Tutorial 2

Computer Language Processing (COMP 4901U)

Monday September 20

- Recap on regular expression marching
- Exercise solving (Zoom breakout rooms available)

Recap: Words

Let A be an alphabet $\{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots\}$

We define words of length n, denoted A^n, as follows:

$A^0 = \{\varepsilon\}$ (only one word of length zero, always denoted ε)

For $n > 0$, $A^n = \{\, \mathrm{a}w \mid w \in A^{n-1} \,\}$

Notation '$\mathrm{a}\varepsilon$' abbreviated to 'a'.

Concatenation: $u \cdot v$, also written uv, is associative.
We can decompose words arbitrarily, as in $w = u\mathrm{a}$ (i.e., $w = u \cdot \mathrm{a}\varepsilon$) if $|w| > 0$

Set of all words: $A^* = \bigcup_{n \geq 0} A^n$

which means: $w \in A^*$ if and only iff there exists n such that $w \in A^n$.

Recap: Languages

A *language* over alphabet A is a set $L \subseteq A^*$.

Examples for $A = \{0,1\}$:

- empty language $\varnothing$;
- finite languages like $L = \{1, 10, 1001\}$;
- language L described by a characteristic function f, i.e., $L = \{\, w \in A^* \mid f(w) \,\}$

Operations on languages:

- Set operations: union ($\cup$), intersection ($\cap$), difference ($\setminus$), etc.
- Language concatenation: $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$, also written $L_1 L_2$
- Language exponentiation: $L^0 = \{\varepsilon\}$ and $L^{n+1} = L \cdot L^n$

Recap: Regular Expressions

Syntax of regular expressions: $e \ ::= \ \varnothing \ | \ \varepsilon \ | \ c \ | \ (e_1 \,|\, e_2) \ | \ e_1 e_2 \ | \ e^*$

The *semantics* of regular expression e is the language $L(e)$ – also written L_e – where:

$$
\begin{aligned}
L(\varnothing) &= \varnothing \\
L(\varepsilon) &= \{\varepsilon\} \\
L(c) &= \{c\} && (c \in A) \\
L(e_1 \,|\, e_2) &= L(e_1) \cup L(e_2) \\
L(e_1 e_2) &= L(e_1) \cdot L(e_2) \\
L(e^*) &= L(e)^*
\end{aligned}
$$

Example: $\mathit{letter}(\mathit{letter} \,|\, \mathit{digit})^*$ where $\mathit{letter} = \mathrm{a} \,|\, \mathrm{b} \,|\, \mathrm{c} \,|\, \ldots$ and $\mathit{digit} = 0 \,|\, 1 \,|\, 2 \ldots |\, 9$

Recap: Closed Operations on Regular Expressions

Regular languages/expressions are closed under these operations:

- Shorthands for finite languages, such as $[\mathrm{a..z}] = \mathrm{a} \,|\, \mathrm{b} \,|\, ... \,|\, \mathrm{z}$
- Optionality $e^? = e \,|\, \varepsilon$
- Repeating at least once $e^+ = ee^*$
- Other repetitions $e^{k..*} = e^k e^*$ and $e^{p..q} = e^p(e^?)^{q-p}$
- Complementation: $!e$, denoting $A^* \setminus L(e)$ — Q: how to translate?
- Intersection $e_1 \,\&\, e_2 = !(!e \,|!e)$, denoting $L(e_1) \cap L(e_2)$

Recap: Nullable

Formal definition of *nullable*: $\quad nullable(e) \;=\; \varepsilon \in L(e)$

Algorithm for computing *nullable*:

$$
\begin{aligned}
nullable(\emptyset) &= false \\
nullable(\varepsilon) &= true \\
nullable(a) &= false \\
nullable(e_1|e_2) &= nullable(e_1) \vee nullable(e_2) \\
nullable(e^*) &= true \\
nullable(e_1 e_2) &= nullable(e_1) \wedge nullable(e_2)
\end{aligned}
$$

Recap: Lexers

Definition of a *lexer* (aka *tokenizer*):

an ordered set of n labelled token definitions

$$\langle\, Token_1 := e_1\,;\; Token_2 := e_2\,;\; \ldots\,;\; Token_n := e_n \,\rangle$$

Disambiguation rules:

- Longest-match
- First-match

Recap: Regular Expression Derivative

Definition (Brzozowski Derivative)

The *derivative* of regexp e with respect to letter c, written $\delta^c(e)$, is defined as:

$$L(\delta^c(e)) = \{ w \mid cw \in L(e) \}$$

Recap: Regular Expression Derivative

Definition (Brzozowski Derivative)

The *derivative* of regexp e with respect to letter c, written $\delta^c(e)$, is defined as:

$$L(\delta^c(e)) = \{ w \mid cw \in L(e) \}$$

A few derivative examples:

$$\delta^{\mathsf{a}}(\mathsf{aaa}) = \mathsf{aa}$$

$$\delta^{\mathsf{a}}(\mathsf{ab} \mid \mathsf{ac} \mid \mathsf{da}) = \mathsf{b} \mid \mathsf{c}$$

$$\delta^{\mathsf{a}}((\mathsf{ab})^*) = \mathsf{b}(\mathsf{ab})^*$$

$$\delta^{\mathsf{a}}((\mathsf{ab}|\mathsf{c})^*\mathsf{ad}) = \mathsf{b}(\mathsf{ab}|\mathsf{c})^*\mathsf{ad} \mid \mathsf{d}$$

Recap: Regular Expression Derivative

Derivative of a regexp e with respect to letter c, written $\delta^c(e)$, can be computed as:

$$
\begin{aligned}
\delta^c(\varnothing) &= \varnothing \\
\delta^c(\varepsilon) &= \varnothing \\
\delta^c(d) &= \begin{cases} \varepsilon & \textbf{if } d = c \\ \varnothing & \textbf{if } d \neq c \end{cases} \\
\delta^c(e_1 \,|\, e_2) &= \delta^c(e_1) \,|\, \delta^c(e_2) \\
\delta^c(e_1\, e_2) &= \begin{cases} \delta^c(e_1)\, e_2 \,|\, \delta^c(e_2) & \textbf{if } \textit{nullable}(e_1) \\ \delta^c(e_1)\, e_2 & \textbf{otherwise} \end{cases} \\
\delta^c(e_1^*) &= \delta^c(e_1)\, e_1^*
\end{aligned}
$$

Exercises

Exercise 1

Question:

Find a regular expression that generates all alternating sequences of 0 and 1 with arbitrary length (including lengths zero, one, two, etc.).
For example, the alternating sequences of length one are 0 and 1, length two are 01 and 10, length three are 010 and 101.
Note that no two adjacent character can be the same in an alternating sequence.

Exercise 1

Question:

Find a regular expression that generates all alternating sequences of 0 and 1 with arbitrary length (including lengths zero, one, two, etc.).
For example, the alternating sequences of length one are 0 and 1, length two are 01 and 10, length three are 010 and 101.
Note that no two adjacent character can be the same in an alternating sequence.

Solution:

$$(10)^*1^? \mid (01)^*0^?$$

or, simpler:

$$0^?(10)^*1^?$$

Exercise 2 – Integer Literals of Scala

Integer literals are in three forms in Scala: decimal, hexadecimal and octal. The compiler discriminates different classes from their beginning.

- Decimal integers are started with a non-zero digit.
- Hexadecimal numbers begin with 0x or 0X and may contain the digits from 0 through 9 as well as upper or lowercase digits from A to F.
- If the integer number starts with zero, it is in octal representation so it can contain only digits 0 through 7.
- l or L at the end of the literal shows the number is Long.

Exercise 2 – Integer Literals of Scala

Integer literals are in three forms in Scala: decimal, hexadecimal and octal. The compiler discriminates different classes from their beginning.

- Decimal integers are started with a non-zero digit.
- Hexadecimal numbers begin with 0x or 0X and may contain the digits from 0 through 9 as well as upper or lowercase digits from A to F.
- If the integer number starts with zero, it is in octal representation so it can contain only digits 0 through 7.
- l or L at the end of the literal shows the number is Long.

Question: Write the corresponding regular expression.

Exercise 2 – Integer Literals of Scala

Integer literals are in three forms in Scala: decimal, hexadecimal and octal. The compiler discriminates different classes from their beginning.

- Decimal integers are started with a non-zero digit.
- Hexadecimal numbers begin with 0x or 0X and may contain the digits from 0 through 9 as well as upper or lowercase digits from A to F.
- If the integer number starts with zero, it is in octal representation so it can contain only digits 0 through 7.
- l or L at the end of the literal shows the number is Long.

Question: Write the corresponding regular expression.

Solution: $((1..9)\text{digit}^* \mid 0(x \mid X)(\text{digit} \mid A..F)^* \mid 0(0..7)^*)(l \mid L)^?$
where $\text{digit} = 0..9 = 0 \mid 1 \mid ... \mid 9$

Exercise 2

Consider this lexer from last week where $A = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$:

$$\left\langle \mathrm{T1} : \mathrm{a(ab)}^*, \mathrm{T2} : \mathrm{b}^*\mathrm{(ac)}^*, \mathrm{T3} : \mathrm{cba}, \mathrm{T4} : cc^* \right\rangle$$

Exercise 2

Consider this lexer from last week where $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$:

$$\langle \mathsf{T1} : \mathsf{a}(\mathsf{ab})^*, \mathsf{T2} : \mathsf{b}^*(\mathsf{ac})^*, \mathsf{T3} : \mathsf{cba}, \mathsf{T4} : cc^* \rangle$$

Question: Construct the successive derivatives of each token for the given sequences:

- c
- ac
- cb
- bacacc

Exercise 2 – Solutions

$$\left\langle \mathsf{T1} : \mathsf{a}(\mathsf{ab})^*, \mathsf{T2} : \mathsf{b}^*(\mathsf{ac})^*, \mathsf{T3} : \mathsf{cba}, \mathsf{T4} : cc^* \right\rangle$$

$\delta^{\mathsf{c}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_2) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_3) = \mathsf{ba} \quad \delta^{\mathsf{c}}(\mathsf{T}_4) = \mathsf{c}^*$

Exercise 2 – Solutions

$$\left\langle \mathsf{T1} : \mathsf{a}(\mathsf{ab})^*, \mathsf{T2} : \mathsf{b}^*(\mathsf{ac})^*, \mathsf{T3} : \mathsf{cba}, \mathsf{T4} : cc^* \right\rangle$$

$\delta^{\mathsf{c}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_2) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_3) = \mathsf{ba} \quad \delta^{\mathsf{c}}(\mathsf{T}_4) = \mathsf{c}^*$

$\delta^{\mathsf{a}}(\mathsf{T}_1) = (\mathsf{ab})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_2) = \mathsf{c}(\mathsf{ac})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_3) = \varnothing \quad \delta^{\mathsf{a}}(\mathsf{T}_4) = \varnothing$

Exercise 2 – Solutions

$$\left\langle \mathsf{T1: a(ab)^*, T2: b^*(ac)^*, T3: cba, T4:} \, cc^* \right\rangle$$

$$\delta^{\mathsf{c}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_2) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_3) = \mathsf{ba} \quad \delta^{\mathsf{c}}(\mathsf{T}_4) = \mathsf{c}^*$$

$$\delta^{\mathsf{a}}(\mathsf{T}_1) = (\mathsf{ab})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_2) = \mathsf{c}(\mathsf{ac})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_3) = \varnothing \quad \delta^{\mathsf{a}}(\mathsf{T}_4) = \varnothing$$
$$\delta^{\mathsf{ac}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{ac}}(\mathsf{T}_2) = (\mathsf{ac})^* \quad \delta^{\mathsf{ac}}(\mathsf{T}_3) = \varnothing \quad \delta^{\mathsf{ac}}(\mathsf{T}_4) = \varnothing$$

Exercise 2 – Solutions

$$\left\langle \mathsf{T1}:\mathsf{a(ab)}^*, \mathsf{T2}:\mathsf{b}^*\mathsf{(ac)}^*, \mathsf{T3}:\mathsf{cba}, \mathsf{T4}:cc^* \right\rangle$$

$\delta^{\mathsf{c}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_2) = \varnothing \quad \delta^{\mathsf{c}}(\mathsf{T}_3) = \mathsf{ba} \quad \delta^{\mathsf{c}}(\mathsf{T}_4) = \mathsf{c}^*$

$\delta^{\mathsf{a}}(\mathsf{T}_1) = (\mathsf{ab})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_2) = \mathsf{c}(\mathsf{ac})^* \quad \delta^{\mathsf{a}}(\mathsf{T}_3) = \varnothing \quad \delta^{\mathsf{a}}(\mathsf{T}_4) = \varnothing$
$\delta^{\mathsf{ac}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{ac}}(\mathsf{T}_2) = (\mathsf{ac})^* \quad \delta^{\mathsf{ac}}(\mathsf{T}_3) = \varnothing \quad \delta^{\mathsf{ac}}(\mathsf{T}_4) = \varnothing$

$\delta^{\mathsf{cb}}(\mathsf{T}_1) = \varnothing \quad \delta^{\mathsf{cb}}(\mathsf{T}_2) = \varnothing \quad \delta^{\mathsf{cb}}(\mathsf{T}_3) = \mathsf{a} \quad \delta^{\mathsf{cb}}(\mathsf{T}_4) = \varnothing$

Exercise 2 – Solutions

$$\langle \mathrm{T1}: \mathrm{a(ab)^*}, \mathrm{T2}: \mathrm{b^*(ac)^*}, \mathrm{T3}: \mathrm{cba}, \mathrm{T4}: cc^* \rangle$$

$\delta^{c}(T_1) = \emptyset \quad \delta^{c}(T_2) = \emptyset \quad \delta^{c}(T_3) = \mathrm{ba} \quad \delta^{c}(T_4) = \mathrm{c}^*$

$\delta^{a}(T_1) = (\mathrm{ab})^* \quad \delta^{a}(T_2) = \mathrm{c(ac)}^* \quad \delta^{a}(T_3) = \emptyset \quad \delta^{a}(T_4) = \emptyset$
$\delta^{ac}(T_1) = \emptyset \quad \delta^{ac}(T_2) = (\mathrm{ac})^* \quad \delta^{ac}(T_3) = \emptyset \quad \delta^{ac}(T_4) = \emptyset$

$\delta^{cb}(T_1) = \emptyset \quad \delta^{cb}(T_2) = \emptyset \quad \delta^{cb}(T_3) = \mathrm{a} \quad \delta^{cb}(T_4) = \emptyset$

...
$\delta^{bacac}(T_1) = \emptyset \quad \delta^{bacac}(T_2) = (\mathrm{ac})^* \quad \delta^{bacac}(T_3) = \emptyset \quad \delta^{bacac}(T_4) = \emptyset$

Exercise 2 – Solutions

$$\left\langle \mathsf{T1} : \mathsf{a(ab)}^*, \mathsf{T2} : \mathsf{b}^*\mathsf{(ac)}^*, \mathsf{T3} : \mathsf{cba}, \mathsf{T4} : cc^* \right\rangle$$

$\delta^{c}(T_1) = \varnothing \quad \delta^{c}(T_2) = \varnothing \quad \delta^{c}(T_3) = ba \quad \delta^{c}(T_4) = c^*$

$\delta^{a}(T_1) = (ab)^* \quad \delta^{a}(T_2) = c(ac)^* \quad \delta^{a}(T_3) = \varnothing \quad \delta^{a}(T_4) = \varnothing$
$\delta^{ac}(T_1) = \varnothing \quad \delta^{ac}(T_2) = (ac)^* \quad \delta^{ac}(T_3) = \varnothing \quad \delta^{ac}(T_4) = \varnothing$

$\delta^{cb}(T_1) = \varnothing \quad \delta^{cb}(T_2) = \varnothing \quad \delta^{cb}(T_3) = a \quad \delta^{cb}(T_4) = \varnothing$

...
$\delta^{bacac}(T_1) = \varnothing \quad \delta^{bacac}(T_2) = (ac)^* \quad \delta^{bacac}(T_3) = \varnothing \quad \delta^{bacac}(T_4) = \varnothing$
$\delta^{bacacc}(T_1) = \varnothing \quad \delta^{bacacc}(T_2) = \varnothing \quad \delta^{bacacc}(T_3) = \varnothing \quad \delta^{bacacc}(T_4) = \varnothing$

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?
  E.g. `()`, `((`, `))`, `)()))(`, ...

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?

  E.g. `()`, `((`, `))`, `)()))(`, ...

  ⇒ **YES**

- Sequence of balanced parentheses

  `( ( () ) ())` – balanced

  `( ) ) ( ( )` – not balanced

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?

  E.g. ( ), ((,)),)()))(, ...

  ⇒ **YES**
- Sequence of balanced parentheses

  ((( )) ( )) – balanced

  ( )) (( ) – not balanced

  ⇒ **NO**
- As many digits before as after decimal point?

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?

  E.g. `()`, `((`, `))`, `)()))(`, ...

  $\Rightarrow$ **YES**

- Sequence of balanced parentheses

  ((( )) ( )) – balanced

  ( )) (( ) – not balanced

  $\Rightarrow$ **NO**

- As many digits before as after decimal point? $\Rightarrow$ **NO**

- Comments from // until LF

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?
  E.g. `()`, `((`, `))`, `)()))(`, ...
  $\Rightarrow$ **YES**
- Sequence of balanced parentheses
  `( ( ( ) ) ( ) )` – balanced
  `( ) ) ( ( )` – not balanced
  $\Rightarrow$ **NO**
- As many digits before as after decimal point? $\Rightarrow$ **NO**
- Comments from // until LF $\Rightarrow$ **YES**
- Nested comments like `/* ... /* */ ... */`

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?
  E.g. `()`, `((`, `))`, `)()))(`, ...
  $\Rightarrow$ **YES**
- Sequence of balanced parentheses
  `( ( ( ) ) ( ) )` – balanced
  `( ) ) ( ( )` – not balanced
  $\Rightarrow$ **NO**
- As many digits before as after decimal point? $\Rightarrow$ **NO**
- Comments from `//` until `LF` $\Rightarrow$ **YES**
- Nested comments like `/* ... /* */ ... */` $\Rightarrow$ **NO**

Exercise 3 – On the expressiveness of regular expressions

For which of the following languages can you find an automaton or regular expression?

- Sequence of open or closed parentheses of even length?
  E.g. `()`, `((`, `))`, `)()))(`, ...
  $\Rightarrow$ **YES**
- Sequence of balanced parentheses
  `( ( ( ) ) ( ) )` – balanced
  `( ) ) ( ( )` – not balanced
  $\Rightarrow$ **NO**
- As many digits before as after decimal point? $\Rightarrow$ **NO**
- Comments from `//` until LF $\Rightarrow$ **YES**
- Nested comments like `/* ... /* */ ... */` $\Rightarrow$ **NO**

Regular expressions cannot "count."

Exercise 4

Let *init* be a function that returns all the symbols of a string except the last one.
For example $init(mama) = mam$
init is undefined for an empty string.
If $L_1 \subseteq A^*$, then $INIT(L_1)$ applies the function to all non-empty words in L_1, ignoring ε if it is in L_1:

$$INIT(L_1) = \{\, v \in A^* \mid \exists \mathrm{a} \in A.\ v\mathrm{a} \in L_1 \,\}$$

Exercise 4

Let *init* be a function that returns all the symbols of a string except the last one.
For example $init(mama) = mam$
init is undefined for an empty string.
If $L_1 \subseteq A^*$, then $INIT(L_1)$ applies the function to all non-empty words in L_1, ignoring ε if it is in L_1:

$$INIT(L_1) = \{ v \in A^* \mid \exists \mathrm{a} \in A.\ v\mathrm{a} \in L_1 \}$$

Question: Give an algorithm that, given a regular expression r for L_1, computes a regular expression $rinit(r)$ for language $INIT(L_1)$

Exercise 4

Let *init* be a function that returns all the symbols of a string except the last one.
For example $init(mama) = mam$
init is undefined for an empty string.
If $L_1 \subseteq A^*$, then $INIT(L_1)$ applies the function to all non-empty words in L_1, ignoring ε if it is in L_1:

$$INIT(L_1) = \{ v \in A^* \mid \exists \mathrm{a} \in A.\ v\mathrm{a} \in L_1 \}$$

Question: Give an algorithm that, given a regular expression r for L_1, computes a regular expression $rinit(r)$ for language $INIT(L_1)$

Solution: Just follow the same approach as for defining derivatives, using the other direction for inductive decomposition of words.