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

Assignment 2, 2018

Released: 24 September. Deadline: 14 October at 23:00

Purpose

To improve and consolidate your understanding of regular and context-free languages, finite-state and pushdown automata. To develop skills in analysis and formal reasoning about complex concepts, and to practise writing down formal arguments.

Challenge 1

This challenge is to design eight finite relations and submit them via Grok. Let $D = \{1, 2, 3\}$. We can think of a binary relation on D as a subset of $D \times D$. For example, the identity relation is $\{(1, 1), (2, 2), (3, 3)\}$. There are 9 elements in $D \times D$, so there are $2^9 = 512$ different binary relations on D.

Construct eight of these, $r_0, \ldots, r_7$, satisfying the criteria given in the table below. Each relation should be presented as a Haskell list of pairs, that is, an expression of type `[(Int,Int)]`. For full marks, each relation has to be as small as it can be.

	Reflexive	**Symmetric**	**Transitive**
r_0	no	no	no
r_1	no	no	yes
r_2	no	yes	no
r_3	no	yes	yes
r_4	yes	no	no
r_5	yes	no	yes
r_6	yes	yes	no
r_7	yes	yes	yes

Challenge 2

This challenge is to design three DFAs and a regular expression and submit them via Grok. An expression such as $(01 \cup 1)^* \cap (0^*11)^*$ is not a regular expression, since $\cap$ is not a regular operation. Nevertheless, the expression denotes a regular language.

a. Give a 3-state DFA D_a for $(01 \cup 1)^*$.

b. Give a 4-state DFA D_b for $(0^*11)^*$.

c. In tute exercise 61 we constructed "product" automata for the intersection of languages. Now find a minimal DFA D_c for $(01 \cup 1)^* \cap (0^*11)^*$. You can build the DFA that is the "product" of D_a and D_b, but be aware that the result may not be minimal, so you may have to apply minimisation.

d. Use the NFA-to-regular-expression translation shown in Lecture 12, to turn D_c into a regular expression r for $(01 \cup 1)^* \cap (0^*11)^*$.

Challenge 3

This challenge is to complete, on Grok, some Haskell functions that will allow us to test for membership of languages given as regular expressions. You have probably used regular expression features in your favourite programming language, and this challenge will help you understand how such features are implemented.[1]

Let us call a language A *volatile* iff $\epsilon \in A$. For example, $L(10 \cup (11)^*)$ is volatile, but $L((0 \cup 1)(11)^*)$ is not. Another useful concept is that of a *derivative* of a language with respect to a (finite) string. The derivative of A with respect to s,

$$d(A, s) = \{w \mid sw \in A\}$$

For example, if $A = \{0, 02, 102, 111, 1102\}$ then $d(A, 11) = \{1, 02\}$. For another example, if B is the language given by the regular expression $(00)^*$ then $d(B, 0) = L((00)^*0) = L(0(00)^*)$.

We can extend the definition of "volatile" to regular expressions r: We say that r is volatile iff $\epsilon \in L(r)$. Similarly we extend d so that the function takes, and produces, regular expressions. Let us first handle the case where the string s is of length 1, that is, s is a single symbol x. We define[2]

$$\begin{array}{rcll}
d_1(\epsilon, x) & = & \emptyset & \\
d_1(\emptyset, x) & = & \emptyset & \\
d_1(y, x) & = & \emptyset & \text{if } y \text{ is a symbol and } y \neq x \\
d_1(x, x) & = & \epsilon & \\
d_1(r_1 \cup r_2, x) & = & d_1(r_1, x) \cup d_1(r_2, x) & \\
d_1(r_1\ r_2, x) & = & d_1(r_1, x)\ r_2 \cup d_1(r_2, x) & \text{if } r_1 \text{ is volatile} \\
d_1(r_1\ r_2, x) & = & d_1(r_1, x)\ r_2 & \text{if } r_1 \text{ is not volatile} \\
d_1(r^*, x) & = & d_1(r, x)\ r^* &
\end{array}$$

Now we can define the general case, that is, the derivative of r with respect to an arbitrary string s. We define $d(r, \epsilon) = r$ and, for $k \geq 1$:

$$d(r, x_1x_2 \cdots x_k) = d_1((\ldots d_1(d_1(r, x_1), x_2), \ldots), x_k)$$

You should think about why the above recursive definitions are correct, before you complete their Haskell implementations on Grok. On Grok you will find a Haskell type `RegExp` for regular expressions.

a. Write a Haskell function `volatile :: RegExp -> Bool` that determines whether a regular expression is volatile.

b. Complete a Haskell function `derivative :: RegExp -> String -> RegExp` so that you can use Haskell to calculate derivatives of regular expressions.

c. Write a function `contains :: RegExp -> String -> Bool` which takes a regular expression r and a string w and decides whether $w \in L(r)$. This function must use the functions `volatile` and `derivative`.

[1] The concept of derivative that we use here goes back to J. Brzozowski, "Derivatives of regular expressions", Journal of the ACM **11**(4): 481–494, 1964. The paper shows how derivatives can be used to translate regular expressions directly to equivalent DFAs. If you are looking for an extension to this assignment, go ahead and implement Brzozowski's DFA construction algorithm. You can access the paper through the ACM Digital Library: `https://dl-acm-org.ezp.lib.unimelb.edu.au/dl.cfm?coll=portal&dl=ACM`.

[2] In the Grok skeleton file, d and d_1 are called `derivative` and `derivative1`, respectively.

Challenge 4

Let Σ be a finite non-empty alphabet and let $x \in \Sigma$ be a symbol. For every $w \in \Sigma^*$ and $x \in \Sigma$, let $\mathit{omit}(x, w)$ be the string w with all occurrences of x removed. For example, if $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ and $w = \mathsf{baabca}$ then $\mathit{omit}(\mathsf{a}, w) = \mathsf{bbc}$. Similarly, $\mathit{omit}(\mathsf{b}, \mathsf{bb}) = \epsilon$.

We now "lift" this function to languages: To "drop" a symbol from a language L will mean to "omit" it from every string in L. That is, for $L \subseteq \Sigma^*$, define

$$\mathit{drop}(x, L) = \{\mathit{omit}(x, w) \mid w \in L\}.$$

For example, if $L = \{\mathsf{aa}, \mathsf{baabca}, \mathsf{bbaac}, \mathsf{bbc}\}$ then $\mathit{drop}(\mathsf{a}, L) = \{\epsilon, \mathsf{bbc}\}$ and $\mathit{drop}(\mathsf{b}, L) = \{\mathsf{aa}, \mathsf{aac}, \mathsf{aaca}, \mathsf{c}\}$.

a. Show that if R is a regular language then $\mathit{drop}(\mathsf{a}, R)$ is a regular language.

b. Assuming $\{\mathsf{a}, \mathsf{b}\} \subseteq \Sigma$, provide a language L such that $\mathit{drop}(\mathsf{a}, L)$ is context-free and non-regular, while $\mathit{drop}(\mathsf{b}, L)$ is regular.

Challenge 5

Every regular language can be recognised by a push-down automaton which has only three states, q_I, q_R, and q_A. Show in detail how to systematically translate an arbitrary DFA to a PDA with only three states. Try to answer this question precisely, that is, make use of the formal definitions of DFAs and PDAs. So, given an arbitrary DFA $(Q, \Sigma, \delta, q_0, F)$, define precisely each component of the corresponding PDA $(Q', \Sigma', \Gamma, \delta', q_0', F')$.

Hints: (a) A PDA does not have to have an empty stack in order to accept a string; (b) its stack alphabet Γ can be whatever (finite) set of symbols you choose; and (c) we named the states q_I, q_R, and q_A for a reason.

Challenge 6

Let us fix the alphabet $\Sigma = \{0, 1\}$. Define a *substring* of $w \in \Sigma^*$ to be any string $x \in \Sigma^*$ such that $w = uxv$ for some $u, v \in \Sigma^*$. Now consider the language

$$A = \{w \in \Sigma^* \mid \text{ 001 is not a substring of } w\}$$

For example, ϵ, 01, 1111, 10101, and 0111010101 are in A, but 10001 and 11001011 are not.

Consider the context-free grammar $G = (\{S\}, \Sigma, R, S)$, where the rules R are:

$$\begin{array}{rcl} S & \to & \epsilon \\ & \mid & S\ 0 \\ & \mid & 1\ S \\ & \mid & 0\ 1\ S \end{array}$$

Show that $L(G) = A$.

Hint: Show $L(G) \subseteq A$ using structural induction and show $A \subseteq L(G)$ by induction on the length of strings in A.

Submission and assessment

Some of the problems are harder than others. All should be solved and submitted by students individually. Your solution will count for 12 marks out of 100 for the subject. Each question 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.

The deadline is 14 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 files on Grok. The required format for submitted solutions will be clear when you open the Grok modules, but briefly, for Challenge 1 we want Haskell expressions of type `[(Int,Int)]`, for Challenge 2, we use the Haskell representations for DFAs and regular expressions used in Worksheets 3 and 4, and for Challenge 3, 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 have made before the deadline.

For challenges 4, 5, and 6, submit a PDF document via the LMS. This document should be no more than 1 MB in size. If you produce an MS Word document, it must be exported and submitted as PDF, and satisfy the space limit of 1 MB. We also accept *neat* handwritten 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 1 MB. The assignment submission site on the LMS explains what you can do if you find it hard to satisfy this space requirement. You can submit as many times as you like, unless the deadline has arrived.

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 Søndergaard
23 September 2018