Lexer

COP-3402 Systems Software

Paul Gazzillo

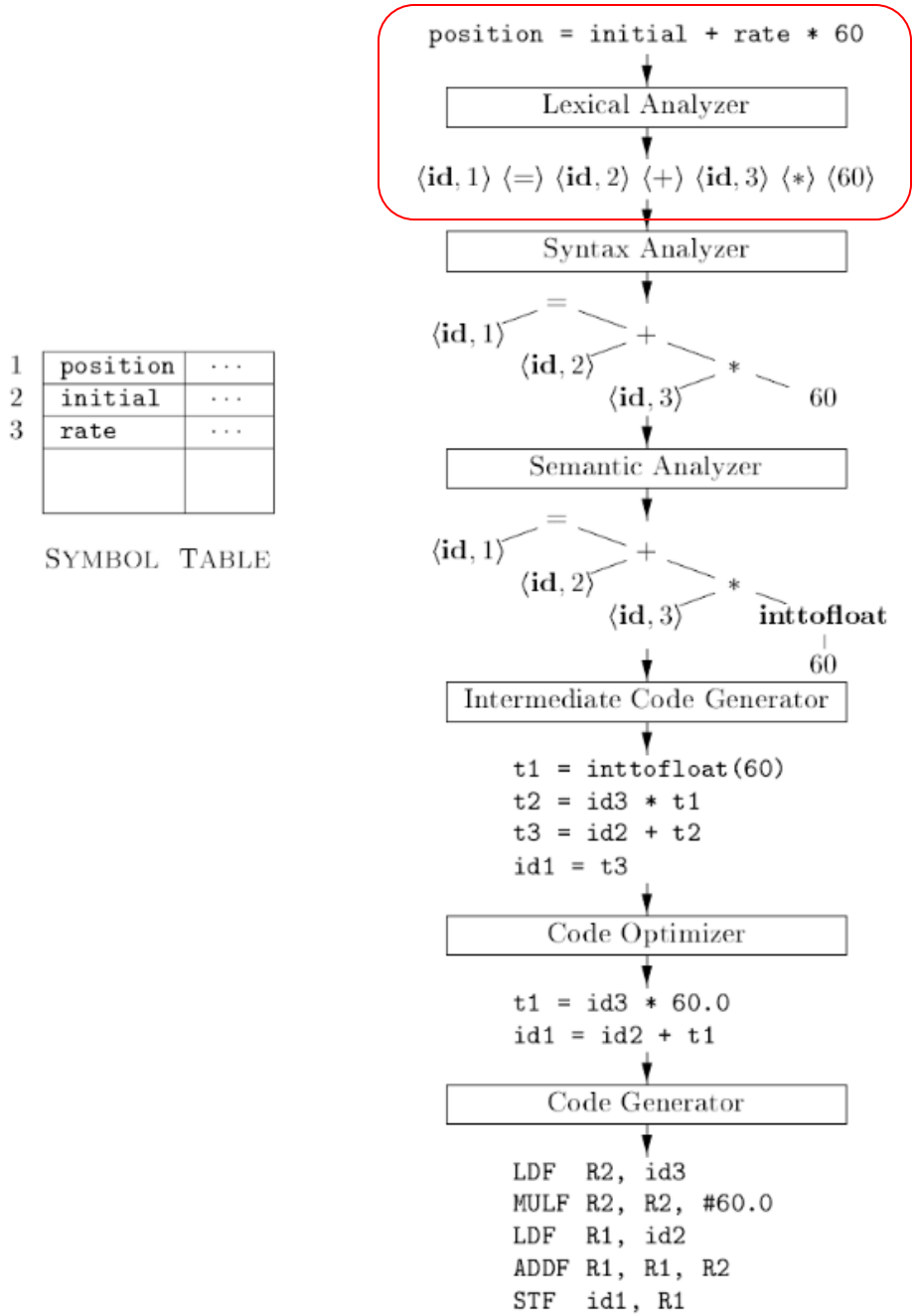

Figure 1.7: Translation of an assignment statement

Core Idea: Convert Characters to Tokens

- Regular expressions *specify* tokens
  - As sets of strings
- Finite automata *recognize* tokens
  - Via state transitions
- A lexer *produces* a stream of tokens
  - From a stream of individual characters

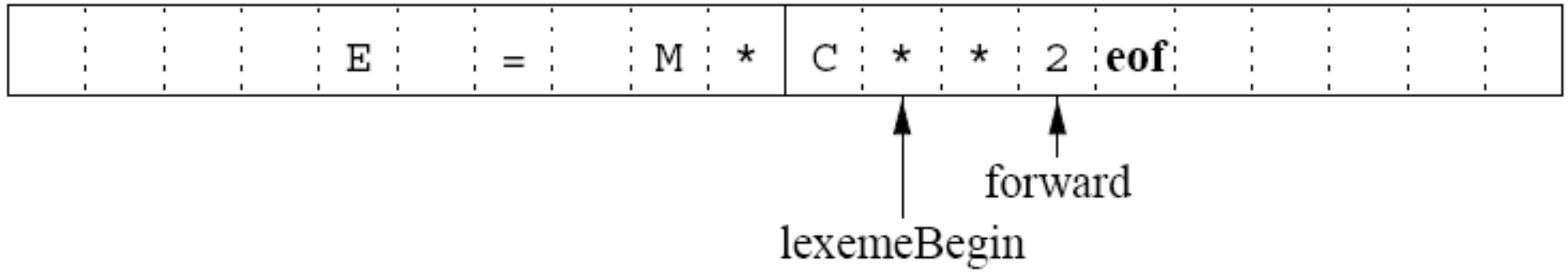

Figure 3.3: Using a pair of input buffers

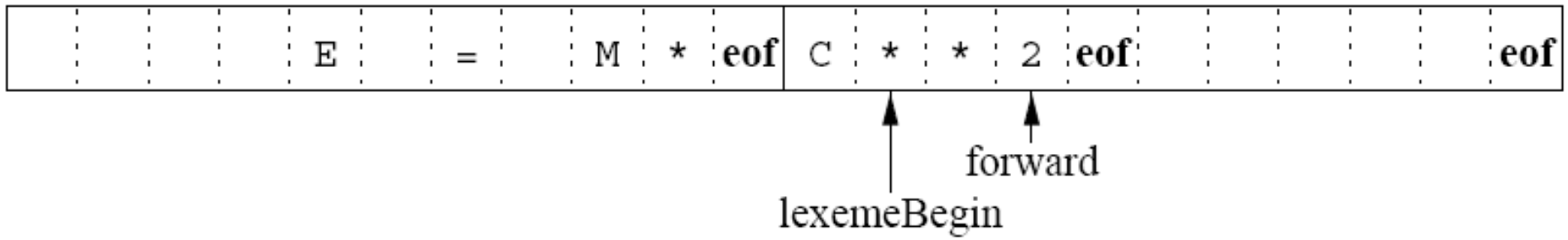

Figure 3.4: Sentinels at the end of each buffer

Token	Informal Description	Sample Lexemes
if	characters `i`, `f`	`if`
else	characters `e`, `l`, `s`, `e`	`else`
comparison	`<` or `>` or `<=` or `>=` or `==` or `!=`	`<=`, `!=`
id	letter followed by letters and digits	`pi`, `score`, `D2`
number	any numeric constant	`3.14159`, `0`, `6.02e23`
literal	anything but `"`, surrounded by `"`'s	`"core dumped"`

Figure 3.2: Examples of tokens

LEXEMES	TOKEN NAME	ATTRIBUTE VALUE
Any *ws*	–	–
`if`	**if**	–
`then`	**then**	–
`else`	**else**	–
Any *id*	**id**	Pointer to table entry
Any *number*	**number**	Pointer to table entry
`<`	**relop**	`LT`
`<=`	**relop**	`LE`
`=`	**relop**	`EQ`
`<>`	**relop**	`NE`
`>`	**relop**	`GT`
`>=`	**relop**	`GE`

Figure 3.12: Tokens, their patterns, and attribute values

Implementing a Lexer

- Regex compiler
  - Regex -> NFA -> DFA -> state table
- Hand-coded lexer
  - Lexer reads one character at a time, e.g., fgetc
  - Concatenation is sequence of statements
  - Union is a conditional
  - Closure is a while loop

Concatenation is sequence of statements, e.g., ab

```
char c;
c = fgetc(lexerin);
assert 'a' == c
c = fgetc(lexerin);
assert 'b' == c
```

Union is a conditional, e.g., a|b

```
char c;
c = fgetc(lexerin);
if ('a' == c) {
  // ...
} else if ('b' == c) {
  // ...
} else {
  error(...)
}
```

Closure is a while loop, e.g., a*

```
char c;
c = fgetc(lexerin);
while('a' == c) {
  c = fgetc(lexerin);
}
// c is now the next character
```

Demo

Lexing PL/0

EBNF Standardizes Language Specifications

- EBNF = Extended Backus-Naur Format
- Different notation for regular expressions
  - **a* -> { a }**
  - **a|ε -> [a]**
  - ab -> a b
  - a | b -> a | b

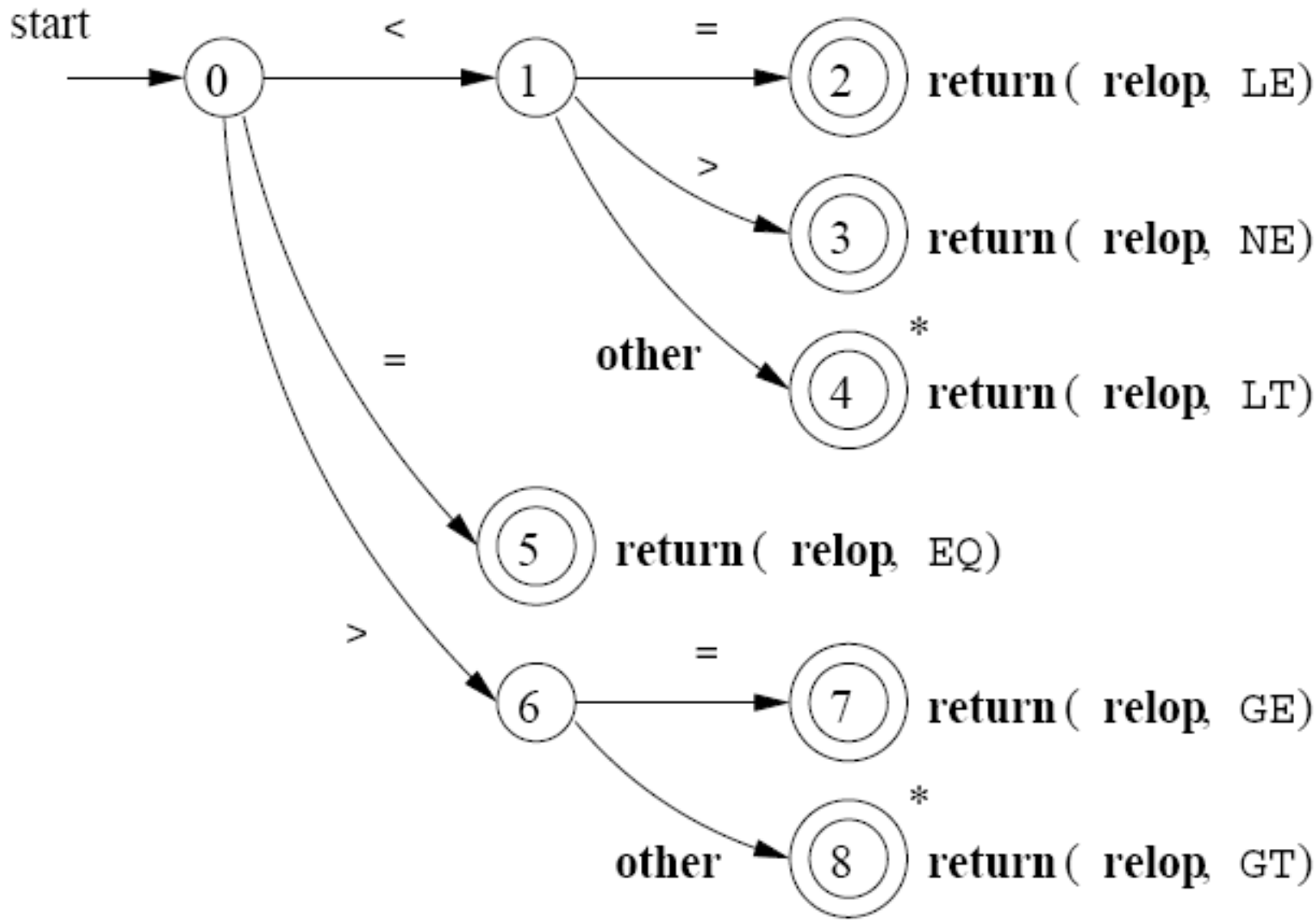

Figure 3.13: Transition diagram for **relop**