Introduction to Language Theory and Compilation: Exercises

Session 2: Regular expressions

Regular expressions (RE)

- Finite automata are an equivalent formalism to regular languages (for each regular language, there exists at least one FA that recognizes it, and each FA recognizes a regular language).
- Regular expressions are another formalism defined inductively just as regular languages.
- It can be proven that regular expressions and regular languages are equivalent (see lecture notes).

Regular expressions (RE) (ctd.)

Base cases:

RE	language
ϕ	$\emptyset$
ε	$\{\varepsilon\}$
$a \quad (\forall a \in \Sigma)$	$\{a\}$

If p and q are regular expressions representing the languages P and Q respectively, then:

RE	language
$p + q$	$P \cup Q$
pq (or $p \cdot q$)	$P \cdot Q$
p^*	P^*

Extended regular expression example: $p^+ \equiv pp^*$

RE examples

- $0 + 1$ denotes the language $\{0, 1\}$
- $a(b + c)$ denotes the language $\{a\} \cdot \{b, c\} = \{ab, ac\}$
  - ... which could also be denoted by $ab + ac$
- x^* denotes $\{x\}^*$
  - ... which could also be denoted by $\varepsilon + x + xxx^*$
- A regular expression is equivalent to *one and one only* regular language, but a regular language can have more than one corresponding regular expression.

ULB

Exercise 1

For each of the following languages (defined on the alphabet $\Sigma = \{0, 1\}$), design a RE that recognizes it:

1. The set of strings ending with 00.
2. The set of strings whose 10^{th} symbol, counted from the end of the string, is a 1.
3. The set of strings where each pair of zeroes is followed by a pair of ones.
4. The set of strings not containing 101.
5. The set of binary numbers divisible by 4.

Solution for exercise 1

1. $(0+1)^*00$
2. $(0+1)^*1(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)(0+1)$
3. $(1+01+0011)^*(0+\varepsilon)$
4. $0^*(1+00^+)^*0^*$
5. $(0+1)^*00+0$

ULB

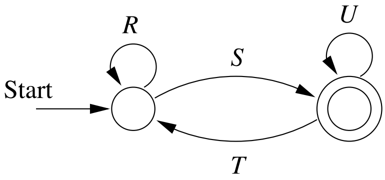
State elimination method

Given a DFA M, we can craft a corresponding regular expression using the *state elimination* method. The general idea is to label transitions in the automaton using regular expressions, pick a final state, then remove all other states step by step to finally reach a simple automaton which can then be used to easily determine a regular expression. There are two possible cases:

1. The start state of M is not final ($q_0 \notin F$)
2. The start state of M is final ($q_0 \in F$)

State elimination method (ctd.)

In the case where $q_0 \notin F$, we reach a two state automaton:

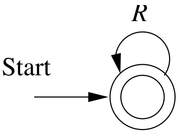


The corresponding regular expression is:

$$(R + SU^*T)^*SU^*$$

State elimination method (ctd.)

In the case where $q_0 \in F$, we reach a single state automaton:



The corresponding regular expression is:

$$R^*$$

State elimination method (ctd.)

For each final state $q^F \in F$, one has to build such a simple automaton to derive a regular expression $RE(q^F)$ that expresses all possible inputs that are accepted when M stops in q^F. The actual regular expression that describes the language $L(M)$ of the automaton M then simply becomes:

$$RE(q_1^F) + RE(q_2^F) + \ldots + RE(q_k^F) \qquad \text{where } \{q_1^F, \ldots, q_k^F\} = F$$

Algorithm

First, preprocess by labeling all transitions by a RE.

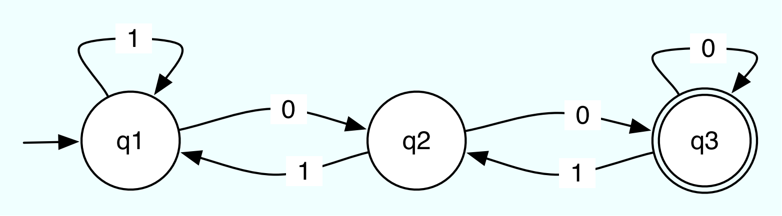
Then, for each state S_x to be eliminated, consider each transition (S_a, S_x), (S_x, S_b) or (S_x, S_x) with respective labels A, B and X.

The transition (S_a, S_b) labeled by E becomes the absorbing transition $E + (AX^*B)$ and remove A, B, X and E.

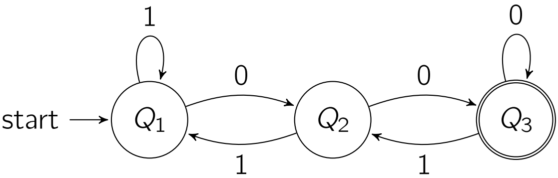
Note: some transitions can be null. In that case, does not consider the transition. For instance, if $E = (S_a, S_b)$ cannot be generated by δ (the transition function, see definition), then the absorption transition will be AX^*B.

Exercise 2.1

Design a RE accepting the same language as:



Solution for exercise 2.1



Transition	$(\mathbf{Q_a}, \mathbf{Q_x}) : \mathbf{A}$	$(\mathbf{Q_x}, \mathbf{Q_b}) : \mathbf{B}$	$(\mathbf{Q_x}, \mathbf{Q_x}) : \mathbf{X}$
(Q_1, Q_1)	$(Q_1, Q_2) : 0$	$(Q_2, Q_1) : 1$	$(Q_2, Q_2) : \emptyset$
(Q_1, Q_3)	$(Q_1, Q_2) : 0$	$(Q_2, Q_3) : 0$	$(Q_2, Q_2) : \emptyset$
(Q_3, Q_3)	$(Q_3, Q_2) : 1$	$(Q_2, Q_3) : 0$	$(Q_2, Q_2) : \emptyset$
(Q_3, Q_1)	$(Q_3, Q_2) : 1$	$(Q_2, Q_1) : 1$	$(Q_2, Q_2) : \emptyset$

We search absorbing transitions (Q_a, Q_a), (Q_a, Q_b), (Q_b, Q_b) and (Q_b, Q_a).

Solution for exercise 2.1

Transition	$(\mathbf{Q_a}, \mathbf{Q_x}) : \mathbf{A}$	$(\mathbf{Q_x}, \mathbf{Q_b}) : \mathbf{B}$	$(\mathbf{Q_x}, \mathbf{Q_x}) : \mathbf{X}$
(Q_1, Q_1)	$(Q_1, Q_2) : 0$	$(Q_2, Q_1) : 1$	$(Q_2, Q_2) : \emptyset$
(Q_1, Q_3)	$(Q_1, Q_2) : 0$	$(Q_2, Q_3) : 0$	$(Q_2, Q_2) : \emptyset$
(Q_3, Q_3)	$(Q_3, Q_2) : 1$	$(Q_2, Q_3) : 0$	$(Q_2, Q_2) : \emptyset$
(Q_3, Q_1)	$(Q_3, Q_2) : 1$	$(Q_2, Q_1) : 1$	$(Q_2, Q_2) : \emptyset$

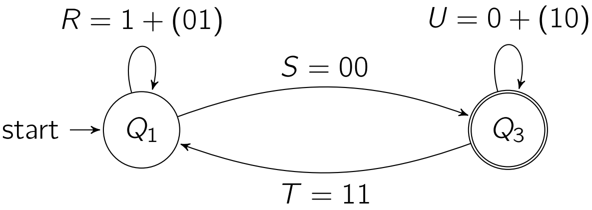
Applying the rule: $E + (AX^*B)$:

Absorbing transitions	$(Q, Q\prime) : E$	AX^*B	Result
$(\mathbf{Q_1}, \mathbf{Q_1}) : \mathbf{R}$	1	01	$1 + (01)$
$(\mathbf{Q_1}, \mathbf{Q_3}) : \mathbf{S}$	$\emptyset$	00	00
$(\mathbf{Q_3}, \mathbf{Q_3}) : \mathbf{U}$	0	10	$0 + (10)$
$(\mathbf{Q_3}, \mathbf{Q_1}) : \mathbf{T}$	$\emptyset$	11	11

Solution for exercise 2.1

Absorbing transitions	$(Q, Q\prime) : E$	AX^*B	Result
$(\mathbf{Q_1}, \mathbf{Q_1}) : \mathbf{R}$	1	01	$1 + (01)$
$(\mathbf{Q_1}, \mathbf{Q_3}) : \mathbf{S}$	$\emptyset$	00	00
$(\mathbf{Q_3}, \mathbf{Q_3}) : \mathbf{U}$	0	10	$0 + (10)$
$(\mathbf{Q_3}, \mathbf{Q_1}) : \mathbf{T}$	$\emptyset$	11	11

Give the automaton:

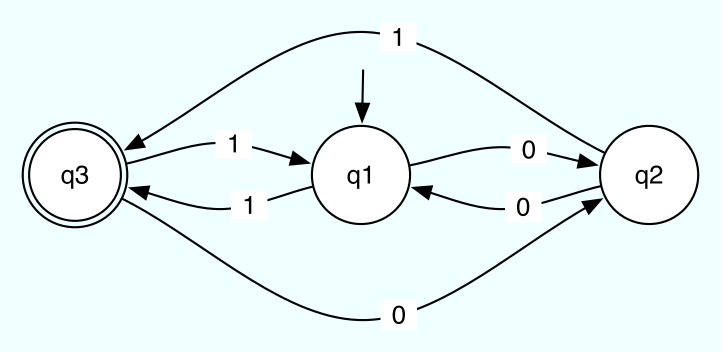


by applying the rule: $(R + SU^*T)^*SU^*$, the RE is:

$$((1 + (01)) + (00)(0 + (10))^*(11))^*(00)(0 + (10))^*$$

Exercise 2.2

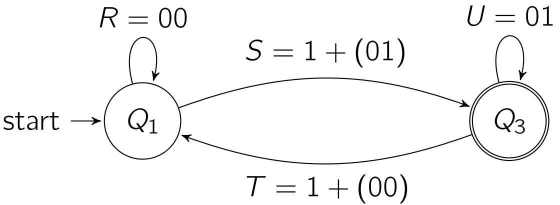
Design a RE accepting the same language as:



Solution for exercise 2.2

Absorbing transitions	$(Q, Q\prime) : E$	AX^*B	Result
$(\mathbf{Q_1}, \mathbf{Q_1}) : \mathbf{R}$	$\emptyset$	00	00
$(\mathbf{Q_1}, \mathbf{Q_3}) : \mathbf{S}$	1	01	$1 + (01)$
$(\mathbf{Q_3}, \mathbf{Q_3}) : \mathbf{U}$	$\emptyset$	01	01
$(\mathbf{Q_3}, \mathbf{Q_1}) : \mathbf{T}$	1	00	$1 + (00)$

Give the automaton:



by applying the rule: $(R + SU^*T)^*SU^*$, the RE is:

$$((00) + (1 + (01))(01)^*(1 + (00)))^*(1 + (01))(01)^*$$

Exercise 3

Convert the following REs into ε-NFAs:

1. 01^*
2. $(0+1)01$
3. $00(0+1)^*$

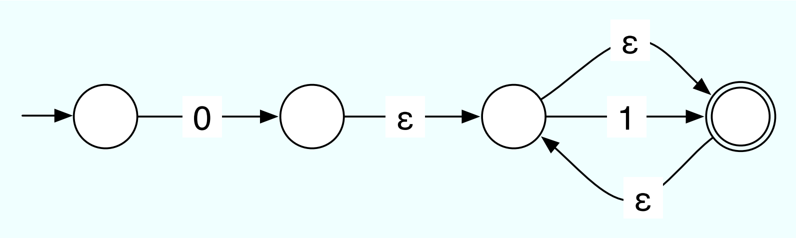
Recall:

$$AB \text{ gives } (Q_1) \longrightarrow_A (Q_2) \longrightarrow_B (Q_3)$$

$$A+B \text{ gives } (Q_4) \longleftarrow_\epsilon (Q\prime_3) \longleftarrow_A (Q\prime_2) \longleftarrow_\epsilon (Q_1) \longrightarrow_\epsilon (Q_2) \longrightarrow_B (Q_3) \longrightarrow_\epsilon (Q_4)$$

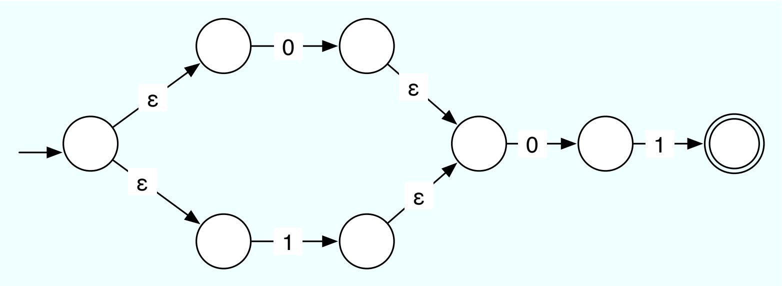
$$X* \; (Q_2) \longleftarrow_X (Q_1) \longleftrightarrow_\epsilon (Q_2)$$

Solution for exercise 3



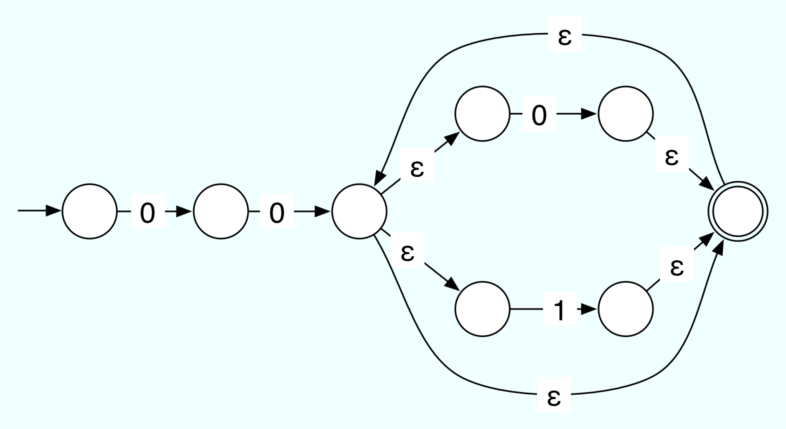
Solution for exercise 3

$(0+1)01$



Solution for exercise 3

$00(0+1)^*$



Extended regular expressions (ERE)

- Very popular on UNIX-like tools (`grep`, `find`, *etc.*)
- Grant more flexibility than traditional regular expressions
- Typically used by scanner generators such as `lex`

ERE syntax

Expression	Accepted language
r*	0 or more rs
r+	1 or more rs
r?	0 or 1 r
[abc]	a or b or c
[a-z]	Any character in the interval a...z
.	Any character except \n
[^s]	Any character but those in s
r{m,n}	Between m and n occurrences of r
r1 r2	The concatenation of r1 and r2

ERE syntax (ctd.)

Expression	Accepted language
`r1	r2`	`r1` or `r2`
`(r)`	`r`
`^r`	`r` if it starts a line
`r$`	`r` if it ends a line
`"s"`	The string `s`
`\c`	The character `c`
`r1(?=r2)`	`r1` when it's followed by `r2`

Examples

Expression	Accepted language
`[a-zA-Z]`	Any letter (upper or lower case)
`[0-9]`	Any digit
`a[^A-Za-z]b`	An `a` followed by a non-alphabetical character and a `b`
`^Silly`	`Silly` if it starts a line
`[a-zA-Z]([a-zA-Z]|[0-9])*`	An identifier in the Pascal language

Exercise 4

1. Give an extended regular expression (ERE) that targets any sequence of 5 characters, including the newline character `\n`
2. Give an ERE that targets any string starting with an arbitrary number of \ followed by any number of `*`
3. UNIX-like shells (such as `bash`) allow the user to write *batch* files in which comments can be added. A line is defined to be a comment if it starts with a # sign. What ERE accepts such comments?
4. Design an ERE that accepts numbers in scientific notation. Such a number must contain at least one digit and has two optional parts:
   - A "decimal" part : a dot followed by a sequence of digits
   - An "exponential" part: an `E` followed by an integer that may be prefixed by + or -
   - Examples : `42`, `66.4E-5`, `8E17`, `...`

ULB

Exercise 4

5. Design an ERE that accepts "correct" phrases that fulfill the following criteria:
   - No prepending/appending spaces
   - The first word must start with a capital letter
   - The phrase must end with a full stop `.`
   - The phrase must be made of one or more words (made of the characters `a...z` and `A...Z`) separated by a single space
   - There must be one sentence per line

   Punctuation signs other than a full stop are not allowed.

6. Craft an ERE that accepts old school DOS-style filenames (8 characters in `a...z`, `A...Z` and `_`) whose extension is `.ext` and that begin with the string `abcde`. We ask that the ERE only accept the filename *without capturing the extension!*
   - Example: on `abcdeLOL.ext`, the ERE must accept `abcdeLOL`

Solution for exercise 4

1. `(.|\n){5}`
2. `\\*\**`
3. `^#.*$`
4. `[0-9]+(\.[0-9]+)?(E[+-]?[0-9]+)?`
5. `^ [A-Z][A-Za-z]*(\ [A-Za-z]+)*\.$`
6. `abcde[A-Za-z_]{3}(?=\.ext)`