Regular definitions

For notational convenience we give names to certain regular expressions. A regular definition, on the alphabet Σ is sequence of definitions of the form:

- $d_1 \to r_1$
- $d_2 \to r_2$
- $\dots$
- $d_n \to r_n$

where:

- Each d_i is a new symbol, not in Σ, and not the same as any other of the d's
- Each r_i is a regular expression over the alphabet $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$

Using regular definitions

The tokens of a language can be defined as:

- $letter \rightarrow a|b|...|z|A|B|...|Z$
- $letter\_ \rightarrow letter|\_$
  - compact syntax: $[a - zA - B]$
- $digit \rightarrow 0|1|...|9$
  - compact syntax: $[0 - 9]$
- $integers \rightarrow (-|\epsilon)digit \cdot digit^*$
- $identifiers \rightarrow letter\_(letter\_|digit)*$
- $expnot \rightarrow digit(.digit^+E(+|-)digit^+)? \quad \text{(Exponential Notation)}$

Exercise

Write regular definitions for the following languages:

- All strings of lowercase letters that contains the five vowels in order
- All strings of lowercase letters in which the letters are in ascending lexicographic order
- All strings of digits with no repeated digits
- All strings with an even number of a's and and an odd number of b's

How does the lexical analyser work?

Suppose we are given a regular definition $R = \{d_1, \ldots, d_m\}$

1. Let the input be $x_0 \cdots x_n \in \Sigma^*$
   For $0 \leq i \leq n$ check if $x_0 \cdots x_i \in \mathscr{L}(d_j)$ for some $j \in \{1, \ldots, m\}$
2. if success then we know that $x_0 \cdots x_i \in \mathscr{L}(d_j)$ for some j
3. remove $x_0 \cdots x_i$ from input and go to 1

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

LA matching rules

Suppose that at the same time for $i < j$, $i, j \in \{0, \ldots, n\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$ for some k
- $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_k)$ or $x_0 \cdots x_i \cdots x_j \in \mathscr{L}(d_h)$ for some $h \neq k$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \in \{0, \ldots, n\}$ and $k \neq h$, $k, h \in \{1, \ldots, m\}$:

- $x_0 \cdots x_i \in \mathscr{L}(d_k)$
- $x_0 \cdots x_i \in \mathscr{L}(d_h)$

Which is the match to consider?

first one listed rule, i.e., d_k

Errors: to manage errors put as last match in the list a regexp for all lexemes not in the language

Finite Automata

- Regular Expressions = specification of tokens
- Finite Automata = recognition of tokens

Finite Automaton

A Finite Automaton $\mathcal{A}$ is a tuple $\langle \mathcal{S}, \Sigma, \delta, s_0, \mathcal{F} \rangle$ where:

- $\mathcal{S}$ represents the set of states
- Σ represents a set of symbols (alphabet)
- δ represents the transition function ($\delta : \mathcal{S} \times \Sigma \rightarrow \ldots$)
- s_0 represents the start state ($s_0 \in \mathcal{S}$)
- $\mathcal{F}$ represents the set of accepting states ($\mathcal{F} \subseteq \mathcal{S}$)

In two flavours: Deterministic Finite Automata (DFA) and Non-Deterministic Finite Automata (NFA)

Finite Automata

- Regular Expressions = specification of tokens
- Finite Automata = recognition of tokens

Finite Automaton

A Finite Automaton $\mathcal{A}$ is a tuple $\langle \mathcal{S}, \Sigma, \delta, s_0, \mathcal{F} \rangle$ where:

- $\mathcal{S}$ represents the set of states
- Σ represents a set of symbols (alphabet)
- δ represents the transition function ($\delta : \mathcal{S} \times \Sigma \rightarrow \ldots$)
- s_0 represents the start state ($s_0 \in \mathcal{S}$)
- $\mathcal{F}$ represents the set of accepting states ($\mathcal{F} \subseteq \mathcal{S}$)

In two flavours: Deterministic Finite Automata (DFA) and Non-Deterministic Finite Automata (NFA)

Finite Automata

DFA vs. NFA

Depending on the definition of δ we distinguish between:

- Deterministic Finite Automata (DFA) - $\delta : \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$
- Nondeterministic Finite Automata (NFA) $\delta : \mathcal{S} \times \Sigma \rightarrow \mathscr{P}(\mathcal{S})$

The transition relation δ can be represented in a table (transition table)

$\mathscr{P}(\mathcal{S}) = 2^{\mathcal{S}}$ is the powerset of the set $\mathcal{S}$ of states, i.e., the set of all the subsets of $\mathcal{S}$

Overview of the graphical notation circle and edges (arrows)

Finite Automata

DFA vs. NFA

Depending on the definition of δ we distinguish between:

- Deterministic Finite Automata (DFA) - $\delta : \mathcal{S} \times \Sigma \rightarrow \mathcal{S}$
- Nondeterministic Finite Automata (NFA) $\delta : \mathcal{S} \times \Sigma \rightarrow \mathscr{P}(\mathcal{S})$

The transition relation δ can be represented in a table (transition table)

$\mathscr{P}(\mathcal{S}) = 2^{\mathcal{S}}$ is the powerset of the set $\mathcal{S}$ of states, i.e., the set of all the subsets of $\mathcal{S}$

Overview of the graphical notation circle and edges (arrows)

Acceptance of Strings for DFAs

Moves of a DFA

A DFA "consumes" an input character c going from a state s to a state s' if $\delta(s, c) = s'$, written $s \xrightarrow{c} s'$

A DFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of states $s_{i+1}, \ldots, s_{i+n-1}, s_{i+n} = s_j$ s.t.

$\forall k \in \{1, \ldots, n\}.\delta(s_{i+k-1}, a_k) = s_{i+k}$, written $s_i \xrightarrow{\mathbf{a}} s_j$

Acceptance of Strings

A DFA accepts a string $\mathbf{a}$ if and only if it consumes $\mathbf{a}$ from the initial state s_0 to a final state s_i, i.e., $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Accepted Language

The language accepted by a DFA is the set of all the strings $\mathbf{a}$ such that $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Acceptance of Strings for DFAs

Moves of a DFA

A DFA "consumes" an input character c going from a state s to a state s' if $\delta(s, c) = s'$, written $s \xrightarrow{c} s'$

A DFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of states $s_{i+1}, \ldots, s_{i+n-1}, s_{i+n} = s_j$ s.t. $\forall k \in \{1, \ldots, n\}.\delta(s_{i+k-1}, a_k) = s_{i+k}$, written $s_i \xrightarrow{\mathbf{a}} s_j$

Acceptance of Strings

A DFA accepts a string $\mathbf{a}$ if and only if it consumes $\mathbf{a}$ from the initial state s_0 to a final state s_i, i.e., $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Accepted Language

The language accepted by a DFA is the set of all the strings $\mathbf{a}$ such that $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Acceptance of Strings for DFAs

Moves of a DFA

A DFA "consumes" an input character c going from a state s to a state s' if $\delta(s, c) = s'$, written $s \xrightarrow{c} s'$
A DFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of states $s_{i+1}, \ldots, s_{i+n-1}, s_{i+n} = s_j$ s.t.
$\forall k \in \{1, \ldots, n\}.\delta(s_{i+k-1}, a_k) = s_{i+k}$, written $s_i \xrightarrow{\mathbf{a}} s_j$

Acceptance of Strings

A DFA accepts a string $\mathbf{a}$ if and only if it consumes $\mathbf{a}$ from the initial state s_0 to a final state s_i, i.e., $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Accepted Language

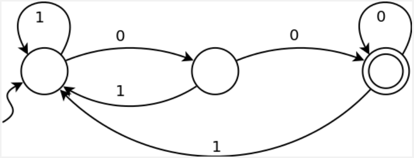
The language accepted by a DFA is the set of all the strings $\mathbf{a}$ such that $s_0 \xrightarrow{\mathbf{a}} s_i$ and $s_i \in \mathcal{F}$

Exercise

Define the following automata:

- DFA for a single 1
- DFA for accepting any number of 1's followed by a single 0
- DFA for any sequence of a or b (possibly empty) followed by 'abb'

Exercise



Which regular expression corresponds to the automaton?

1. (0|1)*
2. (1*|0)(1|0)
3. 1*|(01)*|(001)*|(000*1)*
4. (0|1)*00

ϵ-moves

DFA, NFA and ϵ-moves

- DFA
  - at most one transition for one input in a given state
  - no ϵ-moves
- NFA
  - can have multiple transitions for one input in a given state
  - can have ϵ-moves, i.e., $\delta : \mathcal{S} \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathscr{P}(\mathcal{S})$
  - smaller (exponentially)

Acceptance of Strings for NFAs

Moves of an NFA

An NFA "consumes" an input character c going from a state s to a state s' if $s' \in \delta(s, c)$, written $s \xrightarrow{c} s'$

An NFA can move from a state s to a state s' without consuming any input character, written $s \xrightarrow{\epsilon} s'$

An NFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of moves $s_i \xrightarrow{x_0} s_{i+1} \xrightarrow{x_1} \ldots s_{i+m-1} \xrightarrow{x_{m-1}} s_{i+m} = s_j$ s.t. $\forall k \in \{0, \ldots, m-1\}. s_{i+k} \in \delta(s_{i+k}, x_k)$ and $x_0 x_1 \cdots x_{m-1} = \mathbf{a}$, written $s_i \stackrel{\mathbf{a}}{\Longrightarrow} s_j$

Acceptance of Strings

An NFA accepts a string $\mathbf{a}$ if and only if there exists at least one sequence of moves from the initial state s_0 to a state s_i such that s_i is a final state, i.e., $\exists s_i \in \mathcal{F} : s_0 \stackrel{\mathbf{a}}{\Longrightarrow} s_i$

Accepted Language

The language accepted by an NFA is the set of all the strings $\mathbf{a}$ such that $\exists s_i \in \mathcal{F} : s_0 \stackrel{\mathbf{a}}{\Longrightarrow} s_i$

Acceptance of Strings for NFAs

Moves of an NFA

An NFA "consumes" an input character c going from a state s to a state s' if $s' \in \delta(s, c)$, written $s \xrightarrow{c} s'$

An NFA can move from a state s to a state s' without consuming any input character, written $s \xrightarrow{\epsilon} s'$

An NFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of moves $s_i \xrightarrow{x_0} s_{i+1} \xrightarrow{x_1} \ldots s_{i+m-1} \xrightarrow{x_{m-1}} s_{i+m} = s_j$ s.t. $\forall k \in \{0, \ldots, m-1\}. s_{i+k} \in \delta(s_{i+k}, x_k)$ and $x_0 x_1 \cdots x_{m-1} = \mathbf{a}$, written $s_i \xRightarrow{\mathbf{a}} s_j$

Acceptance of Strings

An NFA accepts a string $\mathbf{a}$ if and only if there exists at least one sequence of moves from the initial state s_0 to a state s_i such that s_i is a final state, i.e., $\exists s_i \in \mathcal{F}\colon\ s_0 \xRightarrow{\mathbf{a}} s_i$

Accepted Language

The language accepted by an NFA is the set of all the strings $\mathbf{a}$ such that $\exists s_i \in \mathcal{F}\colon\ s_0 \xRightarrow{\mathbf{a}} s_i$

Acceptance of Strings for NFAs

Moves of an NFA

An NFA "consumes" an input character c going from a state s to a state s' if $s' \in \delta(s, c)$, written $s \xrightarrow{c} s'$

An NFA can move from a state s to a state s' without consuming any input character, written $s \xrightarrow{\epsilon} s'$

An NFA "consumes" a string $\mathbf{a} = a_1 a_2 \cdots a_n$ going from a state s_i to a state s_j if there is a sequence of moves $s_i \xrightarrow{x_0} s_{i+1} \xrightarrow{x_1} \ldots s_{i+m-1} \xrightarrow{x_{m-1}} s_{i+m} = s_j$ s.t. $\forall k \in \{0, \ldots, m-1\}. s_{i+k} \in \delta(s_{i+k}, x_k)$ and $x_0 x_1 \cdots x_{m-1} = \mathbf{a}$, written $s_i \xRightarrow{\mathbf{a}} s_j$

Acceptance of Strings

An NFA accepts a string $\mathbf{a}$ if and only if there exists at least one sequence of moves from the initial state s_0 to a state s_i such that s_i is a final state, i.e., $\exists s_i \in \mathcal{F}\colon s_0 \xRightarrow{\mathbf{a}} s_i$

Accepted Language

The language accepted by an NFA is the set of all the strings $\mathbf{a}$ such that $\exists s_i \in \mathcal{F}\colon s_0 \xRightarrow{\mathbf{a}} s_i$

From regexp to NFA

Equivalent NFA for a regexp

The Thompson's algorithm permits to automatically derive an NFA from the specification of a regexp. It defines basic NFAs for basic regexps and rules to compose them:

1. for ϵ
2. for 'c'
3. for ab
4. for $a + b$
5. for a^*

Now consider the regexp for (1|0)*1