CS 241, Lecture 10 - Context Free Grammars, Parse Trees, and Parsing

Thurs, Feb 07, 2019

1 CFGs

- Consider the arithmetic operations over $\Sigma = \{a, b, c, +, -, *, /, (,)\}$. Find a CFG for the following:
  - L_1: Arithmetic expressions from Σ without parentheses
  - L_2: Well formed arithmetic expressions from Σ with balanced parentheses

  Also, find a derivation for $a - b$ in the first language and for $(a - b)$ in the second one.

- Solutions:

  For L_1: Arithmetic expressions from Σ without parentheses

  $$\begin{aligned} S &\rightarrow a \,|\, b \,|\, c \,|\, SRS \\ R &\rightarrow + | - | * | / \end{aligned}$$

  For L_2: Arithmetic expressions from Σ with balanced parentheses

  $$\begin{aligned} S &\rightarrow a \,|\, b \,|\, c \,|\, (SRS) \\ R &\rightarrow + | - | * | / \end{aligned}$$

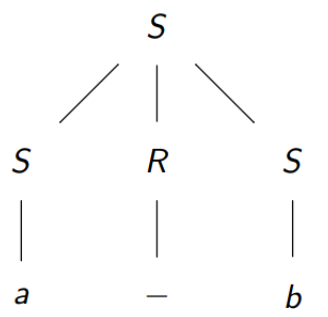
  Derivations:

  $$S \Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - b$$
  $$S \Rightarrow (SRS) \Rightarrow (SRb) \Rightarrow (S - b) \Rightarrow (a - b)$$

- Using the above language, let us create a **parse tree** for the input of $a - b$:

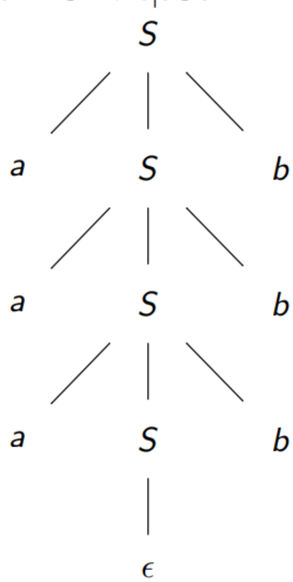
$a - b$ in

$$\begin{aligned} S &\rightarrow a \,|\, b \,|\, c \,|\, SRS \\ R &\rightarrow +|-|*|/ \end{aligned}$$



- Another example: using $aaabbb$ in $S \rightarrow \epsilon|aSb$:

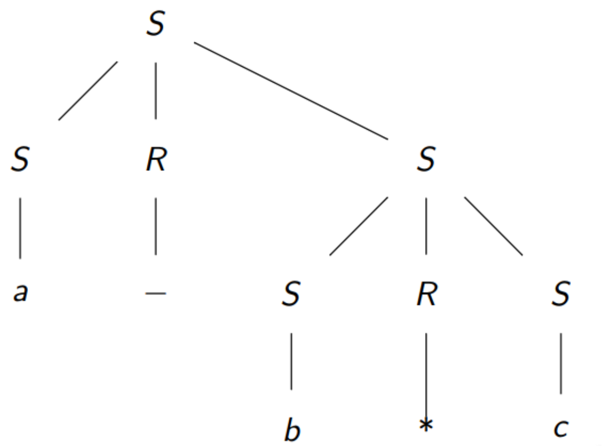
$aaabbb$ in $S \rightarrow \epsilon|aSb$



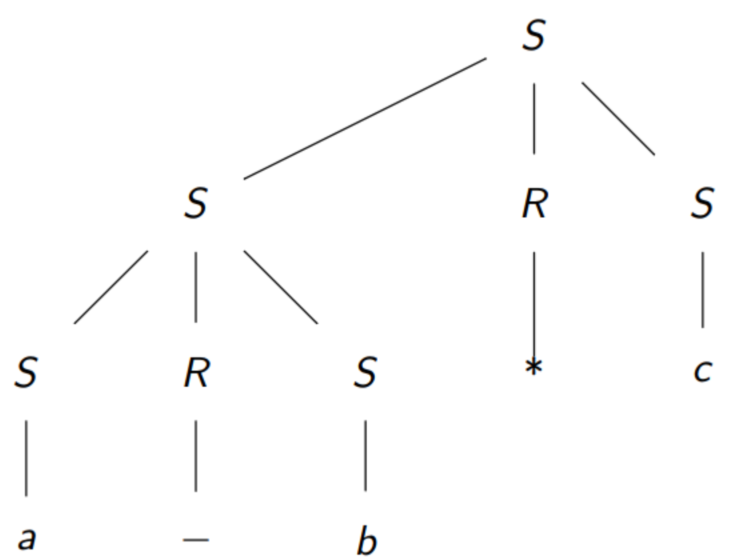
- We note that for every left/right-most derivation, there exists a unique parse tree, and vice versa.

- We also note that given a grammar, every left/right derivation for a string is NOT unique. For example, consider two left-most $a - b * c$:

$$
\begin{aligned}
S &\Rightarrow SRS \Rightarrow aRS \Rightarrow a - S \Rightarrow a - SRS \\
&\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c
\end{aligned}
$$



$$
\begin{aligned}
S &\Rightarrow SRS \Rightarrow SRSRS \Rightarrow aRSRS \Rightarrow a - SRS \\
&\Rightarrow a - bRS \Rightarrow a - b * S \Rightarrow a - b * c
\end{aligned}
$$



- We define a grammar for which some word has more than one distinct leftmost/rightmost derivation/parse tree is called an **ambiguous grammar**. Our example above is an ambiguous example.

- So how can we remove ambiguity? Well, we covered this in rustcc - create *precedence* to force your parse tree to understand, say, $a - b * c$ as $a - (b * c)$, NOT $(a - b) * c$!

- This make it unambiguous. This is what L_2 does in our earlier example.
- What we can also do is insist on what associativity we are using (ie: force right associatve grammar for $a - b * c$.
- If L is a context-free language, is there always an unambiguous grammar s.t. $L(G) = L$? No!
- Can we write a computer program to recognize whether a grammar is ambiguous or not? No!
- This means that given two CFGs G_1 and G_2, we cannot determine if $L(G_1) == L(G_2)$ or even something easier like $L(G_1)\hat{L}(G_2) = \varnothing$. They are both undecideable problems.
- What we *can* do is use pushdown automation, which are just machines that are basically DFAs with an additional stack that we can process in LIFO order.
- But we also need to find the **derivation** - finding this derivation is called **parsing**.

2 Parsing

- Top-down Parsing:
  - Start with S and store intermediate derivations in a stack, and match characters to w.
  - Then every time we pop from the stack, we will have that consumed input + reverse of stack is equal to a intermediate step in our derivation - that is, a step is an α_i where $S \Rightarrow \cdots \Rightarrow \alpha_i \Rightarrow \cdots \Rightarrow w$.
  - We will augment our grammar to include $\vdash$ and $\dashv$ symbolizing the beginning and end of the file. We also include a new start state, S', to begin our parsing.
  - Our original CFG $G = (N, \Sigma, P, S)$ becomes

$$G = (N \cup \{S'\}, \Sigma \cup \{\vdash, \dashv\}, P \cup \{S' \to \vdash S \dashv\}, S')$$

  - The algorithm can be coded as follows:

Algorithm 1 Top-Down Parsing

Push S' onto the stack
while stack is non-empty **do**
    α = pop from stack
    if $\alpha \in N \cup \{S'\}$ **then**
        Push symbols of β of a valid production rule $\alpha \to \beta$ in reverse order on the stack (note derivation)
    else
        c = read_char()
        if $c \neq \alpha$ **then**
            Reject
        end if
    end if
end while
if read_char() = EOF **then**
    Accept
else
    Reject
end if

- Example: Let us determine whether or not $w = abcdef$ is inside $L(G)$ where

$$G = (\{S, A, B\}, \{a, b, c, d, e, f\}, P, S)$$

is defined with P given by:

$$S \to AcB$$
$$A \to ab$$
$$A \to ff$$
$$B \to def$$
$$B \to ef$$

- Solution: We first augment the grammar with $S' \to \vdash S \dashv$, and look for $w = \vdash abcdef \dashv$ in this augmented grammar:

Stack	Read	Processing	Action
S'	ϵ	$\vdash abcdef \dashv$	Pop S', push $\dashv$, S, $\vdash$
$\dashv S \vdash$	ϵ	$\vdash abcdef \dashv$	Match $\vdash$
$\dashv S$	$\vdash$	$abcdef \dashv$	Pop S push B, c then A
$\dashv BcA$	$\vdash$	$abcdef \dashv$	Pop A push b then a
$\dashv Bcba$	$\vdash$	$abcdef \dashv$	match a
$\dashv Bcb$	$\vdash a$	$bcdef \dashv$	match b
$\dashv Bc$	$\vdash ab$	$cdef \dashv$	match c
$\dashv B$	$\vdash abc$	$def \dashv$	Pop B, push f, e then d
$\dashv fed$	$\vdash abc$	$def \dashv$	match d
$\dashv fe$	$\vdash abcd$	$ef \dashv$	match e
$\dashv f$	$\vdash abcde$	$f \dashv$	match f
$\dashv$	$\vdash abcdef$	$\dashv$	match $\dashv$
ϵ	$\vdash abcdef \dashv$	ϵ	Accept (stack = input = ϵ)

3 Correction

- Determining if $L(G) = \emptyset$ is decideable.