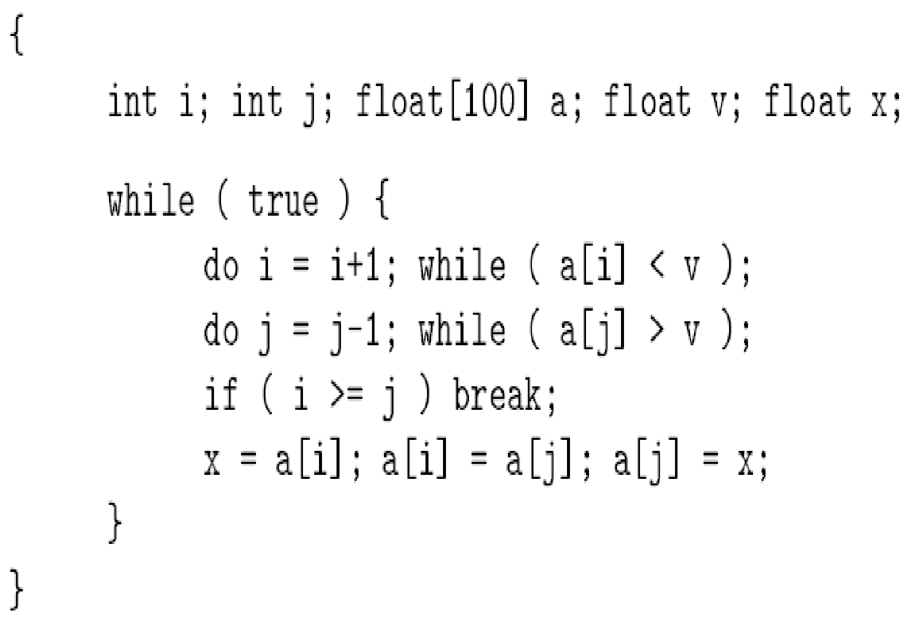
CS480
Translators

Introduction to Compilers

Overview - Chap. 2

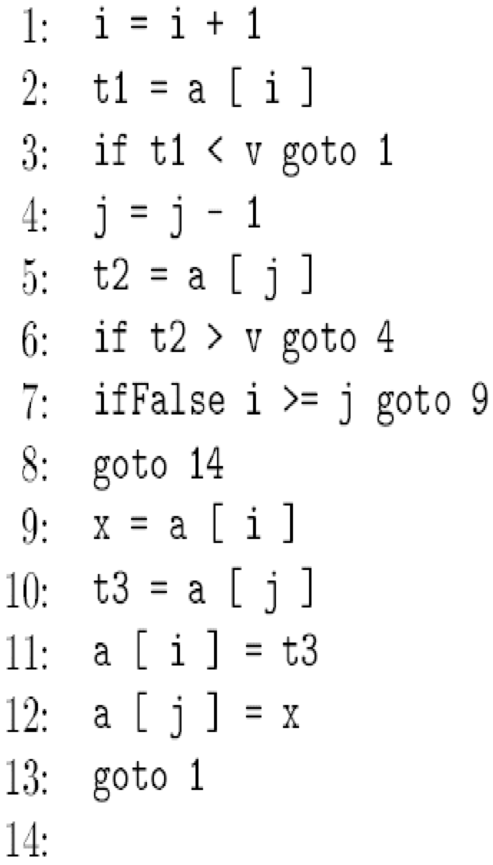
Translation



```
{
    int i; int j; float[100] a; float v; float x;

    while ( true ) {
        do i = i+1; while ( a[i] < v );
        do j = j-1; while ( a[j] > v );
        if ( i >= j ) break;
        x = a[i]; a[i] = a[j]; a[j] = x;
    }
}
```

Figure 2.1: A code fragment to be translated



```
 1:  i = i + 1
 2:  t1 = a [ i ]
 3:  if t1 < v goto 1
 4:  j = j - 1
 5:  t2 = a [ j ]
 6:  if t2 > v goto 4
 7:  ifFalse i >= j goto 9
 8:  goto 14
 9:  x = a [ i ]
10:  t3 = a [ j ]
11:  a [ i ] = t3
12:  a [ j ] = x
13:  goto 1
14:
```

Figure 2.2: Simplified intermediate code for the program fragment in Fig. 2.1

What is syntax-directed translation?

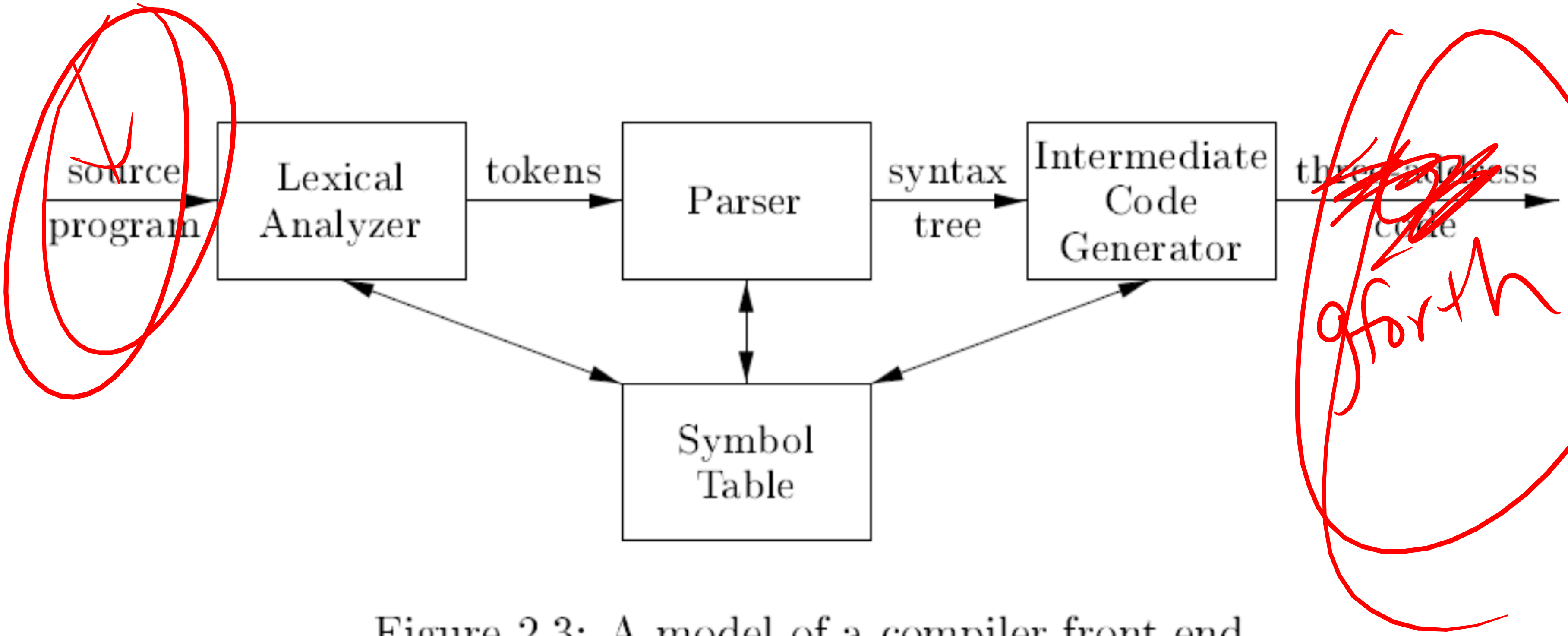
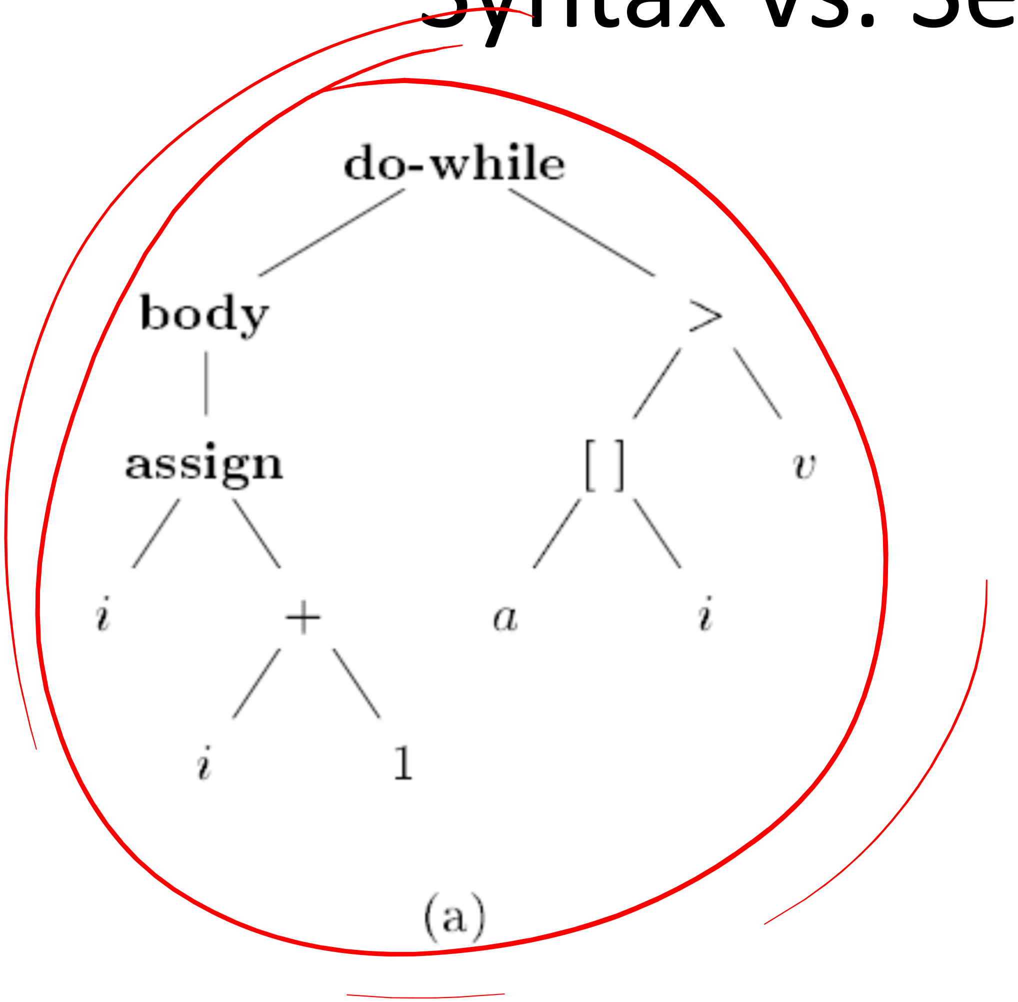


Figure 2.3: A model of a compiler front end

Syntax vs. Semantics



(a)

```
1:  i = i + 1
2:  t1 = a [ i ]
3:  if t1 < v goto 1
```

(b)

Figure 2.4: Intermediate code for “do i = i + 1; while (a[i] < v) ;”

regular expressions - id, Keywords, op, #s

floats wholes

Syntax

compilers

strings

- Regular grammars
- Context-free grammars
- BNF notation
- Example:

stmt -> **if** (*expr*) *stmt* **else** *stmt*

term.

nonterm

Production

binop -> + | - |

What is a CFG?

- Set of terminals (usually bold)
- Set of nonterminals (italic and/or capitalized)
- Set of productions (contains ->)
- Start symbol

↑ mostly it's 1st nonterm prod

Example CFG

list -> list + digit

list -> list - digit

list -> digit

~~*digit ->* 0~~ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ε

op
id

- What can we get using this grammar?
- Can we get the empty string, i.e. ε?

Derivation

- What is a language? — set of strings derived by the grammar
- Deriving strings:
  - Start symbol
  - Replace nonterminals

- What if a string can't be derived?

Parsing

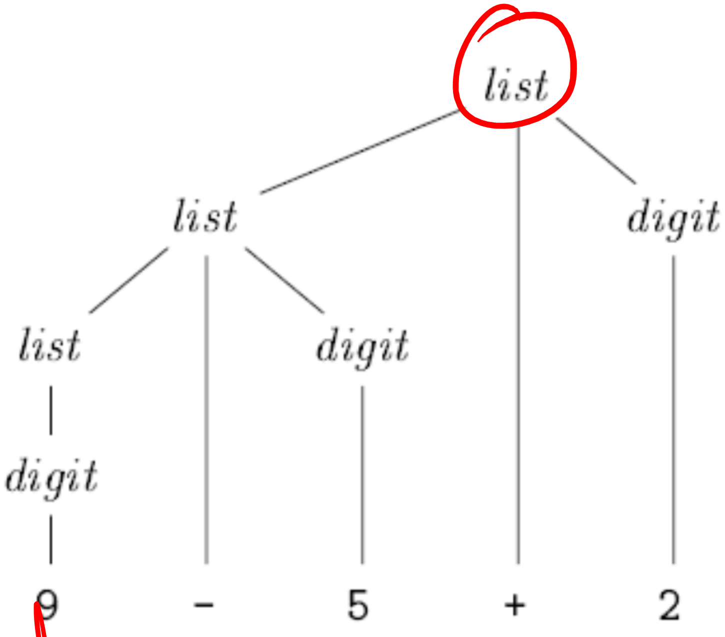


Figure 2.5: Parse tree for 9−5+2 according to the grammar in Example 2.1

1st 2nd

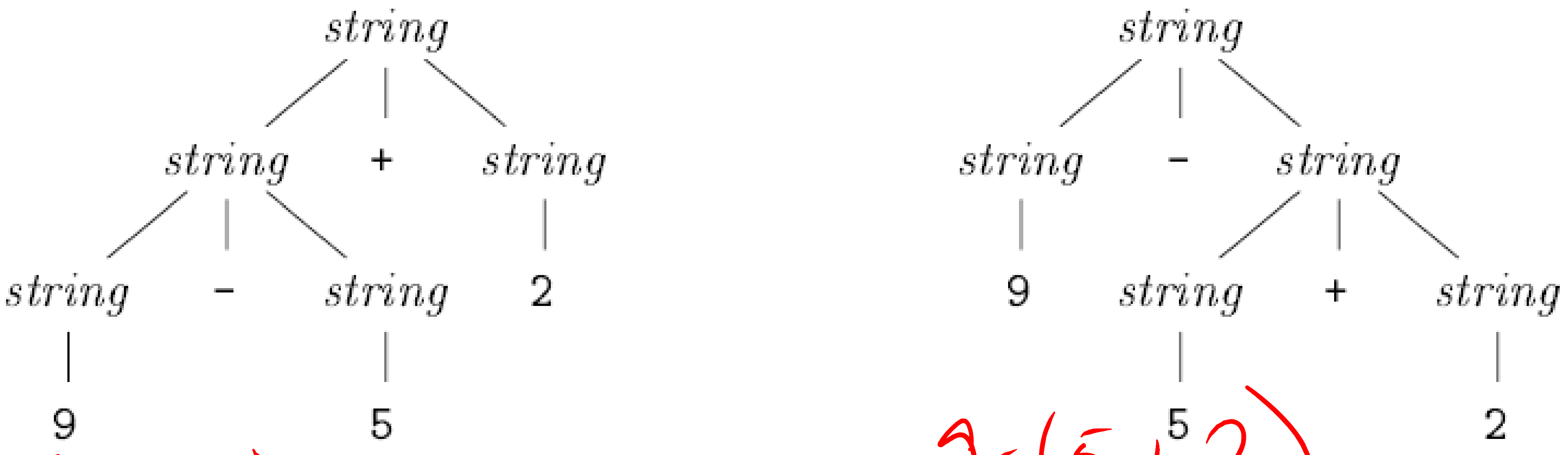
list -> *list* + *digit* | *list* - *digit* | *digit*

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Ambiguity

- Suppose we used:

string -> *string* + *string* | *string* - *string* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



(9-5)+2

9-(5+2)

Figure 2.6: Two parse trees for 9-5+2

Class Example

- What language is generated by these?

S -> 0 S 1 | 0 1

at least one zero followed by any numbers of zeros followed the same # of 1s

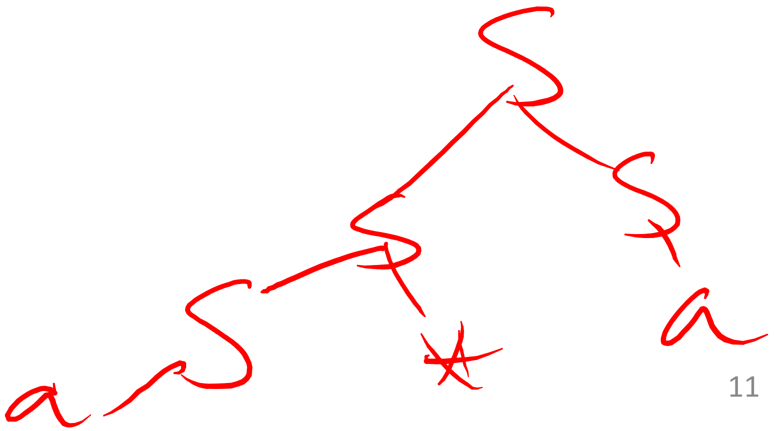
S -> S (S) S | ε

S -> **a** | S S | S *

a

- Which are ambiguous?

a * a



Associativity

list -> *list* + *digit* | *list* - *digit* | *digit*

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Vs.

right -> *letter* = *right* | *letter*

letter -> a | b | … | z

right assoc.

- What do you notice about these grammars?

Associativity

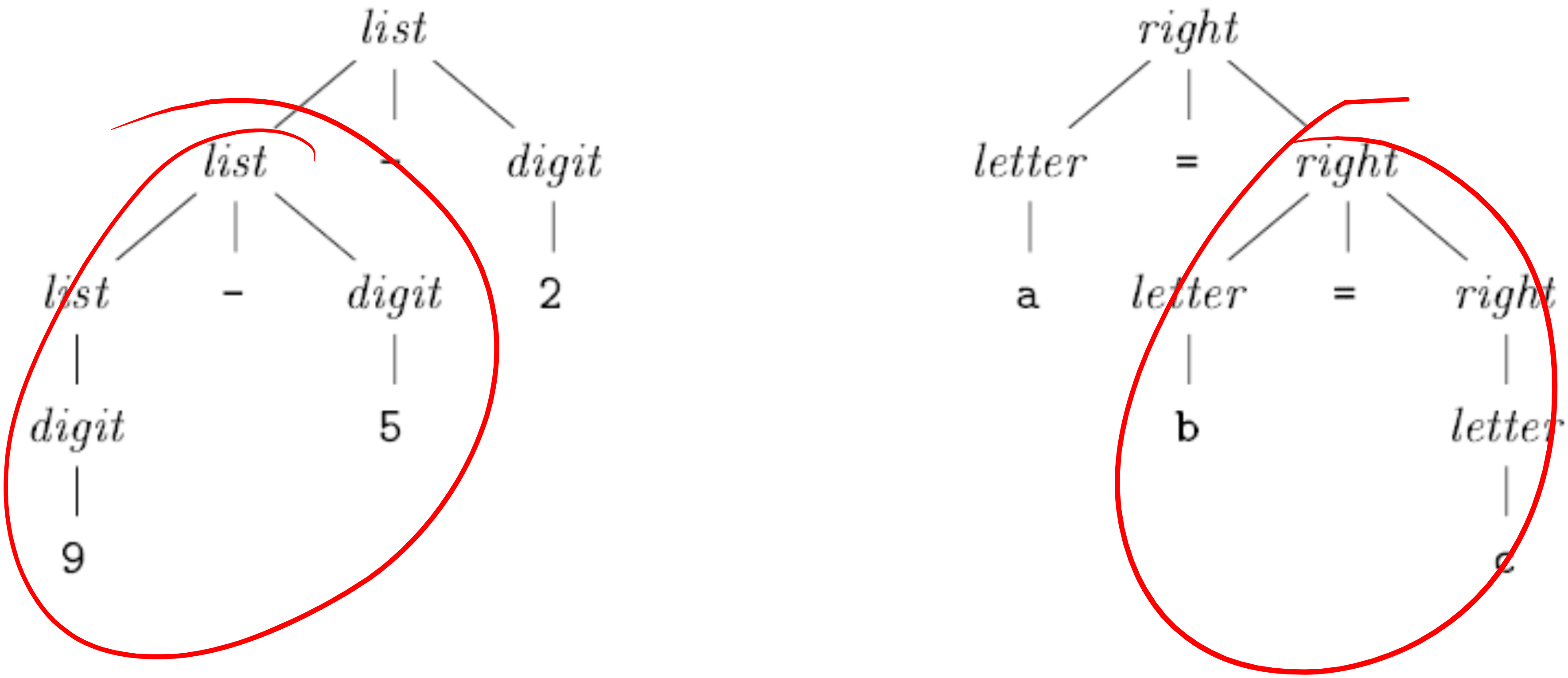


Figure 2.7: Parse trees for left- and right-associative grammars

Associativity vs. Precedence

- What happens with more than one op?
  - How are * and + alike and different?
- Need to resolve ambiguity

left associative: + -

left associative: * /

expr -> *expr* + *term* | *expr* - *term* | *term*

term -> *term* * *factor* | *term* / *factor* | *factor*

factor -> **digit** | (*expr*)