Programming Paradigms

Syntax (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- **Specifying syntax**
  - Regular expressions
  - Context-free grammars
- **Scanning**
- **Parsing**

Are regular expressions enough?

Specifying arithmetic expressions

↳ E.g.: $\boxed{5+7}$

$\boxed{(\boxed{5+7})+6}$

$\boxed{(\boxed{(\boxed{5+7})+6})-23}$

Each $\boxed{}$ is an arithmetic expression

→ Want: Nesting

→ Need recursion

Context-free Grammars

$\approx$ Regular expressions + Recursion

Example: Arithmetic expressions

expr $\rightarrow$ `id` | `number` | **expr op expr** | `(` **expr** `)`

op $\rightarrow$ `+` | `-` | `*` | `/`

Context-free Grammars

≈ **Regular expressions + Recursion**

Example: Arithmetic expressions

expr → `id` | `number` | **expr op expr** | `(` **expr** `)`

op → `+` | `-` | `*` | `/`

Non-terminals

Context-free Grammars

≈ **Regular expressions + Recursion**

Example: Arithmetic expressions

expr → `id` | `number` | **expr op expr** | `(` **expr** `)`

op → `+` | `-` | `*` | `/`

Non-terminals

Terminals = tokens of the PL

Context-free Grammars

≈ **Regular expressions + Recursion**

Example: Arithmetic expressions

expr → `id` | `number` | **expr op expr** | `(` **expr** `)`

op → `+` | `-` | `*` | `/`

Recursion allows for nesting expressions

Definition of CFGs

$G = (N, T, R, s)$

- N .. finite set of non-terminals
- T .. finite set of terminal
  - = alphabet of the languge
  - = (for PLs) tokens of the language
- R .. finite relation from N to $(N \cup T)^*$
  - = production rules
- s .. start symbol

Extension of basic definition

- Kleene star
  - ↳ E.g., id_list → id (,id)* ←---- means zero & more
  - ↳ short-hand for
    id_list → id
    id_list → id_list, id
- Kleene plus
  - ↳ Same, but <u>one</u> or more
- Vertical bar
  - ↳ E.g., op → + | − | * | /
  - ↳ Short-hand for op → +
    op → −
    ⋮

Derivations

Create concrete strings from the grammar

- Begin with start symbol
- Repeat until no non-terminals remain:
  - Choose non-terminal and a production with this non-terminal on the left-hand side
  - Replace it with right-hand side of the production (choose one option if multiple options)

Example:

expr → id | number | expr | (expr) | expr op expr

op → + | - | * | /

Derivation of foo * x + bar

"derives"

expr ⇒ expr op <u>expr</u>

⇒ expr <u>op</u> id

⇒ <u>expr</u> + id

⇒ expr op <u>expr</u> + id

⇒ expr <u>op</u> id + id

⇒ <u>expr</u> * id + id

⇒ id * id + id

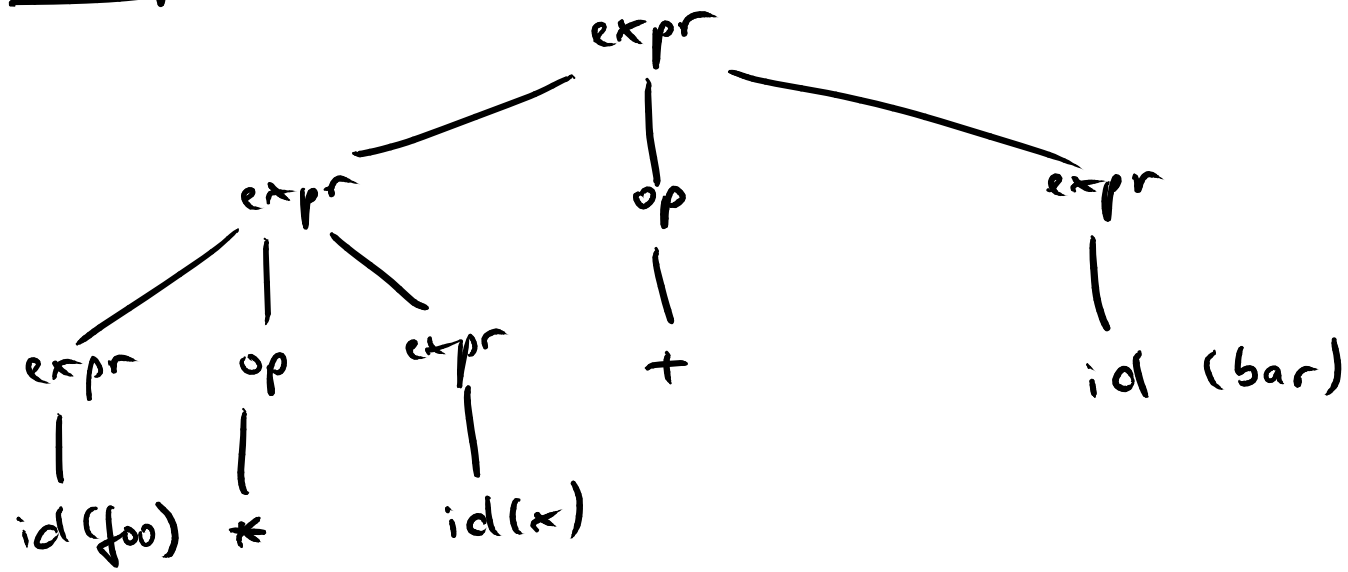
foo x bar

Parse Trees

Tree-structured representation of a derivation

- Root = Start symbol
- Leaf nodes = Tokens that result from derivation
- Intermediate nodes = Application of a production

Example



Not All Grammars are Equal

Each language has infinitely many grammars

Some grammars are ambiguous

- A single string may have multiple derivations
- Unambiguous grammars facilitate parsing

Grammar should reflect the internal structure of the PL

- E.g., associativity and precedence of operators

Example: Revised Grammar

A better version of the grammar of arithmetic expressions:

expr $\rightarrow$ **term** | **expr add_op term**

term $\rightarrow$ **factor** | **term mult_op factor**

factor $\rightarrow$ `id` | `number` | **–** **factor** | `(` **expr** `)`

add_op $\rightarrow$ **+** | **–**

mult_op $\rightarrow$ `*` | `/`

Quiz: Context-free Grammars

Draw the parse tree of `foo * x + bar` with the revised grammar. How many nodes and edges does the tree have?

Please vote via Ilias.

Solution of quiz :