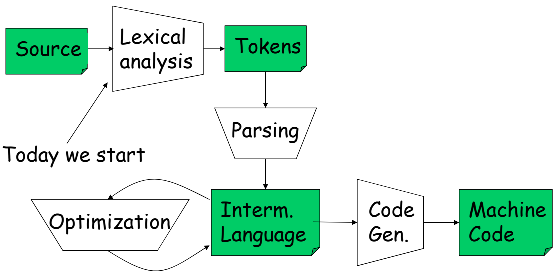
Lexical Analysis

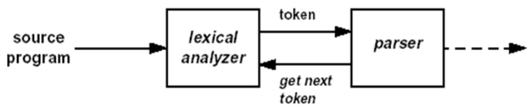
Outline

- Informal sketch of lexical analysis
  - Identifies tokens in input string
- Issues in lexical analysis
  - Lookahead
  - Ambiguities
- Specifying lexers
  - Regular expressions
  - FA: NFA, DFA

Recall: The Structure of a Compiler



Relationship between Lexical Analyzer and Parser



Lexical Analysis

- What do we want to do? Example:

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

- The input is just a sequence of characters:

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

- Goal: Partition input string into substrings(tokens)
  - And classify them according to their role

What's a Token?

- Output of lexical analysis is a stream of tokens
- A token is a syntactic category
  - In English:
    noun, verb, adjective, …
  - In a programming language:
    Identifier, Keyword, Integer, Relation, Whitespace, …
- Parser relies on the token distinctions:
  - E.g., identifiers are treated differently than keywords

Tokens

Tokens correspond to <u>sets of strings</u>:

- Identifier: *strings of letters or digits, starting with a letter*
- Keyword: *"else" or "if" or "begin" or …*
- Integer: *a non-empty string of digits*
- Relation: *<, <=, =, <>, >, >=*
- LeftPar: (
- Whitespace: *a non-empty sequence of blanks, newlines, and tabs*

  … …

7

Lexical Analyzer: Implementation

- An implementation must do two things:

  1. Recognize substrings corresponding to tokens
  2. Return the value or <u>lexeme</u> of the token
     - The lexeme is the substring (instances of the token)

8

Example

- Recall:
  \tif (i == j)\n\t\tz = 0;\n\telse\n\t\tz = 1;
- Token-lexeme pairs returned by the lexer:
  - (Whitespace, "\t")
  - (Keyword, "if")
  - (LeftPar, "(")
  - (Identifier, "i")
  - (Relation, "==")
  - (Identifier, "j")
  - …

9

Lexical Analyzer: Implementation

- The lexer usually discards "uninteresting" tokens that don't contribute to parsing.
- Examples: Whitespace, Comments
- Question: What happens if we remove all whitespace and all comments prior to lexing?

C Program:

```
main()
{
    int i=1, j=2, ij=0;
    i j=i+j;
    printf("%d\n", ij);
}
```

Fortran Program:

```
program main
integer i,j,ij
i=1
j=2
ij=0
i   j=i+j
print *, ij
end
```

10

Ambiguities and Lookahead

- Two important points:
  1. The goal of lexer 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 (especially in Fortran)

```
DO 5  I = 1
DO 5  I = 1.25
DO 5  I = 1, 25

if (then .gt. else) then
      then = else
else
      else = then
endif
```

11

Regular Languages

- In general, there is a set of strings in the input for which the same ***token*** is produced as output. This set of strings is described by a rule called a ***pattern*** associated with the token. The pattern is said to match each string in the set.
- There are several formalisms for specifying tokens
- Regular languages are the most popular
  - Simple and useful theory
  - Easy to understand
  - Efficient implementations

12

Language

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

13

Examples of Languages

- Alphabet = English characters
- Language = English sentences
- Not every string on English characters is an English sentence

- Alphabet = ASCII
- Language = C programs
- Not every string on ASCII characters is a C programs
- Note: ASCII character set is different from English character set

14

Notation

- Languages are sets of strings.
- Need some notation for specifying basic substrings (tokens)
- For lexical analysis we care about *regular languages*, which can be described using *regular expressions*.

15

Regular Expressions and Regular Languages

- Each regular expression is a notation for a regular language (a set of words)
- If A is a regular expression then we write L(A) to refer to the language denoted by A

16

Regular Expressions

- Atomic Regular Expressions
  - Single character: 'c'
    L('**c**') = { "c" }   (for any 'c' ∈ Σ)
  - Epsilon
    L(ε) = { "" }
- Compound Regular Expressions
  - Concatenation: AB (where A and B are reg. exp.)
    L(AB) = { ab | a ∈ L(A) and b ∈ L(B) }
    Example: L('i' 'f') = { "if" }
    (we will abbreviate 'i' 'f' as 'if')

17

Compound Regular Expressions

- Union: A|B
  L(A | B) = { s | s ∈ L(A) or s ∈ L(B) }
  - Examples:
    'if' | 'then'| 'else' = { "if", "then", "else"}
    '0' | '1' | … | '9' = { "0", "1", …, "9" }
    (note the … are just an abbreviation)
  - Another example:
    ('0' | '1') ('0' | '1') = { "00", "01", "10", "11" }

So far we do not have a notation for infinite languages

- Iteration: A^*
  $L(A^*)$ = { "" } I L(A) | L(AA) | L(AAA) | …
  - Examples:
    '0'* = { "", "0", "00", "000", …}
    '1' '0'* = { strings starting with 1 and followed by 0's }

18

Example: Keyword

– Keyword: *"else" or "if" or "begin" or …*

'else' | 'if' | 'begin' | …

(Recall: 'else' abbreviates 'e' 'l' 's' 'e')

Example: Integers

Integer: *a non-empty string of digits*

digit = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
number = digit digit*

Abbreviation: $A^+ = A\ A^*$
number = digit+

Example: Identifier

Identifier: *strings of letters or digits, starting with a letter*

letter = 'A' | … | 'Z' | 'a' | … | 'z'
identifier = letter (letter | digit) *

Is (letter* | digit*) equal to (letter | digit) *?

Example: Whitespace

Whitespace: *a non-empty sequence of blanks, newlines, and tabs*

(' ' | '\t' | '\n')+

Example: Phone Numbers

- Regular expressions are all around you!
- Consider (021) 5135 - 5355

Σ = { 0, 1, 2, 3, …, 9, (,), - }
area = digit³
exchange = digit⁴
phone = digit⁴
number = '(' area ')' exchange '-' phone

Example: Email Addresses

- Consider yuzhang@ustc.edu.cn

Σ = { 'A', 'B', 'C', 'D', …, 'Z', ., @}
name = letter+
address = name '@' name ('.' name)*

Summary

- Regular expressions describe many useful languages
- Next: *Given a string s and a rexp R*, is

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

- But a yes/no answer is not enough !
- Instead: partition the input into lexemes
- We will adapt regular expressions to this goal

25