

# EE 583

## PATTERN RECOGNITION

Non-metric Methods

Decision Trees

Syntactic (Grammatical) Methods

Elements for Formal Grammars

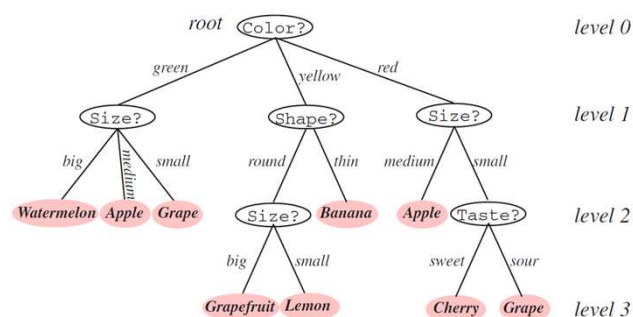
Examples for Pattern Description

## Decision Trees

- It is natural to classify/describe pattern through a sequence of "questions"

→ Decision tree

consisting of *root node, branches, leaf nodes*



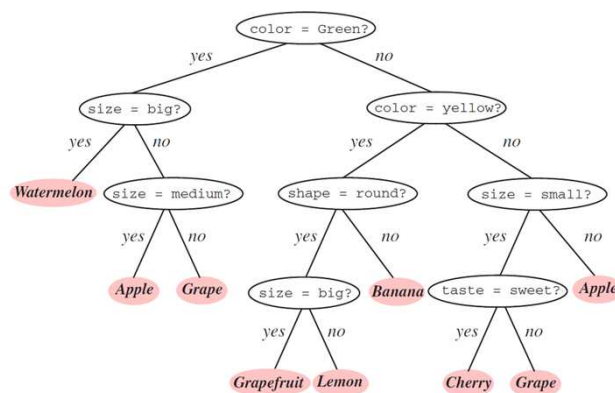
- Apple = (green AND medium) OR (red AND medium)

## Classification and Regression Trees (CART)

- How to create a decision tree from training data?
- A decision tree will progressively split training data into smaller subsets
  - If subset is *pure* → terminate that tree portion (i.e. become a *leaf node*)
  - Otherwise, grow the tree with more nodes
- A general framework : *CART*
  - How many splits will be at each node?
  - Which property should be tested at each node?
  - When does a node become a leaf?
  - How to prune a large tree?
  - If a leaf node is impure, how to assign labels?
  - How to handle missing data?

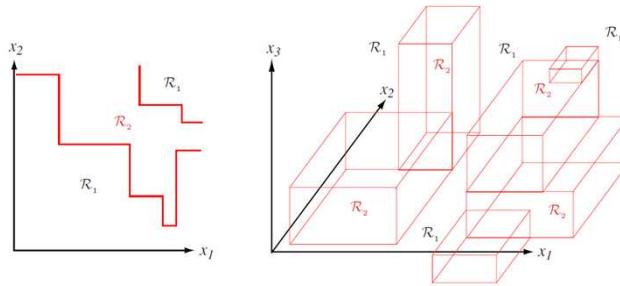
## CART : Number of Splits

- Splitting the training data at each node
  - *Branching factor*
- It is possible to express any problem using binary trees



## CART : Query Selection

- The property to be tested at any node is the fundamental problem
- If the query at each node is of the form  $x_i \leq x_{is}$ , then



- With a sufficiently large tree, any decision boundary can be approximated arbitrarily well

## CART : Node Impurity

- Fundamental principle for tree creation : *simplicity*
- A query is required at each node, makes data reaching its immediate descendants to be as "pure" as possible
- A measure of *impurity*,  $i(N)$  of node  $N$  is necessary
  - Ideally,  $i(N)=0$  ; i.e. all patterns at node  $N$  belongs to same class
  - Let  $P(w_j)$  be the ratio of patterns at node  $N$  belong to class  $w_j$ .

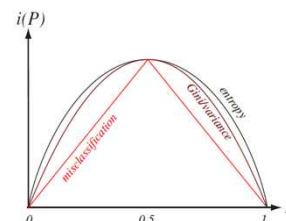
$$i(N) = -\sum_j P(w_j) \log_2 P(w_j) \quad \text{Entropy impurity}$$

$$i(N) = P(w_1)(w_2) \quad \text{Gini impurity for 2 - class}$$

$$i(N) = \sum_{i \neq j} P(w_i)(w_j) \quad \text{Gini impurity for c - class}$$

$$i(N) = 1 - \max_j P(w_j) \quad \text{Misclassification impurity}$$

- Based on these definitions, how to choose the threshold for a query at any node?



## CART : Node Impurity

- The drop in impurity at node  $N$  ( $N_L$  and  $N_R$  are descendants)  
$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_R) i(N_R)$$
- At each node  $N$ , among different query thresholds, maximize  $\Delta i(N)$
- Note that maximization is locally (not globally) optimal
- For  $c$ -class problem, one approach is *twoing*:
  - Let  $C = \{w_1, \dots, w_c\}$ ,  $C_1 = \{w_{i1}, \dots, w_{ik}\}$  and  $C_2 = C - C_1$
  - For every split, compute impurity drops for  $C_2, C_1$
  - Maximize wrt thresholds and selection of  $C_2, C_1$ .

## CART : Criteria for Stop Splitting

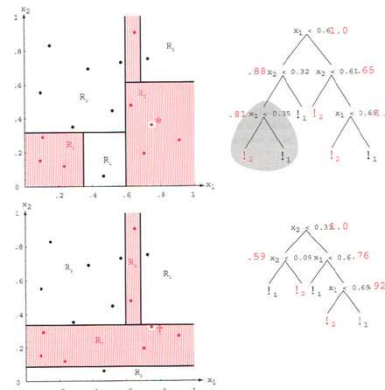
- The traditional approach is to use a *validation set* (% of training data & not used during tree creation)
  - If validation set error is minimized  $\rightarrow$  continue to split
- If best drop in impurity is  $\leq \beta$  at node  $N \rightarrow$  stop split
- If "number of remaining training patterns at node  $N$ "  $\leq \delta \rightarrow$  stop split
- Minimize a global criterion :  $\alpha$  (size of tree) +  $\sum i(N)$ 
  - Relation to *Minimum Description Length (MDL)*
- Check whether a split is "meaningful" by comparing to a random split.

## CART : Pruning

- When a split stops at node  $N$ , there could be loss of possible beneficial splits in the descendant nodes.
- Principal alternative to stop splitting in a decision tree : Pruning
  - A tree is grown fully until all leaf nodes have minimum impurity
  - Then, all pairs of neighboring leaf nodes are examined for elimination using impurity check
- No requirement for a validation set, but complex

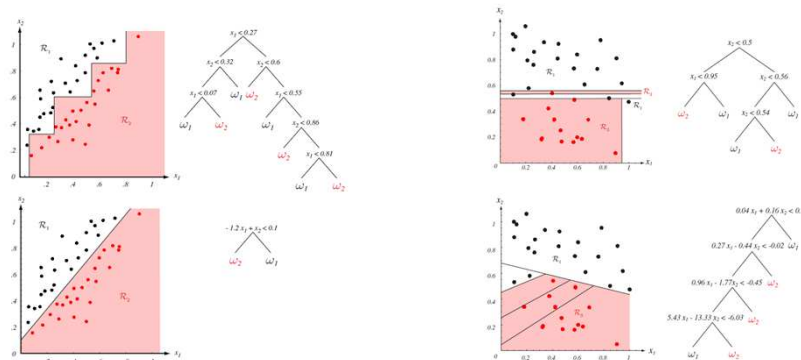
## CART : Assignment of Leaf Node Labels

- Assignment of class labels at leaf nodes is easy, if their impurities are zero.
- With a positive impurity at a leaf node, this node should be labeled by a class with the most points represented
- Problem of *instability* in CART classifiers :
  - Even a single training point can lead to a different decision boundary
- Example on the right →
  - Entropy impurity is used and there is only a single data difference between



## CART : Feature Selection

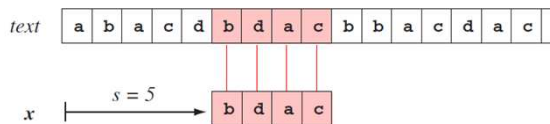
- Feature selection is critical, as expected.
- Principal Component Analysis of the feature vector could be preferable due to obtaining important axes.
- Linear combinations of features also improve results
  - Weights of a general linear classifier is estimated



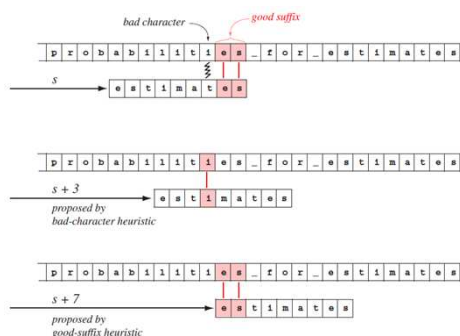
## Recognition with Strings

- Suppose patterns are represented by ordered sequences or *strings* of discrete items (characters)
- Typically, these discrete items are non-numeric and there is no notion of distance between strings
- Important Problems
  - String matching
  - Edit distance
  - String matching with errors
  - String matching with don't-care symbols

## String Matching

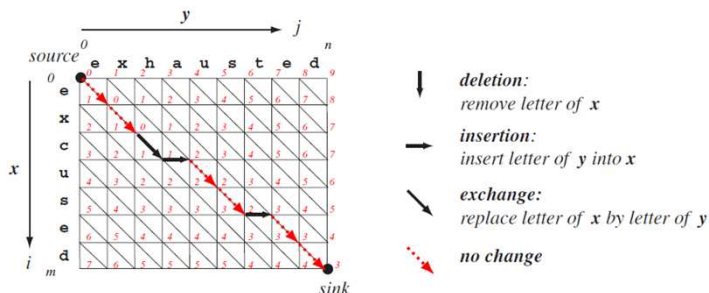


- Apart from naïve string matching, there are more sophisticated matching algorithms
- Boyer-Moore algorithm : Works in reverse order
- Instead of shifts by a single character, multiple character shifts are achieved
- *Good suffix* and *bad character* heuristics try to detect mismatches and proceed multiple characters



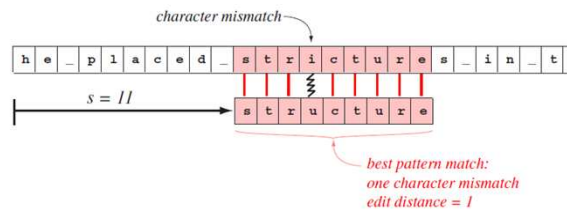
## Edit Distance

- Given two strings  $x$  and  $y$ , compute the minimum number of basic operations, such as character *insertions*, *deletions* and *exchanges*, to transform  $x$  into  $y$ .
- Edit distance calculation : e.g. *excused*  $\rightarrow$  *exhausted*
  - *excused*  $\rightarrow$  *exhused*  $\rightarrow$  *exhoused*  $\rightarrow$  *exhausted*
  - 1 exchange  $\rightarrow$  1 insert  $\rightarrow$  1 insert

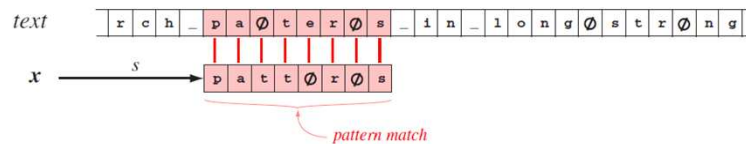


## String Matching with Errors & Don't-cares

- Given a pattern  $x$  and  $text$ , find the shift  $s$  for which the edit distance between  $x$  and  $text$  is minimum



- String matching with don't-care symbols is equivalent to string matching where those symbols can be any charac.



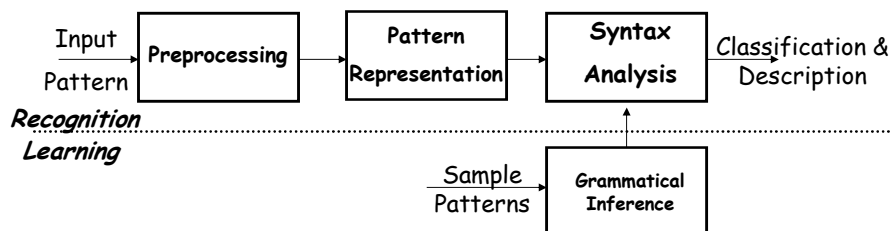
## Introduction

- Earlier developments in pattern recognition research are based on *discriminant* (statistical) approach
- Later, it is realized that in some applications *structural* information that *describes* patterns are more important
- Note the slight difference between
  - Syntactic pattern recognition : recursive description using methods of formal languages
  - Structural pattern recognition : derivation of descriptions using mathematical tools
- SyntPR assumes pattern structure is quantifiable :
  - Formal Grammars  $\rightarrow$  parsing
  - Relational descriptors  $\rightarrow$  relational graph matching



## Typical SyntPR System

- Typical SyntPR consists of 3 major parts :
  - Preprocessing :
    - filtering, restoration, enhancement
  - Pattern description :
    - pattern segmentation, primitive (feature) extraction
  - Syntax analysis
    - Decision whether the representation is syntactically correct against primitives of reference patterns



## Elements of Formal Grammars (1/3)

- Consider parsing of sentence "The students look hopeless"
  1. <sentence>
  2. <noun phrase> <verb phrase>
  3. <article> <noun> <verb phrase>
  4. The <noun> <verb phrase>
  5. The students <verb phrase>
  6. The students <verb> <adverb>
  7. The students look <adverb>
  8. The students look hopeless
- These steps can be described by these set of rules :
  - <sentence> → <noun phrase> <verb phrase>
  - <noun phrase> → <article> <noun>
  - <verb phrase> → <verb> <adverb>
  - <article> → the <noun> → students <verb> → look <adverb> → hopeless
 (Note that → sign means "can be written as")

## Elements of Formal Grammars (2/3)

- Alphabet  $V$  : a finite, nonempty set of *symbols*  
e.g.  $V = \{a, b, c, \dots, z\}$
- String  $S$  : either a single or a sequence of symbols from  $V$   
e.g. 't o h o e' = 'the' ( o produces simply sequence of symbols)
- Null string  $\epsilon$  : an empty string with property  $x \circ \epsilon = \epsilon \circ x = x$
- Set of strings of length  $k$  :  $V \circ V = V^2, V \circ V \circ V = V^3, \dots$
- Set of all nonempty strings :  $V^+ = V \cup V^2 \cup V^3 \cup \dots$  ( $\cup$  : union)
- Set of all strings :  $V^* = \{\epsilon\} \cup V^+$  (closure set of  $V$ )
- Language  $L$  : Any subset of  $V^*$  generated by a *grammar  $G$*

## Elements of Formal Grammars (3/3)

- Grammar  $G$  : a four-tuple consisting of  $G = \{V_T, V_N, P, S\}$ 
  - $V_T$  : a set of **T**erminal symbols  
e.g.  $V_T = \{\text{the, students, look, hopeless}\}$
  - $V_N$  : a set of **N**onterminal symbols  
e.g.  $V_N = \{\langle \text{sentence} \rangle, \langle \text{noun phrase} \rangle, \langle \text{verb phrase} \rangle, \langle \text{article} \rangle, \langle \text{noun} \rangle, \langle \text{verb} \rangle, \langle \text{adverb} \rangle\}$
  - $P$  : Production rules  $a \rightarrow b$  where  $a$  at least once element of  $V_N$   
e.g.  $\langle \text{sentence} \rangle \rightarrow \langle \text{noun phrase} \rangle \langle \text{verb phrase} \rangle, \dots$
  - $S$  : **S**tarting (root) symbol, an element of  $V_N$   
e.g.  $\langle \text{sentence} \rangle$

Note that : 1)  $V_T \cap V_N = \{\}$

$$2) P : A \rightarrow B \quad A \in (V_N \cup V_T)^+ - V_T^+, B \in (V_N \cup V_T)^*$$

- A grammar can be used in either generative (creates strings of terminal symbols) or analytic mode (given a string and  $G$ , find structure of string)
- A *language generated by grammar  $G$* , denoted by  $L(G)$ , is the set of all strings that satisfy (1) each string consists only terminal symbols from  $V_T$  and (2) each string was produced from  $S$  using  $P$  of  $G$

## Types of Grammars (1/2)

According to forms of productions, there are 4 types of grammars :

- Type 0 :  $T_0$  (Free or Unrestricted)
  - No restrictions on the rewrite rules. Too general to be useful; hence in general the problem of deciding whether a particular sentence is generated by this type is usually undecidable
- Type 1 :  $T_1$  (Context sensitive)
  - $aA_i b \rightarrow ab_i b$       $A_i \in V_N, b_i \in (V_N \cup V_T)^* - \epsilon, a, b \in (V_N \cup V_T)^*$   
"b<sub>i</sub> replaces A<sub>i</sub> in the context of a and b"  
e.g.  $P = \{S \rightarrow SC, CB \rightarrow Cb, aB \rightarrow aa, bB \rightarrow bb\}$

## Types of Grammars (2/2)

- Type 2 :  $T_2$  (Context Free [CFG])
  - $A \rightarrow b$       $A \in V_N$  and  $b \in (V_N \cup V_T)^* - \epsilon$   
"b replaces A independent of the context of in which A appears"  
e.g.  $P = \{S \rightarrow aAa, A \rightarrow a, A \rightarrow b\}$
- Type 3 :  $T_3$  (Finite-state [FSG] or Regular )
  - $A \rightarrow aB$  or  $A \rightarrow b$       $A, B \in V_N$  and  $a, b \in V_T$   
"at most 1 non-terminal symbol allowed on each side of production "  
e.g.  $P = \{S \rightarrow aA, S \rightarrow bA, A \rightarrow a, A \rightarrow b\}$

## Comparisons between Grammars

- Given a language, there may be more than one grammar to give the same language; i.e.  $L=L(G1)=L(G2)$ .
- Two grammars,  $G1$  and  $G2$ , are said to be *equivalent* iff  $L(G1)=L(G2)$
- Among 4 types of grammars :
  - $L(T3) \subset L(T2) \subset L(T1) \subset L(T0)$  [ $\subset$  : subset operator]
  - In progressing from a  $T0$  to  $T3$ , production restrictions increase
  - In progressing from a  $T3$  to  $T0$ , representational power increases
  - In progressing from a  $T3$  to  $T0$ , recognition difficulty increases

## Examples for Grammars (1/6)

- CONTEXT-FREE GRAMMAR (CFG) :

$G=\{V_T, V_N, P, S\}$  where  $V_N=\{S, A, B\}$ ,  $V_T=\{a, b\}$ ,  $P$  as

- |                        |                        |                         |                         |
|------------------------|------------------------|-------------------------|-------------------------|
| (1) $S \rightarrow aB$ | (2) $S \rightarrow bA$ | (3) $A \rightarrow aS$  | (4) $A \rightarrow bAA$ |
| (5) $A \rightarrow a$  | (6) $B \rightarrow bS$ | (7) $B \rightarrow aBB$ | (8) $B \rightarrow b$   |

Typical derivations include :

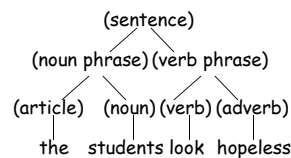
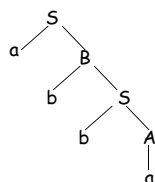
$$S \xRightarrow{(1)} aB \xRightarrow{(8)} ab$$

$$S \xRightarrow{(1)} aB \xRightarrow{(6)} abS \xRightarrow{(2)} abbA \xRightarrow{(5)} abba$$

$$S \xRightarrow{(2)} bA \xRightarrow{(5)} ba$$

$$S \xRightarrow{(2)} bA \xRightarrow{(4)} bbAA \xRightarrow{(5)} bbaA \xRightarrow{(5)} bbaa$$

An alternative method for describing any derivation is to use *derivation trees* :

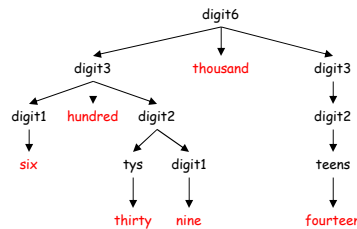


## Examples for Grammars (2/6)

- CONTEXT-FREE GRAMMAR** : A grammar for pronouncing numbers  
 $G = \{V_T, V_N, P, S\}$  where  $S = \{\text{digit6}\}$ ,  
 $V_N = \{\text{digit6, digit3, digit2, digit1, teens, tys}\}$ ,  
 $V_T = \{\text{one, two, ..., ten, eleven, ..., twenty, thirty, ..., ninety, hundred, thousand}\}$ ,  
 $P$  :  

digit6 $\rightarrow$ digit3 thousand digit3	digit6 $\rightarrow$ digit3 thousand OR digit3
digit3 $\rightarrow$ digit1 hundred digit2	digit3 $\rightarrow$ digit1 hundred OR digit2
digit2 $\rightarrow$ teens OR tys digit1 OR digit1	digit1 $\rightarrow$ one OR two OR ... OR nine
teens $\rightarrow$ ten OR eleven OR ... OR nineteen	tys $\rightarrow$ twenty OR thirty OR ... OR ninety

Derivation tree for  
639,014



## Examples for Grammars (3/6)

- FINITE-STATE GRAMMAR (FSG)** :  
 $G = \{V_T, V_N, P, S\}$  where  $V_N = \{S, A, B\}$ ,  $V_T = \{a, b\}$ ,  $P$  as  

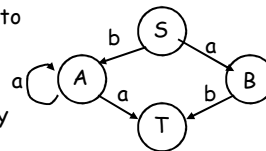
(1) $S \rightarrow aB$	(2) $S \rightarrow bA$	(3) $A \rightarrow a$	(4) $A \rightarrow aA$
(5) $B \rightarrow b$			

Typical derivations include :

$$\begin{array}{l}
 S \xrightarrow{(1)} aB \xrightarrow{(5)} ab \\
 S \xrightarrow{(2)} bA \xrightarrow{(4)} baA \xrightarrow{(3)} baaA \xrightarrow{(3)} baaa
 \end{array}
 \qquad
 \begin{array}{l}
 S \xrightarrow{(2)} bA \xrightarrow{(3)} ba
 \end{array}$$

An alternative method for describing any derivation is to use a graphical representation :

- Nodes  $\rightarrow V_N$  (a special terminal T is also used)
- Arc from X to Y labeled z  $\rightarrow$  production of form  $X \rightarrow zY$
- Arc from X to T labeled z  $\rightarrow$  production of form  $X \rightarrow z$



**Lemma** : Given a FSG, if there is a path from S to T via  $x_1x_2x_3 \dots x_n$ , then string  $x_1x_2x_3 \dots x_n$  is an element of  $L(G)$

## Examples for Grammars (4/6)

- 2-D Line Drawing Description Grammar :

$G_{cyl} = \{V_{cyl}^T, V_{cyl}^N, P_{cyl}, S_{cyl}\}$  where

$V_{cyl}^N = \{Top, Body, Cylinder\}$ ,  $V_{cyl}^T = \{t, b, u, o, s, *, !, +\}$ ,

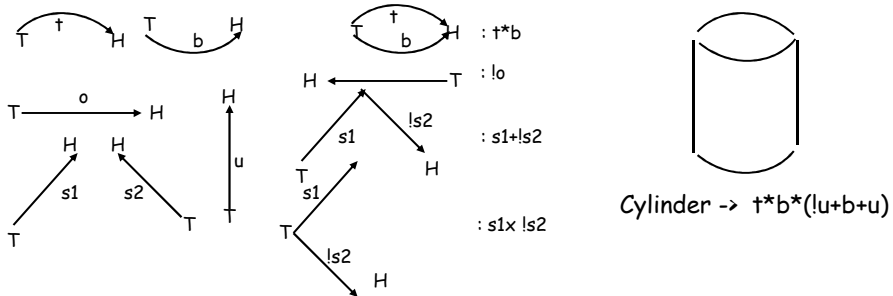
$S_{cyl} = Cylinder$

$P_{cyl}$  : (1)  $Cylinder \rightarrow Top * Body$       (2)  $Top \rightarrow t * b$       (3)  $Body \rightarrow !u + b + u$

[ + represents head to tail concatenation ]

[ \* represents head-head and tail-tail concatenation ]

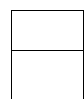
[ ! represents head to tail reversal ]



## Examples for Grammars (5/6)

- Character Description using PDL :

Use a subset of PDL for description of the characters below :



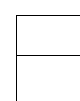
$A = u + ((u + o + !u) * o) + !u$



$C = !o + u + u + o$



$P = u + ((u + o + !u) * o)$

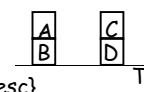
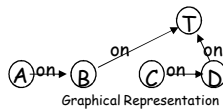


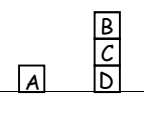
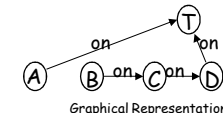
$F = u + (o x u) + o$

## Examples for Grammars (6/6)

- Blocks World Description :

A grammar is presented to describe and classify situations involving stacks of (always 4 but either 2+2 or 3+1) blocks on a table. The difference between two situations is clearly the structure

- $G_2$  (2 blocks) :  $G_2 = \{V_T, V_N, P, S\}$  where
 

  
 $V_N = \{\text{Desc}, \text{left\_stack}, \text{right\_stack}\}, \quad S = \{\text{Desc}\}$   
 $V_T = \{\text{table}, \text{a\_block}, +, \wedge\}$  [ + means "also";  $\wedge$  means "on top of" ]  
 $P = \{\text{Desc} \rightarrow \text{left\_stack} + \text{right\_stack}, \text{Desc} \rightarrow \text{right\_stack} + \text{left\_stack},$   
 $\text{left\_stack} \rightarrow \text{a\_block} \wedge \text{a\_block} \wedge \text{table}, \text{right\_stack} \rightarrow \text{a\_block} \wedge \text{a\_block} \wedge \text{table}\}$

- $G_3$  (3 blocks) :  $G_3 = \{V_T, V_N, P', S\}$  where
 

  
 $P' = \{\text{Desc} \rightarrow \text{left\_stack} + \text{right\_stack},$   
 $\text{left\_stack} + \text{right\_stack} \rightarrow \text{a\_block} \wedge \text{table} + \text{a\_block} \wedge \text{a\_block} \wedge \text{a\_block} \wedge \text{table}$   
 $\text{left\_stack} + \text{right\_stack} \rightarrow \text{a\_block} \wedge \text{a\_block} \wedge \text{a\_block} \wedge \text{table} + \text{a\_block} \wedge \text{table}$

## Remarks on Grammar Generation

- Many concerns might arise about the selection of primitives and grammar generation
  - Primitive selection : Different primitives lead to different grammars
  - Grammars : Some grammars, designed in ad-hoc manner, may generate sentences which can also be produced by others
  - Choice of grammar type is important as well as achieving adequate descriptive power
  - Due to first three reasons, grammar design is an iterative process