School of Computing and Information Systems
COMP30026 Models of Computation
Week 1: Informal Proof

If you finish the exercises early, start on the homework problems!

Exercises

T1.1 (Post's Correspondence Problem) Let's play a game. We have a finite set of "dominoes" such as

$$\left\{\boxed{\frac{\texttt{b}}{\texttt{ca}}}, \boxed{\frac{\texttt{a}}{\texttt{ab}}}, \boxed{\frac{\texttt{ca}}{\texttt{a}}}, \boxed{\frac{\texttt{abc}}{\texttt{c}}}\right\}$$

Each domino has a string written in the upper half, and one in the lower half. The goal is to find a sequence of dominoes which, when laid side by side, spell out identical non-empty strings across the top and the bottom (or alternatively, determine that no solution is possible). You do not have to use every domino, and any domino can be used multiple times.

For example, the set given above has a solution, namely

$$\boxed{\frac{\texttt{a}}{\texttt{ab}}}\boxed{\frac{\texttt{b}}{\texttt{ca}}}\boxed{\frac{\texttt{ca}}{\texttt{a}}}\boxed{\frac{\texttt{a}}{\texttt{ab}}}\boxed{\frac{\texttt{abc}}{\texttt{c}}}$$

(a) Can you find a solution to $\left\{\boxed{\frac{\texttt{baa}}{\texttt{abaaa}}}, \boxed{\frac{\texttt{aaa}}{\texttt{aa}}}\right\}$?

Solution: $\boxed{\frac{\texttt{aaa}}{\texttt{aa}}}\boxed{\frac{\texttt{baa}}{\texttt{abaaa}}}\boxed{\frac{\texttt{aaa}}{\texttt{aa}}}$

(b) How about $\left\{\boxed{\frac{\texttt{a}}{\texttt{cb}}}, \boxed{\frac{\texttt{bc}}{\texttt{ba}}}, \boxed{\frac{\texttt{c}}{\texttt{aa}}}, \boxed{\frac{\texttt{abc}}{\texttt{c}}}\right\}$?

Solution: No solution is possible. Why?

(c) How about $\left\{\boxed{\frac{\texttt{ab}}{\texttt{aba}}}, \boxed{\frac{\texttt{bba}}{\texttt{aa}}}, \boxed{\frac{\texttt{aba}}{\texttt{bab}}}\right\}$?

Solution: No solution is possible. Why?

T1.2 Three playing cards lie face down on a table. One is red, one is black, and one is the joker (which is neither red nor black). Each card has a sentence on its back:

Card 1 Card 2 Card 3

The red card has a true sentence written on its back and the black card has a false sentence.

(a) Determine which card is red, which is black, and which is the joker.

(b) What does it mean for your answer to be correct? How can you check the correctness of your answer? Discuss.

Solution: An answer is correct if it doesn't contradict any of the facts. So, let's start by writing down all the facts. We will simplify a little by eliminating any references to the truth or falsehood of other statements:

i. Each card is exactly one of the following: red, black, or a joker.
ii. One of cards 1, 2, or 3 is red.
iii. One of cards 1, 2, or 3 is black.
iv. One of cards 1, 2, or 3 is the joker.
v. If card 1 is red, then card 3 is the joker.
vi. If card 1 is black, then card 3 is not the joker.
vii. If card 2 is red, then card 1 is black.
viii. If card 2 is black, then card 1 is not black.
ix. If card 3 is red, then it is the joker.
x. If card 3 is black, then it is not the joker.

Now, to check an answer, it suffices to check that it doesn't contradict any of these statements.

(c) Are there multiple answers? If so, give another answer and check its correctness. Otherwise, give an informal proof (in prose) of the uniqueness of your answer.

Solution: Card 3 cannot be red, because the sentence on a red card must be true, but card 3 cannot be both red and the joker. Therefore either card 1 or card 2 is red.

If card 2 is red, then card 1 must be black and hence card 3 is the joker by elimination. However, this means that the sentence on card 1 is true, which cannot happen, because card 1 is black. This is a contradiction, so card 2 is not red.

Since neither of cards 2 or 3 is red, by elimination, card 1 must be red. Then card 3 is the joker, and hence by elimination card 2 is black.

Homework problems

P1.1 For each case from T1.1 which has no solution, write a proof of that fact. Try to make your answer as short as possible, but do not sacrifice rigour.

P1.2 Consider a square grid with x columns and y rows. We want to find out how many different paths there are that one can travel along, starting from the bottom left and ending in the top right cell, given that *one can only move up or right*. Let us call that value $f(x, y)$.

(a) Write down a table of the values of $f(x, y)$ when x and y range from 1 to 4. Draw the table as a grid, so that $f(1, 1)$ is in the lower left corner, $f(2, 1)$ is immediately to the right of it, and so on.

(b) Discuss what we mean by "path". How might we represent it mathematically?

(c) Propose and justify two different formulas for calculating $f(x, y)$. What are the benefits and drawbacks of each option? *Hint:* Think dynamic programming!

Solution: One option is to use a recursive approach:

$$f(x, y) = \begin{cases} 1 & \text{if } x = 1 \text{ or } y = 1, \\ f(x-1, y) + f(x, y-1) & \text{otherwise.} \end{cases}$$

This is correct because:

i. There is only one path from $(1,1)$ to $(x,1)$. That is, moving right $x-1$ times.

ii. Similarly, there is only one path from $(1,1)$ to $(y,1)$.

iii. To reach (x,y) when $x,y>1$, you must first reach either $(x-1,y)$ or $(x,y-1)$, and in either case only one move gets you to (x,y).

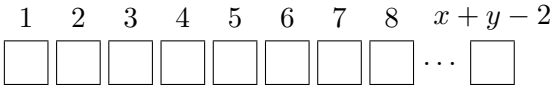
Note that if you try calculate $f(x,y)$ by recursively expanding this formula, it will take an exponential number of steps. However, almost all of those steps are repeated work. Can you think of a way to optimize the evaluation?

This is a more elegant formula:

$$f(x,y) = \frac{(x+y-2)!}{(x-1)!(y-1)!} = \binom{x+y-2}{x-1}.$$

To explain this, consider that every path to (x,y) consists of exactly $x-1$ rightward steps and exactly $y-1$ upward steps. The total number of steps is thus $x+y-2$.

Now, draw an array of $x+y-2$ empty slots:



Then, if we fill in $x-1$ of them with R's for (for *right*), and the rest with U's for (for *up*), we will get a path to (x,y). And vice versa: every path corresponds to a unique string of R's and U's. (That is, there is a *bijection* between these paths and such strings.)

So, how many ways can we do this? Well, how many ways are there to choose $x-1$ slots from a set of $x+y-2$ slots?

(d) Prove by induction that the two formulas are equivalent.

P1.3 Put each premise and deduction in your answer to T1.2c on a separate line of a numbered list.

(a) Next to each premise, write down that it is a premise. Next to each deduction, write down the numbers of the deductions or premises that it follows from.

(b) Split each nontrivial deduction into simpler (possibly still nontrivial) deductions. Repeat the process until the entire proof consists of premises and trivial deductions.

Hint: Your proof can have subproofs. If your original answer proves $\neg P$ by contradiction, convert that into a subproof of $\bot$ whose premise is P. Then you can conclude $\neg P$ from your subproof by *reductio ad absurdum*.

Solution:

(1) Each card is exactly one of the following: red, black, or a joker. (premise)

(2) One of cards 1, 2, or 3 is red. (premise)

(3) One of cards 1, 2, or 3 is black. (premise)

(4) One of cards 1, 2, or 3 is the joker. (premise)

(5) If card 1 is red, then card 3 is the joker. (premise)

(6) If card 1 is black, then card 3 is not the joker. (premise)

(7) If card 2 is red, then card 1 is black. (premise)

(8) If card 2 is black, then card 1 is not black. (premise)

(9) If card 3 is red, then it is the joker. (premise)

(10) If card 3 is black, then it is not the joker. (premise)

(11) Subproof:
   i. Card 3 is red. (assume)
   ii. Card 3 is exactly one of the following: red, black, or a joker. (1)
   iii. Card 3 is not the joker. (P1.3(11)ii, P1.3(11)i)
   iv. Card 3 is the joker. (9, P1.3(11)i)
   v. Contradiction. (P1.3(11)iv, P1.3(11)iii)

(12) Card 3 is not red. (11, proof by contradiction)

(13) Subproof:
   i. Card 2 is red. (assume)
   ii. Card 1 is black. (7, P1.3(13)i)
   iii. Card 3 is not the joker. (6, P1.3(13)ii)
   iv. Card 1 is not the joker. (1, P1.3(13)ii)
   v. Card 2 is not the joker. (1, P1.3(13)i)
   vi. None of cards 1, 2, or 3 are the joker. (P1.3(13)iv, P1.3(13)v, P1.3(13)iii)
   vii. Contradiction. (P1.3(13)vi, 4)

(14) Card 2 is not red. (13, proof by contradiction)

(15) Neither card 3 nor card 2 is red. (12, 14)

(16) Card 1 is red. (15, 2)

(17) Card 3 is the joker. (5, 16)

(18) Card 1 is not black. (16, 1)

(19) Card 3 is not black. (17, 1)

(20) Neither card 1 nor card 3 is black. (18, 19)

(21) Card 2 is black. (3, 20)

(22) Card 1 is red, card 2 is black, and card 3 is the joker. (16, 21, 17)

School of Computing and Information Systems
COMP30026 Models of Computation
Week 2: Propositional Logic

If you find yourself getting stuck on a particular question in the tutorial, try to move onto other questions until you have a chance to ask your tutor for help.

Exercises

T2.1 For each pair of formulas, write down and compare their truth tables. In which rows (if any) are they different?

(i) $\neg P \to Q$ and $P \to \neg Q$

(ii) $\neg(P \wedge \neg Q)$ and $P \to Q$

(iii) $(P \wedge Q) \to R$ and $P \to (Q \to R)$

(iv) $P \to (Q \to R)$ and $(P \to Q) \to R$

Solution:

(i) Not logically equivalent:

P	Q	$\neg P$	$\to$	Q	P	$\to$	$\neg Q$
0	0	1	0	0	0	1	1
0	1	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	1	0	1	1	1	0	0
			✗			✗	

(ii) Logically equivalent:

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

(iii) Logically equivalent:

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

(iv) Not logically equivalent:

P	Q	R	P	$\rightarrow$	$(\ Q$	$\rightarrow$	$R\ )$	$(\ P$	$\rightarrow$	$Q\ )$	$\rightarrow$	R
0	0	0	0	1	0	1	0	0	1	0	0	0
0	0	1	0	1	0	1	1	0	1	0	1	1
0	1	0	0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	1	1	1	0	1	1	1	1
1	0	0	1	1	0	1	0	1	0	0	1	0
1	0	1	1	1	0	1	1	1	0	0	1	1
1	1	0	1	0	1	0	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1
				✗							✗	

T2.2 Find a formula that is equivalent to $(P \wedge \neg Q) \vee P$ but shorter.

Solution: $(P \wedge \neg Q) \vee P$ is logically equivalent to P. You can see this by reasoning by cases on the truth value of P.

T2.3 Write down the definitions of *satisfiable*, *valid*, *unsatisfiable* and *non-valid*. Explain why any given propositional formula must have exactly two of these properties, and describe the following propositional formulas using these terms.

(i) $P \vee Q$

(ii) $P \wedge \neg P$

(iii) $((P \rightarrow Q) \rightarrow P) \rightarrow P$

(iv) $(P \wedge \neg P) \rightarrow P$

Solution:

- A *satisfiable* formula has *at least one* truth assignment that makes it true (it *can* be satisfied by a truth assignment).
- A *valid* formula is one which is made true by *all* truth assignments (it *must* be satisfied by any truth assignment).
- An *unsatisfiable* formula is one which is made false by *all* truth assignments (it *cannot* be satisfied by any truth assignment).
- A *non-valid* formula has *at least one* truth assignment that makes it false (it *can* be made false by a truth assignment)

Consider the truth table of a propositional formula φ.

- If there are some rows of the truth table which make φ true **and** some rows that make φ false, then φ is both *satisfiable* and *non-valid*. φ is not *valid*, since there are rows much make φ false, and φ is not *unsatisfiable*, since there are rows that make φ true.
- If all rows of the truth table make φ true, then φ is *valid* and also *satisfiable*. φ is not *non-valid*, nor *unsatisfiable*, since there are no rows which make φ false.
- If all rows of the truth table make φ false, then φ is *unsatisfiable* and also *non-valid*. φ is not *satisfiable*, nor *valid*, since there are no rows which make φ true.

These are the only possibilites for the rows of any truth table. Note that *valid* is a stronger property than *satisfiable*, and *unsatisfiable* is a stronger property than *non-valid*.

(i) $P \vee Q$ is satisfiable and non-valid.

(ii) $P \wedge \neg P$ is unsatisfiable and non-valid.

(iii) $((P \to Q) \to P) \to P$ is valid and satisfiable.

(iv) $(P \wedge \neg P) \to P$ is valid and satisfiable.

T2.4 Let's try a more challenging puzzle from Smullyan's Island of Knights and Knaves. Recall that knights always tell the truth and knaves always lie. We meet three people from the island, and we are reliably informed that one of the three is also a magician. They make these statements:

A: B is not both a knave and a magician.

B: Either A is a knave or I am not a magician.

C: The magician is a knave.

Who is the magician? Give a proof, and check that your solution indeed solves the problem.

Hint: If you have a disjunction like $P \vee Q$ or a negated conjunction like $\neg(P \wedge Q)$, negating it (e.g. for the sake of contradiction) produces a conjunction, which is usually much easier to work with.

Solution: Suppose to the contrary that A is a knave. Then B is both a knave and a magician. Thus, from B's utterance, A is not a knave and B is a magician. Contradiction! Therefore, A is a knight.

Suppose to the contrary that B is a knave. Then A is not a knave and B is a magician. Hence B is both a knave and a magician. But since A is a knight, we know from A's utterance that this is not the case. Contradiction! Hence B is a knight. Thus, by B's utterance, since A is not a knave, B is not a magician.

Suppose to the contrary that C is a knight. Then the magician is a knave. But A, B and C are all knights, so this is a contradiction. Hence C is a knave, and so the magician is not a knave. Hence the magician must be a knight.

Since B is not a magician and C is not a knight, by elimination it follows that A is the magician.

Homework problems

P2.1 Which of the following pairs of formulas are equivalent?

(a) $\neg P \to Q$ and $P \to \neg Q$

(b) $\neg P \to Q$ and $Q \to \neg P$

(c) $\neg P \to Q$ and $\neg Q \to P$

(d) $(P \to Q) \to P$ and P

(e) $P \to (Q \to R)$ and $Q \to (P \to R)$

(f) $P \to (Q \to R)$ and $(P \to Q) \to R$

(g) $(P \wedge Q) \to R$ and $P \to (Q \to R)$

(h) $(P \vee Q) \to R$ and $(P \to R) \wedge (Q \to R)$

Solution:

(a) Not equivalent:

P	Q	$\neg P$	$\to$	Q	P	$\to$	$\neg Q$
0	0	1	0	0	0	1	1
0	1	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	1	0	1	1	1	0	0
			✗			✗	

We see that the columns for the two implications are different.

(b) Not equivalent:

P	Q	$\neg P$	$\rightarrow$	Q	Q	$\rightarrow$	$\neg P$
0	0	1	0	0	0	1	1
0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	0
1	1	0	1	1	1	0	0
			✗			✗	

(c) Equivalent:

P	Q	$\neg P$	$\rightarrow$	Q	$\neg Q$	$\rightarrow$	P
0	0	1	0	0	1	0	0
0	1	1	1	1	0	1	0
1	0	0	1	0	1	1	1
1	1	0	1	1	0	1	1
			✓			✓	

(d) Equivalent:

P	Q	$(P$	$\rightarrow$	$Q)$	$\rightarrow$	P
0	0	0	1	0	0	0
0	1	0	1	1	0	0
1	0	1	0	0	1	1
1	1	1	1	1	1	1
✓					✓	

(e), (f) : the pair in (e) equivalent; but the pair in (f) are not equivalent:

P	Q	R	P	$\rightarrow$	$(Q$	$\rightarrow$	$R)$	Q	$\rightarrow$	$(P$	$\rightarrow$	$R)$	$(P$	$\rightarrow$	$Q)$	$\rightarrow$	R
0	0	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0
0	0	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1
0	1	0	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0
0	1	1	0	1	1	1	1	1	1	0	1	1	0	1	1	1	1
1	0	0	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0
1	0	1	1	1	0	1	1	0	1	1	1	1	1	0	0	1	1
1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
				✓					✓								
				✗												✗	

Note that the formulas in (e) are both equivalent to $(P \wedge Q) \rightarrow R$, which explains why the order of P and Q does not matter here.

(g) Equivalent:

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

(h) Equivalent:

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

P2.2 Define your own binary connective $\square$ by writing out a truth table for $P \square Q$ (fill in the middle column however you like). Can you write a formula which has the the same truth table as $P \square Q$ using only the symbols P, Q, $\neg$, $\wedge$, $\vee$, and $\rightarrow$? Repeat the exercise once.

Solution: Here's an example

P	Q	P	$\square$	Q
0	0	0	1	0
0	1	0	1	1
1	0	1	1	0
1	1	1	0	1

With this truth table for $\square$, $P \square Q$ is logically equivalent to $\neg(P \wedge Q)$.

P2.3 How many distinct truth tables are there involving two fixed propositional letters? In other words, how many meaningfully distinct connectives could we have defined in the previous question?

Solution: There are four rows in the truth table, so that's 2 choices for each row, which amounts to $2 \times 2 \times 2 \times 2 = 16$ possibilities.

P2.4 Find a formula that is equivalent to $P \leftrightarrow (P \wedge Q)$ but shorter.

Solution: $(P \wedge Q) \leftrightarrow P$ is logically equivalent to $P \rightarrow Q$. This is easily checked with a truth table, but how can we simplify $(P \wedge Q) \leftrightarrow P$ when we don't know what it is supposed to be equivalent to? Well, we can just try. Let us expand the biimplication and obtain $(P \rightarrow (P \wedge Q)) \wedge ((P \wedge Q) \rightarrow P)$. Intuitively, the conjunct on the right is just true, and we can check that with a truth table. So we have found that the original formula is equivalent to $P \rightarrow (P \wedge Q)$, which isn't any shorter, but still. We can rewrite the result as $(P \rightarrow P) \wedge (P \rightarrow Q)$, and now it becomes clear that all we need is $P \rightarrow Q$.

P2.5 Find a formula that is equivalent to $(\neg P \vee Q) \wedge R$ using only $\rightarrow$ and $\neg$ as logical connectives.

Solution: $(\neg P \vee Q) \wedge R$ is logically equivalent to $\neg((P \rightarrow Q) \rightarrow \neg R)$

P2.6 Consider the formula $P \rightarrow \neg P$. Is that a contradiction (is it *unsatisfiable*)? Can a proposition imply its own negation?

Solution: There is no contradiction at all. The formula is true if (and only if) P is false. The point of a conditional formula is to make a claim about the scenario where the premise (P) is true. If the premise of $\rightarrow$ is false, the formula is satisfied. For the same reason, $\neg P \rightarrow P$ is satisfiable; it is not a contradiction. But $P \leftrightarrow \neg P$ is clearly a contradiction. (If you disagree with any of these statements, draw truth tables.)

P2.7 By negating a satisfiable proposition, can you get a tautology? A satisfiable proposition? A contradiction? Illustrate your affirmative answers.

Solution: If you negate a satisfiable proposition, you can never get a tautology, since at least one truth table row will yield false.

You will get another satisfiable proposition iff the original proposition is not valid. For example, P is satisfiable (but not valid), and indeed $\neg P$ is satisfiable.

Finally, if we have a satisfiable formula which is also valid, its negation will be a contradiction. Example: $P \vee \neg P$.

P2.8 For each of the following propositional formulas, determine whether it is satisfiable, and if it is, whether it is a tautology:

(a) $P \leftrightarrow ((P \to Q) \to P)$

(b) $(P \to \neg Q) \wedge ((P \vee Q) \to P)$

Solution: Let us draw the truth tables.

(a)

P	Q	P	$\leftrightarrow$	$((P$	$\to$	$Q)$	$\to$	$P)$
0	0	0	1	0	1	0	0	0
0	1	0	1	0	1	1	0	0
1	0	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1
			⇑					

Hence satisfiable, and in fact valid (all 1).

(b)

P	Q	$(P$	$\to$	$\neg$	$Q)$	$\wedge$	$((P$	$\vee$	$Q)$	$\to$	$P)$
0	0	0	1	1	0	1	0	0	0	1	0
0	1	0	1	0	1	0	0	1	1	0	0
1	0	1	1	1	0	1	1	1	0	1	1
1	1	1	0	0	1	0	1	1	1	1	1
						⇑					

Hence satisfiable (at least one 1), but not valid (not all 1). The truth table shows the formula is equivalent to $\neg Q$.

P2.9 Complete the following sentences, using the words "satisfiable, valid, non-valid, unsatisfiable".

(a) F is satisfiable iff F is not _____
(b) F is valid iff F is not _____
(c) F is non-valid iff F is not _____
(d) F is unsatisfiable iff F is not _____
(e) F is satisfiable iff $\neg F$ is _____
(f) F is valid iff $\neg F$ is _____
(g) F is non-valid iff $\neg F$ is _____
(h) F is unsatisfiable iff $\neg F$ is _____

Solution:

(a) F is satisfiable iff F is not unsatisfiable
(b) F is valid iff F is not non-valid
(c) F is non-valid iff F is not valid
(d) F is unsatisfiable iff F is not satisfiable
(e) F is satisfiable iff $\neg F$ is non-valid
(f) F is valid iff $\neg F$ is unsatisfiable
(g) F is non-valid iff $\neg F$ is satisfiable
(h) F is unsatisfiable iff $\neg F$ is valid

P2.10 Show that $P \leftrightarrow (Q \leftrightarrow R) \equiv (P \leftrightarrow Q) \leftrightarrow R$. This tells us that we could instead write

$$P \leftrightarrow Q \leftrightarrow R \tag{1}$$

without introducing any ambiguity. Mind you, that may not be such a good idea, because many people (incorrectly) tend to read "$P \leftrightarrow Q \leftrightarrow R$" as

$$P, Q, \text{ and } R \text{ all have the same truth value} \tag{2}$$

Show that (1) and (2) are incomparable, that is, neither is a logical consequence of the other.

Solution: Even with three variables the truth table is manageable, so let us construct it.

			(1A)					(1B)					(2)		
P	Q	R	P	$\leftrightarrow$	$(Q$	$\leftrightarrow$	$R)$	$(P$	$\leftrightarrow$	$Q)$	$\leftrightarrow$	R	$P \wedge Q \wedge R$	$\vee$	$\neg P \wedge \neg Q \wedge \neg R$
0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	1	0	0	1	0	1	0	1	1	0	0	0
0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0
0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	0
1	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0
1	0	1	1	0	0	0	1	1	0	0	0	1	0	0	0
1	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
				✓							✓			✗	

B is a logical consequence of A (we write $A \models B$) iff B is true for any assignment of variables which makes A true. So we see that (1) $\not\models$ (2), because cases like 1 0 0 that make (1) true but (2) false. Similarly, (2) $\not\models$ (1), because the case 0 0 0 makes (2) true but (1) false.

P2.11 Let F and G be propositional formulas. What is the difference between "$F \equiv G$" and "$F \leftrightarrow G$"? Prove that "$F \leftrightarrow G$" is valid iff $F \equiv G$.

Solution: The connective $\leftrightarrow$ is part of the language that we study, namely the language of propositional logic. So $A \leftrightarrow B$ is just a propositional formula.

The symbol $\equiv$ belongs to a *meta-language*. The meta-language is a language which we use when we reason *about* some language. In this case we use $\equiv$ to express whether a certain relation holds between formulas in propositional logic.

More specifically, $F \equiv G$ means that we have both $F \models G$ and $G \models F$. In other words, F and G have the same value for every possible assignment of truth values to their variables. The two formulas are logically equivalent.

On the other hand $F \leftrightarrow G$ is just a propositional formula (assuming F and G are propositional formulas). For some values of the variables involved, $F \leftrightarrow G$ may be false, for other values it may be true. By the definition of validity, $F \leftrightarrow G$ is *valid* iff it is true for *every* assignment of propositional variables in F and G.

We want to show that $F \equiv G$ iff $F \leftrightarrow G$ is valid.

(a) Suppose $F \equiv G$. Then F and G have the same values for each truth assignment to their variables[1]. But that means that, when we construct the truth table for $F \leftrightarrow G$, it will have a t in every row, that is, $F \leftrightarrow G$ is valid.

[1] We should perhaps be more careful here, because F and G can be logically equivalent without F having the exact same set of variables as G—can you see how? So we should say that we consider both of F and G to be functions of the *union* of their sets of variables.

(b) Suppose $F \leftrightarrow G$ is valid. That means we find a t in each row of the truth table for $F \leftrightarrow G$. But we get a t for $F \leftrightarrow G$ iff the values for F and G agree, that is, either both are f, or both are t. In other words, F and G agree for every truth assignment. Hence $F \equiv G$.

You may think that this relation between validity and biimplication is obvious and should always be expected, and indeed we will see that it carries over to first-order predicate logic. But there are (still useful) logics in which it does not hold.

P2.12 Is $(P \wedge Q) \leftrightarrow P$ logically equivalent to $(P \vee Q) \leftrightarrow Q$?

Solution: Yes, $(P \wedge Q) \leftrightarrow P \equiv (P \vee Q) \leftrightarrow Q$, and both of these formulas are logically equivalent to $P \rightarrow Q$

P	Q	(P	$\wedge$	Q)	$\leftrightarrow$	P	(P	$\vee$	Q)	$\leftrightarrow$	Q
0	0	0	0	0	1	0	0	0	0	1	0
0	1	0	0	1	1	0	0	1	1	1	1
1	0	1	0	0	0	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1
					✓					✓	

School of Computing and Information Systems
COMP30026 Models of Computation
Week 3: Writing and Checking Proofs

For the homework problems, swap your answers with a friend, and critique each other's work!

Exercises

T3.1 For each of the following, determine whether it is a valid deductive argument. Justify your response.

(a) My neighbours have woken me up every night so far, and therefore they will also wake me up tonight.

Solution: Does not follow deductively; the premises do not guarantee the conclusion. What if the neighbours go on holiday?

(b) Suppose that all birds fly, and that Jo flies. This suggests that Jo is a bird.

Solution: Not a deductive argument. Conclusion is not a declarative statement about the world.

(c) There is no greatest prime number.

Solution: This is a statement, not an argument.

(d) Suppose that cities, villages, and towns exist. Suppose also that some pigeons live in cities, and that some pigeons do not. Therefore, some pigeons live in towns.

Solution: Does not follow deductively. Pigeons could live in villages but not towns.

(e) Suppose that $0 = 1$. Then the set of all sets exists.

Solution: Follows deductively, but is vacuous.

T3.2 Prove by induction the following statement:

Claim: For all positive integers n,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

T3.3 The following "proof" contains a very subtle but significant error. Can you spot it? (Reminder: for all $p, q, n \in \mathbb{Z}$, we have $p \equiv q \pmod{n}$ if and only if $p - q$ is a multiple of n.)

Claim: Let $a, b \in \mathbb{Z}$ where $a \equiv 1 \pmod{3}$ and $b \equiv 2 \pmod{3}$. Then $a + b \equiv 0 \pmod{3}$.

"Proof:" Since $a \equiv 1 \pmod{3}$ there is an integer k such that $a = 3k+1$. Since $b \equiv 2 \pmod{3}$, we can write $b = 3k + 2$. Thus

$$\begin{aligned} a + b &= (3k + 1) + (3k + 2) \\ &= 6k + 3 \\ &= 3(2k + 1), \end{aligned}$$

and so $a + b \equiv 0 \pmod{3}$.

Solution: The variable k is introduced in the first sentence of the proof, and is then illegally reused in the second sentence.

T3.4 Prove by contradiction the following claim:

Claim: Let $a, b \in \mathbb{R}$. If a is rational and ab is irrational, then b is irrational.

Solution: Suppose a is rational and that ab is irrational. Suppose to the contrary that b is rational. Then we have $a = p/q$ and $b = r/s$ for some integers p, q, r, s. Therefore $ab = \frac{p}{q} \cdot \frac{r}{s} = \frac{pr}{qs}$, which is rational by definition. Contradiction! Hence b is not rational.

Homework problems

P3.1 Prove or disprove the following claim:

Claim: For all sets A, B, C, if $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.

Solution: Prove from the definition of subset. (Given sets X and Y, we have $X \subseteq Y$ if and only if every member of X is a member of Y.)

P3.2 Prove or disprove the following claim:

Claim: For all sets A, B, C, if $A \in B$ and $B \in C$, then $A \in C$.

Solution: Give a counterexample, and argue that the premises are satisfied but that the conclusion is not.

P3.3 Prove that $\sqrt{3}$ is irrational.

Solution: Use the same strategy we used when proving the $\sqrt{2}$ is irrational.

P3.4 Write a corrected version of the proof from T3.3.

P3.5 Prove by induction the following claim:

Claim: For all positive integers n,

$$\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6}.$$

Solution: Use the same strategy we used to show that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

P3.6 **Definition:** An integer is even iff it is a multiple of 2.

Definition: An integer is odd iff it is not even.

Prove that the product of any two odd integers is odd.

P3.7 Prove the following claim:

Claim: Let a, b and c be odd integers. Then the polynomial $ax^2 + bx + c$ has no rational roots.

Solution: If you want to use the quadratic formula, remember to handle the case when $a = 0$.

A simpler strategy is to proceed by contradiction, and consider an arbitary rational root. Substitute it into the polynomial, then reason by cases. Which terms are even or odd? Show that this leads to a contradiction in all cases.

School of Computing and Information Systems
COMP30026 Models of Computation
Week 4: Translation and Resolution

Exercises

T4.1 We say that a formula is in RCNF (reduced conjunctive normal form) if it is in CNF and none of the clauses are tautologies, have duplicate literals, or subsume any other clause.

Put the following formulas into RCNF:

(a) $\neg(A \wedge \neg(B \wedge C))$

(b) $A \vee (\neg B \wedge (C \vee (\neg D \wedge \neg A)))$

(c) $(A \vee B) \rightarrow (C \wedge D)$

(d) $A \wedge (B \rightarrow (A \rightarrow B))$

Continued in P4.2 & P4.4

Solution:

(a)

$\neg(A \wedge \neg(B \wedge C))$	
$\neg A \vee (B \wedge C)$	(push negation in)
$(\neg A \vee B) \wedge (\neg A \vee C)$	(distribute $\vee$ over $\wedge$)

The result is now in reduced CNF.

(b)

$A \vee (\neg B \wedge (C \vee (\neg D \wedge \neg A)))$	
$(A \vee \neg B) \wedge (A \vee C \vee (\neg D \wedge \neg A))$	(distribute $\vee$ over $\wedge$)
$(A \vee \neg B) \wedge (A \vee C \vee \neg D) \wedge (A \vee C \vee \neg A)$	(distribute $\vee$ over $\wedge$)

The result is in CNF but not RCNF. To get RCNF we need to eliminate the last clause which is a tautology, and we end up with $(A \vee \neg B) \wedge (A \vee C \vee \neg D)$.

(c)

$(A \vee B) \rightarrow (C \wedge D)$	
$\neg(A \vee B) \vee (C \wedge D)$	(unfold $\rightarrow$)
$(\neg A \wedge \neg B) \vee (C \wedge D)$	(de Morgan)
$(\neg A \vee (C \wedge D)) \wedge (\neg B \vee (C \wedge D))$	(distribute $\vee$ over $\wedge$)
$(\neg A \vee C) \wedge (\neg A \vee D) \wedge (\neg B \vee C) \wedge (\neg B \vee D)$	(distribute $\wedge$ over $\vee$)

The result is in RCNF. We could have chosen different orders for the distributions.

(d)

$A \wedge (B \rightarrow (A \rightarrow B))$	
$A \wedge (\neg B \vee \neg A \vee B)$	(unfold both occurrences of $\rightarrow$)
A	(rightmost clause is tautological: remove it)

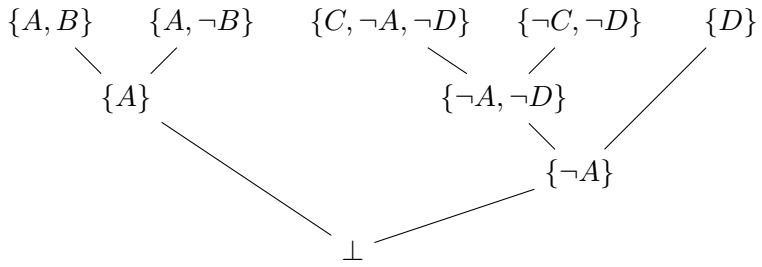
T4.2 Resolution works by producing resolvents of a set of disjunctive clauses. These resolvents are logical consequences of the original set, so if a resolvent is unsatisfiable, so is the original set of clauses.

Using resolution, show that the following set of clauses is unsatisfiable:

$$\{\{A, B\}, \{A, \neg B\}, \{C, \neg A, \neg D\}, \{\neg C, \neg D\}, \{D\}\}.$$

Continued in P4.1

Solution: Here is a refutation:



From this we conclude that $(A \vee B) \wedge (A \vee \neg B) \wedge (C \vee \neg A \vee \neg D) \wedge (\neg C \vee \neg D) \wedge D$ is unsatisfiable.

T4.3 Consider the following premises:

(a) If Jemima eats duck food, then she is healthy.

(b) If Jemima is healthy, she will lay an egg.

(c) Jemima eats duck food.

Translate them into formulas of propositional logic. Then, draw an appropriate resolution **refutation**, and use it to prove that Jemima will lay an egg.

Continued in P4.5 & P4.6

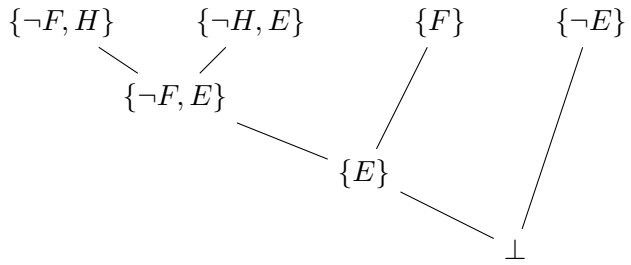
Solution: We can translate these as follows, using F for "Jemima eats duck food", H for "Jemima is healthy", and E for "Jemima will lay an egg".

(i) $F \to H$

(ii) $H \to E$

(iii) F

We want to show that in every scenario where these assumptions are true, E must also be true. In other words, we want to show $(F \to H) \wedge (H \to E) \wedge F \vDash E$. This is equivalent to showing that the formula $(F \to H) \wedge (H \to E) \wedge F \wedge \neg E$ is unsatisfiable. This formula is equivalent to $(\neg F \vee H) \wedge (\neg H \vee E) \wedge F \wedge \neg E$, which is in conjunctive normal form. This corresponds to the set of clauses $\{\{\neg F, H\}, \{\neg H, E\}, \{F\}, \{\neg E\}\}$, which we can show is unsatisfiable with the resolution proof below.



T4.4 We now have 3 different tools to reason about propositional formulas: truth tables/assignments, equivalences, and resolution. Discuss how we can use these tools to do the following. Which is best in each situation?

(i) Prove $F \equiv G$.

(ii) Disprove $F \equiv G$.

(iii) Prove F is unsatisfiable.

(iv) Prove F is satisfiable.

(v) Prove F is valid.

(vi) Prove $F \vDash G$.

(vii) Prove that F must hold under the premises A, B, and C. (That is, prove that $A \wedge B \wedge C \vDash F$.)

(viii) Disprove $A \wedge B \wedge C \vDash F$.

Solution: If we only need to know about one row of the truth table to demonstrate something, then exhibiting that truth assignment is the cleanest and easiest way to do so. Otherwise we can choose between writing out the truth table or using resolution. If the formulas are small enough, then it's often easiest to just draw up the truth tables, but resolution is a lot easier when the formulas become larger.

The important piece of information for each of this situations that use resolution is *how you reduce the problem to showing some set of clauses is unsatisfiable.*

(i) A chain of equivalences can demonstrate $F \equiv G$ if the formulas are similar enough. If they are too different, it might take a while to turn one formula into the other. If the formulas are small enough, we can just draw the truth tables and compare their main columns. Otherwise, we can show that $\neg(F \leftrightarrow G)$ is unsatisfiable via resolution, since $\neg(F \leftrightarrow G)$ is unsatisfiable iff $F \equiv G$.

(ii) We can disprove $F \equiv G$ by exhibiting a truth assignment which makes one true and the other false. For example, $A \rightarrow B \not\equiv B \rightarrow A$, since the truth assignment that makes A true and B false makes $A \rightarrow B$ false and $B \rightarrow A$ true.

(iii) We can show F is unsatisfiable by drawing its truth table if it's small enough. Otherwise, we can just convert F to CNF and apply resolution until we reach the empty clause.

(iv) We can show F is satisfiable by exhibiting a truth assignment that makes it true. For example, $A \vee B$ is satisfiable, because it is true when we assign A true and B false. You might be tempted to apply resolution, then fail to produce the empty clause and therefore conclude it must be satisfiable because you couldn't show it to be unsatisfiable. While this can be done correctly, you would need to prove that whatever algorithm you're using to produce resolvents will *always* find the empty clause in a finite number of steps, whenever the formula is unsatisfiable, and will fail and terminate otherwise. That's *so much more effort* than just exhibiting a truth assignment that makes the formula true. However, using resolution on a satisfiable set of clauses can still give you logical consequences, which can help you construct the truth assignment which satisfies all of the original clauses.

(v) We can show F is valid by showing $\neg F$ is unsatisfiable (or just look at the truth table if it is small enough to write out).

(vi) We can show $F \vDash G$ by showing $F \wedge \neg G$ is unsatisfiable.

(vii) We can show $A \wedge B \wedge C \vDash F$ by showing $A \wedge B \wedge C \wedge \neg F$ is unsatisfiable.

(viii) We can disprove $A \wedge B \wedge C \vDash F$ by exhibiting a truth assignment which makes $A \wedge B \wedge C$ true and F false.

Homework problems

P4.1 Using resolution, show that the following set of conjunctive clauses is unsatisfiable:

$$\{\{P, R, \neg S\}, \{P, S\}, \{\neg Q\}, \{Q, \neg R, \neg S\}, \{\neg P, Q\}\}.$$

P4.2 Find the reduced CNF of $\neg((\neg B \to \neg A) \to ((\neg B \to A) \to B))$ and express the result in clausal form. Then determine whether a refutation of the resulting set is possible.

Solution: Let us follow the method given in a lecture, except we do the double-negation elimination aggressively, as soon as opportunity arises:

$\neg((\neg B \to \neg A) \to ((\neg B \to A) \to B))$	
$\neg(\neg(B \vee \neg A) \vee \neg(B \vee A) \vee B)$	(unfold $\to$ and eliminate double negation)
$(B \vee \neg A) \wedge (B \vee A) \wedge \neg B$	(de Morgan for outermost neg; elim double neg)

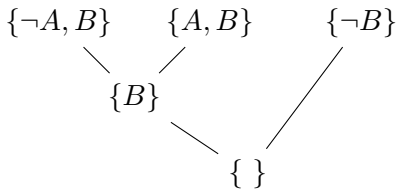
This is RCNF without further reductions.

We could also have used other transformations—sometimes this can shorten the process. For example, we could have rewritten the sub-expression $\neg B \to \neg A$ as $A \to B$ (the contraposition principle). You may want to check that this does not change the result.

The resulting formula, written as a set of sets of literals:

$$\{\{\neg A, B\}, \{A, B\}, \{\neg B\}\}$$

We can now construct the refutation:



P4.3 Use resolution to show that each of these formulas is a tautology:

(a) $(P \vee Q) \to (Q \vee P)$
(b) $(\neg P \to P) \to P$
(c) $((P \to Q) \to P) \to P$
(d) $P \leftrightarrow ((P \to Q) \to P)$

Solution:

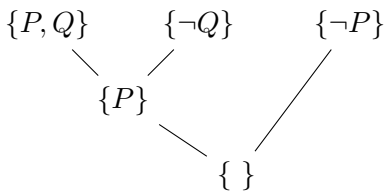
(a) $(P \vee Q) \to (Q \vee P)$. First negate the formula (why?), to get $\neg((P \vee Q) \to (Q \vee P))$. Then we can use the usual techniques to convert the negated proposition to RCNF. Here is a useful shortcut, combining $\to$-elimination with one of de Morgan's laws:

$$\neg(A \to B) \equiv A \wedge \neg B.$$

So:

$\neg((P \vee Q) \to (Q \vee P))$	
$(P \vee Q) \wedge \neg(Q \vee P)$	(shortcut)
$(P \vee Q) \wedge \neg Q \wedge \neg P$	(de Morgan)

The result allows for a straight-forward refutation:



(b) $(\neg P \to P) \to P$. Again, first negate the formula, to get $\neg((\neg P \to P) \to P)$. Then turn the result into RCNF:

$\neg((\neg P \to P) \to P)$	
$(\neg P \to P) \wedge \neg P$	(shortcut from above)
$(\neg\neg P \vee P) \wedge \neg P$	(unfold $\to$)
$(P \vee P) \wedge \neg P$	(eliminate double negation)
$P \wedge \neg P$	($\vee$-absorption)

The resolution proof is immediate; we will leave it out.

(c) $((P \to Q) \to P) \to P$. Again, negate the formula, to get $\neg(((P \to Q) \to P) \to P)$. Then turn the result into RCNF:

$\neg(((P \to Q) \to P) \to P)$	
$((P \to Q) \to P) \wedge \neg P$	(shortcut, outermost $\to$)
$((\neg P \vee Q) \to P) \wedge \neg P$	(unfold $\to$)
$(\neg(\neg P \vee Q) \vee P) \wedge \neg P$	(unfold $\to$)
$((\neg\neg P \wedge \neg Q) \vee P) \wedge \neg P$	(de Morgan)
$((P \wedge \neg Q) \vee P) \wedge \neg P$	(double negation)
$(P \vee P) \wedge (\neg Q \vee P) \wedge \neg P$	(distribution)
$P \wedge (\neg Q \vee P) \wedge \neg P$	(absorption)

Again this gives an immediate refutation: just resolve $\{P\}$ against $\{\neg P\}$.

(d) $P \leftrightarrow ((P \to Q) \to P)$. Negating the formula, we get $P \oplus ((P \to Q) \to P)$. Let us turn the resulting formula into RCNF:

$P \oplus ((P \to Q) \to P)$	
$(P \vee ((P \to Q) \to P)) \wedge (\neg P \vee \neg((P \to Q) \to P))$	(eliminate $\oplus$)
$(P \vee ((P \to Q) \to P)) \wedge (\neg P \vee ((P \to Q) \wedge \neg P))$	(shortcut from above)
$(P \vee (\neg(\neg P \vee Q) \vee P)) \wedge (\neg P \vee ((\neg P \vee Q) \wedge \neg P))$	($\to$-elimination)
$(P \vee (\neg\neg P \wedge \neg Q) \vee P) \wedge (\neg P \vee ((\neg P \vee Q) \wedge \neg P))$	(de Morgan)
$(P \vee (P \wedge \neg Q) \vee P) \wedge (\neg P \vee ((\neg P \vee Q) \wedge \neg P))$	(double negation)
$P \wedge (P \vee \neg Q) \wedge (\neg P \vee ((\neg P \vee Q) \wedge \neg P))$	($\vee$-absorption, distribution)
$P \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge \neg P$	($\vee$-absorption, distribution)

Once again, now just resolve $\{P\}$ against $\{\neg P\}$.

P4.4 For each of the following clause sets, write down a propositional formula in CNF to which it corresponds. Which of the resulting formulas are satisfiable? Give models of those that are.

(a) $\{\{A, B\}, \{\neg A, \neg B\}, \{\neg A, B\}\}$

(b) $\{\{A, \neg B\}, \{\neg A\}, \{B\}\}$

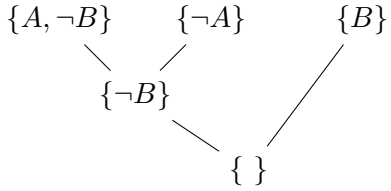
(c) $\{\{A\}, \emptyset\}$

(d) $\{\{A, B\}, \{\neg A, \neg B\}, \{B, C\}, \{\neg B, \neg C\}, \{A, C\}, \{\neg A, \neg C\}\}$

Solution:

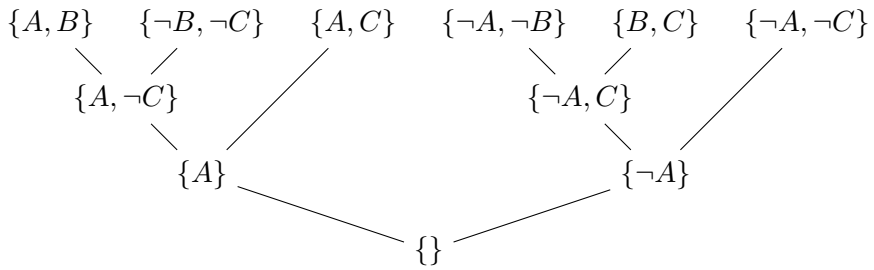
(a) $\{\{A, B\}, \{\neg A, \neg B\}, \{\neg A, B\}\}$ stands for the formula $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee B)$. This is satisfiable by $\{A \mapsto \mathbf{f}, B \mapsto \mathbf{t}\}$.

(b) $\{\{A, \neg B\}, \{\neg A\}, \{B\}\}$ stands for $(A \vee \neg B) \wedge \neg A \wedge B$. A refutation is easy:



(c) $\{\{A\}, \emptyset\}$ stands for $A \wedge \mathbf{f}$, which is clearly not satisfiable.

(d) We have $\{\{A, B\}, \{\neg A, \neg B\}, \{B, C\}, \{\neg B, \neg C\}, \{A, C\}, \{\neg A, \neg C\}\}$. This set is not satisfiable, as a proof by resolution shows:



P4.5 Consider these assumptions:

(a) If Ann can clear 2 meters, she will be selected.

(b) If Ann trains hard then, if she gets the flu, the selectors will be sympathetic.

(c) If Ann trains hard and does not get the flu, she can clear 2 meters.

(d) If the selectors are sympathetic, Ann will be selected.

Does it follow that Ann will be selected? Does she get selected if she trains hard? Use any of the propositional logic techniques we have discussed, to answer these questions.

Solution: Let us give names to the propositions:

- C: Ann clears 2 meters
- F: Ann gets the flu
- K: The selectors are sympathetic
- S: Ann is selected
- T: Ann trains hard

The four assumptions then become:

(a) $C \to S$
(b) $T \to (F \to K)$
(c) $(T \wedge \neg F) \to C$
(d) $K \to S$

It is easy to see that S is not a logical consequence of these, as we can give all five variables the value *false*, and all the assumptions will thereby be true.

To see that $T \to S$ is a logical consequence of the assumptions, we can negate it, obtaining $T \wedge \neg S$. Then, translating everything to clausal form, we can use resolution to derive an empty clause.

Alternatively, note that $T \to (F \to K)$ is equivalent to $(T \wedge F) \to K$. Since also $K \to S$, we have $(T \wedge F) \to S$. Similarly, $(T \wedge \neg F) \to C$ together with $C \to S$ gives us $(T \wedge \neg F) \to S$.

But from $(T \wedge F) \to S$ and $(T \wedge \neg F) \to S$ we get $T \to S$. (You may want to check that by massaging the conjunction of the two formulas.)

P4.6 Consider the following four statements:

(a) The commissioner cannot attend the function unless he resigns and apologises.

(b) The commissioner can attend the function if he resigns and apologises.

(c) The commissioner can attend the function if he resigns.

(d) The commissioner can attend the function only if he apologises.

Identify the basic propositions involved and translate the statements into propositional logic. In particular, what is the translation of a statement of the form "X does not happen unless Y happens"? Identify cases where one of the statements entails some other statement in the list.

Solution: Let us give names to the propositions:

- A: The commissioner apologises
- F: The commissioner can attend the function
- R: The commissioner resigns

The four statements then become

(a) $F \to (A \wedge R)$
(b) $(R \wedge A) \to F$
(c) $R \to F$
(d) $F \to A$

The first translation may not be obvious. But to say "X does not happen unless Y happens" is the same as saying "it is not possible to have X happen and at the same time Y does not happen." That is, $\neg(X \wedge \neg Y)$, which is equivalent to $X \to Y$. Note that (a) entails (d) and (c) entails (b).

P4.7 Letting F and G be two different formulas from the set

$$\{(P \wedge Q) \vee R, \quad (P \vee Q) \wedge R, \quad P \wedge (Q \vee R), \quad P \vee (Q \wedge R)\}$$

list all combinations that satisfy $F \models G$.

P4.8 A formula is in *disjunctive* normal form (DNF) iff it is a disjunction of conjunctions of literals. We say a formula is in RDNF (reduced disjunctive normal form) iff it is in DNF and no conjunctive clause has duplicate literals, is a contradiction, or subsumes another conjunctive clause. (If F and G are *conjunctive* clauses, we say F subsumes G iff $F \vee G \equiv F$.)

We claim that the formula

$$(P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q \wedge \neg R) \vee (\neg P \wedge R) \vee (Q \wedge \neg R)$$

is logically equivalent to the simpler

$$\neg P \vee Q$$

with both being in reduced disjunctive normal form (RDNF). Show that the claim is correct.

P4.9 In P4.8 we saw that truth of this formula does not depend on the truth of R:

$$(P \wedge Q \wedge R) \vee (\neg P \wedge \neg Q \wedge \neg R) \vee (\neg P \wedge R) \vee (Q \wedge \neg R)$$

Find a shortest logically equivalent CNF formula which includes R.

School of Computing and Information Systems
COMP30026 Models of Computation
Week 5: Translation — Predicate Logic

Exercises

T5.1 Express the following statements in predicate logic, using the one-place predicates D and M, where $D(x)$ means "x is a duck", and $M(x)$ means "x is muddy". Use the constant a as a name for Jemima.

(i) Jemima is a duck.
(ii) If Jemima is a duck, then she is muddy.
(iii) Jemima is a muddy duck.
(iv) Every duck is muddy.
(v) There is a muddy duck.
(vi) Everything is a muddy duck.

Solution:

(i) $D(a)$
(ii) $D(a) \to M(a)$
(iii) $D(a) \wedge M(a)$
(iv) $\forall x(D(x) \to M(x))$
(v) $\exists x(M(x) \wedge D(x))$
(vi) $\forall x(M(x) \wedge D(x))$

T5.2 Let's add a new two-place predicate $L(x, y)$, which means "x lays y", and a one-place predicate $E(z)$ which means "z is an egg". Translate the following statements into predicate logic.

(i) Jemima lays an egg.
(ii) Jemima is muddy and lays an egg.
(iii) Everything lays an egg.
(iv) Every duck lays an egg.
(v) Every muddy duck lays an egg.
(vi) There is a duck who lays every egg.

Continued in P5.1 & P5.2

Solution:

(i) $\exists y(L(a, y) \wedge E(y))$
(ii) $M(a) \wedge \exists y(L(a, y) \wedge E(y))$
(iii) $\forall x(\exists y(L(x, y) \wedge E(y)))$
(iv) $\forall x(D(x) \to \exists y(L(x, y) \wedge E(y)))$
(v) $\forall x((M(x) \wedge D(x)) \to \exists y(L(x, y) \wedge E(y)))$
(vi) $\exists x(D(x) \wedge \forall y(E(y) \to L(x, y)))$, or $\forall y(E(y) \to \exists x(D(x) \wedge L(x, y)))$ (natural languages are ambiguous)

T5.3 Translate the following predicate logic formulas to natural language, by interpreting $L(x, y)$ as "x loves y". It may help to translate the *inner* formula first. For example, inside the first formula, $\exists y\, L(x, y)$ says "x loves somebody".

(i) $\forall x \exists y\, L(x, y)$
(ii) $\exists y \forall x\, L(x, y)$
(iii) $\forall x \exists y\, L(y, x)$
(iv) $\exists y \forall x\, L(y, x)$
(v) $\forall x \forall y\, L(x, y)$
(vi) $\forall y \forall x\, L(x, y)$

Solution:

(i) Everybody loves somebody.

(ii) Somebody is loved by everybody.

(iii) Everybody is loved by someone.

(iv) Somebody loves everybody.

(v) Everybody loves everybody.

(vi) Everybody is loved by everybody.

T5.4 What is the effect of quantifier order on a formula? Use your understanding of the formulas in T5.3 to explain which ones are semantic consequences of another. Are any logically equivalent? You do not need to compare every single pair of formulas.

Solution: We will list all of the logical consequences for each formula, and omit all of the pairs where there is no logical consequence relationship.

(i) $\forall x(\exists y\ L(x,y))$

- (i) $\cdots \vDash \forall x(\exists y\ L(x,y))$

(ii) $\exists y(\forall x\ L(x,y))$

- (i) $\cdots \vDash \forall x(\exists y\ L(x,y))$
- (ii) $\cdots \vDash \exists y(\forall x\ L(x,y))$

(iii) $\forall x(\exists y\ L(y,x))$

- (iii) $\cdots \vDash \forall x(\exists y\ L(y,x))$

(iv) $\exists y(\forall x\ L(y,x))$

- (iii) $\cdots \vDash \forall x(\exists y\ L(y,x))$
- (iv) $\cdots \vDash \exists y(\forall x\ L(y,x))$

(v) $\forall x(\forall y\ L(x,y))$

- (i) $\cdots \vDash \forall x(\exists y\ L(x,y))$
- (ii) $\cdots \vDash \exists y(\forall x\ L(x,y))$
- (iii) $\cdots \vDash \forall x(\exists y\ L(y,x))$
- (iv) $\cdots \vDash \exists y(\forall x\ L(y,x))$
- (v) $\cdots \vDash \forall x(\forall y\ L(x,y))$
- (vi) $\cdots \vDash \forall y(\forall x\ L(x,y))$

(vi) $\forall y(\forall x\ L(x,y))$

- (i) $\cdots \vDash \forall x(\exists y\ L(x,y))$
- (ii) $\cdots \vDash \exists y(\forall x\ L(x,y))$
- (iii) $\cdots \vDash \forall x(\exists y\ L(y,x))$
- (iv) $\cdots \vDash \exists y(\forall x\ L(y,x))$
- (v) $\cdots \vDash \forall x(\forall y\ L(x,y))$
- (vi) $\cdots \vDash \forall y(\forall x\ L(x,y))$

Homework problems

P5.1 Translate the following formulas into predicate logic, using $D(x)$ for "x is a duck", $M(x)$ for "x is muddy", $F(x,y)$ for "x follows y" and $R(x,y)$ for "x is muddier than y". Use the constant a as a name for Jemima, and b for Louise.

(i) Every duck that follows a muddy duck is muddy.

(ii) Any muddy duck is muddier than any duck that isn't muddy.

(iii) Jemima is muddier than any duck that is muddier than Louise.

(iv) There is a duck that is muddier than Louise, but not as muddy as Jemima.

(v) Given any two ducks, one is muddier than the other.

(vi) Given any three ducks, if the first is muddier than the second, and the second is muddier than the third, then the first is muddier than the third.

Solution:

(i) $\forall x((D(x) \wedge \exists y(D(y) \wedge M(y) \wedge F(x,y))) \rightarrow M(x))$

(ii) $\forall x((D(x) \wedge M(x)) \rightarrow \forall y((D(y) \wedge \neg M(y)) \rightarrow R(x,y)))$

(iii) $\forall x((D(x) \wedge R(x,b)) \rightarrow R(a,x))$

(iv) $\exists x(D(x) \wedge R(x,b) \wedge \neg R(x,a))$

(v) $\forall x \forall y((D(x) \wedge D(y)) \rightarrow (R(x,y) \vee R(y,x)))$

(vi) $\forall x \forall y \forall z((D(x) \wedge D(y) \wedge D(z)) \rightarrow ((R(x,y) \wedge R(y,z)) \rightarrow R(x,z)))$

P5.2 Translate the following formulas into predicate logic, using $D(x)$ for "x is a duck", $M(x)$ for "x is muddy", $E(x)$ for "x is an egg", $L(x,y)$ for "x lays y", and $R(x,y)$ for "x is muddier y". Use the constant a as a name for Jemima, and b for Louise.

(i) Not every duck lays an egg.

(ii) Every duck doesn't lay an egg.

(iii) There is a duck that doesn't lay an egg.

(iv) There isn't a duck who lays every egg.

(v) Louise is not muddier than every duck.

(vi) Louise is not muddier than any duck.

(vii) Every egg has a duck who laid it.

(viii) Not every egg is laid by a muddy duck.

(ix) If every duck is muddy, then no duck lays an egg.

Solution:

(i) $\neg \forall x(D(x) \rightarrow \exists y(E(y) \wedge L(x,y)))$

(ii) $\forall x(D(x) \rightarrow \neg \exists y(E(y) \wedge L(x,y)))$

(iii) $\exists x(D(x) \wedge \neg \exists y(E(y) \wedge L(x,y)))$

(iv) $\neg \exists x(D(x) \wedge \forall y(E(y) \rightarrow L(x,y)))$

(v) $\neg \forall x(D(x) \rightarrow R(b,x))$

(vi) $\forall x(D(x) \rightarrow \neg R(b,x))$

(vii) $\forall y(E(y) \rightarrow \exists x(D(x) \wedge L(x,y)))$

(viii) $\neg \forall y(E(y) \rightarrow \exists x(M(x) \wedge D(x) \wedge L(x,y)))$

(ix) $(\forall x(D(x) \rightarrow M(x))) \rightarrow (\forall x(D(x) \rightarrow \neg \exists y(L(x,y) \wedge E(y))))$

P5.3 For any formula G and variable x, $\neg \forall x\, G \equiv \exists x\, \neg G$, and $\neg \exists x\, G \equiv \forall x\, \neg G$. Interpret the formula $\neg \forall x(D(x) \rightarrow \exists y(E(y) \wedge L(x,y)))$ in natural language, then use these equivalences to "push the negation" through each of the quantifiers and connectives, and re-interpret the result in natural language. Reflect on why these are saying the same thing.

Solution: For any formula G and variable x, $\neg \forall x G \equiv \exists x \neg G$, and $\neg \exists x G \equiv \forall x \neg G$. Interpret the formula $\neg \forall x(D(x) \rightarrow \exists y(E(y) \wedge L(x,y)))$ in natural language, then use these equivalences to "push the negation" through each of the quantifiers and connectives, and re-interpret the result in natural language. Reflect on why these are saying the same thing. $\neg \forall x(D(x) \rightarrow \exists y(E(y) \wedge L(x,y)))$ says "It's not the case that every duck lays an egg". If we push negation all the way in, the resulting equivalent formula is $\exists x(D(x) \wedge \forall y(E(y) \rightarrow \neg L(x,y)))$. This says that there is a duck which doesn't lay any egg at all, i.e. taking any particular egg, we claim the duck doesn't lay that egg.

P5.4 In the following formulas, identify which variables are bound to which quantifiers, and which variables are free.

(i) $\forall y(D(x) \wedge \exists x(E(y) \leftrightarrow L(x,y)))$

(ii) $\exists z(E(z) \wedge M(y)) \rightarrow \forall y(E(z) \wedge M(y))$

(iii) $\exists x((E(x) \wedge M(y)) \rightarrow \forall y(E(x) \wedge M(y)))$

(iv) $\forall z((\exists z\, D(z)) \rightarrow D(z))$

(v) $\exists u((D(z) \wedge \forall x(M(x) \rightarrow D(x))) \rightarrow \forall z(L(x,z)))$

(vi) $\forall x(\forall y(M(x) \rightarrow D(x)) \wedge \exists y(D(y) \wedge \forall y(L(y,x))))$

Solution: In the following formulas, identify which variables are bound to which quantifiers, and which variables are free.

(i) In $\forall y(D(x) \wedge \exists x(E(y) \leftrightarrow L(x, y)))$ the first occurrence of x is free and the second is bound to the existential quantifier. Both y's are bound to the universal quantifier

(ii) In $\exists z(E(z) \wedge M(y)) \rightarrow \forall y(E(z) \wedge M(y))$, the first occurrence of z is bound to the existential quantifier, and the second is free. The first occurrence of y is free and the second is bound to the universal quantifier.

(iii) In $\exists x((E(x) \wedge M(y)) \rightarrow \forall y(E(x) \wedge M(y)))$, both occurences of x are bound to the existential quantifier. The first occurrence of y is free and the second is bound to the universal quantifier.

(iv) In $\forall z(\exists z(D(z)) \rightarrow D(z))$, the first occurrence of z is bound to the existential quantifier, and the second is bound to the universal quantifier.

(v) In $\exists u((D(z) \wedge \forall x(M(x) \rightarrow D(x))) \rightarrow \forall z(L(x, z)))$, the first occurrence of z is free, and the second is bound to the $\forall z$. The first two occurrences of x are bound to the $\forall x$, and the second is free.

(vi) In $\forall x(\forall y(M(x) \rightarrow D(x)) \wedge \exists y(D(y) \wedge \forall y(L(y, x))))$, all occurrences of x are bound to the $\forall x$. The first occurrence of y is bound to the $\exists y$, and the second is bound to the last $\forall y$.

P5.5 On the Island of Knights and Knaves, everyone is a knight or knave. Knights always tell the truth. Knaves always lie. You meet three people, let us call them A, B and C. A says to you: "If I am a knight then my two friends here are knaves." Use **propositional** logic to determine whether this statement gives you any real information. What, if anything, can be deduced about A, B, and C?

Solution: Let the propositional variable A stand for "A is a knight" and similarly for B and C. Note that if A makes statement S then we know that $A \leftrightarrow S$ holds. Or we can consider the two possible cases for A separately: Either A is a knight, and A's statement can be taken face value; or A is a knave, in which case the negation of the statement holds, that is,

$$\Big(A \wedge \big(A \rightarrow (\neg B \wedge \neg C)\big)\Big) \vee \Big(\neg A \wedge \neg\big(A \rightarrow (\neg B \wedge \neg C)\big)\Big)$$

We can rewrite the implications:

$$\Big(A \wedge \big(\neg A \vee (\neg B \wedge \neg C)\big)\Big) \vee \Big(\neg A \wedge \neg\big(\neg A \vee (\neg B \wedge \neg C)\big)\Big)$$

Then, pushing negation in and using the distributive laws, we get

$$\Big(A \wedge (\neg B \wedge \neg C)\Big) \vee \Big(\neg A \wedge A \wedge (B \vee C)\Big)$$

The second disjunct is false, so the formula is equivalent to $A \wedge \neg B \wedge \neg C$. So A must be a knight, and B and C are knaves.

P5.6 A graph colouring is an assignment of colours to nodes so that no edge in the graph connects two nodes of the same colour. The graph colouring problem asks whether a graph can be coloured using some fixed number of colours. The question is of great interest, because many scheduling problems are graph colouring problems in disguise. The case of three colours is known to be hard (NP-complete).

How can we encode the three-colouring problem in propositional logic—more precisely, in CNF? (One reason we might want to do so is that we can then make use of a SAT solver to determine colourability.) Using propositional variables

- B_i to mean node i is blue,
- G_i to mean node i is green,
- R_i to mean node i is red;
- E_{ij} to mean i and j are different but connected by an edge,

write formulas in CNF for these statements:

(a) Every node (0 to n inclusive) has a colour.
(b) Every node has at most one colour.
(c) No two connected nodes have the same colour.

For a graph with $n+1$ nodes, what is the total size of these CNF formulas? (The size of a CNF formula is the number of literals it contains.)

Solution: These are the clauses generated:

(a) For each node i generate the clause $B_i \vee G_i \vee R_i$. That's $n+1$ clauses of size 3 each.

(b) For each node i generate three clauses: $(\neg B_i \vee \neg G_i) \wedge (\neg B_i \vee \neg R_i) \wedge (\neg G_i \vee \neg R_i)$. That comes to $3n+3$ clauses of size 2 each.

(c) For each pair (i,j) of nodes with $i < j$ we want to express $E_{ij} \to (\neg(B_i \wedge B_j) \wedge \neg(G_i \wedge G_j) \wedge \neg(R_i \wedge R_j)$. This means for each pair (i,j) we generate three clauses: $(\neg E_{ij} \vee \neg B_i \vee \neg B_j) \wedge (\neg E_{ij} \vee \neg G_i \vee \neg G_j) \wedge (\neg E_{ij} \vee \neg R_i \vee \neg R_j)$. There are $n(n+1)/2$ pairs, so we generate $3n(n+1)/2$ clauses, each of size 3.

Altogether we generate $3n + 3 + 6n + 6 + 9n(n+1)/2$ literals, that is, $9(n+1)(n/2+1)$.

School of Computing and Information Systems
COMP30026 Models of Computation
Week 5: Predicate Logic — Semantics and Resolution

Exercises

T5.1 For each of the following predicate logic formulas, give a model in which the formula is true, and a model in which the formula is false.

(a) $\forall x \forall y\, P(x, y)$

(b) $\forall x \exists y (P(x, y) \wedge \neg P(y, x))$

(c) $(\forall x \exists y\, \neg P(x, y)) \wedge (\forall x \exists y\, P(y, x))$

Solution: In each case below we use two models with the same universe. That is just a coincidence—we could have chosen differently.

(a) Let the universe be very simple, say $\{0\}$. Interpreting P as "equal to" makes this true. Interpreting P as "not equal to" makes this false.

(b) Let the universe be the set of integers. Interpreting P as "less than" makes this true. Interpreting P as "equal to" makes it false.

(c) The same model will work here.

T5.2 Use resolution to show that the following set of clauses is unsatisfiable:

$$\{Q(f(u), g(x)), P(x)\}, \{\neg P(b)\}, \{\neg Q(z, y), R(z, z)\}, \{\neg R(w, f(u))\}$$

T5.3 Turn $\exists z \forall x\ \exists y\ \left[\exists u\ \forall v \Big(P(x, y) \vee Q(z, u, v) \Big) \rightarrow \forall u\ \Big(\neg R(f(x), u, y) \Big)\right]$ into clausal form.

Solution: First we must remove the $\rightarrow$ connective, and then push negations all the way in.

$$\begin{aligned}
& \exists z \forall x\ \exists y\ \left[\exists u\ \forall v \Big(P(x, y) \vee Q(z, u, v) \Big) \rightarrow \forall u\ \Big(\neg R(f(x), u, y) \Big)\right] \\
\equiv\ & \exists z \forall x\ \exists y\ \left[\neg \exists u\ \forall v \Big(P(x, y) \vee Q(z, u, v) \Big) \vee \forall u\ \Big(\neg R(f(x), u, y) \Big)\right] \\
\equiv\ & \exists z \forall x\ \exists y\ \left[\forall u\ \exists v \neg \Big(P(x, y) \vee Q(z, u, v) \Big) \vee \forall u\ \Big(\neg R(f(x), u, y) \Big)\right] \\
\equiv\ & \exists z \forall x\ \exists y\ \left[\forall u\ \exists v \Big(\neg P(x, y) \wedge \neg Q(z, u, v) \Big) \vee \forall u\ \Big(\neg R(f(x), u, y) \Big)\right]
\end{aligned}$$

Now we can remove the existential quantifiers via *Skolemization*. The $\exists v$ occurs inside the scope of $\forall x$ and $\forall u$ (the left one), so we will pick a fresh function symbol g and replace v with $g(x, u)$, and then remove the existential quantifier. Note that we cannot use f, because f is not *fresh*: it has been used elsewhere in the formula.

$$\begin{aligned}
& \exists z \forall x\ \exists y\ \left[\forall u\ \exists v \Big(\neg P(x, y) \wedge \neg Q(z, u, v) \Big) \vee \forall u\ \Big(\neg R(f(x), u, y) \Big)\right] \\
\approx\ & \exists z \forall x\ \exists y\ \left[\forall u\ \Big(\neg P(x, y) \wedge \neg Q(z, u, g(x, u)) \Big) \vee \forall u\ \Big(\neg R(f(x), u, y) \Big)\right]
\end{aligned}$$

similarly, we will Skolemize the usages of y with $h(x)$ and z with a.

$$\begin{aligned} & \exists z \forall x\ \exists y\ \left[\forall u\ \left(\neg P(x,y) \wedge \neg Q(z,u,g(x,u))\right) \vee \forall u\ \left(\neg R(f(x),u,y)\right)\right] \\ \approx\ & \forall x\ \left[\forall u\ \left(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\right) \vee \forall u\ \left(\neg R(f(x),u,h(x))\right)\right] \end{aligned}$$

Now we need to deal with the universal quantifiers. It would be incorrect to just drop all of the quantifiers, because there are two seperate quantifications over u. While we might still preserve satisfiability by doing this, we are *losing information* by forcing both instance of u to coordinate. We should first rename the variable u in one of the disjuncts before we proceed.

$$\begin{aligned} & \forall x\ \left[\forall u\ \left(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\right) \vee \forall u\ \left(\neg R(f(x),u,h(x))\right)\right] \\ \equiv\ & \forall x\ \left[\forall u\ \left(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\right) \vee \forall w\ \left(\neg R(f(x),w,h(x))\right)\right] \end{aligned}$$

Now that all universal quantifiers use distinct variables, we can now just drop all of them from the formula, while preserving satisfiability.

$$\begin{aligned} & \forall x\ \left[\forall u\ \left(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\right) \vee \forall w\ \left(\neg R(f(x),w,h(x))\right)\right] \\ \approx\ & \big(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\big) \vee \big(\neg R(f(x),w,h(x))\big) \end{aligned}$$

Finally we need to convert to CNF, and write in clausal form.

$$\begin{aligned} & \big(\neg P(x,h(x)) \wedge \neg Q(a,u,g(x,u))\big) \vee \big(\neg R(f(x),w,h(x))\big) \\ \approx\ & \{\{\neg P(x,h(x)), \neg R(f(x),w,h(x))\}, \{\neg Q(a,u,g(x,u)), \neg R(f(x),w,h(x))\}\} \end{aligned}$$

T5.4 Use the following theorem to prove that $\forall x\, P(x) \models \exists x\, P(x)$ holds. (Recall that the universe of discourse is nonempty by definition.) Does $\exists x\, P(x) \models \forall x\, P(x)$ also hold?

Theorem. Given any model M, variable assignment v, variable x and formula F, if $M, v \models \forall x\, F$, then $M, v \models F$.

Solution:

Proof. Let M be a model of $\forall x\, P(x)$. Then, by definition, given any variable assignment v, we have $M, v \models \forall x\, P(x)$, and thus $M, v \models P(x)$ by the above theorem. Hence $M, v \models \exists x\, P(x)$ by definition. Since this holds for arbitrary v, we have $M \models \exists x\, P(x)$, as desired. □

The converse does not hold. Let M be the model whose universe is $\mathbb{Z}$ (the set of integers), and let P stand for "is zero". Then M satisfies $\exists y\, P(y)$ but not $\forall x\, P(x)$.

Homework problems

P5.1 Consider the following predicates:

- $C(x)$, which stands for "x is a cat"
- $M(x)$, which stands for "x is a mouse"
- $L(x, y)$, which stands for "x likes y"

Express the statement "No mouse likes a cat who likes all mice" as a formula in first-order predicate logic. Once you have, convert that formula into clausal form.

Solution: Here is how we might capture "z is a mouse": $M(z)$, and here is how we can say that "x is a cat who likes mice": $C(x) \wedge \forall y(M(y) \to L(x, y))$. Now we want to say that if both of those two statements are true then z does not like x, and that's that case no matter which z and which x we are talking about:

$$\forall x \forall z \Big(M(z) \wedge C(x) \wedge \forall y(M(y) \to L(x, y)) \to \neg L(z, x) \Big)$$

This might not follow with a natural reading of the statement in English. However, $\forall x, P(x)$ will hold iff $\neg \exists x \, \neg P(x)$). With a natural reading of the sentence in English, for some definition of "natural", one may read the following:

$$\neg \exists z \Big(M(z) \wedge \exists x \big(C(x) \wedge \forall y(M(y) \to L(x, y)) \wedge L(z, x) \big) \Big)$$

P5.2 For this question use the following predicates:

- $G(x)$ for "x is a green dragon"
- $R(x)$ for "x is a red dragon"
- $H(x)$ for "x is a happy dragon"
- $S(x)$ for "x can spit fire"
- $P(x, y)$ for "x is a parent of y"
- $C(x, y)$ for "x is a child of y"

(a) Express the following statements as formulas in first-order predicate logic:

i. x is a parent of y if and only if y is a child of x.

ii. A dragon is either green or red; not both.

iii. A dragon is green if and only if at least one of its parents is green.

iv. Green dragons can spit fire.

v. A dragon is happy if all of its children can spit fire.

(b) Translate each of the five formulas to clausal form.

(c) Prove, using resolution, that all green dragons are happy.

P5.3 Consider the following formulas:

(a) $\forall x \, \neg L(x, x)$

(b) $\forall x \exists y \, L(x, y)$

(c) $\forall x \forall z (L(x, z) \to \exists y (L(x, y) \wedge L(y, z)))$

Give a model which satisfies all three formulas. Can this be done with a finite universe? If so, how many elements does the smallest such universe have?

Solution: These formulas are true in the model where the universe is $\mathbb{Q}$ and L stands for the usual "less than" predicate.

We can also do this with a finite universe. We can find a finite model M by thinking about what the directed graph of the relation $I(L)$ has to look like, where I is the interpretation function of M. In terms of directed graphs, the formulas say the following things, in order:

(a) There are no self-loops.

(b) Every node has an edge going out of it.

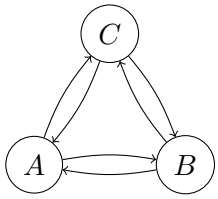
(c) If there is an edge from a node x to a node z, then there is also a node y such that there is an edge from x to y and from y to z.

Can a graph with just one node satisfy these conditions? No, because there has to be an edge going out of the node, but if there is only one node, then that edge must a self-loop.

How about a two-node graph? Well, we can satisfy the first two conditions with this graph:

This is also the only two-node graph satisfying the first two conditions: we cannot delete any edge without violating the second condition, and we cannot add any edge without violating the first condition. But the third condition is not satisfied: there is an edge from A to B, but there is no node y such that there is an edge from A to y and from y to B.

So a two-node graph won't work, but we seem to be close. All we need to do is add one extra node to satisfy the third condition:



This one works! No self-loops, every node has an edge coming out of it, and you can always find a "detour" to satisfy the third condition. So a three-node graph is the smallest solution you can get. This immediately gives us a model where the universe is the set of vertices $V = \{A, B, C\}$, and $I(L)$ is the set of edges $E = \{(A, B), (B, A), (A, C), (C, A), (B, C), (C, B)\}$. This model is also the only 3-element model, up to renaming. Larger (but still finite) models exist too. Can you figure out how to construct a model of any given size greater than 3?

P5.4 Using equivalences, show that $\neg\forall x\exists y\ (\neg P(x) \wedge P(y))$ is valid. Then convert the formula to clausal form. **Solution:** It is easy to see that $\neg\forall x\exists y\ (\neg P(x) \wedge P(y))$ is valid. For example, first push negation in, to get $\exists x\forall y\ (P(x) \vee \neg P(y))$. Both quantifiers can be pushed in, since there is no y in the left conjunct. So this formula is equivalent: $\exists x\ (P(x)) \vee \forall y\ (\neg P(y))$. This is clearly valid. It is also straight-forward to turn into clausal form; we get just one clause: $P(a) \vee \neg P(y)$.

P5.5 Prove the theorem given in T5.4.

P5.6 Turn $\neg\forall x\ \exists y\ \left[\forall z\ \Big(Q(x, z) \wedge P(y)\Big) \wedge \forall u\ \Big(\neg Q(u, x)\Big)\right]$ into clausal form.

Solution: Let's start by pushing negation in. The formula becomes

$$\exists x\ \forall y\Big(\exists z\ (\neg Q(x, z) \vee \neg P(y)) \vee \exists u\ Q(u, x)\Big)$$

Now we can Skolemize and remove universal quantifiers. The result is a single clause:

$$\neg Q(a, f(y)) \vee \neg P(y) \vee Q(g(y), a)$$

P5.7 Turn $\forall x\forall y\exists z\Big(P(x) \to \forall y\forall z(Q(y, z))\Big)$ into a simpler, equivalent formula of the form $\varphi \to \psi$.

Solution: We can use the rules of passage for the quantifiers. First, note that the sub-formula

$$\Big(P(x) \to \forall y\forall z(Q(y, z))\Big)$$

has no free occurrence of y or z. Hence, we can simply drop the quantifiers $\forall y \exists z$ that we find in front of that sub-formula, which leaves us with

$$\forall x \Big(P(x) \to \forall y \forall z (Q(y, z)) \Big).$$

Since the right-hand side of the arrow has no free occurrence of x, we can push the remaining universal quantifier in, which yields $\exists x(P(x)) \to \forall y \forall z(Q(y, z))$. This is of the required form. If you wonder about the universal quantifier turning into an existential quantifier, complete this exercise by filling in the details (start by rewriting the implication to a disjunction).

P5.8 For each of the following pairs of terms, determine whether the pair is unifiable. If it is, give the most general unifier. (Don't forget our agreed convention: for constants we use letters from the beginning of the alphabet, here a and b, whereas for variables we use letters from the end of the alphabet.)

(i) $h(f(x), g(y, f(x)), y)$ and $h(f(u), g(v, v), u)$
(ii) $h(f(g(x, y)), y, g(y, y))$ and $h(f(u), g(a, v), u)$
(iii) $h(g(x, x), g(y, z), g(y, f(z)))$ and $h(g(u, v), g(v, u), v)$
(iv) $h(v, g(v), f(u, a))$ and $h(g(x), y, x)$
(v) $h(f(x, x), y, y, x)$ and $h(v, v, f(a, b), a)$

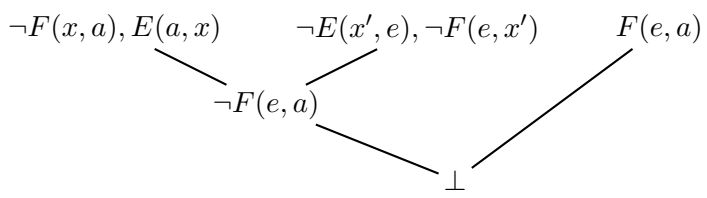
P5.9 Consider the following predicates:

- $E(x, y)$, which stands for "x envies y"
- $F(x, y)$, which stands for "x is more fortunate than y"

(a) Using 'a' for Adam, express, in first-order predicate logic, the sentence "Adam envies everyone more fortunate than him."
(b) Using 'e' for Eve, express, in first-order predicate logic, the sentence "Eve is no more fortunate than any who envy her."
(c) Formalise an argument for the conclusion that "Eve is no more fortunate than Adam." That is, express this statement in first-order predicate logic and show that it is a logical consequence of the other two.

Solution:

(a) $\forall x \ (F(x, a) \to E(a, x))$
(b) $\forall x \ (E(x, e) \to \neg F(e, x))$
(c) We capture "Eve is no more fortunate than Adam" as $\neg F(e, a)$. To show that this is a logical consequence of the other two statements, we need to show that every model of $\forall x \ (F(x, a) \to E(a, x)) \wedge \forall x \ (E(x, e) \to \neg F(e, x))$ makes $F(e, a)$ false. Assume (for contradiction) that there is a model in which $F(e, a)$ is true. Then, by the left conjunct, $E(a, e)$ is also true in this model. But then, by the right conjunct, $\neg F(e, a)$ is also true, that is, $F(e, a)$ is false. But this is a contradiction, so $F(e, a)$ must be false. Indeed a proof by resolution is easy:



P5.10 Using the unification algorithm, determine whether $Q(f(g(x), y, f(y, z, z)), g(f(a, y, z)))$ and $Q(f(u, g(a), v), u)$ are unifiable. If they are, give a most general unifier. (As usual, we use letters from the end of the alphabet for variables, and letters from the beginning of the alphabet for constants.)

P5.11 Determine whether $P(f(g(x), f(g(x), g(a))), x)$ and $P(f(u, f(v, v)), u)$ are unifiable. If they are, give a most general unifier.

Solution:

(i) The pair of terms $(h(f(x), g(y, f(x)), y), h(f(u), g(v, v), u))$ is not unifiable. Applying rule 1 (decomposition) to $\{h(f(x), g(y, f(x)), y) = h(f(u), g(v, v), u)\}$, we get

$$\left\{ \begin{array}{rcl} f(x) & = & f(u) \\ g(y, f(x)) & = & g(v, v) \\ y & = & u \end{array} \right\}$$

Applying rule 1 (decomposition) again, to each of the first two equations, yields

$$\left\{ \begin{array}{rcl} x & = & u \\ y & = & v \\ f(x) & = & v \\ y & = & u \end{array} \right\}$$

Applying rule 6 (substitution) with the first equation, we get

$$\left\{ \begin{array}{rcl} x & = & u \\ y & = & v \\ f(u) & = & v \\ y & = & u \end{array} \right\}$$

Applying rule 4 (reorientation) to the third equation, followed by rule 6 to the result yields

$$\left\{ \begin{array}{rcl} x & = & u \\ y & = & f(u) \\ v & = & f(u) \\ y & = & u \end{array} \right\}$$

Applying rule 6 (substitution) with the last equation yields

$$\left\{ \begin{array}{rcl} x & = & u \\ u & = & f(u) \\ v & = & f(u) \\ y & = & u \end{array} \right\}$$

Now the occur check applied to the second equation yields failure.

(ii) The pair of terms $(h(f(g(x, y)), y, g(y, y)), h(f(u), g(a, v), u))$ is unifiable. Applying rule 1 (decomposition) to $\{h(f(g(x, y)), y, g(y, y)) = h(f(u), g(a, v), u)\}$, we get

$$\left\{ \begin{array}{rcl} f(g(x, y)) & = & f(u) \\ y & = & g(a, v) \\ g(y, y) & = & u \end{array} \right\}$$

and a second application yields

$$\left\{ \begin{array}{rcl} g(x, y) & = & u \\ y & = & g(a, v) \\ g(y, y) & = & u \end{array} \right\}$$

Applying rule 4 (reorientation) to the first and the third equation, we have

$$\left\{\begin{array}{rcl} u & = & g(x,y) \\ y & = & g(a,v) \\ u & = & g(y,y) \end{array}\right\}$$

Applying rule 6 (to the first equation) we then get

$$\left\{\begin{array}{rcl} u & = & g(x,y) \\ y & = & g(a,v) \\ g(x,y) & = & g(y,y) \end{array}\right\}$$

which, after an application of rule 1 gives

$$\left\{\begin{array}{rcl} u & = & g(x,y) \\ y & = & g(a,v) \\ x & = & y \\ y & = & y \end{array}\right\}$$

The last equation is dropped, by rule 3, and then rule 6 applied to the third equation gives

$$\left\{\begin{array}{rcl} u & = & g(y,y) \\ y & = & g(a,v) \\ x & = & y \end{array}\right\}$$

Finally, rule 6 applied to the second equation gives

$$\left\{\begin{array}{rcl} u & = & g(g(a,v),g(a,v)) \\ y & = & g(a,v) \\ x & = & g(a,v) \end{array}\right\}$$

This is a normal form and $\{u \mapsto g(g(a,v),g(a,v)), y \mapsto g(a,v), x \mapsto g(a,v)\}$ is the most general unifier.

(iii) The pair of terms $(h(g(x,x),g(y,z),g(y,f(z))),h(g(u,v),g(v,u),v))$ is not unifiable. Applying rule 1 (decomposition) to $\{h(g(x,x),g(y,z),g(y,f(z))) = h(g(u,v),g(v,u),v)\}$, we get

$$\left\{\begin{array}{rcl} g(x,x) & = & g(u,v) \\ g(y,z) & = & g(v,u) \\ g(y,f(z)) & = & v \end{array}\right\}$$

Applying rule 1 (decomposition) again, to each of the first two equations, yields

$$\left\{\begin{array}{rcl} x & = & u \\ x & = & v \\ y & = & v \\ z & = & u \\ g(y,f(z)) & = & v \end{array}\right\}$$

Applying rule 4 (reorientation) to the last equation, followed by rule 6 applied to v yields

$$\left\{\begin{array}{rcl} x & = & u \\ x & = & g(y,f(z)) \\ y & = & g(y,f(z)) \\ z & = & u \\ v & = & g(y,f(z)) \end{array}\right\}$$

Now the occur check (rule 5) applied to the third equation yields failure.

(iv) The pair of terms $(h(v, g(v), f(u, a)), h(g(x), y, x))$ is unifiable. Applying rule 1 (decomposition) to $\{h(v, g(v), f(u, a)) = h(g(x), y, x)\}$, we get

$$\left\{ \begin{array}{rcl} v & = & g(x) \\ g(v) & = & y \\ f(u,a)) & = & x \end{array} \right\}$$

Reorienting the last two equations:

$$\left\{ \begin{array}{lcl} v & = & g(x) \\ y & = & g(v) \\ x & = & f(u,a)) \end{array} \right\}$$

Now replacing x (rule 6):

$$\left\{ \begin{array}{lcl} v & = & g(f(u,a)) \\ y & = & g(v) \\ x & = & f(u,a)) \end{array} \right\}$$

Finally replacing v (rule 6):

$$\left\{ \begin{array}{lcl} v & = & g(f(u,a)) \\ y & = & g(g(f(u,a))) \\ x & = & f(u,a)) \end{array} \right\}$$

we have a normal form and $\{v \mapsto g(f(u, a)), x \mapsto f(u, a), y \mapsto g(g(f(u, a)))\}$ is the most general unifier.

(v) The pair of terms $(h(f(x, x), y, y, x), h(v, v, f(a, b), a))$ is not unifiable. Applying rule 1 (decomposition) to $\{h(f(x, x), y, y, x) = h(v, v, f(a, b), a)\}$, we get

$$\left\{ \begin{array}{rcl} f(x,x) & = & v \\ y & = & v \\ y & = & f(a,b) \\ x & = & a \end{array} \right\}$$

Reorienting the first equation yields

$$\left\{ \begin{array}{lcl} v & = & f(x,x) \\ y & = & v \\ y & = & f(a,b) \\ x & = & a \end{array} \right\}$$

Now applying rule 6 to x and then to v, we get

$$\left\{ \begin{array}{lcl} v & = & f(a,a) \\ y & = & f(a,a) \\ y & = & f(a,b) \\ x & = & a \end{array} \right\}$$

Now apply rule 6 to, say, the second equation and get

$$\left\{ \begin{array}{rcl} v & = & f(a,a) \\ y & = & f(a,a) \\ f(a,a) & = & f(a,b) \\ x & = & a \end{array} \right\}$$

Decomposition (rule 1) then yields

$$\left\{ \begin{array}{lcl} v & = & f(a,a) \\ y & = & f(a,a) \\ a & = & a \\ a & = & b \\ x & = & a \end{array} \right\}$$

Now the second-last equation gives match failure (rule 2 applies), and so the original pair of terms were not unifiable.

Solution:

(a) i. $\forall x \forall y(P(x,y) \leftrightarrow C(y,x))$
ii. $\forall x(G(x) \oplus R(x))$
iii. $\forall x(G(x) \leftrightarrow \exists y(P(y,x) \wedge G(y)))$
iv. $\forall x(G(x) \rightarrow S(x))$
v. $\forall x(\forall y[C(y,x) \rightarrow S(y)] \rightarrow H(x))$

(b) Before we generate clauses, let us simplify the third formula. Replacing $\leftrightarrow$, we get

$$\forall x(\neg G(x) \vee \exists y(P(y,x) \wedge G(y)) \wedge (G(x) \vee \neg \exists y(P(y,x) \wedge G(y))))$$

Pushing negation in:

$$\forall x(\neg G(x) \vee \exists y(P(y,x) \wedge G(y)) \wedge (G(x) \vee \forall y(\neg P(y,x) \vee \neg G(y))))$$

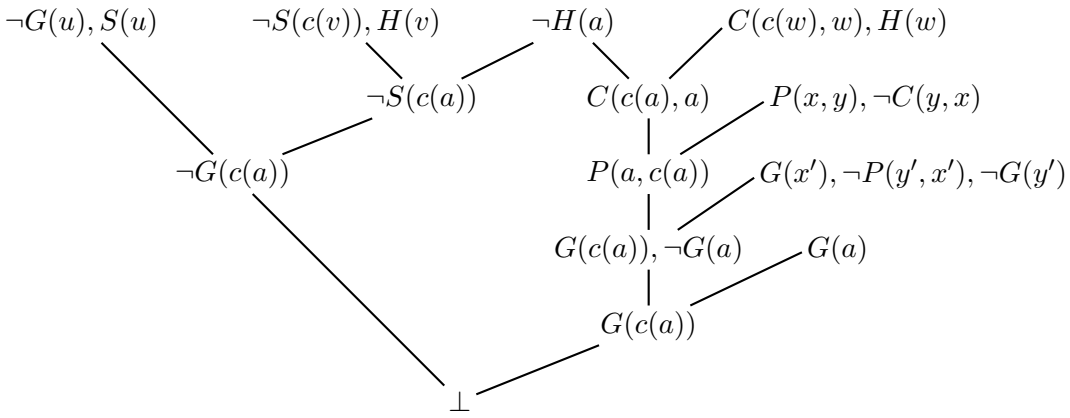
We see that the existentially quantified y needs to be Skolemized. Let us use the function symbol p, so that $p(x)$ reads "parent of x".
Similarly, let us simplify the fifth formula. Replacing the implication symbols, we get $\forall x[\neg \forall y[\neg C(y,x) \vee S(y)] \vee H(x)]$. Pushing the negations in, we then get

$$\forall x[\exists y[C(y,x) \wedge \neg S(y)] \vee H(x)]$$

Again, the existentially quantified y needs to be Skolemized, and we must use a fresh function symbol—let us choose c, so that $c(x)$ reads "child of x".
We can now list the clauses:

i. Two clauses: $\{\neg P(x,y), C(y,x)\}$ and $\{P(x,y), \neg C(y,x)\}$
ii. Two clauses: $\{G(x), R(x)\}$ and $\{\neg G(x), \neg R(x)\}$
iii. Three clauses: $\{\neg G(x), P(p(x),x)\}$, $\{\neg G(x), G(p(x))\}$, and $\{G(x), \neg P(y,x), \neg G(y)\}$
iv. One clause: $\{\neg G(x), S(x)\}$
v. Two clauses: $\{C(c(x),x), H(x)\}$ and $\{\neg S(c(x)), H(x)\}$

(c) The statement to prove is $\forall x(G(x) \rightarrow H(x))$. Negating this statement, we have $\exists x(G(x) \wedge \neg H(x))$. In clausal form this is $G(a)$ and $\neg H(a)$ (two clauses). Altogether we now have 12 clauses, but fortunately a refutation can be found that uses just seven:



P5.12 The barber paradox is a variant of Russell's paradox: say there is a barber who shaves those and *only* those who do not shave themselves. Does the barber shave themselves? The question has no answer: both "yes" and "no" immediately lead to a contradiction.

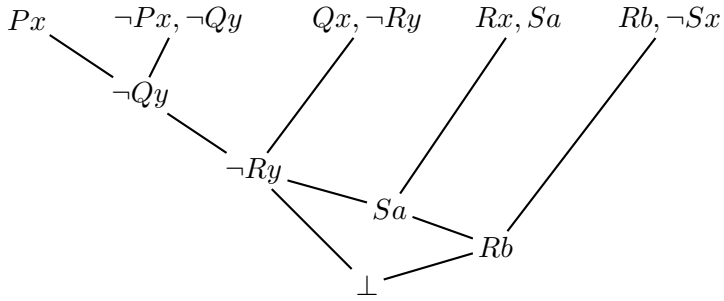
Using $B(x)$ to mean "x is a barber" and $S(u, v)$ to mean "u shaves v", translate the premise of the paradox into predicate logic. Then, using resolution (with factoring!), show that it is unsatisfiable.

P5.13 Consider the following unsatisfiable set of clauses:

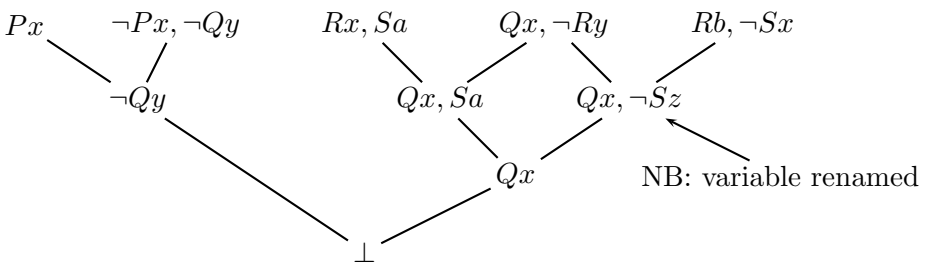
$$\{\{P(x)\}, \{\neg P(x), \neg Q(y)\}, \{Q(x), \neg R(y)\}, \{R(x), S(a)\}, \{R(b), \neg S(x)\}\}$$

What is the simplest refutation proof, if "simplest" means "the refutation tree has minimal depth"? What is the simplest refutation proof, if "simplest" means "the refutation tree has fewest nodes"?

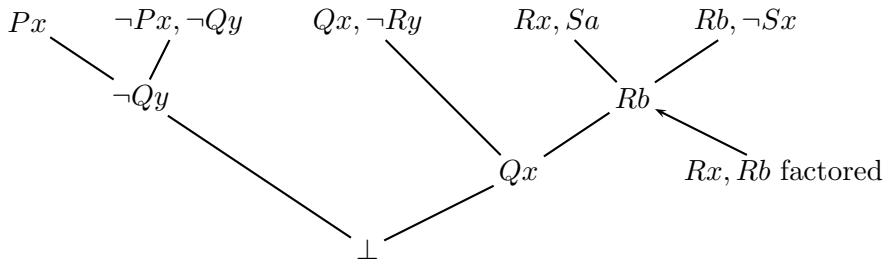
Solution: (We should rename clauses apart, but in this case, no confusion arises, so we omit that.) We can construct the refutation in 5 resolution steps, that is, the refutation tree has only 5 internal nodes:



Here is another way (5 steps), in which the depth of the refutation tree is somewhat smaller:



With factoring, we can do it in 4 resolution steps, plus one factoring step:



P5.14 Consider these statements:

S_1: "No politician is honest."
S_2: "Some politicians are not honest."
S_3: "No Australian politician is honest."
S_4: "All honest politicians are Australian."

(a) Using the predicate symbols P and H for being a politician and being honest, respectively, express S_1 and S_2 as formulas of predicate logic F_1 and F_2.

(b) Is $F_1 \to F_2$ satisfiable?

(c) Is $F_1 \to F_2$ valid?

(d) Using the predicate symbol A for "is Australian", express S_3 and S_4 in clausal form.

(e) Using resolution, show that S_1 is a logical consequence of S_3 and S_4.

(f) Prove or disprove the statement "S_2 is a logical consequence of S_3 and S_4."

Solution:

(a) The two statements

S_1:	"No politician is honest."	become	F_1	: $\forall x\ (\neg P(x) \vee \neg H(x))$
S_2:	"Some politicians are not honest."		F_2	: $\exists x\ (P(x) \wedge \neg H(x))$

(b) $F_1 \to F_2$ is satisfiable. First let us simplify the formula. Normally it would be a good idea to rename the bound variables, but in this case, it will be preferable to keep the x.

	$F_1 \to F_2$	
$\equiv$	$\forall x\ (\neg P(x) \vee \neg H(x)) \to \exists x\ (P(x) \wedge \neg H(x))$	spell out
$\equiv$	$\neg\forall x\ (\neg P(x) \vee \neg H(x)) \vee \exists x\ (P(x) \wedge \neg H(x))$	eliminate implication
$\equiv$	$\exists x\ (P(x) \wedge H(x)) \vee \exists x\ (P(x) \wedge \neg H(x))$	push negation in
$\equiv$	$\exists x\ ((P(x) \wedge H(x)) \vee (P(x) \wedge \neg H(x)))$	$\exists$ distributes over $\vee$
$\equiv$	$\exists x\ (P(x) \wedge (H(x) \vee \neg H(x)))$	factor out $P(x)$
$\equiv$	$\exists x\ P(x)$	eliminate trivially true conjunct

For this formula we can clearly find a model that makes it true. For example, take the universe $\{alf, bill, charlie\}$ and let P and H hold for all elements. Or, take the universe $\mathbb{Z}$, let P stand for "is a prime" and let H stand for "is zero".

(c) $F_1 \to F_2$ is not valid. It is easy to find a model that makes $\exists x\ P(x)$ false. For example, take the universe $\{alf, bill, charlie\}$ and let P hold for none of the elements (H can be given any interpretation). Or, take the universe $\mathbb{Z}$, let P stand for "is an even prime greater than 2" and let H stand for "is zero".

(d) The statements

S_3: "No Australian politician is honest."
S_4: "All honest politicians are Australian."

can be expressed

$$S_3: \quad \forall x\ ((A(x) \wedge P(x)) \rightarrow \neg H(x))$$
$$S_4: \quad \forall y\ ((P(y) \wedge H(y)) \rightarrow A(y))$$

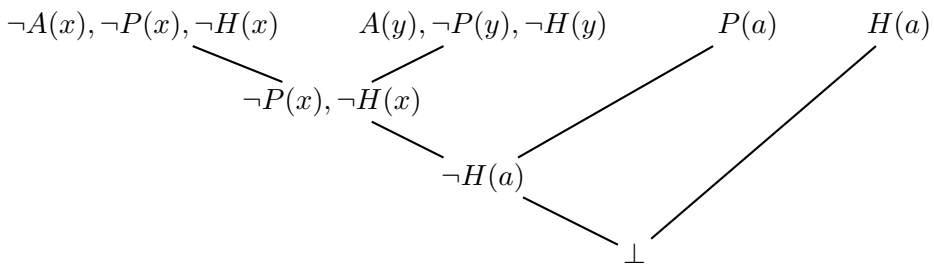
Each of these formulas corresponds to exactly one clause. The clausal forms are:

$$\{\{\neg A(x), \neg H(x), \neg P(x)\}\}$$
$$\{\{A(y), \neg H(y), \neg P(y)\}\}$$

(e) We can show that S_1 is a logical consequence of S_3 and S_4 by refuting $S_3 \wedge S_4 \wedge \neg S_1$. So let us write $\neg S_1$ in clausal form (note that we *must* apply the negation *before* "clausifying"; the other way round generally gives an incorrect result):

$\neg\forall x\ (\neg P(x) \vee \neg H(x))$	
$\exists x\ (P(x) \wedge H(x))$	push negation in
$P(a) \wedge H(a)$	Skolemize

Or, written as a set of sets: $\{\{P(a)\}, \{H(a)\}\}$. Added to the other clauses, these allow us to complete the proof by resolution:



(f) The statement "S_2 is a logical consequence of S_3 and S_4" is false. We can show this by constructing a model which makes S_3 and S_4 true, while making S_2 false. Any model with universe D, in which P is false for all elements of D, will do.

P5.15 Consider a model M with universe $U = \{\text{Jemima}, \text{Thelma}, \text{Louise}\}$ and interpretation function I such that

$$I(a) = \text{Jemima}, \quad I(b) = \text{Thelma}, \quad I(c) = \text{Louise},$$
$$I(F) = \{(\text{Jemima}, \text{Louise}), (\text{Thelma}, \text{Jemima}), (\text{Thelma}, \text{Thelma}), (\text{Louise}, \text{Thelma})\},$$
$$I(M) = \{\text{Jemima}, \text{Louise}\}.$$

Let v be a variable assignment such that $v(x) = \text{Louise}, v(y) = \text{Thelma}, v(z) = \text{Jemima}$.

For each of the following formulas, determine whether it is true or false in the model M under the variable assignment v. In each instance, prove your claim from the formal semantics.

(i) $F(x, a)$
(ii) $\exists y\, F(x, y)$
(iii) $\forall x \exists y\, F(x, y)$
(iv) $\forall y\, F(b, y)$
(v) $\exists x \forall y\, F(x, y)$

P5.16 Show that following "equivalences" are incorrect, by specifying a model which makes one formula true and the other false.

(i) $\exists x(F(x) \wedge G(x)) \stackrel{?}{\equiv} \exists x F(x) \wedge \exists x G(x)$

(ii) $\forall x(F(x) \vee G(x)) \stackrel{?}{\equiv} \forall x F(x) \vee \forall x G(x)$

(iii) $\forall x \exists y R(x, y) \stackrel{?}{\equiv} \exists x \forall y R(x, y)$

School of Computing and Information Systems
COMP30026 Models of Computation
Week 7: Regular Languages, DFAs and NFAs

Exercises

T7.1 Consider the two languages $L_1 = \{\texttt{ab}, \texttt{c}\}$ and $L_2 = \{\texttt{ca}, \texttt{c}\}$. What strings are in $(L_1 \cup L_2) \circ L_2$ What about $L_1^* \setminus L_2^*$?

Solution: For languages $L_1 = \{\texttt{ab}, \texttt{c}\}$ and $L_2 = \{\texttt{ca}, \texttt{c}\}$:

- $(L_1 \cup L_2) \circ L_2 = \{\texttt{abca}, \texttt{abc}, \texttt{cca}, \texttt{cc}, \texttt{caca}, \texttt{cac}\}$
- $L_1^* \setminus L_2^*$ is an infinite language, which contains strings like `abcababccc`, since that's a concatenation of seven strings from L_1 and it's not a concatenation of any number of strings from L_2. It does not contain strings like ϵ and `ccc`, since both of these strings are in L_2^*. In fact we can see that any string containing a `b` is not in L_2^*, so the only thing left to remove from L_1^* are zero or more concatenations of the string `c`, which are all present in L_2^*. So we can say that $L_1^* \setminus L_2^* = \{\texttt{ab}, \texttt{c}\}^* \setminus \{\texttt{c}\}^*$.

T7.2 Draw DFAs recognising the following languages. Assume that the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.

(i) $\{w \mid w$ begins with a `1` and ends with a `0`$\}$

(ii) $\{w \mid w$ contains the substring `0101`$\}$ (so $w = x\texttt{0101}y$ for some strings x and y)

(iii) $\{w \mid w$ is any string except `11` and `111`$\}$

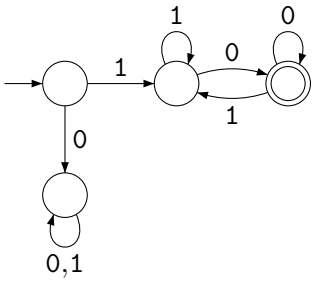
(iv) $\{\epsilon\}$

(v) The empty set

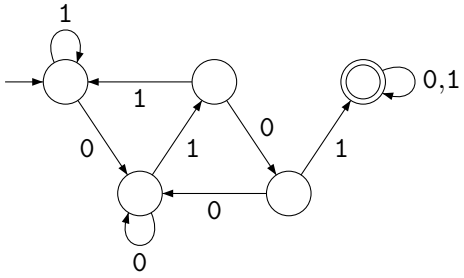
Continued in P7.1

Solution:

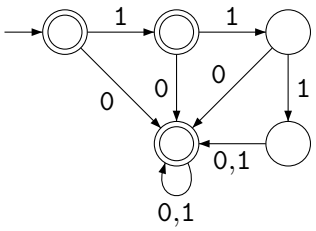
(i) $\{w \mid w$ begins with a `1` and ends with a `0`$\}$



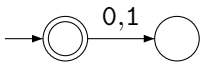
(ii) $\{w \mid w$ contains the substring `0101`$\}$



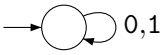
(iii) $\{w \mid w$ is any string except `11` and `111`$\}$



(iv) $\{\epsilon\}$



(v) The empty set (any DFA with an empty set of accept states will do)



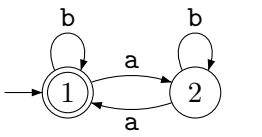
T7.3 The language

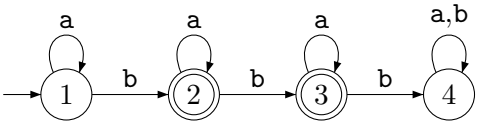
$$\{w \mid w \text{ has an even number of } \texttt{a}\text{s and one or two } \texttt{b}\text{s}\}$$

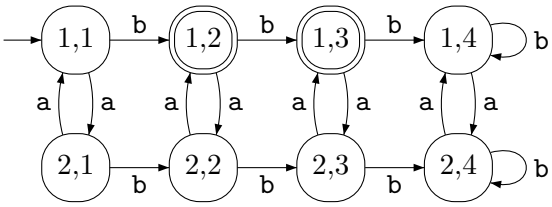
over the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$, can be written as the intersection of two simpler languages. First construct the DFAs for the simpler languages, then combine to get a DFA for the intersection using the following idea: If the set of states for DFA D_1 is Q_1 and the set of states for D_2 is Q_2, we let the set of states for the combined DFA D be $Q_1 \times Q_2$. We construct D so that, having consumed a string s, D will be in state (q_1, q_2) iff D_1 is in state q_1, and D_2 is in state q_2 when they have consumed s.

Continued in P7.2 & P7.4

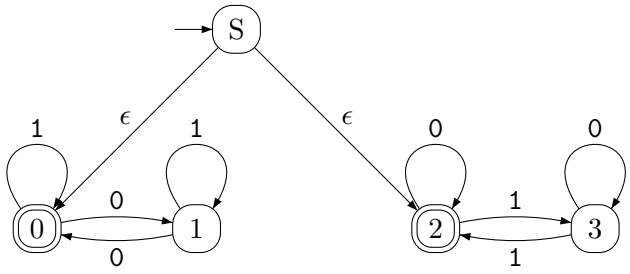
Solution: $\{w \mid w \text{ has an even number of } \texttt{a}\text{s}\} \cap \{w \mid w \text{ has one or two } \texttt{b}\text{s}\}$





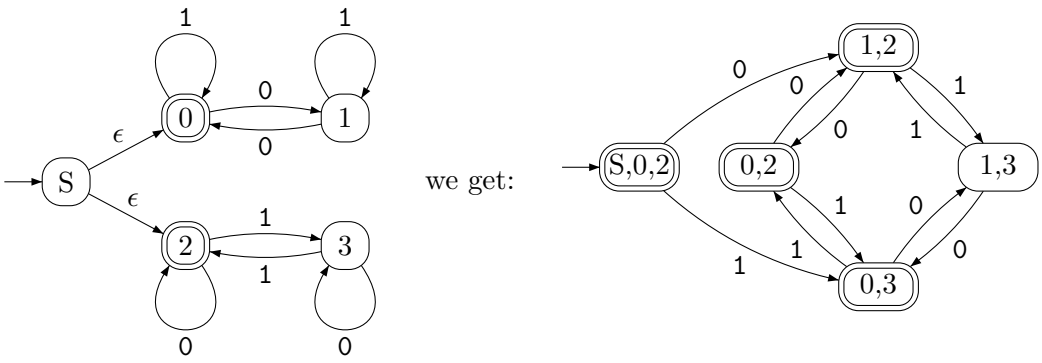


T7.4 Use the subset construction method to turn the following NFA into an equivalent DFA.

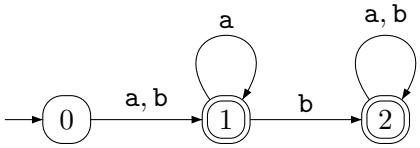


Continued in P7.6 & P7.7

Solution: From this NFA:



T7.5 Use the minimization algorithm to turn this DFA into a minimal DFA which recognises the same language.



Continued in P7.8 & P7.9

Solution: The minimisation algorithm is presented as a construction on NFAs, but we can simply view a DFA as an NFA everytime we need to reverse one. Formally this involves taking a DFA $D = (Q, \Sigma, \delta, q_0, F)$, and converting it to an NFA $(Q, \Sigma, \delta', \{q_0\}, F)$, where $\delta'(q, x) = \{\delta(q, x)\}$.

The steps of the minimisation algorithm are as follows

(1) Reverse the NFA
(2) Determinise the result
(3) Reverse again
(4) Determinise

Determinising is applying the subset construction from the last exercise. Reversing is swapping the sets of initial and accept states, and swapping the source and target of each transition.

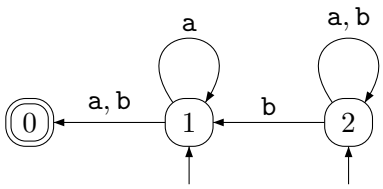
So the reversal of the NFA $(Q, \Sigma, \delta, I, F)$ is $(Q, \Sigma, \delta', F, I)$, where

$$\delta'(q, x) = \{q' \in Q \mid q \in \delta(q', x)\}$$

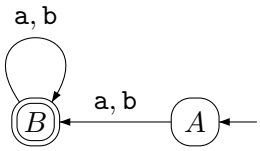
Think carefully about why this corresponds to swapping the direction of the transition arrows in an NFA. You will need to first understand how to look at the drawing of an NFA and interpret what it's transition function, $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is.

Reversing only makes sense if the set of accept states is non-empty, since the set of initial states must always be non-empty. We could skip the whole minimising processes for these NFAs with no accept states by simply constructing the one-state DFA that rejects every string.

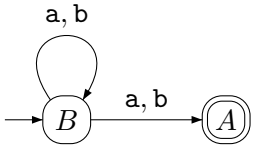
Since there is at least one accept state, we can reverse this DFA to get the following NFA.



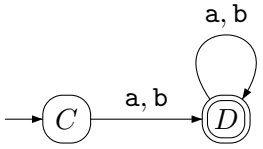
We determinise this NFA, and relabel such that $A = \{1, 2\}, B = \{0, 1, 2\}$



Then we reverse again, to get the following NFA.



which we can determinise, with states $C = \{B\}$ and $D = \{A, B\}$.



Homework problems

P7.1 Draw DFAs recognising the following languages. Assume that the alphabet $\Sigma = \{0, 1\}$.

(i) $\{w \mid w$ is not empty and contains only 0s or only 1s$\}$

(ii) $\{w \mid w$ has length at least 3 and its third symbol is 0$\}$

(iii) $\{w \mid$ the length of w is at most 5$\}$

(iv) $\{w \mid$ the length of w is a multiple of 3$\}$

(v) $\{w \mid$ every odd position of w is a 1$\}$

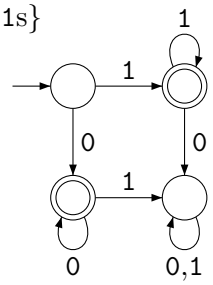
(vi) $\{w \mid w$ contains at least two 0s and at most one 1$\}$

(vii) $\{w \mid$ the last symbol of w occurs at least twice in $w\}$

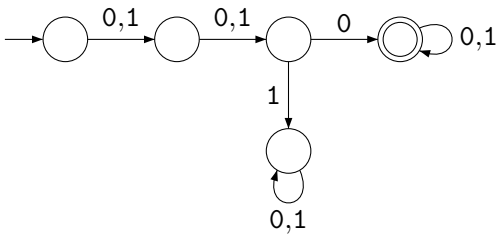
(viii) All strings except the empty string

Solution:

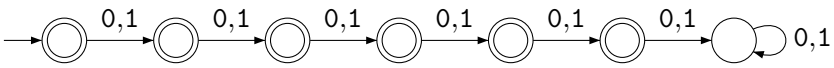
(a) $\{w \mid w$ is not empty and contains only 0s or only 1s$\}$



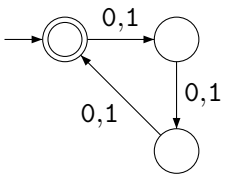
(b) $\{w \mid w$ has length at least 3 and its third symbol is 0$\}$



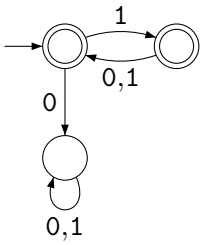
(c) $\{w \mid$ the length of w is at most 5$\}$



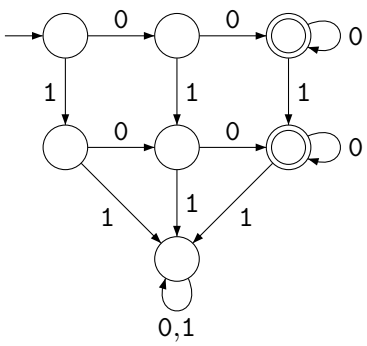
(d) $\{w \mid$ the length of w is a multiple of 3$\}$



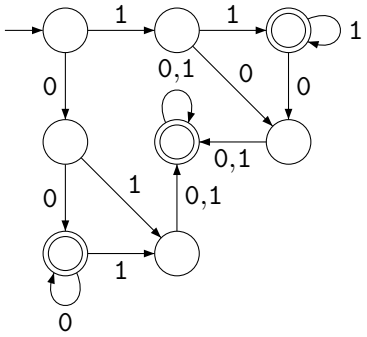
(e) $\{w \mid$ every odd position of w is a 1$\}$



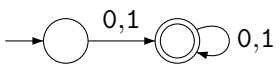
(f) $\{w \mid w$ contains at least two `0`s and at most one `1`$\}$



(g) $\{w \mid$ the last symbol of w occurs at least twice in $w\}$



(h) All strings except the empty string



P7.2 Each of the following languages is the intersection of two simpler languages. First construct the DFAs for the simpler languages, then combine them using the construction from T7.3 to create a DFA for the intersection language

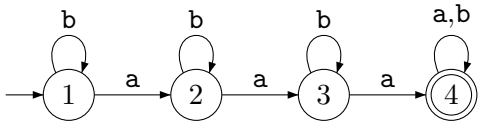
(a) $\{w \mid w$ has at least three `a`s and at least two `b`s$\}$

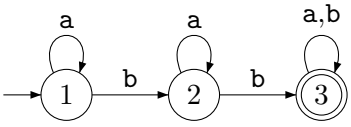
(b) $\{w \mid w$ has an odd number of `a`s and ends with `b`$\}$

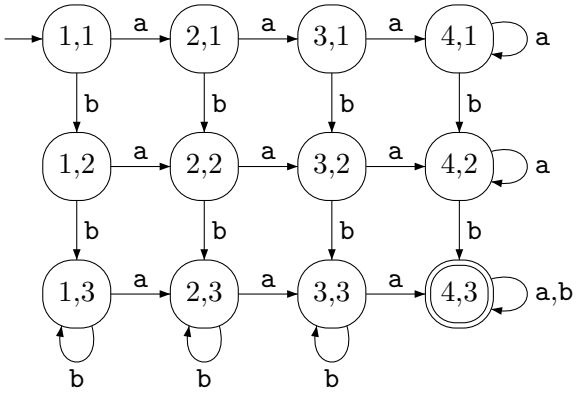
(c) $\{w \mid w$ has an odd number of `a`s and has even length$\}$

Solution:

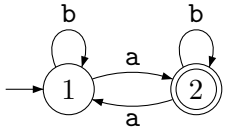
(a) $\{w \mid w$ has at least three `a`s$\} \cap \{w \mid w$ has at least two `b`s$\}$

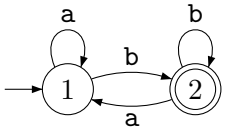


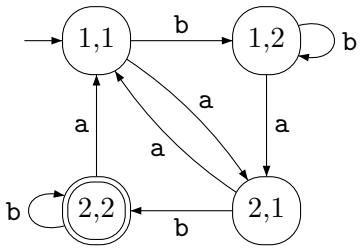




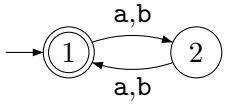
(b) $\{w \mid w \text{ has an odd number of } \texttt{a}\text{s}\} \cap \{w \mid w \text{ ends with } \texttt{b}\}$

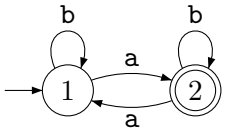


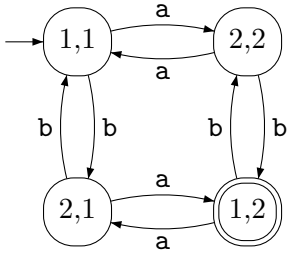




(c) $\{w \mid w \text{ has an even length}\} \cap \{w \mid w \text{ has an odd number of } \texttt{a}\text{s}\}$







P7.3 Each of the following languages is the complement of a simpler language. Again, the best way to proceed is to first construct a DFA for the simpler language, then find a DFA for the complement by transforming that DFA appropriately. Throughout this question, assume that the alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$.

(a) $\{w \mid w \text{ does not contain the substring } \texttt{bb}\}$

(b) $\{w \mid w \text{ contains neither the substring } \texttt{ab} \text{ nor } \texttt{ba}\}$

(c) $\{w \mid w \text{ is any string not in } \texttt{A}^* \circ \texttt{B}^*, \text{ where } \texttt{A} = \{\texttt{a}\}, \texttt{B} = \{\texttt{b}\}\}$

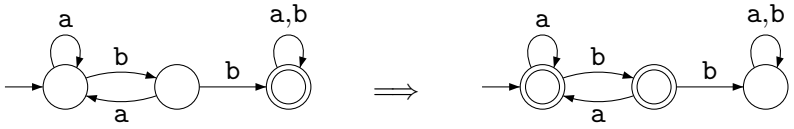
(d) $\{w \mid w \text{ is any string not in } \texttt{A}^* \cup \texttt{B}^*, \text{ where } \texttt{A} = \{\texttt{a}\}, \texttt{B} = \{\texttt{b}\}\}$

(e) $\{w \mid w \text{ is any string that doesn't contain exactly two } \texttt{a}\text{s}\}$

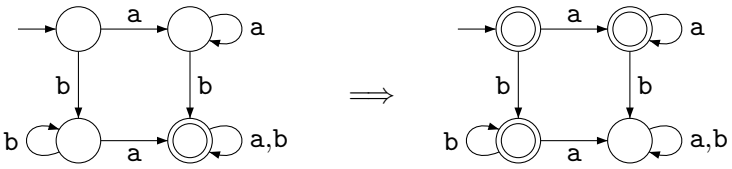
(f) $\{w \mid w \text{ is any string except } \texttt{a} \text{ and } \texttt{b}\}$

Solution:

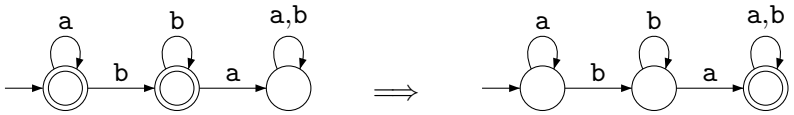
(a) $\{w \mid w \text{ does not contain the substring } \texttt{bb}\}$



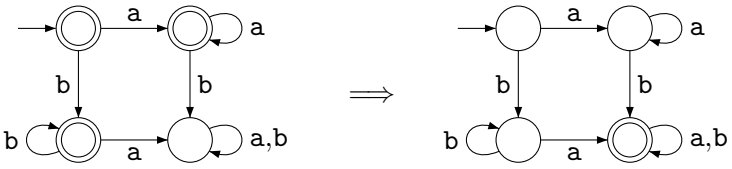
(b) $\{w \mid w$ contains neither the substring ab nor ba$\}$



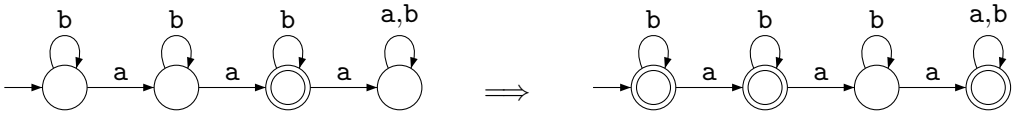
(c) $\{w \mid w$ is any string not in $\mathtt{A}^* \circ \mathtt{B}^*$, where $\mathtt{A} = \{\mathtt{a}\}, \mathtt{B} = \{\mathtt{b}\}\}$



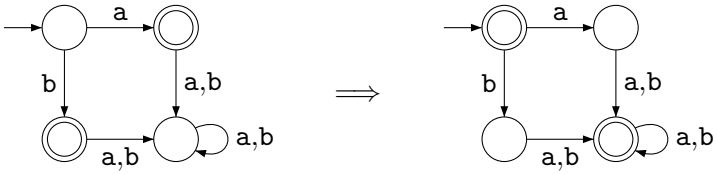
(d) $\{w \mid w$ is any string not in $\mathtt{A}^* \cup \mathtt{B}^*$, where $\mathtt{A} = \{\mathtt{a}\}, \mathtt{B} = \{\mathtt{b}\}\}$ (compare to (b)!)



(e) $\{w \mid w$ is any string that doesn't contain exactly two as$\}$



(f) $\{w \mid w$ is any string except a and b$\}$

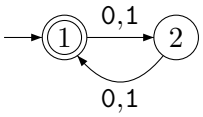


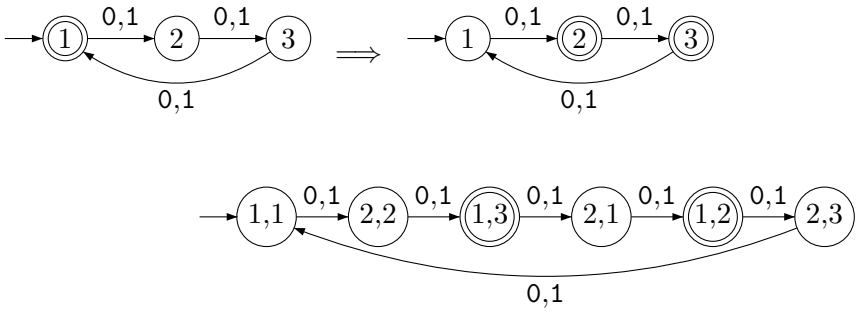
P7.4 The language

$$L = \{w \mid \text{the length of } w \text{ is a multiple of 2 and is not multiple of 3}\}$$

is the difference of two simpler languages. First construct DFAs for the simpler languages, then construct a DFA for L. Assume the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

Solution: $\{w \mid$ the length of w is a multiple of 2 and is not multiple of 3$\}$





P7.5 A language is regular iff it is recognised by some DFA. Prove that for any languages $L, K \subseteq \Sigma^*$

(i) If L is regular then L^c is regular

(ii) If L and K are both regular and then $L \cap K$ is regular

(iii) If L and K are both regular and then $L \setminus K$ is regular

Another way to say this is "The class of regular languages is closed under intersection, complement and difference". You have demonstrated this already for some example languages. For this exercise you should specify how this works in general, using the formal definition of a DFA. *Hint:* Your proof for (iii) can be much shorter.

Solution:

(i) Suppose L is regular. Then there is some DFA $D = (Q, \Sigma, \delta, q_0, F)$ which recognises L. Another way to say this is that the language recognised by D is exactly L. We define the language recognised by D as the set

$$L(D) = \{w \in \Sigma^* \mid D \text{ accepts } w\}$$

We claim that L^c is regular, so we must show that there is a DFA D' such that $L(D') = L^c$. Let $D' = (Q, \Sigma, \delta, q_0, Q \setminus F)$, i.e. it has the exact same set of states, transition function and start state as D, but all the non-accept states are now accept states (and vice versa). Then we claim that $L(D') = L^c$, since

$$\begin{aligned} L(D') &= \{w \in \Sigma^* \mid D' \text{ accepts } w\} \\ &= \{w \in \Sigma^* \mid D \text{ rejects } w\} \\ &= \{w \in \Sigma^* \mid w \notin L(D)\} \\ &= \{w \in \Sigma^* \mid w \notin L\} \\ &= L^c \end{aligned}$$

Hence L^c is regular, since the DFA D' recognises it. The core of this proof is the step "D' accepts w iff D rejects w", which can be shown by unwrapping the definition of *acceptance* for DFAs. Another way to explain it, is that if D' rejects w, then after running D' on input w, it should finish in a reject state, $q \notin Q \setminus F$, since $Q \setminus F$ is the set of accept states of D'. But $q' \notin Q \setminus F$ iff $q \in F$. So if we run D on w, it will take the exact same transitions and move through the same states as in D', ending with q, and $q \in F$, so D must accept w. We can reason similarly to show that if D accepts w then D' rejects w.

(ii) Before we dive into the proof, note that we assume L and K are languages over the same alphabet. If we wanted to intersect languages over distinct alphabets, we could think

of them as languages over the union of their alphabets. Suppose L and K are regular languages. Then there are DFAs

$$D_L = (Q_L, \Sigma, \delta_L, q_{L0}, F_L)$$
$$D_K = (Q_K, \Sigma, \delta_K, q_{K0}, F_K)$$

such that $L(D_L) = L$ and $L(D_K) = K$. Define

$$D' = (Q_L \times Q_K, \Sigma, \delta', (q_{L0}, q_{K0}), F_L \times F_K)$$

where $\delta' : (Q_L \times Q_K) \times \Sigma \to Q_L \times Q_K$ is defined

$$\delta'((q_1, q_2), x) = (\delta_L(q_1, x), \delta_K(q_2, x))$$

Check that this definition makes sense, by inspecting the function signature of $\delta_L : Q_L \times \Sigma \to Q_L$ and $\delta_K : Q_K \times \Sigma \to Q_K$, and δ'.
We claim that $L(D') = L \cap K$. There are a few ways of reasoning about this. Firstly we could show by induction on the length of the input string that if D_L is in state $q_1 \in Q_L$ after consuming input w, and D_K is in state $q_2 \in Q_K$ after consuming input w, then D' is in state (q_1, q_2) after consuming input w. This is true by definition for the empty string, since D' starts in state (q_{L0}, q_{K0}). Now suppose this true for strings of length n. We want to show that it's also true for any string of length $n + 1$. Let wx be such a string, where w is a string of length n and $x \in \Sigma$ is a symbol. Then after consuming input w on each of D_L, D_K, D', if D_L is in state q_1 and D_K is in state q_2, we know by the inductive hypothesis that D' is in state (q_1, q_2). After then consuming x on all three machines, we know that D_L will be in state $\delta_L(q_1, x)$, and D_K will be in state $\delta_K(q_2, x)$. So we just need to show that D' will be in state $(\delta_L(q_1, x), \delta_K(q_2, x))$ after consuming x. But this is true by definition, since,

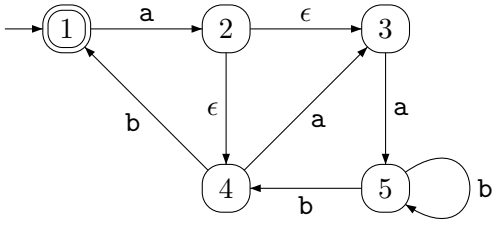
$$\delta'((q1, q2), x) = (\delta_L(q_1, x), \delta_L(q_2, x))$$

Then we can show $L(D') = L \cap K$ directly, since

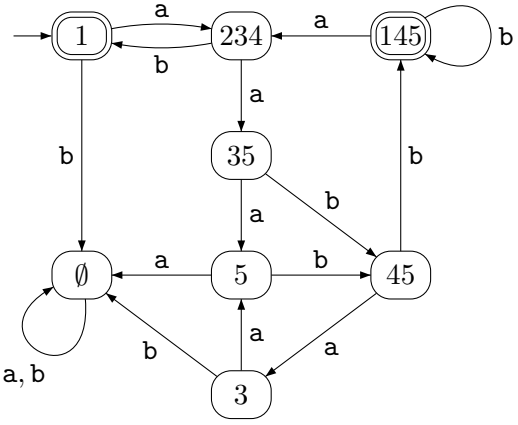
$$\begin{aligned}
L(D') &= \{w \in \Sigma^* \mid D' \text{ accepts } w\} \\
&= \{w \in \Sigma^* \mid D' \text{ is in state } (q_1, q_2) \text{ after consuming } w \text{ and } (q_1, q_2) \in F_L \times F_K\} \\
&= \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} D_L \text{ is in state } q_1 \text{ after consuming } w \text{ and } q_1 \in F_L, \\ D_K \text{ is in state } q_2 \text{ after consuming } w \text{ and } q_2 \in F_K \end{array} \right\} \\
&= \left\{ w \in \Sigma^* \;\middle|\; \begin{array}{l} D_L \text{ accepts } w, \\ D_K \text{ accepts } w \end{array} \right\} \\
&= \{w \in \Sigma^* \mid D_L \text{ accepts } w\} \cap \{w \in \Sigma^* \mid D_K \text{ accepts } w\} \\
&= L \cap K
\end{aligned}$$

(iii) Suppose L and K are both regular. Then K^c is regular using (i), and therefore $L \cap K^c$ is regular using (ii). But $L \setminus K = L \cap K^c$, so we are done.

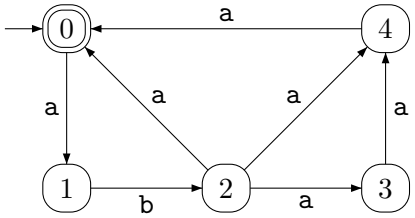
P7.6 Use the subset construction method to turn this NFA into an equivalent DFA:



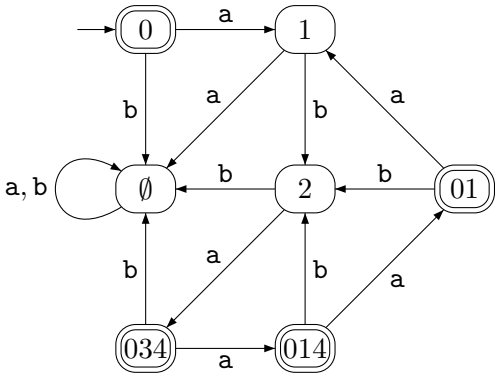
Solution:



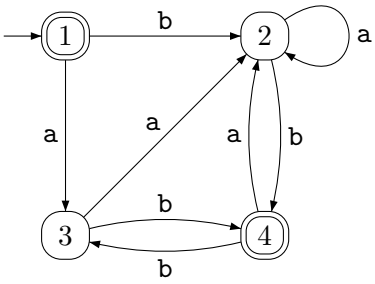
P7.7 Use the subset construction method to turn this NFA into an equivalent DFA:



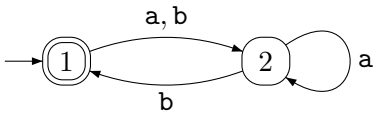
Solution:



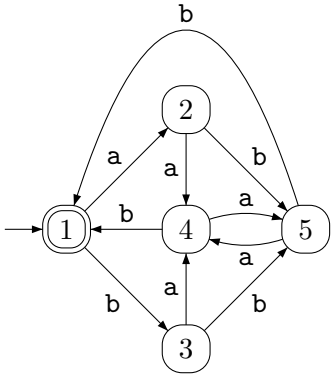
P7.8 Construct a minimal DFA equivalent to this one:



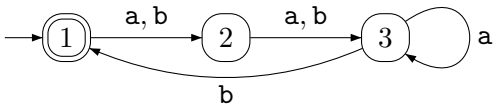
Solution:



P7.9 Construct a minimal DFA equivalent to this one:



Solution:



School of Computing and Information Systems
COMP30026 Models of Computation
Week 8: Regular Expressions, Context-Free Grammars, and the Pumping Lemma for Regular Languages

Exercises

T8.1 Give regular expressions for the following languages over the alphabet $\Sigma = \{0, 1\}$.

(i) $\{w \mid w$ begins with a `1` and ends with a `0`$\}$

(ii) $\{w \mid w$ contains the substring `0101`$\}$

(iii) $\{w \mid$ the length of w is at most 5$\}$

(iv) $\{w \mid w$ contains at least two `0`s and at most one `1`$\}$

(v) $\{w \mid w$ is any string except the empty string$\}$

Continued in P8.1

Solution:

(i) $1(0 \cup 1)^*0$

(ii) $(0 \cup 1)^*0101(0 \cup 1)^*$

(iii) $(\epsilon \cup 0 \cup 1)(\epsilon \cup 0 \cup 1)(\epsilon \cup 0 \cup 1)(\epsilon \cup 0 \cup 1)(\epsilon \cup 0 \cup 1)$

(iv) $0^*(00 \cup 001 \cup 010 \cup 100)0^*$

(v) $(0 \cup 1)(0 \cup 1)^*$

T8.2 Give context-free grammars for the following languages. Assume the alphabet is $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

(i) $\{w \mid$ the length of w is odd and its middle symbol is `0`$\}$

(ii) $\{w \mid w$ is a palindrome$\}$

Continued in P8.5 & P8.6

Solution:

(i)
$$S \;\to\; 0 \mid 0\ S\ 0 \mid 0\ S\ 1 \mid 1\ S\ 0 \mid 1\ S\ 1$$

(ii)
$$S \;\to\; 0\ S\ 0 \mid 1\ S\ 1 \mid 0 \mid 1 \mid \epsilon$$

T8.3 A context-free grammar is called *ambiguous* if there is a string in its language which has two distinct parse-trees. A context-free language is called ambiguous (or *inherently ambiguous*), if all of its context-free grammars are ambiguous. Consider the context-free grammar $(\{A, B, T\}, \{\mathtt{a}, \mathtt{b}\}, R, T)$ with rules R:

$$\begin{array}{lcl} T & \to & A \mid B \\ A & \to & \mathtt{a}\ \mathtt{b} \mid \mathtt{a}\ A\ \mathtt{b} \\ B & \to & \epsilon \mid \mathtt{a}\ \mathtt{b}\ B \end{array}$$

Show that G is ambiguous, but $L(G)$ is not ambiguous.

Continued in P8.12

Solution: The grammar is ambiguous because ab can be derived from A and also from B (and we can get either A or B from the start variable T). However, it is the only string that can be derived from both, so we can make this grammar unambiguous simply by making sure that ab cannot be derived from A, or more precisely, making sure that the set of strings that can be derived from A is $\{\mathtt{a}^n\mathtt{b}^n \mid n > 1\}$. To do this, change the first rule for A like so:

$$\begin{array}{lcl} T & \to & A \mid B \\ A & \to & \mathtt{a\ a\ b\ b} \mid \mathtt{a}\ A\ \mathtt{b} \\ B & \to & \epsilon \mid \mathtt{a\ b}\ B \end{array}$$

Since this grammar recognises the same language, and is not ambiguous, we've shown that the language is not ambiguous.

T8.4 The pumping lemma for regular languages is the following statement: if A is a regular language, then there exists an integer $p \geq 1$ (called the *pumping length*) such that, for any string $s \in A$ with $|s| \geq p$, there exist strings x, y and z such that $s = xyz$ and

- $y \neq \epsilon$,
- $|xy| \leq p$,
- $xy^iz \in A$ for all integers $i \geq 0$.

Use the pumping lemma for regular languages to prove that the following language is not regular:

$$A = \{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^n \mid n \geq 0\}$$

Continued in P8.4

Solution: Suppose to the contrary that A is regular, and let p be the pumping length. Consider $s = \mathtt{0}^p\mathtt{1}^p\mathtt{2}^p \in A$. Since $|s| \geq p$, by the pumping lemma, we have $s = xyz$ for some x, y, z such that that $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in A$ for all $i \geq 0$. In particular, we have $xz \in A$. But since $|xy| \leq p$, it follows that y only contains 0's and that z ends with $\mathtt{1}^p\mathtt{2}^p$. Thus xz has strictly fewer 0's than 1's or 2's, and hence $xz \notin A$. Contradiction! Therefore A is not regular.

Homework problems

P8.1 Give regular expressions for the following languages over the alphabet $\Sigma = \{0, 1\}$.

(i) $\{w \mid w$ has length at least 3 and its third symbol is $\mathtt{0}\}$

(ii) $\{w \mid$ every odd position of w is a $\mathtt{1}\}$

(iii) $\{\epsilon, \mathtt{0}\}$

(iv) The empty set

Bonus: Draw minimal NFAs for these languages, and those from T8.1

Solution:

(i) $(0 \cup 1)(0 \cup 1)0(0 \cup 1)^*$

(ii) $(1(0 \cup 1))^*(\epsilon \cup 1)$

(iii) $\epsilon \cup 0$

(iv) $\emptyset$

P8.2 String s is a *suffix* of string t iff there exists some string u (possibly empty) such that $t = us$. For any language L we can define the set of suffixes of strings in L:

$$\mathit{suffix}(L) = \{x \mid x \text{ is a suffix of some } y \in L\}$$

Let A be any regular language. Show that $\mathit{suffix}(A)$ is a regular language. *Hint:* think about how a DFA for A can be transformed to recognise $\mathit{suffix}(A)$.

Solution: If A is regular then $\mathit{suffix}(A)$ is regular. Namely, let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA for A. Assume every state in Q is *reachable* from q_0. Then we can turn D into an NFA N for $\mathit{suffix}(A)$ by adding a new state q_{-1} which becomes the NFA's start state. For each state $q \in Q$, we add an epsilon transition from q_{-1} to q.

That is, we define N to be $(Q \cup \{q_{-1}\}, \Sigma, \delta', q_{-1}, F)$, with transition function

$$\delta'(q, x) = \begin{cases} \{\delta(q, x)\} & \text{for } q \in Q \text{ and } x \in \Sigma \\ Q & \text{for } q = q_{-1} \text{ and } x = \epsilon \\ \emptyset & \text{for } q \neq q_{-1} \text{ and } x = \epsilon \end{cases}$$

The restriction we assumed, that all of D's states are reachable, is not a severe one. It is easy to identify unreachable states and eliminate them (which of course does not change the language of the DFA). To see why we need to eliminate unreachable states before generating N in the suggested way, consider what happens to this DFA for $\{\epsilon\}$: $(\{q_0, q_1, q_2\}, \{\mathtt{a}\}, \delta, q_0, \{q_0\})$, where $\delta(q_0, \mathtt{a}) = \delta(q_1, \mathtt{a}) = q_1$ and $\delta(q_2, \mathtt{a}) = q_0$.

P8.3 In general it is difficult, given a regular expression, to find a regular expression for its complement. However, it can be done, and you have been given all the necessary tricks and algorithms. This question asks you to go through the required steps for a particular example.

Consider the regular language $(\mathtt{ba^*a})^*$. Assuming the alphabet is $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, we want to find a regular expression for its complement, that is, for

$$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ is not in } (\mathtt{ba^*a})^*\}$$

To complete this task, go through the following steps.

(a) Construct an NFA for $(\mathtt{ba^*a})^*$. Two states suffice.

(b) Turn the NFA into a DFA using the subset construction method.

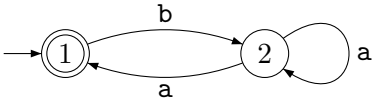
(c) Do the "complement trick" to get a DFA D for L.

(d) Reflect on the result: Wouldn't it have been better/easier to apply the "complement trick" directly to the NFA?

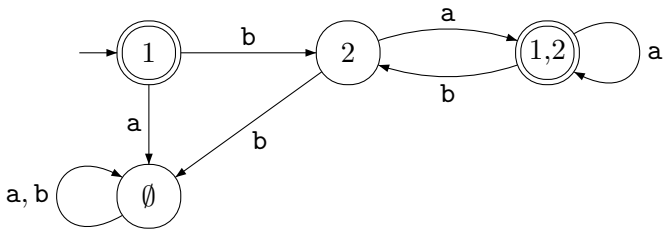
(e) Turn DFA D into a regular expression for L using the NFA-to-regular-expression conversion process shown in the lecture on regular expressions.

Solution:

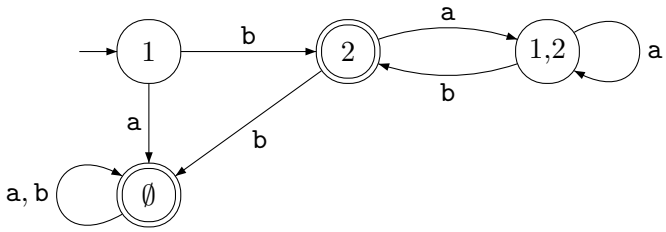
(a) Here is an NFA for $(\mathtt{ba}^*\mathtt{a})^*$:



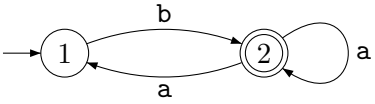
(b) Here is an equivalent DFA, obtained using the subset construction:



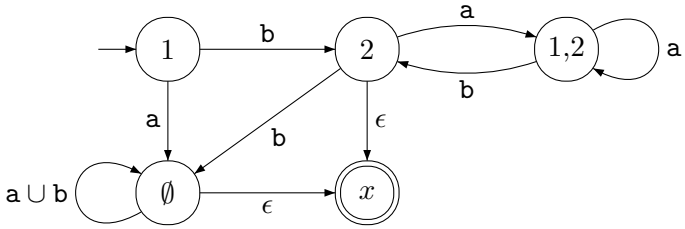
(c) It is easy to get a DFA for the complement:



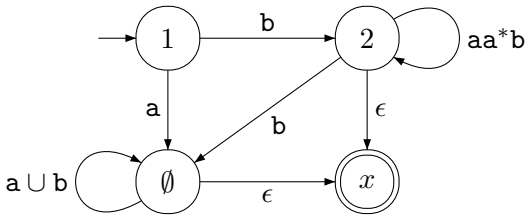
(d) It would be problematic to do the "complement trick" on the NFA, as it is only guaranteed to work on DFAs. We would get the following, which accepts, for example, baa:



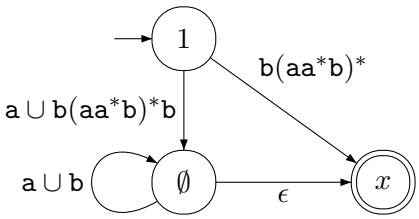
(e) Starting from the DFA that we found in (c), we make sure that we have just one accept state:



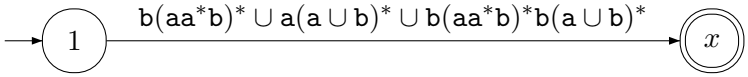
Let us first remove the state labeled 1,2:



Now we can remove state 2:



Finally, eliminating the state labelled $\emptyset$, we are left with:



The resulting regular expression can be read from that diagram.

P8.4 A *palindrome* is a string that reads the same forwards and backwards. Use the pumping lemma for regular languages and/or closure results to prove that the following languages are not regular:

(a) $B = \{\mathtt{a}^i\mathtt{b}\mathtt{a}^j \mid i > j \geq 0\}$

(b) $C = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ is not a palindrome}\}$

(c) $D = \{www \mid w \in \{\mathtt{a}, \mathtt{b}\}^*\}$

Solution:

(a) We use the pumping lemma to show that B is not regular. Assume that it were. Let p be the pumping length, and consider $\mathtt{a}^{p+1}\mathtt{b}\mathtt{a}^p$. This string is in B and of length $2p+2$. By the pumping lemma, there are strings x, y, and z such that $\mathtt{a}^{p+1}\mathtt{b}\mathtt{a}^p = xyz$, with $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in B$ for all $i \in \mathbb{N}$. By the first two conditions, y must be a non-empty string consisting of as only. But then, pumping down, we get xz, in which the number of as on the left no longer is strictly larger than the number of as on the right. Hence we have a contradiction, and we conclude that B is not regular.

(b) We want to show that $C = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ is not a palindrome}\}$ is not regular. But since regular languages are closed under complement, it will suffice to show that $C^c = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ is a palindrome}\}$ is not regular. Assume that C^c *is* regular, and let p be the pumping length. Consider $\mathtt{a}^p\mathtt{b}\mathtt{a}^p \in C^c$. Since $|s| \geq p$, by the pumping lemma, s can be written $s = xyz$, so that $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in C^c$ for all $i \geq 0$. But since $|xy| \leq p$, y must contain as only. Hence $xz \notin C^c$. We have a contradiction, and we conclude that C^c is not regular. Now, if C was regular, C^c would be regular too. Hence C cannot be regular.

(c) Suppose to the contrary that D is regular, and let p be the pumping length of D. Consider the string $s = (\mathtt{a}^p\mathtt{b})^3 \in D$. Since $|s| \geq p$, by the pumping lemma, s can be written $s = xyz$, so that $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in D$ for all $i \geq 0$. In particular, we

have $xz \in D$. Now, since $|xy| \leq p$, the string y must consist entirely of a's. Therefore $xz = \mathtt{a}^{p-|y|}\mathtt{b}(\mathtt{a}^p\mathtt{b})^2 \notin D$ since $|y| \geq 1$ and hence there are fewer a's before the first b than in the other two copies. Contradiction! Hence D is not regular.

P8.5 Give context-free grammars for the following languages. Assume the alphabet is $\Sigma = \{0, 1\}$.

(i) $\{w \mid w$ starts and ends with the same symbol$\}$

(ii) $\{w \mid$ the length of w is odd$\}$

Solution:

(a)

$$\begin{array}{lcl} S & \rightarrow & 0\ T\ 0 \mid 1\ T\ 1 \mid 0 \mid 1 \\ T & \rightarrow & 0\ T \mid 1\ T \mid \epsilon \end{array}$$

(b)

$$S \rightarrow 0 \mid 1 \mid 0\ 0\ S \mid 0\ 1\ S \mid 1\ 0\ S \mid 1\ 1\ S$$

P8.6 Construct a context-free grammar for the language $\{\mathtt{a}^i\mathtt{b}\mathtt{a}^j \mid i > j \geq 0\}$.

Solution:

$$\begin{array}{lcl} S & \rightarrow & A\ B \\ A & \rightarrow & \mathtt{a} \mid \mathtt{a}\ A \\ B & \rightarrow & \mathtt{b} \mid \mathtt{a}\ B\ \mathtt{a} \end{array}$$

P8.7 Show that the class of context-free languages is closed under the regular operations: union, concatenation, and Kleene star. *Hint:* show how context-free grammars for A and B can be manipulated to produce context-free grammars for $A \cup B$, AB, and A^*. Careful: The variables used in the grammars for A and in B could overlap.

P8.8 If we consider English words the "symbols" (or primitives) of English, we can use a context-free grammar to try to capture certain classes of sentences and phrases. For example, we can consider articles (A), nouns (N), adjectives (Q), intransitive verbs (IV), transitive verbs (TV), noun phrases (NP), verb phrases (VP), and sentences (S). List 5–10 sentences generated by this grammar:

S	$\rightarrow$	$NP\ VP$	N	$\rightarrow$	cat	IV	$\rightarrow$	hides
A	$\rightarrow$	a	N	$\rightarrow$	dog	IV	$\rightarrow$	runs
A	$\rightarrow$	the	Q	$\rightarrow$	lazy	IV	$\rightarrow$	sleeps
NP	$\rightarrow$	$A\ N$	Q	$\rightarrow$	quick	TV	$\rightarrow$	chases
NP	$\rightarrow$	$A\ Q\ N$	VP	$\rightarrow$	IV	TV	$\rightarrow$	hides
N	$\rightarrow$	bone	VP	$\rightarrow$	$TV\ NP$	TV	$\rightarrow$	likes

Are they all meaningful? Discuss "well-formed" versus "meaningful".

Solution: Here are some sentences generated from the grammar:

- A dog runs
- A dog likes a bone
- The quick dog chases the lazy cat
- A lazy bone chases a cat
- The lazy cat hides
- The lazy cat hides a bone

The grammar is concerned with the structure of well-formed sentences; it says nothing about meaning. A sentence such as "a lazy bone chases a cat" is syntactically correct—its structure makes sense; it could even be semantically correct, for example, "lazy bone" may be a derogatory characterisation of some person. But in general there is no guarantee that a well-formed sentence carries meaning.

P8.9 How would you change the grammar from the previous question so that "adverbial modifiers" such as "angrily" or "happily" can be used? For example, we would like to be able to generate sentences like "the dog barks constantly" and "the black cat sleeps quietly".

Solution: We can easily extend the grammar so that a sentence may end with an optional adverbial modifier:

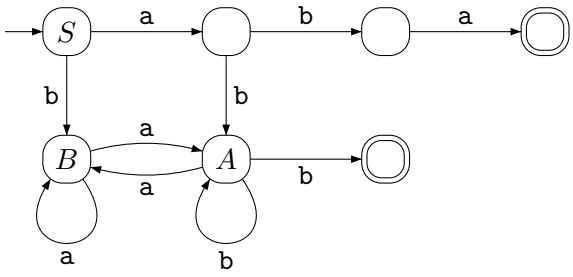
$$\begin{array}{lcl} S & \to & NP\ VP\ PP \\ & \vdots & \\ PP & \to & \epsilon \\ PP & \to & \text{quietly} \\ PP & \to & \text{all day} \\ & \vdots & \end{array}$$

P8.10 Consider this context-free grammar G with start symbol S:

$$\begin{array}{lcl} S & \to & \texttt{a b a} \mid \texttt{a b}\ A \mid \texttt{b}\ B \\ A & \to & \texttt{b} \mid \texttt{b}\ A \mid \texttt{a}\ B \\ B & \to & \texttt{a}\ A \mid \texttt{a}\ B \end{array}$$

Draw an NFA which recognises $L(G)$. *Hint:* The grammar is a regular grammar; you may want to use the labels S, A, and B for three of the NFA's states.

Solution: Here is a suitable NFA:



P8.11 Consider the context-free grammar G with rules

$$S \to \texttt{a b} \mid \texttt{a}\ S\ \texttt{b} \mid S\ S$$

Use structural induction to show that no string $w \in L(G)$ starts with `abb`.

Solution: Let w be a string in $L(G)$, that is, w is derived from S. We will use structural induction to show a stronger statement than what was required; namely we show that, for every string $w \in L(G)$, w starts with neither `b` nor `abb`. That is, if w is derived from S then it starts with neither `b` nor `abb`. (To express the property formally, we may write $\forall w' \in \{\texttt{a}, \texttt{b}\}^*(w \neq \texttt{b}w' \wedge w \neq \texttt{abb}w')$.

There is one base case: If $w =$ `ab` then w does not start `b` and it does not start with `abb`.

For the first recursive case, let $w = \texttt{a}w'\texttt{b}$, where $w' \in L(G)$. By the induction assumption, w' starts with neither `b` nor `abb`. Hence, in this case, w does not start with `b` (because it starts with `a`), and it does not start with `abb` (because w' does not start with `b`).

For the second recursive case, let $w = w'w''$, with $w', w'' \in L(G)$. By the induction assumption, w' starts with neither `b` nor `abb` (similarly for w''). Let us do case analysis on the length of w'.

- $|w'| = 0$: If $w' = \epsilon$ then $w = w''$ which starts with neither `b` nor `abb`, by assumption.
- $|w'| = 1$: In this case we must have $w' =$ `a` since, by assumption, w' does not start with `b`. But then w doesn't start with `b`, and it doesn't start with `abb` either, because w'' does not start with `b`, by assumption.
- $|w'| \geq 2$: In this case w' must start with either `aa` or `ab`. That means w does not start with `b`. And w can only start with `abb` if $w' =$ `ab` and w'' starts with `b`. But the latter is impossible, by assumption.

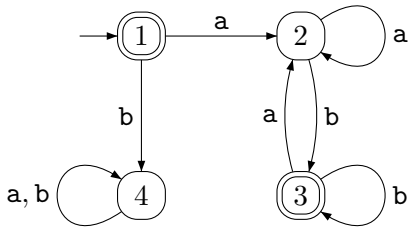
Hence in no case does w start with `abb`.

P8.12 Consider the context-free grammar $(\{S\}, \{\texttt{a}, \texttt{b}\}, R, S)$ with rules R:

$$S \;\rightarrow\; \texttt{a} \mid \texttt{b} \mid S\ S$$

Show that the grammar is ambiguous; then find an equivalent unambiguous grammar.

P8.13 Give a context-free grammar for the language recognised by this DFA:



Solution: Here is a context-free grammar that will do the job (S is the start symbol):

$$\begin{array}{lcl} S & \rightarrow & \epsilon \mid \texttt{a}\ A \\ A & \rightarrow & \texttt{a}\ A \mid \texttt{b}\ B \\ B & \rightarrow & \epsilon \mid \texttt{a}\ A \mid \texttt{b}\ B \end{array}$$

School of Computing and Information Systems
COMP30026 Models of Computation
Week 9: Pushdown Automata and Context-Free Languages

Exercises

T9.1 Construct pushdown automata (PDAs) for the following languages.

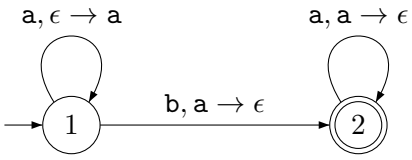
(i) $\{\mathtt{a}^i\mathtt{b}\mathtt{a}^j \mid i > j \geq 0\}$

(ii) $\{w \in \{\mathtt{a},\mathtt{b}\}^* \mid w \text{ is a palindrome}\}$

Continued in P9.1 & P9.2

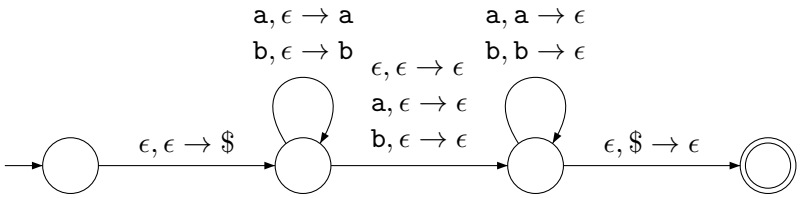
Solution:

(i)



Note that the stack might not be empty when this PDA halts; and that's okay.

(ii)



T9.2 A PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$ is *deterministic* if, from any configuration, it has at most one available move. That is,

$$\forall q \in Q\, \forall v \in \Sigma\, \forall a \in \Gamma \big(|\delta(q, v, a)| + |\delta(q, v, \epsilon)| + |\delta(q, \epsilon, a)| + |\delta(q, \epsilon, \epsilon)| \leq 1\big)$$

Which of your PDAs from T9.1 are deterministic?

Solution: In the solutions to T9.1, (i) is deterministic, (ii) is not deterministic. It is possible that your solution to (i) was not deterministic, but it is not possible that your PDA for (ii) was deterministic (unless it was incorrect).

T9.3 (a) Consider the language $A = \{\mathtt{a}^i\mathtt{b}^j\mathtt{a}^i\mathtt{b}^j \mid i, j \geq 0\}$. Use the pumping lemma for context-free languages to show that A is not context-free.

(b) Now consider $B = \{\mathtt{a}^i\mathtt{b}^j\mathtt{a}^j\mathtt{b}^i \mid i, j \geq 0\}$. Give a context-free grammar for B.

(c) A and B look very similar. We might try to prove that B is not context-free by doing what we did to prove that A is not context-free. Where does the attempted proof fail?

Solution:

(a) Assume that A is context-free and let p be the pumping length. Consider the string $\mathtt{a}^p\mathtt{b}^p\mathtt{a}^p\mathtt{b}^p \in A$. The pumping lemma tells us that the string can be written $uvxyz$, with v and y not both empty, and with $|vxy| \leq p$, such that $uv^ixy^iz \in A$ for all i. Clearly, if one (or both) of v and y contains an $\mathtt{a}$ as well as a $\mathtt{b}$ then pumping up will lead

to a string that is not in A, as the result will have more than two substrings $\mathtt{ab}$. So each of v and y must contain $\mathtt{a}$s only, or $\mathtt{b}$s only, unless it is empty. If neither v nor y contains a $\mathtt{b}$ then both must come from the same $\mathtt{a}^i$ segment (the first or the last), because $|vxy| \leq p$; and then, when we pump up, that segment alone grows, while the other $\mathtt{a}^i$ segment is untouched. Similarly, if neither v nor y contains an $\mathtt{a}$. So in these cases the result of pumping is not in A. The only remaining cases are when v is from the first $\mathtt{a}^i$ segment and y is from the first $\mathtt{b}^j$ segment, or v is from the first $\mathtt{b}^j$ segment and y is from the second $\mathtt{a}^i$ segment, or v is from the second $\mathtt{a}^i$ segment and y is from the second $\mathtt{b}^j$ segment. In each case, pumping up will take the string outside A. We conclude that A is not context-free.

(b) Here is a context-free grammar for B:

$$\begin{array}{lcl} S & \to & T \mid \mathtt{a}\ S\ \mathtt{b} \\ T & \to & \epsilon \mid \mathtt{b}\ T\ \mathtt{a} \end{array}$$

(c) If we pick the obvious candidate string $\mathtt{a}^p\mathtt{b}^p\mathtt{a}^p\mathtt{b}^p \in B$, we fail to get a contradiction with the pumping lemma. Namely we might have $v = \mathtt{b}^k$ and $y = \mathtt{a}^k$ $(0 < k \leq p/2)$ where v is a substring of the first $\mathtt{b}^j$ segment and y is a substring of the second $\mathtt{a}^j$ segment. In that case, the result of pumping (up or down) is in B, and we don't get a contradiction.

T9.4 Let A and B be arbitrary context-free languages. Show how context-free grammars for A and B can be manipulated to construct context-free grammars for $A \cup B$, $A \circ B$, and A^*.

Hint: Try to construct a new grammar using grammars for A and B that can derive strings from *A or B, etc.* **Continued in P9.3**

Solution: The class of context-free languages is closed under the regular operations: union, concatenation, and Kleene star.

Let G_1 and G_2 be context-free grammars generating L_1 and L_2, respectively. First, if necessary, rename variables in G_2 so that the two grammars have no variables in common. Let the start variables of G_1 and G_2 be S_1 and S_2, respectively. Then we get a context-free grammar for $L_1 \cup L_2$ by keeping the rules from G_1 and G_2, adding

$$\begin{array}{lcl} S & \to & S_1 \\ S & \to & S_2 \end{array}$$

where S is a fresh variable, and making S the new start variable.

We can do exactly the same sort of thing for $L_1 \circ L_2$. The only difference is that we now just add one rule:

$$S \to S_1\ S_2$$

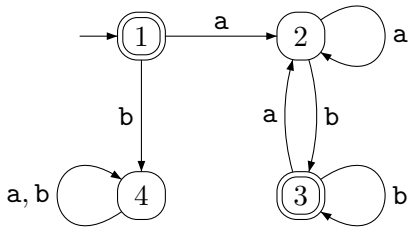
again making (the fresh) S the new start variable.

Let G be a context-free grammar for L and let S be fresh. If we add two rules to those from G:

$$\begin{array}{lcl} S & \to & \epsilon \\ S & \to & S\ S' \end{array}$$

where S' is G's start variable, then we have a context-free grammar for L^* (it has the fresh S as its start variable).

T9.5 Give a context-free grammar for the language recognised by this DFA:



Solution: Here is a context-free grammar that will do the job (S is the start symbol):

$$\begin{array}{lcl} S & \rightarrow & \epsilon \mid \mathtt{a}\ A \\ A & \rightarrow & \mathtt{a}\ A \mid \mathtt{b}\ B \\ B & \rightarrow & \epsilon \mid \mathtt{a}\ A \mid \mathtt{b}\ B \end{array}$$

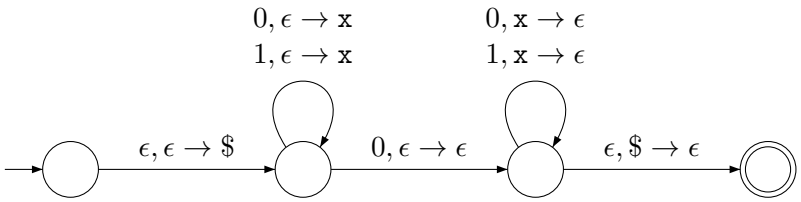
Homework problems

P9.1 Construct a pushdown automaton for

$$\{w \in \{\mathtt{0},\mathtt{1}\}^* \mid \text{the length of } w \text{ is odd and its middle symbol is } \mathtt{0}\}.$$

What's the minimum number of states you can achieve?

Solution:



P9.2 Take any context-free grammar in any problem set or tutorial, and construct a pushdown automata which recognises the language of that grammar. Try using the CFG to PDA conversion from the lectures, otherwise you can try to understand what the language is, and construct a PDA from scratch. Repeat this a few times for practice.

P9.3 We have seen that the set of context-free languages is not closed under intersection. However, it *is* closed under intersection with regular languages. That is, if L is context-free and R is regular then $L \cap R$ is context-free.

We can show this if we can show how to construct a push-down automaton P' for $L \cap R$ from a push-down automaton P for L and a DFA D for R. The idea is that we can do something similar to what we did in T8.3 when we built "product automata", that is, DFAs for languages $R_1 \cap R_2$ where R_1 and R_2 were regular languages. If P has state set Q_P and D has state set Q_D, then P' will have state set $Q_P \times Q_D$.

More precisely, let $P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, F_P)$ and let $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$. Recall the types of the transition functions:

$$\delta_P : (Q_P \times \Sigma_\epsilon \times \Gamma_\epsilon) \to \mathcal{P}(Q_P \times \Gamma_\epsilon)$$
$$\delta_D : (Q_D \times \Sigma) \to Q_D$$

We construct P' with the following components: $P' = (Q_P \times Q_D, \Sigma, \Gamma, \delta, (q_P, q_D), F_P \times F_D)$. Discuss how P' can be constructed from P and D. Then give a formal definition of δ, the transition function for P'.

Solution: For the case $v \neq \epsilon$ we define

$$\delta((q_p, q_d), v, x) = \{((r_p, r_d), y) \mid (r_p, y) \in \delta_P(q_p, v, x) \wedge r_d = \delta_D(q_d, v)\}$$

But we must also allow transitions that don't consume input, so:

$$\delta((q_p, q_d), \epsilon, x) = \{((r_p, q_d), y) \mid (r_p, y) \in \delta_P(q_p, \epsilon, x)\}$$

P9.4 Give a context-free grammar for $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \vee j = k \text{ where } i, j, k \geq 0\}$. Is your grammar ambiguous? Why or why not?

P9.5 Consider the context-free grammar $G = (\{S, A, B\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$ with rules R:

$$\begin{array}{lcl} S & \to & A\ B\ A \\ A & \to & \mathtt{a}\ A \mid \epsilon \\ B & \to & \mathtt{b}\ B \mid \epsilon \end{array}$$

(a) Show that G is ambiguous.

(b) The language generated by G is regular; give a regular expression for $L(G)$.

(c) Give an unambiguous context-free grammar, equivalent to G. *Hint:* As an intermediate step, you may want to build a DFA for $L(G)$.

Solution: We are looking at the context-free grammar G:

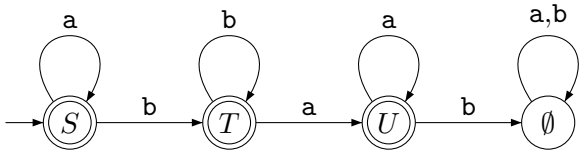
$$\begin{array}{lcl} S & \to & A\ B\ A \\ A & \to & \mathtt{a}\ A \mid \epsilon \\ B & \to & \mathtt{b}\ B \mid \epsilon \end{array}$$

(a) The grammar is ambiguous. For example, a has two different leftmost derivations:

$$S \Rightarrow A\ B\ A \Rightarrow B\ A \Rightarrow A \Rightarrow \mathtt{a}\ A \Rightarrow \mathtt{a}$$
$$S \Rightarrow A\ B\ A \Rightarrow \mathtt{a}\ A\ B\ A \Rightarrow \mathtt{a}\ B\ A \Rightarrow \mathtt{a}\ A \Rightarrow \mathtt{a}$$

(b) $L(G) = \mathtt{a}^*\mathtt{b}^*\mathtt{a}^*$.

(c) To find an unambiguous equivalent context-free grammar it helps to build a DFA for $\mathtt{a}^*\mathtt{b}^*\mathtt{a}^*$. (If this is too hard, we can always construct an NFA, which is easy, and then translate the NFA to a DFA using the subset construction method, which is also easy.) Below is the DFA we end up with. The states are named S, T, and U to suggest how they can be made to correspond to variables in a context-free grammar. The DFA translates easily to the grammar on the right. The resulting grammar is a so-called *regular* grammar, and it is easy to see that it is unambiguous—there is never a choice of rule to use.



$$\begin{array}{lcl} S & \to & \epsilon \mid \mathtt{a}\ S \mid \mathtt{b}\ T \\ T & \to & \epsilon \mid \mathtt{b}\ T \mid \mathtt{a}\ U \\ U & \to & \epsilon \mid \mathtt{a}\ U \end{array}$$

School of Computing and Information Systems
COMP30026 Models of Computation
Week 10: Turing Machines

Exercises

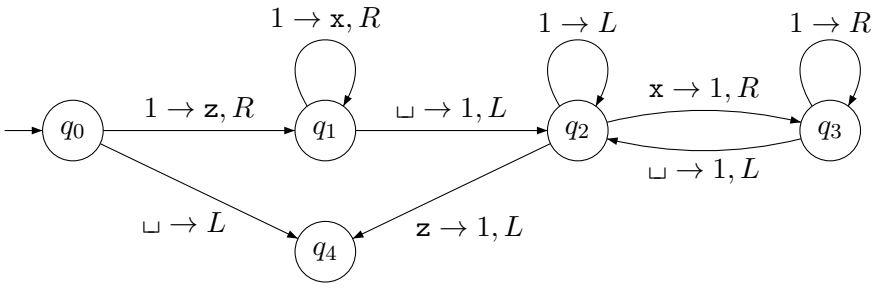
T10.1 The following Turing machine D was written to perform certain manipulations to its input—it isn't intended as a recogniser for a language, and so we don't bother to identify an accept or a reject state. The machine stops when no transition is possible, and whatever is on its tape at that point is considered output.

D's set of states is $\{q_0, q_1, q_2, q_3, q_4\}$, with q_0 being the initial state. The input alphabet is $\{1\}$ and the tape alphabet is $\{1, \mathtt{x}, \mathtt{z}, \sqcup\}$, where, as usual, $\sqcup$ stands for 'blank', or absence of a proper symbol. D's transition function δ is defined like so:

$$\begin{array}{lcllcllcl}
\delta(q_0, 1) & = & (q_1, \mathtt{z}, R) & \delta(q_1, \sqcup) & = & (q_2, 1, L) & \delta(q_2, \mathtt{z}) & = & (q_4, 1, L) \\
\delta(q_0, \sqcup) & = & (q_4, \sqcup, L) & \delta(q_2, 1) & = & (q_2, 1, L) & \delta(q_3, 1) & = & (q_3, 1, R) \\
\delta(q_1, 1) & = & (q_1, \mathtt{x}, R) & \delta(q_2, \mathtt{x}) & = & (q_3, 1, R) & \delta(q_3, \sqcup) & = & (q_2, 1, L)
\end{array}$$

Draw D's diagram and determine what D does to its input.

Solution: Here is the Turing machine D:



Note that, for the two transitions into q_4, the 'L' has no effect, since, at that point, the tape head is positioned over the leftmost tape cell. The machine doubles the number of ones that it finds on the tape.

T10.2 Consider the following language over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}, \#\}$,

$$A = \{w\#w \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}$$

Show that this language is decidable by following these steps:

(i) Describe, using pseudocode, the behaviour of a Turing machine that recognises A.

(ii) Draw a Turing machine that does exactly what you described in your pseudocode. You may omit the reject state, to which all missing transitions are assumed to go.

(iii) Argue that your Turing machine will halt on every input.

Continued in P10.6, P10.7

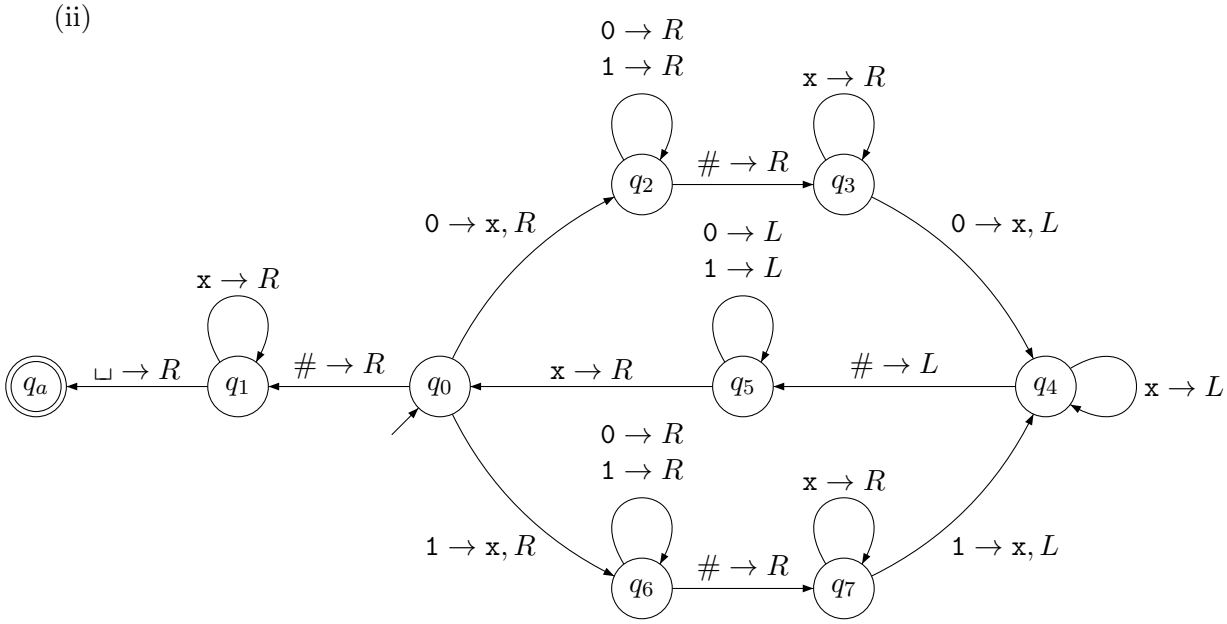
Solution:

(i) We will construct a loop which verifies the equality of the strings either side of the #, by reading a symbol from the left string, and verifying it appears in the correct position of the right string. Any failed verifications that occur while running this algorithm should be assumed to trigger rejection.

> On input w:
> (1) If there is a 0 below the tapehead, write x and go to (2);
> if there's a 1 below the tapehead, write x and go to (3);
> if there's a # below the tapehead, go to (5);
> (2) Move right over the section that matches $(0 \cup 1)^*\#x^*$,
> verify that 0 is the next symbol on the tape,
> replace it with x, then go to (4)
> (3) Move right over the section that matches $(0 \cup 1)^*\#x^*$,
> verify that 1 is the next symbol on the tape
> replace it with x, then go to (4)
> (4) Move left over the section that matches $(0 \cup 1)^*\#x^*$,
> and stop on the left-most 0 or 1 (otherwise the #), then go to (1)
> (5) After the # symbol, verify that the rest of the string is all x's
> followed by a blank. If so; accept

For brevity, we did not describe how the TM skips over the substring matching $(0 \cup 1)^*\#x^*$, but this is fairly straightforward to break down into a few states and transitions that describe "move right while there is a 0 or a 1, then move right when you see the #, then move right while there is an x". Likewise for moving left over this substring.

(ii)



(iii) We first argue that the two loops from q_0 to q_4 and back cannot be taken infinitely many times. These loops are only entered when there is a 0 or a 1 on the left of the first #. However, each time one of these loops are taken, the number of 0's and 1's on the left of the first # strictly decreases. Since this value is a strictly decreasing natural number, and the natural numbers are a well-founded structure, these loops can only be entered a finite number of times.

None of the self-loops which move the tapehead to the right, can run forever, since they will eventually hit a blank symbol, and none of them are labelled $\sqcup \to R$ or modify a blank at any point.

Lastly we should consider whether there are any infinite loops involving moving left when the tapehead is on the left-most cell of the tape. Could we take the $\mathtt{x} \to L$ transition on q_4 forever? No, since we can only reach q_4 if # is not the first symbol of the input.

T10.3 Here is how we can see that the class of decidable languages is closed under intersection. Let A and B be decidable languages, let M_A be a decider for A and M_B a decider for B. We construct a decider for $A \cap B$ as a Turing machine which implements this routine:

> On input w:
> (1) Run M_A on input w and reject if M_A rejects.
> (2) Run M_B on input w and reject if M_B rejects; else accept.

Show that the class of decidable languages is closed under union.

Continued in P10.1, P10.2 & P10.3

Solution: Let A and B be decidable languages, let M_A be a decider for A and M_B a decider for B. We construct a decider for $A \cup B$ as a Turing machine which implements this routine:

> On input w:
> (1) Run M_A on input w and accept if M_A accepts.
> (2) Run M_B on input w and accept if M_B accepts; else reject.

T10.4 The class of Turing recognisable languages is closed under intersection, and the construction we gave in the previous question, for decidable languages, can equally be used to prove this. (Convince yourself that the argument is still right, even though we now have no guarantee that M_A and M_B always terminate.)

The class of Turing recognisable languages is also closed under union, but we can't argue that the same way, by constructing a Turing machine which effectively runs M_A and M_B, one after the other. (Why not?)

Show that the class of Turing recognisable languages is closed under union.

Solution: Let M_A and M_B be recognisers for A and B, respectively. We want to construct a recogniser for $A \cup B$ but we can't use the construction from the last question. The reason is that there could be some string $w \in B$, which should obviously be accepted by the recogniser we construct, but for which M_A fails to terminate. That is, step (1) above never terminates.

Instead we construct a recogniser for $A \cup B$ by alternating the recognisers for A and B:

> On input w:
> (1) Let M_A take a single execution step; accept if M_A accepts;
> go to step (4) if M_A rejects.
> (2) Let M_B take a single execution step; accept if M_B accepts;
> go to step (5) if M_B rejects.
> (3) Go to step (1).
> (4) Run M_B alone; accept if M_B accepts; reject if it rejects.
> (5) Run M_A alone; accept if M_A accepts; reject if it rejects.

Note that this machine may fail to terminate on w, which will happen only if both of M_A and M_B fail to terminate (in which case $w \notin A \cup B$). That's okay, of course, as we are constructing a recogniser, not a decider.

Homework problems

P10.1 Show that the class of decidable languages is closed under complement. Why can't we use the same argument to show that the class of Turing recognisable languages is closed under complement?

Solution: Let M be a decider for A. We get a decider for A^c simply by swapping the 'reject' and 'accept' states in M.

The construction won't work if all we know about M is that it is a recogniser for A. Namely, M may fail to terminate for some string $w \in A^c$.

P10.2 Show that the class of decidable languages is closed under concatenation.

Solution: We just show the case for concatenation. Let M_A and M_B be deciders for A and B, respectively. We want to construct a decider for $A \circ B$. It will make our task easier if we utilise nondeterminism. We can construct a nondeterministic Turing machine to implement this routine:

> On input w:
> (1) Split w nondeterministically so that $w = xy$.
> (2) Run M_A on x; reject w if M_A rejects x.
> (2) Run M_B on y; reject w if M_B rejects y.
> (3) Otherwise accept w.

This makes good use of the nondeterministic Turing machine's bias towards acceptance.

P10.3 Show that the class of decidable languages is closed under Kleene star.

P10.4 A 2-PDA is a pushdown automaton that has two stacks instead of one. In each transition step it may consume an input symbol, pop and/or push to stack 1, and pop an/or push to stack 2. It can also leave out any of these options (using ϵ moves) just like the standard PDA.

In the lectures, we used the pumping lemma for context-free languages to established that the language $B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbb{N}\}$ is not context-free. However, B has a 2-PDA that recognises it. Outline in English or pseudo-code how that 2-PDA operates.

Solution: Here is how the 2-PDA recogniser for B operates:

(a) Push a \$ symbol onto stack 1 and also onto stack 2.
(b) As long as we find an `a` in input, consume it and push an `a` onto stack 1.
(c) As long as we find a `b` in input, consume it and push a `b` onto stack 2.
(d) As long as we find a `c` in input, consume it and pop both stacks.
(e) If the top of each stack has a \$ symbol, pop these.
(f) If we got to this point and the input has been exhausted, accept.

If the 2-PDA got stuck at any point, that meant reject.

P10.5 In fact, a 2-PDA is as powerful as a Turing machine. Outline an argument for this proposition by showing how a 2-PDA can simulate a given Turing machine. *Hint:* arrange things so that, at any point during simulation, the two stacks together hold the contents of the Turing machine's tape, and the symbol under the tape head sits on top of one of the stacks.

Solution: To simulate M running on input $x_1x_2\cdots x_n$, the 2-PDA P first pushes a \$ symbol onto stack 1 and also onto stack 2. It then runs through its input, pushing $x_1, x_2, \ldots x_{n-1}, x_n$ onto stack 1. It then pops each symbol from stack 1, pushing it to stack 2. That is, it pushes $x_n, x_{n-1}, \ldots x_2, x_1$ onto stack 2, in that order. Note that x_1 is on top.

P is now ready to simulate M. Note that it has consumed all of its input already, but it is not yet in a position to accept or reject.

For each state of M, P has a corresponding state. Assume P is in the state that corresponds to some M state q.

For each M-transition $\delta(q, a) = (r, b, R)$, P has a transition that pops a off stack 2 and pushes b onto stack 1. If stack 2 now has \$ on top, P pushes a blank symbol onto stack 2. Then P goes to the state corresponding to r.

For each M-transition $\delta(q, a) = (r, b, L)$, P has a transition that first pops a off stack 2, replacing it by b. It then pops the top element off stack 1 and transfers it to the top of stack 2, unless it happens to be \$. And then of course P goes to the state corresponding to r.

If this seems mysterious, try it out for a simple Turing machine and draw some diagrams along to way, with snapshots of the Turing machine's tape and tape head next to the 2-PDA's corresponding pair of stacks. The invariant is that what sits on top of the 2-PDA's stack 2 is exactly what is under the Turing machine's tape head at the corresponding point in its computation.

P10.6 For each of the following languages, write an algorithm in pseudocode which describes a Turing machine that decides the following languages. Assume that $\Sigma = \{\texttt{0}, \texttt{1}\}$

(i) $\{w \mid w$ has an equal number of `0`'s and `1`'s$\}$

(ii) $\{w \mid w$ has twice as many `0`'s as `1`'s$\}$

(iii) $\{w \mid w$ does not have twice as many `0`'s as `1`'s$\}$

P10.7 For each language in P10.6, draw a Turing machine that carries out the pseudocode. Then write down the sequence of configurations your TM goes through on an interesting input.

P10.8 Show that the problem of whether the language of a DFA is empty, is decidable. That is, show that the language

$$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the graph of the DFA, and argue that your algorithm will not run forever on any input DFA $\langle D \rangle$.

P10.9 Show that the problem of whether the language of a CFG is empty, is decidable. That is, show that the language

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the rules of the CFG, and argue that your algorithm will not run forever on any input CFG $\langle G \rangle$.

School of Computing and Information Systems
COMP30026 Models of Computation
Week 11: Turing-Decidable and Turing-Recognisable Languages

Exercises

T11.1 Here is how we can see that the class of decidable languages is closed under intersection. Let A and B be decidable languages, let M_A be a decider for A and M_B a decider for B. We construct a decider for $A \cap B$ as a Turing machine which implements this routine:

> On input w:
> (1) Run M_A on input w and reject if M_A rejects.
> (2) Run M_B on input w and reject if M_B rejects; else accept.

Show that the class of decidable languages is closed under union.

Continued in P11.1, P11.2 & P11.3

Solution: Let A and B be decidable languages, let M_A be a decider for A and M_B a decider for B. We construct a decider for $A \cup B$ as a Turing machine which implements this routine:

> On input w:
> (1) Run M_A on input w and accept if M_A accepts.
> (2) Run M_B on input w and accept if M_B accepts; else reject.

T11.2 The class of Turing-recognisable languages is closed under intersection, and the construction we gave in the previous question, for decidable languages, can equally be used to prove this. (Convince yourself that the argument is still right, even though we now have no guarantee that M_A and M_B always terminate.)

The class of Turing-recognisable languages is also closed under union, but we can't argue that the same way, by constructing a Turing machine which effectively runs M_A and M_B, one after the other. (Why not?)

Show that the class of Turing-recognisable languages is closed under union.

Solution: Let M_A and M_B be recognisers for A and B, respectively. We want to construct a recogniser for $A \cup B$ but we can't use the construction from the last question. The reason is that there could be some string $w \in B$, which should obviously be accepted by the recogniser we construct, but for which M_A fails to terminate. That is, step (1) above never terminates.

Instead we construct a recogniser for $A \cup B$ by alternating the recognisers for A and B:

> On input w:
> (1) Let M_A take a single execution step; accept if M_A accepts;
> go to step (4) if M_A rejects.
> (2) Let M_B take a single execution step; accept if M_B accepts;
> go to step (5) if M_B rejects.
> (3) Go to step (1).
> (4) Run M_B alone; accept if M_B accepts; reject if it rejects.
> (5) Run M_A alone; accept if M_A accepts; reject if it rejects.

Note that this machine may fail to terminate on w, which will happen only if both of M_A and

M_B fail to terminate (in which case $w \notin A \cup B$). That's okay, of course, as we are constructing a recogniser, not a decider.

T11.3 Show that the language $ALL_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$ is decidable.

Solution: (Sketch) Checking whether $L(A) = \Sigma^*$ is equivalent to checking whether $L(A)^c = \emptyset$. Thus, we can compute the DFA B that recognizes $L(A)^c$ and run the DFA Emptiness TM on B. If the DFA Emptiness TM accepts B, we reject, and vice versa.

T11.4 Let M be a Turing machine with alphabet Σ and $w \in \Sigma^*$ a string. Define a language A containing only the single string s, where

$$s = \begin{cases} \texttt{1} & \text{if } M \text{ halts on input } w \\ \texttt{0} & \text{if } M \text{ does not halt on input } w \end{cases}$$

Is A decidable? Why or why not?

Solution: A is decidable. A is either the language $\{\texttt{0}\}$ or the language $\{\texttt{1}\}$, and both of these languages are finite, hence decidable (there are very simple Turing machines for both of these languages that are deciders). The fact that we might not know whether $A = \{\texttt{0}\}$ or $A = \{\texttt{1}\}$ for a given Turing machine M and string w, does not prevent it from being decidable. Remember a language is decidable iff *there exists* a Turing machine D which decides it.

The difference between this language and $Halt_{TM}$, is that a decider for $Halt_{TM}$ must determine whether any given Turing machine halts on an input, whereas a decider for A only has to do this for one particular Turing machine and input string. In fact a decider for A doesn't have to perform any computation/reasoning that "figures out" whether M halts on w, it just has to give the right answer.

Homework problems

P11.1 Show that the class of decidable languages is closed under complement. Why can't we use the same argument to show that the class of Turing recognisable languages is closed under complement?

Solution: Let M be a decider for A. We get a decider for A^c simply by swapping the 'reject' and 'accept' states in M.

The construction won't work if all we know about M is that it is a recogniser for A. Namely, M may fail to terminate for some string $w \in A^c$.

P11.2 Show that the class of decidable languages is closed under concatenation.

Solution: We just show the case for concatenation. Let M_A and M_B be deciders for A and B, respectively. We want to construct a decider for $A \circ B$. It will make our task easier if we utilise nondeterminism. We can construct a nondeterministic Turing machine to implement this routine:

> On input w:
> (1) Split w nondeterministically so that $w = xy$.
> (2) Run M_A on x; reject w if M_A rejects x.
> (2) Run M_B on y; reject w if M_B rejects y.
> (3) Otherwise accept w.

This makes good use of the nondeterministic Turing machine's bias towards acceptance.

P11.3 Show that the class of decidable languages is closed under Kleene star.

P11.4 Show that the problem of whether the language of a DFA is empty, is decidable. That is, show that the language

$$E_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA and } L(D) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the graph of the DFA, and argue that your algorithm will not run forever on any input DFA $\langle D \rangle$.

P11.5 Show that the problem of whether the language of a CFG is empty, is decidable. That is, show that the language

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

is decidable. *Hint:* write pseudocode for an algorithm which analyses the rules of the CFG, and argue that your algorithm will not run forever on any input CFG $\langle G \rangle$.

P11.6 Consider the alphabet $\Sigma = \{0, 1\}$. The set Σ^* consists of all the *finite* bit strings, and the set, while infinite, turns out to be countable. (At first this may seem obvious, since we can use the function $binary : \mathbb{N} \to \Sigma^*$ defined by

$$binary(n) = \text{ the binary representation of } n$$

as enumerator; however, that is not a surjective function, because the legitimate use of leading zeros means there is no unique binary representation of n. For example, both 101 and 00101 denote 5. Instead the idea is to list all binary strings of length 0, then those of length 1, then those of length 2, and so on: $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots$.)

Now consider instead the set $\mathcal{B}$ of *infinite* bit strings. Show that $\mathcal{B}$ is much larger than Σ^*. More specifically, use diagonalisation to show that $\mathcal{B}$ is not countable.

Solution: Assume $\mathcal{B}$ is countable. Then we can enumerate $\mathcal{B}$:

Element								
b_1	$\boxed{0}$	0	1	0	1	1	1	...
b_2	1	$\boxed{0}$	1	1	1	0	1	...
b_3	1	0	$\boxed{1}$	1	1	0	1	...
b_4	0	1	0	$\boxed{0}$	1	0	0	...
$\vdots$								

However, the infinite sequence which has

$$i\text{'th bit} = \begin{cases} 0 & \text{if the } i\text{th bit of } b_i \text{ is 1} \\ 1 & \text{if the } i\text{th bit of } b_i \text{ is 0} \end{cases}$$

is different from each of the b_i. Hence no enumeration can exist, and $\mathcal{B}$ is uncountable. This should not be surprising, because the set $\mathcal{B}$ is really the same as (or is isomorphic to) $\mathbb{N} \to \Sigma$.

P11.7 (a) Let A be a DFA with n states. Show that $L(A) = \emptyset$ iff A does not accept any string of length at most $n - 1$.

(b) Argue that the following "brute force" algorithm decides E_{DFA}, and that E_{DFA} is thus decidable:

> On input DFA A with n states:
> (1) For each string w of length at most $n - 1$:
> (2) Run A on input w and reject if A accepts.
> (3) Otherwise, accept.

P11.8 Let G be a context-free grammar in Chomsky normal form and let $w \in L(G)$. Show that every derivation of w in G has $2|w| - 1$ steps. (Or equivalently, that every parse tree of w in G has $2|w| - 1$ internal nodes.)

School of Computing and Information Systems
COMP30026 Models of Computation Tutorial Week 12

Content: reducibility, mapping reducibility

For tutors: Expect students to get tripped up by the subtleties in the definition of reducibility, hence tute question 1 is simply getting them to restate stuff we've done before to ensure they get the definition. The primary objective of this week's tutorial is getting them to understand the definition so even if you don't move past q1 this tute, that's ok. The secondary objective is to show them how to construct reductions.

Common things for students to be tripped up by:

- Reducing the wrong way. In particular, when trying to show A reduces to B, they may incorrect start with an instance of B and map it to an instance of A
- Concluding undecidability/unrecognizability incorrectly. For example, they may incorrectly conclude from A reduces to B and B is undecidable, then A is undecidable. This is part of the objective of T12.3.

Exercises

T12.1 Show that each the following reduciblity relations hold by giving a computable mapping reduction:

(a) $A_{NFA} \leq_m A_{DFA}$

(b) $A_{DFA} \leq_m A_{NFA}$

(c) $A_{REX} \leq_m A_{DFA}$

(d) $EQ_{DFA} \leq_m E_{DFA}$

Remember to justify why your mapping reduction is correct and why your mapping reduction is computable. You can do the latter by referencing a procedure from slides/tutes.

Solution:

(a) $f(\langle B, w\rangle) = \langle B', w\rangle$ where B' is the DFA equivalent of B. We have that B' is computable due to the subset construction method.

(b) $f(\langle B, w\rangle) = \langle B, w\rangle$. This works because a DFA is also an NFA.

(c) $f(\langle P, w\rangle) = \langle G, w\rangle$ where G is the CFG equivalent of P. We can compute G from P since we can always convert a pushdown automata into an equivalent CFG.

(d) $f(\langle A, B\rangle) = \langle C\rangle$ where C is the DFA with $L(C) = (L(A) \cap L(B)^c) \cup (L(A)^c \cap L(B))$. First, C can be computed from A and B due regular languages being closed under union, intersection and complement. Second, the set $(L(A) \cap L(B)^c) \cup (L(A)^c \cap L(B)$ is empty if and only if $L(A) = L(B)$. Thus, f is a computable mapping reduction.

T12.2 Consider the languages $ALL_{CFG} = \{\langle G\rangle \mid G \text{ is a grammar and } L(G) = \Sigma^*\}$ and $EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are grammars and } L(G) = L(H)\}$.

(a) Show that $ALL_{CFG} \leq_m EQ_{CFG}$.

(b) It is a fact that ALL_{CFG} is undecidable. What can we conclude from this fact and the above?

Solution:

(a) Consider the mapping reduction f where $f(\langle G \rangle) = \langle G, G_{all} \rangle$ where G_{all} is the grammar that generates Σ^*. Since Σ^* is regular and hence context-free, G_{all} can be computed. The mapping reduction is correct since $L(G) = L(G_{all})$ if and only if $L(G) = \Sigma^*$.

(b) Since ALL_{CFG} is undecidable and $ALL_{CFG} \leq_m EQ_{CFG}$, we conclude that EQ_{CFG} is also undecidable.

T12.3 Show that $E_{TM} \leq_m EQ_{TM}$. Using only the facts that $E_{TM} \leq_m EQ_{TM}$ and that E_{TM} is undecidable, can we conclude that EQ_{TM} is also undecidable?

Solution: Consider the mapping reduction f where $f(\langle M \rangle) = \langle M, M_{reject} \rangle$ where M_{reject} is the TM that rejects all strings, i.e. $L(M_{reject}) = \emptyset$. It is easy to construct M_{reject}: there is only one state which is both the initial state and also the reject state. The mapping reduction is correct since $L(M) = L(M_{reject})$ if and only if $L(M) = \emptyset$.

Yes, the conclusion follows. $E_{TM} \leq_m EQ_{TM}$ implies that if E_{TM} is undecidable, then EQ_{TM} is undecidable.

T12.4 Consider the problem of whether a given context-free grammar with alphabet $\{\mathtt{0}, \mathtt{1}\}$ is able to generate a string in $L(\mathtt{1}^*)$. Is that decidable? In other words, is the language

$$\{\langle G \rangle \mid G \text{ is a context-free grammar over } \{\mathtt{0}, \mathtt{1}\} \text{ and } L(G) \cap L(\mathtt{1}^*) \neq \emptyset\}$$

decidable? Show that it is decidable, using a decider E_{CFG} for the decidable language

$$\{\langle G \rangle \mid G \text{ is a context-free grammar over } \{\mathtt{0}, \mathtt{1}\} \text{ and } L(G) = \emptyset\}$$

Hint: consider known closure properties for context-free languages.

Solution: The key insight for this exercise is the following closure theorem:

Theorem 1. If A is a context-free language and B is a regular language, then $A \cap B$ is context-free.

So since $L(G)$ is context-free and $L(\mathtt{1}^*)$ is regular, $L(G) \cap L(\mathtt{1}^*)$ is context-free. Therefore, by the definition of a context-free language, $L(G) \cap L(\mathtt{1}^*)$ has a context-free grammar, G', that generates it. So we could run the decider, E_{CFG} on input $\langle G' \rangle$ to check if $L(G') = \emptyset$, accept if it rejects, and reject if it accepts.

Is that a sufficient description of a decidable algorithm we could implement on a Turing machine? Think about this before proceeding to the next page.

The problem with this description is that it does not explain how one constructs the grammar G' from the grammar G. We just know that a suitable G' *exists*, by Theorem 1.

What saves us is that the proof of Theorem 1 is *constructive*, meaning it describes how to construct a "witness PDA" to the claim that $A \cap B$ is context-free, by using a PDA for A and a DFA for B. Not all proofs are constructive. There are many theorems of mathematics which are not constructive. A simple example is the Intermediate Value Theorem.

Intermediate Value Theorem (I.V.T.) If $f(x)$ is continuous on the interval $[a, b]$ and N is a value between $f(a)$ and $f(b)$, then there exists a number $c \in (a, b)$ such that $f(c) = N$.

There is no way to prove this theorem constructively, that is, we cannot actually determine a correct value for c by any finite method. In contrast, many proofs in computer science are constructive, such as Theorem 1, and involve constructions we can reuse.

If we inspect the proof of Theorem 1, from T12.4, we find that it uses a PDA for A and a DFA for B to construct a PDA P' for $A \cap B$. In the decider for this question, we do not get a PDA as input, we have a context-free grammar. But we have a theorem that says that a language is context-free iff there is a pushdown automata that recognises it, and it's constructive! So the first step of our algorithm will be to convert the CFG G into a PDA P which recognises the same language.

Similarly, we need a DFA to perform the construction, but we only have the regular expression $\mathtt{1}^*$. But it's easy enough to construct a DFA for $L(\mathtt{1}^*)$, which we'll call D. In fact, this step does not need to be a part of the algorithm that takes $\langle G \rangle$ as input, because we already know what DFA it should be (unlike P).

We will be left with a PDA P' for $L(P) \cap L(D)$, but we need a grammar to run E_{CFG}. For this, we use the other direction of the "context-free iff has PDA" proof, to obtain a CFG, G', for $L(P')$. We summarise these steps in the following algorithm:

On input w:
(1) If w is not of the form $\langle G \rangle$; reject.
(2) Perform the CFG to PDA algorithm on G to get a PDA P for $L(G)$.
(3) Use the PDA-DFA product construction on P and D
to get a new PDA P' for $L(P) \cap L(D)$.
(4) Perform the PDA to CFG algorithm on P' to get a CFG G' for $L(P')$.
(5) Run E_{CFG} on input $\langle G' \rangle$; reject if it accepts; accept if it rejects.

T12.5 **Now continue with any other practice problems you may have missed.**

Practice Problems

P12.1 Show that $A_{DFA} \leq_m A_{CFG} \leq_m A_{TM}$.

P12.2 Earlier, we gave a mapping reduction from an undecidable problem ALL_{CFG} to EQ_{CFG} and concluded that EQ_{CFG} is undecidable. Show that the language

$$\overline{EQ_{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are grammars and } L(G) \neq L(H)\}$$

is Turing-recognisable. Conclude that EQ_{CFG} is in fact not Turing-recognisable.

P12.3 Let $T = \{\langle M\rangle \mid M \text{ is a TM that accepts } w^R \text{ whenever it accepts } w\}$. Show that T is undecidable.

P12.4 Show that if A is Turing-recognisable and $A \leq_m \overline{A}$, then A is decidable.