Simple translator (pg. 1)

Lecture topics

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

I. Syntax definition

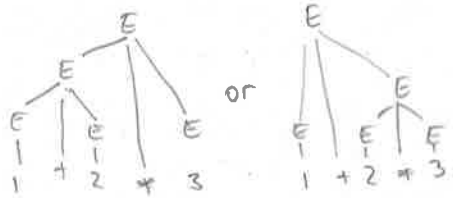
Example grammar G_1:

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

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:

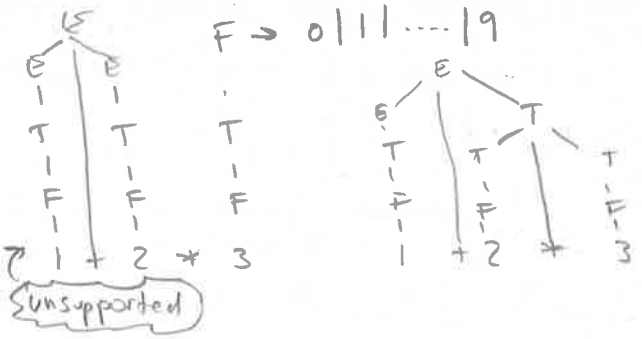
- multiple different parse trees
- from same input

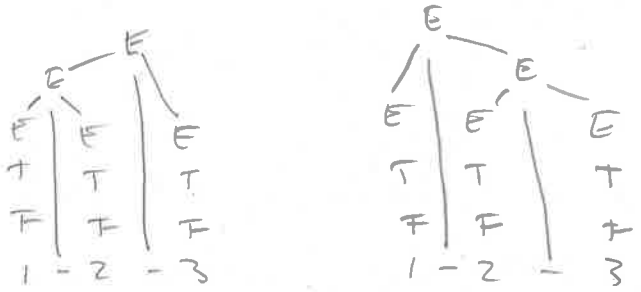
Precedence = binding strength

- *, / have higher precedence than +, -
- *, / bind stronger than +, -

"layered-grammar" technique:

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





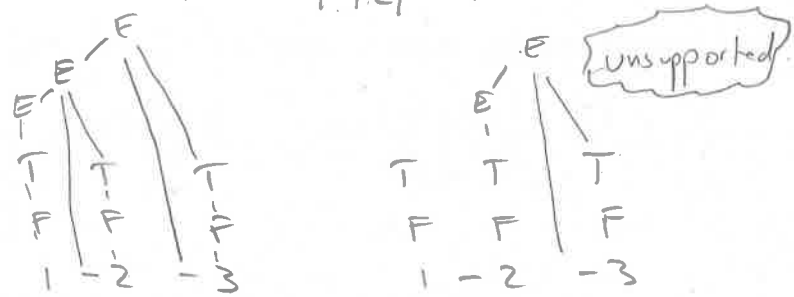
still ambiguous!

Associativity = grouping direction

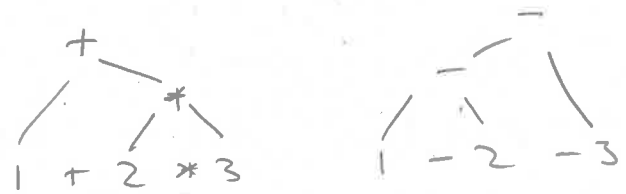
+, -, *, / are all left associative (group to the left)

"asymetric layered grammar" technique:

G_3: $E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to 0 \mid 1 \mid 2 \mid \ldots 9$



Parse tree = concrete syntax tree
AST = abstract syntax tree



II Syntax-directed translation (pg 2)

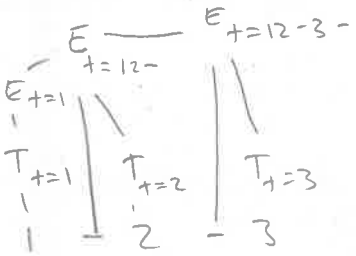
Running example:

infix	postfix
3+3	33+
1+2*3	12 3* -
1-2-3	12 - 3 -

Task: translate infix → postfix

Translation scheme G_4:

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



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

Inherited attributes
- child copies from parent
- After parsing, during tree traversal
  - e.g., as field of AST node

III Recursive-descent Parsers (pg. 3)

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

$E \to TR$	$E.t \parallel R.t$
$R \to +TR_1$	$R.t = T.t \parallel '+' \parallel R_1.t$
$\mid -TR_1$	$R.t = T.t \parallel '-' \parallel R_1.t$
$\mid \varepsilon$	$R.t = \varepsilon$
$T \to 0$	$T.t = '0'$
$\mid 1$	$T.t = '1'$
$\mid \ldots$	$\ldots$
$\mid 9$	$T.t = '9'$

```
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 "";
}
```