram of the function  $f$



and this is a diagram of  $g$ .

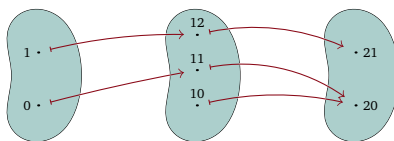


- (A) By inspection  $f$  is both one-to-one and onto.  
 (B) Reverse the association made by  $f$  so that  $f^{-1}(10) = 0$ ,  $f^{-1}(11) = 1$ , and  $f^{-1}(12) = 2$ .  
 (C) The map  $g$  is not one-to-one since  $g(0) = g(1)$ . (In addition,  $g$  is not onto since no input is sent to  $11 \in C$ .)  
 (D) If a  $h$  were the inverse of  $g$  then both  $h(10) = 0$  and  $h(10) = 1$ , which would violate the definition of a function.

**B.5**

- (A) Let  $f: D \rightarrow C$  and  $g: C \rightarrow B$  be one-to-one. Suppose that  $g \circ f(x_0) = g \circ f(x_1)$  for some  $x_0, x_1 \in D$ . Then  $g(f(x_0)) = g(f(x_1))$ . The function  $g$  is one-to-one and so, since the two values  $g(f(x_0))$  and  $g(f(x_1))$  are equal, the arguments  $f(x_0)$  and  $f(x_1)$  are also equal. The function  $f$  is one-to-one and so  $x_0 = x_1$ . Because  $g \circ f(x_0) = g \circ f(x_1)$  implies that  $x_0 = x_1$ , the composition is one-to-one.  
 (B) Let  $f: D \rightarrow C$  and  $g: C \rightarrow B$  be onto. If  $B$  is the empty set then by the definition of function,  $C = \emptyset$  also, and again by the definition of function, in turn  $D = \emptyset$ , and so the composition is onto, vacuously. Otherwise, fix an element of the codomain of the composition,  $b \in B$ . The function  $g$  is onto so there is a  $c \in C$  such that  $g(c) = b$ . The function  $f$  is onto so there is a  $d \in D$  such that  $f(d) = c$ . Then  $g \circ f(d) = b$ , and so the composition is onto.  
 (C) Let  $f: D \rightarrow C$  and  $g: C \rightarrow B$  be such that the composition  $g \circ f$  is one-to-one. Suppose that  $f$  is not one-to-one. Then there are  $d_0, d_1 \in D$  with  $d_0 \neq d_1$  such that  $f(d_0) = f(d_1)$ . But this implies that  $g(f(d_0)) = g(f(d_1))$ , contradicting that the composition is one-to-one.  
 (D) Let  $f: D \rightarrow C$  and  $g: C \rightarrow B$  be such that the composition  $g \circ f$  is onto. Suppose for a contradiction that  $g$  is not onto. Then for some  $b \in B$  there is no  $c \in C$  with  $g(c) = b$ . But this means that there is no  $d \in D$  such that  $b = g(f(d))$ , contradicting that the composition is onto. So  $g$  must be onto.  
 (E) The answer to both questions is “no.”

Let  $D = \{0, 1\}$ ,  $C = \{10, 11, 12\}$ , and  $B = \{20, 21\}$ . Let  $f$  and  $g$  be as shown below. Then  $g \circ f$  is onto because  $g \circ f(0) = 20$  and  $g \circ f(1) = 21$ . But  $f$  is not onto because no element of its domain  $D$  is mapped to the element 10 of its codomain  $C$ .



The same function is an example of a composition  $g \circ f$  that is one-to-one but where the function performed second,  $g$ , is not one-to-one.

**B.6**

- (A) We first prove that if the function  $f: D \rightarrow C$  has an inverse  $f^{-1}: C \rightarrow D$  then it must be a correspondence. To verify that  $f$  is one-to-one, suppose that  $d_0, d_1 \in D$  are such that  $f(d_0) = f(d_1) = c$  for some  $c \in C$ . Consider  $f^{-1}(c)$ . Because  $f^{-1} \circ f$  is the identity map on  $D$  we have both that  $f^{-1}(c) = f^{-1}(f(d_0)) = d_0$  and that  $f^{-1}(c) = f^{-1}(f(d_1)) = d_1$ . Thus  $d_0 = d_1$  and so  $f$  is one-to-one.

Next we verify that  $f$  is onto. Suppose otherwise, so that there is a  $c \in C$  that is not associated with any  $d \in D$ . The map  $f \circ f^{-1}$  cannot give  $c = f \circ f^{-1}(c) = f(f^{-1}(c))$  because  $c \notin \text{ran}(f)$ , so this is impossible. Therefore  $f$  is onto.

To finish, we prove that if a map  $f: D \rightarrow C$  is a correspondence then it has an inverse. We will define an association  $g: D \rightarrow C$  and show that it is well-defined. For each  $c \in C$ , let  $g(c)$  be the element  $d \in D$  such that  $f(d) = c$ ; note that there is such an element because  $f$  is onto, and that there is a unique such element because  $f$  is one-to-one. So  $g$  is well-defined. That  $g \circ f$  and  $f \circ g$  are identity maps is clear.

- (B) Suppose that  $f: D \rightarrow C$  has two inverses  $g_0, g_1: C \rightarrow D$ . We will show that the two make the same associations, that is, we will show that they have the same graph and thus are the same function.

Let  $c$  be an element of the domain  $C$  of the two. The prior item shows that  $f$  is onto so there exists a  $d \in D$  such that  $f(d) = c$ . The prior item also shows that  $f$  is one-to-one, so there is only one  $d$  with that property:  $d = g_0(c)$  and  $d = g_1(c)$ . Therefore  $g_0(c) = g_1(c)$ .

- (C) An inverse is a function that has an inverse: the inverse of  $f^{-1}$  is  $f$ . By the first item then,  $f^{-1}$  is a correspondence.
- (D) Here is the verification that one order gives the identity function  $(f^{-1} \circ g^{-1}) \circ (g \circ f)(x) = (f^{-1} \circ g^{-1})(g(f(x))) = f^{-1}(g^{-1}(g(f(x)))) = f^{-1}(f(x)) = x$ . Verification of the other order is similar.

### B.7

- (A) We will do induction on the number of elements in the range. Before the proof we give an illustration. Fix  $D = \{d_0, d_1, d_2, d_3\}$  and  $C = \{c_0, c_1, c_2\}$ . Consider the function  $f: D \rightarrow C$  given here, which is onto and so  $\text{ran}(f) = C$ .

$$d_0 \mapsto c_0 \quad d_1 \mapsto c_1 \quad d_2 \mapsto c_2 \quad d_3 \mapsto c_2$$

To do the induction, we pick an element of the range and omit it; suppose that we omit  $c_2$ . We then want to consider a restriction map that is onto  $C - \{c_2\}$ . For this map to be well-defined, we must omit from its domain all of the elements that map to  $c_2$ , the set  $f^{-1}(c_2) = \{d_2, d_3\}$ . We then have the function  $\hat{f}$  with domain  $\hat{D} = \{d_0, d_1\}$  and codomain  $\hat{C} = \{c_0, c_1\}$  given here.

$$d_0 \mapsto c_0 \quad d_1 \mapsto c_1$$

The inductive hypothesis gives that  $\hat{f}$ 's domain,  $\hat{D}$ , has at least as many elements as its range,  $\hat{C}$ . To finish the argument, we will add back the omitted elements to get  $f$  and conclude that its domain,  $D$ , has at least as many elements as its range,  $C$ .

Time for the proof. The base step is to consider a function  $f$  whose range is empty. The domain must also be empty since the definition of function requires that every element of the domain is mapped somewhere. Hence  $|\text{ran}(f)| \leq |D|$ , because both of them are zero.

Next assume that the statement holds for all functions  $f$  between finite sets, where  $|\text{ran}(f)| = 0$ ,  $|\text{ran}(f)| = 1, \dots$ ,  $|\text{ran}(f)| = k$ , for  $k \in \mathbb{N}$ . Consider a function with  $|\text{ran}(f)| = k + 1$ .

Fix some  $\hat{c} \in \text{ran}(f)$  (the range is not empty because  $k + 1 > 0$ ). Let  $\hat{C} = \text{ran}(f) - \{\hat{c}\}$ . Because  $\hat{c}$  is in the range of  $f$ , the set  $\hat{D} = f^{-1}(\hat{c}) = \{d \in D \mid f(d) = \hat{c}\}$  has at least one element, that is,  $|D - \hat{D}| + 1 \leq |D|$ . Now the key point: the map  $\hat{f}: D - \hat{D} \rightarrow \hat{C}$  is onto, as any element  $c \in \text{ran}(f)$  is the image of some  $d \in D$  and so any such element with  $c \neq \hat{c}$  must be the image of some  $d \in D - \hat{D}$ . Therefore the inductive hypothesis applies, giving that  $|\hat{C}| \leq |D - \hat{D}|$ . Finish by adding one,  $|\text{ran}(f)| = |\hat{C}| + 1 \leq |D - \hat{D}| + 1 \leq |D|$ .

*Alternate proof.* We can instead do induction on the number of elements in the domain,  $D$ . As above, we first illustrate the argument. Fix a domain  $D = \{d_0, d_1, d_2, d_3\}$  and codomain  $C = \{c_0, c_1, c_2\}$ . To do induction, we will omit an element of the domain, in this example  $d_3$ , giving  $\hat{D} = D - \{d_3\}$ . There are two cases. The first is that the omitted element  $d_3$  maps to the same member of the codomain as at least one other, not omitted, element of the domain. This case happens with this function  $f$ .

$$d_0 \mapsto c_0 \quad d_1 \mapsto c_1 \quad d_2 \mapsto c_2 \quad d_3 \mapsto c_2$$

Consider the restriction,  $\hat{f}: \hat{D} \rightarrow C$  given by  $\hat{f}(d) = f(d)$ . The inductive hypothesis applies, giving  $|\text{ran}(\hat{f})| \leq |\hat{D}|$ . In this first case,  $d_3$  is not the only element of  $D$  associated with  $c_2$ , so  $|\text{ran}(f)| = |\text{ran}(\hat{f})|$  and the desired conclusion is immediate,  $|\text{ran}(f)| = |\text{ran}(\hat{f})| \leq |\hat{D}| < |D|$ .

The other case is that the omitted domain element  $d_3$  does not map to the same member of the codomain as any other domain element. That happens for this function  $f$ .

$$d_0 \mapsto c_0 \quad d_1 \mapsto c_1 \quad d_2 \mapsto c_2 \quad d_3 \mapsto c_3$$

To do the induction, we omit  $d_3$  from the domain, giving  $\hat{D} = D - \{d_3\}$ . Consider the restriction map  $\hat{f}: \hat{D} \rightarrow C$  defined by  $\hat{f}(d) = f(d)$ . The inductive hypothesis applies, so  $|\text{ran}(\hat{f})| \leq |\hat{D}|$ . Then adding 1 to both sides gives  $|\text{ran}(f)| = |\text{ran}(\hat{f})| + 1 \leq |\hat{D}| + 1 = |D|$ .

We are ready for the proof. The base step is to consider a function  $f$  whose domain that  $D$  has no elements at all. The only function with an empty domain is the empty function, whose range is empty, so  $0 = |\text{ran}(f)| \leq |D| = 0$ .

For the inductive step, assume that the statement is true for functions between finite sets when  $|D| = 0, |D| = 1, \dots, |D| = k$ , with  $k \in \mathbb{N}$ . Consider a function  $f$  where  $|D| = k + 1$ .

Fix some  $\hat{d} \in D$  (there must be such an element since the domain is not empty, as  $k + 1 > 0$ ). Let  $\hat{D} = D - \{\hat{d}\}$  and let  $\hat{f}: \hat{D} \rightarrow C$  be the restriction of  $f$  to  $\hat{D}$ , defined by  $\hat{f}(d) = f(d)$ . The inductive hypothesis gives  $|\text{ran}(\hat{f})| \leq |\hat{D}|$ .

Consider the codomain element  $\hat{c} = f(\hat{d})$ . There are two cases: either (1)  $\hat{c}$  is also the image of another domain element, that is, there is a  $d \in \hat{D}$  with  $d \neq \hat{d}$  and  $f(d) = \hat{c}$ , or else (2) it is not. In the first case,  $\text{ran}(f) = \text{ran}(\hat{f})$  and we have  $|\text{ran}(f)| = |\text{ran}(\hat{f})| \leq |\hat{D}| < |D|$ . In the second case,  $\text{ran}(f) = \text{ran}(\hat{f}) + 1$  and we have  $|\text{ran}(f)| = |\text{ran}(\hat{f})| + 1 \leq |\hat{D}| + 1 = |D|$ .

(B) We will use induction on the number of elements in the range. The base step is that the range is empty. The only function with an empty range is the empty function, which has an empty domain, and so  $0 = |\text{ran}(f)| = |D| = 0$ .

For the inductive step, assume that the statement is true for any one-to-one function  $f$  between finite sets with  $|\text{ran}(f)| = 0, |\text{ran}(f)| = 1, \dots, |\text{ran}(f)| = k$ , where  $k \in \mathbb{N}$ . Consider a one-to-one function  $f$  with  $|\text{ran}(f)| = k + 1$ .

Fix  $\hat{c} \in \text{ran}(f)$  (such an element exists because  $k + 1 > 0$ ). Let  $\hat{C} = C - \{\hat{c}\}$ . Because  $\hat{c}$  is an element of the range of  $f$ , there is a domain element that maps to it. Because  $f$  is one-to-one, there is only one such element. Call it  $\hat{d}$  and let  $\hat{D} = D - \{\hat{d}\}$ .

The restriction function  $\hat{f}: \hat{D} \rightarrow \hat{C}$  is defined by  $\hat{f}(d) = f(d)$ . This map is also one-to-one: thinking of  $f$  as a set of ordered pairs, the assumption that it is one-to-one means that no two pairs have the same second element, and the restriction  $\hat{f}$  is a subset of  $f$  so it also has no two with the same second element. By the inductive hypothesis,  $|\text{ran}(\hat{f})| = |\hat{D}|$ . Adding 1 gives  $|\text{ran}(f)| = |\text{ran}(\hat{f})| + 1 = |\hat{D}| + 1 = |D|$ .

*Alternate proof.* We can instead do induction on the number of elements in the domain  $D$ . The base step is that the domain is empty,  $|D| = 0$ . The only function with an empty domain is the empty function and we have  $0 = |\text{ran}(f)| = |D| = 0$ .

For the inductive step, assume that the statement is true for any one-to-one function between finite sets with  $|D| = 0, \dots, |D| = k$ , where  $k \in \mathbb{N}$ . Consider a one-to-one function  $f$  whose domain has  $|D| = k + 1$ .

Fix some  $\hat{d} \in D$  (the set  $D$  is not empty because  $k + 1 > 0$ ) and let  $\hat{D} = D - \{\hat{d}\}$ . Consider  $\hat{c} = f(\hat{d})$ . Because  $f$  is one-to-one,  $\hat{c}$  is not the image of any element of the domain other than  $\hat{d}$ . So the restriction map  $\hat{f}: \hat{D} \rightarrow C$  has one less element in its range,  $\text{ran}(f) = \text{ran}(\hat{f}) + 1$ .

The key point is that because  $f$  is one-to-one,  $\hat{f}$  is also one-to-one: thinking of  $f$  as a set of ordered pairs, one-to-one-ness means that no two pairs have the same second element, and  $\hat{f}$  is a subset of  $f$  so it has the same property. Therefore the inductive hypothesis applies, giving  $|\text{ran}(\hat{f})| = |\hat{D}|$ . Add 1 to get the desired conclusion, that  $|\text{ran}(f)| = |\text{ran}(\hat{f})| + 1 = |\hat{D}| + 1 = |D|$ .