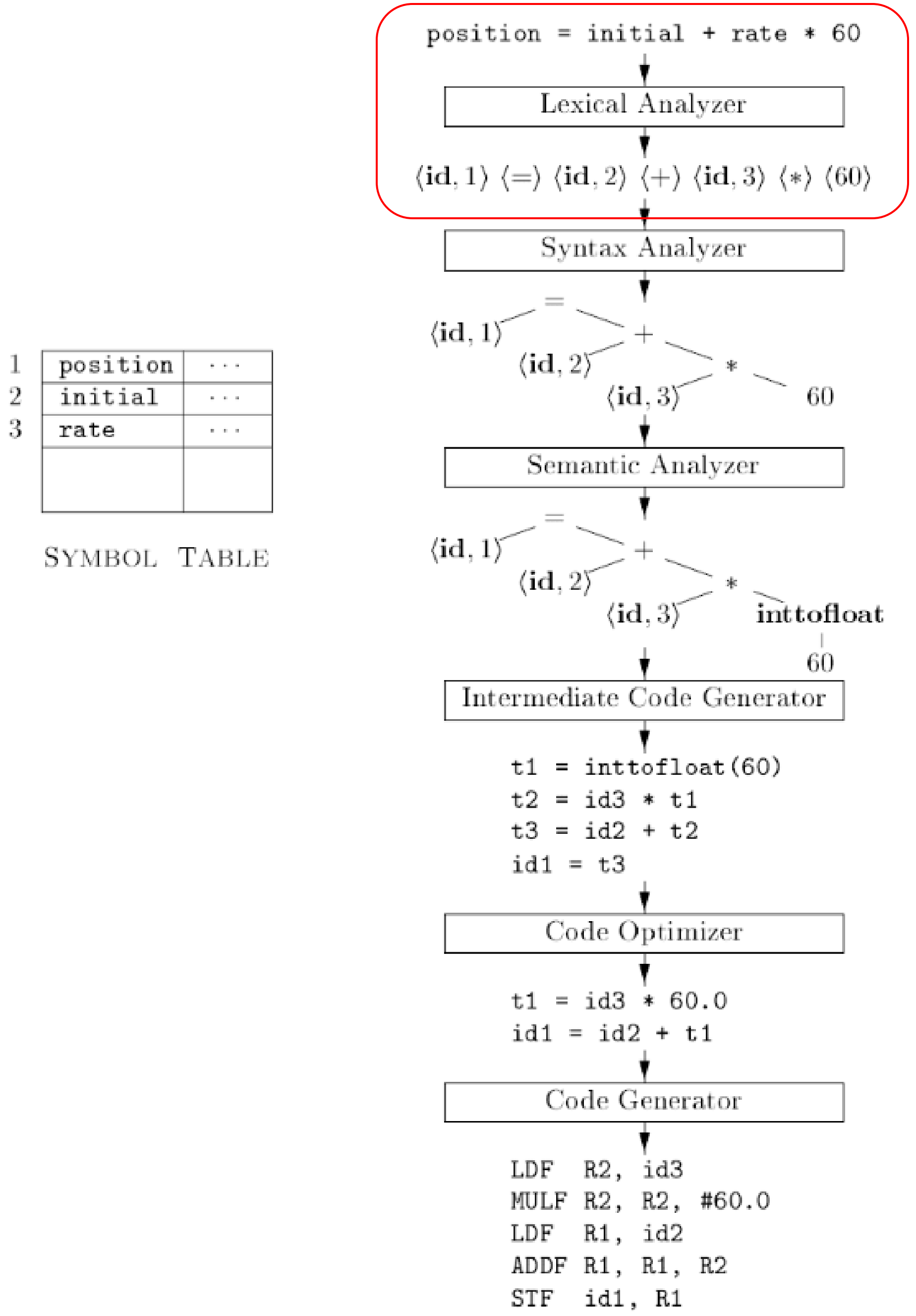
# Regular Expressions

COP-3402 Systems Software

Paul Gazzillo

# Core Problem: Specifying a Language

- The machine does not "understand" language
- Compilers process and translate language
- How do we specify a language?



```
position = initial + rate * 60
```

Lexical Analyzer

⟨**id**, 1⟩ ⟨=⟩ ⟨**id**, 2⟩ ⟨+⟩ ⟨**id**, 3⟩ ⟨∗⟩ ⟨60⟩

Syntax Analyzer

Semantic Analyzer

Intermediate Code Generator

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

Code Optimizer

```
t1 = id3 * 60.0
id1 = id2 + t1
```

Code Generator

```
LDF  R2, id3
MULF R2, R2, #60.0
LDF  R1, id2
ADDF R1, R1, R2
STF  id1, R1
```

|   |   |   |
|---|---|---|
| 1 | position | ··· |
| 2 | initial | ··· |
| 3 | rate | ··· |
|   |   |   |

SYMBOL TABLE

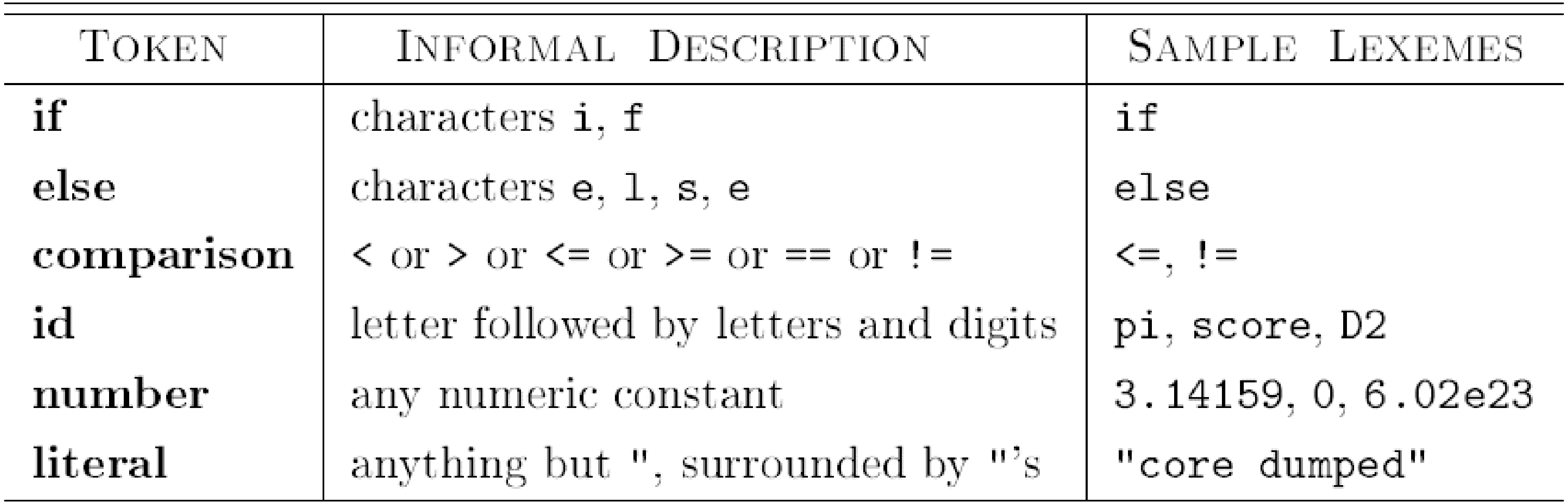
Figure 1.7: Translation of an assignment statement

# Lexical Analysis Picks out Words

- Words are groups of characters
- Machine stores text as sequence of bytes
- Lexical analysis groups bytes/characters into words or *tokens*

# Demo

Using regular expressions for searching text



| 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

# Formal Languages

- An *alphabet* is a **finite** set of symbols
  - e.g., ASCII characters
- A *string* over an alphabet is a **finite** sequence of symbols
  - e.g., tokens in PL/0
- ε is the empty string
- A *language* is a **possibly infinite** set of strings over an alphabet
  - Ø is the empty language

| Operation | Definition and Notation |
|---|---|
| *Union* of $L$ and $M$ | $L \cup M = \{s \mid s \text{ is in } L \text{ or } s \text{ is in } M\}$ |
| *Concatenation* of $L$ and $M$ | $LM = \{st \mid s \text{ is in } L \text{ and } t \text{ is in } M\}$ |
| *Kleene closure* of $L$ | $L^* = \cup_{i=0}^{\infty} L^i$ |
| *Positive closure* of $L$ | $L^+ = \cup_{i=1}^{\infty} L^i$ |

Figure 3.6: Definitions of operations on languages

# Regular Languages

- A *regular language* is a language defined using
  - Union "a|b" means *a or b*
  - Concatenation "ab" means *a followed by b*
  - Closure "a*" means a repeated *zero or more times*
- *Closed* under union, concatenation, and closure
  - i.e., operations over regular languages results in regular languages

# Examples

| Expression | Matches | Example |
|---|---|---|
| $c$ | the one non-operator character $c$ | `a` |
| $\backslash c$ | character $c$ literally | `\*` |
| "$s$" | string $s$ literally | `"**"` |
| . | any character but newline | `a.*b` |
| ^ | beginning of a line | `^abc` |
| $\$$ | end of a line | `abc$` |
| $[s]$ | any one of the characters in string $s$ | `[abc]` |
| $[\hat{}s]$ | any one character not in string $s$ | `[^abc]` |
| $r*$ | zero or more strings matching $r$ | `a*` |
| $r+$ | one or more strings matching $r$ | `a+` |
| $r?$ | zero or one $r$ | `a?` |
| $r\{m,n\}$ | between $m$ and $n$ occurrences of $r$ | `a{1,5}` |
| $r_1 r_2$ | an $r_1$ followed by an $r_2$ | `ab` |
| $r_1 \mid r_2$ | an $r_1$ or an $r_2$ | `a\|b` |
| $(r)$ | same as $r$ | `(a\|b)` |
| $r_1/r_2$ | $r_1$ when followed by $r_2$ | `abc/123` |

Figure 3.8: Lex regular expressions

# 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

# PL/0 Lexical Specification

project/grammar.md