

CS20a: summary (Oct 15, 2002)

- So-far: regular languages
 - *DFA = NFA = e-NFA = Regex*
 - *Minimization, equivalence is decidable*
 - *Many languages are not regular*
 - *Balanced parentheses*
 - *Arithmetic expressions*
- Next: context-free languages
 - *(PDA = NPDA = CFG)*
 - *Add LIFO (stack) memory*
 - *Expressive enough for*
 - *Balanced parentheses*
 - *Arithmetic expressions*

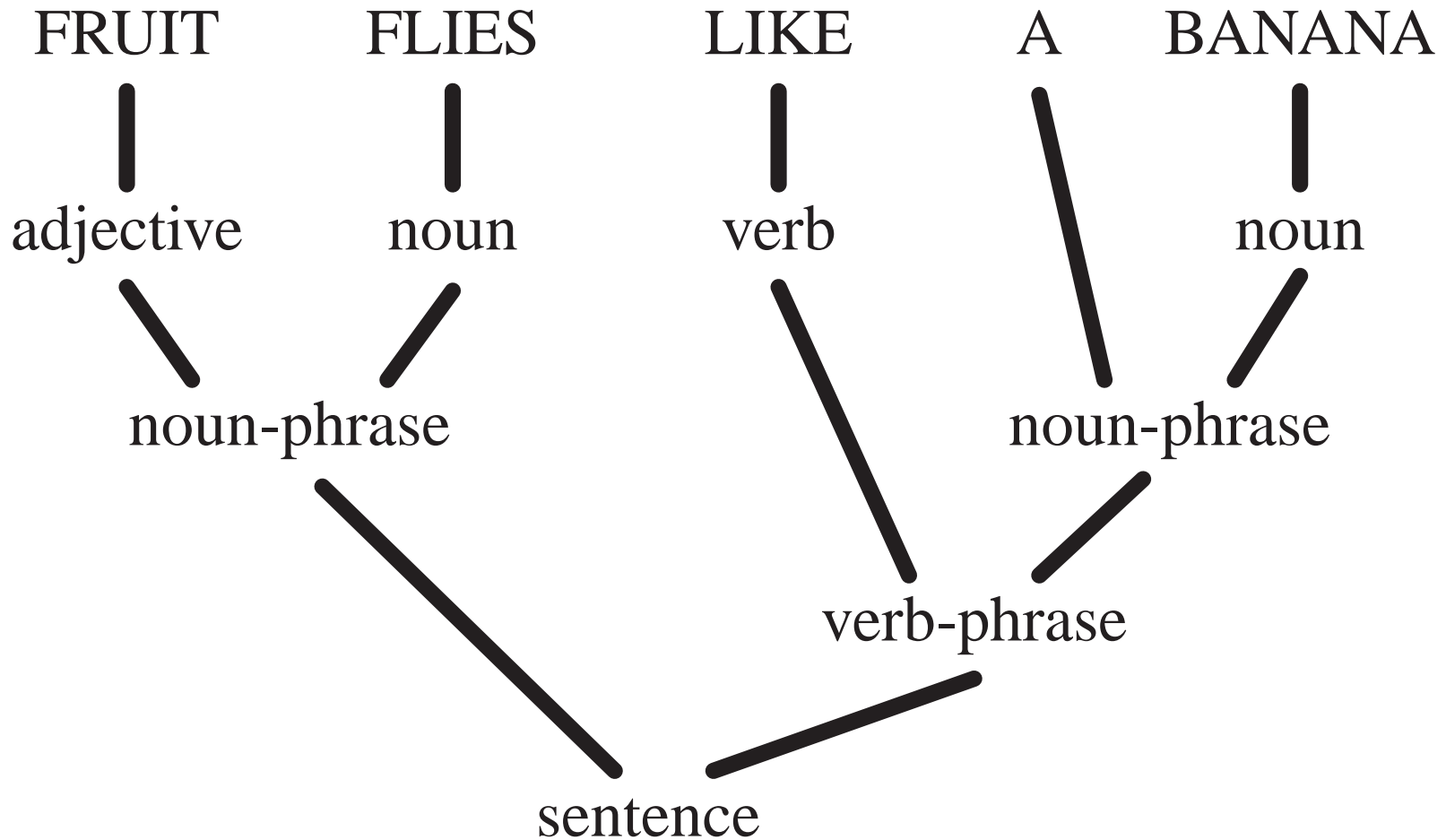


Context-free languages

- Originally defined to describe natural languages
 - *sentence ::= noun-phrase verb-phrase*
 - *noun-phrase ::= adjective noun-phrase | noun | A noun*
 - *verb-phrase ::= verb | noun-phrase*
 - *noun ::= FRUIT | BANANA | SQUASH | FLIES*
 - *adjective ::= SOUR | SWEET | FRUIT*
 - *verb ::= RUN | JUMP | LOVE | LIKE | SQUASH*



Sentence diagramming; derivation trees



Context free grammars

- A context free grammar is:
 - A finite set of variables (called nonterminals)
 - noun-phrase, noun, verb, preposition, ...
 - A finite set of terminals (we often use uppercase)
 - BANANA, FLIES, LIKE, FRUIT, ...
 - A finite set of productions
 - $\text{noun-phrase} ::= \text{adjective noun-phrase} \mid \text{noun} \mid A \text{ noun}$



Programming languages and parsing

- Arithmetic

- $e ::= e + e \mid e - e \mid e * e \mid e / e$
 - | $- e$
 - | (e)
 - | $NUMBER$

- Notation:

- We often use $::=$ (programming language convention)
 - Also, \rightarrow is often used
 - $E ::= e + e \mid e - e \mid \dots$ is actually notation for several productions
 - $e ::= e + e$
 - $e ::= e - e$
 - ...
 - $e ::= - e$
 - $e ::= NUMBER$



CFG: formal definition

- A CFG is a four-tuple (V, T, P, S)
 - V is a finite set of nonterminals
 - T is a finite set of terminals (V and T are disjoint)
 - P is a finite set of productions
 - S is a nonterminal called the *start symbol*



Arithmetic

- Consider the grammar

$$e ::= e + e \mid e * e \mid (e) \mid \textit{NUMBER}$$

- $V = \{e\}$
- $T = \{\textit{NUMBER}\}$
- $P =$ the four productions
- $S = e$



Derivations: definitions

- Let $G = (V, T, P, S)$
- Define \rightarrow_G to be the application of a production:
 - If $A \rightarrow \beta$ is a production
 - α and γ are strings in $(V \cup T)^*$
 - Then $\alpha A \gamma \rightarrow_G \alpha \beta \gamma$
- \rightarrow_G^* is the transitive closure:
 - $\alpha \rightarrow_G^* \alpha$
 - If $\alpha \rightarrow_G \beta$ then $\alpha \rightarrow_G^* \beta$
 - If $\alpha \rightarrow_G \beta$ and $\beta \rightarrow_G \gamma$, then $\alpha \rightarrow_G^* \gamma$



Definitions

- The *language generated by* G is

$$L(G) = \{w \in T^* \mid S \rightarrow_G^* w\}$$

- A language L is *context-free* if $L = L(G)$ for some CFG G
- A string $\alpha \in (T \cup V)^*$ is in *sentential form* (or, it is a *sentence*) if $S \rightarrow_G^* \alpha$
- Two grammars G_1 and G_2 are *equal* if $L(G_1) = L(G_2)$



Balanced parens

- Let G be the grammar $S ::= () \mid (S) \mid SS$
- Then $L(G)$ is the language containing all non-empty strings of balanced parentheses



Balanced parens

- Let G be the grammar $S ::= () \mid (S) \mid SS$
- Then $L(G)$ is the language containing all non-empty strings of balanced parentheses
- Proof (by structural induction on G)
- Induction hypothesis: $S \rightarrow_G^* w$ iff:
 - Each prefix of w has at least as many (as)
 - w has an equal number of (and)



Base case

Base For $S ::= ()$, then $w = ()$

- Each prefix of $()$ has at least as many $($ as $)$
- $()$ has an equal number of $($ and $)$



Induction step

- For $S ::= (S)$**
- Each prefix of S has at least as many (as), so each prefix of (S) has at least as many (as)
 - S has an equal number of parens, so (S) has an equal number of parens

- For $S ::= S_1S_2$**
- Each prefix of S_1 and S_2 has at least as many (as), so each prefix of S_1S_2 has at least as many (as)
 - S_1 and S_2 have an equal number of parens; so does S_1S_2



Another example

$S ::= aB$
| bA

$A ::= a$
| aS
| bAA

$B ::= b$
| bS
| aBB



Another example

$S ::= aB$	$A ::= a$	$B ::= b$
$ bA$	$ aS$	$ bS$
	$ bAA$	$ aBB$

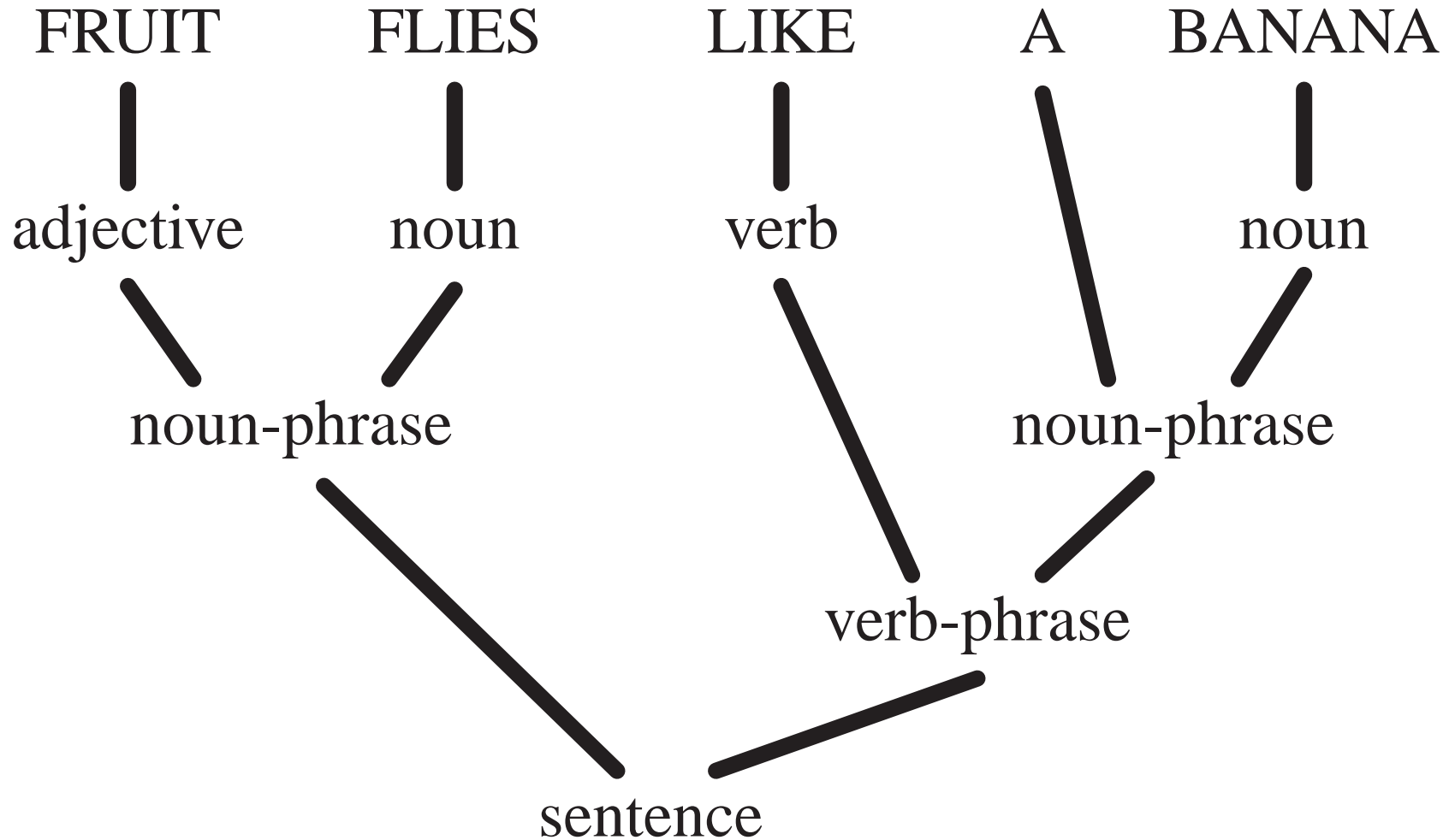
Theorem $L(G)$ the non-empty strings containing an equal number of a 's and b 's

Hypothesis

- $S \rightarrow_G^* w$ iff w has an equal number of a 's and b 's
- $A \rightarrow_G^* w$ iff w has one more a than b 's
- $B \rightarrow_G^* w$ iff w has one more b than a 's



Derivation trees



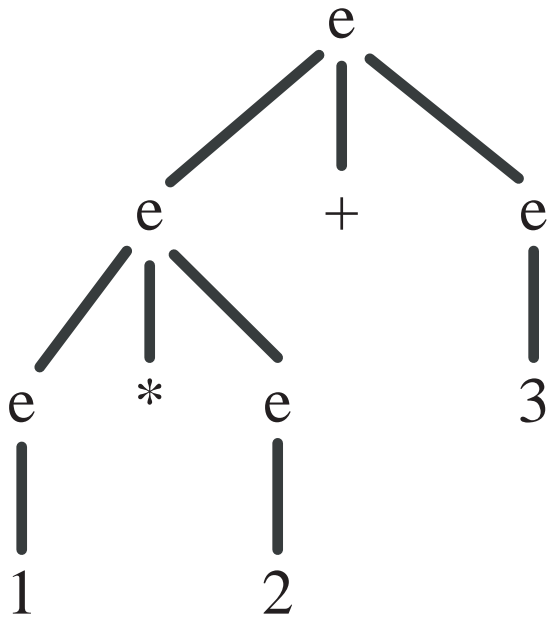
Derivation trees: formal definition

- Every vertex has a label in $T \cup V \cup \{\epsilon\}$
- The root has label S
- Each interior (non-leaf) node has a label in V
- If n has label A with children labeled X_1, \dots, X_k , then $A ::= X_1 \cdot \dots \cdot X_k$ is a production
- If n has label ϵ , then n is a leaf and is the only child of its parent



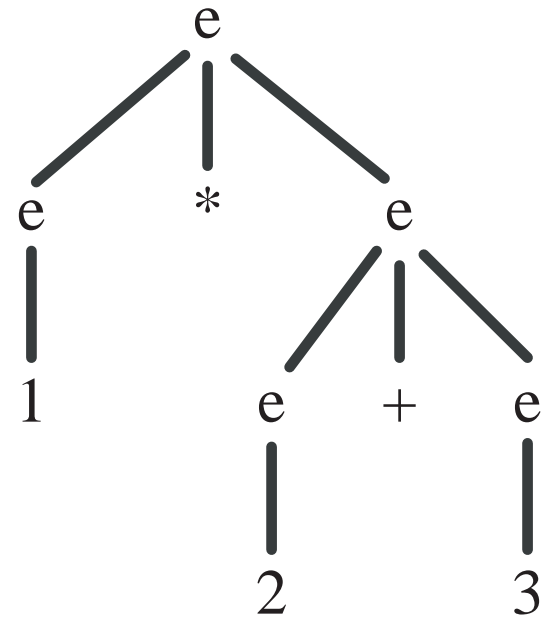
Ambiguity

$e ::= e + e \mid e * e \mid \text{NUMBER}$



$(1 * 2) + 3$

Leftmost derivation



$1 * (2 + 3)$

Rightmost derivation



Ambiguity

- A leftmost-derivation is a derivation in which a production is always applied to the leftmost symbol
 - *A rightmost derivation applies to the rightmost symbol*
- In general, a string may have multiple left and rightmost derivations
- A grammar in which some word has two parse trees is said to be ambiguous



Grammar operations

- Simplify
 - *Not unique*
- Eliminate epsilon-productions
- Normalize



Simplification

- A nonterminal A is *useless* iff
 - $S \rightarrow^* xAy \rightarrow^* xzy$
 - *Otherwise, it is useless*



Garbage collection from the terminals

Lemma For $G = (V, T, P, S)$, we can find an equivalent $G' = (V', T, P', S)$ such that, for each $A \in V$, $A \rightarrow_G^* w$.

let step $V =$

let $\{A \mid A \rightarrow \alpha \text{ for some } \alpha \in (T \cup V)^*\}$

in

let rec fixpoint $V =$

let $V' = \text{step } V$ **in**

if $V' = V$ **then**

V

else

fixpoint V'

in

fixpoint $\{A \mid A \rightarrow w \text{ for some } w \in T^*\}$



Garbage collection from the start

Lemma For $G = (V, T, P, S)$, we can find an equivalent $G' = (V', T', P', S)$ such that, for each $X \in V' \cup T'$, $\exists \alpha, \beta \in (V' \cup T'). S \rightarrow_G^* \alpha X \beta$.

- Place S in V'
- If $A \in V'$ and $A \rightarrow \alpha_1, \dots, \alpha_n$
 - Place all nonterminals of $\alpha_1, \dots, \alpha_n$ in V'
 - Place all terminals of $\alpha_1, \dots, \alpha_n$ in T'
- Repeat until fixpoint



Garbage collection

Theorem Every grammar G is equivalent to a grammar G' with no useless symbols.

Proof Apply the two GC algorithms.



Epsilon-Productions

- An ϵ -production has the form $A \rightarrow \epsilon$
- A definition for balanced parentheses:

$$S ::= \epsilon \mid (S) \mid SS$$



Eliminating epsilon-productions

Theorem All ϵ -productions can be eliminated, except possibly a production of the form $S \rightarrow \epsilon$

Definition A nonterminal A is *nullable* if $A \rightarrow_G^* \epsilon$

Algorithm (finding nullable nonterminals)

- Base: if $A \rightarrow \epsilon$ then A is nullable
- Step: if $A \rightarrow \alpha_1 \cdots \alpha_n$ and $\alpha_1 \cdots \alpha_n$ are all nullable, then A is nullable



Eliminating epsilon-productions

Algorithm (eliminating ϵ productions)

- For each $A \rightarrow X_1 \cdots X_n$ in P , add $A \rightarrow \alpha_1 \cdots \alpha_n$ to P' , where
 - If X_i is not nullable, $\alpha_i = X_i$
 - If X_i is nullable, then $\alpha_i = X_i$ or $\alpha_i = \epsilon$
 - Not all $\alpha_1, \dots, \alpha_n$ are ϵ



Final cleanup

- Add $S \rightarrow \epsilon$ if S is nullable
- Eliminate all *unit* productions of the form $A \rightarrow B$ by replacing all occurrences of A with B
- Every grammar is equivalent to a grammar with no useless symbols, no epsilon production (except possibly for $S \rightarrow \epsilon$), and no unit productions



Normal forms

- Chomsky normal form: every production has the form
 - $A \rightarrow a$
 - $A \rightarrow BC$
- Algorithm
 - *For each terminal a , introduce a production $A \rightarrow a$, and replace all occurrences of a with A*
 - *Eliminate unit productions*
 - *For each production of the form $A \rightarrow X_1 X_2 \dots X_n$ for $n > 2$,*
 - *Add a new production $B \rightarrow X_2 \dots X_n$ (and normalize)*
 - *Add $A \rightarrow X_1 B$*

