

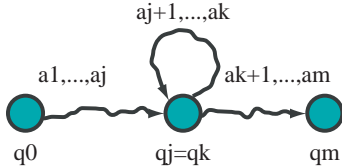
CS20a: summary (Oct 24, 2002)

- Context-free languages
 - Grammars $G = (V, T, P, S)$
- Pushdown automata
 - N-PDA = CFG
 - D-PDA < CFG
- Today
 - What languages are context-free?
 - Pumping lemma (similar to pumping lemma for regular languages)
 - Ogden's lemma



Regular languages

- Intuition: if a FA accepts a string that is "long enough," it must repeat a state
 - But it can't remember that the state was repeated
 - So it can be forced to repeat the state over and over



Pumping lemma for regular languages

Lemma (the Pumping Lemma)

- Let L be a regular set.
- There is a constant n s.t. for any z where $|z| \geq n$, then z can be written $z = uvw$, where
 - $|uv| \leq n$
 - $|v| \geq 1$
 - For all $i \geq 0$, $uv^i w \in L$
 - n is bounded by $|Q|$



Pumping lemma for CFL

Lemma Let L be a CFL. Then there is a constant n such that for any string $z \in L$ where $|z| \geq n$, then $z = uvxyz$ and

- $|vx| \geq 1$
- $|vwx| \leq n$
- $uv^iwx^iy \in L$ for $i \geq 0$

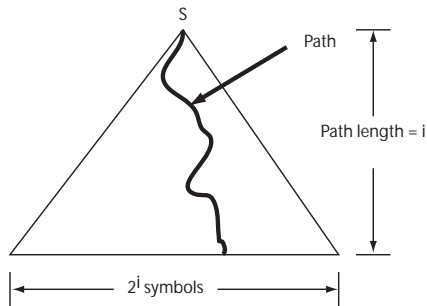


Pumping lemma

- Suppose L is in Chomsky Normal Form
 - $A \rightarrow a$
 - $A \rightarrow BC$



Derivation trees



Pumping lemma is too weak

The language $L = \{a^i b^j c^k d^l \mid i = 0 \vee j = k = l\}$ is not context-free.

- Attempt 1
 - Consider a string $z = b^j c^k d^l \in L$, and write it as $z = uvwxy$
 - Then $vwxy$ might contain only b 's, no use
- Attempt 2
 - Consider a string $z = a^i b^j c^k d^l$
 - Then $vwxy$ might contain only a 's, no use



Ogden's lemma

Ogden's Lemma Let L be a CFL. Then there is a constant n , then for any $z \in L$ we can mark n or more positions of $z = uvwxy$, such that:

- v and x have at least one marked symbol
- $vwxy$ has at most n marked symbols
- $uv^i wx^i y \in L$ for $i \geq 0$



Building a path

Proof Let G be a grammar for $L - \{\epsilon\}$ in Chomsky normal form, with k productions. Let $n = 2^k + 1$. Construct a path P by the following algorithm:

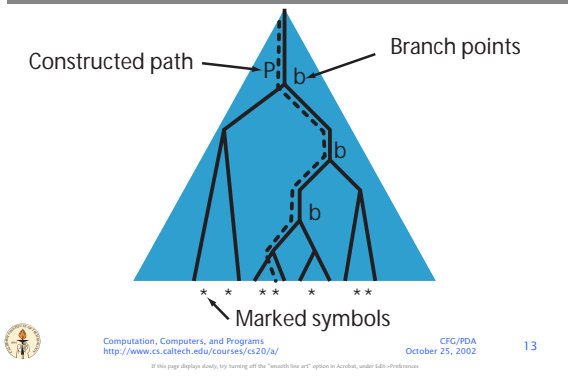
Base Add the root to P

Step Let r be the last vertex on P

- If r is a terminal, stop
- If r has two children, pick the one that has the most marked descendants and add it to r .



Building the path



Branch points

- A point r in P is called a *branch point* if both children have marked descendants.
 - Each branch point in P has at least half as many marked descendants as the previous branch point.
 - Since there are at least n marked symbols in z , there are at least $k + 1$ branch points in P .
 - Two of the branch points must have the same label.
- Computation, Computers, and Programs
<http://www.cs.caltech.edu/courses/cs20/a/>
 CFG/PDA October 25, 2002 14

An example (part 1)

The language $L = \{a^i b^j k^l \mid i, j, k \text{ are different}\}$ is not context-free.

- Let n be the constant on Ogden's lemma
- Consider $z = a^n b^{n+n!} c^{n+2n!}$
- Mark all the a 's, and let $z = uvwxy$
- If either v or x contains two different symbols, then pumping will destroy the symbol order
- Otherwise, at least one of v or x contains a 's (since a 's are marked)

Computation, Computers, and Programs
<http://www.cs.caltech.edu/courses/cs20/a/>
 CFG/PDA October 25, 2002 15

An example (part 2)

- Consider the case where $v \in a^+$ and $x \in b^*$ (other cases are similar)
- Let $p = |v|$, so p divides $n!$; let $pq = n!$
- Then $z' = wv^{2q+1}wx^{2q+1}y$ is in L
- But $a^{2q+1} = a^{2pq+p} = a^{2n!+p}$
- So, z' has $(n + 2n!)$ a 's; it also has $(n + 2n!)$ c 's, a contradiction.



Closure properties

- CFLs are closed under union.
 - $S ::= S_1 \mid S_2$
- CFLs are closed under concatenation
 - $S ::= S_1S_2$
- CFLs are closed under Kleene closure
 - $S ::= \epsilon \mid SS_1$



Intersection

Theorem CFLs are not closed under intersection.

- $L_1 = \{a^i b^j c^j \mid i, j \geq 0\}$
 - $S ::= AB$
 - $A ::= aAb \mid ab$
 - $B ::= c \mid Bc$
- $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$
 - (Similar)

Then $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 0\}$, which is not context-free



Complement

Theorem CFLs are not closed under complementation

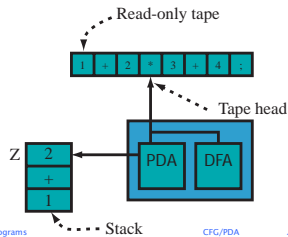
Proof Note that $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$



Intersection with regular languages

Theorem If L is a CFL, and R is regular, then $L \cap R$ is context-free

Proof Build a new machine that simulates both automata.



Intersection with regular languages

- Let $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$
- Let $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$
- Build $M' = (Q_A \times Q_M, \Sigma, \Gamma, \delta, (p_0, q_0), Z_0, F_A \times F_M)$
- where
 - $\delta((p, q), c, X) = \bigcup_{(q, \gamma) \in \delta_M(q, c, X)} ((\delta_A(p), q), \gamma)$
 - $\delta((p, q), \epsilon, X) = \bigcup_{(q, \gamma) \in \delta_M(q, \epsilon, X)} ((p, q), \gamma)$



Determining if a string is in a language

The Cocke-Younger-Kasumi algorithm (CYK)

- Given a string x , with $|x| \geq 1$, a grammar G in Chomsky normal form, determine if $x \in L(G)$
- Intuition: for each nonterminal A , determine if $A \rightarrow_G^* x_{ij}$ where x_{ij} is the substring of x starting at position i of length j

Base If $j = 1$, then $A \rightarrow_G^* x_{ij}$ iff $A ::= x_{ij}$

Step If $j > 1$, then consider each production $A ::= BC$, and test if $B \rightarrow_G^* x_{ik}$ and $C \rightarrow_G^* x_{i+k, j-k}$ for any k



CYK algorithm

(* Let V_{ij} be the nonterminals that derive x_{ij} *)

let $\text{cyk } x =$

for $i = 0$ to $n - 1$ do

$V_{i1} = \{A \mid A ::= x[i]\}$

done

for $j = 1$ to $n - 1$ do

for $i = 0$ to $n - j$ do

$V_{ij} \leftarrow \{\}$;

for $k = 0$ to $j - 2$ do

$V_{ij} \leftarrow V_{ij} \cup$

$\{A \mid A ::= BC \wedge B \in V_{ik} \wedge C \in V_{i+k, j-k}\}$

done

done

done



LR(k) Parsing

- Left-to-right parsing, rightmost derivation, k tokens lookahead
- Stack machine
- Example:
 - $S \rightarrow E \$$
 - $E \rightarrow \text{number}$
 - $E \rightarrow E + E$
 - $E \rightarrow E * E$



FIRST function

- Returns the set of terminals that may begin a production

$$\text{FIRST}(X) = \{X\} \text{ if } X \text{ is a terminal}$$

$$\text{FIRST}(Y_1 Y_2 \dots Y_n) = \begin{cases} \text{FIRST}(Y_1) \cup \text{FIRST}(Y_2 \dots Y_n) & \text{if } Y_1 \text{ is nullable} \\ \text{FIRST}(Y_1) & \text{otherwise} \end{cases}$$



Closure function

- Modified to get the next lookahead symbol

```
let closure I =
  repeat for I does not change:
    for each (A → α.Xβ, z) ∈ I do
      for each production X → γ
        for each w ∈ FIRST(βz)
          I ← I ∪ {(X → .γ, w)}
  return I
```



Goto function

- Modified for LR(1) items

```
let Goto I X =
  let J = {}
  for each (A → α.Xβ, z) ∈ I do
    J ← J ∪ {(A → α.X.β, z)}
  return (closure J)
```

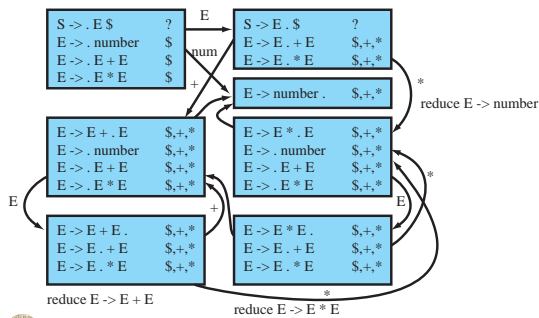


An example

- $E \rightarrow \text{number}$
- $E \rightarrow E + E$
- $E \rightarrow E * E$



LR(1) state diagram



Disambiguating expressions

- $E \rightarrow \text{number} \mid E + E \mid E * E \mid (E)$
- Inherently ambiguous
- Rewrite into a two-level grammar:
 - E : general expression
 - $E \rightarrow T \mid E + T$
 - T : pure multiplication
 - $T \rightarrow S \mid T * S$
 - S : "simple" expression
 - $S \rightarrow \text{number} \mid (E)$



How can we disambiguate ifthenelse?

- $E \rightarrow \text{number}$
- $E \rightarrow E + E \mid E * E \mid (E)$
- $E \rightarrow \text{if } E \text{ then } E$
- $E \rightarrow \text{if } E \text{ then } E \text{ else } E$



Matched expression

- $E \rightarrow M \mid U$
- $M \rightarrow \text{number} \mid M + M \mid M * M \mid (E)$
- $M \rightarrow \text{if } E \text{ then } M \text{ else } M$
- $U \rightarrow \text{if } E \text{ then } U$
- $U \rightarrow \text{if } E \text{ then } M \text{ else } U$