Compilers Design

M. T. Bennani
Assistant Professor, FST - El Manar University, URAPOP-FST

-Academic Year 2020-2021-

Outline

- Introduction
  - Identifies tokens in input string
- Issues in lexical analysis
  - Lookahead
  - Ambiguities
- Specifying lexers
  - Regular expressions
  - Examples of regular expressions
- Exercises

Identifies tokens in input string (1/5): Goal

- What do we want to do? Example:

```
//Example
while (i < 5)
  cpt++;
```

- The input is just a string of characters: `//Example\nwhile ( i < 5)\n\tcpt++;`
- Goal: Partition input string into substrings
  - The substrings are tokens
- A token is a syntactic category
  - In English: noun, verb, adjective, etc.
  - In a programming language: Identifier, Keyword, Whitespace, etc.

Identifies tokens in input string (2/5): Tokens

- Tokens correspond to sets of strings
- Identifiers: Strings of letters or digits stating with a letter
- Integer: a non-empty string of digits
- Keyword: "`if`" or "`else`" or "`while`", etc.
- Whitespace: a non-empty sequence of blanks, newlines, and tabs.

Identifies tokens in input string (3/5): Tokens target

- Classify program substrings according to role
- Output of lexical analysis is a stream of tokens
- This set of tokens is the input of the parser
- Parser relies on token distinctions
  - An identifier is treated differently than a keyword
- There are two steps to design a lexical analyzer
  - Define a finite set of tokens
  - Describe which strings belong to each token

Identifies tokens in input string (4/5): Step 1

- Define a finite set of tokens
  - Tokens describe all items of interest
  - Choice of tokens depends on language, design of parser
- Example: //Example\nwhile (i < 5)\n\tcpt++;
  - Useful tokens for this expression
    - Integer, Keyword, Relation, Identifier, Whitespace, Comment, (,), ++, ;
  - Note1: "(", ")", "++", ";" are token, not characters, here
  - Note2: We may define the Token "`Operator`" instead of "++"

Identifies tokens in input string (5/5): Step 2

//Example\nwhile (i < 5)\n\tcpt++;

- Describe which strings belong to each token
- Recall
  - Identifiers: Strings of letters or digits stating with a letter
    "cpt", "i".
  - Integer: a non-empty string of digits
    "5".
  - Keyword: "if" or "else" or "while", etc.
    "while".
  - Whitespace: a non-empty sequence of blanks, newlines, and tabs.
    "\n", "\t", " ".
  - Define the token "Commnent".
    Hint: Dont forget the multi-lines comment!

Lexical Analyzer: Implementation

- An implementation must do two things:
  1. Recognize substrings corresponding to tokens
  2. Return the value or lexeme of the token
     - The lexeme is the substring
- Example: //Example\nwhile (i < 5)\n\tcpt++;
  - The lexer usually discards "uninteresting" tokens that don't contribute to parsing
  - Whitespaces, Comments

Three important points:

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.
3. ”Conflict” two different tokens may be generated

Lookahead

Even our simple example has lookahead issues:
”w” VS. ”while” and ”$+$” VS. ”$++$”

Conflict

- Template: `GetMax<int>`
- Stream: `cin >> var`
- Nested templates: `template <othertype<sometype>>`

Review

The goal of lexical analysis is to:
- Partition the input string into lexemes
- Identify the token of each lexeme

Left-to-right scan ⇒ lookahead sometimes required

Next

- How to describe the lexemes of each token
- How to resolve ambiguities
  - is ”`while`” five variables?
  - is ”++” two addition operations ”+ +”?

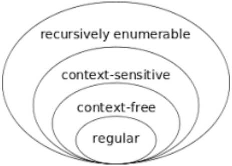
Definitions

Language

Let S be a set of characters. A language over S is a set of strings of characters drawn from S.

Regular language

A regular language (also called a rational language) is a formal language that can be expressed using a regular expressions.



Examples

- Alphabet = English characters
- Language = English sentences

Note1

Not every string of English characters is an English sentence

- Alphabet = ASCII
- Language = C programs

Note2

ASCII character set is different from English character set

Atomic Regular Expressions

- Single Character: 'c'={"c"}
- Epsilon: $'\varepsilon' = \{""\}$

Compound Regular Expressions

- Union: $\mathrm{A} + \mathrm{B} = \{\mathrm{s} \mid s \in A \text{ or } s \in B\}$
- Concatenation: $\mathrm{AB} = \{\mathrm{ab} \mid a \in A \text{ and } b \in B\}$
- Iteration: $A^* = \bigcup_{i \geq 0} A^i$ where $A^i = \mathrm{AA}$ (i times) A

Definiton

The regular expressions over S are the smallest set of expressions including:

- ε
  - $\text{L}(\varepsilon) = \{\text{""}\}$
- 'c' where c $\in \Sigma$
  - $\text{L}(\text{'c'}) = \{\text{"c"}\}$
- A+B where A,B are rexp over Σ
  - $\text{L}(\text{A+B}) = \text{L(A)} \cup \text{L(B)}$
- AB where A,B are rexp over Σ
  - $\text{L(AB)} = \{\text{ab} \mid a \in L(A) \text{ and } b \in L(B)\}$
- A^* where A is a rexp over Σ
  - $\text{L}(A^*) = \bigcup_{i \geq 0} \text{L}(A^i)$

Question

What's L(R) where R=(a+b)(a+b)?

Examples

- Keyword = 'else' + 'if' + 'while' + ...
- Integer:
  - digit = '0'+'1'+'2'+'3'+'4'+'5'+'6'+'7'+'8'+'9'
  - Integer = $digit.digit^*$
  - Note: $A^+ = A.A^*$
- Identifier: strings of letters or digits, starting with a letter
  - letter = 'A'+'B'+...'Z'+'a'+'b'+...+'z'
  - Identifier = letter.$(letter + digit)^*$
- Whitespace: non-empty sequence of blanks, newlines, and tabs
  - Whitespace = $(' \ '+'\backslash\ t'+'\backslash\ n')^+$

Exercise 1

Define the regular expression of the phone numbers in tunisia like: +(216)21-345-676.

Exercise 2

Define the regular expression of the e-mail addresses like: anyone@fst.utm.tn

Exercise 3

Define the regular expression of the comment sections like:

- `// line of comment` or
- `/* region of comments */`

Summary

- Regular expressions describe many useful languages
- Regular languages are a language specification
    - We still need an implementation
- Next time: Given a string s and a rexp R, is

$$S \in L(R)?$$