- find a right derivation for (*x*+*y*)**z*

$$\begin{aligned} E &\Longrightarrow T \\ &\Longrightarrow T * F \\ &\Longrightarrow T * z \\ &\Longrightarrow F * z \\ &\Longrightarrow (E) * z \\ &\Longrightarrow (E+T) * z \\ &\Longrightarrow (E+F) * z \\ &\Longrightarrow (E+y) * z \\ &\Longrightarrow (T+y) * z \\ &\Longrightarrow (F+y) * z \\ &\Longrightarrow (x+y) * z \end{aligned}$$

$$\begin{aligned} E &\longrightarrow E+T \\ E &\longrightarrow T \\ T &\longrightarrow T * F \\ T &\longrightarrow F \\ F &\longrightarrow (E) \\ F &\longrightarrow x \\ F &\longrightarrow y \\ F &\longrightarrow z \end{aligned}$$

Parsing Context-Free Grammars

- in an *unambiguous* grammar *G*, at each step in a left derivation there's only one production that can be applied that will lead to a correct derivation of *w*
- *G* is an *LL(1) grammar* if that one production can be determined by (only) looking ahead to the next symbol in *w*
  - *LL*(*1*) means *w* is read *L*eft-to-right and a *L*eft derivation is constructed by looking ahead at most *1* character in *w*
  - not all unambiguous context-free grammars are LL(1)
- *G* is an *LR(1) grammar* if it is always possible to tell which rule to apply at each step of the *right* derivation by (only) looking ahead to the next symbol in *w*
  - *LR*(*1*) means *w* is read *L*eft-to-right and a *R*ight derivation is constructed by looking ahead at most *1* character in *w*

LR(1) Parsing

- shift/reduce algorithm
  - two operations –
    - *shift* the current position one place to the right
    - *reduce* by applying a production to the string to the immediate left of the current position
      - if *A* → *xy* is a production, then reduce *C***bxy** | *ijk* ⇒ *Cb***A** | *ijk*
  - maintain a current position in *w*, starting at the left end: |*w*
  - reduce if possible, shift otherwise
  - resolve ambiguities in reductions by looking ahead one character
    - if this is always possible, the grammar is an LR(1) grammar

---

- parse (*x*+*y*)**z* using the LR(1) parsing algorithm

$$\begin{aligned} E &\longrightarrow E+T \\ E &\longrightarrow T \\ T &\longrightarrow T * F \\ T &\longrightarrow F \\ F &\longrightarrow (E) \\ F &\longrightarrow x \\ F &\longrightarrow y \\ F &\longrightarrow z \end{aligned}$$

6. Show the full sequence of shift and reduce operations that are used in the LR(1) parsing of the string $x + (y) * z$ according to the grammar G_3, and give the corresponding right derivation of the string.

$$\begin{aligned} E &\longrightarrow E+T \\ E &\longrightarrow T \\ T &\longrightarrow T * F \\ T &\longrightarrow F \\ F &\longrightarrow (E) \\ F &\longrightarrow x \\ F &\longrightarrow y \\ F &\longrightarrow z \end{aligned}$$

LL(1) vs LR(1)

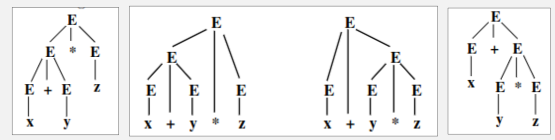
- LR(1) is strictly more powerful than LL(1)
  - there are LL(1) grammars that are not LR(1), but every LL(1) grammar is guaranteed to be LR(1)
- LL(1) and LR(1) are just the tip of the iceberg
  - LL(0), LR(0)
  - SLR(1)
  - LALR(1)
  - LL(k),LR(k)
  - differ in power but also memory requirements and complexity of the algorithm
  - LL(1) and LALR(1) are most common in real compilers
    - even though LR(1) is more powerful, it has high memory requirements and in general, LL(1) and LALR(1) grammars can be constructed for real programming languages
    - real parsers are typically produced by parser generators

Parse Trees

- a *parse tree* is another way of showing a derivation

$$\begin{aligned} E &\Longrightarrow E * E \\ &\Longrightarrow E + E * E \\ &\Longrightarrow x + E * E \\ &\Longrightarrow x + y * E \\ &\Longrightarrow x + y * z \end{aligned}$$

$$\begin{aligned} E &\Longrightarrow E + E \\ &\Longrightarrow x + E \\ &\Longrightarrow x + E * E \\ &\Longrightarrow x + y * E \\ &\Longrightarrow x + y * z \end{aligned}$$



Parse Trees

Theorem 4.5. *Let G be a context-free grammar. There is a one-to-one correspondence between parse trees and left derivations based on the grammar G.* (also right derivations)

Based on this theorem, we can say that a context-free grammar G is ambiguous if and only if there is a string $w \in L(G)$ which has two parse trees.

3. Draw a parse tree for the string $(x+y)*z*x$ according to each of the grammars G_1, G_2, and G_3, as given in this section.

4. Draw three different parse trees for the string $ababbaab$ based on the grammar given in part a) of exercise 1.

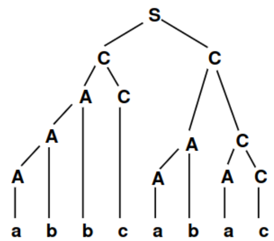
$$\begin{aligned} E &\longrightarrow E + E \\ E &\longrightarrow E * E \\ E &\longrightarrow (E) \\ E &\longrightarrow x \\ E &\longrightarrow y \\ E &\longrightarrow z \end{aligned}$$

$$\begin{aligned} E &\longrightarrow TA \\ A &\longrightarrow +TA \\ A &\longrightarrow \varepsilon \\ T &\longrightarrow FB \\ B &\longrightarrow *FB \\ B &\longrightarrow \varepsilon \\ F &\longrightarrow (E) \\ F &\longrightarrow x \\ F &\longrightarrow y \\ F &\longrightarrow z \end{aligned}$$

$$\begin{aligned} E &\longrightarrow E + T \\ E &\longrightarrow T \\ T &\longrightarrow T * F \\ T &\longrightarrow F \\ F &\longrightarrow (E) \\ F &\longrightarrow x \\ F &\longrightarrow y \\ F &\longrightarrow z \end{aligned}$$

a)
$$\begin{aligned} S &\longrightarrow SS \\ S &\longrightarrow aSb \\ S &\longrightarrow bSa \\ S &\longrightarrow \varepsilon \end{aligned}$$

5. Suppose that the string *abbcabac* has the following parse tree, according to some grammar G:



a) List five production rules that must be rules in the grammar G, given that this is a valid parse tree.

b) Give a left derivation for the string *abbcabac* according to the grammar G.

c) Give a right derivation for the string *abbcabac* according to the grammar G.