Compiler Construction

(Week 3, Lecture 1)

Parser

- Takes in the stream of tokens, recognizes context- free syntax and reports errors.
- Guides context-sensitive “semantic” analysis for tasks like type checking.
- Builds IR for source program.

Parser

- The syntax of most programming languages is specified using Context-Free Grammars (CFG).
- Context- free syntax is specified with a grammar G=(S,N,T,P) where
  - S is the start symbol
  - N is a set of non-terminal symbols
  - T is set of terminal symbols or words
  - P is a set of productions or rewrite rules

CFG

```
1. goal → expr
2. expr → expr op term
3.      | term
4. term → number
5.      | id
6. op   → +
7.      | -
```

S = goal
T = { number, id, +, -}
N = { goal, expr, term, op}
P = { 1, 2, 3, 4, 5, 6, 7}

Parser

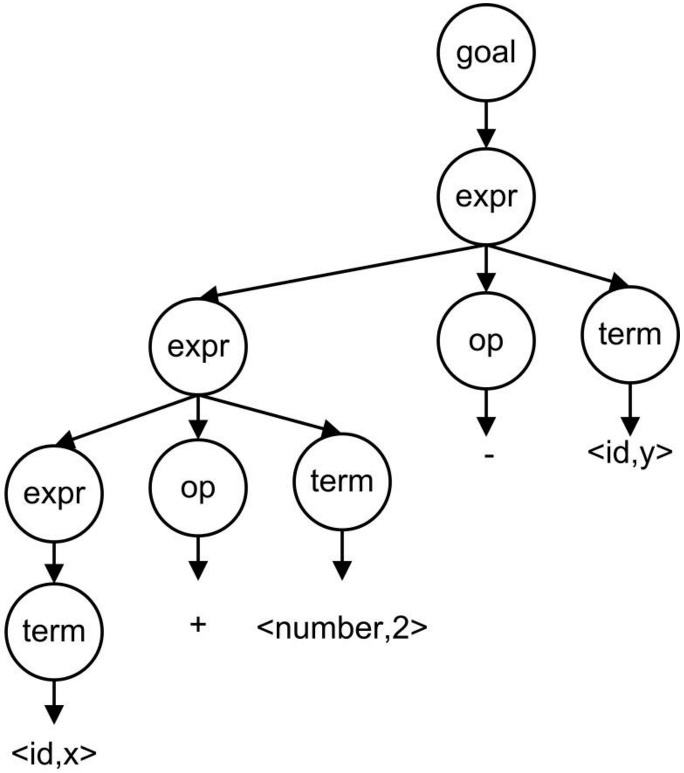
- Given a CFG, sentences can be derived by repeated substitution.
- Example: x + 2 – y

Parser

Production	Result
	goal
1: goal → expr	expr
2: expr → expr op term	expr op term
5: term → id	expr op y
7: op → –	expr - y
2: expr → expr op term	expr op term – y
4: term → number	expr op 2 – y
6: op → +	expr + 2 – y
3: expr → term	term + 2 – y
5: term → id	x + 2 – y

Parse Tree

- Captures all rewrite during the derivation.
- The derivation can be extracted by starting at the root of the tree and working towards the leaf nodes.



Abstract Syntax Tree

- The parse tree contains a lot of unneeded information.
- Compilers often use an abstract syntax tree (AST).