The University of Melbourne
School of Computing and Information Systems
COMP30026 Models of Computation

Assignment 2, 2019

Released: 27 September. Deadline: 21 October at 23:00

Purpose

To improve and consolidate your understanding of functions, relations, regular languages, context-free languages, finite-state automata and push-down automata. To develop skills in analysis and formal reasoning about complex concepts, including proof by induction. To practise putting rigorous arguments in writing.

Challenge 1

Give context-free grammars for these languages:

a. The set A of odd-length strings in $\{\mathtt{a}, \mathtt{b}\}^*$ whose first, middle and last symbols are all the same. For example, `b` and `ababa` are in A, but ϵ, `aaaa`, and `abbbb` are not.

b. The set $B = \{\mathtt{a}^i\mathtt{b}\mathtt{a}^j \mid i \neq j\}$. For example, `ab` and `abaaa` are in B, but ϵ, `a`, `b`, and `aabaa` are not.

Challenge 2

Consider the language

$$C = \{\ w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains at least as many } \mathtt{a}\text{s as } \mathtt{b}\text{s}\ \}$$

For example, ϵ, `aaa`, `aba`, and `bbaababaa` are all in C, but `bbb` and `bbaaabb` are not.

a. Construct a 3-state push-down automaton to recognise C. Provide the solution as a transition diagram. Partial marks are given for a C recogniser with more than 3 states.

b. Prove formally that the following context-free grammar G generates C:

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

Hint: Proceed in two steps; prove that every string in $L(G)$ is in C (by structural induction) and prove that every string in C is in $L(G)$ (by induction on the length of the string).

Note: For this challenge we have designed the marking scheme so that you can get away with solving just **one** of 2(a) and 2(b) if you want. Each of 2(a) and 2(b) is marked to a maximum of 2, and if you submit both, your mark for Challenge 2 will be the maximum of your marks for the two sub-challenges.

Challenge 3

Consider the two language-transformer functions *triple* and *snip* defined as follows:

$$\begin{aligned} triple(L) &= \{www \mid w \in L\} \\ snip(L) &= \{xz \mid xyz \in L \text{ and } |x| = |y| = |z|\} \end{aligned}$$

Note that $snip(L)$ discards a string w from L unless w has length $3k$ for some $k \in \mathbb{N}$ (possibly 0), and then the strings whose lengths are multiples of 3 have their middle thirds removed. For example, if $L = \{\mathsf{ab}, \mathsf{bab}, \mathsf{bbb}, \mathsf{babba}, \mathsf{aabbaa}\}$ then $snip(L) = \{\mathsf{bb}, \mathsf{aaaa}\}$.

a. Let R be a regular language. Is $R^3 = R \circ R \circ R$ necessarily regular? Justify your answer.

b. Let R be a regular language. Is $triple(R)$ necessarily regular? Justify your answer.

c. Let R be a regular language. Show that $snip(R)$ is not necessarily regular.

Challenge 4

This challenge is to be answered on Grok. See the last page for submission instructions.

Let $\leq$ be a partial order on the set S. We say that a function $h : S \to S$ is:

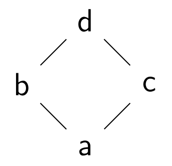
- *idempotent* (`Idem`) iff $\forall x \in S\ (h(h(x)) = h(x))$
- *isotone* (`Iso`) iff $\forall x, y \in S\ (x \leq y \Rightarrow h(x) \leq h(y))$
- *a closure operator* iff it is idempotent and isotone
- *increasing* (`Inc`) iff $\forall x \in S\ (x \leq h(x))$
- *an upper closure operator* (`UCO`) iff it is a closure operator and increasing

Closure operators are important and appear in many different contexts. We have met several; for example, when we take the transitive closure of a relation, we are really applying an upper closure operator to the relation. Here is an example of a upper closure operator on $(\mathcal{P}(\mathbb{Z}), \subseteq)$, that is, the set of integer sets (ordered by the subset ordering):

$$addEvens(S) = S \cup \{n \in \mathbb{Z} \mid n \text{ is even}\}$$

Namely, (1) it is idempotent: *addEvens* always produces a set which includes all even integers, and when applied to such a set, *addEvens* is just the identity function; (2) it is isotone: if $S \subseteq S'$ then $addEvens(S) \subseteq addEvens(S')$; and (3) it is increasing: $addEvens(S)$ is always a superset of S.

Consider $T = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$ and the relation $\leq$ on T, defined by $x \leq y$ iff $x = \mathsf{a} \lor x = y \lor y = \mathsf{d}$. The Hasse diagram for T is shown on the right. There are 256 functions in $T \to T$. The table below the Hasse diagram lists 10 of these functions. For example, f_3 is the mapping $\{\mathsf{a} \mapsto \mathsf{b}, \mathsf{b} \mapsto \mathsf{d}, \mathsf{c} \mapsto \mathsf{a}, \mathsf{d} \mapsto \mathsf{c}\}$.

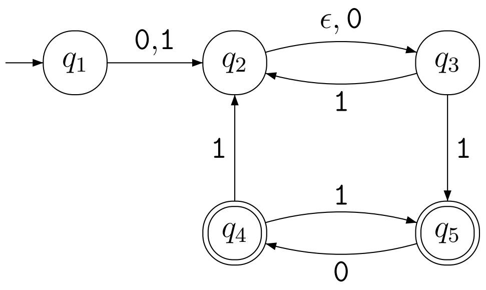


	a	b	c	d
f_0	a	a	a	a
f_1	a	a	a	b
f_2	a	b	c	d
f_3	b	d	a	c
f_4	b	d	d	d
f_5	c	c	c	d
f_6	c	d	c	d
f_7	d	b	b	d
f_8	d	b	c	a
f_9	d	b	c	d

On Grok, define 10 lists `f0` ... `f9`, such that the list `f`i gives those properties (a selection of `Idem`, `Iso`, `Inc`, and `UCO`) that f_i possesses (and only those properties). Grok defines a suitable Haskell type for this. For example, for the function $g : T \to T$ defined by $g(t) = \mathsf{c}$, the list of properties would be `g = [Idem,Iso]`.

Challenge 5

This challenge is to design a regular expression and three DFAs, and submit them via Grok. See the last page for submission instructions.



a. An NFA N is shown on the right. Give a regular expression for $L(N)$, the language recognised by N. The expression should use only the regular operations, that is, union, concatenation, and Kleene star (apart from ϵ, `0`, and `1`).

Now let the alphabet $\Sigma = \{\texttt{1}, \texttt{2}, \texttt{3}\}$. Design DFAs to recognise these three regular languages:

b. The set P of reverse-sorted strings. By "reverse-sorted" we mean: No `1` comes before any `2` or `3`, and no `2` comes before any `3`. For example, ϵ, `2`, and `33111` are reverse-sorted, but `321211` is not.

c. The set Q of strings, the sum of whose elements is divisible by 4. That is,

$$Q = \left\{ \; a_1 a_2 \cdots a_n \; \middle| \; n \geq 0, \sum_{i=1}^{n} a_i = 4k, \text{for some } k \in \mathbb{N} \; \right\}$$

For example, ϵ, `31`, `3333`, and `11111112111` are all in Q (because each string of digits adds up to some multiple of 4), but `2`, `311`, `333`, and `1111111211` are not in Q.

d. The set $R = P \setminus Q$, of strings that are in P but not in Q.

Challenge 6

This challenge is to design a Haskell function that generates certain DFAs. It should be solved on Grok.

In Lecture 14 we went through the exercise of constructing a DFA that could recognise a certain subset of $\{\texttt{0}, \texttt{1}\}^*$, namely the set of binary strings that represent natural numbers that are multiples of 5 (call that language M_5). For example, ϵ, `000`, `101`, `00101101`, and `1100100` were all elements of M_5 and accepted by the DFA, whereas `11`, `0110`, `101110`, and `1100111` were not members, and were all rejected. We now want to generalise that idea.

Write a Haskell function `multiples :: Int -> DFA` so that '`multiples` n' ($n > 0$) produces a DFA (with alphabet $\{\texttt{0}, \texttt{1}\}$) that recognises the language M_n of binary strings representing natural numbers that are multiples of n.

We will use the following marking scheme for Challenge 6:

- +0.5 marks if the function works (produces a correct DFA) for $1 \leq n \leq 6$.
- +0.5 marks if the function produces minimal DFAs for the cases $1 \leq n \leq 6$.
- +0.5 marks if the function works for any n.
- +0.5 marks if the function always produces a minimal DFA, for any n.

We suggest you work in stages:

1. First think through and hand-code the solutions for $1 \leq n \leq 6$.
2. Then think about minimising these 6 DFAs, by hand (no need to program anything, so far).
3. The first stage should help you discover the general pattern of the problem, leading to a general solution for all n.
4. The second stage might help you find the general pattern for minimal DFAs, leading to an enhanced general solution.

Submission and assessment

All challenges should be solved and submitted by students individually. Your solution will count for 12 marks out of 100 for the subject. Each challenge is worth 2 marks. Marks are primarily allocated for correctness, but elegance and how clearly you communicate your thinking will also be taken into account.

Some of the challenges are harder than others, and that is a deliberate design. For example, you may find 3c harder than 3a and 3b, and the last 0.5 marks for Challenge 6 may require more work than the rest of that challenge.

The deadline is 21 October at 23:00. Late submission will be possible, but a late submission penalty will apply: a flagfall of 1 mark, and then 1 mark per 12 hours late.

For challenges 1–3, submit a PDF document via the LMS. This document should be no more than 2 MB in size. If you produce an MS Word document, it must be exported and submitted as PDF, and satisfy the space limit of 2 MB. We also accept *neat* hand-written submissions, but these must be scanned and provided as PDF, and again, they must respect the size limit. If you scan your document, make sure you set the resolution so that the generated document is no more than 2 MB. The assignment submission site on the LMS explains what you can do if you find it hard to satisfy this space requirement.

For challenges 4–6, submit on Grok. The required format for submitted solutions will be clear when you open the Grok modules, but briefly, for Challenge 4 you define ten Haskell lists of type `[FunctionProperty]`, where the inhabitants of `FunctionProperty` are `Idem`, `Iso`, `Inc`, and `UCO`. For Challenge 5 you use the Haskell representations for DFAs and regular expressions introduced in Worksheets 3 and 4, and for Challenge 6, you will need to write some Haskell code. To submit, you need to click "mark". The feedback you will receive at submission time is limited to well-formedness checking; the correctness of your solutions is something you will need to test and be confident about. You can submit as many times as you like. What gets marked is the last submission you made before the deadline.

Once again, for those who want to use the assignment to get some LaTeX practice, the source of this document will be available in the LMS, in the content area where you find the PDF version.

Make sure that you have enough time towards the end of the assignment to present your solutions carefully. A nice presentation is sometimes more time consuming than solving the problems. Start early; note that time you put in early usually turns out more productive than a last-minute effort.

Individual work is expected, but if you get stuck, email Matt or Harald a precise description of the problem, bring up the problem at the lecture, or (our preferred option) use the LMS discussion board. The COMP30026 LMS discussion forum is both useful and appropriate for this; soliciting help from sources other than the above will be considered cheating and will lead to disciplinary action.

Harald and Matt
27 September 2019