Abstract Syntax Tree

Lembit Jürimägi

Production Rules (Regular Grammar)

- Rules can be recursive, meaning the same nonterminal can appear on both sides of the rule
- However a nonterminal can appear only once on the right side and either left or right from nonterminal for every rule in grammar (left-recursive vs right-recursive)

G = {S, N, T, R}; N : {S, A}; T : {a}; R :

S -> A

A -> aA

A -> Ø

- During parsing this can be represented with a straight sequence:

aaa -> aaaØ -> aaaA -> aaA -> aA -> A -> S

Production Rules (Contex-Free Grammar)

- Rules can be recursive and there is no restriction on the number and order of nonterminals on the right side of production rule

G = {S, N, T, R}; N : {S, E}; T : {+, –, x, {num} }; R :

S -> E

E -> {num}

E -> E + E

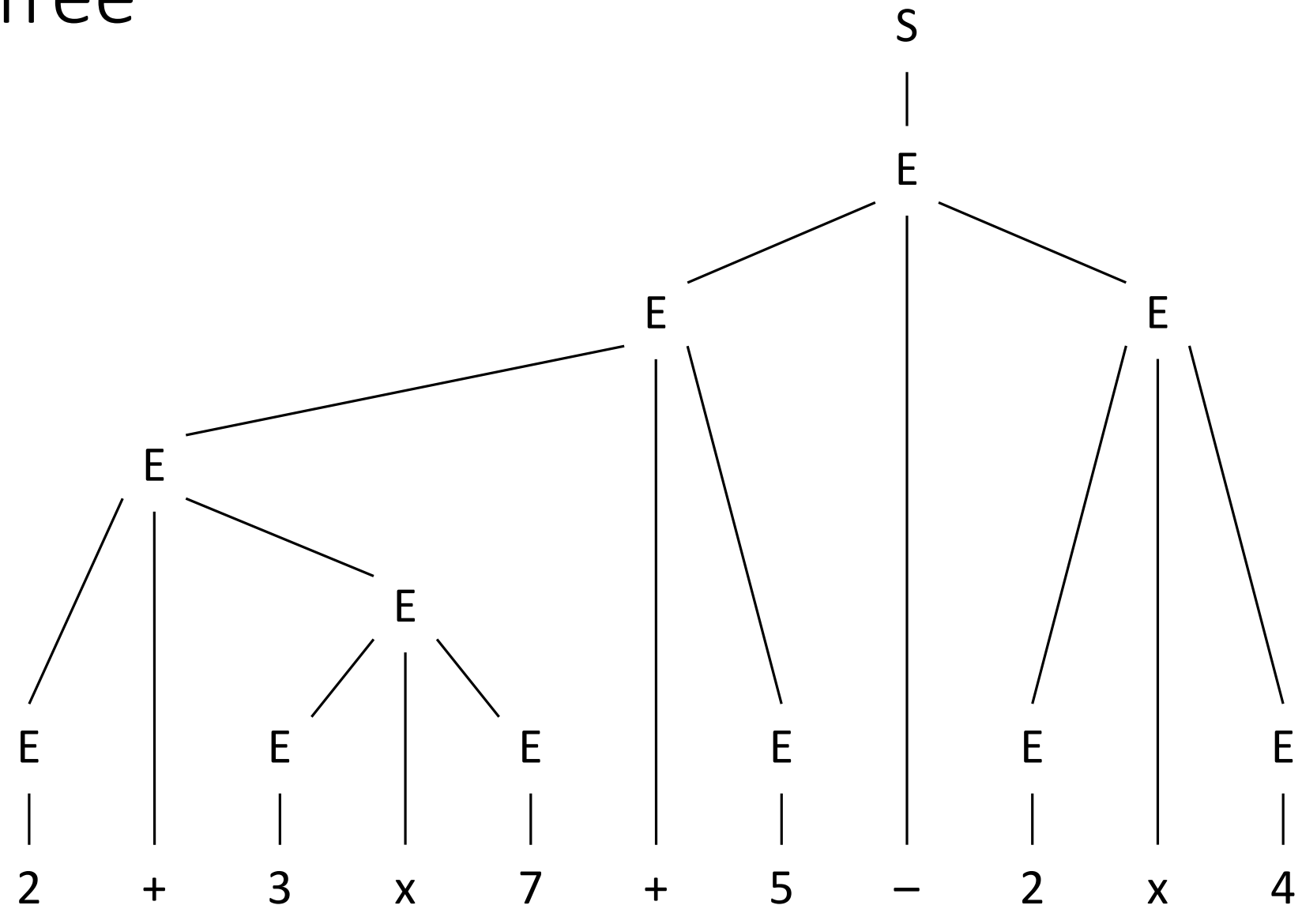
E -> E – E

E -> E x E

- A sequence is no longer enough to show the parsing process, a tree is needed

2 + 3 x 7 + 5 – 2 x 4

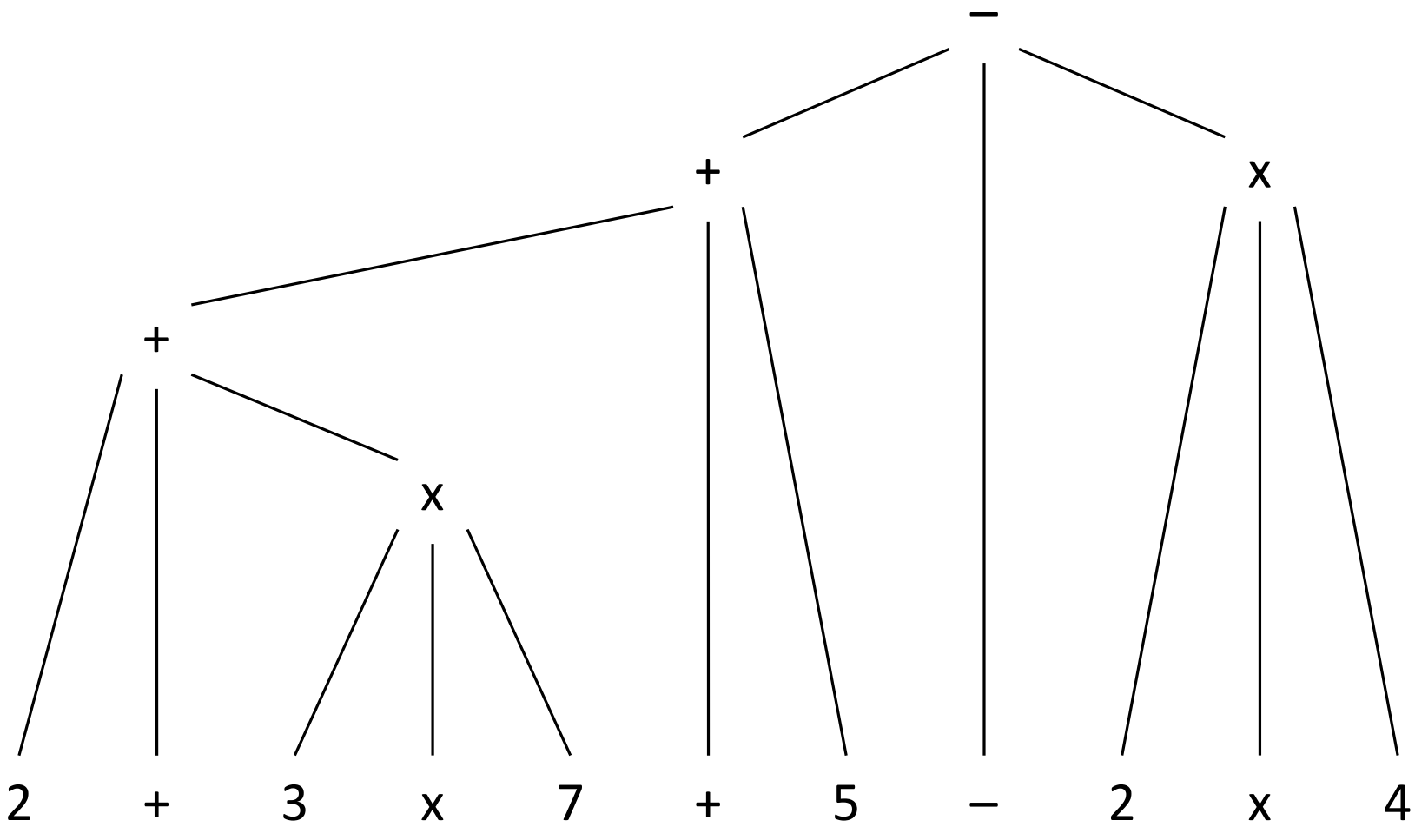
Parse Tree



Parse Tree

- Contains all terminal and nonterminal elements
- Original sentence can be reconstructed
- Useful for visualizing the parse process
- Parser usually omits building it
- If grammar is unambiguous then operations are correctly ordered

Abstract Syntax Tree



Abstract Syntax Tree

- Gets rid of most nonterminal elements and unimportant terminal elements
- Primary focus is execution so some terminals may be altered
- Original sentence usually cannot be reconstructed
- Necessary to build for most interpreters and all compilers

Multiple Token Types in Bison and Flex

- By default the token type is int
- This can be changed by union clause:

```
%union{
      int val;
      char *s;
      t_node *node;
}
```

- Tokens (and nonterminals) can now have a type:

```
%token <val> NUM
%type <node> expr
```

- yylval becomes union so Flex rules need to account for that:

```
yylval.val = atoi(yytext);
```

Dangling Else Problem

- Suppose we have grammar:

```
stmt  : expr ';'
            | IF '(' expr ')' THEN stmt
            | IF '(' expr ')' THEN stmt ELSE stmt
```

- The IF-ELSE has 2 stmt parts and each could be another IF stmt

```
IF (a < b) THEN IF (b < c) THEN PRINT c; ELSE PRINT a;
```

- Which IF does the ELSE belong to?