# INDUCTIVE LOGIC PROGRAMMING

COMP3411/9814 Artificial Intelligence

# Shape of Discriminator

- Machine Learning algorithms can be characterised by the way the divide up the attribute space.
- What is the shape of the surface that separates classes?

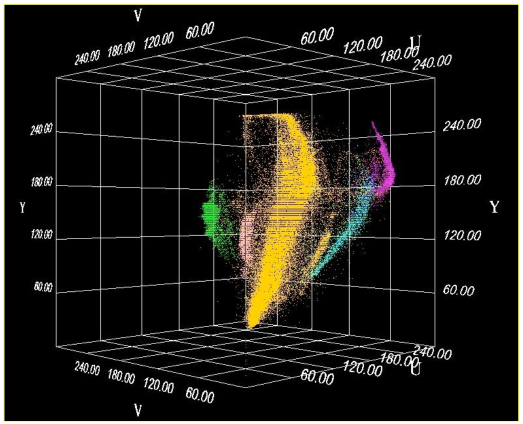
# Learning in Perception

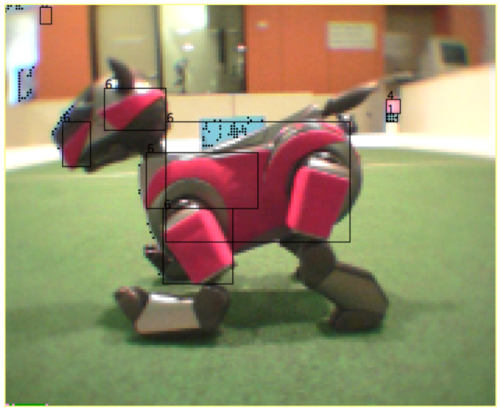
105, 117, 113, orange
105, 116, 112, orange
102, 117, 113, orange
102, 116, 114, orange
103, 117, 111, orange
103, 117, 112, orange
103, 118, 110, orange
99, 117, 112, orange
98, 116, 118, orange
99, 116, 117, orange
106, 111, 114, orange
114, 115, 123, yellow
128, 111, 124, yellow
150, 112, 121, yellow
173, 111, 117, yellow
171, 110, 110, yellow
145, 112, 108, yellow
121, 111, 110, yellow
106, 111, 112, orange
107, 112, 112, orange
104, 114, 114, orange
100, 115, 114, orange
100, 117, 117, orange
98, 115, 113, orange
100, 114, 116, orange
97, 117, 112, orange
102, 115, 109, orange
104, 118, 109, orange
100, 114, 108, orange
97, 115, 110, orange
101, 114, 110, orange
99, 116, 113, orange
98, 116, 113, orange

C4.5

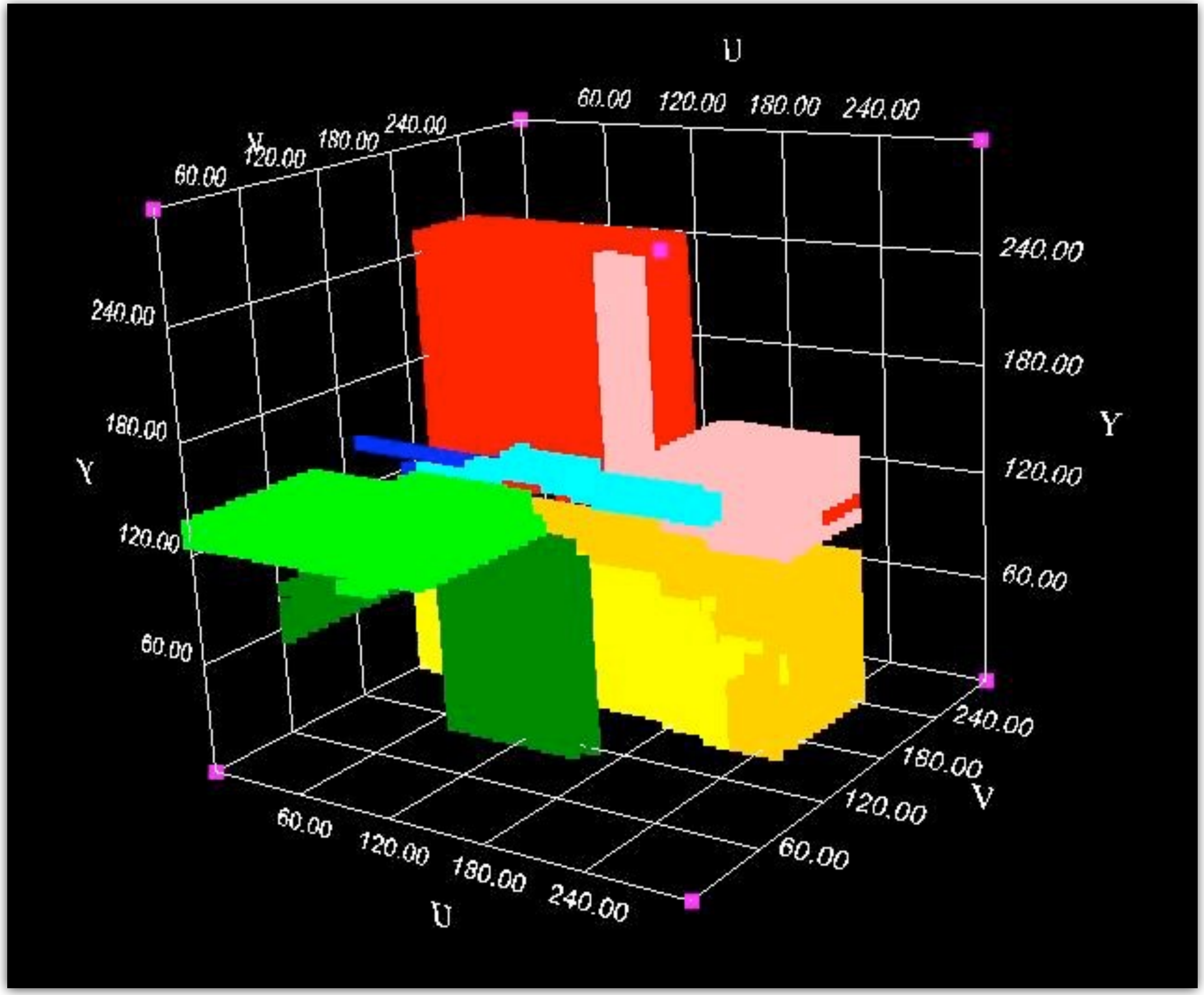
Warning: In practice data sets and decision trees are much larger than this example!

```
if (u <= 107)
    yellow;
  else
    if (v <= 100)
        orange;
      else
        if (y <= 136)
            orange;
          else
            yellow;
```

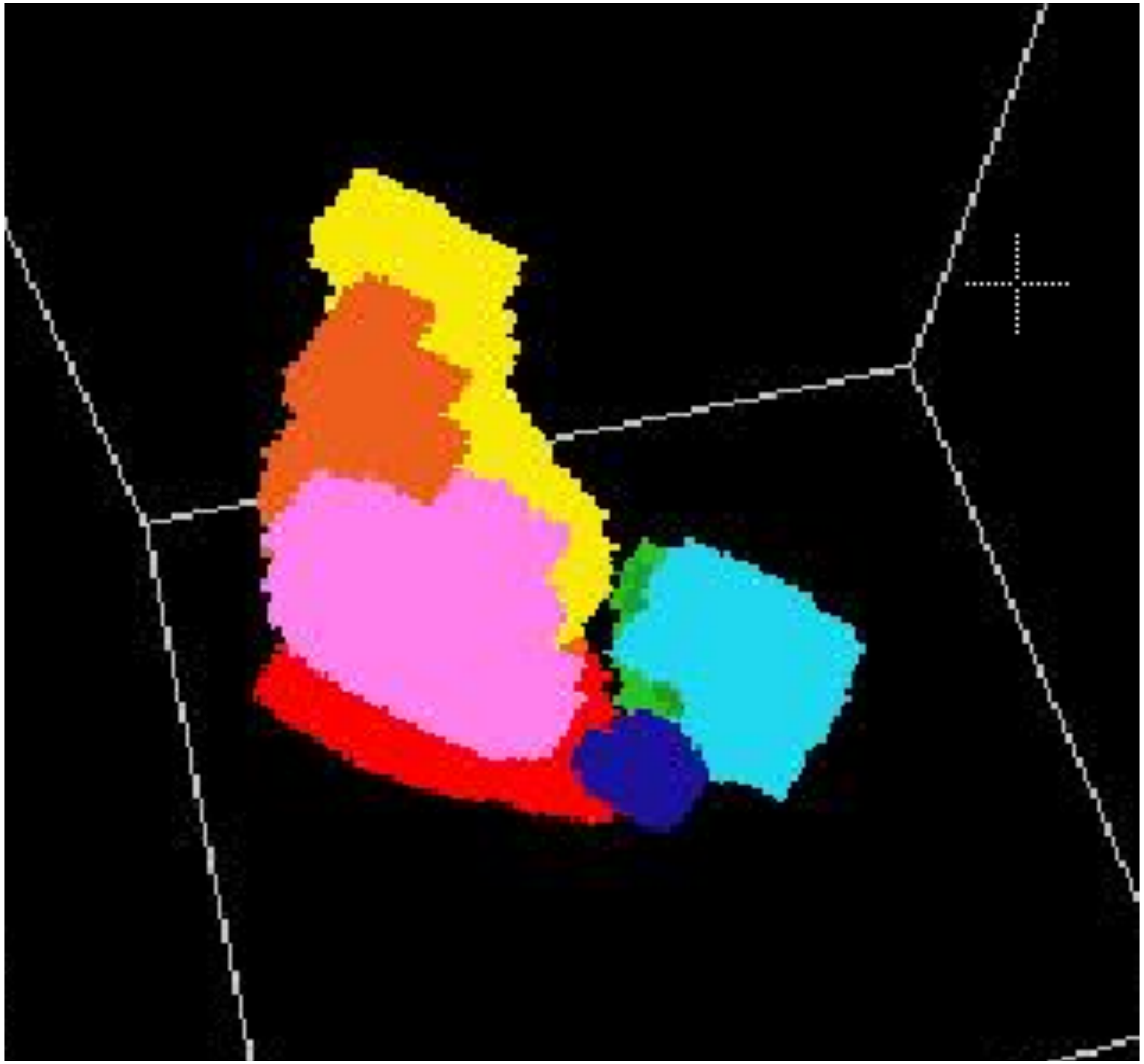


# Colour Classes using C4.5



# Nearest Neighbour



# Description Language

- A concept can also be represented by sentences in a description language.

- May be if-then-else, or rules, like Horn clauses (Prolog):

The colour decision tree can be written as:

```
yellow :- u =< 107.
yellow :- h > 107, v =< 100, y > 136.
orange :- u > 107, v =< 100, y =< 136.
```

# Generalisation Ordering

- If we can define a generalisation ordering on a language, learning can be done by syntactic transformations.

- E.g

$$class \leftarrow size = large \tag{1}$$

is a generalisation of

$$class \leftarrow size = large \wedge colour = red \tag{2}$$

because (2) describes a more constrained set

# Subsumption

A clause $C_1$ *subsumes*, or is more general than, another clause $C_2$ if there is a substitution σ such that $C_2 \supseteq C_1 \sigma$.

$$class \leftarrow size = large$$

$$class \leftarrow size = large \wedge colour = red$$

The least general generalisation of

| | | |
|---|---|---|
| | p(g(a), a) | (3) |
| and | p(g(b), b) | (4) |
| is | p(g(X), X). | (5) |

Under the substitution {a / X} (5) is equivalent to (3).

Under the substitution {b / X} (5) is equivalent to (4).

# Inverse Substitution

The least general generalisation of

p(g(a), a)

and p(g(b), b)

is p(g(X), X).

and results in the inverse substitution {X / {a, b}}

# Least General Generalisation

E.g.

The result of heating this bit of iron to 419˚C was that it melted.

The result of heating that bit of iron to 419˚C was that it melted.

---

The result of heating any bit of iron to 419˚C was that it melted.

We can formalise this as:

melted(bit1) :– bit_of_iron(bit1), heated(bit1, 419).

melted(bit2) :– bit_of_iron(bit2), heated(bit2, 419).

---

melted(X) :– bit_of_iron(X), heated(X, 419).

# Least General Generalisation

- Find a substitution so that there is no other clause that is more general

# LGG of Clauses

q(g(a)) :– p(g(a), h(b)), r(h(b), c), r(h(b), e).

q(x) :– p(x, y), r(y, z), r(h(w), z), ~~s(a, b)~~.

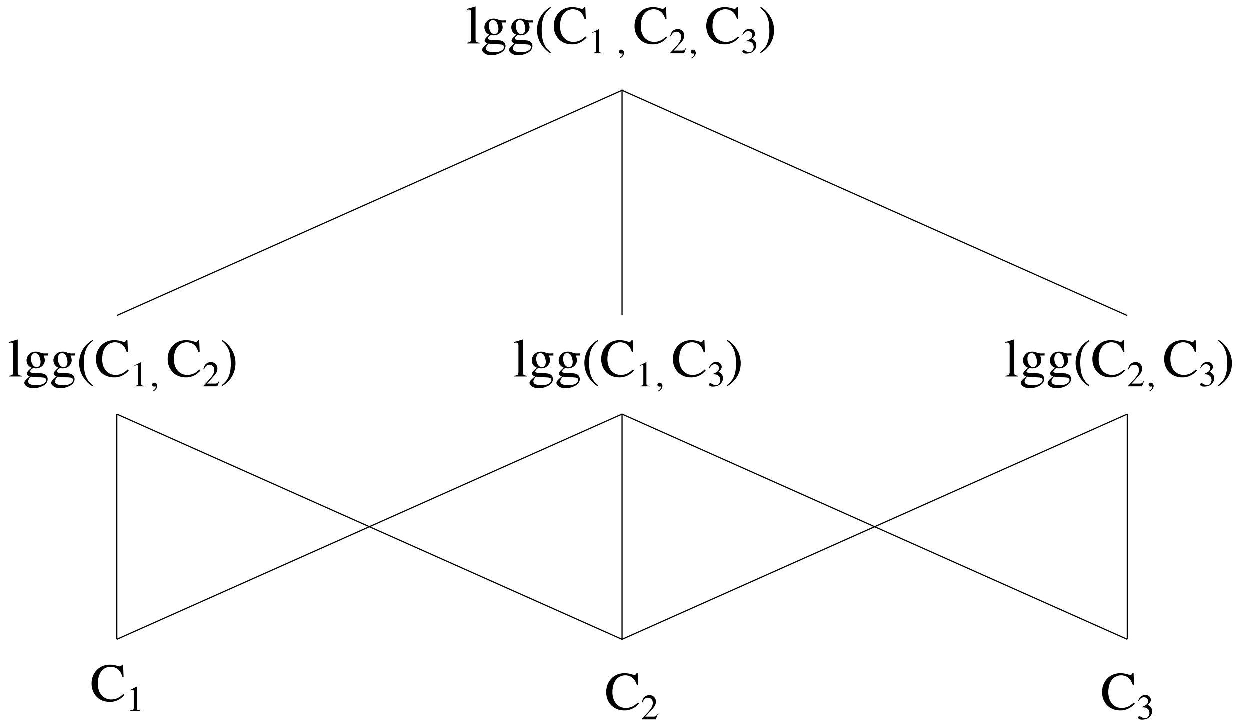
results in an LGG:

q(X) :– p(X, Y) , r(Y, Z) , r(h(U), Z) , r(Y, V) , r(h(U), V)

with inverse substitutions:

{X/(g(a), x), Y/(h(b), y), Z/(c, z), U/(b, w), V/(e, z)}

# LGG of sets of clauses



# Background Knowledge

- Background knowledge can assist learning
- It must be possible to interpret a concept description as a recognition procedure.
- If the description of chair has been learned, then it should be possible to refer to chair in other concept descriptions.
- E.g. the chair “program” will recognise the chairs in an office scene.

# Saturation

Given a set of clauses, the body of one of which is completely contained in the bodies of the others, such as:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E$$

$$Y \leftarrow A \wedge B \wedge C$$

we can *saturate* the first clause:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E \wedge Y$$

# Saturation Example

Suppose we are given two instances of a concept cuddly_pet,

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge dog(X$$

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge cat(X)$$

and:

$$pet(X) \leftarrow dog(X)$$

$$pet(X) \leftarrow cat(X)$$

Saturated clauses are:

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge dog(X) \wedge pet(X)$$

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge cat(X) \wedge pet(X)$$

LGG is

$$cuddly\_pet(X) \leftarrow fluffy(X) \wedge pet(X)$$

# Relative Least General Generalisation (RLGG)

- Apply background knowledge to *saturate* example clauses.
- Find LGG of saturated clauses

```
heavier(A, B) :– denser(A, B), larger(A, B).
```

```
fall_together(hammer, feather) :-
    same_height(hammer, feather),
    denser(hammer, feather),
    larger(hammer, feather).
```

```
fall_together(hammer, feather) :-
    same_height(hammer, feather),
    denser(hammer, feather),
    larger(hammer, feather),
    heavier(hammer, feather).
```

# GOLEM

- LGG is very inefficient for large numbers of examples
- GOLEM uses a *hill-climbing* as an approximation
  - Randomly select pairs of examples
  - Find LGG's and pick the one that covers most positive examples and excludes all negative examples, call it *S.*
    - Randomly select another set of examples
    - Find all LGG's with S
    - Pick best one
    - Repeat as long as cover of positive examples increases.

# Inverting Resolution

- Resolution provides an efficient  means of deriving a solution to a problem, giving a set of axioms which define the task environment.
- Resolution takes two terms and resolves them into a most general unifier.
- Anti-unification finds the ***least general generalisation*** of two terms.

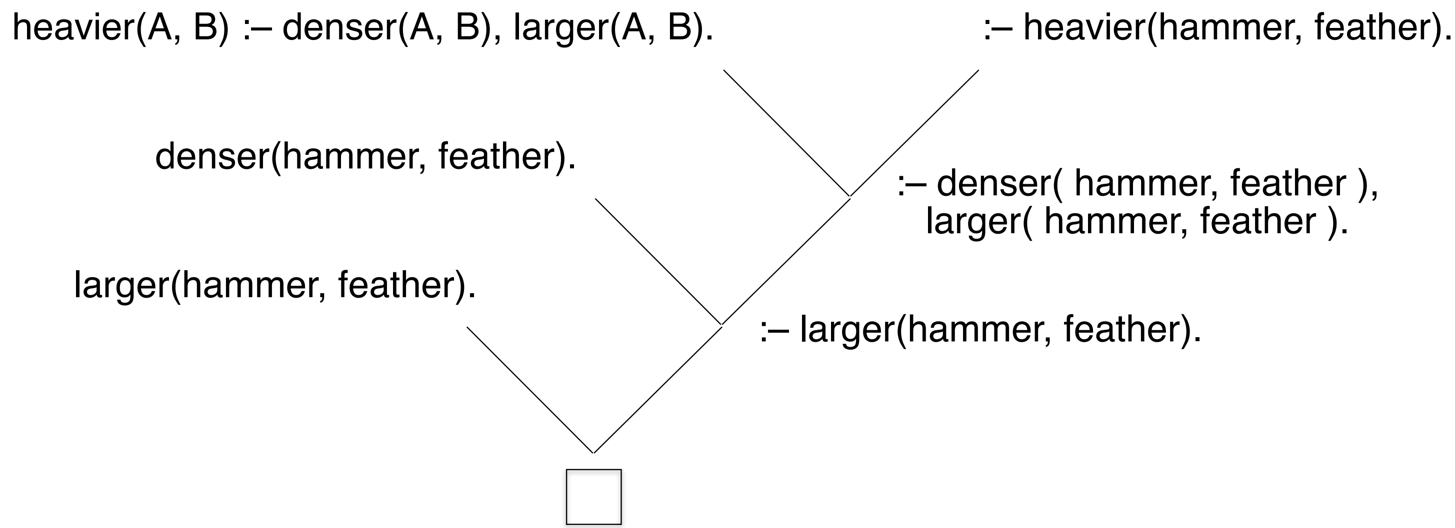
# Resolution Proofs

larger(hammer, feather).

denser(hammer, feather).

heavier(A, B) :– denser(A, B), larger(A, B).

heavier(hammer, feather)?



# Absorption

Given a set of clauses, the body of one of which is completely contained in the bodies of the others, such as:

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E$$

$$Y \leftarrow A \wedge B \wedge C$$

we can hypothesise:

$$X \leftarrow Y \wedge D \wedge E$$

$$Y \leftarrow A \wedge B \wedge C$$

# Intra-construction

This is the distributive law of Boolean equations. Intra-construction takes a group of rules all having the same head, such as:

$$X \leftarrow B \wedge C \wedge D \wedge E$$

$$X \leftarrow A \wedge B \wedge D \wedge F$$

and replaces them with:

$$X \leftarrow B \wedge D \wedge Z$$

$$Z \leftarrow C \wedge E$$

$$Z \leftarrow A \wedge F$$

Intra-construction automatically creates a new term in its attempt to simplify descriptions.

# Automatic Programming

```
member(blue, [blue]).
member(eye, [eye, nose, throat]).
```

**Is** `member(A, [A|B])` **always true?** y

**Is** `member(A, [B|C])` **always true?** n

```
member(2,[1,2,3,4,5,6]).
```

**Is** `member(A,[B,A|C])` **always true?** y

**Is** `member(A,[B|C]) :- member(A,C)` **always true?** y

**Generalisation:**

```
    member(A, [A|B]).
    member(A, [B|C]) :- member(A, C).
```

# Problems with Incremental Learning

- Experiments can never validate a world model.
- Experiments usually involve noisy data, they can cause damage to the environment, they may cause misleading side-effects.
- A robot may have an incomplete theory and incorrect model.
- Need to be able to handle exceptions.
- Need to be able to repair knowledge base.
- If concepts are represented by Horn clauses, we can use a program debugger (declarative diagnosis).

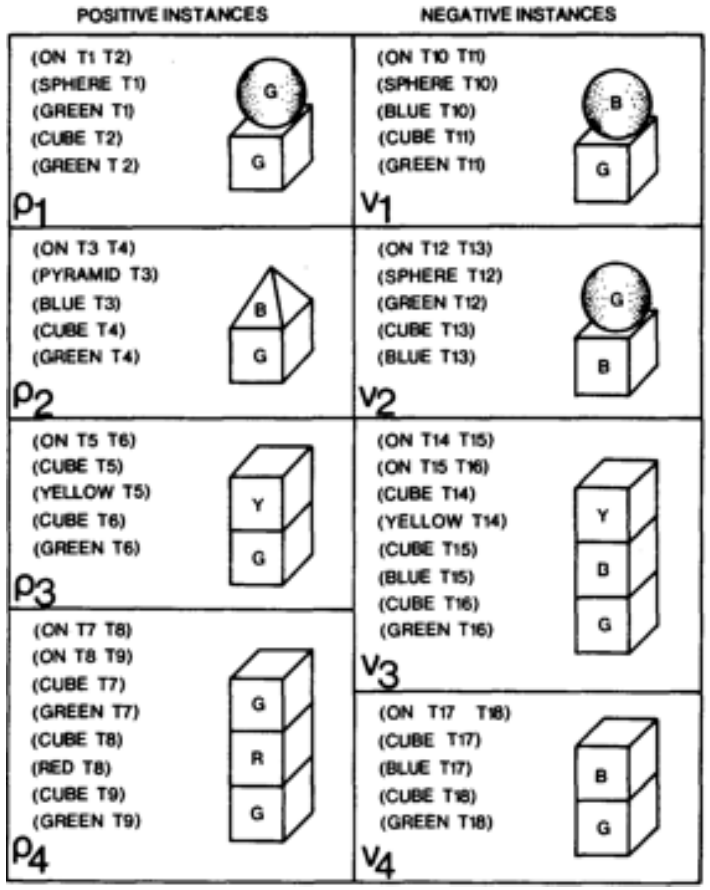
# Repairing Theories

Set the theory *T* to { }
<u>repeat</u>
    Examine the next example
    <u>repeat</u>
        <u>while</u> the theory *T* is too general (covers -ve example) <u>do</u>
            Specialise *T*
        <u>while</u> the theory is too specific (doesn't cover +ve example) <u>do</u>
            Generalise *T*
    <u>until</u> the conjecture *T* is neither too general nor too
        specific with respect to the known facts
    Output *T*
<u>forever</u>

# Exceptions

Multi-level Counterfactuals

- Form a cover for +ve examples
- If -ve examples are also covered, for a new cover of -ve examples and add it as an exception
- If +ve examples are excluded now, reverse process




1. (ON .X .Y)(GREEN .Y)(CUBE .Y)
2. (ON .X .Y)(GREEN .Y)(CUBE .Y)~((BLUE .X) ~(PYRAMID .X))

# Exceptions or Noise?

- If there is noise, then exceptions will start to track noise, causing, "over-fitting".
- Must have a stopping criterion that prevents clause from growing too much.
- Some -ve examples may still be covered and some +ve examples may not.
- Use Minimum Description Length heuristic.

# Minimum Description Length

- Devise an encoding that maps a theory (set of clauses) into a bit string.
- Also need an encoding for examples.
- Number of bits required to encode theory should not exceed number of bits to encode +ve examples.

# Compaction

- Use a measure of compaction to guide search.
- More than one compaction operator applicable at any time.
- A measure is applied to each rule to determine which one will result in the greatest compaction.
- The measure of compaction is the reduction in the number of symbols in the set of clauses after applying an operator.
- Each operator has an associated formula for computing this reduction.
- Best-first search.

# Summary

- If a concept can be represented by sentences in a description language, concepts can be learned by generalising sentences in language
- Machine Learning as search through the space of possible sentences for the most compact that best covers +ve examples and excludes –ve examples
  - Least general generalisation
  - Inverse resolution
  - Automatic Programming

# References


G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1970.

G. D. Plotkin. A further note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 6*. Elsevier, New York, 1971.

C. A. Sammut and R. B. Banerji. Learning concepts by asking questions. In R. S. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, Vol 2, pages 167–192. Morgan Kaufmann, Los Altos, California, 1986.

S. Muggleton. Inductive logic programming. *New Generation Computing*, 8:295–318, 1991.

S. Muggleton and C. Feng. Efficient induction of logic programs. In *First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Omsha.

S. Muggleton, W.-Z. Dai, C. Sammut, A. Tamaddoni-Nezhad, J. Wen, and Z.-H. Zhou. Meta-interpretive learning from noisy images. *Machine Learning*, 107(7):1097 – 1118, 2018.