2. Lexical Analysis II

Andrea Polini

Formal Languages and Compilers
Master in Computer Science
University of Camerino

October 28^{st}, 2014

Meaning function $\mathscr{L}$

- The meaning function L maps syntax to semantics

$$\mathscr{L}(e) = \mathscr{M} \text{ where } e \text{ is a RE and } \mathscr{M} \text{ is a set of strings}$$

Therefore:

- $\mathscr{L}(\epsilon) = \{\text{`` ''}\}$
- $\mathscr{L}('c') = \{\text{``}c\text{''}\}$
- $\mathscr{L}(A + B) = \mathscr{L}(A) \cup \mathscr{L}(B)$
- $\mathscr{L}(AB) = \{ab | a \in \mathscr{L}(A) \wedge b \in \mathscr{L}(B)\}$
- $\mathscr{L}(A^*) = \{\cup_{i \geq 0} \mathscr{L}(A^i)\}$

Meaning function $\mathscr{L}$

- Why use a meaning function?
  - Makes clear what is syntax, what is semantics
  - Allows us to consider notation as a separate issue
  - Because expressions and meanings are not 1 to 1
    - consider the case of arabic number and roman numbers

Many different RE can be used to identify the same regular language

It should never happen that the same syntactical structure permits to define more than one meaning

Meaning function $\mathscr{L}$

- Why use a meaning function?
  - Makes clear what is syntax, what is semantics
  - Allows us to consider notation as a separate issue
  - Because expressions and meanings are not 1 to 1
    - consider the case of arabic number and roman numbers

Many different RE can be used to identify the same regular language

It should never happen that the same syntactical structure permits to define more than one meaning

Meaning function $\mathscr{L}$

- Why use a meaning function?
  - Makes clear what is syntax, what is semantics
  - Allows us to consider notation as a separate issue
  - Because expressions and meanings are not 1 to 1
    - consider the case of arabic number and roman numbers

Many different RE can be used to identify the same regular language

It should never happen that the same syntactical structure permits to define more than one meaning

Regular definitions

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

- $d_1 \rightarrow r_1$
- $d_2 \rightarrow r_2$
- $\ldots$
- $d_n \rightarrow r_n$

So token of a language can be defined as:

- $letter \rightarrow a|b|...|z|A|B|...|Z$
  - compact syntax: $[a - zA - B]$
- $digit \rightarrow 0|1|...|9$
  - compact syntax: $[0 - 9]$
- $Identifier \rightarrow letter(letter|digit)*$
- $Integer \rightarrow ...$

Lexical Specification

- At least one: $A^{+} \equiv AA^{*}$
- Union: $A|B \equiv A + B$
- Option: $A? \equiv A + \epsilon$
- Range: $'a' +' b' + ... +' z' \equiv [a - z]$
- Excluded range: complement of $[a - z] \equiv [^{\wedge}a - z]$

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier, . . .)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 \ldots X_n$
   For $1 \leq i \leq n$ check if $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 \ldots X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,...)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 \ldots X_n$
   For $1 \leq i \leq n$ check if $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 \ldots X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,. . .)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 \ldots X_n$
   For $1 \leq i \leq n$ check if $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 \ldots X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,. . .)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 \ldots X_n$
   For $1 \leq i \leq n$ check if $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 \ldots X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 \ldots X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,...)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 ... X_n$
   For $1 \leq i \leq n$ check if $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 ... X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,...)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 ... X_n$
   For $1 \leq i \leq n$ check if $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 ... X_i$ from input and go to (3)

Lexical Specification

We want to derive a regular expression for all tokens of a language:

$s \in \mathscr{L}(R)$ – where R is the RE for all different kind ot tokens

How can we define it?

1. write a rexp for the lexemes of each token class (number, keyword, identifier,. . .)
2. Constructs R matching all lexemes for all tokens
3. Let input be $X_1 ... X_n$
   For $1 \leq i \leq n$ check if $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
4. if success then we know that $X_1 ... X_i \in \mathscr{L}(R_j)$ for some j
5. remove $X_1 ... X_i$ from input and go to (3)

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 ... X_i \in \mathscr{L}(R)$
- $X_1 ... X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | ... | R_n$:

- $X_1 ... X_k \in \mathscr{L}(R_i)$
- $X_1 ... X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 ... X_i \in \mathscr{L}(R)$
- $X_1 ... X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | ... | R_n$:

- $X_1 ... X_k \in \mathscr{L}(R_i)$
- $X_1 ... X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 \ldots X_i \in \mathscr{L}(R)$
- $X_1 \ldots X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | \ldots | R_n$:

- $X_1 \ldots X_k \in \mathscr{L}(R_i)$
- $X_1 \ldots X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 ... X_i \in \mathscr{L}(R)$
- $X_1 ... X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | ... | R_n$:

- $X_1 ... X_k \in \mathscr{L}(R_i)$
- $X_1 ... X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1...X_i \in \mathscr{L}(R)$
- $X_1...X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1|R_2|...|R_n$:

- $X_1...X_k \in \mathscr{L}(R_i)$
- $X_1...X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 ... X_i \in \mathscr{L}(R)$
- $X_1 ... X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | ... | R_n$:

- $X_1 ... X_k \in \mathscr{L}(R_i)$
- $X_1 ... X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

LA matching rules

Suppose that at the same time for $i \neq j$:

- $X_1 ... X_i \in \mathscr{L}(R)$
- $X_1 ... X_j \in \mathscr{L}(R)$

Which is the match to consider?

longest match rule

Suppose that at the same time for $i \neq j \in [1..n]$ and $R = R_1 | R_2 | ... | R_n$:

- $X_1 ... X_k \in \mathscr{L}(R_i)$
- $X_1 ... X_k \in \mathscr{L}(R_j)$

Which is the match to consider?

first one listed rule

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

Finite Automata

- Regular Expressions = specification
- Finite Automata = implementation

A Finite Automata is a tuple: $< \mathscr{S}, \Sigma, \delta, s_0, \mathscr{F} >$

DFA vs. NFA

Depending on the definition of δ we distinguish between:

- Deterministic Finite Automata (DFA)
- Nondeterministic Finite Automata (NFA)

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

Overview of the graphical notation circle and edges (arrows)

Finite Automata

- Regular Expressions = specification
- Finite Automata = implementation

A Finite Automata is a tuple: $< \mathscr{S}, \Sigma, \delta, s_0, \mathscr{F} >$

DFA vs. NFA

Depending on the definition of δ we distinguish between:

- Deterministic Finite Automata (DFA)
- Nondeterministic Finite Automata (NFA)

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

Overview of the graphical notation circle and edges (arrows)

Finite Automata

- Regular Expressions = specification
- Finite Automata = implementation

A Finite Automata is a tuple: $< \mathscr{S}, \Sigma, \delta, s_0, \mathscr{F} >$

DFA vs. NFA

Depending on the definition of δ we distinguish between:

- Deterministic Finite Automata (DFA)
- Nondeterministic Finite Automata (NFA)

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

Overview of the graphical notation circle and edges (arrows)

Finite Automata

- Regular Expressions = specification
- Finite Automata = implementation

A Finite Automata is a tuple: $< \mathscr{S}, \Sigma, \delta, s_0, \mathscr{F} >$

DFA vs. NFA

Depending on the definition of δ we distinguish between:

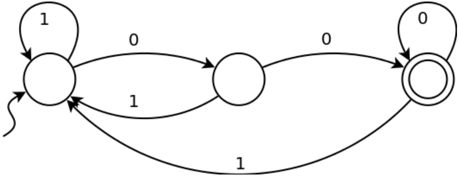
- Deterministic Finite Automata (DFA)
- Nondeterministic Finite Automata (NFA)

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

Overview of the graphical notation circle and edges (arrows)

examples

- 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'



Which rexp correctly defines the automata:

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
  - one transition per input per state
  - no ϵ-moves
  - faster
- NFA
  - can have multiple transitions for one input in a given state
  - can have ϵ-moves
  - smaller (exponentially)

ϵ-moves

DFA, NFA and ϵ-moves

- DFA
  - one transition per input per state
  - no ϵ-moves
  - faster
- NFA
  - can have multiple transitions for one input in a given state
  - can have ϵ-moves
  - smaller (exponentially)

From rexp to NFA

Equivalent NFA for a rexp

1. for ϵ
2. for 'a'
3. for AB
4. for A|B
5. for A*

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