Lecture topics

I Syntax definition
II Syntax-directed translation
III Recursive-descent parsers

I Syntax definition

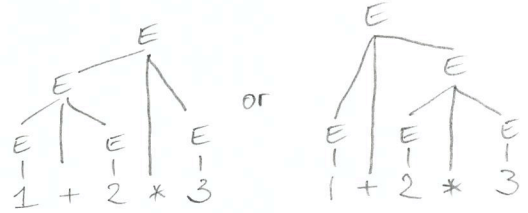
Example grammar G_1:

$E \to E+E \mid E-E \mid E*E \mid E/E \mid 0 \mid 1 \mid \ldots \mid 9$

Grammar concepts:

- nonterminals (e.g. E)
  - syntactic variables
  - defined in syntax-analyzer grammar
- terminals (e.g. +, -, *, /, 0, 1, ..., 9)
  - token
  - defined in lexical-analyzer grammar
- rule (head → body)
  - define nonterminal
  - a.k.a. "production"
- sequence (e.g. E+E)
- alternation (e.g. 0 | 1)
- empty string (ε = epsilon)
- start symbol (e.g. E)
- not in syntax grammar: +, *, ?, (...), [...]

Parse trees:

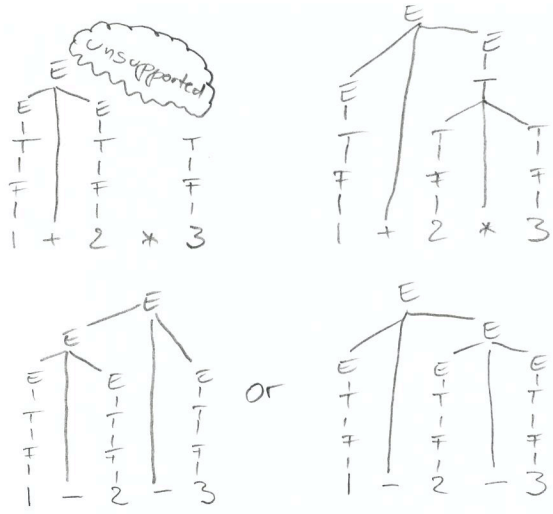


ambiguous grammar:
multiple different parse trees for same input

Precedence = binding strength
*, / have higher precedence than +, -
*, / bind stronger than +, -

"layered-grammar" technique

G_2: $E \to E+E \mid E-E \mid T$
$T \to T*T \mid T/T \mid F$
$F \to 0 \mid 1 \mid \ldots \mid 9$

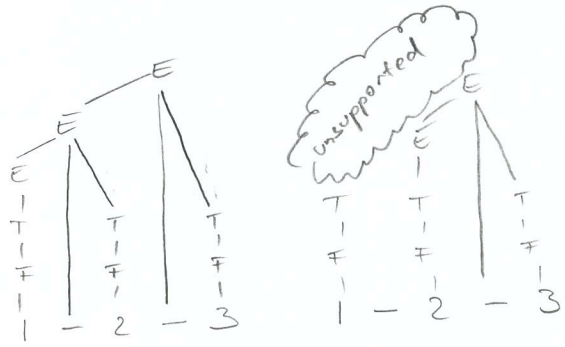


Still ambiguous!

Associativity = grouping direction
+, -, *, / are all left-associative (group to the left)

"asymmetric layered grammar" technique

G_3: $E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to 0 \mid 1 \mid \ldots \mid 9$



Parse tree = concrete syntax tree
AST = abstract syntax tree

II Syntax-directed translation

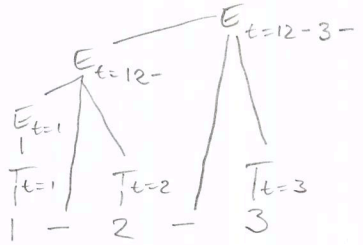
Running example:

infix	postfix
3+3	3 3 +
1+2*3	1 2 3 * +
1-2-3	1 2 - 3 -

Task: translate infix → postfix automatically

Translation scheme G4:

$E \to E_1 + T$	$E.t = E_1.t \parallel T.t \parallel '+'$
$\mid E_1 - T$	$E.t = E_1.t \parallel T.t \parallel '-'$
$\mid T$	$E.t = T.t$
$T \to 0$	$T.t = '0'$
$\mid 1$	$T.t = '1'$
$\mid \ldots$	$\ldots$
$\mid 9$	$T.t = '9'$



Synthesized attribute
Parent computed from child
During parsing, e.g.
as part of return value

Inherited attribute
Child computed from parent
After parsing, during tree traversal,
e.g., as field of AST node

III Recursive-descent parsers

First attempt: implementation of G4

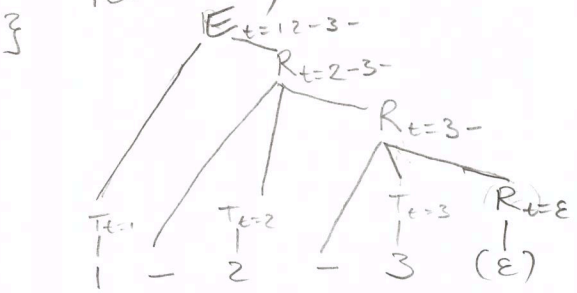
```
String pE() {
    switch (lookahead) {
        case '0': case '1': ... : case '9':
            return pT();
    }
    String e1 = pE();
    switch (lookahead) {
        case '+': {
            match('+');
            String t = pT();
            return e1 + t + '+';
        }
        case '-': {
            match('-');
            String t = pT();
            return e1 + t + '-';
        }
    }
    return "";
}
```

Problem: left-recursion,
leads to stack overflow

"right-recursive predictive grammar" technique

$E \to T\ R$	$E.t = T.t \parallel R.t$
$R \to +\ T\ R_1$	$R.t = T.t \parallel '+' \parallel R_1.t$
$\mid -\ T\ R_1$	$R.t = T.t \parallel '-' \parallel R_1.t$
$\mid \varepsilon$	$R.t = \varepsilon$
$T \to 0$	$T.t = '0'$
$\mid 1$	$T.t = '1'$
$\mid \ldots$	$\ldots$
$\mid 9$	$T.t = '9'$

```
String pE() {
    String t = pT();
    String r = pR();
    return t + r;
}
String pR() {
    switch (lookahead) {
        case '+': {
            match('+');
            String t = pT();
            String r1 = pR();
            return t + '+' + r1;
        }
        case '-': {
            match('-');
            String t = pT();
            String r1 = pR();
            return t + '-' + r1;
        }
    }
    return "";
}
```



Reminders

- hw 1 due Fr 9/16 at 1pm
- hw 2 due Fr 9/23 at 1pm
- read 3.1-3.6 for lecture on We 9/21
- example solutions