2. Lexical Analysis

Andrea Polini

Formal Languages and Compilers
Master in Computer Science
University of Camerino

October 21^{st} . . . , 2014

Lexical Analysis

```
if (i==j)
  z=0;
else
  z=1;
```

```
\tif (i==j)\n\t\tz=0;\n\telse\n\t\tz=1;
```

Lexical Analysis

```
if (i==j)
  z=0;
else
  z=1;
```

```
\tif (i==j)\n\t\tz=0;\n\telse\n\t\tz=1;
```

Lexical Analysis

- Token Class (or Class)
  - In English: *Noun, Verb, Adjective, Adverb, Article, . . .*
  - In a programming language: *Identifier, Keywords, "(", ")", Numbers, . . .*

Lexical Analysis

- Token classes corresponds to sets of strings
- Identifier
  - strings of letter or digits starting with a letter
- Integer
  - a non-empty string of digits
- Keyword
  - "else", "if", "while", . . .
- Whitespace
  - a non-empty sequence of blanks, newlines, and tabs

Lexical analysis

Therefore the role of the lexical analyzer (Lexer) is:

- Classify program substring according to role (token class)
- communicate tokens to parser

Lexical Analysis

Let's analyze these lines of code:

```
\tif (i==j)\n\t\tz=0;\n\telse\n\t\tz=1;
```

```
x=0;\n\twhile (x<10) {\n\tx++;\n}
```

Token Classes: Identifier, Integer, Keyword, Whitespace

Lexical Analysis

Therefore an implementation of a lexical analyzer must do two things:

- Recognize substrings corresponding to tokens
  - the lexemes
- Identify the token class for each lexemes

Lexical Analysis - Tricky problems

- FORTRAN rule: whitespace is insignificant
  - i.e. VA R1 is the same as VAR1

```
DO 5 I = 1,25

DO 5 I = 1.25
```

In FORTRAN the “5” refers to a label you will find in the following of the program code

Lexical Analysis - Tricky problems

1. The goal is to partition the string. This is implemented by reading left-to-right, recognizing one token at a time
2. "Lookahead" may be required to decide where one token ends and the next token begins

```
if (i==j)
  z=0;
else
  z=1;
```

Lexical Analysis - Tricky problems

- PL/1 keywords are not reserved

```
IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN
```

```
DECLARE(ARG1,...,ARGN)
```

Is DECLARE a keyword or an array reference?

Need for an unbounded lookahead

Lexical Analysis - Tricky problems

- PL/1 keywords are not reserved

```
IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN
```

```
DECLARE(ARG1,...,ARGN)
```

Is `DECLARE` a keyword or an array reference?

Need for an unbounded lookahead

Lexical Analysis - Tricky problems

- PL/1 keywords are not reserved

```
IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN
```

```
DECLARE(ARG1,...,ARGN)
```

Is `DECLARE` a keyword or an array reference?

Need for an unbounded lookahead

Lexical Analysis - Tricky problems

- C++ template syntax:

```
Foo<Bar>
```

- C++ stream syntax:

```
cin >> var;
```

```
Foo<Bar<Barr>>
```

Lexical Analysis - Tricky problems

- C++ template syntax:

```
Foo<Bar>
```

- C++ stream syntax:

```
cin >> var;
```

```
Foo<Bar<Barr>>
```

Regular Languages

- We need a way to define which is the set of strings in a token class
  - Use of regular languages is enough

Regular expressions are a suitable way to syntactically identify strings belonging to a regular language

Regular expressions

- Single character: 'c' = {"c"}
- Epsilon: ϵ = {" "}
- Union: A+B = $\{a|a \in A\} \cup \{b|b \in B\}$
- Concatenation: AB = $\{ab|a \in A \wedge b \in B\}$
- Iteration: A* = $\cup_{i \geq 0} A^i$

- **Def.** The regular expressions over Σ are the smallest set including ϵ, all the character 'c' in Σ and that is closed with respect to union, concatenation and iteration.
- Algebraic laws for RE:
  - $+$ is commutative and associative
  - concatenation is associative
  - concatenation distributes over $+$
  - ϵ is the identity for concatenation
  - ϵ is guaranteed in a closure
  - the Kleene star is idempotent

Regular expressions

- Single character: 'c' = $\{$"c"$\}$
- Epsilon: ϵ = $\{$" "$\}$
- Union: A+B = $\{a|a \in A\} \cup \{b|b \in B\}$
- Concatenation: AB = $\{ab|a \in A \wedge b \in B\}$
- Iteration: A* = $\cup_{i \geq 0} A^i$

- **Def.** The regular expressions over Σ are the smallest set including ϵ, all the character 'c' in Σ and that is closed with respect to union, concatenation and iteration.
- Algebraic laws for RE:
  - $+$ is commutative and associative
  - concatenation is associative
  - concatenation distributes over $+$
  - ϵ is the identity for concatenation
  - ϵ is guaranteed in a closure
  - the Kleene star is idempotent

Regular expressions

- Single character: 'c' = {"c"}
- Epsilon: ϵ = {" "}
- Union: A+B = $\{a|a \in A\} \cup \{b|b \in B\}$
- Concatenation: AB = $\{ab|a \in A \wedge b \in B\}$
- Iteration: A* = $\cup_{i \geq 0} A^i$

- **Def.** The regular expressions over Σ are the smallest set including ϵ, all the character 'c' in Σ and that is closed with respect to union, concatenation and iteration.
- Algebraic laws for RE:
  - + is commutative and associative
  - concatenation is associative
  - concatenation distributes over +
  - ϵ is the identity for concatenation
  - ϵ is guaranteed in a closure
  - the Kleene star is idempotent

Regular expressions - example

- Consider $\Sigma = \{0, 1\}$. What are the sets defined by the following REs?
  - 1^*
  - $(1 + 0)1$
  - $0^* + 1^*$
  - $(0 + 1)^*$
- Given the regular language identified by $(0 + 1)^*1(0 + 1)^*$ which are the regular expressions identifying the same language among the following one:
  - $(01 + 11)^*(0 + 1)^*$
  - $(0 + 1)^*(10 + 11 + 1)(0 + 1)^*$
  - $(1 + 0)^*1(1 + 0)^*$
  - $(0 + 1)^*(0 + 1)(0 + 1)^*$

Regular expressions - example

- Consider $\Sigma = \{0, 1\}$. What are the sets defined by the following REs?
  - 1^*
  - $(1 + 0)1$
  - $0^* + 1^*$
  - $(0 + 1)^*$
- Given the regular language identified by $(0 + 1)^*1(0 + 1)^*$ which are the regular expressions identifying the same language among the following one:
  - $(01 + 11)^*(0 + 1)^*$
  - $(0 + 1)^*(10 + 11 + 1)(0 + 1)^*$
  - $(1 + 0)^*1(1 + 0)^*$
  - $(0 + 1)^*(0 + 1)(0 + 1)^*$

Regular expressions - example

- Choose the regular languages that are correct specifications of the following English-language description:
  - Twelve-hour times of the form “04:13PM”. Minutes should always be a two digit number, but hours may be a single digit
- $(0+1)?[0-9] : [0-5][0-9](AM+PM)$
- $((0+\epsilon)[0-9]+1[0-2]) : [0-5][0-9](AM+PM)$
- $(0^*[0-9]+1[0-2]) : [0-5][0-9](AM+PM)$
- $(0?[0-9]+1(0+1+2) : [0-5][0-9](a+P)M$

Regular expressions (syntax)
specify regular languages (semantics)

Regular expressions - example

- Choose the regular languages that are correct specifications of the following English-language description:
  - Twelve-hour times of the form “04:13PM”. Minutes should always be a two digit number, but hours may be a single digit
- $(0 + 1)?[0 - 9] : [0 - 5][0 - 9](AM + PM)$
- $((0 + \epsilon)[0 - 9] + 1[0 - 2]) : [0 - 5][0 - 9](AM + PM)$
- $(0^*[0 - 9] + 1[0 - 2]) : [0 - 5][0 - 9](AM + PM)$
- $(0?[0 - 9] + 1(0 + 1 + 2) : [0 - 5][0 - 9](a + P)M$

Regular expressions (syntax)
specify regular languages (semantics)

RE and Languages

- **Def.** Let Σ be a set of characters (*alphabet*). A language over Σ is a set of strings of characters drawn from Σ

Alphabet = English character $\Longrightarrow$ Language = English sentences
Alphabet = ASCII $\Longrightarrow$ Language = C programs

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 language

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 language

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 language