Lexical Analysis

Chapter 2

Lexical Analysis

- Eliminate white space and comments
- Group characters into tokens
- Speed is important

Terminology

- Lexeme:

  String of input characters matched for a token

- Token:

  Data structure containing token type and value

Types of Tokens

- Values
  1, 3.14, true, 'c', "abc", …
- Identifiers
  x, yz, x42,
- Keywords
  if, while, ...
- Symbols
  +, <, <=, ;, …

Token Examples

- Input:
  x = y * 5;
- Output:
  (ID, x), (ASSIGN), (ID, y), (MUL), (INTCONST, 5), (SEM)

Construction of Lexical Analyzer

- Describe lexemes as regular expressions (REs)
- Translate REs into non-deterministic finite automaton (NFA)
- Translate NFA into deterministic finite automaton (DFA)
- Implement table-driven DFA
- Use JLex for translating Res into DFA

Regular Expressions

• Symbols	a
• Alternation	a | b
• Concatenation	a b
• Repetition	a*
• Parentheses	(a|b)
• Empty	ε

Examples

• Integer constants

0 | (1|2|3|4|5|6|7|8|9)(0|1|2|3|4|5|6|7|8|9)*

• Identifiers

(a|…|z) (a|…|z|A|…Z|0|…|9|_|$)*

Abbreviations

ab|c	(ab)|c
[abcd]	(a|b|c|d)
[a-z]	(a|…|z)
[~X]	any character other than X
X?	X|ε
X+	X(X*)
"+"	the string itself
\+	"+"
.	[~\n], i.e., anything but newline

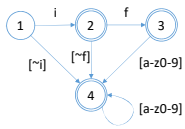
Lexical Analyzer for Tiger

- ErrorMsg/ErrorMsg.java
  Keeps track of line count, prints error message
- Parse/Lexer.java
  Lexical analyzer interface
- Parse/Main.java
  Test program that calls lexical analyzer and prints tokens
- Parse/sym.java
  Enumeration constants for tokens, generated by JavaCUP
- Parse/Tiger.lex
  JLex source code for lexical analyzer

JLex Source

```
package Parse;
import ErrorMsg.ErrorMsg;
%%
%{
...
%}
%function nextToken
...
digits=[0-9]+
%%
if    { return new Token(IF); }
...   ...
.     { error("Illegal character"); }
```

What's a DFA?



DFA Implementation

- 2-dimensional table

State	a	b	c	d	e	f	...
0	0	0	0	0	0	0	
1	2	2	2	...			
2	...						
3	...						

DFA Implementation

- Code with state variable

```
state = 0;
while (! end_of_file()) {
    switch (state) {
        case 0: ...;  break;
        case 1: ...;  state = 17;  break;
        ...
    }
}
```

DFA Implementation

- Or simply

```
state0:
    ...;
    goto state17;
state1:
    ...;
    goto state1;
...

end:
```