



Regular Expressions (REs)

Expressions

- In arithmetic:

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parantheses

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parentheses
 2. expressions represent computations with numbers

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parantheses
 2. expressions represent computations with numbers
 3. results of expressions evaluation are numbers

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parantheses
 2. expressions represent computations with numbers
 3. results of expressions evaluation are numbers
- In automata theory:

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parentheses
 2. expressions represent computations with numbers
 3. results of expressions evaluation are numbers
- In automata theory:
 1. regular expressions are constructed from regular languages using regular operations and parentheses

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parantheses
 2. expressions represent computations with numbers
 3. results of expressions evaluation are numbers
- In automata theory:
 1. regular expressions are constructed from regular languages using regular operations and parentheses
 2. regular expressions represent computations with regular languages

Expressions

- In arithmetic:
 1. expressions are constructed from numbers and variables using arithmetic operations and parantheses
 2. expressions represent computations with numbers
 3. results of expressions evaluation are numbers
- In automata theory:
 1. regular expressions are constructed from regular languages using regular operations and parentheses
 2. regular expressions represent computations with regular languages
 3. result of regular expressions evaluation are regular languages

Example

- $(5 + 3) \times 4$ is an arithmetic expression constructed with operations $+$ and \times ; its value is the number 32

Example

- $(5 + 3) \times 4$ is an arithmetic expression constructed with operations $+$ and \times ; its value is the number 32
- $(0 \cup 1)0^*$ is a regular expression constructed from the languages 0 and 1 using regular operations;

Example

- $(5 + 3) \times 4$ is an arithmetic expression constructed with operations $+$ and \times ; its value is the number 32
- $(0 \cup 1)0^*$ is a regular expression constructed from the languages 0 and 1 using regular operations;
- it evaluates to the language that contains all strings of zero-s and 1 followed by zero-s

-
-
-

Regular expression evaluation

Similar to the evaluation of arithmetic expression

Regular expression evaluation

Similar to the evaluation of arithmetic expression

- Identify first the constants (languages) involved.

Regular expression evaluation

Similar to the evaluation of arithmetic expression

- Identify first the constants (languages) involved.
- Example: 0 represents the language $\{0\}$ and 1 represents the language $\{1\}$

Regular expression evaluation

Similar to the evaluation of arithmetic expression

- Identify first the constants (languages) involved.
- Example: 0 represents the language $\{0\}$ and 1 represents the language $\{1\}$
- Apply the regular operations on the languages thus identified following their set-theory definitions.

Regular expression evaluation

Similar to the evaluation of arithmetic expression

- Identify first the constants (languages) involved.
- Example: 0 represents the language $\{0\}$ and 1 represents the language $\{1\}$
- Apply the regular operations on the languages thus identified following their set-theory definitions.
- Example: $(0 \cup 1)$ represents the language $\{0, 1\}$, 0^* represents the language $\{\epsilon, 0, 00, \dots\}$ and $(0 \cup 1)0^*$ represents the language $\{0, 1, 00, 10, 000, 100, \dots\}$

Regular expression evaluation

Similar to the evaluation of arithmetic expression

- Identify first the constants (languages) involved.
- Example: 0 represents the language $\{0\}$ and 1 represents the language $\{1\}$
- Apply the regular operations on the languages thus identified following their set-theory definitions.
- Example: $(0 \cup 1)$ represents the language $\{0, 1\}$, 0^* represents the language $\{\epsilon, 0, 00, \dots\}$ and $(0 \cup 1)0^*$ represents the language $\{0, 1, 00, 10, 000, 100, \dots\}$

Note: by convention, the symbol \circ of concatenation op. is not written in regular expressions

Applications

1. Regular expressions provide a powerful method to describe patterns searched in various texts

Applications

1. Regular expressions provide a powerful method to describe patterns searched in various texts
2. Utilities such as AWK, GREP in Unix, modern programming languages, such as PERL, text editors, all use regular expressions for their pattern description

Applications

1. Regular expressions provide a powerful method to **describe patterns** searched in various texts
2. Utilities such as AWK, GREP in Unix, modern programming languages, such as PERL, text editors, **all use regular expressions** for their pattern description
3. The lexicon of programming languages is specified by **regular expressions**. Hence, **lexical analysis** is done using regular expressions.

Applications

1. Regular expressions provide a powerful method to describe patterns searched in various texts
2. Utilities such as AWK, GREP in Unix, modern programming languages, such as PERL, text editors, all use regular expressions for their pattern description
3. The lexicon of programming languages is specified by regular expressions. Hence, lexical analysis is done using regular expressions.
4. Other examples?

For a given alphabet Σ

- The regular expression Σ describes the language consisting of all strings of length 1 over Σ

For a given alphabet Σ

- The regular expression Σ describes the language consisting of all strings of length 1 over Σ
- Σ^* describes the language of all strings over the alphabet Σ

For a given alphabet Σ

- The regular expression Σ describes the language consisting of all strings of length 1 over Σ
- Σ^* describes the language of all strings over the alphabet Σ
- Σ^*1 is the language of all strings over Σ that ends in 1

For a given alphabet Σ

- The regular expression Σ describes the language consisting of all strings of length 1 over Σ
- Σ^* describes the language of all strings over the alphabet Σ
- Σ^*1 is the language of all strings over Σ that ends in 1
- Language $(0\Sigma^*) \cup (\Sigma^*1)$ describes the language that consists of all strings that either start with 0 or end in 1

Precedence relation

Denoting \preceq the rule describing the order of ops in the evaluation of arithmetic and regular expressions we have:

Precedence relation

Denoting \preceq the rule describing the order of ops in the evaluation of arithmetic and regular expressions we have:

- **Arithmetic expressions:** $() \preceq \{\times, /\} \preceq \{+, -\}$

Precedence relation

Denoting \preceq the rule describing the order of ops in the evaluation of arithmetic and regular expressions we have:

- **Arithmetic expressions:** $() \preceq \{\times, /\} \preceq \{+, -\}$
- **Regular expressions:** $() \preceq \{*\} \preceq \{\cup, \circ\}$

Precedence relation

Denoting \preceq the rule describing the order of ops in the evaluation of arithmetic and regular expressions we have:

- **Arithmetic expressions:** $() \preceq \{\times, /\} \preceq \{+, -\}$
- **Regular expressions:** $() \preceq \{*\} \preceq \{\cup, \circ\}$

Note: $()$ denotes expressions enclosed in parentheses

Formal definitions

Consider Σ an alphabet. A regular expression (RE) R over Σ is defined recursively by the rules:

Formal definitions

Consider Σ an alphabet. A regular expression (RE) R over Σ is defined recursively by the rules:

Recursion basis:

Formal definitions

Consider Σ an alphabet. A regular expression (RE) R over Σ is defined recursively by the rules:

Recursion basis:

1. For any $a \in \Sigma$, a is a RE describing the language $L(a) = \{a\}$

Formal definitions

Consider Σ an alphabet. A regular expression (RE) R over Σ is defined recursively by the rules:

Recursion basis:

1. For any $a \in \Sigma$, a is a RE describing the language $L(a) = \{a\}$
2. ϵ is a RE describing the language $L(\epsilon) = \{\epsilon\}$

Formal definitions

Consider Σ an alphabet. A regular expression (RE) R over Σ is defined recursively by the rules:

Recursion basis:

1. For any $a \in \Sigma$, a is a RE describing the language $L(a) = \{a\}$
2. ϵ is a RE describing the language $L(\epsilon) = \{\epsilon\}$
3. \emptyset is a RE describing the empty language $L(\emptyset) = \emptyset$.

Formal definitions, continuation

Recursion body: If R_1 and R_2 are REs evaluating to the languages $L(R_1)$ and $L(R_2)$ then:

Formal definitions, continuation

Recursion body: If R_1 and R_2 are REs evaluating to the languages $L(R_1)$ and $L(R_2)$ then:

1. $(R_1 \cup R_2)$ is a RE evaluating to $L(R_1) \cup L(R_2)$

Formal definitions, continuation

Recursion body: If R_1 and R_2 are REs evaluating to the languages $L(R_1)$ and $L(R_2)$ then:

1. $(R_1 \cup R_2)$ is a RE evaluating to $L(R_1) \cup L(R_2)$
2. $(R_1 \circ R_2)$ is a RE evaluating to $L(R_1) \circ L(R_2)$

Formal definitions, continuation

Recursion body: If R_1 and R_2 are REs evaluating to the languages $L(R_1)$ and $L(R_2)$ then:

1. $(R_1 \cup R_2)$ is a RE evaluating to $L(R_1) \cup L(R_2)$
2. $(R_1 \circ R_2)$ is a RE evaluating to $L(R_1) \circ L(R_2)$
3. R_1^* is a RE evaluating to $\bigcup_{i \geq 0} L(R_1)^i$

where $L(R_1)^i = L(R_1)^{i-1} \circ L(R_1)$ and $L(R_1)^0 = \epsilon$.

Note

A definition of the form expressed by rule (4) above is called an **inductive definition**

Potential confusions

- Do not confuse the REs ϵ and \emptyset ;

Potential confusions

- Do not confuse the REs ϵ and \emptyset ;
 1. ϵ represents the language containing just the empty string

Potential confusions

- Do not confuse the REs ϵ and \emptyset ;
 1. ϵ represents the language containing just the empty string
 2. \emptyset is the RE that represents the the empty language

Potential confusions

- Do not confuse the REs ϵ and \emptyset ;
 1. ϵ represents the language containing just the empty string
 2. \emptyset is the RE that represents the the empty language
- Note the distinction between R and $L(R)$. R is an expression and $L(R)$ is the set of strings specified by R

Example REs

1. 0^*10^* ,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 10$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$
8. $(0 \cup \epsilon)1^*$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$
8. $(0 \cup \epsilon)1^*$, $L((0 \cup \epsilon)1^*) = \{01^* \cup 1^*\}$

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$
8. $(0 \cup \epsilon)1^*$, $L((0 \cup \epsilon)1^*) = \{01^* \cup 1^*\}$
9. $(0 \cup \epsilon)(1 \cup \epsilon)$,

Example REs

1. 0^*10^* , $L(0^*10^*) = \{w \mid w \text{ contains exactly a single } 1\}$
2. $\Sigma^*1\Sigma^*$, $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
3. $\Sigma^*001\Sigma^*$, $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains } 001 \text{ as a substring}\}$
4. $(\Sigma\Sigma)^*$, $L(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$
5. $(\Sigma\Sigma\Sigma)^*$, $L(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$
6. $01 \cup 01$, $L(01 \cup 01) = \{01, 01\}$
7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$, $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) = \{w \mid w \text{ starts and ends with the same symbol}\}$
8. $(0 \cup \epsilon)1^*$, $L((0 \cup \epsilon)1^*) = \{01^* \cup 1^*\}$
9. $(0 \cup \epsilon)(1 \cup \epsilon)$, $L((0 \cup \epsilon)(1 \cup \epsilon)) = \{\epsilon, 0, 1, 01\}$

More example

10. $1^*\emptyset$, $L(1^*\emptyset) = \emptyset$;

Note: concatenating \emptyset to any set **yields** \emptyset

More example

10. $1^*\emptyset$, $L(1^*\emptyset) = \emptyset$;

Note: concatenating \emptyset to any set **yields \emptyset**

11. \emptyset^* , $L(\emptyset^*) = \{\epsilon\}$; by definition,

ϵ is in the **star operation** applied on any language;

if the **language is empty** ϵ becomes **the one element**

Identities

If R is a RE then the following identities take place:

Identities

If R is a RE then the following identities take place:

- $L(R \cup \emptyset) = L(R)$; adding the empty language to any other language will not change that language

Identities

If R is a RE then the following identities take place:

- $L(R \cup \emptyset) = L(R)$; adding the empty language to any other language will not change that language
- $L(R \circ \epsilon) = L(R)$; concatenating any string with the empty string does not change that string

Note

- $L(R \cup \epsilon) \neq L(R)$. Example: if $R = 0$ then $L(R) = 0$;
 $L(R \cup \epsilon) = \{0, \epsilon\}$

Note

- $L(R \cup \epsilon) \neq L(R)$. Example: if $R = 0$ then $L(R) = 0$;
 $L(R \cup \epsilon) = \{0, \epsilon\}$
- $L(R \circ \emptyset) \neq L(R)$. Example: if $R = 0$ then $L(R) = \{0\}$
but $L(R \circ \emptyset) = \emptyset$

Application

- REs are useful tools for the design of compilers

Application

- REs are useful tools for the design of compilers
- Language lexicon is described by REs.

Application

- REs are useful tools for the design of compilers
- Language lexicon is described by REs.

Example: numerical constants can be described by:

$\{+, -, \epsilon\}(DD^* \cup DD^*.D^* \cup D.DD^*)$ where

$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Application

- REs are useful tools for the design of compilers
- Language lexicon is described by REs.

Example: numerical constants can be described by:

$\{+, -, \epsilon\}(DD^* \cup DD^*.D^* \cup D.DD^*)$ where

$D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- From the lexicon description by REs one can generate automatically lexical analyzers

New terminology

A RE R that evaluates to the language $L(R)$ is further said that specifies the language $L(R)$

Equivalence with FA

- REs and finite automata are equivalent in their descriptive power

Equivalence with FA

- REs and finite automata are equivalent in their descriptive power
- Any RE E can be converted into a finite automaton, A_E , that recognizes the language specified by E

Equivalence with FA

- REs and finite automata are equivalent in their descriptive power
- Any RE E can be converted into a finite automaton, A_E , that recognizes the language specified by E
- Vice-versa, any finite automaton recognizing a language A can be converted into a RE E_A that specifies the language A

Theorem 1.54

Language is regular iff some RE specifies it

Theorem 1.54

Language is regular iff some RE specifies it

Proof idea: this proof has two parts:

Theorem 1.54

Language is regular iff some RE specifies it

Proof idea: this proof has two parts:

- **First part:** we show that a language specified by a RE is regular, i.e., there is a finite automaton that recognizes it.

Theorem 1.54

Language is regular iff some RE specifies it

Proof idea: this proof has two parts:

- **First part:** we show that a language specified by a RE is regular, i.e., there is a finite automaton that recognizes it.
- **Second part:** we show that if a language is regular then there is a RE that specifies it

Lemma 1.55

If a language is specified by a RE, then it is regular.

Lemma 1.55

If a language is specified by a RE, then it is regular.

Proof idea: Assume that we have a RE R that evaluate to the language A .

Lemma 1.55

If a language is specified by a RE, then it is regular.

Proof idea: Assume that we have a RE R that evaluate to the language A .

1. We will show how to convert R into an NFA that recognizes A .

Lemma 1.55

If a language is specified by a RE, then it is regular.

Proof idea: Assume that we have a RE R that evaluate to the language A .

1. We will show how to convert R into an NFA that recognizes A .
2. Then by corollary 1.40, if an NFA recognizes A then A is regular

Proof

Convert R into an NFA N by the following six-steps procedure:

Step 1:

If $R = a \in \Sigma$, then $L(R) = \{a\}$ and the NFA N recognizing $L(R)$ is in Figure 1

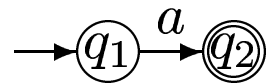


Figure 1: NFA N recognizing $\{a\}$

Step 1:

If $R = a \in \Sigma$, then $L(R) = \{a\}$ and the NFA N recognizing $L(R)$ is in Figure 1

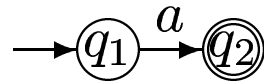


Figure 1: NFA N recognizing $\{a\}$

Note: this is an NFA but not a DFA because it has states with no exiting arrow for each possible input symbol

Formal construction

Formally $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$ where:

$$\delta(q_1, a) = \{q_2\},$$

$$\delta(r, b) = \emptyset, \text{ for } r \neq q_1 \text{ and } b \neq a$$

Step 2:

If $R = \epsilon$ then $L(R) = \{\epsilon\}$ and the NFA N that recognizes $L(R)$ is in Figure 2

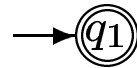


Figure 2: The NFA N recognizing $\{\epsilon\}$

Formal construction

Formally $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$ where:

$\delta(r, b) = \emptyset$ for any r and $b \in \Sigma$

Step 3:

If $R = \emptyset$ then $L(R) = \emptyset$, and the NFA N that recognizes $L(R)$ is in Figure 3

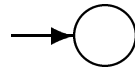


Figure 3: The NFA N recognizing \emptyset

Formal construction

Formally $N = (\{q\}, \Sigma, \delta, q, \emptyset)$ where:

$\delta(r, b) = \emptyset$ for any r and b

Step 4:

If $R = R_1 \cup R_2$ then $L(R) = L(R_1) \cup L(R_2)$.

Step 4:

If $R = R_1 \cup R_2$ then $L(R) = L(R_1) \cup L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

Step 4:

If $R = R_1 \cup R_2$ then $L(R) = L(R_1) \cup L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$

Step 4:

If $R = R_1 \cup R_2$ then $L(R) = L(R_1) \cup L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$
2. $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ is an NFA recognizing $L(R_2)$

Step 4:

If $R = R_1 \cup R_2$ then $L(R) = L(R_1) \cup L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$
2. $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ is an NFA recognizing $L(R_2)$

The NFA N recognizing $L(R_1 \cup R_2)$ is given in Figure 4

NFA recognizing $L(R_1) \cup L(R_2)$

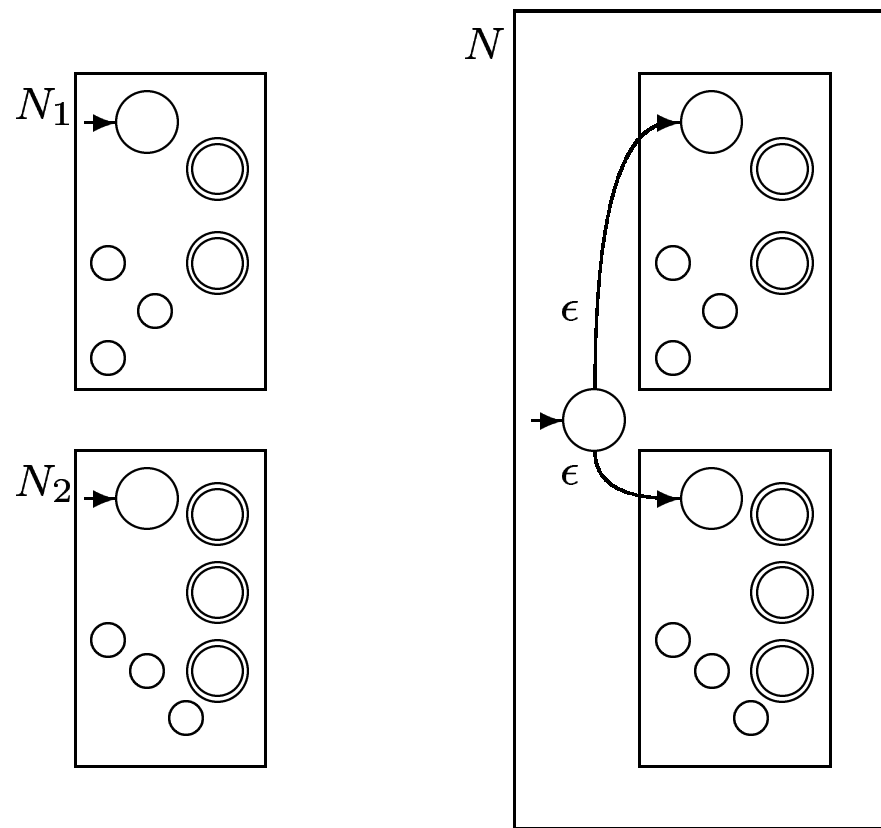


Figure 4: Construction of N to recognize $L(R_1 \cup R_2)$

Construction procedure

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$: That is, the states of N are all states on N_1 and N_2 with the addition of a new state q_0

Construction procedure

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$: That is, the states of N are all states on N_1 and N_2 with the addition of a new state q_0
2. The start state of N is q_0

Construction procedure

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$: That is, the states of N are all states on N_1 and N_2 with the addition of a new state q_0
2. The start state of N is q_0
3. The accept states of N are $F = F_1 \cup F_2$: That is, the accept states of N are all the accept states of N_1 and N_2

Construction procedure

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$: That is, the states of N are all states on N_1 and N_2 with the addition of a new state q_0
2. The start state of N is q_0
3. The accept states of N are $F = F_1 \cup F_2$: That is, the accept states of N are all the accept states of N_1 and N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \\ \delta_2(q, a), & \text{if } q \in Q_2 \\ \{q_1, q_2\}, & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset, & \text{if } q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

Step 5:

If $R = R_1 \circ R_2$ then $L(R) = L(R_1) \circ L(R_2)$.

Step 5:

If $R = R_1 \circ R_2$ then $L(R) = L(R_1) \circ L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

Step 5:

If $R = R_1 \circ R_2$ then $L(R) = L(R_1) \circ L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$

Step 5:

If $R = R_1 \circ R_2$ then $L(R) = L(R_1) \circ L(R_2)$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$
2. $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ is an NFA recognizing $L(R_2)$.

NFA recognizing $L(R_1) \circ L(R_2)$

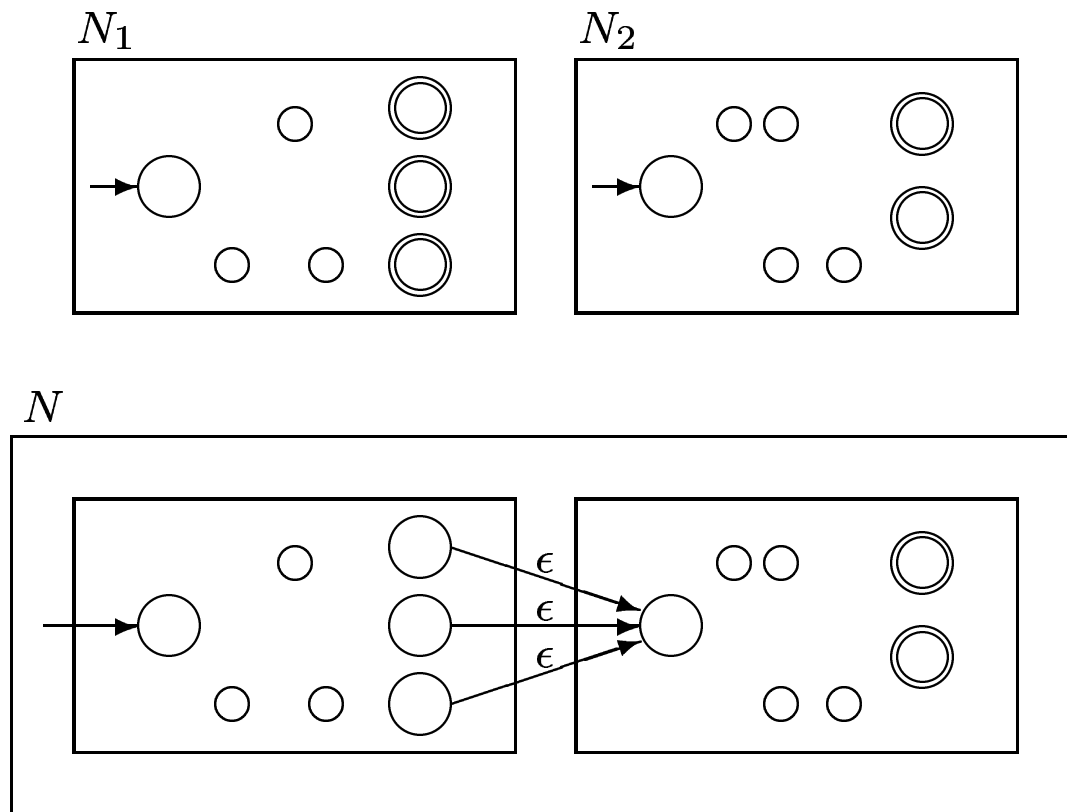


Figure 5: Construction of N to recognize $L(R_1 \circ R_2)$

Construction procedure

1. $Q = Q_1 \cup Q_2$. The states of N are all states of N_1 and N_2

Construction procedure

1. $Q = Q_1 \cup Q_2$. The states of N are all states of N_1 and N_2
2. The start state is the state q_1 of N_1

Construction procedure

1. $Q = Q_1 \cup Q_2$. The states of N are all states of N_1 and N_2
2. The start state is the state q_1 of N_1
3. The accept states is the set F_2 of the accept states of N_2

Construction procedure

1. $Q = Q_1 \cup Q_2$. The states of N are all states of N_1 and N_2
2. The start state is the state q_1 of N_1
3. The accept states is the set F_2 of the accept states of N_2
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a), & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_2\}, & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_2(q, a), & \text{if } q \in Q_2. \end{cases}$$

Step 6:

If $R = R_1^*$ then $L(R) = \cup_{i \geq 0} L(R_1)^i$.

Step 6:

If $R = R_1^*$ then $L(R) = \cup_{i \geq 0} L(R_1)^i$.

Note: in view with the inductive nature of R we may assume that:

Step 6:

If $R = R_1^*$ then $L(R) = \cup_{i \geq 0} L(R_1)^i$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$

Step 6:

If $R = R_1^*$ then $L(R) = \cup_{i \geq 0} L(R_1)^i$.

Note: in view with the inductive nature of R we may assume that:

1. $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ is an NFA recognizing $L(R_1)$

The NFA N recognizing $L(R_1^*)$ is given in Figure 6

NFA recognizing $L(R_1^*)$

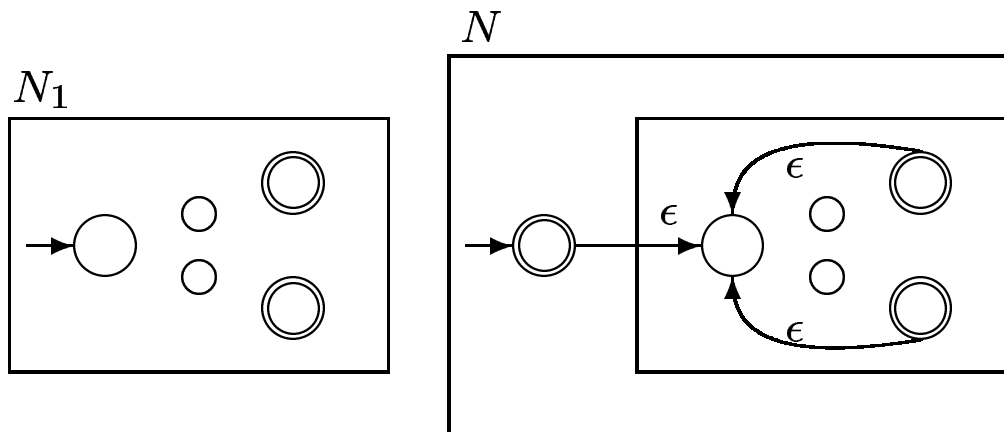


Figure 6: Construction of N to recognize $L(R_1^*)$

Construction procedure

1. $Q = \{q_0\} \cup Q_1$; that is, states of N are the states of N_1 plus a new state q_0

Construction procedure

1. $Q = \{q_0\} \cup Q_1$; that is, states of N are the states of N_1 plus a new state q_0
2. Start state of N is q_0

Construction procedure

1. $Q = \{q_0\} \cup Q_1$; that is, states of N are the states of N_1 plus a new state q_0
2. Start state of N is q_0
3. $F = \{q_0\} \cup F_1$; that is, the accept states of N are the accept states of N_1 plus the new start state

Construction procedure

1. $Q = \{q_0\} \cup Q_1$; that is, states of N are the states of N_1 plus a new state q_0
2. Start state of N is q_0
3. $F = \{q_0\} \cup F_1$; that is, the accept states of N are the accept states of N_1 plus the new start state
4. Define δ so that for any $q \in Q$ and $a \in \Sigma_\epsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a), & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a), & \text{if } q \in F_1 \text{ and } a \neq \epsilon \\ \delta_1(q, a) \cup \{q_1\}, & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \{q_1\}, & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset, & \text{if } q = q_0 \text{ and } a \neq \epsilon. \end{cases}$$

Examples conversion

Convert the following REs into NFA following the procedure presented above

1. $(ab \cup a)^*$ to an NFA
2. $(a \cup b)^*aba$