MODULE 4

Learning – Rote Learning – Learning by advice – Learning in Problem Solving – By parameter adjustment with macro operators, Chunking, Learning from examples – Winston's Learning program, Version spaces – Positive and Negative examples – Candidate Elimination – Decision Trees – ID3 Decision Tree Induction algorithm.

Learning

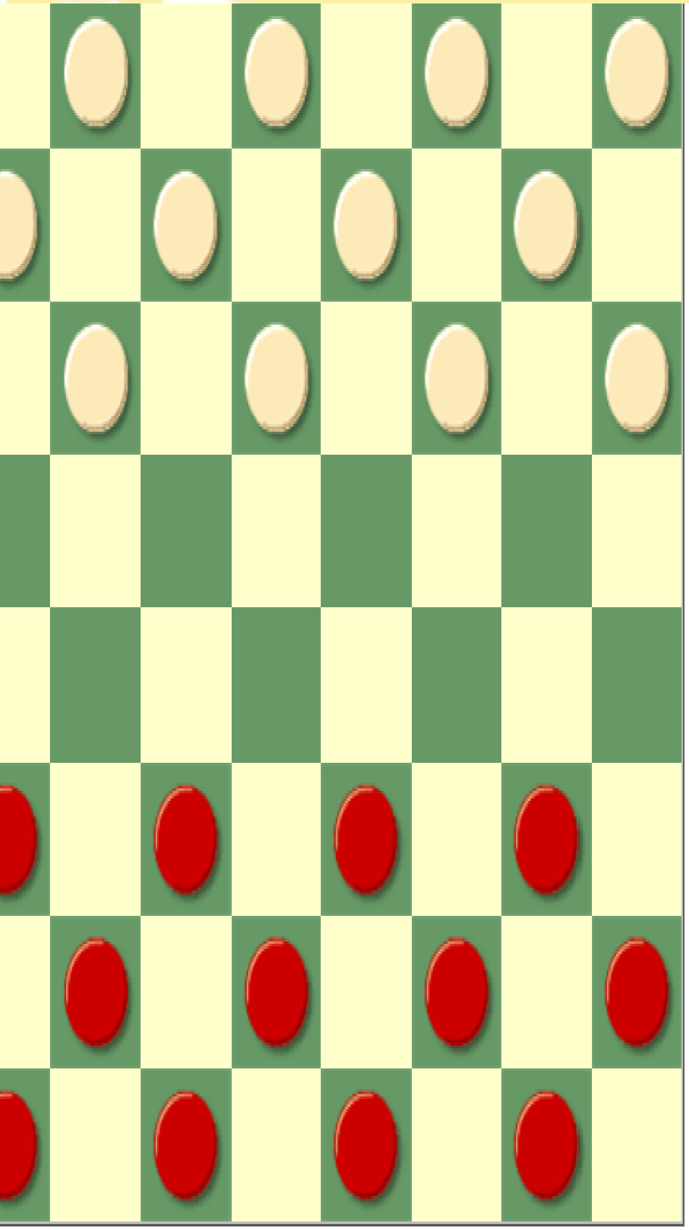
- The ability to adapt to new surroundings and to solve new problems.
- An important characteristic of intelligent entities.
- Machines are called intelligent only if they are able to learn new things and adapt to new situations.

Ways in which learning is done

- Skill Refinement.
- Knowledge acquisition.
- Taking advice from others.
- Learning from problem solving experience.
- Learning from examples.

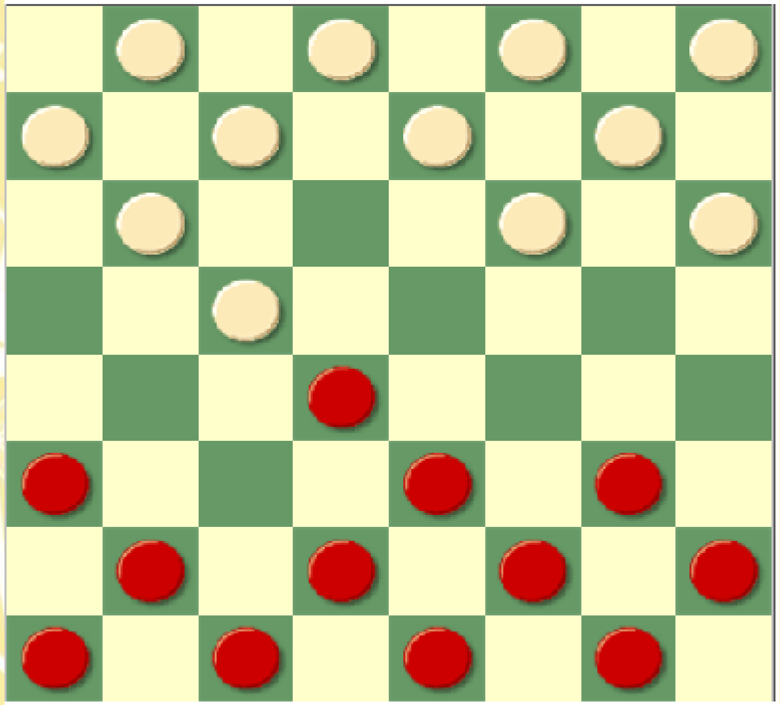
Rote Learning

- Basic Learning Activity.
- It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base.
- Also called memorization as knowledge without any modification is simply copied.
- Computed values are stored.
- Data caching saves time.



Samuel's Checkers Program - Robert Nealy

[Event "Exhibition Game"]
[Black "Samuel's Checkers Program"]
[White "Robert Nealy"]
[Result "1-0"]
1. 11-15 23-19 2. 8-11 22-17 3. 4-8 17-13 4. 15-18 24-20 5. 9-14 26-23 6. 10-15 19x10 7. 6x15 28-24 8. 15-19 24x15 9. 5-9 13x6 10. 1x26 31x15 11. 11x18 30-26 {? what an ugly move! 25-22 is the obvious choice here. Why weaken your back rank like this when it's not necessary?} **12. 8-11 25-22 13. 18x25 29x22 14. 11-15 27-23 15. 15-19 23x16 16. 12x19 32-27** {?? 20-16 here is said to draw. I haven't checked this.} **17. 19-24 27-23 18. 24-27 22-18 19. 27-31 18x9 20. 31x22 9-5 21. 22-26 23-19 22. 26-22 19-16 23. 22-18 21-17 24. 18-23 17-13 25. 2-6** {?? awful! 23-18 wins!} **16-11** {?? horrible! 16-12 instead draws.} **26. 7x16 20x11 27. 23-19** {and white resigns as black will win the man on 11. It would have been interesting to see whether Samuel's program would really have won with 3 vs 2 kings!} 1-0



1. 11-15 23-19

- Rote Learning is simple. Does not involve any complex problem solving capabilities.
- There are certain features/capabilities that are necessary to solve complex problems.
    - Organized storage of information.
    - Generalization.

Learning by taking Advice

- Computer do nothing without a program to run.
- Program in the form of instructions is given.
- Advice given has to be properly represented and operationalized before using them.
- Eg : FOO(First Operational Operationaliser) is a learning system which is used to learn the game of hearts, a card game.

Learning by taking Advice

- Computer do nothing without a program to run.
- Program in the form of instructions is given.
- Advice given has to be properly represented and operationalized before using them.
- Eg : FOO(First Operational Operationaliser) is a learning system which is used to learn the game of hearts, a card game.

Learning in Problem Solving

- 3 basic methods in which system can learn from its own experiences.

<u>Learning by Parameter Adjustment</u>

- Many programs use static evaluation function which combines many factors into a single score giving the desirability of a particular board position.
- In learning there is a slight variation in the evaluation function used.

- The function is represented as a polynomial of the form
  - $c_1t_1 + c_2t_2 + c_3t_3 + \ldots$

  Here t terms are the values of the features and the c terms are the weights.

- **So the basic idea of parameter adjustment is to:**
  - Start with some estimate of the correct weight.
  - Modify the weight in the program on the basis of accumulated experiences.
  - Features that appear to be good predictors will have their weights increased and that of bad ones will be decreased.

Learning by Macro Operators

- Sequence of actions treated as a whole is called macro operators.
- Once a problem is solved, the learning component takes the computed plan and stores it as macro operator.
- Preconditions are initial conditions of the problem just solved and its post conditions correspond to the goal just achieved.
- Macro-operators were used in early problem solving systems. Eg: STRIPS used macro operator in its learning phase.

- STRIPS is used in block world situation.
  - Given : ON(C,B) and ON(A,Table) are both true.
  - GOAL: ON(A,B) and ON(C,Table)
  - Plan with 4 steps : UNSTACK(C,B), PUTDOWN(C), PICKUP(A), STACK(A,B).
  - STRIPS now build a macro-operator called MACROP with preconditions, postconditions and the body containing 4 steps.
  - This MACROP can be used in future operation by generalizing with variables.

Learning by Chunking

- Used by problem solver systems that make use of production systems.
- A production system consists of a set of rules that are in if-then form. It also contains knowledge base, control strategy and a rule applier.
- By applying rules new results can be obtained which are called chunks.
- SOAR is a general architecture for building intelligent systems. It is based on production system.

- In SOAR:
  - Knowledge in the form of productions or rules are stored in long term memory.
  - Short term memory(working memory) is a buffer that acts as a storage area for facts deduced (which are called chunks).
  - Problem solving activity takes place as state space traversal.
  - All intermediate and final results are remembered(or chunked) for future references.

Learning from Examples: Induction

- Inductive Learning(or Concept learning) is the process of learning from trained positive and negative examples.
- It tries to induce a general rule from a set of observed instances.
- Used in classification process.
- Classification is the process of assigning to a particular input, the name of a class to which it belongs.
- Classes has to be defined before doing classification. It can be done in many ways:

- Classes has to be defined before doing classification. It can be done in many ways.
  - Isolate a set of features that are relevant to the task domain. Each class is defined by a weighted sum of values of these features. Such a fn has the form:

c1t1 + c2t2 + c3t3 +...

Here t corresponds to a value of relevant parameter and each c represents weight to be attached to the corresponding t.

  - Classes can also be defined as a structure consisting of features. Eg: if task is to identify animals, body of each type of animal is stored as structure with features such as color, length of neck, feathers etc.

Version Spaces

- Goal is to produce a description that is consistent with all positive examples and none of negative examples in training set.
- EG1: Consider example task of learning target concept “Days on which Jane enjoys her favorite water sport”.
  - Selected attributes(or features) are Sky, Air temp,Humidity, Wind, Water, Forecast.
  - For each attribute , the value of it will be either:
    - “?” or a variable that denotes any value for attribute that it can take.
    - A specific value (eg warm for attribute Air temp) .
    - “Ø” that indicates no value is acceptable.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

- <u>Positive Examples</u>
  - Examples that satisfies all constraints(or values) of description(or hypothesis) thereby achieving the target concept.
- <u>Negative Examples</u>
  - Examples that violate any or all of the constraints of description of target concept.
- EG2: Target Concept : Japanese Economy Car.

Car023

origin :	Japan
manufacturer :	Honda
color :	Blue
decade :	1970
type :	Economy

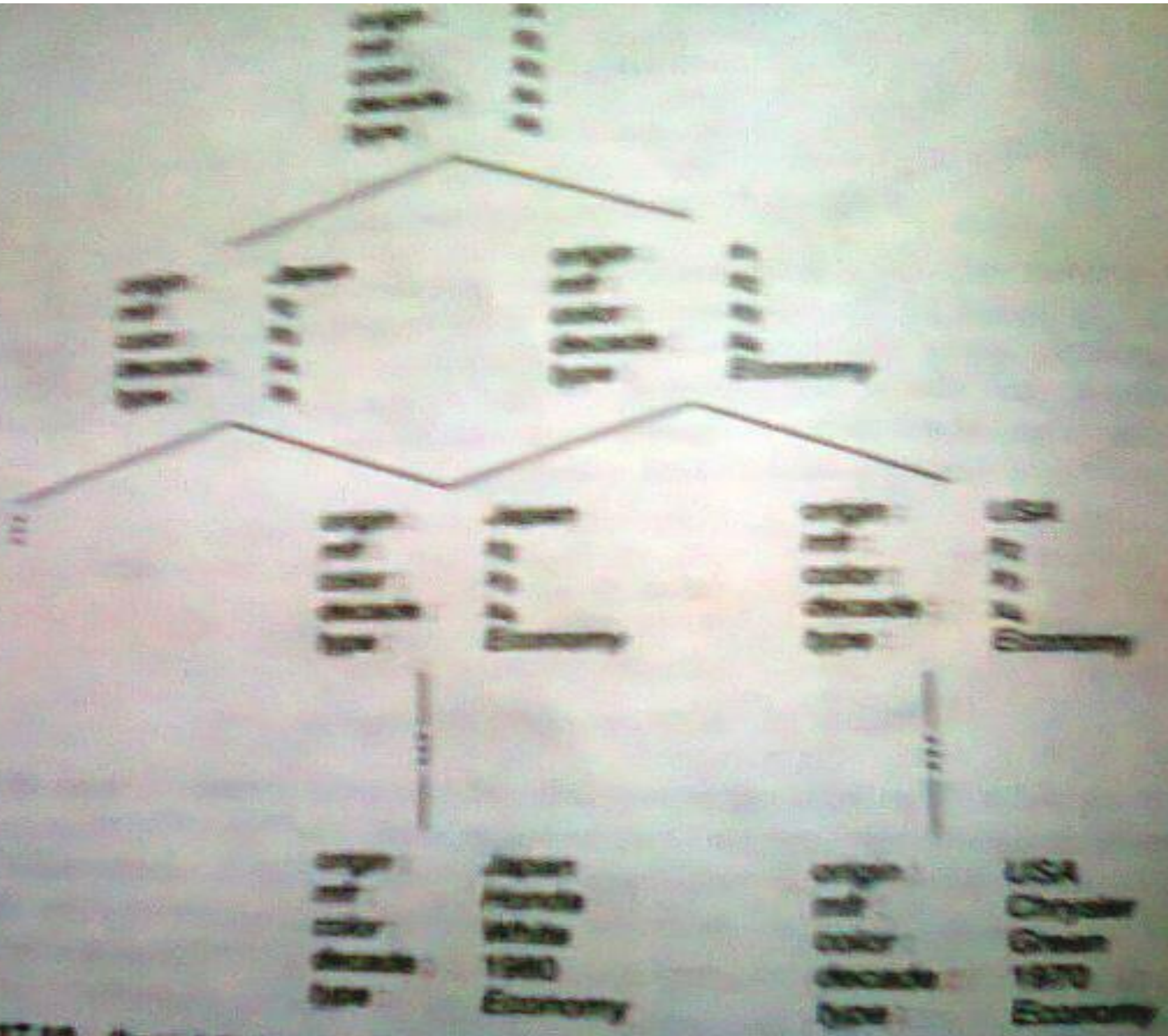
Fig. 17.7 *An Example of the Concept Car*

origin	$\in$	{*Japan*, USA, Britain, Germany, Italy}
manufacturer	$\in$	{*Honda*, Toyota, Ford, Chrysler, Jaguar, BMW, Fiat}
color	$\in$	{*Blue*, Green, Red, White}
decade	$\in$	{1950, 1960, 1970, 1980, 1990, 2000}
type	$\in$	{*Economy*, Luxury, Sports}

Fig. 17.8 *Representation Language for Cars*

origin :	*Japan*
manufacturer :	x_1
color :	x_2
decade :	x_3
type :	*Economy*

Fig. 17.9 *The Concept 'Japanese economy car'*



17.10 *Partial Ordering of Concepts Specified by the Representation Language*

Zoom

- The entire partial ordering is called concept space, and is shown below:

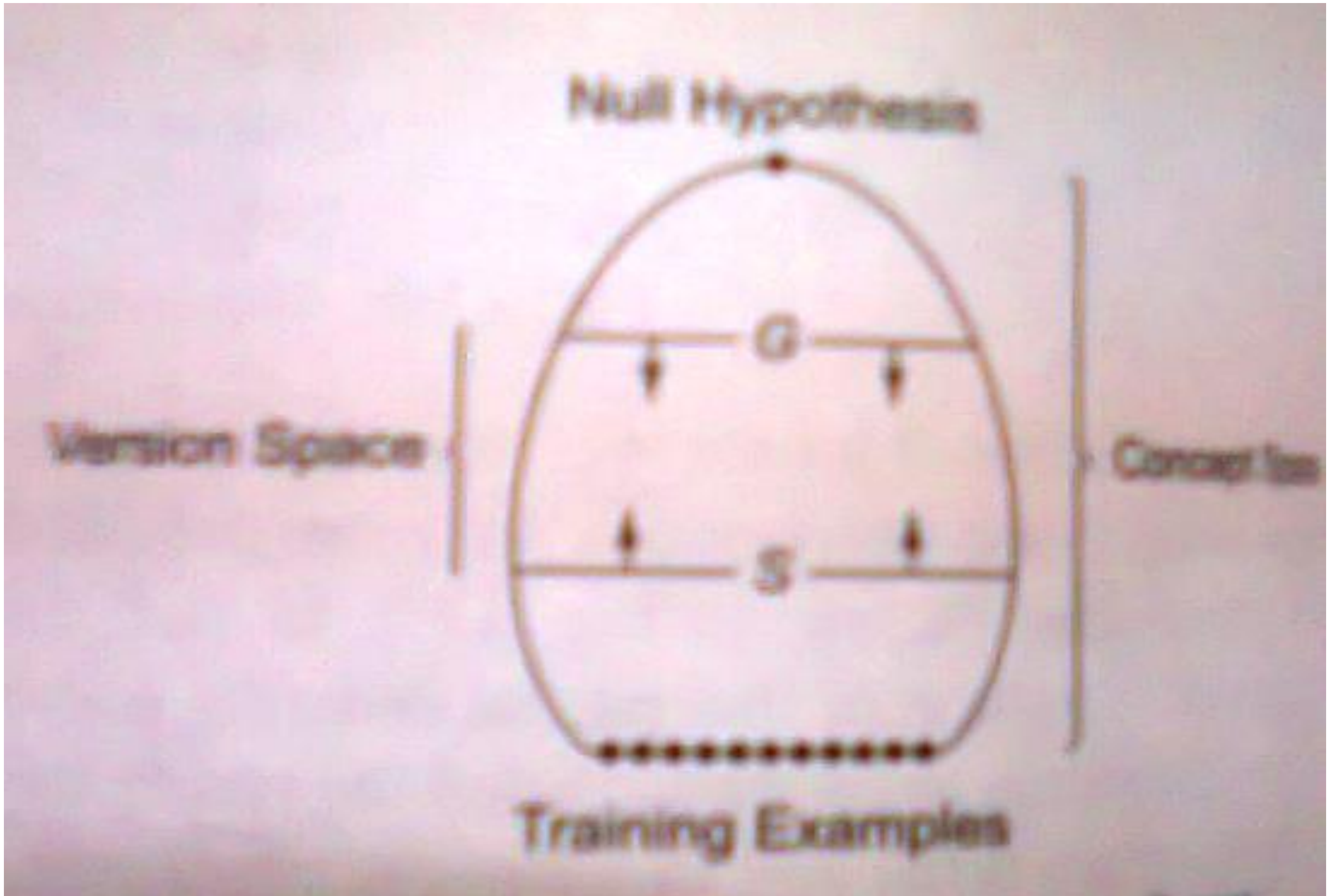


Fig. 17.11 *Concept and Version Space*

- At the top of concept space is null hypothesis that consists of only variables and at the bottom are all possible training instances, which contain no variables.
- Target concept lies somewhere between two extremes.
- <u>Representation of version space</u>
  - Version space consists of two subsets G and S.
  - G refers to most general description(or hypothesis) seen so far.
  - S refers to the most specific description consistent with training examples.

- Version space is the set of all descriptions that lies between some element of G and S in the partial order of concept space.
- The algorithm used to narrow down the version space is called **candidate elimination algorithm**.
- In this algorithm when a positive training example is received, the S set is made more general. On the other hand, negative training examples serve to make G more specific.
- If S and G sets converge, the range of hypothesis will narrow to a single concept description.

Algorithm: Candidate Elimination

Given: A representation language and a set of positive and negative egs.

Compute: A concept description that is consistent with all positive egs and none of the negative egs.

1. Initialize G to contain one element : the null description.

2. Initialize S to contain one example: the first positive example.

3. Accept a new training example.

If it is a positive example, first remove from G any descriptions that do not cover the example .

Remove from S all descriptions that violate the example.

Generalize the elements of S as little as possible to cover the new training eg

If it is a negative example, first remove from S any descriptions that cover the example .

Remove from G descriptions that are not consistent with the example.

Specialize the elements of G as little as possible so that negative eg is no longer covered by any of the elements of G.

4. If S and G are both singleton sets,

  - if they are identical, output their value and halt.

  - if they are different, training cases were inconsistent. Output this result and halt.

Otherwise goto step 3.

- Consider the target concept Japanese Economy car. Training egs: 1, 3 and 5 are +ve training and egs. 2 and 4 are –ve ones

origin	:	Japan
mfr	:	Honda
color	:	Blue
decade	:	1980
type	:	Economy

origin	:	Japan
mfr	:	Toyota
color	:	Green
decade	:	1970
type	:	Sports

origin	:	Japan
mfr	:	Toyota
color	:	Blue
decade	:	1990
type	:	Economy

origin	:	USA
mfr	:	Chrysler
color	:	Red
decade	:	1980
type	:	Economy

origin	:	Japan
mfr	:	Honda
color	:	White
decade	:	1980
type	:	Economy

Winston’s Learning Program

- Describes structural concept of learning problem.
- Program is operated in a simple Blocks World Domain. Goal is to construct representation of the concepts in block domain.
- Eg: it learned concepts such as house, tent and arch. It then use the learned concept to classify future examples into house, tent or arch.
- A semantic representation is used to represent structural description of the object.

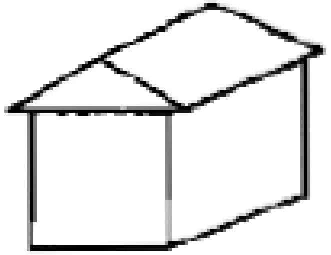
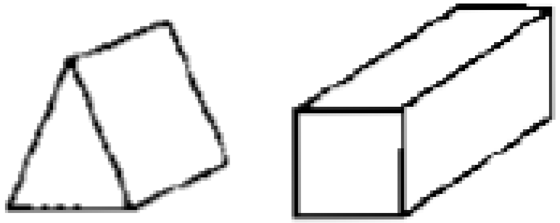
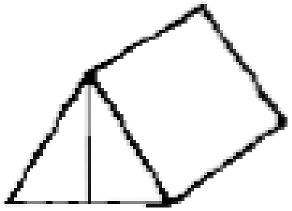
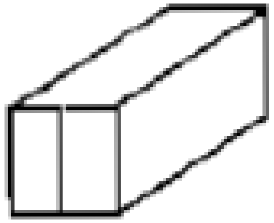
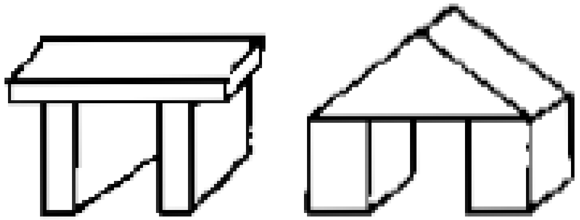
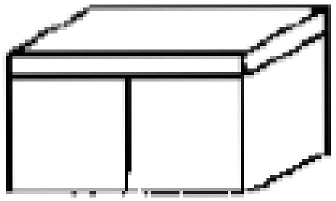
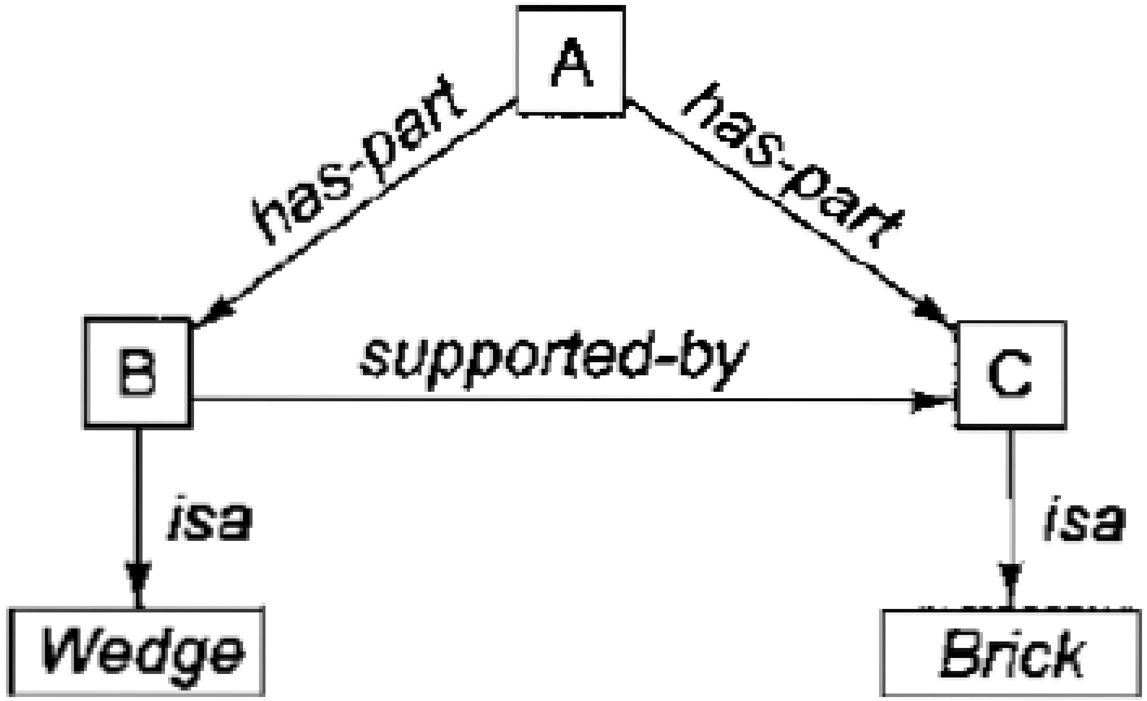
	Concept	Near Miss
House		
Tent		
Arch		

Fig. 17.2 *Some Blocks World Concepts*

- A near miss is an object that is not an instance of the concept in question but is very similar to such instances.
- Basic approaches of Winston's Program
  1. Begin with structural description of one known instance of the concept. Call that description concept definition.
  2. Examine descriptions of other known instances of the concept. Generalize the definition to include them.
  3. Examine descriptions of near misses of the concept. Restrict the description to exclude them.
- .


A
has-part
has-part
B
supported-by
C
isa
isa
Wedge
Brick

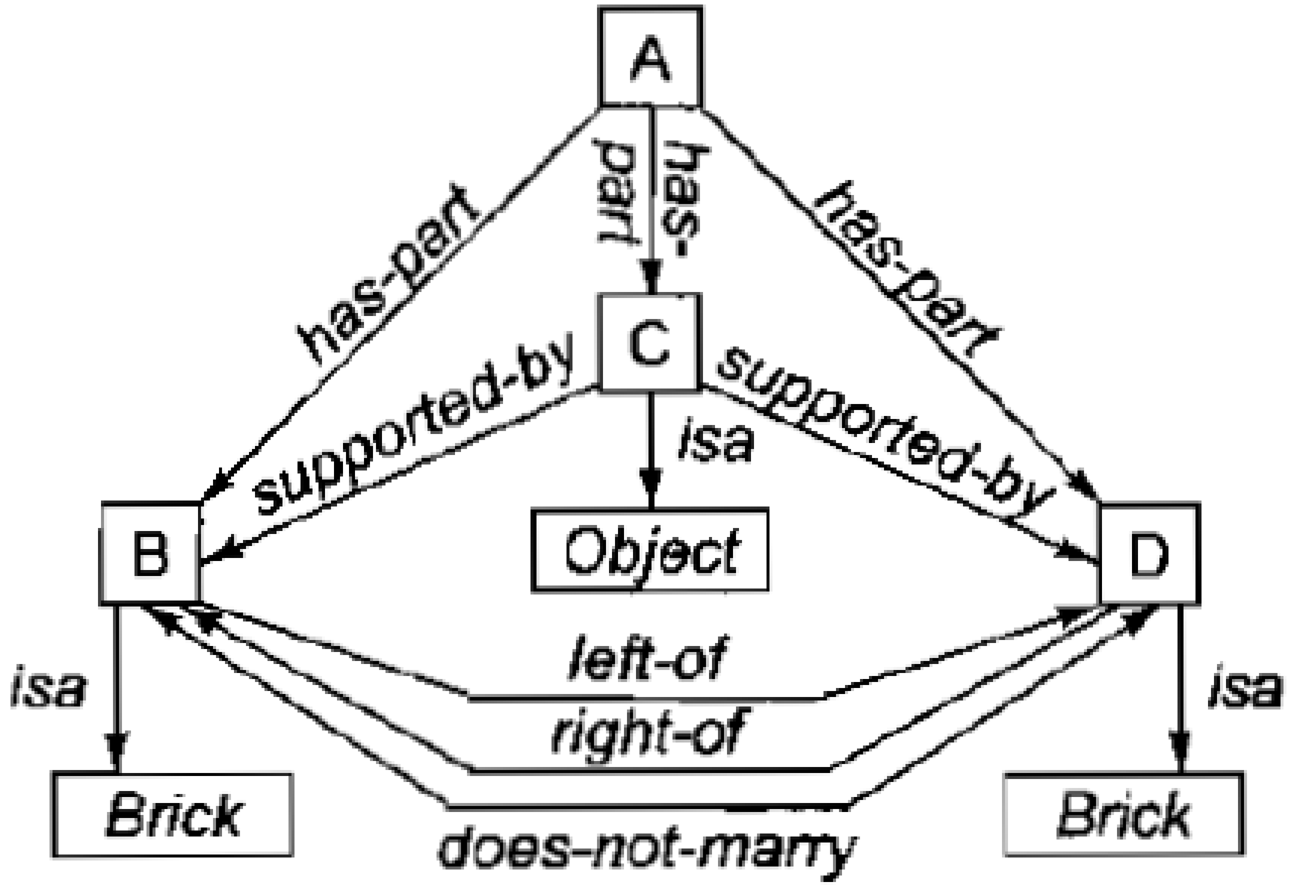


Fig. 17.5 *The Arch Description after Two Examples*

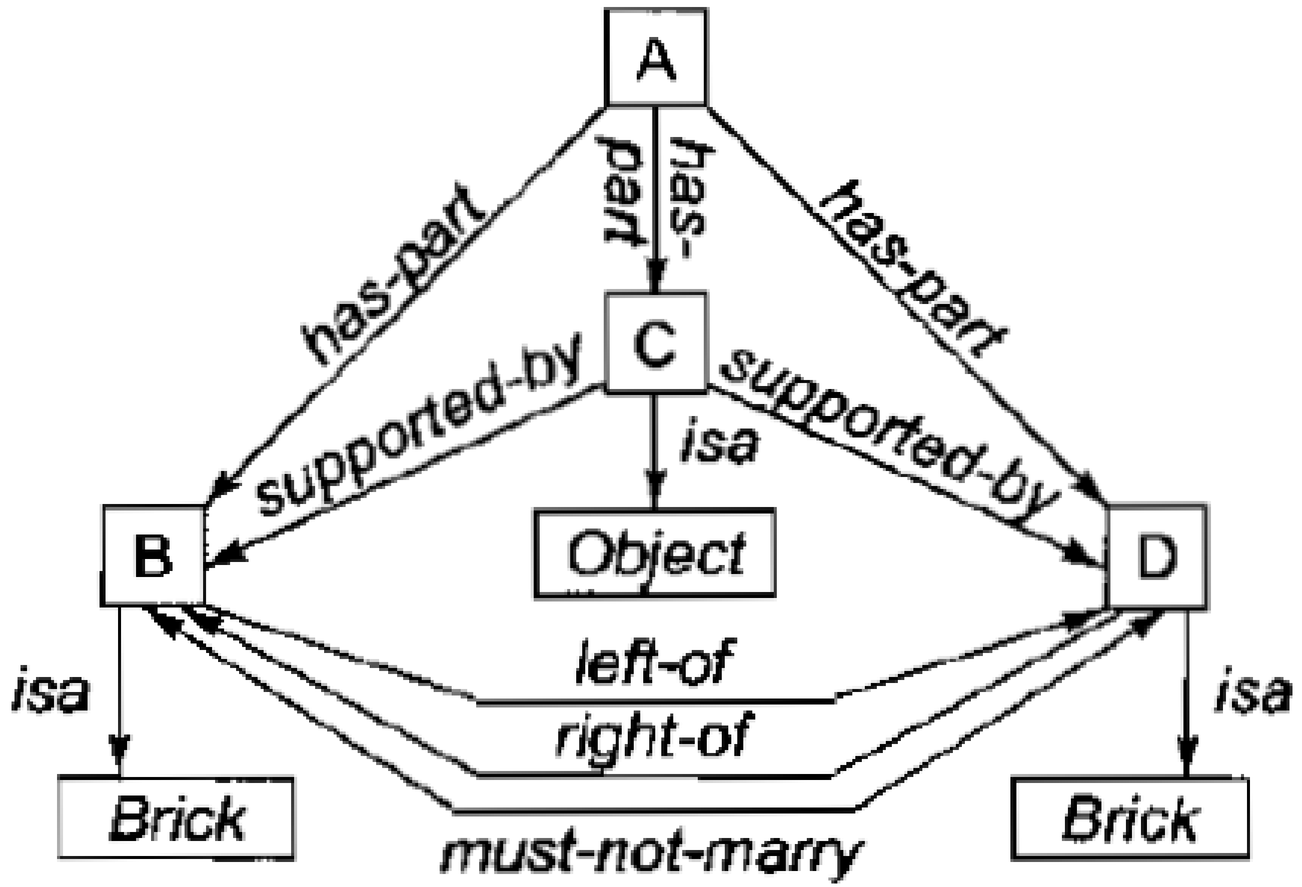


Fig. 17.6 *The Arch Description after a Near Miss*

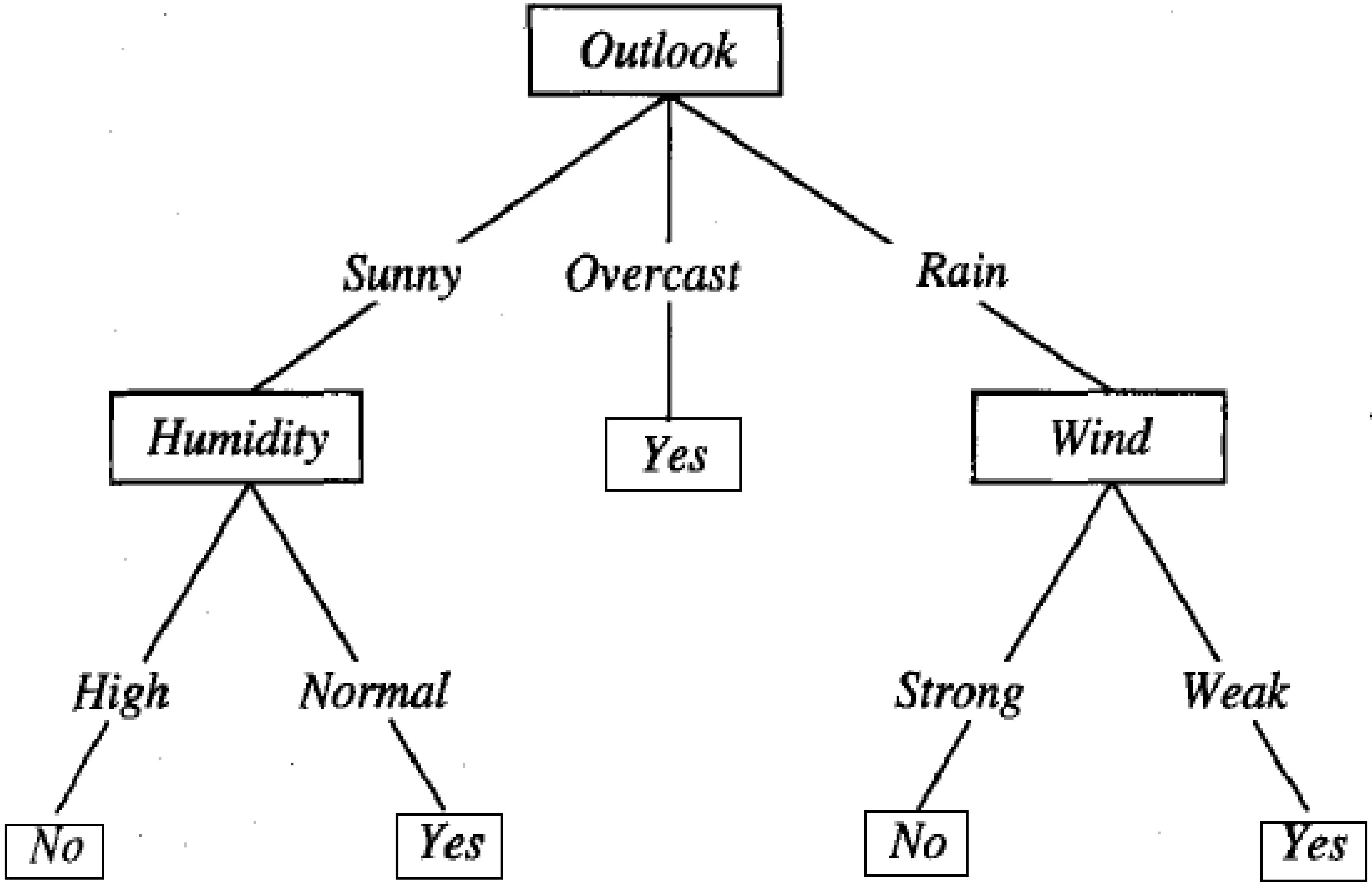
Decision Tree

- A decision tree is a simple representation for classifying examples.
- A decision tree takes as input an object or situation described by a set of properties, and returns a decision -the predicted output value for the input.
- A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labelled with an input feature.
- The arcs coming from a node are labelled with each of the possible values of the feature.
- Each leaf of the tree is labelled with a class.

- Decision trees usually represent Boolean functions where each example is classified as true(positive) or false(negative).
- They classify future examples by sorting them down from the root to some leaf node, which provides classification of examples.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

TABLE 3.2
Training examples for the target concept *PlayTennis*.


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes

ID3 Decision Tree Induction Algorithm

- ID3 is a decision tree learning algorithm.
- It builds a decision tree from a fixed set of examples. And the resulting tree is used to classify future samples.
- The leaf nodes of the decision tree contain the class name whereas a non-leaf node is a decision node.
- The decision node is an attribute test with each branch being a possible value of the attribute.
- ID3 uses information gain to help it decide which attribute goes into a decision node.

<u>Working of Algorithm</u>

- ID3, learns decision trees by constructing them top down, beginning with the question “which attribute should be tested at the root of the tree?”
- To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples.
- A statistical property, called ***information gain*** is used that measures how well a given attribute separates the training examples according to their target classification.

- In order to define information gain precisely, ***Entropy*** is defined.
- Entropy as a measure of the impurity in a collection of training examples.
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this boolean classification is

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where $p_{\oplus}$ is the proportion of positive examples in S and $p_{\ominus}$, is the proportion of negative examples in S.

In all calculations involving entropy we define 0 log 0 to 0

$$\begin{aligned} Entropy([9+, 5-]) &= -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) \\ &= 0.940 \end{aligned}$$

- Next, a measure of the effectiveness of an attribute in classifying the training data is defined. The measure used is ***information gain,*** which is the expected reduction in entropy caused by partitioning the examples according to the selected attribute.
- The information gain, ***Gain(S, A)*** of an **attribute A** relative to a **collection of examples** ***S***, is defined as,

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Where, Values(A) is the set of all possible values for attribute A, and Sv, is the subset of S for which attribute A has value v

- The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node.(i.e., down the branch corresponding to the example's value for this attribute).
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

- The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node.(i.e., down the branch corresponding to the example's value for this attribute).
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MAJORITY-VALUE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value $v_i$ of best do
      add a branch to tree with label $v_i$
      $examples_i$ ← {elements of examples with best = $v_i$}
      subtree ← DECISION-TREE-LEARNING($examples_i$, attributes − best,
                                           MAJORITY- VALUE(examples))
      end
      return tree
```