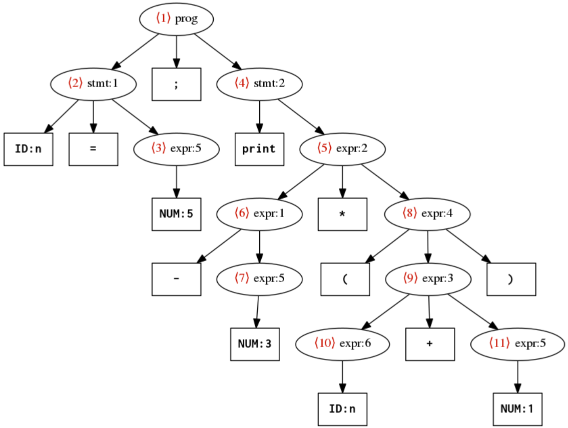
Grammars and CFG



Textbook: 2.1

Grammar example

Def: Reverse Polish Notation. The following are **production rules**. If a statement can be derived using these rules, it is a member of RPN.

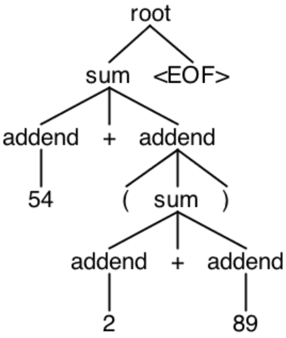
1. A valid RPN string is an `EXPRESSION`
2. An `EXPRESSION` is a number literal or `FORMULA`
3. A `FORMULA` is of the form `a b o`, where `a, b` are `EXPRESSION`s and `o` is an `OPERATOR`
4. An `OPERATOR` is `+`, `-`, `*`, or `/`

- So `1` and `2` are valid `EXPRESSION`s, since they are number literals
- `+` is a valid `OPERATOR`
- Therefore `1 2 +` is a valid `EXPRESSION`
- Then we could derive `3 1 2 + *` or `4 3 1 2 + * /` or `4 3 1 2 + * / 1 5 + -` or infinitely many others

A set of production rules is called a grammar.

Using grammars

- Grammars are commonly seen in parsing
- `bison` is a common tool
- Builds a "parse tree" for a given language that is then transformed and compiled



Formal definition of a context-free grammar

Def: Context-free grammar.

A **context-free grammar** is a 4-tuple (V, Σ, R, S) where:

1. V is a finite set called the **variables** (these will be replaced by rules before we are done)
2. Σ is a finite set $V \cup \Sigma = \emptyset$ called the **terminals** (these cannot be replaced by rules in a CFG)
3. R is a finite set of **rules**, each rule being **of the form** $A \to \alpha$ for some $A \in V, \alpha \in (V \cup \Sigma)^*$
4. $S \in V$ is the start variable

Important: The difference between CFG and other grammars is in the restriction of rules to the form $A \to \alpha$. Other grammar classes don't have this restriction and are more powerful! This distinction is not made clear in the textbook.

Grammar languages

Def: Derivation. If $u, v, w \in (V \cup \Sigma)^*$ and $(A \to w) \in R$, we say that uAv **yields** uwv, written $uAv \Rightarrow uwv$. Say that u **derives** v, written $u \stackrel{*}{\Rightarrow} v$ iff:

1. $u = v$ **or**
2. A sequence $u_1, u_2, \cdots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

Def: Language of a grammar. The language of a grammar G is the set of all strings $\in \Sigma^*$ which can be derived by G.

$$\mathcal{L}(G) = \{w \in \Sigma^* : S \stackrel{*}{\Rightarrow} w\}$$

EBNF (via UCI)

- Just as regular expressions describe regular languages, Extended Backus-Naur Form (EBNF) describes context-free languages
- Also like regex, there is a formal and informal definition
- We will look at the informal definition as provided by GNU `bison`

```
lhs: some_name;
some_name: first_option | second_option | third_option ;
recursive: "hi there" | "hi there" recursive ;
```

EBNF RPN

- Using `bison` EBNF, our RPN grammar becomes:

```
// Entry rule
EXPRESSION: NUMBER_LITERAL | FORMULA ;

FORMULA: EXPRESSION EXPRESSION OPERATOR ;
OPERATOR: '+' | '-' | '*' | '/' ;
NUMBER_LITERAL: DIGITS | DIGITS '.' DIGITS ;
DIGIT: '0' | '1' | '2' | '3' | '4' |
       '5' | '6' | '7' | '8' | '9' ;
DIGITS: DIGIT | DIGIT DIGITS ;
```

Ambiguity

Def: Leftmost derivation. A derivation of a string via a grammar is **leftmost** if, on each step, the leftmost remaining variable is the one replaced.

Def: Ambiguity. If a string can be derived in two separate ways, we say that it is **derived ambiguously**. A string w is derived ambiguously on grammar G if it has two or more different leftmost derivations. Any grammar with one or more ambiguous strings is said to be ambiguous.

Chomsky normal form

Def: Chomsky normal form (CNF). A CFG is in Chomsky normal form if every rule is of the form:

$$A \to BC$$
$$A \to a$$

(also written $A \to BC|a$ sometimes)

where a is any terminal and A, B, C are variables where B and C are **not** the start variable. We also allow the rule $S \to \epsilon$, where S is the start variable.

Converting to CNF

Thm: All CFGs can be written in CNF.

Pf. By construction. Let us operate on some CFG. We will construct an equivalent CFG in Chomsky normal form. We will follow the following steps:

1. Create a new starting rule which is not on any RHSs
2. Eliminate all ϵ rules of the form $A \to \epsilon$
3. Eliminate all unit rules of the form $A \to B$
4. Convert all remaining rules

The following executes these states for an arbitrary CFG.

CNF pf cont.

1. Add the new starting state S' and the rule $S' \to S$ (a unit rule we will address later). This ensures S is on no RHSs.
2. For any rule of the form $A \to \epsilon$, we delete the rule and erase any RHS occurrence of A (leaving the RHS otherwise untouched). If we have $R \to A$, replace it with $R \to \epsilon$ and repeat step 1.
3. For any rule of the form $A \to B$, we delete the rule and replace any rule of the form $B \to u$ to $A \to u$. Repeat until no unit rules remain.

CNF pf cont.

4. For all remaining rules, all of which will be of form $A \to u_1 u_2 \ldots u_k$ (where each u_i is a variable or terminal), do one of the following:
   - If $k = 1$, do nothing
   - If $k = 2$, replace any terminal u_i in that rule with a new variable U_i and add the rule $U_i \to u_i$
   - If $k \geq 3$, replace it with the rules

$$\begin{aligned} A &\to u_1 A_1 \\ A_1 &\to u_2 A_2 \\ A_2 &\to u_3 A_3 \\ &\vdots \\ A_{k-2} &\to u_{k-1} u_k \end{aligned}$$

After this procedure halts, the new grammar will be in CNF.

Next up: PDA, non-context-free languages, the Chomsky hierarchy