Monash University
Faculty of Information Technology
Semester 2, 2023

FIT2014
Exercises 1
Languages and Logic

ASSESSED PREPARATION: Question 1.
You must provide a serious attempt at all parts of this question at the start of your tutorial.

1. A short footpath is divided into four numbered squares. You start at square 1 (at time 0). Time is a series of discrete steps. At each timestep, you either step one square forwards or one square backwards, never standing still. You must not move backwards from square 1. For each square $s = 1, 2, 3, 4$ and each time $t = 0, 1, \ldots$, let $v_{t,s}$ be the proposition with the meaning:

At time t you are at square s.

Write logical expressions for the following propositions.

(a) At time 0, you are at square 1.
(b) At time 0, you are not at square 2.
(c) At time 0, you are at square 1 and nowhere else.
(d) At time 5, you are in at least one square.
(e) At time 5, you are not in both square 1 and square 3.
(f) At time 5, you are not in two squares simultaneously.
(g) At time 5, you are in exactly one square.
(h) If you are at square 2 at time 3, then at time 4 you have moved one step forwards or backwards.
(i) How many clauses would be needed in a CNF expression to completely describe your rules of movement for $t = 0, 1, 2, 3$?

2. Let ODD-ODD be the language of strings, over the alphabet {a,b}, that contain an odd number of a's and an odd number of b's. Let $\overline{\text{ODD-ODD}}$ be its complement.
Prove that PALINDROMES $\subseteq \overline{\text{ODD-ODD}}$.

3. Distributive Law for propositional logic:
(a) Prove that

$$P \vee (Q \wedge R) \quad \text{is logically equivalent to} \quad (P \vee Q) \wedge (P \vee R).$$

(b) Prove that

$$P \wedge (Q \vee R) \quad \text{is logically equivalent to} \quad (P \wedge Q) \vee (P \wedge R),$$

using part (a) and a rule named after a friend of Charles Babbage.

4. Prove that

$$(P_1 \wedge \cdots \wedge P_n) \Rightarrow C \quad \text{is logically equivalent to} \quad \neg P_1 \vee \cdots \vee \neg P_n \vee C$$

5. A meeting about moon mission software is held at NASA in 1969. Participants may include Judith Cohen (electrical engineer), Margaret Hamilton (computer scientist), and Katherine Johnson (mathematician). Let Judith, Margaret and Katherine be propositions with the following meanings.

Judith	Judith Cohen is in the meeting.
Margaret	Margaret Hamilton is in the meeting.
Katherine	Katherine Johnson is in the meeting.

For each of the following statements, write a proposition in Conjunctive Normal Form with the same meaning.

(a) Judith and Margaret are not both in the meeting.
(b) Either Judith or Margaret, but not both of them, is in the meeting. (This is the "exclusive-OR".)
(c) At least one of Judith, Margaret and Katherine is in the meeting.
(d) At most one of Judith, Margaret and Katherine is in the meeting.
(e) Exactly one of Judith, Margaret and Katherine is in the meeting.
(f) At least two of Judith, Margaret and Katherine are in the meeting.
(g) At most two of Judith, Margaret and Katherine are in the meeting.
(h) Exactly two of Judith, Margaret and Katherine are in the meeting.
(i) Exactly three of Judith, Margaret and Katherine are in the meeting.
(j) None of Judith, Margaret and Katherine is in the meeting.

6. Suppose we have propositions Tree, Leaf, Bipartite, Internal about a connected graph with at least two vertices, with the following meanings.[1]

Tree	The graph is a tree.
Bipartite	The graph is bipartite.
Leaf	The graph has a leaf.
Internal	The graph has a vertex of degree ≥ 2. (Such a vertex is sometimes called an *internal vertex*.)

Using these propositions, write a proposition in CNF with the following meaning:

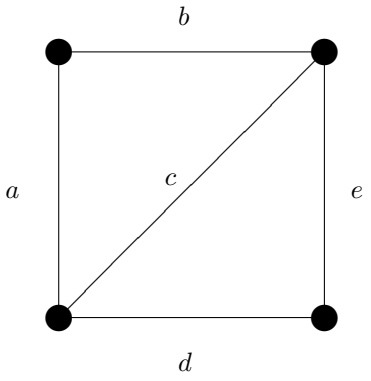
> If the graph is a tree, then it's bipartite and has a leaf, but if it's not a tree, then it has a vertex of degree ≥ 2.

7. *(mostly from FIT2014 Final Exam, 2015)*

A **perfect matching** in a graph G is a subset X of the edge set of G that meets each vertex exactly once. In other words, no two edges in X share a vertex, and each vertex of G is incident with exactly one edge in X.

For example, in the following graph, the edge set $\{a, e\}$ is a perfect matching. But $\{a, b, e\}$ is not a perfect matching (since, for example, a and b share a vertex), and $\{a\}$ is not a perfect matching (since some vertices are not incident with the edge in this set).

[1] Thanks to FIT2014 tutor Ben Jones for detecting and reporting a small error in an earlier version of this question.



Let W be the above graph.
Construct a Boolean expression E_W in Conjunctive Normal Form such that the satisfying truth assignments for E_W correspond to perfect matchings in the above graph W.

When doing this, use Boolean variables a, b, c, d, e which are each True if and only if the edge with the same name belongs to the perfect matching.

8. Using the function symbol **father**, the predicate **taller**, and the constant symbols **claire** and **max**, convert the following sentences to Predicate Logic. Assume that **taller(X,Y)** means **X** is taller than **Y** and the universe of discourse is "all people".

i. Max's father is taller than Max but not taller than Claire's father.

ii. Someone is taller than Claire's father.

iii. Everyone is taller than someone else.

iv. Everyone who is taller than Claire is taller than Max.

9. Suppose you have access to the following functions and relations:

- $|x|$ = the length of the string x
- equality and inequality relations: $=, <, \leq, >, \geq$
- set membership, denoted by $\in$ as usual.

Let L be a language.

(a) Using quantifiers, write down a logical statement about L that is True if and only if L is finite.

(b) Now write a statement about L that is True if and only if L is infinite.

Supplementary exercises

10. *Three boys, Adam, Brian and Claude, are caught, suspected of breaking a glass window. When the boys were questioned by police:*

Adam said: 'Brian did it; Claude is innocent'.

Brian said: 'If Adam is guilty then so is Claude'.

Claude said: 'I didn't do it; one of the others did'.

The police believed that all the boys were telling the truth, and therefore concluded that Brian broke the window and the others didn't.

Using the following propositions express the statements of the boys and the police conclusion in propositional logic.

A: **Adam broke the window.**

B: **Brian broke the window.**

C: **Claude broke the window.**

Assuming that all the boys were telling the truth, was the police conclusion logically valid?

11. Recall the logical expression given near the end of Lecture 2 for your dinner party guest list:

$$\begin{aligned}&(\text{Harry} \vee \text{Ron} \vee \text{Hermione} \vee \text{Ginny})\\ &\wedge\ (\neg\text{Hagrid} \vee \text{Norberta})\\ &\wedge\ (\neg\text{Fred} \vee \text{George}) \ \wedge\ (\text{Fred} \vee \neg\text{George})\\ &\wedge\ (\neg\text{Voldemort} \vee \neg\text{Bellatrix}) \ \wedge\ (\neg\text{Voldemort} \vee \neg\text{Dolores}) \ \wedge\ (\neg\text{Bellatrix} \vee \neg\text{Dolores})\end{aligned}$$

How long would an equivalent DNF expression be? Specifically, how many disjuncts — smaller expressions combined using $\vee$ to make the whole expression — would it have?

12. For each past or present Monash unit, we'll use its unit code to denote the proposition that you have passed the unit. So, if ABC1234 is a unit code, then we'll also use ABC1234 for a proposition with the following meaning:

$$\text{ABC1234} \ = \ \begin{cases} \text{True}, & \text{if you have passed unit ABC1234;} \\ \text{False}, & \text{if you have not passed unit ABC1234} \\ & \text{(either through never having enrolled in it, or only failing it,} \\ & \text{or doing it currently so that you haven't finished it yet).} \end{cases}$$

Here is an edited extract from the Monash Handbook 2022, specifying the conditions under which you may enrol in FIT2014:

Prerequisite:

- One of FIT1045, FIT1048, FIT1051, FIT1053, ENG1003, ENG1013 or (FIT1040 and FIT1029)

AND

- One of MAT1830, MTH1030, MTH1035, ENG1005

Prohibition:

- CSE2303

(a) Using these rules and the propositions corresponding to all these unit codes, construct an expression in Conjunctive Normal Form that specifies the conditions under which you may enrol in FIT2014.

Now consider how you would construct an equivalent expression in Disjunctive Normal Form:

$$\underbrace{(\ldots \wedge \ldots \wedge \ldots\ldots)}_{\text{disjunct}} \vee \underbrace{(\ldots \wedge \ldots \wedge \ldots\ldots)}_{\text{disjunct}} \vee \cdots\cdots\cdots \vee \underbrace{(\ldots \wedge \ldots \wedge \ldots\ldots)}_{\text{disjunct}}$$

(b) Give *three* of the disjuncts in such an expression.

(c) How many disjuncts would such an expression have?

13. Leonard Books, Cedric Smith, Arthur Stone and Bill Tutte[2] used the pseudonym Blanche Descartes for some of their writings. Each work by Blanche Descartes was written by some nonempty subset of the four. Let Leonard, Cedric, Arthur and Bill be propositions about one of Blanche Descartes's works with the following meanings.

Leonard	Leonard Books was one of the authors.
Cedric	Cedric Smith was one of the authors.
Arthur	Arthur Stone was one of the authors.
Bill	Bill Tutte was one of the authors.

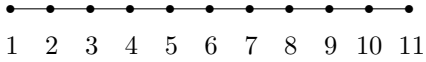
Write a proposition in Conjunctive Normal Form meaning that exactly two of the four were authors of the work.

14. One-dimensional Go.

This question uses some concepts from the ancient east Asian board game known as *Go* in the West, *Wéiqí* in China, *Go* or *Igo* in Japan, and *Baduk* in Korea. This game is over 2,000 years old, and is generally regarded as harder than Chess. Indeed, until very recently, computer programs for Go could not defeat human professionals (in contrast to Chess, where the best computer players have been stronger than human world champions since the late 1990s).

The situation changed dramatically early in 2016, with stunning performances by the program AlphaGo, created by Google DeepMind. (This company began as a start-up in London in 2010 and was acquired by Google in 2014.) In October 2015, it defeated the European champion Fan Hui in every game of a five-game match: `http://www.nature.com/news/google-ai-algorithm-masters-ancient-game-of-go-1.19234`. Then in March 2016 it defeated Lee Sedol of Korea, generally regarded as the best player in the world in recent times: `https://www.theatlantic.com/technology/archive/2016/03/the-invisible-opponent/475611/`. The final score in that five-game match was 4-1 in favour of AlphaGo. In May 2017 it defeated the top-ranked player in the world, Ke Jie of China, 3-0. See `https://deepmind.com/research/alphago/` or `http://361points.com/articles/thoughtsonalphago/`.

Go is played on a graph, usually a square lattice (grid) of 19×19 vertices. But we will use much simpler graphs in this question, namely path graphs with n vertices and $n-1$ edges, where $n \geq 1$. For example, with $n = 11$ we get the following path graph with 11 vertices and 10 edges.



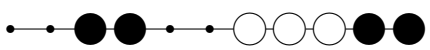
A *position* consists of a placement of black and white stones on some of the vertices of the graph. Each vertex may have a black stone, or a white stone (but not both), or be uncoloured (i.e., vacant). A position is *legal* if every vertex with a stone can be linked to an uncoloured vertex by a path consisting entirely of vertices with stones of that same colour (except for the uncoloured vertex at the end of that path).

For example, the following position is legal, since each of its three "chains" of consecutive vertices of the same colour either starts or ends with an uncoloured vertex.

[2]While they were still undergraduate students at Cambridge, these four wrote a paper which solved a famous open problem of recreational mathematics, helped create modern graph theory and influenced the course of history: R. L. Brooks, C. A. B. Smith, A. H. Stone, and W. T. Tutte, The dissection of rectangles into squares, *Duke Mathematical Journal* **7** (1940) 312–340.

But the following position is illegal, since it has a chain of white vertices with black vertices at each end. (The position has four chains altogether, and three are ok. But it only takes one without an uncoloured neighbour to make the position illegal.)

We number the vertices on the path graph from 1 to n, from left to right. We say that a position on this path graph is *almost legal* if vertex n is coloured (i.e., has a stone) and its chain is not next to an uncoloured vertex, but every other chain is next to an uncoloured vertex. In other words, it is illegal, but the only chain making it illegal is the chain containing vertex n; all other chains are ok. The two positions given above are *not* almost legal: the first is legal (so it isn't *almost* legal), while the second is illegal but the illegality is not due to the last vertex (which in this case is uncoloured). The following position is almost legal. All its chains are ok except the last one on the right.



Let $V_{B,n}$, $V_{W,n}$, $V_{U,n}$, $L_{B,n}$, $L_{W,n}$, $L_{U,n}$, $A_{B,n}$, $A_{W,n}$ be the following propositions about a position on the n-vertex path graph.

$V_{B,n}$	Vertex n is Black.
$V_{W,n}$	Vertex n is White.
$V_{U,n}$	Vertex n is Uncoloured.
$L_{B,n}$	The position is legal, and vertex n is Black.
$L_{W,n}$	The position is legal, and vertex n is White.
$L_{U,n}$	The position is legal, and vertex n is Uncoloured.
$A_{B,n}$	The position is almost legal, and vertex n is Black.
$A_{W,n}$	The position is almost legal, and vertex n is White.

(a) Use the propositions $L_{B,n}$, $L_{W,n}$, $L_{U,n}$ (together with appropriate connectives) to write a logical expression for the proposition that the position is legal.

Now consider how legality and almost-legality on the n-vertex path graph are affected by extending the path to vertex $n+1$.

(b) If $L_{B,n}$ is true, what possible states (Black/White/Uncoloured) can vertex $n+1$ be in, if we want the position to be legal on the $n+1$-vertex path as well?
Do the same for $L_{W,n}$ and $L_{U,n}$.

(c) If $A_{B,n}$ is true, what possible states can vertex $n+1$ be in, if we want the position to be legal on the $n+1$-vertex path?
Do the same for $A_{W,n}$.
Why is there no line for $A_{U,n}$ in the table?

(d) Construct a logical expression for $L_{B,n+1}$ using some of the propositions $V_{\_,n+1}, L_{\_,n}, A_{\_,n}$ in the above table. (In other words, you can only use the L-propositions and A-propositions for the

n-vertex path graph, and the V-propositions for vertex $n+1$.)
Do the same for $L_{W,n+1}$, $L_{U,n+1}$, $A_{B,n+1}$, $A_{W,n+1}$.

15. A *vertex cover* in a graph G is a set VC of vertices such that every edge of G is incident with some vertex in VC.

A *clique* in a graph is a set of vertices that are pairwise adjacent. (I.e., every pair of vertices in the clique is linked by an edge.)

The *complement* of a graph G, written $\overline{G}$, is defined as follows. It has the same vertex set of G, and its edge set consists of every pair of vertices that are *not* adjacent in G.

Let n denote the number of vertices of the graph under discussion.

Suppose we have a graph, and that **chosen** is a unary predicate that takes a vertex of our graph as its argument. This predicate therefore defines a subset of the vertex set of the graph (the "chosen vertices"). Suppose also that we have the following predicates, with the indicated meanings.

vertex(X)	**X** is a vertex in our graph
edge(X,Y)	there is an edge between vertices **X** and **Y**

(a) Write a statement in predicate logic, using the predicates **vertex**, **edge** and **chosen**, to say that the chosen vertices form a vertex cover.

(b) Write a statement in predicate logic, using the same predicates, to say that the chosen vertices form a clique.

(c) Prove that, for any k, the number of vertex covers of size k in G equals the number of cliques of size $n-k$ in $\overline{G}$.

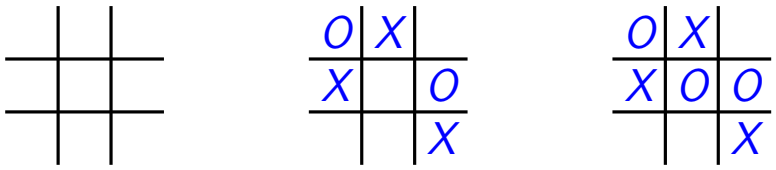
(d) Give the relationship between the size of the smallest vertex cover in G and the size of the largest clique in $\overline{G}$.

Monash University
Faculty of Information Technology
Semester 2, 2023

FIT2014
Exercises 2
Quantifiers, Games, Proofs

ASSESSED PREPARATION: Question 3.
You must provide a serious attempt at this entire question at the start of your tutorial.

1. This question is about the game *Noughts-and-Crosses* (known as *Tic-Tac-Toe* in the US). This game is played using a 3×3 grid, usually drawn in the manner of Figure **??**(a). Two players, *Crosses* and *Noughts*, each take turns to place *X* and *O*, respectively, in one cell of the grid. Once a cell is occupied by one player, its entry cannot be changed, and neither player can play there again. A player wins when they have three of their symbols in a line, horizontally, vertically, or diagonally, there being eight possible lines altogether; when that happens, the game stops. If all cells are occupied (five by Crosses and four by Noughts) and none of the eight three-cell lines have three identical symbols, then the game stops and is a Draw. (For simplicity, we forbid resignations and agreed draws.)



(a) (b) (c)

Figure 1: Some positions in Noughts-and-Crosses

The variable P always stands for a position in Noughts-and-Crosses, which represents the state of the array after the players have played some number of moves. A position thus corresponds to a 3×3 array with some subset of the cells occupied, in which the number of Crosses either equals, or exceeds by one, the number of Noughts. In the former case, then the next player to move must be Crosses; in the latter case, the next player to move must be Noughts. The diagram in Figure **??**(a) shows the position before anyone has moved; the next player to move is Crosses. The diagram in Figure **??**(b) shows a possible position after Crosses has had three turns and Noughts has had two turns; the next player to move is Noughts.

A move can be specified by naming the player whose turn it is and specifying the cell into which they place their symbol. For example, from the position of Figure **??**(b), the move "Noughts: centre cell" gives the position in Figure **??**(c).

A *winning move* is a move by a player that wins immediately, i.e., that completes the first line of three identical symbols seen in the game so far.

We use the following variables, predicates and function:

- The predicate $\mathsf{CrossesWins}(P)$ is True if, in position P, there is a line of three Crosses and no line of three Noughts.
- The predicate $\mathsf{NoughtsWins}(P)$ is True if, in position P, there is a line of three Noughts and no line of three Crosses.

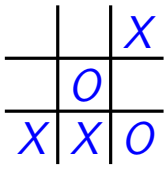
- The predicate CrossesToMove(P) is True if, in position P, it is the turn of Crosses to move (in other words, the numbers of Crosses and Noughts are equal, and no-one has won yet).
- The predicate NoughtsToMove(P) is True if, in position P, it is the turn of Noughts to move (in other words, the number of Crosses is one greater than the number of Noughts, and no-one has won yet).
- The function ResultingPosition($P, X_1, X_2, \ldots, X_k$) returns the position produced by starting with position P and then playing moves $X_1, X_2, \ldots, X_k$ (where $1 \leq k \leq 9$). For example, if P denotes the position in Figure **??**(b), then ResultingPosition(P,"Noughts: centre cell") gives the position in Figure **??**(c). If a move X_i is illegal — because it is out of turn, or in a cell that is already occupied — then it has no effect and leaves the position unchanged.

Using quantifiers and the variables, predicates and function described above, write statements in predicate logic with each of the following meanings.

(a) Crosses has a winning move in position P.

(b) Noughts has a winning move in position P.

(c) For each of the statements you wrote for (a) and (b), determine whether the statement is True or False when P is the following position.



By analogy with the definition of "winning move" given above, we could define a *losing move* to be a move that gives a position that is a win for the opponent, i.e., where there is a line of three of the opponent's symbols that did not exist in the game before. In Noughts-and-Crosses, this never happens. (Why?)

(d) Write a predicate logic statement to say that losing moves are impossible.

Now write predicate logic statements with each of the following meanings, where again P can be any Noughts-and-Crosses position, and P_0 is the initial position (when the 3×3 grid is empty: see Figure **??**(a)).

(e) Crosses has a strategy for winning within three moves from position P (where the three moves are one by Crosses, one by Noughts, and another by Crosses).

(f) Crosses does not have a strategy for winning within three moves from position P.
For this question, you must ensure that no logical negation occurs immediately in front of any quantifier.

(g) Crosses has a winning strategy from the initial position P_0.

(h) Noughts has a winning strategy from the initial position P_0.

(i) With best possible play from both sides from the initial position P_0, the game ends in a Draw.

(j) It is possible for Noughts to win from the initial position P_0.

2. A language L is called **hereditary** if it has the following property:

> For every nonempty string x in L, there is a character in x which can be deleted from x to give another string in L.

Prove by contradiction that every nonempty hereditary language contains the empty string.

3. Prove the following statement, by mathematical induction:

(*) For all n, the number of trees on n vertices is at least $(n-1)!$.

(A **tree** is a graph without cycles. These trees don't have roots but they do have leaves.)

Notation: let t_n be the number of trees on n vertices. Assume the vertices are numbered $1, 2, \ldots, n$. Trees with the same structure are still considered different if their vertices are numbered differently.

(a) *Pre-proof exploration:* First, draw all trees on one, two, and three vertices.
(b) Inductive basis: prove the statement $(*)$ for $n = 1$.

Now let $n \geq 1$.
Assume the statement $(*)$ true for n. This is our **Inductive Hypothesis**.

(c) Given a tree T on vertices $1, 2, \ldots, n$, show how to construct n different trees on vertices $1, 2, \ldots, n, n+1$ from T.
(d) Is it possible for two different trees on n vertices to give the same tree on $n+1$ vertices by the construction of part (c)?
(e) Using parts (c) and (d), write an inequality that relates t_n and t_{n+1}.
(f) Use your inequality from (e), together with the Inductive Hypothesis, to deduce a lower bound for t_{n+1}.
(g) When drawing your final conclusion, don't forget to briefly state that you are using the Principle of Mathematical Induction!

4. Prove, by induction on n, that for all $n \geq 1$, every tree on n vertices has $n-1$ edges.

Use the fact that every tree has a leaf, except for the trivial tree with one vertex and no edge. But it's also interesting to try to *prove* this fact.

5. (a) Prove, by induction on n, that for all $n \geq 3$,

$$n! \leq (n-1)^n.$$

(b) [Challenge]
Can you use a similar proof to show that $n! \leq (n-2)^n$? What assumptions do you need to make

about n? How far can you push this: what if 2 is replaced by a larger number? What is the best upper bound of the form $f(n)^n$ that you can find, where $f(n)$ is some function of n?

Supplementary exercises

6. Let P_n be the proposition $x_1 \wedge x_2 \wedge \cdots \wedge x_n$. Note that P_1 consists just of x_1, and P_n is equivalent to $P_{n-1} \wedge x_n$.

Prove by induction on n that, if $x_1 = \mathrm{F}$, then P_n is False.

7. Prove the following statement, by mathematical induction:

(*) The sum of the first k odd numbers equals k^2.

(a) First, give a simple expression for the k-th odd number.

(b) Inductive basis: now prove the statement $(*)$ for $k = 1$.

Assume the statement $(*)$ true for a specific value k. This is our **Inductive Hypothesis**.

(c) Express the sum of the first $k+1$ odd numbers ...

$$1 + 3 + \cdots + ((k+1)\text{-th odd number})$$

... in terms of the sum of the first k odd numbers, plus something else.

(d) Use the inductive hypothesis to replace the sum of the first k odd numbers by something else.

(e) Now simplify your expression. What do you notice?

(f) When drawing your final conclusion, don't forget to briefly state that you are using the Principle of Mathematical Induction!

8. This question is based on Lab 0, Section 8, Exercise 1.

Let *program* be any program that can be run in Linux and produces standard output. Suppose we do *program* | `wc` as above, followed by a sequence of further applications of | `wc`.

```
$ program | wc
    ...
$ program | wc | wc
    ...
$ program | wc | wc | wc
    ...
⋮
```

(a) Determine how many pipes are required before the output ceases to change, and what that output will be.

(b) Prove by induction that, whatever *program* is (as long as it produces some standard output), continued application of | `wc` eventually produces this same fixed output.

This is probably best done by proving, by induction on n, that if | `wc` is applied repeatedly to a file of $\leq n$ characters, it will eventually produce the fixed output you found in part (a).

9. The n-th *harmonic number* H_n is defined by

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

These numbers have many applications in computer science.

In this exercise, we prove by induction that $H_n \geq \log_e(n+1)$. (It follows from this that the harmonic numbers increase without bound, even though the differences between them get vanishingly small so that H_n grows more and more slowly as n increases.)

(i) Inductive basis: prove that $H_1 \geq \log_e(1+1)$.

(ii) *Assume* that $H_n \geq \log_e(n+1)$; this is our *inductive hypothesis*. Now, consider H_{n+1}. Write it recursively, using H_n. Then *use the inductive hypothesis* to obtain $H_{n+1} \geq \ldots\ldots$ (where you fill in the gap). Then use the fact that $\log_e(1+x) \leq x$, and an elementary property of logarithms, to show that $H_{n+1} \geq \log_e(n+2)$.

(iii) In (i) you showed that $H_1 \geq \log_e(1+1)$, and in (ii) you showed that **if** $H_n \geq \log_e(n+1)$ **then** $H_{n+1} \geq \log_e((n+1)+1)$. What can you now conclude, and why?

Advanced afterthoughts:

- The above inequality implies that $H_n \geq \log_e n$, since $\log_e(n+1) \geq \log_e n$. It is instructive to try to prove directly, by induction, that $H_n \geq \log_e n$. You will probably run into a snag. This illustrates that for induction to succeed, you sometimes need to prove something that is *stronger* than what you set out to prove.
- Would your proof work for logarithms to other bases, apart from e? Where in the proof do you use the base e?
- It is known that $H_n \leq (\log_e n) + 1$. Can you prove this?

10. The *Fibonacci numbers* F_n, where $n \in \mathbb{N}$, are defined by $F_1 = 1$, $F_2 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 3$. The first few numbers in the sequence are 1, 1, 2, 3, 5, 8, 13, ...

Prove by induction that the n-th Fibonacci number is given by

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

To make the algebra easier, give names to the two numbers $(1 \pm \sqrt{5})/2$ and use the fact that they both satisfy the equation $x^2 - x - 1 = 0$.

Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014
Exercises 3
Regular Languages, Inductive Definitions, Finite Automata, Kleene's Theorem I

There may not be time in class to cover all these exercises, so please attempt them *before* the class and make sure you ask questions about the ones you did not manage to solve confidently.

Remember that classes are not just about giving answers: you'll get these anyway, on Moodle, after the week of the class. The main aim is to *make you think* about the material covered during the lectures and, in particular, to generate discussion about some of the more interesting and challenging concepts.

The Supplementary Exercises are there for extra practice, and it is recommended that you attempt a selection of these, although you will not have time to cover them during the class.

ASSESSED PREPARATION: Question 5.
You must provide a serious attempt at this entire question at the start of your tutorial.

---

1. Write down all the words of length less than or equal to 8 described by the following regular expression.

$$(\mathbf{ab} \cup \mathbf{ba})(\mathbf{aa} \cup \mathbf{bb})^*\mathbf{ab}$$

2. For each of the following languages construct a regular expression defining each of the following languages over the alphabet $\{a, b\}$.

(i) All words that consist of exactly two letters.

(ii) All words that contain exactly two *b's* or exactly three *b's*, not more.

(iii) All strings that end in a double letter.

(iv) All strings that do not end in a double letter.

(v) All strings that have exactly one double letter in them.

(vi) All strings that have an even length.

(vii) All strings in which the letter b is never tripled. This means that no word contain the string bbb.

3. This question is inspired by the abstract game Zendo[1][2]

One round of the game has the following structure.

One player is known as the Master. When playing in the tutorial class, your Tutor will play this role to begin with. The Master does the following:

1. devise a regular expression, which we'll denote by R, and keep it secret;

[1] designed by Kory Heath (`http://www.koryheath.com/zendo/`), published by Looney Labs (`https://www.looneylabs.com/games/zendo`) on New Year's Eve, 1999

[2] Thanks to FIT2014 tutor Ben Jones for the idea for this question.

2. devise a string x which matches R, and another string y which does not match R;

3. reveal x and y to the other players, indicating which matches R and which does not.

The other players are each called Students. They do the following.

1. Each Student devise a string z and reveals it to everyone;

2. The Master tells everyone whether each student's string z matches R or not (but does not reveal R).

3. A Student — or group of Students working together — may guess what R is. Let S be the regular expression they guess.

4. If the guess S is right ($S = R$), then the Student wins this round, and the round ends.
   if the guess is wrong, then the Master devises and reveals to everyone a new string w, different from all previously revealed strings, on which R and S disagree (i.e., w matches R but not S, or matches S but not R).

5. Go back to Step 1.

Try this activity with a group of your fellow students. A group size of up to half-a-dozen should work ok.

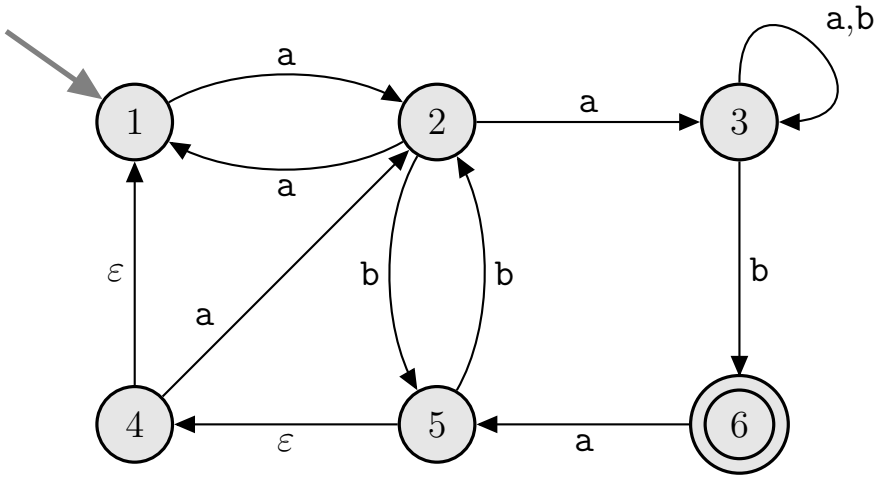
Some questions to consider as you do so:

- What sorts of regular expressions do you find work well for the Master?
- What sorts of regular expressions are easiest for the Students to guess?
- What modifications do you need to make to the rules, to make the game work better?

4. Build a Finite Automaton that accepts only those words that do not end in *ba*.

5.

Consider the following Nondeterministic Finite Automaton.



We will build up a proof, by induction on n, that

for all n, the string $(\texttt{aaab})^n$ is accepted by this NFA. (*)

(a) If the NFA is in state 1, what states can it be in after reading `aaab`?

(b) If the NFA is in state 6 and it then reads `aaab`, what states can it be in when it finishes reading that string?

(c) Is the string `aaab` accepted by the NFA? Why or why not?

Let $n \geq 1$.

(d) Suppose the string $(\texttt{aaab})^n$ is accepted by the NFA. What happens to the string $(\texttt{aaab})^{n+1}$, i.e., does the NFA accept or reject it? Why?

(e) Explain how to put (some of) your answers together to make a proof by induction of (*).

6. Find a Nondeterministic Finite Automaton which accepts the languages defined by the following Regular Expressions:

(i) $(a \cup b)^*(aa \cup bb)$

(ii) $((a \cup b)(a \cup b))^*$

(iii) $(aa \cup bb)^*$

(iv) $(ab)^*aa(ba)^*$

(v) $(ab \cup ba)(aa \cup bb)^*(ab \cup ba)$

7. Convert each of the Nondeterministic Finite Automata you found in the previous question into Finite Automata.

Supplementary exercises

8. Write down all the words described by the following regular expression.

$(\mathbf{aa} \cup \mathbf{bb})(\mathbf{ba} \cup \mathbf{ab})(\mathbf{aa} \cup \mathbf{bb})$

9. Write down all the words of length 6 described by the following regular expression.

$(\mathbf{aa} \cup \mathbf{bb})*$

10. A programmer used the following regular expression to describe dates where the day, month, and year are separated by "/".

$([\mathbf{0}-\mathbf{9}][\mathbf{0}-\mathbf{9}] \cup [\mathbf{0}-\mathbf{9}])/([\mathbf{0}-\mathbf{9}][\mathbf{0}-\mathbf{9}])/[\mathbf{0}-\mathbf{9}]^+$

Give an example of a valid date not described by this regular expression, and an invalid date described by this regular expression.

11. Construct a regular expression to describe times for the twenty-four hour clock, where the hour, minute, and second are separated by ":".

12. In the book "*The C programming Language*", by B.W. Kernighan and D.M. Ritchie, a floating point number is defined as an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an **E** or **e** followed by a possible signed integer. Construct a regular expression which describes a floating point number.

13.

Consider the recursive definition of regular expressions given in lectures.

Suppose we dropped subparts (i) and (ii) of part 3 of the definition (so we can't group or concatenate any more), but kept all other parts of the definition. What "regular expressions" would we get? What "regular languages" would result?

Challenge: Think about the effect of dropping other parts of the definition. Is any part of the definition redundant? If so, which one, and why? For any *essential* part of the definition, find a regular expression which would no longer satisfy the definition if that part were dropped.

14. In Linux, consult the man page for **grep** or **egrep** to learn the basics of how to use these utilities. (You may find **sed** and **wc** useful for this exercise too.)

Find a file of all English words, which may often be found in Unix/Linux systems in a location like `/usr/dict/words` or `/usr/share/dict/words`, or can easily be found on the web. You could also use one that you constructed in Lab 0.

Write a regular expression to match any vowel.

Write a regular expression to match any consonant.

Write a regular expression to match any word with no vowel or 'y'. Use **grep**, or one of its relatives, to find how many such words there are in your list.

Similarly, determine how many words have no consonants.

Write a regular expression to match any word whose letters alternate between consonants and vowels. What fraction of English words have this property?

What is the longest run of consonants in an English word? The longest run of vowels?

Can you use this tool to find a list of all English palindromes?

15. This question is about **one-dimensional Go**: see Tute 1, Q13.

Define

$\ell_{B,n}$:= the number of **legal** Go positions on the n-vertex path graph, where vertex n is **Black**.

$\ell_{W,n}$:= the number of **legal** Go positions on the n-vertex path graph, where vertex n is **White**.

$\ell_{U,n}$:= the number of **legal** Go positions on the n-vertex path graph, where vertex n is **Uncoloured**.

$a_{B,n}$:= the number of **almost legal** Go positions on the n-vertex path graph, where vertex n is **Black**.

$a_{W,n}$:= the number of **almost legal** Go positions on the n-vertex path graph, where vertex n is **White**.

(a) State the values of $\ell_{B,1}$, $\ell_{W,1}$, $\ell_{U,1}$, $a_{B,1}$, and $a_{W,1}$.
(b) Derive recursive expressions for $\ell_{B,n+1}$, $\ell_{W,n+1}$, $\ell_{U,n+1}$, $a_{B,n+1}$, and $a_{W,n+1}$, in terms of $\ell_{B,n}$, $\ell_{W,n}$, $\ell_{U,n}$, $a_{B,n}$, and $a_{W,n}$.
(c) How many of these quantities do you really need to keep, for each n, in order to work out the values for $n+1$?
(d) How do you work out the total number of legal positions on the n-vertex path graph, from $\ell_{B,n}$, $\ell_{W,n}$, $\ell_{U,n}$, $a_{B,n}$, and $a_{W,n}$?

16. This is a follow-up question to Q15, on one-dimensional Go.
(a) Write a regular expression for the set of strings that represent legal positions.
(b) Write a regular expression for the set of strings that represent almost legal positions.
(c) Does there exist a regular expression for the set of strings that represent positions that are neither legal nor almost legal?

17. In the lectures regular expressions were defined by a recursive definition. Give a recursive definition for arithmetic expressions, which use integers, the operators $+$, $-$, $/$, $*$, and brackets, ().

18. The language **PALINDROME** over the alphabet $\{a, b\}$ consists of those strings of a and b which are the same read forward or backward. Give a recursive definition for the language **PALINDROME**.

19. Construct a Finite Automaton that accepts only words with b as the second letter.

20. Construct a Finite Automaton that only accepts the words *baa*, *ab*, and *abb* and no other strings longer or shorter.

21. Build a Finite Automaton that accepts only those words that have *more* than four letters.

22. Build a Finite Automaton which only rejects the string *aba*.

23. Build a Finite Automaton that accepts only those words that have an even number of substrings *ab*.

24. Build a Finite Automaton that accepts precisely the strings of decimal digits that represent positive integers (with no leading 0s) that are multiples of 3.

25. Consider the Finite Automaton represented by the following table.

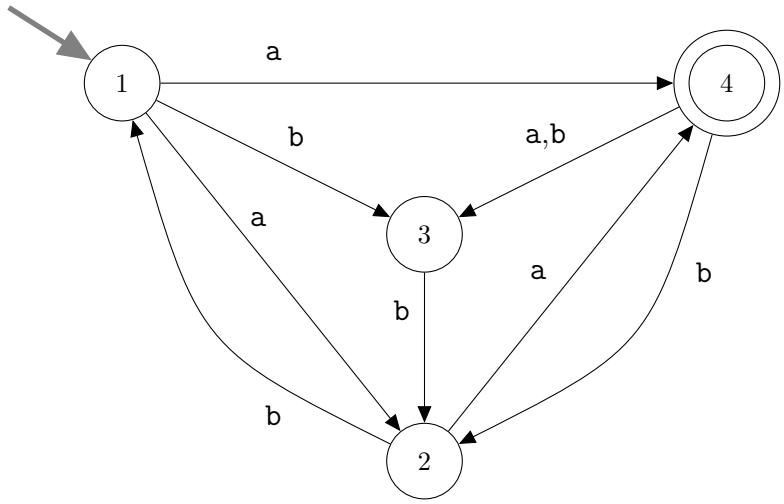
state		`a`	`b`
Start	1	2	1
Final	2	1	2

Prove, by induction on the string length, that this FA accepts every string in which `a` occurs an *odd* number of times.

26.

From FIT2014 Final Exam, 2nd semester 2016:

Consider the following Nondeterministic Finite Automaton (NFA).



(a) What are the possible states that this NFA could be in, after reading the input string `abba`?

(b) Prove, by induction on n, that for all positive integers n, the string $(\texttt{abba})^n$ is accepted by this NFA. (This string is obtained by n repetitions of `abba`.)

Monash University
Faculty of Information Technology
2^{nd} Semester 2023

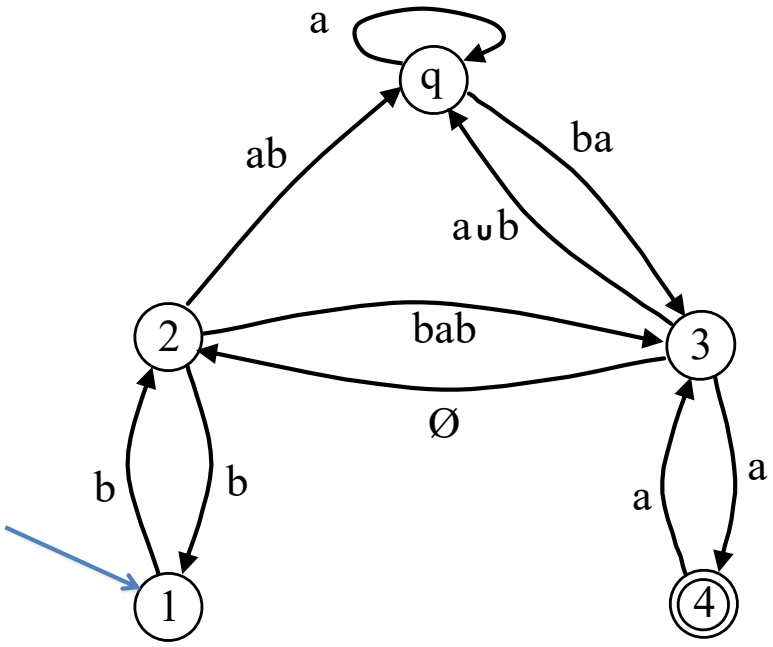
FIT2014
Exercises 4
Kleene's Theorem II and FA State Minimisation

ASSESSED PREPARATION: Question 4.
You must provide a serious attempt at this entire question at the start of your tutorial.

---

1. *Based on a question from FIT2014 Final Exam, 2nd semester 2015:*

Consider the following Generalised Nondeterministic Finite Automaton (GNFA). Construct an equivalent GNFA with the top state, q, removed.



2. [1]
You are in a maze represented by a rectangular grid. One cell is the start cell, another is the cell you want to get to. Some pairs of adjacent cells have a wall between them, preventing you from moving directly from one to the other. There is at least one path from the start to the end. Suppose that the characters U, D, L, R represent moving up, down, left and right by one grid cell. A string of these characters represents a sequence of movements through the maze, but is only valid if it does not make you bump into a wall. It's ok to visit cells more than once.

[1] Thanks to FIT2014 tutor Nathan Companez for this question.

Describe an algorithm for converting a given maze into a regular expression over the alphabet {U,D,L,R} which matches exactly those strings which correspond to sequences of moves which solve the maze.

Your algorithm may call any algorithm presented in lectures; if you do this, you do not have to list the steps in the algorithm from lectures that you're calling.

3. Consider the five-state Finite Automaton represented by the following table.

	state	a	b
Start	1	2	3
	2	1	5
	3	1	4
Final	4	2	5
Final	5	3	5

Convert this into an equivalent FA with the minimum possible number of states.

4. Consider the six-state Finite Automaton represented by the following table.

	state	a	b
Start	1	2	4
	2	2	6
	3	3	4
Final	4	5	3
Final	5	5	5
Final	6	5	1

Convert this into an equivalent FA with the minimum possible number of states.

5. For each Finite Automaton you found in last week's Question 7 (in Exercise Sheet 3), find the corresponding minimum state Finite Automaton.

6. For each Finite Automaton you found in the previous question, find the corresponding regular expression (using the GNFA approach).

Supplementary exercises

7. Construct a minimum state Finite Automaton that accepts only those strings defined by the regular expression: -?($N|$ $N.|$ $N.N|$ $.N$), where $N = [0–9]^+$.

8. We can prove that two regular expressions are equivalent by showing that their minimum state Finite Automaton are the same, except for state names. Using this technique, show that the following regular expressions are equivalent:

i. $(a \cup b)^*$

ii. $(a^* \cup b^*)^*$

iii. $((\epsilon \cup a)b^*)^*$

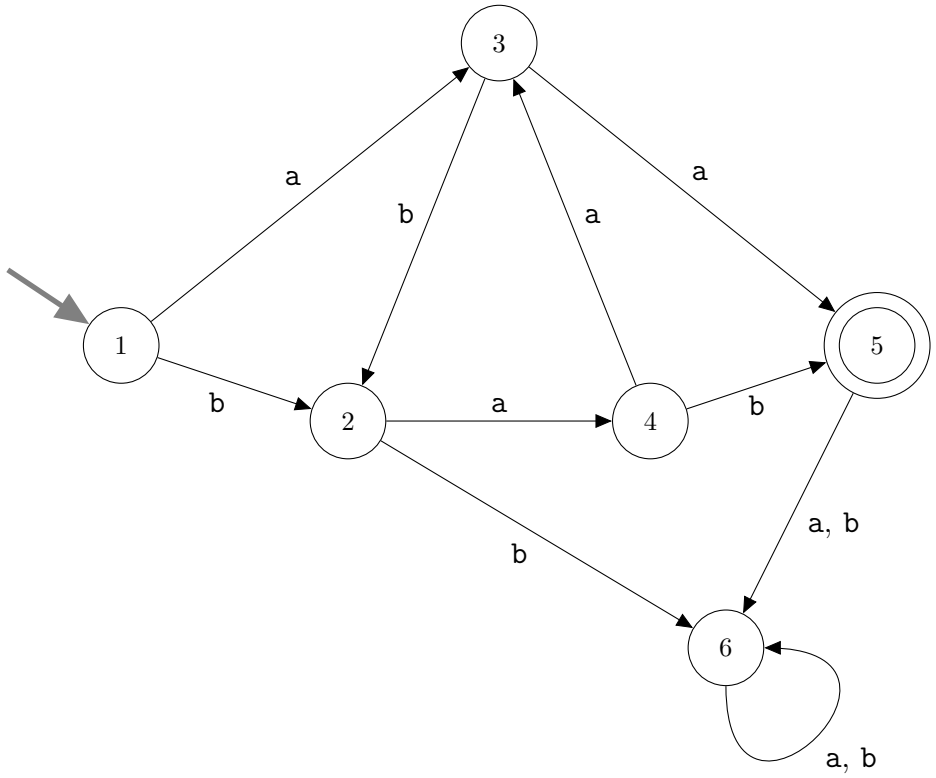
Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014
Exercise Sheet 5
Pumping Lemma, and Context Free Languages

Although you may not need to do all the many exercises in this Exercise Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

ASSESSED PREPARATION: Question 5.
You must provide a serious attempt at this entire question at the start of your class.

1.
Consider the following Finite Automaton, called SHAUN:



The **SHAUN Game** is played between two players, Reg and Nona, as follows. They take turns, with Reg moving first. The game is very short: Reg moves first, then Nona moves, then Reg moves, then Nona moves, making only four moves in total. The rules for their moves are as follows.

- Reg first chooses a number N.
- Nona chooses a string w accepted by SHAUN the Finite Automaton such that $|w| > N$.
- Then Reg divides w up into substrings x, y, z such that $y \neq \varepsilon$ and $|xy| \leq N$.
- Then Nona chooses a non-negative integer i.

The result of the game is that:

- if xy^iz is accepted by SHAUN, then Reg wins;
- if xy^iz is rejected by SHAUN, then Nona wins.

For example, here are two possible games between Reg and Nona.

- **First game:**
  1. **Reg:** chooses $N = 2$.
  2. **Nona:** chooses $w =$ `bab`.
  3. **Reg:** chooses $x = \varepsilon$, $y =$ `ba`, $z =$ `b`.
  4. **Nona:** chooses $i = 3$.

  Outcome: **Nona wins**, because $xy^iz = xy^3z =$ `bababab` is *rejected* by SHAUN.
- **Second game:**
  1. **Reg:** chooses $N = 5$
  2. **Nona:** chooses $w =$ `abaaa`.
  3. **Reg:** chooses $x =$ `a`, $y =$ `baa`, $z =$ `a`.
  4. **Nona:** chooses $i = 2$.

  Outcome: **Reg wins**, because $xy^iz = xy^2z =$ `abaabaaa` is *accepted* by SHAUN.

(a) Play this game *twice* with one of your fellow FIT2014 students (or a friend or family member, if you prefer), using different moves to those shown in the example games above, and different moves in each game. For the second game, you must reverse the roles you had in the first game. So each player plays one game as Reg, and one as Nona. Record each of the games, showing for each game:

- the name of each player, and which role they each played (Reg/Nona),
- the sequence of moves by each player,
- the outcome of the game.

(b) Using quantifiers, write down the assertion that Reg has a winning strategy. (I.e., he can choose a first move such that, no matter what Nona does in reply, Reg can choose his second move so that, for any last move by Nona, the outcome is that Reg wins.)

When doing this, assume you have a predicate $\mathsf{ShaunAccepts}(w)$ which takes a string w as its sole argument and is True if SHAUN accepts w and False if SHAUN rejects w.

(c) Using quantifiers, write down the assertion that Nona has a winning strategy.

(d) One of the players has a winning strategy. Determine who this is, and describe the winning strategy.

2.

Let SHAWN be the language represented by the regular expression $(\mathtt{a} \cup \varepsilon)(\mathtt{baa})^*(\mathtt{a} \cup (\mathtt{bab}))$.
The **SHAWN Game** is played just like the SHAUN Game, except that

- in Nona's first move, she is required to choose a string w of length $> N$ that *belongs to the language* SHAWN (instead of being accepted by the Finite Automaton SHAUN);
- the condition for Reg to win is that the final string xy^iz *belongs to the language* SHAWN (instead of being accepted by the Finite Automaton SHAUN);
- the condition for Nona to win is that the final string xy^iz *does not* belong to the language SHAWN (instead of being rejected by the Finite Automaton SHAUN).

One of the players has a winning strategy. Determine who this is, and describe the winning strategy.

3.

The HALF-AND-HALF *language* is defined by

$$\text{HALF-AND-HALF} \;=\; \{\mathtt{a}^n\mathtt{b}^n \;:\; n \in \mathbb{N}\}.$$

The **HALF-AND-HALF Game** is also played between Reg and Nona, exactly as for the SHAWN game except that the language HALF-AND-HALF is used instead. So, the string w must be chosen to belong to HALF-AND-HALF, and the outcome is determined by whether or not the string xy^iz belongs to HALF-AND-HALF. The rules for the moves are as follows.

- Reg first chooses N
- Nona chooses a string $w \in$ HALF-AND-HALF such that $|w| > N$.
- Then Reg divides w up into substrings x, y, z such that $y \neq \varepsilon$ and $|xy| \leq N$.
- Then Nona chooses a non-negative integer i.

The result of the game is that:

- if $xy^iz \in$ HALF-AND-HALF, then Reg wins;
- if $xy^iz \notin$ HALF-AND-HALF, then Nona wins.

(a) Play this game twice, for practice.
(b) One of the players has a winning strategy. Determine who this is, and describe the winning strategy.

4. The games SHAWN and HALF-AND-HALF are instances of a huge class of games called **Pumping Games**. You can play a Pumping Game for *any* language. As usual, the players are Reg and Nona. First, the players are given a language L (which may or may not be regular). Reg moves first, then Nona moves, then Reg moves, then Nona moves. The rules for their moves are as follows.

- Reg first chooses a number N
- Nona chooses a string $w \in L$ with $|w| > N$.
- Then Reg divides w up into substrings x, y, z such that $y \neq \varepsilon$ and $|xy| \leq N$.
- Then Nona chooses a non-negative integer i.

The result of the game is that:

- if $xy^iz \in L$, then Reg wins;
- if $xy^iz \notin L$, then Nona wins.

(a) Using quantifiers, write down the assertion that Reg has a winning strategy.
(b) Using quantifiers, write down the assertion that Nona has a winning strategy.
(c) What does the Pumping Lemma tell us about circumstances in which winning strategies exist?

5. Let VERY-EVEN be the language, over {0,1}, consisting of all binary representations of positive integers that (i) are multiples of 2^n and (ii) have at most $3n$ bits altogether, where $n \in \mathbb{N}$.

Note that a binary number is a multiple of 2^n if and only if its last n bits are all zero. We assume that our positive binary numbers have no leading zeros, so that their leftmost bit is always 1.

Examples of strings in this language:

- With $n = 1$, our number must be even and have at most three bits, so we have the strings 10, 100 and 110, which are binary representations of 2, 4 and 6 respectively.
- With $n = 2$, our number must be a multiple of $2^2 = 4$ and have at most six bits, so we have the strings

  100, 1000, 1100, 10000, 10100, ..., 111000, 111100,

  being binary representations of all multiples of 4 between 4 and 60 inclusive:

  4, 8, 12, 16, 20, ..., 56, 60.

Examples of strings <u>not</u> in this language:

- 1, because in this case there are $n = 0$ zeros on the right, so the length bound is $3n = 3 \times 0 = 0$, so this string is too long!
- 1000100: it has two zeros on the right, signifying that it is a multiple of $2^2 = 4$ (but not of 8). But it has length 7 which violates the length bound of $3n = 6$.

We will first use the Pumping Lemma for Regular Languages to prove, step by step, that VERY-EVEN is not regular. Then we will show that it is context-free and give a derivation using it.

(a) First, state your initial assumption about VERY-EVEN.

(b) From that assumption, you know that there exists a certain number which we usually call N. Where does N come from? What assumption can we make about it?

(c) Now design a string w of length $\geq N$ whose structure depends on N in some way.

- The intention is that the structure of w will help in the later steps of the proof. You may not know *yet* how to choose w; that's ok, you can try some different possibilities and see how they work and come back and revisit your choice of w later, if necessary.
- Make sure that w depends on N. The design of w should be such that, for any *specific* value of N, you get one *specific* string w. So you are really describing an infinite *family* of strings, parameterised by N.
- It is not necessary for your design to encompass *all possible* strings in the language. In fact, some strings will help this proof but others may not help. So you only need to design a *certain type* of string in the language that helps the proof.

(d) Now look at all possible placements of a nonempty substring y within w such that y lies within the first N bits of w. Describe these possible placements. Can you classify them into a small number of cases?

(e) Show how, for each such placement, you can find some $i \geq 0$ such that xy^iz is not in VERY-EVEN.

If your w doesn't help you do this, think about why not, and go back to step (c) and try a different w.

(f) What do you conclude, and why?

(g) Let's be evil and reconsider step (c) from a sinister angle. Suppose your enemy is trying to answer this question and they are struggling to design their string w. Which string w could you suggest to them so that the rest of their proof will not work? In order for your deception to be credible, your evil w still needs to be in the language, must still depend on N, and must have length $\geq N$.

(h) Find a Context-Free Grammar for VERY-EVEN.

(i) Using your CFG, give a derivation of the string `100100000` (the binary representation of 288).

6. Some programs (e.g., the editor **emacs**), allow a regular expression to contain $\backslash n$, where n is a digit. This must be matched by another copy of whatever substring matched the subexpression in the n-th pair of parentheses in the regular expression.

For example, suppose we have the expression **a(b*)c\1a**. This matches the string **abbbbcbbbba**, since the first **bbbb** matches what's within the first parentheses (i.e., **b***), so that any occurrence of **\1** must also be matched by that exact string, **bbbb**. The expression is not matched by the string **abbbbcbba**, since the second string of **b**'s is different to the first and therefore cannot match **\1**.

Prove or disprove: every language described by an extended regular expression of this type is regular.

7. Let CENTRAL-ONE be the language of binary strings of odd length whose middle bit is 1.
(a) Prove or disprove: CENTRAL-ONE is regular.
(b) Prove or disprove: CENTRAL-ONE is context-free.

8. Consider the following Context Free Grammar for arithmetic expressions:

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E + T \mid E - T \mid T \\ T &\rightarrow T * F \mid T/F \mid F \\ F &\rightarrow \textbf{int} \mid (E) \end{aligned}$$

where **int** stands for any integer. Find parse trees for each of the following arithmetic expressions.

i) $4 + 6 - 8 + 9$

ii) $(4 + 10)/(8 - 6)$

iii) $(2 - 3) * (4 - 8)/(3 - 8) * (2 - 4)$

9. In this question, you will write a Context Free Grammar for a very small subset of English.

For many years, children in Victorian schools learned to read from *The Victorian Readers* (Ministry of Education, Victoria, Australia, 1928; many reprints since). Some of you may have grandparents (or even parents!) who used these books in school.

The *First Book* of this series (there were eight altogether) contains simple sentences, with illustrations. Among these sentences are:

I can hop.
I can run.
I can stop.
I am big.
I am six.
I can dig.
I can run and hop and dig.
Tom can hop and dig.
Tom is big.
Tom and I can run.

(a) Write a simple CFG in BNF which can generate these sentences and a variety of others.
(b) Using your grammar, give a derivation and a parse tree for the sentence

Tom and I can dig and hop and run.

10. Prove the following:

> If a string in a context-free language has a derivation of length n, then it has a leftmost derivation of length n.

The simplest approach is to think about parse trees ...
For a more complicated proof, by induction, see Q20.

Supplementary exercises

11.

Recall that the EVEN-EVEN *language* consists of all strings over alphabet $\{\mathtt{a}, \mathtt{b}\}$ that contain an even number of as and an even number of bs.

The **EVEN-EVEN Game** is played between two players, Reg and Nona, as follows. They take turns, with Reg moving first. The game is very short: Reg moves first, then Nona moves, then Reg moves, then Nona moves, making only four moves in total. The rules for their moves are as follows.

- Reg first chooses a number N.
- Nona chooses a string $w \in$ EVEN-EVEN such that $|w| > N$.
- Then Reg divides w up into substrings x, y, z such that $y \neq \varepsilon$ and $|xy| \leq N$.
- Then Nona chooses a non-negative integer i.

The result of the game is that:

- if $xy^iz \in$ EVEN-EVEN, then Reg wins;
- if $xy^iz \notin$ EVEN-EVEN, then Nona wins.

Here are two possible games between Reg and Nona.

- **First game:**
  1. **Reg:** first chooses $N = 4$.
  2. **Nona:** chooses $w = \mathtt{ababbaab}$.
  3. **Reg:** chooses $x = \varepsilon$, $y = \mathtt{abab}$, $z = \mathtt{baab}$.
  4. **Nona:** chooses $i = 2$.

  Outcome: **Reg wins**, because $xy^iz = xy^2z = \mathtt{abababbbaab} \in$ EVEN-EVEN.
- **Second game:**
  1. **Reg:** first chooses $N = 4$.
  2. **Nona:** chooses $w = \mathtt{ababbaab}$.
  3. **Reg:** chooses $x = \mathtt{ab}$, $y = \mathtt{ab}$, $z = \mathtt{baab}$.
  4. **Nona:** chooses $i = 0$.

  Outcome: **Nona wins**, because $xy^iz = xy^0z = \mathtt{abbaab} \notin$ EVEN-EVEN.

(a) Play this game twice, for practice.
(b) Using quantifiers, write down the assertion that Reg has a winning strategy.

(c) Using quantifiers, write down the assertion that Nona has a winning strategy.

(d) One of the players has a winning strategy, provided $N \geq 4$. Determine who this is, and describe the winning strategy.

12. Prove or disprove:

> In a Context-Free Grammar, if the right-hand side of every production is a palindrome, then any string in the generated language is a palindrome.

13. Recall that $\overleftarrow{x}$ denotes the reverse of string x. If L is any language, define $\overleftarrow{L} = \{x : \overleftarrow{x} \in L\}$, i.e., the set of all reversals of strings in x.

Prove that, if L is a Context-Free Language, then so is $\overleftarrow{L}$.

14. Use Questions 10 and 13 to prove that, if a string in a context-free language has a derivation of length n, then it has a rightmost derivation of length n.

15. For our purposes, a *permutation* is represented as a string, over the three-symbol alphabet containing 0, 1 and "," (comma), as follows: each of the numbers $1, 2, \ldots, n$ is represented in binary, and the numbers are given in some order in a comma-separated list. Each number in $1, 2, \ldots, n$ appears exactly once, in binary, in this list. For example, the permutation 3,1,2 is represented by the string "11,1,10".

Prove that the language of permutations is not regular.

16.

(a) Prove that the difference between two squares of two consecutive positive integers increases as the numbers increase.

(b) Use the Pumping Lemma to prove that the language $\{\mathbf{a}^{n^2} : n \in \mathbb{N}\}$ is not regular.

(c) Hence prove that the language of binary string representations of adjacency matrices of graphs is not regular.

Definitions.

The *adjacency matrix* $A(G)$ of a graph G on n vertices is an $n \times n$ matrix whose rows and columns are indexed by the vertices of G, with the entry for row v and column w being 1 if v and w are adjacent in G, and 0 otherwise.

The *binary string representation* of $A(G)$ is obtained from $A(G)$ by just turning each row of $A(G)$ into a string of n bits, and then concatenating all these strings in row order, to form a string of n^2 bits.

17.

For this question, you may use the fact that there exist arbitrarily long sequences of positive integers that are not prime. In other words, for any N, there is a sequence of numbers x, $x+1$, $\ldots$, $x+N$ none of which is prime.[1]

Prove that the language $\{\mathbf{a}^p : p \text{ is prime}\}$ is not regular.

18. Consider the following Context Free Grammar.

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow E \cup T \mid T \\ T &\rightarrow TF \mid F \\ F &\rightarrow F^* \mid (E) \mid a \mid b \mid \epsilon \end{aligned}$$

[1] Mathematically-inclined students are encouraged to try to prove this fact for themselves.

Note: in the last line, ϵ is an actual symbol (i.e., one letter); it is not the empty string itself, but rather a symbol used in regular expressions to match the empty string.

(a) Find the **leftmost** and **rightmost** derivations of the following words generated by the above Context Free Grammar.

i) $a^*b^* \cup b^*a^*$

ii) $(aa \cup bb)^*$

iii) $(a \cup \epsilon)(b \cup \epsilon)(a \cup \epsilon)$

(b) Prove that the language generated by this grammar is not regular.

19. You saw in Q18 that the language of regular expressions, over the eight-character alphabet $\{$a,b,ε,$\cup$,(,),$*$,$\emptyset\}$, is not, itself, regular(!), but it is context-free.

Now, prove that the language of context-free grammars is regular. Assume that the non-terminal symbols are S, $X_1, \ldots X_m$, where S is the start symbol, and that the terminal symbols are $x_1, \ldots, x_n$. So the alphabet for your regular expression is $\{S, X_1, \ldots X_m, x_1, \ldots, x_n, \varepsilon, \rightarrow\}$.

But don't get confused about this! It certainly does *not* follow that every context-free language is regular! Remind yourself of the actual relationship between regular languages and context-free languages.

20. Prove the following theorem, by induction on derivation length:

> If a string in a context-free language has a derivation of length n, then it has a leftmost derivation of length n.

It may help to do this by proving the following stronger result.

> Let σ be a string, which can contain terminal and non-terminal symbols. If a string in a context-free language has a derivation from σ of length n, then it has a leftmost derivation from σ of length n.

Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 10
Complexity: P, NP, Polynomial Time Reductions

ASSESSED PREPARATION: Question 2.
You must provide a serious attempt at parts (a)–(g) of this question at the start of your class.

---

1. Recall Exercise Sheet 7, Question 4, where you built a Turing machine that interchanges the first and last letters of the input string. Review the discussion in part (d) of the time complexity $t(n)$.

Justify the claim in the solutions that this time complexity satisfies $t(n) = O(n)$.

2. This question is about developing big-O bounds for the time complexity of constructing φ_G from G, in Assignment 1, Problem 2.

- Assume that, as in the Assignment, the input graph is given as an edge list.
- Assume that G has no isolated vertices.
- Assume that G is a **simple graph**, i.e., that it has no loops or multiple edges.
- Use the expression from Problem 3.
- Assume that the time taken to construct each clause is at most a constant.

(a) Give an upper bound for the number n of vertices of G in terms of the number m of edges of G.
(b) Express the time complexity in big-O as a function of m.
(c) Give an upper bound for m in terms of n.
(d) Express the time complexity in big-O as a function of n.
(e) The actual length of G, as an input to this construction, is taken to be the number of characters in its input file, ignoring all spaces. Give a lower bound for the length of G in terms of m.
(f) Express the time complexity of the algorithm in big-O notation as a function of the length of G (which we can denote by $|G|$).
(g) We can now see that this algorithm, for $G \mapsto \varphi_G$, is a mapping reduction. What language does it reduce <u>from</u> and what language does it reduce <u>to</u>? What type of mapping reduction is it?

(h) Give an upper bound for the length of G in terms of m and n.
(i) Prove that the length of G is $O(m \log m)$.

3.

You are running a program with running time an^5, where a is a constant.

(a) Assuming Moore's Law, how long do you have to wait before your program can solve inputs that are four times as large as those you can solve now, in the same time?

(b) Your friend has a program that runs in time 2^n. Give a lower bound on how long she has to wait, under the same scenario. (Assume for this part that your input size is at least 40.)

4.

Suppose you work in bioinformatics, and you have written a program that takes, as input, a DNA string, regarded as a sequence of *bases*, where each base is one of C,G,A,T. Your program runs in time $5n^3$, where n is the length of the input string.

Now, your boss insists on measuring string length in bits, where each of the four bases is represented by two bits.

(a) How would you state your program's running time, if n now represents the number of *bits* in the input string?

(b) You rush off to change the description of the program's running time in the documentation. You discover that, in your team, documentation gives all running times in big-O notation. What changes do you need to make to the big-O statement of the running time?

5.

Suppose algorithm A takes an adjacency matrix[1] of a graph as input and solves a particular graph-theoretic problem Π in polynomial time.

1. Prove that, if the graph is instead given as an edge-list representation[2], and we restrict to graphs without isolated vertices, then the problem Π can still be solved in polynomial time.

2. Does this mean that the way the input is represented does not matter? (See also Q10.)

[1] An *adjacency matrix* for a connected graph with v vertices is a $v \times v$ matrix whose rows and columns are indexed by the vertices, where the entry in row i and column j is 1 if vertices i and j are adjacent in the graph, and 0 otherwise. So the matrix consists entirely of 0s and 1s. How many bits do you need to represent a graph in this way?

[2] An *edge list* for a graph with v vertices and e edges consists of a list of e pairs, one for each edge. A pair (i, j) represents an edge between vertices i and j. If vertices i and j are not adjacent, then the pair (i, j) does not appear in the list. How many bits do you need to represent a graph in this way?

6. A *k-edge-colouring* of a graph G is a function that assigns to each edge a colour from the set $\{1, 2, \ldots, k\}$ such any two edges that have an endpoint in common receive different colours.

Consider the following problem.

EDGE-COLOURING
INPUT: Graph G, positive integer k.
QUESTION: Does G have a k-edge-colouring?

(a) Prove that EDGE-COLOURING $\in$ NP.

(b) Can you find a value of k such that, if we restrict to that particular k, the problem can be solved in polynomial time?

7.

Consider the following problem.

SHORTEST REGEXP
INPUT: Regular expression E, positive integer k.
QUESTION: Does there exist another regular expression F that is equivalent to E (i.e., they both represent the same language) and which has length $\leq k$?

Here is an *attempt* at an outline of a proof that SHORTEST REGEXP belongs to NP. Comment on what is correct and incorrect in this proof.

(1) Consider the following verifier:
   (i) Input: Regular expression E, positive integer k.
   (ii) Certificate: another regular expression, F.
   (iii) Convert E to a Finite Automaton A that recognises the language that E represents (using the methods from the proof of Kleene's Theorem).
   (iv) Convert F to a Finite Automaton B that recognises the language that F represents.
   (v) Apply the state minimisation algorithm to A. This produces a Finite Automaton $A^{\min}$ that recognises the same language as A and is the unique FA with fewest states that does this.
   (vi) Apply the state minimisation algorithm to B. This produces a Finite Automaton $B^{\min}$ that recognises the same language as B and is the unique FA with fewest states that does this.

(vii) **If** $A^{\min} = B^{\min}$ **and** $|F| \leq k$ **then** ACCEPT,

(viii) **else** REJECT.

(2) Each of the algorithms used in this verifier runs in polynomial time.

(3) To test whether $A^{\min} = B^{\min}$, just test that the state names correspond and that the transitions between state pairs also correspond. This takes polynomial time.

(4) Testing whether $|F| \leq k$ can be done in polynomial time.

(5) Therefore the entire verifier runs in polynomial time.

(6) The verifier accepts if and only if the two regular expressions E and F represent the same language (by Kleene's Theorem) *and* $|F| \leq k$. Therefore the verifier accepts if and only if $(E, k) \in$ SHORTEST REGEXP. So it is indeed a verifier for SHORTEST REGEXP.

(7) Since it is a polynomial time verifier for SHORTEST REGEXP, we conclude that SHORTEST REGEXP $\in$ NP.

8.

Consider the following problem.

NEARLY SAT
INPUT: Boolean expression φ in CNF.
QUESTION: Is there a truth assignment that satisfies all, or all except one, of the clauses of φ?

(a) Prove that NEARLY SAT belongs to NP.

(b) Give a polynomial-time reduction from SAT to NEARLY SAT and prove that it is such a polynomial-time reduction.

Supplementary exercises

9. Recall Exercise Sheet 9, Q8, about **doubling** and **halving**, and the following operation on languages.

Let L be any nonempty non-universal language that is closed under doubling and halving. Let L^{even} be the set of all strings of even length in L.

Prove that $L \in \mathrm{P}$ if and only if $L^{\text{even}} \in \mathrm{P}$.

10.

(a) Write a simple, naive algorithm which takes a positive integer x as input and outputs the smallest factor of x which is greater than 1 and less than x (if such exists).

For example, if $x = 6$, your algorithm should output 2. If $x = 7$, your algorithm should report that no such factor exists (since the only factors of 7 are 1 and 7).

For this question, you may assume (not quite realistically) that testing integer divisibility takes constant time.

(b) If the integer is given in *unary* (i.e., as a string of x 1s), what is the complexity of the algorithm, in big-O notation, in terms of the input size n?

(c) Does this algorithm run in polynomial time?

(d) If the integer is given in *binary*, what is the complexity of the algorithm, in big-O notation, in terms of the input size n?

(e) Does the algorithm run in polynomial time now?

(f) What would happen if you used some other base $b > 2$?

(g) [Advanced] Now suppose that each positive integer x is encoded by a sequence giving the successive exponents in its unique prime factorisation. So, if $x = 126$, then its prime factorisation is

$$126 = 2 \cdot 3 \cdot 3 \cdot 7 = 2^1 \cdot 3^2 \cdot 5^0 \cdot 7^1$$

so this x is represented by the sequence $(1, 2, 0, 1)$.

(i) How feasible would it be to use your algorithm with this way of encoding integers?

(ii) Write a faster algorithm, based on this representation, and determine its big-O complexity.

11.

A **short verifier** is a verifier whose certificate consists of ≤ 100 characters.

Let NP[SHORT] be the class of languages with polynomial-time *short* verifiers.

(a) Prove that P = NP[SHORT].

Challenge:

More generally, for any function $f : \mathbb{N} \cup \{0\} \to \mathbb{N} \cup \{0\}$, define NP$[f(n)]$ be the class of languages accepted by polynomial-time verifiers whose certificates consist of $\leq f(n)$ characters, where n is the input size. So, for example, NP[100] is just NP[SHORT]. In that case the function is just the constant function whose value is always 100. But think about what can happen when f is *unbounded*, i.e., there is no constant b such that $f(n) \leq b$ for all n.

(b) Can you find an *unbounded* function f such that P = NP$[f(n)]$?

(c) Can you find an *unbounded* function f such that NP = NP$[f(n)]$?

Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 7
Turing Machines

ASSESSED PREPARATION: Question 4.
You must provide a serious attempt at this entire question at the start of your class.

1.

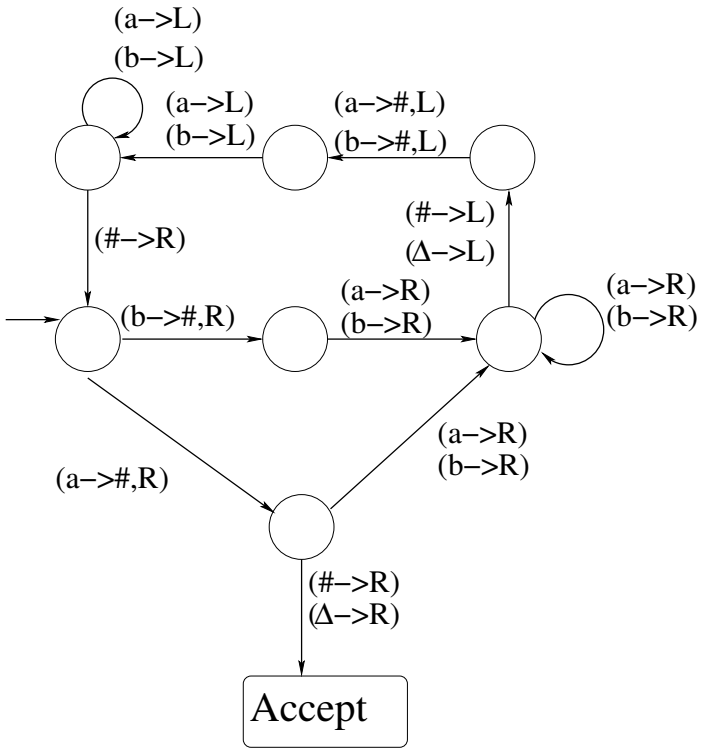
Trace the execution of the following input strings on the machine shown.

i) *aaa*

ii) *aba*

iii) *baaba*

iv) *ababb*



2. For the following inputs trace the execution of the Turing Machine built in Lectures to compute the function $f(n) = 2n$.

i) Δ

ii) a

iii) aa

3. What is the main difference between Final States in Finite Automata and Accept States in Turing machines?

4. In this question, you will build a Turing machine that interchanges the first and last letter of the input string.

Some examples:

input		output
ε	$\longmapsto$	ε
`a`	$\longmapsto$	`a`
`b`	$\longmapsto$	`b`
`ab`	$\longmapsto$	`ba`
`baa`	$\longmapsto$	`aab`
`abba`	$\longmapsto$	`abba`
`babbb`	$\longmapsto$	`babbb`

(a)

First, design it on paper, and write your Turing machine as a table or a diagram.

(b)

Then build it in Tuatara v2.1 and demonstrate it there.

Some things to consider:

- Develop your TM in small stages. Here's one approach. To begin with, try writing a TM that just replaces the last letter by `a`. Then modify it so that it replaces the last letter by the first letter. Then try to extend it to do the whole task.
- Once you've built your TM, think about what it does when it returns to the first tape cell. Does it try to move left from that first cell? Our TMs in lectures crash if that happens, but Tuatara TMs don't crash in that situation: instead, they just stay still at the first tape cell. If you Tuatara TM exploits this slackness of Tuatara, modify it so that it behaves properly.

(c) (optional)

If your Tuatara TM diagram is particularly beautiful and elegant, ask your tutor's opinion of its aesthetic and artistic qualities.

(d)

Let $t(n)$ be the maximum number of steps your TM takes, over all input strings of length n. Find an expression for $t(n)$, as a function of n.

If finding an exact expression is too hard, find the best upper bound expression that you can. (Your expression should still be a function of n, and it must be $\geq t(n)$ for all n, but as close to $t(n)$ as you can get.)

5. Build a Turing Machine that accepts the language $\{a^n b^{n+1}\}$.

6. Build a Turing Machine which computes the function, $f(n) = n/2$.

7.

Describe the classes of languages recognised by each of the following:

(i) TMs which cannot move on the tape.

(ii) TMs which only move to the right.

(iii) TMs which only stay still or move to the right (i.e., cannot move to the left).

(iv) TMs which operate like normal TMs except that, if they attempt to move Left when in the first tape cell, they stay still instead of crashing.

8. Explain how a Turing machine can be simulated by a generalised Pushdown Automaton with two stacks (2PDA).

(This is like an ordinary PDA except that a transition is specified by: the input character, the characters at the top of each of the two stacks (which are both popped), and the characters that are then pushed onto each of the two stacks. Any of these characters may be replaced by ε, with the usual meaning.)

9.

Suppose you have a Turing machine M. In this question, we will use the following propositions about M.

- For each time t, each tape position i and each symbol s, the propostion $A_{t,i,s}$ means:

  At time t, tape cell i contains symbol s.

- For each time t and each state q, the propostion $B_{t,q}$ means:

  At time t, the machine is in state q.

- For each time t and each tape position i, the propostion $C_{t,i}$ means:

  At time t the Tape Head is at tape cell i.

The time t can be any non-negative integer (with start time being $t = 0$). The tape cell i can be any positive integer. The state q can be any number in $\{1, 2, \ldots, N\}$, where N is the number of states. The symbol s is a member of $\{\mathbf{a},\mathbf{b},\Delta\}$, where as usual Δ denotes the blank symbol.

Use the above propositions to construct logical expressions with the following meanings.

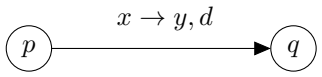
(a) At time t, tape cell i has at least one symbol in it.

(b) At time t, tape cell i has at most one symbol in it.

(c) At time t, tape cell i has exactly one symbol in it.

(d) At time t, and with Tape Head at tape cell i, **if** the current state is p and the symbol in cell i is x, **then** at time $t + 1$ the state is q, the symbol in cell i is y, and the Tape Head is at cell $i + d$, where $d \in \{\pm 1\}$.

This describes the following transition:



10.

Suppose that a particular Turing machine T always stops in $\leq n^2$ steps, where n is the length of the input string.

Let U be a Universal Turing Machine (UTM), which takes as input a pair (M, x) where M is any Turing machine and x is an input string for M.

Suppose that any computation by any M that takes t steps can be simulated by U in $\leq t^3$ steps.

Derive an upper bound for the time taken by U to simulate T on an input of length n.

Supplementary exercises

11. Build a TM that accepts the language of all words that contain the substring *bbb*.

12. Build a TM that accepts the language of all words that **do not** contain the substring *bbb*.

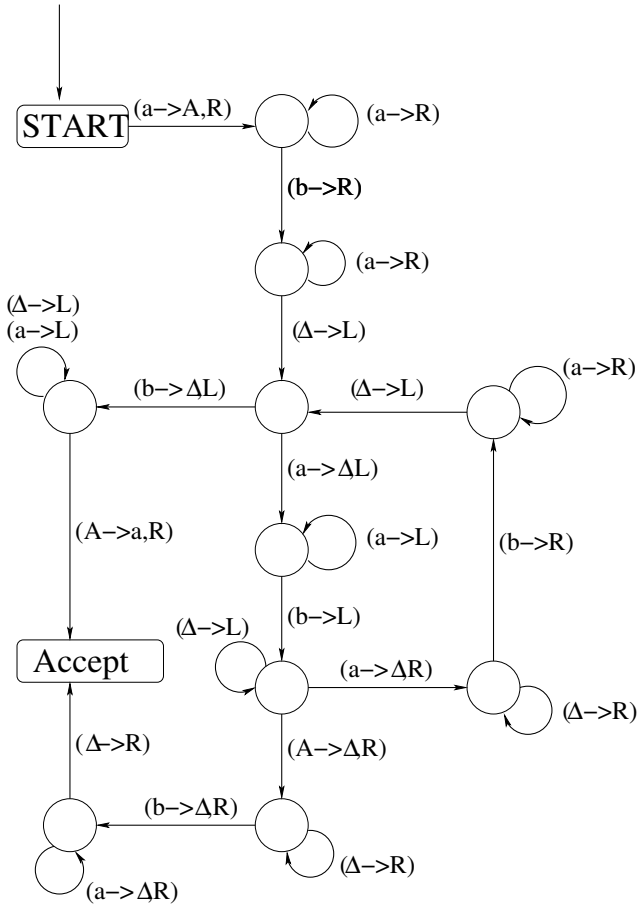
13. Build a Turing Machine that accepts all strings with more a's than b's.

14. The Turing Machine below computes something related to unary subtraction. Run the following inputs on this machine and interpret the results.

i) *aaaba*

ii) *baaa*

iii) *aabaa*



15.

Explain how a Turing machine can be simulated by a Queue Automaton.

(This is superficially like a PDA, except that instead of a stack, it has a queue. A transition $x, y \to z$ means that x is the current input character, y is at the head of the queue and is removed

from it (i.e., served), and z is then appended to the queue. Any of these characters may be replaced by ε, with the usual meaning.)

16. Build a Turing machine that takes as its input a string of the form $x\#b$ where $x \in \{0,1\}^*$ is a binary string and $b \in \{0,1\}$ is a single bit, and:

- if $b = 0$, just outputs x;
- if $b = 1$, flips every bit of x (i.e., changes every 0 to 1, and every 1 to 0).

Before stopping, your TM must erase $\#b$, since the output does not include these characters.

See also Q17.

17. Extend your TM from Q17 so that it now has two bits, $b_0 b_1$, after the `#`, and the decision of whether or not to flip a bit of x depends on the *parity* of the position of the bit on the tape: for bits of x in even-numbered tape cells, we flip or not according to b_0, while for bits of x in odd-numbered tape cells, we flip or not according to b_1. (We imagine the tape cells to be numbered $0, 1, 2, 3, \ldots$, from left to right.)

As in Q16, your TM must erase `#` and everything beyond it before stopping.

You can use states of the TM to remember whether you currently want to flip a bit or not. But remembering whether the next bit to be flipped is on an even-numbered cell or an odd-numbered cell is best done using appropriate marks on the tape, rather than choice of state.

Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 8
Decidability and Mapping Reductions

ASSESSED PREPARATION: Question 1.
You must provide a serious attempt at this entire question at the start of your class.

1.

(a) How could you encode binary trees over our alphabet {a,b}?

A binary tree has a special vertex called a root. Every non-root vertex has one parent vertex, and every vertex has at most two children: at most one left child and at most one right child. A vertex with no children is called a leaf.

Your encoding scheme should be such that each binary tree can be encoded and different binary trees are encoded differently. Subject to this, it should be reasonably efficient, i.e., not produce strings that are unduly long.

You can ignore any numbering or naming of nodes. We are only interested in representing the abstract structure of the tree.

(b) Give the best upper bound you can for the length of the string used, in your scheme, for binary trees on n nodes.

(c) Outline how a Turing machine would recognise the language of full binary trees.

A binary tree is **full** if every non-leaf node has two children.

(d) How many steps would a Turing machine need to take, in the worst case, to decide if a binary tree on n nodes is full or not?

(e) Now express your time bound in terms of the number ℓ of letters in the string that encodes the tree.

2.

Prove that the following problem is decidable.

Input: two regular expressions R_1 and R_2, using an alphabet Σ.
Question: are the languages described by these expressions disjoint?

3.

Let 2SAT2 be the set of Boolean expressions in CNF, with exactly two literals in each clause (as in 2SAT), such that each variable appears at most twice (which could be as identical literals — both negated, or both unnegated — or could be one negated and one normal).

(a) Prove that every member of 2SAT2 is satisfiable.

(b) How would you work out the number of satisfying truth assignments for such an expression?

4. Consider the mapping reduction from 3-COLOURABILITY to SATISFIABILITY that you did in Assignment 1.

Suppose you only use two colours, say Red and White. How would you change your reduction to make use of this restriction? What is the maximum size of any clause created in the reduction? What mapping reduction are you doing now — in particular, from what language and to what language?

5. Consider the following four problems.

FINITE AUTOMATON ACCEPTANCE
INPUT: Finite Automaton M, string x.
QUESTION: Does M accept x?

NFA ACCEPTANCE
INPUT: Nondeterministic Finite Automaton M, string x.
QUESTION: Does M accept x?

PDA ACCEPTANCE
INPUT: Pushdown Automaton M, string x.
QUESTION: Does M accept x?

TM ACCEPTANCE
INPUT: Turing machine M, string x.
QUESTION: Does M accept x?

(a) Outline a mapping reduction from NFA ACCEPTANCE to FINITE AUTOMATON ACCEPTANCE.

(b) Outline a mapping reduction from FINITE AUTOMATON ACCEPTANCE to TM ACCEPTANCE.

(c) Outline a mapping reduction from PDA ACCEPTANCE to FINITE AUTOMATON ACCEPTANCE.

(d) Given that FINITE AUTOMATON ACCEPTANCE is decidable, what does part (a) tell you about NFA ACCEPTANCE?

(e) Given that FINITE AUTOMATON ACCEPTANCE is decidable, what does part (b) tell you about TM ACCEPTANCE?

6. Consider the following two problems.

CFG GENERATION
INPUT: Context-Free Grammar G, string x.
QUESTION: Does G generate x?

CNF CFG GENERATION
INPUT: Context-Free Grammar G in Chomsky Normal Form, string x.
QUESTION: Does G generate x?

(a) Does there exist a mapping reduction from CFG GENERATION to CNF CFG GENERATION? Why or why not?

(b) Does there exist a mapping reduction from CNF CFG GENERATION to CFG GENERATION? Why or why not?

Supplementary exercises

7.

Suppose you have a Boolean expression φ in Conjunctive Normal Form (CNF), with exactly two literals in each clause. You want to determine whether or not it is in 2SAT.

(a) Suppose some variable appears only in unnegated form (i.e., if the variable is x, then it only appears as x, and never as $\neg x$). How can you eliminate this variable from consideration, and simplify your expression as a result?

(b) Suppose some variable appears only in negated form (i.e., always as $\neg x$, never as x). How can you eliminate this variable from consideration, and simplify your expression as a result?

(c) How can you simplify the expression when some variable appears twice in a single clause?

(d) If there is a clause with just one literal, what can you do about its truth value?

(e) Write an algorithm, SIMPLIFY, which takes, as input, a Boolean expression φ in Conjunctive Normal Form (CNF), with **at most** two literals in each clause, and repeatedly applies the actions identified in (a)–(d) for as long as possible. The end result should be *either* rejection, if it is discovered that φ is not satisfiable, *or* it outputs

- a Boolean expression φ' which is satisfiable if and only if φ is satisfiable, and such that in φ': every clause has exactly two literals; every variable appears in both normal and negated form; and that these never appear in the same clause.
- a truth assignment to all variables that appear in φ but not in φ', and which satisfies all clauses of φ that were eliminated in constructing φ'.

(f) Prove, by induction on the number of clauses of φ, that if φ has a clause with just one literal, then SIMPLIFY either rejects φ if it is not satisfiable, or outputs a satisfying truth assignment for it and accepts it.

(g) Put it all together to make a polynomial time algorithm for 2SAT.

(h) Would this approach work for 3SAT? Why, or why not?

Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 9
Undecidability and recursively enumerable languages

Although you may not need to do all the exercises in this Tutorial Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

ASSESSED PREPARATION: Question 3.
You must provide a serious attempt at this entire question at the start of your class.

---

1. Prove that the following problem is undecidable, using a mapping reduction from the Diagonal Halting Problem.

HALT ON EVEN STRINGS
INPUT: Turing machine M.
QUESTION: Does M always halt when the input string has even length?

2.

Recall the languages TM ACCEPTANCE and FINITE AUTOMATON ACCEPTANCE from Exercise Sheet 8, Question 5.

Does there exist a mapping reduction from TM ACCEPTANCE to FINITE AUTOMATON ACCEPTANCE? Why or why not?

3.

A **two-way infinite tape** (TWIT) is a Turing machine tape that is infinite in both directions, i.e., infinite in both left and right directions. The tape cells may be numbered by the integers instead of just by the *positive* integers. When an input string x of n letters is placed on the tape, it is still placed on the first n tape cells in the positive direction, i.e., tape cells 1 to n. Initially, all other tape cells are blank, in both directions.

A **TWIT Turing machine** is just like a normal Turing machine except that it uses a TWIT. There is therefore no possibility of the tape head crashing off the left end of the tape. When the machine accepts, the output string is taken to be the string on the positive portion of the tape, up to but not including the first blank on the positive side, at the time the machine halts.

(a)

Given a normal Turing machine, how would you construct a TWIT Turing machine that simulates it and, in particular, has the same eventual outcome (Accept/Reject/Loop) in all cases and computes the same function?

Now consider the following problem.

TWIT Halting Problem
INPUT: A TWIT Turing machine M, a string x
QUESTION: When M is run with input x, does it eventually halt?

(b)

Give a mapping reduction from the Diagonal Halting Problem to the TWIT Halting Problem.

(c)

Is the TWIT Halting Problem decidable or undecidable? Justify your answer.

(d)

Is the TWIT Halting Problem recursively enumerable? Justify your answer.

4.

For each of the following decision problems, indicate whether or not it is decidable.

You may assume that, when Turing machines are encoded as strings, this is done using the Code-Word Language (CWL).

Decision Problem	your answer (tick **one** box in each row)	
Input: a Java program P (as source code). Question: Does P contain an infinite loop?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: Does P contain the infinite loop `for (int i=0; i>=0; );` ?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: Does P contain recursion?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: If P is run, will some method of P keep calling itself forever?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: is there some input string, of length **at most** 12, for which M halts in at most 144 steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: is there some input string, of length **at least** 12, for which M halts in at most 144 steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Does M halt on an even number of steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Is there a palindrome which, if given as input, causes M to eventually halt?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Is M a palindrome?	☐ Decidable	☐ Undecidable

5. Define the function $f : \mathbb{N} \to \mathbb{N}$ by

$$f(x) = \begin{cases} 3x+1, & \text{if } x \text{ is odd,} \\ x/2, & \text{if } x \text{ is even,} \end{cases}$$

for all $x \in \mathbb{N}$.

We consider *iteration* of f. Given a positive integer x, form the sequence

$$x \longrightarrow f(x) \longrightarrow f(f(x)) \longrightarrow f(f(f(x))) \longrightarrow f(f(f(f(x)))) \longrightarrow \cdots$$

For example, starting with $x = 3$, we have

$$3 \longrightarrow 10 \longrightarrow 5 \longrightarrow 16 \longrightarrow 8 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1 \longrightarrow 4 \longrightarrow \cdots,$$

so it eventually cycles forever through the numbers 4,2,1.

Let COLLATZ be the language of all positive integers for which repeated application of f eventually produces 1. The above example shows that $3 \in$ COLLATZ, and in fact that $\{1, 2, 3, 4, 5, 8, 10, 16\} \subseteq$ COLLATZ.

(a) Using any computational tool of your choice, compute the sequence of iterates of f that begins with $x = 27$, until it starts repeating. (Explain your method briefly, in your submission.)

(b) Prove that COLLATZ is recursively enumerable.

(c) Is your algorithm for (b) a decider for COLLATZ? Why or why not?

(d) Can you prove anything more specific about the class of languages to which COLLATZ belongs? For example, is it decidable?[1]

[1] If you can answer (d), you may become famous! The *Collatz conjecture* (currently unproved) is that COLLATZ = $\mathbb{N}$. In other words, the conjecture asserts that, whatever number you start with, iterating f eventually reaches 1. The conjecture is attributed to Lothar Collatz in 1937 and has attracted a huge amount of human attention and computer time over the decades since then. See, e.g., Jeffrey C. Lagarias, The 3x + 1 problem and its generalizations, *American Mathematical Monthly* **92** (1) 3–23; `https://doi.org/10.1080/00029890.1985.11971528`. Some of the most significant progress in recent years has been made by the Australian mathematician Terence Tao. See: Kevin Hartnett, Mathematician Proves Huge Result on 'Dangerous' Problem, *Quanta Magazine*, 11 December 2019; `https://www.quantamagazine.org/mathematician-terence-tao-and-the-collatz-conjecture-20191211/`. Currently, the conjecture seems far out of reach: it is not even known if COLLATZ is *decidable*.

6.

Prove the following theorem (stated in Lecture 23, slide 19):
A language L is r.e. if and only if there is a decidable two-argument predicate P such that:

$$x \in L \text{ if and only if there exists } y \text{ such that } P(x, y). \quad [2]$$

7.

Give an example of a co-r.e., non-r.e. problem about each of the following:

(a) Turing machines

(b) Context-free grammars

(c) Polynomials and their roots

Supplementary exercises

8. If x is any string, we can *double* it by concatenating it with a copy of itself, forming the string xx. If x is any string of even length, then we can *halve* it by dividing it in half exactly, giving two strings x^L and x^R, being the left and right halves, respectively, of x. (So $x = x^L x^R$.)

A language is *closed under doubling* if every string formed by doubling a string in the language is also in the language. A language is *closed under halving* if halving a string in the language always gives two other strings that are also in the language.

Let L be any nonempty non-universal[3] language that is closed under doubling and halving. Let L^{even} be the set of all strings of even length in L.

(a) Prove that there exists a mapping reduction from L to L^{even}.

(b) Prove that there exists a mapping reduction from L^{even} to L.

(c) Prove that L is undecidable if and only if L^{even} is undecidable.

(d) Determine the time complexity, in big-O notation, of the mapping reductions you found in (a) & (b).

[2] A predicate is said to be *decidable* if its corresponding function — from the set of all its inputs to {True, False}, always correctly indicating whether or not the predicate is True for its input — is computable.

[3] The *universal* language is Σ^*, i.e., the set of all finite strings over the alphabet being used.

9.

A program is *self-reproducing* if it outputs a copy of itself and then stops. Self-reproducibility is a property of programs such as computer viruses and worms.

Prove that the language of self-reproducing programs is undecidable.

Assumptions you may make:

- All programs under discussion are written in some fixed programming language such as Python or Java.
- The Diagonal Halting Problem, for programs in this language, is undecidable.
- There exists a specific self-reproducing program S with the property that you can insert a code chunk C (of a certain type, to be explained shortly) into S, at some special point in S, so as to get another self-reproducing program S_C which executes C before reproducing itself. The inserted code chunk C is required to be "independent" of all the code in S, in that it has uses no variables, functions, methods, etc. that are used in S.

What does this mean for the possibility of an infallible virus-testing program?

10.

For any Turing machine U whose input is a pair (e, x) of strings, we say that U is *outwardly universal* if there is a computable function f which takes any Turing machine and produces a string (intended to be an encoding of the TM, which might be in the Code-Word Language, or might use some other scheme) such that, for all Turing machines M and all strings x,

- U halts for input $(f(M), x)$ if and only if M halts for input x, and
- when they halt, they produce the same output.

Observe that this definition is broader than the usual definition of UTMs! This is not only because we allow any computable encoding scheme, and not just the one used in lectures. It is mainly because we do not require step-by-step simulation, but only that the results of the computations be the same.

Prove that determining whether or not a Turing machine is outwardly universal is undecidable.

Monash University
Faculty of Information Technology
2^{nd} Semester 2023

FIT2014
Exercises 6
Context-Free Grammars and the Pumping Lemmas

Attempt all questions in the main section before your class.

ASSESSED PREPARATION: Question 3.
You must provide a serious attempt at this entire question at the start of your class.

1.

(a) Find a Context-Free Grammar for the language

$$\{\mathbf{a}^n\mathbf{b}^i\mathbf{c}^{2n} : i, n \in \mathbb{N}\}.$$

(b) Prove, by induction on the string length, that every string of the form $\mathbf{a}^n\mathbf{b}^i\mathbf{c}^{2n}$ can be generated by your grammar.

(c) Find a Pushdown Automaton that recognises this language.

2. Let LOL be the language generated by the following Context-Free Grammar.

$$\begin{aligned} S &\to P && (1)\\ P &\to PP && (2)\\ P &\to \texttt{ha}Q && (3)\\ Q &\to Q\texttt{a} && (4)\\ Q &\to \varepsilon && (5) \end{aligned}$$

(a) **Warm-up:**

(i). What is the shortest string in LOL? Give a derivation for it.

(ii). Is the above grammar regular?

(iii). Is LOL a regular language? If so, give a regular expression for it. If not, prove it.

(iv). Is the above grammar in Chomsky Normal Form? If so, why; if not, state why not.

(b) Here is an attempted proof that LOL is infinite. Comment on what is correct and incorrect in this proof.

1. Let x be some string in LOL.
2. The string x must have a derivation, using the above grammar.
3. The only rule in the grammar that does not have a nonterminal symbol on its right-hand side is the last one, (5).
4. So any derivation of x must finish with an application of (5).
5. Therefore the string just before x, in the derivation of x, must have a Q.
6. Therefore the derivation of x has the form

$$S \Longrightarrow \cdots \Longrightarrow x_{\text{left}} Q x_{\text{right}} \overset{(5)}{\Longrightarrow} x_{\text{left}} x_{\text{right}} \quad = \; x,$$

   where x_{left} denotes the portion of x before Q, and x_{right} denotes the portion of x after Q.
7. So, instead of applying rule (5) to this last occurrence of Q, we could apply rule (4) followed by rule (5), giving

$$S \Longrightarrow \cdots \Longrightarrow x_{\text{left}} Q x_{\text{right}} \overset{(4)}{\Longrightarrow} x_{\text{left}} Q \mathsf{a} x_{\text{right}} \overset{(5)}{\Longrightarrow} x_{\text{left}} \mathsf{a} x_{\text{right}}.$$

   This new string is also in LOL and is one letter longer than x. Call it y.
8. We can apply this argument again to y, to get a string in LOL that is one letter longer than y, and then we can get strings that are even longer than that, and so on, indefinitely.
9. So we can generate infinitely many strings. Therefore LOL is infinite.

(c) Prove by induction that, for all $n \geq 2$, the language LOL contains a string of length at least n.

(d) Here is an attempted *proof by contradiction* that LOL is infinite. It uses, in the middle, the proof given in (c). Comment on what is good and bad about this proof.

1. Assume, by way of contradiction, that LOL is finite.
2. ⟨ *Insert here a correct proof, based on (c), that LOL has arbitrarily long strings.* ⟩
3. This contradicts our assumption that LOL is finite.
4. Therefore LOL is infinite.

(e) Construct a correct proof by contradiction that LOL is infinite, using the following approach:

1. Start by assuming LOL is finite, as in (c) step 1.
2. Show that LOL contains a nonempty string. (No need to use the assumed finiteness of LOL just yet.)
3. Among all the nonempty strings in LOL, choose x carefully . . . How should you choose it?
4. Show how to construct a string in LOL that is longer than x.
5. . . . . .
6. Therefore LOL is infinite.

3.

Let `i` be a terminal symbol which we think of as representing an integer.

Let PLUS-MINUS be the language over the five-symbol alphabet {`i`,`+`,`-`,`(`,`)`} defined by the following rules:

- `i` belongs to PLUS-MINUS.
- If x belongs to PLUS-MINUS then so does `(`x`)`.
- If x and y belong to PLUS-MINUS, then so do x`+`y and x`-`y.

(a) Write a Context-Free Grammar for PLUS-MINUS.

(b) Using your grammar, give a derivation for the string `(i+(i-i))-i`.

Now let `d` be a terminal symbol that can stand for any date.

> You can think of `i` and `d` as two tokens used by a hypothetical lexical analyser, where the lexemes for the token `i` would be actual integers and the lexemes for the token `d` would be actual dates (such as 24/9/1966). But you don't need to do anything with lexical analysers in this question.

A **date expression** is an expression using symbols from the six-symbol alphabet {`d`,`i`,`+`,`-`,`(`,`)`}, formed according to the following rules:

- `d` is a date expression
- An integer expression may be added or subtracted after any date expression, giving another date expression (but the date expression must come *before* the integer expression). (This represents adding/subtracting some number of days to/from a date to get another date.)
- One date expression may be subtracted from another date expression, giving an integer expression. (This represents finding the number of days between two dates.)
- Parentheses may be put around any date expression, giving another date expression (representing the same date).
- Integer expressions are constructed using the same rules as for PLUS-MINUS along with the extra rule that the difference between two date expressions is an integer expression.

Examples of valid date expressions: `d`, `d+i`, `d-(i+i)`, `d+d-d`, `((d+i)-(d-d))`, `d-((d-i)-(d+i))`.
Some invalid date expressions: ε, `i`, `i+d`, `i-d`, `d+d`, `d-d`, `((d+i)-(d-i))`.

(c) Write a Context-Free Grammar for the language of date expressions.

(d) Is the language of date expressions regular? Justify your answer.

π.

(a)

Suppose a PDA has the property that every transition *either* pushes *or* pops (not both).[1] Suppose that, if the PDA is in state q with an empty stack, then reads string w, and is then at state r

[1] This is a property of the modified PDA we construct when constructing a CFG from a PDA. See Lecture 14, slides 12–15.

with an empty stack. Let P be any directed path through the PDA from q to r that w can take for this to happen.

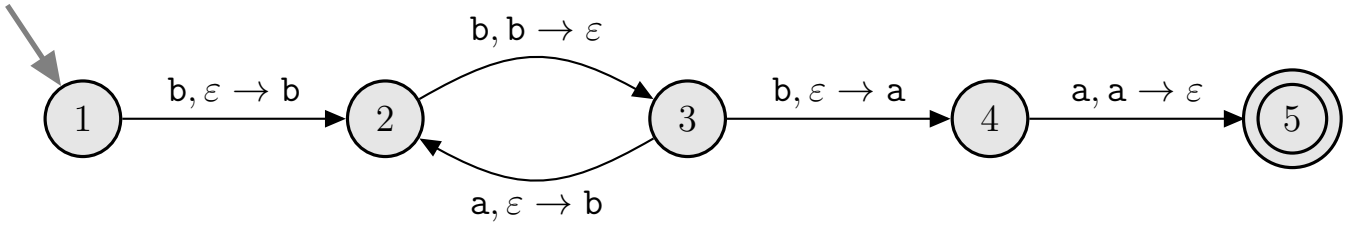
What can you say about the length of P?

(b)

How can we use this observation in deciding which symbols A_{qr} are needed for the process of converting the PDA to a CFG for the same language?

(c)

Consider the following PDA.



Using your observations in (a) and (b), which symbols A_{qr} do you actually need when applying the PDA-to-CFG algorithm to this PDA?

(d)

Convert this PDA to a CFG for the same language.

4. Recall the **Pumping Games** of Exercise Sheet 5. An expansion pack has now been released, called **Double Pumping Games**.

You can play a Double Pumping Game for *any* language. The players are Con and Noni. First, the players are given a language L (which may or may not be context-free). Con moves first, then Noni moves, then Con moves, then Noni moves. The rules for their moves are as follows.

- Con first chooses a number $k \in \mathbb{N}$.
- Then Noni chooses a string $w \in L$ with $|w| > 2^{k-1}$.
- Then Con divides w up into substrings u, v, x, y, z such that $vy \neq \varepsilon$ and $|vxy| \leq 2^k$.
- Then Noni chooses a non-negative integer i.

The result of the game is that:

- if $uv^ixy^iz \in L$, then Con wins;
- if $uv^ixy^iz \notin L$, then Noni wins.

(a) Play the game for (i) PALINDROMES, and (ii) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{a}^n : n \geq 0\}$.

(b) Using quantifiers, write down the assertion that Con has a winning strategy.

(c) Using quantifiers, write down the assertion that Noni has a winning strategy.

(d) What does the Pumping Lemma for CFLs tell us about circumstances in which winning strategies exist?

5. Let G be the following CFG.

$$S \rightarrow AB \quad (6)$$
$$A \rightarrow AC \quad (7)$$
$$B \rightarrow CD \quad (8)$$
$$A \rightarrow \texttt{a} \quad (9)$$
$$C \rightarrow \texttt{b} \quad (10)$$
$$D \rightarrow \texttt{a} \quad (11)$$

Use the Cocke-Younger-Kasami (CYK) algorithm to determine whether or not the string `abbba` is generated by G.

6.
(a) Convert the CFG in Q2 into a Chomsky Normal Form grammar for LOL.
(b) Using this Chomsky Normal Form grammar, apply the CYK algorithm to the string `hahaa`.

7. In the game of Cricket, a *bowler* "bowls" a cricket ball at a *batsman*. Each time this is done, the batsman tries to hit the ball, and might score some number of *runs*, or the bowler might *take the wicket* of the batsman (in which case the batsman has to leave, and is replaced by another batsman). A bowler bowls the ball six times, making up an *over*, and then makes way for another bowler to have their turn.

In cricket statistics, a bowler's activity is summarised by four integers: O, M, R, W, where

- O is the total number of **overs** bowled by that bowler,
- M is the number of **maiden overs** by that bowler: these are those overs in which no run was scored by batsmen,
- R is the number of **runs** scored from the bowling of that bowler,
- W is the number of **wickets** taken by that bowler.

We assume:[2]

- A batsman can score up to six runs from each ball bowled, making up to 36 runs from an over.
- At most one wicket can be taken with each ball, making up to 6 wickets per over.

We also use the fact that the number of maiden overs is clearly at most the total number of overs.

Let CricketBowlingFigures be the language of quadruples (O, M, R, W) of nonnegative integers, each in unary[3], where $O \geq M$, $O - M \leq R \leq 36\,(O - M)$, and $W \leq 6\,O$. We assume that any of these numbers can be arbitrarily large, subject to these constraints. (The statistics can be compiled over entire lifetimes, not just single matches, and some bowlers may be immortal.) For example,

- $(\overbrace{\texttt{111}\cdots\texttt{11}}^{27}, \texttt{1111}, \overbrace{\texttt{111}\cdots\texttt{11}}^{72}, \texttt{11}) \in$ CricketBowlingFigures. It represents the bowling figures (27,4,72,2). [4]
- $(\texttt{11}, \texttt{1}, \overbrace{\texttt{111}\cdots\texttt{11}}^{36}, \texttt{111111111111}) \in$ CricketBowlingFigures. It represents the bowling figures (2,1,36,12).

[2] Cricket experts might notice some simplifying assumptions here ...

[3] In *unary* representation, a number n is represented as a string of n 1s. For example, in unary, the number 5 is represented by 11111, and 0 is represented by the empty string.

[4] Question for cricket tragics: what is the significance of these bowling figures, in the history of Test Cricket?

- $(,,,) \in$ CricketBowlingFigures. It represents the bowling figures (0,0,0,0) of a bowler who does not get a turn at bowling.
- $(\texttt{1}, \texttt{11}, \texttt{111}, \texttt{1111}) \notin$ CricketBowlingFigures. The bowling figures (1,2,3,4) are impossible, since $O < M$.
- $(\texttt{11}, \texttt{1}, \overbrace{\texttt{111}\cdots\texttt{11}}^{36}, \texttt{1111111111111}) \notin$ CricketBowlingFigures. The bowling figures (2,1,36,13) are impossible, since $W > 6\,O$.

(a) Is the language CricketBowlingFigures regular? Prove or disprove.

(b) Is the language CricketBowlingFigures context-free? Prove or disprove.

Supplementary exercises

8. Find Regular Grammars for the following Regular Expressions:

i) $(a \cup b)^*(aa \cup bb)$

ii) $((a \cup b)(a \cup b))^*$

iii) $(aa \cup bb)^*$

iv) $(ab)^*aa(ba)^*$

v) $(ab \cup ba)(aa \cup bb)^*(ab \cup ba)$

9. Describe PDAs for each of the Regular Grammars you found in the previous question.

10. You have seen how to construct a grammar for any regular language, using an NFA for the language. But suppose we wanted to construct a grammar for a regular language *directly from a regular expression*, and *without constructing an NFA first*. How would you do it? (The grammar need not be regular.)

11.

Let k be a fixed positive integer. A *k-limited Pushdown Automaton* is a Pushdown Automaton whose stack can store at most k symbols (regardless of the length of the input string).

(a) Explain how a k-limited Pushdown Automaton can be simulated by a NFA.

(b) Deduce that a language is recognised by a k-limited PDA if and only if it is regular.

12. In Australian Rules Football, a team gains *points* by kicking *goals* and *behinds*. A goal is worth 6 points, and a behind is worth one point. The winner is the team with the greatest number of points at the end of the game.

A team's score is given as a triple of numbers (g, b, p), where g is the number of goals, b is the number of behinds, and p is the number of points. These numbers must be nonnegative and satisfy $6g + b = p$. For example, $(1, 2, 8)$ is a valid score, since $6 \times 1 + 2 = 8$.

Let FootyScore be the language of valid football scores (g, b, p), where each number is represented in unary, and the alphabet consists of 1, the two parentheses, and the comma. For example,

$$\texttt{(1,11,11111111)} \in \text{FootyScore}.$$

In this language, there is no upper limit on the scores.

(a) Is the language FootyScore regular? Prove or disprove.
(b) Is the language FootyScore context-free? Prove or disprove.

13. Prove by induction that the CYK algorithm correctly determines whether or not a given string x is generated by a given context-free grammar G.

More specifically:

1. Prove by induction on n that, for every substring y of x, where $|y| = n$, the algorithm correctly finds all nonterminals in G that can generate y.

2. Explain briefly how, if we know all the nonterminals that can generate x, we can then determine whether x can be generated by G.

The idea of the proof is given in Lecture 16, slide 18. Your task is to avoid the gap in that sketch proof ("Continue, in this way, ...") and produce a correct proof by induction.

14. This question is a sequel to Exercise Sheet 5, Q16.

(a) Review Exercise Sheet 5, Q16, if necessary correcting your attempt at the question.

(b) Use the Pumping Lemma for CFLs to prove that the language $\{\mathbf{a}^{n^2} : n \in \mathbb{N}\}$ is not context-free.

(c) Hence prove that the language of binary string representations of adjacency matrices of graphs is not context-free.

You may assume that the intersection of a context-free language with a regular language is context-free. (Challenge: prove this.) Note that, unlike regular languages, the class of context-free languages is not closed under intersection. (Can you prove this?)

15. A finite sequence $(x_1, x_2, \ldots, x_n)$ of positive integers is *strictly increasing* if, for all i such that $1 \leq i \leq n-1$, we have $x_i < x_{i+1}$.

In this question, sequences are represented as binary numbers separated by `#`. For example, the sequence $(1, 2, 5, 14, 42)$, consisting of the first five Catalan numbers[5], is represented as

`1#10#101#1110#101010`

Let SIS be the language, over the three-letter alphabet {`0`,`1`,`#`}, consisting of all strictly increasing sequences of positive integers, with each sequence represented as described above.

Prove that SIS is not context-free.

16.

A Stock Exchange publishes daily information on the share price of a company as a 5-tuple of numbers

$$(p, c, s, p_{\max}, p_{\min})$$

where

p = the price per share, in cents, when trading closed at the end of the day;

c = the difference between p and the published price for the previous trading day;

s = the number of shares sold during the day;

$p_{\max}$ = the maximum share price for this company during the most recent 52 weeks (including this week);

$p_{\min}$ = the minimum share price for this company during the most recent 52 weeks (including this week).

6

The language SharePriceInfo (SPI), over the seven-symbol alphabet {`0`, `1`, `-`, `+`, `(`, `)`, `,`}, consists of all numerically valid 5-tuples of binary numbers that could (hypothetically, allowing arbitrarily high prices and arbitrary trading volumes) represent information on the share price according to the above description. We require:

[5] The n-th Catalan number counts the number of members of the Dyck language that have n pairs of matching parentheses. The Catalan number sequence has links to a wide range of topics in Computer Science and "occurs in connection with so many recursive algorithms" (Donald E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, Addison-Wesley, Reading, Ma., 1973, p. 296.)

[6] These definitions contain some simplifications for the purposes of this exercise. In practice, Australian share prices are usually quoted in dollars, with cents given after the decimal point. But increases or decreases in share prices are indeed typically given in cents, as here.

- p, $p_{\min}$ and $p_{\max}$ can be any positive integers subject to the requirement that p lies between $p_{\min}$ and $p_{\max}$ inclusive.
- c can be any integer whose value is consistent with the above definitions of all the five numbers. If it is negative, it is prefixed by a minus sign; if it is positive, it is prefixed by a plus sign; and if it is zero, no sign is given.
- s can be any nonnegative integer. Its sole constraint is that if $s = 0$ then $c = 0$ too, since if no shares are traded then the price cannot change (since it is trading in the sharemarket that determines the price).
- Binary numbers must have no leading zeros, except that the number 0 itself is represented as a single `0`.

Is the language SPI context-free? If so, give a Context-Free Grammar for it. If not, prove that it is not, using the Pumping Lemma for Context-Free Languages.

Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 11
Complexity: NP-completeness

ASSESSED PREPARATION: Question 1.
You must provide a serious attempt at this entire question at the start of your class.

---

1. A **singleton clause** in a CNF expression is a clause containing only one literal.

Consider the following decision problem.

SINGLETON CLAUSE SAT (abbreviated SCS)
INPUT: a Boolean expression φ in CNF which has at least one singleton clause.
QUESTION: Is φ satisfiable?

(a) Give a polynomial-time reduction from SAT to SINGLETON CLAUSE SAT.

Your main task here is to work out how you can construct, from any CNF expression φ, a new CNF expression φ' which has a singleton clause, such that φ is satisfiable if and only if φ' is satisfiable.

Remember that φ does not necessarily have a singleton clause. The new expression will need to be related to φ but needs to *always* have a singleton clause.

(b) Prove that, for every CNF expression φ,

$$\varphi \in \text{SAT} \iff \varphi' \in \text{SINGLETON CLAUSE SAT}.$$

(c) What else do you need to prove about your reduction from (a), in order to prove that SAT $\leq_P$ SINGLETON CLAUSE SAT?

(d) What else do you need to prove, in order to prove that SINGLETON CLAUSE SAT is NP-complete?

2. Recall Exercise Sheet 10, Q8, which was about the following problem.

NEARLY SAT
INPUT: Boolean expression φ in CNF.
QUESTION: Is there a truth assignment that satisfies all, or all except one, of the clauses of φ?

In that exercise, we proved that (a) NEARLY SAT belongs to NP, and (b) SAT $\leq_m^p$ NEARLY SAT.

(a) What can we say about NEARLY SAT, given parts (a) and (b) of that exercise?

(b) Suppose we alter the Question in the problem definition so that the truth assignment has to satisfy *all except* k of the clauses, where k is some *fixed* number (i.e., it's fixed in advance, and *not* part of the input). Would the problem still be in NP? Could you still give a polynomial-time reduction from SAT to this new problem? What can we say about the complexity classification of this problem now?

3. This question is from the final exam in 2014.

A **Hamiltonian path** in a graph G is a path that includes every vertex of G. All the vertices on the path must be distinct.

A **Hamiltonian circuit** in a graph G is a circuit that includes every vertex of G. All the vertices on the circuit must be distinct.

The decision problems HAMILTONIAN PATH and HAMILTONIAN CIRCUIT are defined as follows:

HAMILTONIAN PATH
INPUT: Graph G.
QUESTION: Does G have a Hamiltonian path?

HAMILTONIAN CIRCUIT
INPUT: Graph G.
QUESTION: Does G have a Hamiltonian circuit?

(a) Prove that HAMILTONIAN CIRCUIT belongs to NP.

(b) Give a polynomial-time reduction (a.k.a. polynomial transformation) from HAMILTONIAN PATH to HAMILTONIAN CIRCUIT.

(c) Assuming you know that HAMILTONIAN PATH is NP-complete, what can you say about HAMILTONIAN CIRCUIT from your solutions to parts (a) and (b)?

4. Recall:

- GRAPH COLOURING $:=$ $\{\,(G,k) : G$ is a graph, $k \in \mathbb{N}$, and G is k-colourable $\}$. It is NP-complete.
- A graph is **bipartite** if its vertex set can be partitioned into two subsets A and B such that every edge of the graph joins a vertex in A to a vertex in B.

Consider the following variants of GRAPH COLOURING.

(a) BIPARTITE GRAPH COL. $:=$ $\{$ (G,k) : G is a **bipartite** graph, $k \in \mathbb{N}$, and G is k-colourable $\}$

(b) BAD GRAPH COLOURING $:=$ $\{$ (G,k,b) : G is a graph, $k, b \in \mathbb{N}$, and it is possible to assign colours to the vertices of G so that $\leq k$ different colours are used and $\leq b$ edges are **bad** $\}$

(Here, an edge is **bad** if its endpoints get the same colour.)

(c) GRAPH ODD-COLOURING $:=$ $\{$ (G,k) : G is a graph, $k \in \mathbb{N}$, k is **odd**, and G is k-colourable $\}$

(d) ODD GRAPH COLOURING $:=$ $\{$ (G,k) : G is a graph, $|V(G)|$ is **odd**, $k \in \mathbb{N}$, and G is k-colourable $\}$

For each of (a), (b), (c), (d), determine if it is in P or NP-complete, and justify your answers.

Supplementary exercises

5.
(a) Prove that MOSTLY SAT belongs to NP.

MOSTLY SAT
INPUT: Boolean expression φ in CNF.
QUESTION: Is there a truth assignment that satisfies at least three-quarters of the clauses of φ?

(b) Give a polynomial-time reduction from SAT to MOSTLY SAT, and prove that it is such a polynomial-time reduction.

(c) What can we say about MOSTLY SAT, given (a) and (b)?

6.
Consider the following variants of SATISFIABILITY.

(a)	MONOTONE SAT	:=	{ satisfiable Boolean expressions in Conjunctive Normal Form in which no variables are negated }
(b)	ODD SAT	:=	{ satisfiable Boolean expressions in Conjunctive Normal Form in which every clause has an odd number of literals }
(c)	EVEN SAT	:=	{ satisfiable Boolean expressions in Conjunctive Normal Form in which every clause has an even number of literals }
(d)	DNF-SAT	:=	{ satisfiable Boolean expressions in **Disjunctive** Normal Form }

For each of (a), (b), (c), (d), determine if it is in P or NP-complete, and justify your answers.

The following information relates to Questions 9–12.

Before attempting these questions, it is recommended that you revise previous Tutorial exercises about reduction to SATISFIABILITY. These are: Tutorial 1, Q10; Tutorial 5, Q9; Tutorial 6, Q9. If your solutions to any of these were not in Conjunctive Normal Form (CNF), then think about how you would convert them to CNF. See also the notes by FIT2014 tutor Chris Monteith (available under week 3).

Suppose you have a set $T \subseteq A \times A \times A$ of triples, with each element of each triple belonging to some set A of n elements. A *three-dimensional matching* is a subset, $S \subseteq T$, consisting of exactly n triples from T, such that all members of S are disjoint (i.e., if (a, b, c) and (d, e, f) are both in S then $a \neq d$, $b \neq e$ and $c \neq f$).

These requirements ensure that every member of A appears as a *first* member of some triple in S, and also as the *second* member of some triple in S, and also as the *third* member of some triple in S.

For example, suppose that $A = \{1, 2\}$, and $T = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (2, 2, 1)\}$. Then T has the 3D matching $S = \{(1, 1, 2), (2, 2, 1)\}$.

If, instead, $T = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (2, \mathbf{1}, 1)\}$, then T has no 3D matching.

Let 3DM be the language consisting of every set T (consisting of triples) that has a 3D matching.

7. Prove that 3DM belongs to NP.

8.

Give a polynomial-time reduction (a.k.a. polynomial transformation) from 3DM to SATISFIABILITY.

This should be described as an algorithm.

Show that it runs in polynomial-time.

9.

Now suppose that $T \subseteq A \times A \times A \times A$ is a set of *4-tuples*, where again $|A| = n$. A *four-dimensional matching* is a subset, $S \subseteq T$, consisting of exactly n *4-tuples* from T, such that all members of S are disjoint.

Let 4DM be the language consisting of those sets T of 4-tuples that have a 4D matching.

Give a polynomial-time reduction from 3DM to 4DM.

Prove that it is a polynomial-time reduction.

10.

For any fixed positive integer k, we can define the language kDM along similar lines: T is a set of k-tuples $(x_1, x_2, \ldots, x_k)$, where each $x_i \in A$ and $|A| = n$; a *k-dimensional matching* is a set of n disjoint members of T; and kDM is the set of all such T that have a k-dimensional matching.

Suppose you have proved that, for all $k \geq 1$, there is a polynomial-time reduction from kDM to $(k + 1)$DM. Use induction to prove that, for all $k \geq 1$ and all $\ell \geq k$, there is a polynomial-time reduction from kDM to ℓ DM.