CA320 - Computability & Complexity

Context-Free Languages

Dr. David Sinclair

Context-Free Grammars

A *context-free grammar* (CFG) is a 4-tuple $G = (N, \Sigma, S, P)$ where

- N is a set of *nonterminal symbols*, or *variables*,
- Σ is a set of *terminal symbols*, and N and Σ are disjoint finite sets,
- S is a special nonterminal ($S \in N$) called the *start symbol*,
- P is a finite set of *grammar rules*, or *productions*, of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (N \cup \Sigma)$.

The set $V = N \cup \Sigma$ is called the vocabulary of G.

Context-Free Grammars (2)

A *derivation*, $\alpha \Rightarrow \beta$, is the application of one or more production rules starting with the string α and resulting in the string β.

Consider the following grammar.

$$\begin{array}{lcl} S & \rightarrow & SS \\ S & \rightarrow & (S) \\ S & \rightarrow & \epsilon \end{array}$$

An example derivation is:

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S((S)) \Rightarrow S(()) \Rightarrow (S)(()) \Rightarrow ()(())$$

If $A \rightarrow \gamma$ then $\alpha A \beta \Rightarrow \alpha\gamma\beta$ is a *single step derivation* using $A \rightarrow \gamma$.

The grammar is a *context-free grammar* since the production rule, $A \rightarrow \gamma$, does not depend on the *context* surrounding nonterminal, A.

Context-Free Grammars (3)

Derivations requiring ≥ 0 and ≥ 1 steps are denoted by $\Rightarrow^*$ and $\Rightarrow^+$ respectively.

The *start symbol* denotes the entire set of strings that can be generated by G, $L(G)$.

$$L(G) = \{x \in \Sigma^* | S \Rightarrow^+ x\}$$

Example: *AnBn*

$$\begin{array}{lcl} S & \rightarrow & aSb \\ S & \rightarrow & \epsilon \end{array}$$

This grammar yields derivations such as
$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.

$L(G) = \{a^k b^k | k \geq 0\}$

Context-Free Grammars (4)

Example: *Expr*

$$
\begin{array}{lcl}
S & \rightarrow & E\ O\ E \\
E & \rightarrow & id \\
E & \rightarrow & num \\
E & \rightarrow & (\ E\ ) \\
O & \rightarrow & + \\
O & \rightarrow & - \\
O & \rightarrow & * \\
O & \rightarrow & /
\end{array}
$$

This grammar generates simple expressions over number and identifiers.

Derivations

A *derivation* is a sequence of steps where in each step a non-terminal is replaced by the left-hand side of a productions rule that starts with the non-terminal.

If the leftmost non-terminal is always chosen to be replaced then it is a *leftmost derivation*.

If the rightmost non-terminal is always chosen to be replaced then it is a *rightmost derivation*.

A derivation is also called a *parse*. The process of discovering a derivation is called *parsing*.

Parse Tree

Consider the CFG grammar *Expr*.

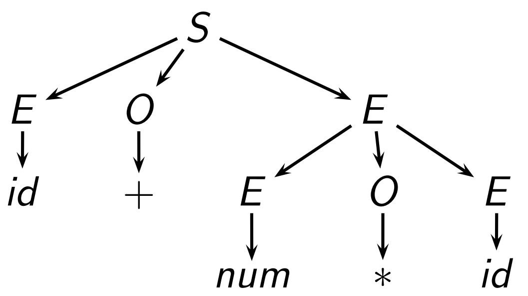
$$
\begin{aligned}
S &\Rightarrow E\ O\ E \\
&\Rightarrow id\ O\ E \\
&\Rightarrow id\ +\ E \\
&\Rightarrow id\ +\ E\ O\ E \\
&\Rightarrow id\ +\ num\ O\ E \\
&\Rightarrow id\ +\ num\ *\ E \\
&\Rightarrow id\ +\ num\ *\ id
\end{aligned}
$$

This can be denoted as $S \Rightarrow^* id + num * id$.

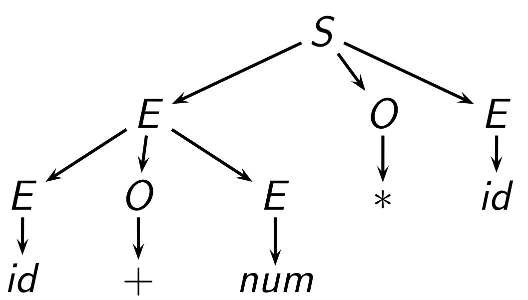
Because we always choose the leftmost non-terminal this is a leftmost derivation.

Parse Tree (2)

This derivation can also be represented as a *parse tree*.

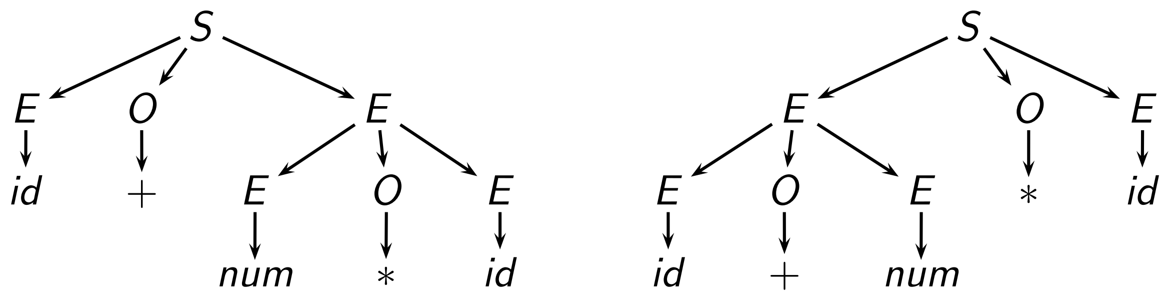


But what if we choose a rightmost derivation of the same expression.



Ambiguity

What is the implication of the fact that there at least 2 parse trees for the same expression?



A CFG G is *ambiguous* if $\exists x \in L(G)$ such that x has more than one derivation tree. This is equivalent to saying it has more than one distinct leftmost or rightmost derivations tree.

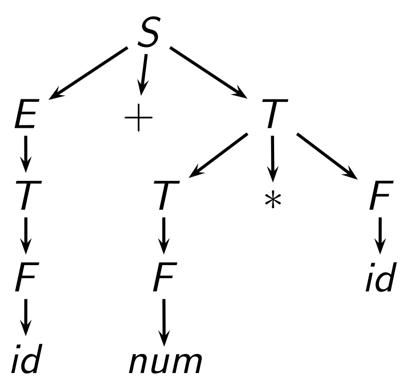
Ambiguity (2)

Sometimes redesigning the grammar can remove the ambiguity. The following grammar, *Expr1*, does not have the ambiguity of *Expr*.

Expr1

$$
\begin{array}{lcl}
S & \rightarrow & E \\
E & \rightarrow & E + T \\
E & \rightarrow & E - T \\
E & \rightarrow & T \\
T & \rightarrow & T * F \\
T & \rightarrow & T / F \\
T & \rightarrow & F \\
F & \rightarrow & id \\
F & \rightarrow & num \\
F & \rightarrow & (E)
\end{array}
$$

The only parse tree for $id + num * id$ is:



Pushdown Automaton

We know that the language *AnBn* is not a regular language and cannot be recognised by a *Finite State Automaton*. The following language $SimplPal = \{wcw^r | w \in \Sigma^*\}$, where w^r is the reverse of w, is also not a regular language. Both of these languages require a machine that can remember something. In the case of *SimpPal* it must remember w. Then after seeing c it then checks for w^r.

We can extend a Nondeterministic Finite State Automaton by adding a *stack memory*. This is called a *Pushdown Automaton*.

Pushdown Automaton (2)

A *Pushdown Automaton* (PDA) is a 7-tuple
$M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ where

- Q is a finite set of *states*;
- Σ and Γ are finite *input* and *stack alphabets*;
- $q_0 \in Q$ is the *initial state*;
- $Z_0 \in \Gamma$ is the *initial stack symbol*;
- $A \subseteq Q$ is the set of *accepting states*;
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{(Q \times \Gamma^*)}$ is the *transition function*.

The PDA is nondeterministic because the transition function δ can map the same element of the range to different elements of the range.

A *configuration* of the PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is a triple (q, x, α) where $q \in Q, x \in \Sigma^*$ and $\alpha \in \Gamma^*$.

Pushdown Automaton (3)

$(p, x, \alpha) \vdash_M (q, y, \beta)$ represents the PDA M moving from configuration (p, x, α) to configuration (q, y, β). This can occur in two ways:

- an input symbol is read; or
- a Λ-transition occurs.

i.e. $x = \sigma y, \sigma \in \Sigma \cup \{\epsilon\}$.

If $\alpha = X\gamma, X \in \Gamma, y \in \Gamma^*$ then $\beta = \xi\gamma$ where $(q, \xi) \in \delta(p, \sigma, X)$.

$(p, x, \alpha) \vdash_M^n (q, y, \beta)$ denotes moving from configuration (p, x, α) to configuration (q, y, β) in n steps.

$(p, x, \alpha) \vdash_M^* (q, y, \beta)$ denotes moving from configuration (p, x, α) to configuration (q, y, β) in zero or more steps.

Pushdown Automaton (4)

If $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ and $x \in \Sigma^*$, then x is *accepted by M* if $\exists \alpha \in \Gamma^*, q \in A$ such that

$$(q_0, x, Z_0) \vdash_M^* (q, \epsilon, \alpha)$$

A language $L \subseteq \Sigma^*$ is said to be accepted by M if L is precisely the set of strings accepted by M.

Theorem

A language is context-free iff it can be recognized by a pushdown automaton.

Pushdown Automaton (5)

Example PDAs

AnBn
$Q = \{q_0, q_1, q_2, q_3\}$
$A = \{q_3\}$
The transition table for *AnBn* is:

State	Input	Top of Stack	Move(s)
q_0	ϵ	Z_0	(q_3, Z_0)
q_0	a	Z_0	(q_1, aZ_0)
q_1	a	a	(q_1, aa)
q_1	b	a	(q_2, ϵ)
q_2	b	a	(q_2, ϵ)
q_2	ϵ	Z_0	(q_3, Z_0)

The sequence of moves that accepts *aabb* is:
$(q_0, aabb, Z_0) \vdash (q_1, abb, aZ_0) \vdash (q_1, bb, aaZ_0) \vdash (q_2, b, aZ_0) \vdash (q_2, \epsilon, Z_0) \vdash (q_3, \epsilon, Z_0)$

Pushdown Automaton (6)

SimplPal
$Q = \{q_0, q_1, q_2\}$ $A = \{q_2\}$

The transition table for *SimplPal* is:

State	Input	Top of Stack	Move(s)
q_0	a	Z_0	(q_0, aZ_0)
q_0	b	Z_0	(q_0, bZ_0)
q_0	a	a	(q_0, aa)
q_0	b	a	(q_0, ba)
q_0	a	b	(q_0, ab)
q_0	b	b	(q_0, bb)
q_0	c	Z_0	(q_1, Z_0)
q_0	c	a	(q_1, a)
q_0	c	b	(q_1, b)
q_1	a	a	(q_1, ϵ)
q_1	b	b	(q_1, ϵ)
q_1	ϵ	Z_0	(q_2, Z_0)

$(q_0, abcba, Z_0) \vdash$
$(q_0, bcba, aZ_0) \vdash$
$(q_0, cba, baZ_0) \vdash$
$(q_1, ba, baZ_0) \vdash$
$(q_1, a, aZ_0) \vdash$
$(q_1, \epsilon, Z_0) \vdash$
(q_2, ϵ, Z_0)

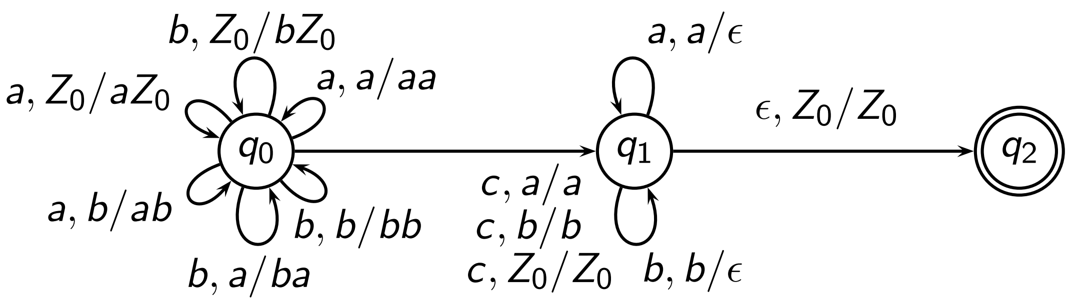
Pushdown Automaton (7)

We can rewrite the same PDA as:

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b, c\} \\
\Gamma &= \{a, b\} \\
A &= \{q_2\} \\
\delta &= ((q_0, a, Z_0), (q_0, aZ_0)), ((q_0, b, Z_0), (q_0, bZ_0)), \\
&\quad ((q_0, a, a), (q_0, aa)), ((q_0, b, a), (q_0, ba)), \\
&\quad ((q_0, a, b), (q_0, ab)), ((q_0, b, b), (q_0, bb)) \\
&\quad ((q_0, c, Z_0), (q_1, Z_0)), ((q_0, c, a), (q_1, a)) \\
&\quad ((q_0, c, b), (q_1, b)), ((q_1, a, a), (q_1, \epsilon)) \\
&\quad ((q_1, b, b), (q_1, \epsilon)), ((q_1, \epsilon, Z_0), (q_2, Z_0))
\end{aligned}
$$

Pushdown Automaton (8)

Or draw it as a transition diagram;



Pushdown Automaton (9)

Consider the language $EvenPal = \{ww^r | w \in \{a, b\}^*\}$

The PDA that accepts $EvenPal$ is:

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b\} \\
\Gamma &= \{a, b\} \\
A &= \{q_2\} \\
\delta &= ((q_0, a, Z_0), (q_0, aZ_0)), ((q_0, a, a), (q_0, aa)), \\
&\quad ((q_0, a, b), (q_0, ab)), ((q_0, b, Z_0), (q_0, bZ_0)), \\
&\quad ((q_0, b, a), (q_0, ba)), ((q_0, b, b), (q_0, bb)), \\
&\quad ((q_0, \epsilon, \epsilon), (q_1, \epsilon)), \\
&\quad ((q_1, a, a), (q_1, \epsilon)), ((q_1, b, b), (q_1, \epsilon)), \\
&\quad ((q_1, \epsilon, Z_0), (q_2, \epsilon))
\end{aligned}
$$

Deterministic PDA

A PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is deterministic if we can always decide which transition will be used next, i.e.

- $\forall q \in Q, \sigma \in \Sigma \cup \{\epsilon\}, X \in \Gamma$, the set $\delta(q, \sigma, X)$ has at most one element; **and**
- $\forall q \in Q, \sigma \in \Sigma, X \in \Gamma$, the sets $\delta(q, \sigma, X)$ and $\delta(q, \epsilon, X)$ cannot both be nonempty.

The *SimplPal* language is deterministic. The *EvenPal* language is nondeterministic. **Why?**

A language L is deterministic context-free if there is some deterministic PDA that recognizes L.

Theorem

Not every non-deterministic PDA can be converted to an equivalent deterministic PDA.

This has serious implications for the efficient parsing of context-free languages.

Pumping Lemma for Context-Free Languages

Theorem

If L is a context-free language there is an integer n such that $\forall u \in L$ with $|u| \geq n$, $u = vwxyz$ such that

1. $|wy| > 0$
2. $|wxy| \leq n$
3. $\forall m \geq 0, vw^m xy^m z \in L$

Example: The language $AnBnCn = \{a^n b^n c^n | n \geq 0\}$ is not a context-free language.

Pumping Lemma for Context-Free Languages (2)

Proof.

Let's assume $AnBnCn$ is context-free. Then

$$a^n b^n c^n = vw^m xy^m z, \forall m \geq 0.$$

Conditions 1 and 2 imply that wxy has at least one symbol and no more that 2 distinct symbols.
Let σ_1 be one of the symbols occurring in wy and σ_2 be the symbol that does not occur in wy.
Then the string $vw^0 xy^0 z$, which deletes w and y from u, will have less than n occurrences of σ_1 but exactly n occurrences of σ_2.
But $u = vw^0 xy^0 z \in AnBnCn$ must have an equal number each symbol but we have shown that number of occurrences of σ_2 is greater than the number of occurrences of σ_1. This contradiction invalidates the assumption that $AnBnCn$ is context-free. $\square$