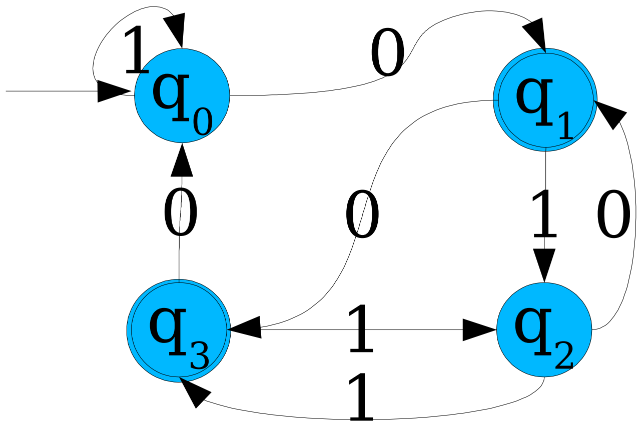
Context-Free Grammar of Regular Languages

DFA **->** **CFG**



S -> 0A | 1S
A -> 0C | 1B | E
B -> 1C | 0A
C -> 0S | 1B | E

q_0 = S, q_1 = A, q_2 = B, q_3 = C

Rules of Conversion:
(q_0, 0) => q_2 to CFG: $q_2 = q_0 0$
(q_i, a) => q_j to CFG: $q_j = q_i a$ or $q_i = a q_j$
if q_i is an accept state, also include q_i = E

For Example
011010

Trace in DFA:
(q_0, 0) -> (q_1, 1) -> (q_2, 1) -> (q_3, 0) -> (q_0, 1) -> (q_0, 0) -> (q_1, E)

Trace in CFG:
S -> 0A -> 01B -> 011C -> 0110S -> 01101S -> 011010A -> 011010

Ambigious Grammars

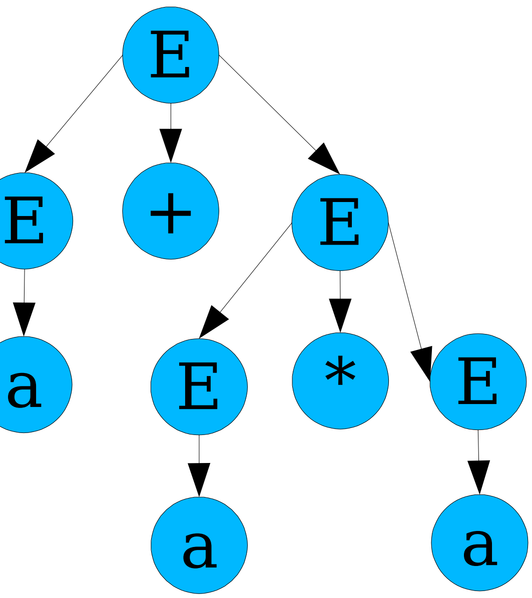
A grammar in which the same string can be created using two different parse trees.

Example
E -> E + E | E * E | E | a

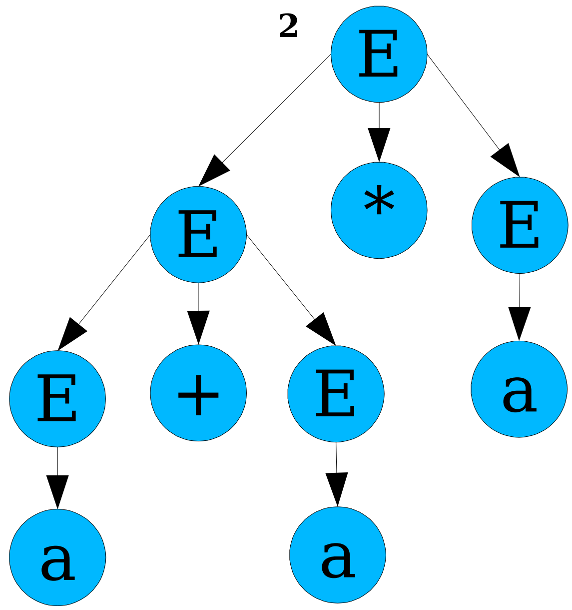
a + a * a

Derivation 1: E -> E + E -> E + E * E -> a + a * a
Derivation 2: E -> E * E -> E + E * E -> a + a * a

1



2



Programming languages **must** be unambiguous. In an ambigious language strings that look the same may have different meanings.

This example can be made unambiguous:
E -> E + T | T
T -> T * F | F
F -> (E) | a

If you restrict all derivations to lefmost derivations, it will show that two different derivations correspond to two different parse trees (or meanings).

Chomsky Normal Form

All CFGs can be expressed in CNF
Restricts the definition without hindering capability

Restricted Rule Forms

A->BC	(B & C are not start variable)
A->a	(a is a terminal)
S->E	(no other variable may go to epsilon)

Conversion

1. S_0 -> S (Prevents S_0 from being on the right-hand-side of a rule)
2. A -> E (where A != S_0) is **not** allowed, and must be eliminated.

R -> uAv | uAvAu
R -> uAv | uAvAu **| uv | uvu | uvAu | uAvu**
(bold portions remove A -> E)

Will add a rule for each time A appears on the RHS of a production

If R -> ... | E, there is a new problem. If R -> E was previously eliminated, do not add it, but if not, do so and repeat the process to eliminate until all productions of the form A -> E are gone (where A != S0)

3. A -> B: If there is a rule B-> u (where u is a string of terminals and variables), then A-> u. Then remove all rules of the form A->u (unless if such a rule was previously removed)
4. A -> U1U2...Uk – convert to:
   $A \to U_1A_1$
   $A_1 \to U_2A_2$
   ...
   $A_{k-2} \to U_{k-1}U_k$

   Example
   A -> aBbB – convert to:
   $A \to U_2A_1$
   $A_1 \to BA_2$
   $A_2 \to U_1B$
   $U_1 \to b$
   $U_2 \to a$