Lexical Analysis and Automata Theory

CSCI 3136
Principles of Programming Languages

Faculty of Computer Science
Dalhousie University

Winter 2013

Reading: Chapter 2
Motivation

Front end

Source program (character stream) → Scanner (lexical analysis) → Token stream

Parser (syntactic analysis) → Parse tree → Semantic analysis and code generation

Back end

Modified intermediate form → Machine-independent code improvement → Target code generation → Target language (e.g., assembly)

Modified target language → Machine-specific code improvement

Symbol table

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
A formal language $\mathcal{L}$ is a set of strings over an alphabet $\Sigma$. 
A **formal language** $L$ is a set of strings over an alphabet $\Sigma$.

- **Alphabet $\Sigma$:** set of characters that can be used to form strings (letters, digits, punctuation, . . .)
- **String:** finite sequence of characters
- $\varepsilon$ denotes the *empty string* (string with no letters: ““)
- **Length** $|s|$ of a string $s =$ number of characters in $s$
  - $|\varepsilon| = 0$, $|a| = 1$, $|abc| = 3$, . . .
- $\Sigma^0 =$ set of strings of length 0: $\Sigma^0 = \{\varepsilon\}$
  - $\Sigma^1 =$ set of strings of length 1: $\Sigma^1 = \{a, b, c, . . .\}$
  - $\Sigma^2 =$ set of strings of length 2: $\Sigma^2 = \{aa, ab, . . ., ca, . . .\}$
- **Kleene star:** $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup . . .$ (set of all strings over alphabet $\Sigma$)
- A **formal language** $L$ over alphabet $\Sigma$ is a subset $L \subseteq \Sigma^*$
Examples of Formal Languages

Some finite languages
- \{\}, \{\epsilon\}, \{0, 1\}, \{\epsilon, 0, 1, 00, 01, 100\}
- \{the, is, I, you, he, she, it, man, are\}
- ...

Some infinite languages
- \{0\}^*, \{0, 1\}^*, \{a, b, c\}^*
- \{01^n 0 \mid n \geq 0\}
- \{a^p \mid p \text{ is a prime number}\}
- Set of all positive integers
- Set of all syntactically correct C programs
- ...

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Examples of Formal Languages

Some finite languages

- {}, {ε}, {0, 1}, {ε, 0, 1, 00, 01, 100}
- {the, is, I, you, he, she, it, man, are}
- ...

Some infinite languages

- {0}*, {0, 1}*, {a, b, c}*
- {01^n0 | n ≥ 0}
- {a^p | p is a prime number}
- Set of all positive integers
- Set of all syntactically correct C programs
- ...

A simple application of formal languages

- Accept valid email addresses and reject invalid ones:

  \[
  \text{email_address} \rightarrow \text{user_id} \, \text{@} \, \text{domain} \\
  \text{user_id} \rightarrow \text{word} \\
  \text{domain} \rightarrow \text{word} \, | \, \text{word} \, . \, \text{domain}
  \]
Formal Languages and Automata

- **Regular languages**
  - Recognized (decided) by finite automata
  - Useful for tokenizing program text (lexical analysis)

- **Context-free languages**
  - Recognized (decided) by non-deterministic push-down automata
  - Useful for parsing the syntax of a program (syntactic analysis)
Formal Languages and Automata

- **Regular languages**
  - Recognized (decided) by finite automata
  - Useful for tokenizing program text (lexical analysis)

- **Context-free languages**
  - Recognized (decided) by non-deterministic push-down automata
  - Useful for parsing the syntax of a program (syntactic analysis)
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
A recursive definition

- $\emptyset$, \{\epsilon\}, and \{a\} are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab | a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$
- Any finite language
- $\{0\}^*$
- $\{0, 1\}^*$, $\{a, b, c\}^*$
- $\{01^n0 \mid n \geq 0\}$
- Set of all positive integers in decimal representation
- $\{0^m1^n \mid m \geq 0, n \geq 0\}$
- $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{\binom{n}{k}^n \mid n \geq 0\}$
- Set of all syntactically correct C programs
- $\{a^p \mid p$ is a prime number$\}$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$ Yes
- Any finite language
- $\{0\}^*$
- $\{0, 1\}^*$, $\{a, b, c\}^*$
- $\{01^n0 \mid n \geq 0\}$
- Set of all positive integers in decimal representation
- $\{0^m1^n \mid m \geq 0, n \geq 0\}$,
  $\{a^kb^mc^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{(\cdot)^n \mid n \geq 0\}$
- Set of all syntactically correct C programs
- $\{a^p \mid p \text{ is a prime number}\}$
Regular Languages

A recursive definition

- \( \emptyset, \{ \varepsilon \}, \) and \( \{ a \} \) are regular languages, where \( a \in \Sigma \)
- If \( A \) and \( B \) are regular languages, then the following are also regular languages:
  - \( A \cup B \)
  - \( AB := \{ ab \mid a \in A, b \in B \} \) (the concatenation of two strings in \( A \) and \( B \))
  - \( A^* \)

A few examples

- \{ a, b, ab \} \hspace{1cm} Yes
- Any finite language \hspace{1cm} Yes
- \{ 0 \}^* \hspace{1cm} Yes
- \{ 0, 1 \}^*, \{ a, b, c \}^* \hspace{1cm} Yes
- \{ 01^n 0 \mid n \geq 0 \} \hspace{1cm} Yes
- Set of all positive integers in decimal representation \hspace{1cm} Yes
- \{ 0^m 1^n \mid m \geq 0, n \geq 0 \}, \hspace{1cm} Yes
  \{ a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0 \}
- \{ (n)^n \mid n \geq 0 \}
- Set of all syntactically correct C programs
- \{ a^p \mid p \text{ is a prime number} \}
Regular Languages

A recursive definition

• $\emptyset$, {$e$}, and {$a$} are regular languages, where $a \in \Sigma$

• If $A$ and $B$ are regular languages, then the following are also regular languages:
  – $A \cup B$
  – $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  – $A^*$

A few examples

• Yes
  • $\{a, b, ab\}$
  • Any finite language
  • $\{0\}^*$
  • $\{0, 1\}^*, \{a, b, c\}^*$
  • $\{01^n0 \mid n \geq 0\}$
  • Set of all positive integers in decimal representation

  • Yes
  • $\{0^m1^n \mid m \geq 0, n \geq 0\}$,
  • $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
  • $\{(n)^n \mid n \geq 0\}$
  • Set of all syntactically correct C programs
  • $\{a^p \mid p$ is a prime number$\}$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$  Yes
- Any finite language  Yes
- $\{0\}^*$  Yes
- $\{0, 1\}^*, \{a, b, c\}^*$  Yes
- $\{01^n0 \mid n \geq 0\}$  Yes
- Set of all positive integers in decimal representation
- $\{0^m1^n \mid m \geq 0, n \geq 0\},$  Yes
  $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{(n)^n \mid n \geq 0\}$  No
- Set of all syntactically correct C programs
- $\{a^p \mid p \text{ is a prime number}\}$  Yes

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Regular Languages

A recursive definition

• $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$

• If $A$ and $B$ are regular languages, then the following are also regular languages:
  – $A \cup B$
  – $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  – $A^*$

A few examples

• $\{a, b, ab\}$
• Any finite language
• $\{0\}^*$
• $\{0, 1\}^*$, $\{a, b, c\}^*$
• $\{01^n0 \mid n \geq 0\}$
• Set of all positive integers in decimal representation

• $\{0^m1^n \mid m \geq 0, n \geq 0\}$,
• $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
• $\{(\binom{n}{k})^n \mid n \geq 0\}$
• Set of all syntactically correct C programs
• $\{a^p \mid p \text{ is a prime number}\}$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$
- Any finite language
- $\{0\}^*$
- $\{0, 1\}^*, \{a, b, c\}^*$
- $\{01^n0 \mid n \geq 0\}$
- Set of all positive integers in decimal representation
- $\{0^m1^n \mid m \geq 0, n \geq 0\}$
- $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{(n)^n \mid n \geq 0\}$
- Set of all syntactically correct C programs
- $\{a^p \mid p \text{ is a prime number}\}$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$ Yes
- Any finite language Yes
- $\{0\}^*$ Yes
- $\{0, 1\}^*, \{a, b, c\}^*$ Yes
- $\{01^n0 \mid n \geq 0\}$ Yes
- Set of all positive integers in decimal representation Yes
- $\{0^m1^n \mid m \geq 0, n \geq 0\}$ Yes
- $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$ Yes
- $\{(\binom{n}{k})^n \mid n \geq 0\}$ Yes
- Set of all syntactically correct C programs
- $\{a^p \mid p \text{ is a prime number}\}$
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$ Yes
- Any finite language Yes
- $\{0\}^*$ Yes
- $\{0, 1\}^*, \{a, b, c\}^*$ Yes
- $\{01^n 0 \mid n \geq 0\}$ Yes
- Set of all positive integers in decimal representation Yes
- $\{0^m 1^n \mid m \geq 0, n \geq 0\}$, Yes
- $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{(n)^n \mid n \geq 0\}$ No
- Set of all syntactically correct C programs
- $\{a^p \mid p \text{ is a prime number}\}$
Regular Languages

A recursive definition

- \( \emptyset, \{ \epsilon \}, \) and \( \{ a \} \) are regular languages, where \( a \in \Sigma \)
- If \( A \) and \( B \) are regular languages, then the following are also regular languages:
  - \( A \cup B \)
  - \( AB := \{ ab \mid a \in A, b \in B \} \) (the concatenation of two strings in \( A \) and \( B \))
  - \( A^* \)

A few examples

- \( \{ a, b, ab \} \)
- Any finite language
- \( \{ 0 \}^* \)
- \( \{ 0, 1 \}^*, \{ a, b, c \}^* \)
- \( \{ 01^n 0 \mid n \geq 0 \} \)
- Set of all positive integers in decimal representation

- \( \{ 0^m 1^n \mid m \geq 0, n \geq 0 \}, \)
- \( \{ a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0 \} \)
- \( \{ (^n)^n \mid n \geq 0 \} \)
- Set of all syntactically correct C programs
- \( \{ a^p \mid p \text{ is a prime number} \} \)
Regular Languages

A recursive definition

- $\emptyset$, $\{\varepsilon\}$, and $\{a\}$ are regular languages, where $a \in \Sigma$
- If $A$ and $B$ are regular languages, then the following are also regular languages:
  - $A \cup B$
  - $AB := \{ab \mid a \in A, b \in B\}$ (the concatenation of two strings in $A$ and $B$)
  - $A^*$

A few examples

- $\{a, b, ab\}$ Yes
- Any finite language Yes
- $\{0\}^*$ Yes
- $\{0, 1\}^*, \{a, b, c\}^*$ Yes
- $\{01^n0 \mid n \geq 0\}$ Yes
- Set of all positive integers in decimal representation Yes
- $\{0^m1^n \mid m \geq 0, n \geq 0\}$, Yes
- $\{a^k b^m c^n \mid k \geq 0, m \geq 0, n \geq 0\}$
- $\{(n)^n \mid n \geq 0\}$ No
- Set of all syntactically correct C programs No
- $\{a^p \mid p$ is a prime number$\}$ No
More concise notation to represent regular languages

- **∅**: the empty language
- **a**, for \( a \in \Sigma \): \{a\}
- **\( \varepsilon \)**: \{\( \varepsilon \)\}
- **\( R_1 | R_2 \)**: the union of the two languages \( L_1 \) and \( L_2 \) defined by regular expressions \( R_1 \) and \( R_2 \)
- **\( R_1 R_2 \)**: the concatenation of the two languages \( L_1 \) and \( L_2 \) defined by \( R_1 \) and \( R_2 \)
- **\( R^* \)**: the Kleene star of the language defined by \( R \)
Regular Expressions

More concise notation to represent regular languages

- \( \emptyset \): the empty language
- \( a \), for \( a \in \Sigma \): \( \{a\} \)
- \( \varepsilon \): \( \{\varepsilon\} \)
- \( R_1 | R_2 \): the union of the two languages \( L_1 \) and \( L_2 \) defined by regular expressions \( R_1 \) and \( R_2 \)
- \( R_1 R_2 \): the concatenation of the two languages \( L_1 \) and \( L_2 \) defined by \( R_1 \) and \( R_2 \)
- \( R^* \): the Kleene star of the language defined by \( R \)

Usual way to indicate grouping using parentheses

- \( ab | c \): \( \{ab, c\} \)
- \( a(b|c) \): \( \{ab, ac\} \)
Regular Expressions

More concise notation to represent regular languages

- \(\emptyset\): the empty language
- \(a\), for \(a \in \Sigma\): \(\{a\}\)
- \(\varepsilon\): \(\{\varepsilon\}\)
- \(R_1 | R_2\): the union of the two languages \(L_1\) and \(L_2\) defined by regular expressions \(R_1\) and \(R_2\)
- \(R_1 R_2\): the concatenation of the two languages \(L_1\) and \(L_2\) defined by \(R_1\) and \(R_2\)
- \(R^*\): the Kleene star of the language defined by \(R\)

Usual way to indicate grouping using parentheses

- \(ab|c\): \(\{ab, c\}\)
- \(a(b|c)\): \(\{ab, ac\}\)

Operator precedence

- Parentheses
- Kleene star (\(^*\))
- Concatenation
- Union (\(|\))
Examples of Regular Expressions

\[(0|1)^*0\]  all binary strings that end in 0

\[(0|1)00^*\]  all binary strings that start with 0 or 1, followed by one or more 0s

\[(0|1)^*\]  all binary strings
Examples of Regular Expressions

\[(0|1)^*0\] all binary strings that end in 0
\[(0|1)00^*\] all bin. strings that start with 0 or 1, followed by one or more 0s
\[(0|1)^*\] all binary strings

Which regular expression expresses the set of strings that do not contain 101 as a substring?
Examples of Regular Expressions

\[(0|1)^*0\] all binary strings that end in 0
\[(0|1)00^*\] all bin. strings that start with 0 or 1, followed by one or more 0s
\[(0|1)^*\] all binary strings

Which regular expression expresses the set of strings that do not contain 101 as a substring?

\[(0|\varepsilon)(1|000^*)^*(0|\varepsilon)\]
Applications of Regular Expressions

Two common operations

- Searching for patterns in a text
- Replacing text portions matching a pattern

Used in

- Text editors: emacs, vim, …
- System tools: shells, grep, lex, flex, sed, awk, …
- Programming languages: Perl, Ruby, Python, C/C++ (with regex library), Java (with regex package), …
No $\varepsilon$ nor $\emptyset$

- The empty string is represented as an empty string: $a(b\mid)$ as opposed to $a(b\mid\varepsilon)$ to denote the language $\{a, ab\}$
- Recognizing the empty language is not very useful in practice
Regular Expressions in Practice

No $\varepsilon$ nor $\emptyset$

- The empty string is represented as an empty string: $a(b\mid)$ as opposed to $a(b\mid\varepsilon)$ to denote the language \{a, ab\}
- Recognizing the empty language is not very useful in practice

Additional notation to make it easier to express common constructs

- $R^+ = RR^*$ (one or more repetitions of $R$; $R^* = \text{zero or more repetitions}$)
- $R? = (R\mid \ )$ (zero or one repetition of $R$)
- $R\{n\}$, $R\{,n\}$, $R\{m,n\}$, $R\{m,\}$ to denote exactly $n$, at most $n$, between $m$ and $n$, and at least $m$ repetitions of $R$ (supported e.g., in Perl)
Regular Expressions in Practice

No $\varepsilon$ nor $\emptyset$

- The empty string is represented as an empty string: $a(b|$) as opposed to $a(b|\varepsilon)$ to denote the language $\{a, ab\}$
- Recognizing the empty language is not very useful in practice

Additional notation to make it easier to express common constructs

- $R^+ = RR^*$ (one or more repetitions of $R$; $R^* =$ zero or more repetitions)
- $R? = (R|$) (zero or one repetition of $R$)
- $R\{n\}, R\{, n\}, R\{m, n\}, R\{m,\}$ to denote exactly $n$, at most $n$, between $m$ and $n$, and at least $m$ repetitions of $R$ (supported e.g., in Perl)

Some capabilities beyond regular languages

- Allow, for example, recognition of strings of the form $\alpha\beta\alpha$, were $\alpha, \beta \in \Sigma^*$
Regular Expressions in Practice: Character Classes (1)

Allow us to write tedious expressions such as a|b|c|\ldots|z more succinctly.

Examples

• “Recent” years:

199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))

→ 199[6-9]|20(0[0-9]|1[012])

• Identifier:

(a|b|\ldots|z|A|B|\ldots|Z|_)(a|b|\ldots|z|A|B|\ldots|Z|0|1|\ldots|9|_)*

→ [a-zA-Z_][a-zA-Z0-9_]*

• More examples:

– \[abc+-\] = (a|b|c|+|-)

– \[a-zA-Z\] = any letter

– \[^a-z\] = anything but a lowercase letter

– . = any character but newline

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Allow us to write tedious expressions such as $a|b|c|\ldots|z$ more succinctly.

Examples

- “Recent” years:
  
  $199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))$
Allow us to write tedious expressions such as $a|b|c|...|z$ more succinctly.

Examples

- “Recent” years:
  $$199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))$$

- Identifier:
  $$(a|b|...|z|A|B|...|Z|_)(a|b|...|z|A|B|...|Z|0|1|...|9|_)^*$$

Allow us to write tedious expressions such as $a \mid b \mid c \mid \ldots \mid z$ more succinctly.

Examples

- “Recent” years:
  
  $199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))$

  → $199[6-9]|20(0[0-9]|1[012])$

- Identifier:
  
  $(a|b|\ldots|z|A|B|\ldots|Z|\_)(a|b|\ldots|z|A|B|\ldots|Z|0|1|\ldots|9|\_)*$
Allow us to write tedious expressions such as $a|b|c|\ldots|z$ more succinctly.

**Examples**

- “Recent” years:
  \[
  199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))
  \]
  \[
  \rightarrow 199[6-9] | 20(0[0-9] | 1[012])
  \]

- Identifier:
  \[
  (a|b|\ldots|z|A|B|\ldots|Z|\_)(a|b|\ldots|z|A|B|\ldots|Z|0|1|\ldots|9|\_)*
  \]
  \[
  \rightarrow [a-zA-Z\_][a-zA-Z0-9\_]*
  \]
Allow us to write tedious expressions such as a|b|c|...|z more succinctly.

Examples

• “Recent” years:
  \[199(6|7|8|9)|20(0(0|1|2|3|4|5|6|7|8|9)|1(0|1|2))\]
  \[\rightarrow 199[6-9]|20(0[0-9]|1[012])\]

• Identifier:
  \[(a|b|...|z|A|B|...|Z|_)(a|b|...|z|A|B|...|Z|0|1|...|9|_)\]*
  \[\rightarrow [a-zA-Z_][a-zA-Z0-9_]\]*

• More examples:
  - [abc+-] = (a|b|c|+|-)
  - [a-zA-Z] = any letter
  - [^a-z] = anything but a lowercase letter
  - . = any character but newline
More examples

- `[]` matches `]`
- `[]-` matches `]` and `-`
- `[$~]` matches `$` and `~`
Regular Expressions in Practice: Character Classes (2)

More examples

- `[]` matches `[]`
- `[]-` matches `[]` and `-`
- `[$~]` matches `$` and `~`

Some additional classes available in Perl (and some other languages)

- `\d` matches any digit (`[0-9]`)  
- `\D` matches anything but a digit  
- `\s` matches any white space character (space, tab, newline)  
- `\S` matches anything but white space  
- `\w` matches any word character  
- `\W` matches anything but a word character
More examples

• [][] matches ]
• [ ]-[] matches ] and -
• [$^~] matches $ and ~

Some additional classes available in Perl (and some other languages)

• \d matches any digit ([0–9])
• \D matches anything but a digit
• \s matches any white space character (space, tab, newline)
• \S matches anything but white space
• \w matches any word character
• \W matches anything but a word character

Examples

• \d{3,10} 3–10 digits
• \d{1,3}(.\d{1,3}){3} an IP address
Anchors are used to match some characteristic positions between characters in the string.

- `^` matches at the beginning of the string
- `\$` matches at the end of the string
- `\b` matches a word boundary
Anchors are used to match some characteristic positions between characters in the string.

- `^` matches at the beginning of the string
- `$` matches at the end of the string
- `` matches a word boundary

Examples

- `^From:` matches email header line specifying the sender
- `#.*$` matches shell comment
Anchors are used to match some characteristic positions between characters in the string.

- `^` matches at the beginning of the string
- `$` matches at the end of the string
- `\b` matches a word boundary

**Examples**

- `^From:` matches email header line specifying the sender
- `#.*$` matches shell comment

Some languages (most notably Perl) provide a host of other types of anchors.
Back references match previously captured sub-expressions and thus allow us to express certain kinds of non-regular languages.

Examples

- \3 matches 3\textsuperscript{rd} parenthesized sub-expression
- (a*)b\1 matches the (non-regular) language \{a^nba^n \mid n \geq 0\}
Login to a UNIX box and type

- `man regex` ... to learn about regular expression support in the C library,
- `man perlre` ... to learn more about Perl regular expressions.
History of Regular Languages and Regular Expressions

Theory of regular languages (regular sets)


Used mathematical notion of *regular sets* to describe models of the nervous system by McCulloch and Pitts (1940s) as small simple automata.

Implementation

- Stephen Kleene (1956)
- Ken Thompson (1968): Editor QED and later ed and grep
  - Patented algorithm
  - Also used in awk, vi, lex, emacs
- Henry Spencer (1986): C regular expression library used in tcl
  - Also used in Perl
Deterministic Finite Automata (DFA)

A deterministic finite automaton (DFA) is a simple machine that accepts or rejects a given input string.

This defines a formal language: the set of strings accepted by the DFA. We say the DFA recognizes this language.
Deterministic Finite Automata (DFA)

A deterministic finite automaton (DFA) is a simple machine that accepts or rejects a given input string.

This defines a formal language: the set of strings accepted by the DFA. We say the DFA recognizes this language.

We will show that a language is regular if and only if there exists a DFA that recognizes this language.
Deterministic Finite Automata (DFA)

A **deterministic finite automaton (DFA)** is a simple machine that **accepts** or **rejects** a given input string.

This defines a formal language: the set of strings accepted by the DFA. We say the DFA **recognizes** this language.

We will show that a language is regular if and only if there exists a DFA that recognizes this language.

**Informal definition of a DFA**

- Finite set of **states**
- Designated **start state**
- Designated set of **final states**
- Reads input one symbol at a time. New state is calculated as a function of current state and read symbol.
- String is accepted if and only if a final state is reached after reading the entire string.
Example of a DFA

Transition function:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td>( q_1 )</td>
<td>( q_1 )</td>
<td>( q_2 )</td>
</tr>
<tr>
<td>Final:</td>
<td>( q_2 )</td>
<td>( q_1 )</td>
<td>( q_2 )</td>
</tr>
</tbody>
</table>

\( q_1 \) and \( q_2 \) are states.

The diagram shows:
- \( q_1 \) starts
- Transitions: \( 0 \) goes to \( q_1 \), \( 1 \) goes to \( q_2 \)
- \( q_2 \) is a final state
Example of a DFA

Transition function:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>Start</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What language does this DFA recognize?
Example of a DFA

What language does this DFA recognize?

\[(0 | 1)^* 1\]
Another Example of a DFA

What language does this DFA recognize?
Another Example of a DFA

What language does this DFA recognize?

\[ a(b^*a)^* | b(a*b)^* \]
Formal Definition of a DFA

A DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set of states
- \(\Sigma\) is a finite set of input symbols (i.e., an alphabet)
- \(\delta : Q \times \Sigma \rightarrow Q\) is a transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final or accepting states
A DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set of \textit{states}
- \(\Sigma\) is a finite set of \textit{input symbols} (i.e., an \textit{alphabet})
- \(\delta : Q \times \Sigma \rightarrow Q\) is a \textit{transition function}
- \(q_0 \in Q\) is the \textit{start state}
- \(F \subseteq Q\) is the set of \textit{final} or \textit{accepting states}

The DFA \textit{accepts} a string \(\sigma \in \Sigma^*\) if and only if it reaches a final state after consuming all letters in \(\sigma\).
Formal Definition of a DFA

A DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set of \textit{states}
- \(\Sigma\) is a finite set of \textit{input symbols} (i.e., an \textit{alphabet})
- \(\delta : Q \times \Sigma \rightarrow Q\) is a \textit{transition function}
- \(q_0 \in Q\) is the \textit{start state}
- \(F \subseteq Q\) is the set of \textit{final} or \textit{accepting states}

The DFA \textit{accepts} a string \(\sigma \in \Sigma^*\) if and only if it reaches a final state after consuming all letters in \(\sigma\).

The language \textit{recognized} by a DFA \(D\) is the set of strings \(\mathcal{L}(D) := \{\sigma \in \Sigma^* \mid D\ \text{accepts}\ \sigma\}\).
Formal Definition of a DFA

A DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set of states
- \(\Sigma\) is a finite set of input symbols (i.e., an alphabet)
- \(\delta : Q \times \Sigma \rightarrow Q\) is a transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final or accepting states

The DFA accepts a string \(\sigma \in \Sigma^*\) if and only if it reaches a final state after consuming all letters in \(\sigma\).

The language recognized by a DFA \(D\) is the set of strings \(\mathcal{L}(D) := \{\sigma \in \Sigma^* \mid D\) accepts \(\sigma\}\).

We will prove that a language \(\mathcal{L}\) is regular if and only if there exists a DFA \(D\) such that \(\mathcal{L} = \mathcal{L}(D)\).
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \}
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

```
I have seen an even number of 0s
q1 ----> q2
0 ----> 0
1 ----> 1
```

I have seen an odd number of 0s
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

• \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

• \( \{ \sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring } 101 \} \)
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

• \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

• \( \{ \sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring } 101 \} \)
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

- \( \{ \sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring } 101 \} \)
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- $\{\sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even}\}$

- $\{\sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring 101}\}$
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

```
0 1
q1 -- > q2
| 0 |
```

- \( \{ \sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring 101} \} \)

```
0 1
q1 ----> q2 ----> q3
| 0 |
```

- Valid C comments
Examples of DFAs (1)

Construct DFAs that recognize the following languages:

- \( \{ \sigma \in \{0, 1\}^* \mid \text{the number of 0s in } \sigma \text{ is even} \} \)

- \( \{ \sigma \in \{0, 1\}^* \mid \sigma \text{ does not contain the substring 101} \} \)

- Valid C comments
Examples of DFAs (2)

- Binary numbers divisible by 3
Examples of DFAs (2)

- Binary numbers divisible by 3
Examples of DFAs (3)

• \( \mathcal{L} \subseteq \{0, 1\}^* \) defined by the Perl regular expression /.*1../
Examples of DFAs (3)

- $L \subseteq \{0, 1\}^*$ defined by the Perl regular expression /.*1../
Examples of DFAs (3)

- \( \mathcal{L} \subseteq \{0, 1\}^* \) defined by the Perl regular expression / .*1 . . /
Non-Deterministic Finite Automata (NFA)

A DFA is *deterministic* in the sense that every input traces exactly one path through the automaton.

A *non-deterministic finite automaton* (NFA) is identical to a DFA, except that there are possibly many paths traced by an input.
Non-Deterministic Finite Automata (NFA)

A DFA is *deterministic* in the sense that every input traces exactly one path through the automaton.

A *non-deterministic finite automaton* (NFA) is identical to a DFA, except that there are possibly many paths traced by an input.

**Two sources of non-determinism**

- \(\epsilon\)-transitions

  ![Graph](image1)

- Multiple successor states for the same input symbol

  ![Graph](image2)
Non-Deterministic Finite Automata (NFA)

A DFA is *deterministic* in the sense that every input traces exactly one path through the automaton.

A *non-deterministic finite automaton* (NFA) is identical to a DFA, except that there are possibly many paths traced by an input.

**Two sources of non-determinism**

- $\epsilon$-transitions

  ![epsilon_transition_diagram]

- Multiple successor states for the same input symbol

  ![multiple_successor_diagram]

An NFA *accepts* a string $\sigma \in \Sigma^*$ if one of the paths traced by $\sigma$ ends in an accepting state.
Example of an NFA

Transition function:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>${q_1}$</td>
<td>${q_1, q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>1</td>
<td>${q_3}$</td>
<td>$\emptyset$</td>
<td>${q_3}$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$\emptyset$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Final:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_4^*$</td>
<td>0</td>
<td>${q_4}$</td>
<td>${q_4}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Example of an NFA

Transition function:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
<th>ε</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₁</td>
<td>0/1</td>
<td>{q₁}</td>
<td>{q₁,q₂}</td>
<td>0</td>
</tr>
<tr>
<td>q₂</td>
<td>1</td>
<td>{q₃}</td>
<td>0</td>
<td>{q₃}</td>
</tr>
<tr>
<td>q₃</td>
<td>0</td>
<td>0</td>
<td>{q₄}</td>
<td>0</td>
</tr>
<tr>
<td>q₄*</td>
<td>0</td>
<td>0</td>
<td>{q₄}</td>
<td>0</td>
</tr>
</tbody>
</table>

What language does this NFA recognize?
Example of an NFA

Transition function:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_1, q_2$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>0</td>
<td></td>
<td>$q_3$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0</td>
<td>$q_4$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Final:</td>
<td>$q_4^*$</td>
<td>$q_4$</td>
<td>$q_4$</td>
<td>0</td>
</tr>
</tbody>
</table>

What language does this NFA recognize?

$$(0|1)*10?1(0|1)*$$
Formal Definition of an NFA

An NFA is a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \)

- \( Q \) is a finite set of \textit{states}
- \( \Sigma \) is a finite set of \textit{input symbols} (i.e., an \textit{alphabet})
- \( \delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q \) is a \textit{transition function}
- \( q_0 \in Q \) is the \textit{start state}
- \( F \subseteq Q \) is the set of \textit{final} or \textit{accepting states}

Every input \( \sigma \in \Sigma^* \) defines a set of paths traced by \( \delta \) while consuming \( \sigma \). The NFA \textit{accepts} \( \sigma \) if and only if one such path defined by \( \sigma \) ends in a final states.

The language \textit{recognized} by an NFA \( D \) is the set of strings
\[ \mathcal{L}(D) := \{ \sigma \in \Sigma^* \mid D \text{ accepts } \sigma \}. \]
Formal Definition of an NFA

An NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

- \(Q\) is a finite set of states
- \(\Sigma\) is a finite set of input symbols (i.e., an alphabet)
- \(\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q\) is a transition function
- \(q_0 \in Q\) is the start state
- \(F \subseteq Q\) is the set of final or accepting states

Every input \(\sigma \in \Sigma^*\) defines a set of paths traced by \(\delta\) while consuming \(\sigma\). The NFA accepts \(\sigma\) if and only if one such path defined by \(\sigma\) ends in a final states.

The language recognized by an NFA \(D\) is the set of strings 
\[ \mathcal{L}(D) := \{\sigma \in \Sigma^* \mid D\ \text{accepts}\ \sigma\}. \]
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language $L$ recognized by an NFA that cannot be recognized by a DFA?
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language $\mathcal{L}$ recognized by an NFA that cannot be recognized by a DFA?

No! We will prove this.
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language \( L \) recognized by an NFA that cannot be recognized by a DFA?

**No!** We will prove this.

Is it easier to construct an NFA for a regular language than to construct a DFA for the same language?
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language $\mathcal{L}$ recognized by an NFA that cannot be recognized by a DFA?

No! We will prove this.

Is it easier to construct an NFA for a regular language than to construct a DFA for the same language?

• $\mathcal{L} \subseteq \{0, 1\}^*$ defined by the Perl regular expression /.*1../
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language \( \mathcal{L} \) recognized by an NFA that cannot be recognized by a DFA?

**No!** We will prove this.

Is it easier to construct an NFA for a regular language than to construct a DFA for the same language?

- \( \mathcal{L} \subseteq \{0, 1\}^* \) defined by the Perl regular expression /.*1../

---

**DFA**

![DFA Diagram]

---

**Lexical Analysis and Automata Theory**

CSCI 3136: Principles of Programming Languages
Are NFAs More Powerful Than DFAs?

The answer depends on what we mean by this question.

Is there a language $\mathcal{L}$ recognized by an NFA that cannot be recognized by a DFA?

No! We will prove this.

Is it easier to construct an NFA for a regular language than to construct a DFA for the same language?

- $\mathcal{L} \subseteq \{0, 1\}^*$ defined by the Perl regular expression / .*1../
Theorem: The following statements are equivalent:

- $\mathcal{L}$ is a regular language.
- $\mathcal{L}$ is the language described by a regular expression (without extensions as, e.g., in Perl).
- $\mathcal{L}$ is recognized by an NFA.
- $\mathcal{L}$ is recognized by a DFA.
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$AB$</td>
<td>$\Rightarrow$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$\Rightarrow$</td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td>$A \cup B$</td>
</tr>
<tr>
<td>$AB$</td>
<td>$AB$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$A^*$</td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td></td>
</tr>
<tr>
<td>$A \cup B$</td>
<td></td>
</tr>
<tr>
<td>$AB$</td>
<td></td>
</tr>
<tr>
<td>$A^*$</td>
<td></td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td></td>
</tr>
<tr>
<td>$AB$</td>
<td></td>
</tr>
<tr>
<td>$A^*$</td>
<td></td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>$\Rightarrow$</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\Rightarrow$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\Rightarrow$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$\Rightarrow$</td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td>$\Rightarrow$</td>
<td>$R_A \mid R_B$</td>
</tr>
<tr>
<td>$AB$</td>
<td>$\Rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$A^*$</td>
<td>$\Rightarrow$</td>
<td></td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>$\Rightarrow$</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td></td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td></td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td></td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td></td>
<td>$R_A</td>
</tr>
<tr>
<td>$AB$</td>
<td></td>
<td>$R_AR_B$</td>
</tr>
<tr>
<td>$A^*$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Regular Languages and Regular Expressions

A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>$\Rightarrow$</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\Rightarrow$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\Rightarrow$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$\Rightarrow$</td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td>$\Rightarrow$</td>
<td>$R_A</td>
</tr>
<tr>
<td>$AB$</td>
<td>$\Rightarrow$</td>
<td>$R_AR_B$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$\Rightarrow$</td>
<td>$R_A^*$</td>
</tr>
</tbody>
</table>
A language $\mathcal{L}$ is regular if and only if it can be expressed using a regular expression.

<table>
<thead>
<tr>
<th>Regular language</th>
<th>Regular expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${\varepsilon}$</td>
<td>$\varepsilon$</td>
</tr>
<tr>
<td>${a}, a \in \Sigma$</td>
<td>$a$</td>
</tr>
<tr>
<td>$A \cup B$</td>
<td>$R_A \mid R_B$</td>
</tr>
<tr>
<td>$AB$</td>
<td>$R_A R_B$</td>
</tr>
<tr>
<td>$A^*$</td>
<td>$R_A^*$</td>
</tr>
</tbody>
</table>
If a language $\mathcal{L}$ can be expressed using a regular expression, there exists an NFA that recognizes it.

\[
\begin{array}{c|c}
\text{Regular expression} & \text{NFA} \\
\hline
\emptyset & \rightarrow \\
\varepsilon & \rightarrow \\
a, a \in \Sigma & \rightarrow \\
\end{array}
\]
If a language $\mathcal{L}$ can be expressed using a regular expression, there exists an NFA that recognizes it.

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>$a, a \in \Sigma$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>
If a language $\mathcal{L}$ can be expressed using a regular expression, there exists an NFA that recognizes it.

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$a, a \in \Sigma$</td>
<td>$q_1$</td>
</tr>
</tbody>
</table>
From Regular Expression to NFA (1)

If a language $\mathcal{L}$ can be expressed using a regular expression, there exists an NFA that recognizes it.

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>NFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$a, a \in \Sigma$</td>
<td>$q_1 \xrightarrow{a} q_2$</td>
</tr>
</tbody>
</table>
From Regular Expression to NFA (2)

Regular expression             NFA

$R_A \mid R_B$  $\Rightarrow$  $M_A$

$R_A R_B$  $\Rightarrow$  $M_B$

$R_A^*$  $\Rightarrow$
From Regular Expression to NFA (2)

Regular expression  \[ R_A | R_B \]  \[ R_A R_B \]  \[ R_A^* \]  \[ \Rightarrow \]  \[ \Rightarrow \]  \[ \Rightarrow \]

NFA

\[ M_A \]

\[ M_B \]

\[ \varepsilon \]
From Regular Expression to NFA (2)

Regular expression

\[ R_A \mid R_B \]

\[ R_A R_B \]

\[ R_A^* \]
From Regular Expression to NFA (2)

Regular expression

\[ R_A | R_B \]  \quad \Rightarrow \quad \epsilon

\[ R_A R_B \]  \quad \Rightarrow \quad M_A \quad \epsilon \quad M_B \quad \epsilon

\[ R_A^* \]  \quad \Rightarrow \quad M_A \quad \epsilon \quad M_B \quad \epsilon
From Regular Expression to NFA (2)

Regular expression

\[ R_A \mid R_B \]

\[ R_A R_B \]

\[ R_A^* \]
From Regular Expression to NFA (2)

Regular expression

$R_A | R_B \quad \Rightarrow \quad M_A \quad M_B$

$R_A R_B \quad \Rightarrow \quad M_A \quad M_B$

$R_A^* \quad \Rightarrow \quad M_A$

NFA
If a language $\mathcal{L}$ can be recognized by an NFA, it can be expressed using a regular expression.
From NFA to Regular Expression (1)

If a language $\mathcal{L}$ can be recognized by an NFA, it can be expressed using a regular expression.

A generalized NFA (GNFA)

- Edges are labelled with regular expressions.
- When in configuration $(q_1, \alpha\beta)$ (state $q_1$ with input $\alpha\beta$ still left to be read), we can transition into configuration $(q_2, \beta)$ (state $q_2$ with input $\beta$ still left to be read) if and only if the edge $q_1 \rightarrow q_2$ is labelled with a regular expression that matches $\alpha$. 
From NFA to Regular Expression (1)

If a language \( \mathcal{L} \) can be recognized by an NFA, it can be expressed using a regular expression.

A generalized NFA (GNFA)

- Edges are labelled with regular expressions.
- When in configuration \((q_1, \alpha\beta)\) (state \(q_1\) with input \(\alpha\beta\) still left to be read), we can transition into configuration \((q_2, \beta)\) (state \(q_2\) with input \(\beta\) still left to be read) if and only if the edge \(q_1 \rightarrow q_2\) is labelled with a regular expression that matches \(\alpha\).

**Proof idea:** NFA \(\Rightarrow\) GNFA\(_1\) \(\Rightarrow\) GNFA\(_2\) \(\Rightarrow\) \(\cdots\) \(\Rightarrow\) GNFA\(_k\) \(\Rightarrow\) regular expression
From NFA to Regular Expression (1)

If a language $\mathcal{L}$ can be recognized by an NFA, it can be expressed using a regular expression.

A generalized NFA (GNFA)

- Edges are labelled with regular expressions.
- When in configuration $(q_1, \alpha\beta)$ (state $q_1$ with input $\alpha\beta$ still left to be read), we can transition into configuration $(q_2, \beta)$ (state $q_2$ with input $\beta$ still left to be read) if and only if the edge $q_1 \rightarrow q_2$ is labelled with a regular expression that matches $\alpha$.

Proof idea: $\text{NFA} \Rightarrow \text{GNFA}_1 \Rightarrow \text{GNFA}_2 \Rightarrow \cdots \Rightarrow \text{GNFA}_k \Rightarrow \text{regular expression}$

An NFA is a GNFA. “Normalize” it by adding $\epsilon$-transitions from its final states to a new final state $f$ and making all original final states non-final.
From GNFA to Regular expression

- Transform GNFA into equivalent GNFA with only two states: start state and final state
- Transform this two-state GNFA into a regular expression
From GNFA to Regular expression

- Transform GNFA into equivalent GNFA with only two states: start state and final state
- Transform this two-state GNFA into a regular expression

Extract regular expression

\[ R_1^*R_2(R_3R_1^*R_2 | R_4)^* \]
State reduction

This may create loops because some states may simultaneously be in- and out-neighbours of $s$. 

$$S \Rightarrow s q_i r_j S^* R_j$$
If a language $\mathcal{L}$ can be recognized by a DFA, it can be recognized by an NFA.
If a language $L$ can be recognized by a DFA, it can be recognized by an NFA.

A DFA is an NFA!
If a language $\mathcal{L}$ can be recognized by an NFA, it can be recognized by a DFA.

**Proof idea:** Construct a DFA whose states represent the sets of states the NFA may be in at any given point in time.
If a language \( \mathcal{L} \) can be recognized by an NFA, it can be recognized by a DFA.

**Proof idea:** Construct a DFA whose states represent the sets of states the NFA may be in at any given point in time.

**Start state**

- Before consuming any input, the NFA can be in its start state \( q_0 \) or in any state that can be reached from \( q_0 \) using a sequence of \( \epsilon \)-transitions.
- We call this the **\( \epsilon \)-closure** \( \text{ECLOSE}(q_0) \) of \( q_0 \).
- \( \text{ECLOSE}(q_0) \) is the start state of the DFA.
Transition function and construction of more DFA states

- Assume that after reading some input, the NFA can be in any of the states in a set $Q$ represented by a DFA state. Which states can the NFA be in after reading an input symbol $a$?

$Q'': = \bigcup q' \in Q' \text{ECLOSE}(q')$, where $Q': = \bigcup q \in Q \delta(q, a)$.

- If $Q''$ is not a state of the DFA yet, we add it to the set of DFA states.
- We define $\delta'(Q, a): = Q''$.

- We continue to inspect all DFA state-symbol pairs until we do not discover any new states.
From NFA to DFA (2)

Transition function and construction of more DFA states

- Assume that after reading some input, the NFA can be in any of the states in a set $Q$ represented by a DFA state. Which states can the NFA be in after reading an input symbol $a$?

$$Q'' := \bigcup_{q' \in Q'} \text{ECLOSE}(q'), \text{ where } Q' := \bigcup_{q \in Q} \delta(q, a)$$
Transition function and construction of more DFA states

- Assume that after reading some input, the NFA can be in any of the states in a set $Q$ represented by a DFA state. Which states can the NFA be in after reading an input symbol $a$?

$$Q'' := \bigcup_{q' \in Q'} \text{ECLOSE}(q'), \text{ where } Q' := \bigcup_{q \in Q} \delta(q, a)$$

- If $Q''$ is not a state of the DFA yet, we add it to the set of DFA states.
- We define $\delta'(Q, a) := Q''$.
- We continue to inspect all DFA state-symbol pairs until we do not discover any new states.
From NFA to DFA (2)

Transition function and construction of more DFA states

- Assume that after reading some input, the NFA can be in any of the states in a set $Q$ represented by a DFA state. Which states can the NFA be in after reading an input symbol $a$?

$$Q'' := \bigcup_{q' \in Q'} \text{ECLOSE}(q'), \text{ where } Q' := \bigcup_{q \in Q} \delta(q, a)$$

- If $Q''$ is not a state of the DFA yet, we add it to the set of DFA states.
- We define $\delta'(Q, a) := Q''$.
- We continue to inspect all DFA state-symbol pairs until we do not discover any new states.

Accepting states
From NFA to DFA (2)

Transition function and construction of more DFA states

- Assume that after reading some input, the NFA can be in any of the states in a set $Q$ represented by a DFA state. Which states can the NFA be in after reading an input symbol $a$?

$$Q'' := \bigcup_{q' \in Q'} \text{ECLOSE}(q'), \text{ where } Q' := \bigcup_{q \in Q} \delta(q, a)$$

- If $Q''$ is not a state of the DFA yet, we add it to the set of DFA states.
- We define $\delta'(Q, a) := Q''$.
- We continue to inspect all DFA state-symbol pairs until we do not discover any new states.

Accepting states

- A state $Q$ of the DFA is accepting if, viewed as a set of NFA states, it contains an accepting state of the NFA.
From NFA to DFA: The Worst Case

In the worst case, the construction may turn an NFA with $n$ states into a DFA with $2^n$ states (every possible subset of NFA states becomes a DFA state).

Example: /.*1.{n}/
From NFA to DFA: The Worst Case

In the worst case, the construction may turn an NFA with \( n \) states into a DFA with \( 2^n \) states (every possible subset of NFA states becomes a DFA state).

Example: \( /.*1.{n}/ \)

In practice, the worst case usually does not arise.
From NFA to DFA: Example

Regular expression:
01(00|11)*10

```
<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q1}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
```
From NFA to DFA: Example

Regular expression:
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_1}</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Final:

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

![NFA to DFA Diagram]

<table>
<thead>
<tr>
<th>Symbol</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{q_1}</td>
</tr>
<tr>
<td>1</td>
<td>{q_2}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression:
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_1}</td>
<td>0</td>
</tr>
<tr>
<td>{q_2}</td>
<td>{q_2}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>1</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression:
\[01(00|11)^*10\]

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_2}</td>
<td>{q_3}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_7}</td>
<td>{q_8}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_2}</td>
<td>{q_3}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_7}</td>
<td>{q_8}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_2}</td>
<td>{q_3}</td>
<td>{q_9}</td>
</tr>
<tr>
<td>{q_7}</td>
<td>{q_8}</td>
<td>{q_9}</td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

### Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td>{q_3}</td>
<td>{q_7}</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
<tr>
<td>{q_7}</td>
<td>{q_8}</td>
<td>{q_9}</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression: 01(00|11)*10

Start:  
{q₁}  
{q₂}  
∅  
{q₃, q₄, q₇, q₁₀}

Symbol  
0  
{q₂}  
∅  
{q₃, q₄, q₇, q₁₀}

1  
∅  
{q₃, q₄, q₇, q₁₀}
From NFA to DFA: Example

Regular expression: 01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\emptyset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
## From NFA to DFA: Example

### Regular expression:
\[01(00|11)^*10\]

### NFA Diagram:
![NFA Diagram]

### State Transition Table:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td></td>
<td>{q_2}</td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>\emptyset</td>
<td></td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td></td>
<td></td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
</tbody>
</table>

### DFA Diagram:
![DFA Diagram]
From NFA to DFA: Example

Regular expression: $01(00|11)^*10$

### State Transition Table

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start: ${q_1}$</td>
<td></td>
<td>${q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_2}$</td>
<td></td>
<td>$\emptyset$</td>
<td>${q_3,q_4,q_7,q_{10}}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_3,q_4,q_7,q_{10}}$</td>
<td></td>
<td>${q_{11}}$</td>
<td>${q_5,q_8}$</td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory

CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

State transitions:

<table>
<thead>
<tr>
<th>Start</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q₁}</td>
<td>{q₂}</td>
<td>∅</td>
</tr>
<tr>
<td>{q₂}</td>
<td>0</td>
<td>{q₃,q₄,q₇,q₁₀}</td>
</tr>
<tr>
<td>∅</td>
<td>0</td>
<td>{q₁₁}</td>
</tr>
<tr>
<td>{q₃,q₄,q₇,q₁₀}</td>
<td>{q₁₁}</td>
<td>{q₅,q₈}</td>
</tr>
<tr>
<td>{q₁₁}</td>
<td>{q₁₁}</td>
<td>{q₅,q₈}</td>
</tr>
<tr>
<td>{q₅,q₈}</td>
<td>{q₅,q₈}</td>
<td>{q₅,q₈}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression:
$01(00|11)^*10$

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start: ${q_1}$</td>
<td></td>
<td>${q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_2}$</td>
<td></td>
<td>$\emptyset$</td>
<td>${q_3,q_4,q_7,q_{10}}$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td></td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_3,q_4,q_7,q_{10}}$</td>
<td></td>
<td>${q_{11}}$</td>
<td>${q_5,q_8}$</td>
</tr>
<tr>
<td>${q_{11}}$</td>
<td></td>
<td>${q_{11}}$</td>
<td>${q_3,q_4,q_7,q_{10},q_{12}}$</td>
</tr>
<tr>
<td>${q_5,q_8}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression:
01(00|11)*10

![NFA Diagram]

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_2}</td>
<td>\emptyset</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>{q_3}</td>
<td>{q_5, q_8}</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td>{q_{11}}</td>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
</tr>
<tr>
<td>{q_{11}}</td>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
<td></td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_6}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

State transition diagram:

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_{11}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression: $01(00|11)^*10$

![Diagram of NFA and DFA transition graph]

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>${q_1}$</td>
<td>${q_2}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_2}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_3, q_4, q_7, q_{10}}$</td>
<td>${q_{11}}$</td>
<td></td>
</tr>
<tr>
<td>${q_{11}}$</td>
<td>${q_3, q_4, q_7, q_{10}}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_5, q_8}$</td>
<td>${q_6}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_3, q_4, q_7, q_{10}, q_{12}}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression: 
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_2}</td>
<td>\emptyset</td>
<td>{q_3,q_4,q_7,q_{10}}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>{q_{11}}</td>
<td>{q_5,q_8}</td>
</tr>
<tr>
<td>{q_3,q_4,q_7,q_{10}}</td>
<td>{q_{11}}</td>
<td>{q_5,q_8}</td>
</tr>
<tr>
<td>{q_{11}}</td>
<td>{q_6}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_5,q_8}</td>
<td>{q_{11}}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_3,q_4,q_7,q_{10},q_{12}}</td>
<td>{q_6}</td>
<td>{q_3,q_4,q_7,q_9,q_{10}}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression:
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_2}</td>
<td>\emptyset</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>{q_5, q_8}</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td>{q_{11}}</td>
<td>{q_5, q_8}</td>
</tr>
<tr>
<td>{q_{11}}</td>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td>{q_6}</td>
<td>{q_3, q_4, q_7, q_9, q_{10}}</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
<td>{q_6}</td>
<td>{q_3, q_4, q_7, q_9, q_{10}}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression: 01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>{q_1} {q_2} \emptyset {q_3,q_4,q_7,q_{10}} {q_{11}} {q_5,q_8} {q_3,q_4,q_7,q_{10},q_{12}} {q_6} {q_3,q_4,q_7,q_9,q_{10}} {q_3,q_4,q_7,q_{12}} {q_6^*} {q_3,q_4,q_7,q_{10}}</td>
<td>{q_2} \emptyset {q_3,q_4,q_7,q_{10}} {q_{11}} {q_5,q_8} {q_3,q_4,q_7,q_{10},q_{12}} {q_6} {q_3,q_4,q_7,q_{10}} {q_3,q_4,q_7,q_{12}} {q_6^*} {q_3,q_4,q_7,q_{10}}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression: $01(00|11)^*10$

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start: {q_1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>∅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_10}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_11}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_10, q_12}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_6^*}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_9, q_10}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q₁}</td>
<td>0</td>
</tr>
<tr>
<td>{q₂}</td>
<td>0</td>
</tr>
<tr>
<td>∅</td>
<td>1</td>
</tr>
<tr>
<td>{q₃, q₄, q₇, q₁₀}</td>
<td>{q₂}</td>
</tr>
<tr>
<td>{q₁₁}</td>
<td></td>
</tr>
<tr>
<td>{q₅, q₈}</td>
<td>∅</td>
</tr>
<tr>
<td>{q₃, q₄, q₇, q₁₀, q₁₂}</td>
<td></td>
</tr>
<tr>
<td>{q₆}</td>
<td></td>
</tr>
<tr>
<td>{q₃, q₄, q₇, q₉, q₁₀}</td>
<td></td>
</tr>
<tr>
<td>{q₆*}</td>
<td></td>
</tr>
<tr>
<td>{q₃, q₄, q₇, q₉, q₁₀}</td>
<td></td>
</tr>
</tbody>
</table>

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
From NFA to DFA: Example

Regular expression:
01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol 0</th>
<th>Symbol 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td>{q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_2}</td>
<td>\emptyset</td>
<td>{q_3, q_4, q_7, q_{10}}</td>
</tr>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td>{q_{11}}</td>
<td>{q_5, q_8}</td>
</tr>
<tr>
<td>{q_{11}}</td>
<td>{q_{3}, q_4, q_7, q_{10}, q_{12}}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td>{q_6}</td>
<td>{q_3, q_4, q_7, q_{9}, q_{10}}</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}, q_{12}}</td>
<td>{q_{11}}</td>
<td>{q_5, q_8}</td>
</tr>
<tr>
<td>Final:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_{6}^*}</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{9}, q_{10}}</td>
<td>{q_{11}}</td>
<td>{q_5, q_8}</td>
</tr>
</tbody>
</table>
From NFA to DFA: Example

Regular expression: 01(00|11)*10

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_1}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_2}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\emptyset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_{10}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_{11}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_5, q_8}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_9, q_{10}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_6^*}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_3, q_4, q_7, q_9, q_{10}}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
From NFA to DFA: Exercise

What language is accepted by the following NFA?
From NFA to DFA: Exercise

What language is accepted by the following NFA?

01(00|11)*10
From NFA to DFA: Exercise

What language is accepted by the following NFA?

01(00|11)*10

Convert it to a DFA.
From NFA to DFA: Exercise

What language is accepted by the following NFA?

$01(00|11)^*10$

Convert it to a DFA.
Scanning

A *scanner* produces a token (token type, value) stream from a character stream.
Scanning

A *scanner* produces a token (token type, value) stream from a character stream.

**Two modes of operation**

- Complete pass produces the token stream, which is then passed to the parser.
- Parser calls scanner to request next token.
A **scanner** produces a token (token type, value) stream from a character stream.

**Two modes of operation**

- Complete pass produces the token stream, which is then passed to the parser.
- Parser calls scanner to request next token.

In either case, the scanner always recognizes the longest possible token.
Scanning

A *scanner* produces a token (token type, value) stream from a character stream.

Two modes of operation

- Complete pass produces the token stream, which is then passed to the parser.
- Parser calls scanner to request next token.

In either case, the scanner always recognizes the longest possible token.

Scanner implementation

- Hand-written, ad-hoc. Usually done when speed is a concern.
- From regular expression using scanner generator. More convenient.

Result:

- Case statements representing transitions of the DFA.
- Table representing the DFA’s transition function plus driver code to implement the DFA.
Regular expression → NFA → DFA → minimized DFA
Building a Scanner

Regular expression → NFA → DFA → minimized DFA

Extensions to pure DFAs:

- Accepting a token is not enough. Need to know which token was accepted and its value.
  - One accepting state per token type
  - Return string read along the path to the accepting state
- Keywords are not identifiers
  - Look up identifier in keyword table (e.g., hash table) to see whether it is in fact a keyword
- “Look-ahead” to distinguish tokens with common prefix (e.g., 100 vs 100.5)
  - Try to find the longest possible match by continuing to scan from an accepting state.
  - Backtrack to last accepting state when “stuck”.

Regular expression → NFA → DFA → minimized DFA
Extended Example of a Scanner

An incomplete scanner for Pascal

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Extended Example of a Scanner

An incomplete scanner for Pascal

Input: 2.10]
Token: real

Input: 2..10]
Token: int
Minimizing the DFA

Idea: Group states into classes of equivalent states (accepting/non-accepting, same transitions)

Procedure

- Initially, start with two equivalence classes: accepting and non-accepting states.
- Find an equivalence class $C$ and a letter $a$ such that, upon reading $a$, the states in $C$ transition to states in $k > 1$ equivalence classes $C'_1, C'_2, \ldots, C'_k$. Partition $C$ into subclasses $C_1, C_2, \ldots, C_k$ such that, upon reading $a$, the states in $C_i$ transition to states in $C'_i$.
- Repeat until no such “partitionable” equivalence class $C$ can be found.
- Final set of equivalence classes is the state set of the minimized DFA.
Minimizing the DFA: Example

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Minimizing the DFA: Example

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Minimizing the DFA: Example
Minimizing the DFA: Example

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Minimizing the DFA: Example

$q_1$  0  $q_1$  1  $q_2$
$q_3$  0  $q_2$  1  $q_4$
$q_1$  0  $q_3$  1  $q_4$
$q_5$  0  $q_4$  1  $q_4$
$q_6$  0  $q_5$  1  $q_4$
$q_6$  0  $q_6$  1  $q_4$

$0$  0  $1$  0  $0$
Minimizing the DFA: Example

The diagram illustrates a DFA with states $q_1, q_2, q_3, q_4, q_5, q_6$. The transitions are labeled with input symbols 0 and 1.
Minimizing the DFA: Example

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Minimizing the DFA: Example
Minimizing the DFA: Example

Lexical Analysis and Automata Theory
CSCI 3136: Principles of Programming Languages
Constructing a Scanner: Example (1)

Language: Strings of 0s and 1s containing an even number of 0s.
Language: Strings of 0s and 1s containing an even number of 0s.

Regular expression:
Language: Strings of 0s and 1s containing an even number of 0s.

Regular expression: \((1*01*0)^*1*\)
Constructing a Scanner: Example (1)

Language: Strings of 0s and 1s containing an even number of 0s.

Regular expression: \((1*01*0)^*1^*\)

NFA
Constructing a Scanner: Example (1)

Language: Strings of 0s and 1s containing an even number of 0s.

Regular expression: \((1*01*0)*1*\)

NFA
Constructing a Scanner: Example (2)

DFA
Constructing a Scanner: Example (2)

DFA
Constructing a Scanner: Example (2)

DFA

Minimized DFA
Constructing a Scanner: Example (2)

DFA

Minimized DFA
Constructing a Scanner: Example (2)

DFA

Minimized DFA
How General Are Regular Languages? (1)

Some properties: If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
How General Are Regular Languages? (1)

Some properties: If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
  - By definition
How General Are Regular Languages? (1)

Some properties: If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
  - By definition
- $\Sigma^* \setminus R$ (the complement of $R$)
How General Are Regular Languages? (1)

Some properties: If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
  - By definition

- $\Sigma^* \setminus R$ (the complement of $R$)
  - Build a DFA for $\Sigma^* \setminus R$ from a DFA for $R$ by making accepting states non-accepting and vice versa.
Some properties: If $R$ and $S$ are regular languages, then so are

- $RS, R \cup S, R^*$
  - By definition
- $\Sigma^* \setminus R$ (the complement of $R$)
  - Build a DFA for $\Sigma^* \setminus R$ from a DFA for $R$ by making accepting states non-accepting and vice versa.
- $R^R := \{ \sigma^R \mid \sigma \in R \}$, where $\sigma^R$ is $\sigma$ written backwards
**Some properties:** If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
  - By definition
- $\Sigma^* \setminus R$ (the complement of $R$)
  - Build a DFA for $\Sigma^* \setminus R$ from a DFA for $R$ by making accepting states non-accepting and vice versa.
- $R^R := \{ \sigma^R \mid \sigma \in R \}$, where $\sigma^R$ is $\sigma$ written backwards
  - The regular expression for $R$ “written backwards” is a regular expression for $R^R$. 
How General Are Regular Languages? (1)

Some properties: If \( R \) and \( S \) are regular languages, then so are

- \( RS, R \cup S, R^* \)
  - By definition

- \( \Sigma^* \setminus R \) (the complement of \( R \))
  - Build a DFA for \( \Sigma^* \setminus R \) from a DFA for \( R \) by making accepting states non-accepting and vice versa.

- \( R^R := \{ \sigma^R | \sigma \in R \} \), where \( \sigma^R \) is \( \sigma \) written backwards
  - The regular expression for \( R \) “written backwards” is a regular expression for \( R^R \).

- \( R \cap S \)
How General Are Regular Languages? (1)

Some properties: If $R$ and $S$ are regular languages, then so are

- $RS$, $R \cup S$, $R^*$
  - By definition
- $\Sigma^* \setminus R$ (the complement of $R$)
  - Build a DFA for $\Sigma^* \setminus R$ from a DFA for $R$ by making accepting states non-accepting and vice versa.
- $R^R := \{\sigma^R | \sigma \in R\}$, where $\sigma^R$ is $\sigma$ written backwards
  - The regular expression for $R$ “written backwards” is a regular expression for $R^R$.
- $R \cap S$
  - $R \cap S = \Sigma^* \setminus ((\Sigma^* \setminus R) \cup (\Sigma^* \setminus S))$
How General Are Regular Languages? (1)

Some properties: If \( R \) and \( S \) are regular languages, then so are

- \( RS, R \cup S, R^* \)
  - By definition
- \( \Sigma^* \setminus R \) (the complement of \( R \))
  - Build a DFA for \( \Sigma^* \setminus R \) from a DFA for \( R \) by making accepting states non-accepting and vice versa.
- \( R^R := \{ \sigma^R \mid \sigma \in R \} \), where \( \sigma^R \) is \( \sigma \) written backwards
  - The regular expression for \( R \) “written backwards” is a regular expression for \( R^R \).
- \( R \cap S \)
  - \( R \cap S = \Sigma^* \setminus ((\Sigma^* \setminus R) \cup (\Sigma^* \setminus S)) \)
- \( R \setminus S \)
How General Are Regular Languages? (1)

Some properties: If \( R \) and \( S \) are regular languages, then so are

- \( RS, R \cup S, R^* \)
  - By definition
- \( \Sigma^* \setminus R \) (the complement of \( R \))
  - Build a DFA for \( \Sigma^* \setminus R \) from a DFA for \( R \) by making accepting states non-accepting and vice versa.
- \( R^R := \{ \sigma^R \mid \sigma \in R \} \), where \( \sigma^R \) is \( \sigma \) written backwards
  - The regular expression for \( R \) “written backwards” is a regular expression for \( R^R \).
- \( R \cap S \)
  - \( R \cap S = \Sigma^* \setminus ((\Sigma^* \setminus R) \cup (\Sigma^* \setminus S)) \)
- \( R \setminus S \)
  - \( R \setminus S = R \cap (\Sigma^* \setminus S) \)
Not all languages are regular!

The language $\mathcal{L} = \{0^n 1^n \mid n \geq 0\}$ is not regular!
How General Are Regular Languages? (2)

Not all languages are regular!

**Pumping Lemma:** For every regular language $\mathcal{L}$, there exists a constant $n$ such that every $\sigma \in \mathcal{L}$ with $|\sigma| \geq n$ can be divided into three substrings $\sigma = \alpha \beta \gamma$ with the following properties:

- $|\alpha \beta| \leq n$,
- $|\beta| > 0$, and
- $\alpha \beta^k \gamma \in \mathcal{L}$, for all $k \geq 0$.

The language $\mathcal{L} = \{0^n 1^n \mid n \geq 0\}$ is not regular!
How General Are Regular Languages? (2)

Not all languages are regular!

**Pumping Lemma:** For every regular language $L$, there exists a constant $n$ such that every $\sigma \in L$ with $|\sigma| \geq n$ can be divided into three substrings $\sigma = \alpha \beta \gamma$ with the following properties:

- $|\alpha \beta| \leq n$,
- $|\beta| > 0$, and
- $\alpha \beta^k \gamma \in L$, for all $k \geq 0$.

The language $L = \{0^n 1^n | n \geq 0\}$ is not regular!

**Proof:** If it was, there would exist an $n$ as in the pumping lemma. Choose the string $\sigma = 0^n 1^n$. By the pumping lemma, we can divide this string into three parts $\sigma = \alpha \beta \gamma$ with $|\alpha \beta| \leq n$ and $|\beta| > 0$ and such that the string $\alpha \beta \beta \gamma$ also belongs to $L$. However, $|\alpha \beta| \leq n$ implies that $\alpha = 0^k$ and $\beta = 0^m$, where $m > 0$. Thus, $\alpha \beta \beta \gamma = 0^{m+n} 1^n$, a contradiction.
Proof of the Pumping Lemma

\[ D := \text{DFA for } \mathcal{L} \]

\[ n := \text{number of states of } D + 1 \]
Proof of the Pumping Lemma

\[ D := \text{DFA for } \mathcal{L} \]

\[ n := \text{number of states of } D + 1 \]
Proof of the Pumping Lemma

\[ D := \text{DFA for } \mathcal{L} \]
\[ n := \text{number of states of } D + 1 \]
Pumping Lemma: For every regular language $\mathcal{L}$, there exists a constant $n$ such that every $\sigma \in \mathcal{L}$ with $|\sigma| \geq n$ can be divided into three substrings $\sigma = \alpha \beta \gamma$ with the following properties:

- $|\alpha \beta| \leq n$,
- $|\beta| > 0$, and
- $\alpha \beta^k \gamma \in \mathcal{L}$, for all $k \geq 0$. 

Applying the Pumping Lemma: Examples
Applying the Pumping Lemma: Examples

**Pumping Lemma:** For every regular language \( \mathcal{L} \), there exists a constant \( n \) such that every \( \sigma \in \mathcal{L} \) with \( |\sigma| \geq n \) can be divided into three substrings \( \sigma = \alpha\beta\gamma \) with the following properties:

- \( |\alpha\beta| \leq n \),
- \( |\beta| > 0 \), and
- \( \alpha\beta^k\gamma \in \mathcal{L} \), for all \( k \geq 0 \).

\[ \mathcal{L} = \{ {m \choose m}^m \mid m \geq 0 \} \]

Assume \( \mathcal{L} \) is regular. For \( m \geq n \), \( \sigma = {m \choose m}^m \in \mathcal{L} \). Hence, \( \sigma = \alpha\beta\gamma \) with \( \alpha = {a \choose a} \), \( \beta = {b \choose b} \), \( a + b \leq n \), and \( b > 0 \). Then \( \alpha\beta\beta\gamma = {m+b \choose m}^m \in \mathcal{L} \), a contradiction.
Applying the Pumping Lemma: Examples

**Pumping Lemma:** For every regular language $\mathcal{L}$, there exists a constant $n$ such that every $\sigma \in \mathcal{L}$ with $|\sigma| \geq n$ can be divided into three substrings $\sigma = \alpha \beta \gamma$ with the following properties:

- $|\alpha \beta| \leq n$,
- $|\beta| > 0$, and
- $\alpha \beta^k \gamma \in \mathcal{L}$, for all $k \geq 0$.

$\mathcal{L} = \{(m)^m \mid m \geq 0\}$

Assume $\mathcal{L}$ is regular. For $m \geq n$, $\sigma = (m)^m \in \mathcal{L}$. Hence, $\sigma = \alpha \beta \gamma$ with $\alpha = (a^a, \beta = (b^b, a + b \leq n$, and $b > 0$. Then $\alpha \beta \beta \gamma = (m+b)^m \in \mathcal{L}$, a contradiction.

$\mathcal{L} = \{a^p \mid p \text{ is a prime number}\}$

Assume $\mathcal{L}$ is regular. For $p \geq n+2$, $\sigma = a^p \in \mathcal{L}$. Hence $\sigma = \alpha \beta \gamma$, where $\alpha = a^a, \beta = a^b, a + b \leq n$, and $b > 0$. Let $c = p - b \geq 2$, that is, $c = |\alpha \gamma|$. We have $\alpha \beta^c \gamma \in \mathcal{L}$. However, $|\alpha \beta^c \gamma| = (b + 1)c$, which is not prime because $b + 1 \geq 2$ and $c \geq 2$. 