Homework 1

6CCS3CFL - Compilers & Formal Languages

Finley Warman

September 29, 2020

Contents

Question 1

Installing Ammonite REPL - `brew install ammonite-repl` on macOS :)

Question 2

An example of an evil regex in Perl that works on *very* short inputs:
`^((((\w*){1,9})*)*){1,9}\s$`
e.g. on my machine, taking over 30 seconds to check only 6 characters:

```
perl -e '"aaaaaa" =~ /^((((\w*){1,9})*)*){1,9}\s$/'

30.44s user
0.06s system
99% cpu 30.569 total
```

Source: https://twitter.com/jupenur/status/997135478263549952

Question 3

Q: What is meant by the term *language*?
A: A language is simply a set of strings. e.g. *{[], foo, bar}*
Regular languages can be described with regular expressions e.g. `13{2,}7(h[a4]x)?`

Question 4

Q: Give the (inductive) definition for regular expressions. What is the meaning of a regular expression?
A: i. Definition of basic regular expressions:

```
r  ::=  0       nothing
     |  1       empty string / "" / []
     |  c       character
     |  r1+r2   alternative / choice
     |  r1.r2   sequence
     |  r*      star (zero or more)
```

A: ii. The meaning of a regular expression:

$$
\begin{array}{ll}
\mathtt{L(0)} & \equiv \{\} \\
\mathtt{L(1)} & \equiv \{[]\} \\
\mathtt{L(c)} & \equiv \{[c]\} \\
\mathtt{L(r_1 + r_2)} & \equiv \mathtt{L(r_1)} \cup \mathtt{L(r_2)} \\
\mathtt{L(r_1 \cdot r_2)} & \equiv \{\mathtt{s1@s2} \mid \mathtt{s1} \in \mathtt{L(r_1)} \wedge \mathtt{s_2} \in \mathtt{L(r_2)}\} \\
\mathtt{L(r*)} & \equiv \bigcup_{0 \leq n} \mathtt{L(r)}^n \\
\mathtt{L(r)}^0 & \equiv \{[]\} \\
\mathtt{L(r)}^{n+1} & \equiv \mathtt{L(r)@L(r)}^n \\
\text{... (append on sets)} & \{\mathtt{s}_1\mathtt{@s}_2 \mid \mathtt{s}_1 \in \mathtt{L(r)} \wedge \mathtt{s}_2 \in \mathtt{L(r)}^n\}
\end{array}
$$

where `L(0)` denotes the empty language, and `L(1)` denotes the language containing only the empty string.

Question 5

Q: Define the operation of *concatenating* two **sets** of strings. (Given the string concatenation operator `@`)
A: i. Set-builder definition of language concatenation:

$$L_1@L_2 \equiv \{s_1@s_2 \mid s_1 \in L_1 \wedge s_2 \in L_2\}$$

i.e. the concatenation (@) of languages L_1 and L_2 consists of all possible strings formed by concatenating one string (s_1) from L_1 with another (s_2) from L_2.

Example: `{hello, []}@{world, []} = {hello, world, helloworld}`

A: ii. `A@{}` is by this definition therefore `{}` (the empty set)

A: iii. `A@B` is not in general equivalent to `B@A` - concatenation is not commutative. Counterexample of commutativity:
`if A={foo}, B={bar} then A@B={foobar} and B@A={barfoo}, therefore A@B != B@A`

For language concatenation, `{[]}` acts as the identity element, as does the empty string `[]` for string concatenation. The empty set `{}` acts as a sort of zero-element.

Question 6

Q: `|A| = 4` and `|B| = 7`, $[\,] \notin A \wedge [\,] \notin B$. How many strings in `A@B`?
A:
e.g. simple case, at most 28: `{A,B,C,D}@{1,2,3,4,5,6,7}`
= `{A1,...,A7, B1,...,B7, C1,...,C7, D1,...,D7}`
⇒ `|A@B| = 4x7 = 28`
However, there *can* be fewer if there are duplicates in the resulting string concatenations.
e.g. `{1, 11, 1111, 1111}@{1, 11, 1111, 11111, 111111, 1111111, 11111111}` - the resulting set has only length **9**.

Question 7

Q: How is the power of a language defined?
A: The n^{th} power of a language, is the concatenation of itself with its previous power: (where the base case $n = 0$ is the language $L(1) = \{[]\}$, containing only the empty string)

$\texttt{A}^{\texttt{0}} \equiv \{[\,]\}$
$\texttt{A}^{\texttt{n+1}} \equiv \texttt{A@A}^{\texttt{n}}$

e.g. $A^4 = A@A@A@A$ (@{[ ]}), $A^1 = A$, $A^0 = \{[\,]\}$
so $\{a\}^4 = \{aaaa\}$ and $\{a, b\}^2 = \{aa, ab, ba, bb\}$

Question 8

Q: Let A = {[a], [b], [c], [d]}
A: i. How many strings in $\{A\}^4$?

```
|A@A|      = 4^2          = 16
|A@A@A|    = 4^2x4 = 4^3  = 64
|A@A@A@A|  = 4^3x4 = 4^4  = 256
           => {A}^4 has 256 strings
```

A: ii. If A = {[a], [b], [c], []}, how many strings in $\{A\}^4$?

```
|A@A|      = 9 + 4 = 13
|A@A@A|    = 40
|A@A@A@A|  = 121
           => {A}^4 has 121 strings
```

Question 9

Q: i. How many basic regular expressions to match `abcd`?
Q: ii. How many if excluding `1` and `0`?
Q: iii. How many if not allowed to contain stars (`*`)?
Q: iiii. How many if not allowed to contain choices (`_+_`)?

A: i. A boundless number, since you can append **0** or **1** indefinitely to a valid regex.
A: ii. Still any number: `(z* . (a.b.c.d)` or `((x.z)* . (a.b.c.d)`, etc.
A: iii. Still any number: `(a.b.c.d) + (a.b.c.d) + ...`
A: iiii. 5: (Remember, regular expressions are trees):

- `(a.(b.c)).d`
- `(a.b).(c.d)`
- `((a.b).c).d`
- `a.((b.c).d)`
- `a.(b.(c.d))`

N.B. In this case, this is equivalent to the *Catalan number* for `n=3`, i.e.

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1}\binom{2n}{n} \text{ for } n \geq 0$$

C_n is the number of ways to of associating n applications of a binary operator (aka. the parenthesisation of $n+1$ values connected by an associative binary operator).

This is because successive applications of a binary operator can be represented in terms of a full binary tree, so this value is the number of distinct ways to arrange such a tree with $n+1$ (unlabelled) leaves
i.e. C_n is the number of non-isomorphic ordered trees with $n+1$ vertices
The first few Catalan numbers are

$$1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862$$

Question 10

Q: i. When are two regular expressions equivalent?
Q: ii. Example(s) of non-obviously identical expressions.

A: i. Two regular expressions are equivalent when their languages are the same, i.e. the match the same set of strings.

An approach for determining equivalence between two regular expressions is to convert them both to deterministic finite automata (DFA). (Possibly by first converting to an NFA, then converting this to a DFA).
If these DFA are minimised, then it is simple to determine if they accept the same language as each minimal DFA is unique, and so both should be isomorphic if the languages are the same.

A: ii. The following are equivalent, though it is not immediately obvious:

- (a.a.b + b.a.a.b) + ((a + b)* . a.a.b)
- (a + b)* . ((b.a.a.b + a.a.b + b.b.b.b.b.b.a.a.a.a.a.a.b + b.b.b.b.b.a.a.b) + ((a.a + a.a.a + b.a.a) . b))

Both only matching strings containing 'a' and 'b', ending with the substring 'aab'.

A general example: `r* == 1 + r · r*` for any regular expression **r**.
Since `r*` matches zero-or-more of **r**, this is equivalent to matching the empty string **1**, OR one-or-more of **r**.

Question 11

Q: What is meant by *evil regular expressions* and *catastrophic backtracking*?
A: 'Evil' regular expressions are expressions that have an exponential time complexity, meaning for sufficiently large strings (which may be smaller than expected!), they become prohibitively expensive to compute a match. They can be accidental, or malicious to launch an intentional ReDOS (Regex Denial-Of-Service) attack.

The exponential time complexity is due to *catastrophic backtracking*; this can occur when expressions contains nested quantifiers, and one of the inner expressions is a match which is also the suffix of another match. If this does not reach a complete match early, the regex engine will exhaustively search all the possible matching substrings for each combination of quantifier in order to find a match.

Question 12

Q: Given the regular expression $(a + b)^* \cdot b \cdot (a + b)^*$, which are equivalent?
A: "any amount of a or b, followed by b, followed by any amount of a or b"
= any non-empty string containing 'b', consisting only of 'a' and 'b'.

1) $(ab + bb)^* \cdot (a + b)^*$ - NO (matches empty string)
2) $(a + b)^* \cdot (ba + bb + b) \cdot (a + b)^*$ - YES

3) $(a+b)^* \cdot (a+b) \cdot (a+b)^*$ - NO (matches 'a')

Question 13

Q: Feedback!
A: The online lecture format is great - easily digestible, clear, and to-the-point. The amount of available material and resources on KEATS is also great.
Initially looking at the lecture slides, I found these hard to follow alone, without context. However, the handouts supplement this well (and this is not an issue when reading along while watching the lecture, as intended!).
I'm looking forward to the rest of this module!