INDUCTIVE LOGIC PROGRAMMING

COMP34341 Robot Software Architectures

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).

Subsumption

The method of least general generalisations is based on the idea of *subsumption*.

A clause C1 subsumes, or is more general than, another clause C2 if there is a substitution σ such that $C2 \supseteq C1\sigma$.

The least general generalisation of

$$p(g(a), a) \tag{4}$$

and

$$p(g(b), b) \tag{5}$$

is

$$p(g(X), X). \tag{6}$$

Under the substitution {a/X} (6) is equivalent to (4).

Under the substitution {b/X} (6) is equivalent to (5).

Inverse Substitution

The least general generalisation of

$$p(g(a), a)$$

and $\quad p(g(b), b)$

is $\quad p(g(X), X).$

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

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).

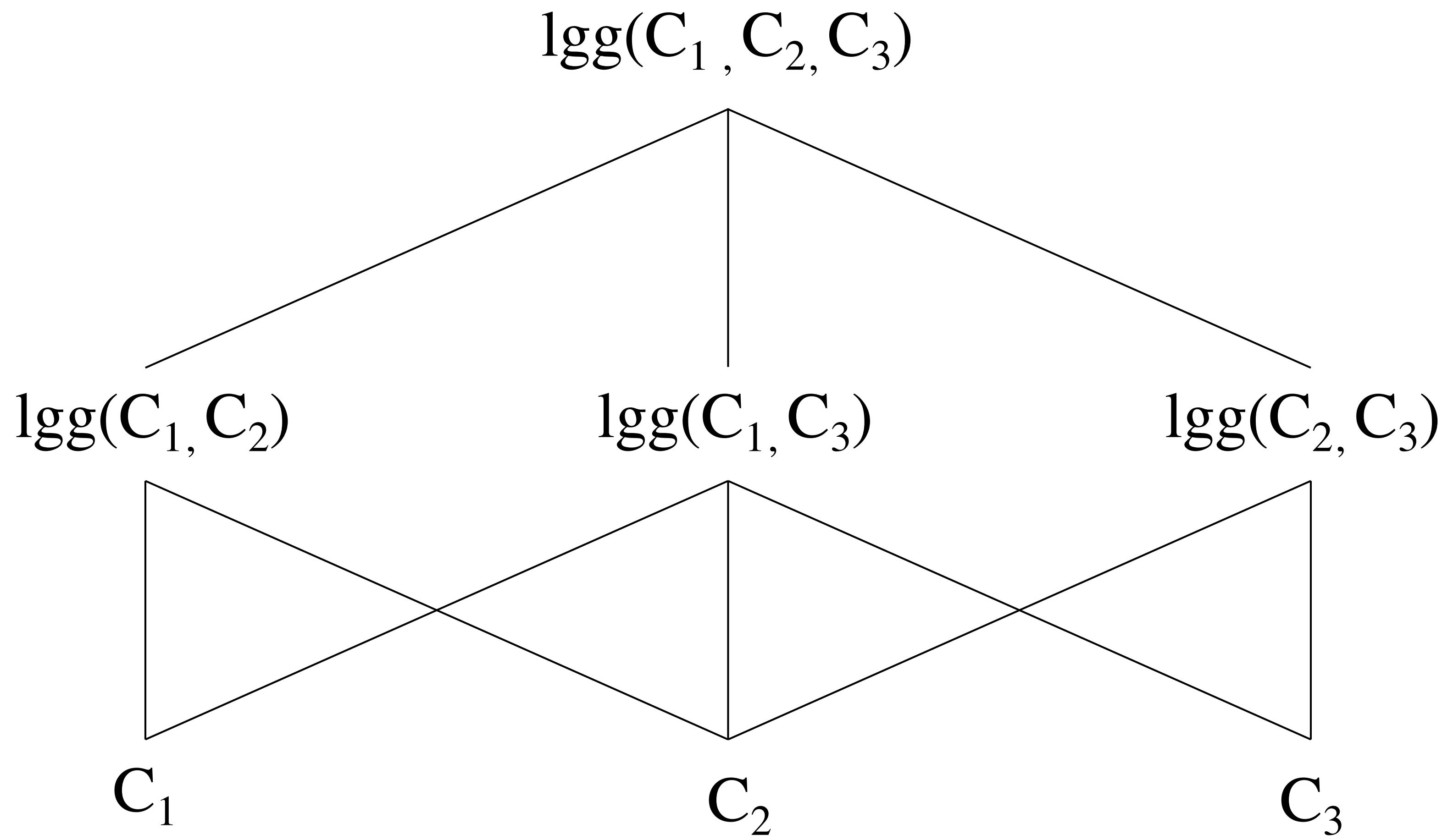
results in an LGG:

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



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).
```

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.

Horn Clauses
Recognising Patterns

Suppose we have a set of clauses:

$$C1 \leftarrow P11 \wedge P12 \tag{1}$$

$$C2 \leftarrow P21 \wedge P22 \wedge C1 \tag{2}$$

and an instance:

$$P11 \wedge P12 \wedge P21 \wedge P22 \tag{3}$$

Clause (1) recognises the first two terms in expression (3) reducing it to

$$P21 \wedge P22 \wedge C1$$

Clause (2) reduces this to C2.

I.e. clauses (1) and (2) recognise expression (3) as the description of an instance of concept C2.

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.

Generalised Subsumption

Simple subsumption is unable to take advantage of background information which may assist generalisation.

Suppose we are given two instances of a concept cuddly_pet,

$$\text{cuddly\_pet}(X) \leftarrow \text{fluffy}(X) \wedge \text{dog}(X) \tag{7}$$

$$\text{cuddly\_pet}(X) \leftarrow \text{fluffy}(X) \wedge \text{cat}(X) \tag{8}$$

Suppose we also know the following:

$$\text{pet}(X) \leftarrow \text{dog}(X) \tag{9}$$

$$\text{pet}(X) \leftarrow \text{cat}(X) \tag{10}$$

Limitations of Subsumption

According to subsumption, the least general generalisation of (7) and (8) is:

$$\text{cuddly\_pet}(X) \leftarrow \text{fluffy}(X) \tag{11}$$

This is an over-generalisation.

A better one is:

$$\text{cuddly\_pet}(X) \leftarrow \text{fluffy}(X) \wedge \text{pet}(X) \tag{12}$$

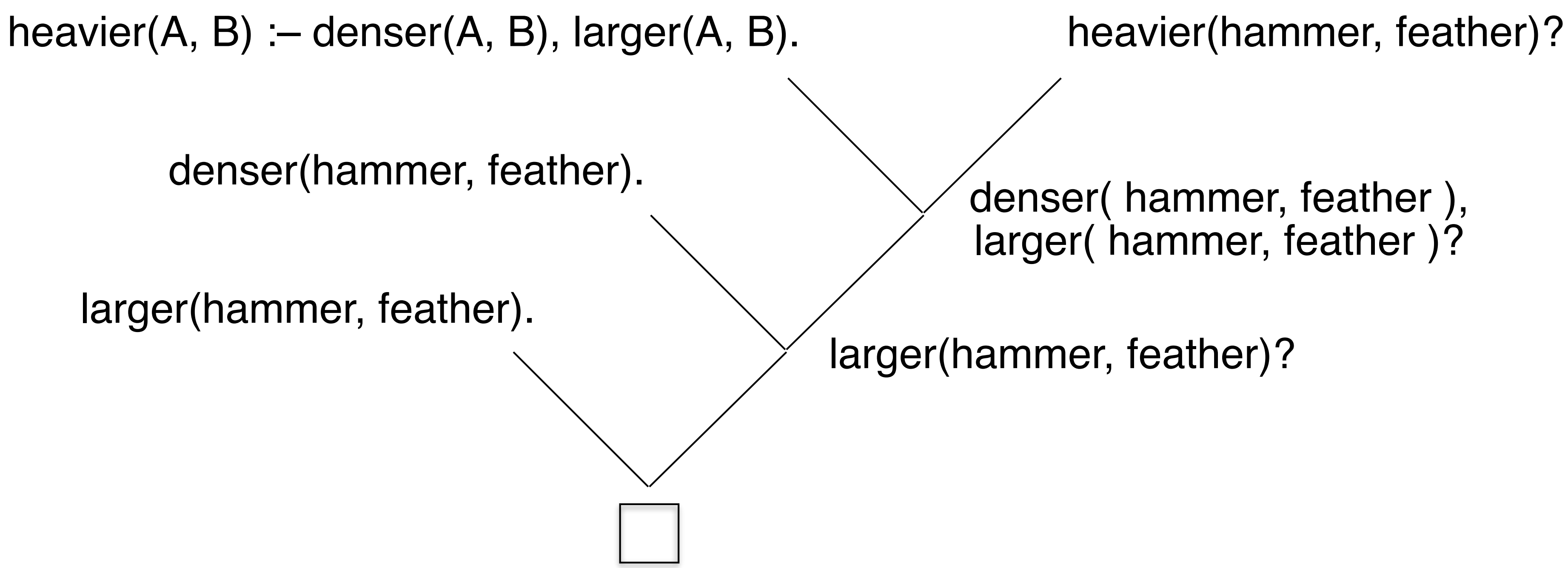
Resolution Proofs

larger(hammer, feather).

denser(hammer, feather).

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

heavier(hammer, feather)?



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.

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:

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

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

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

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

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.

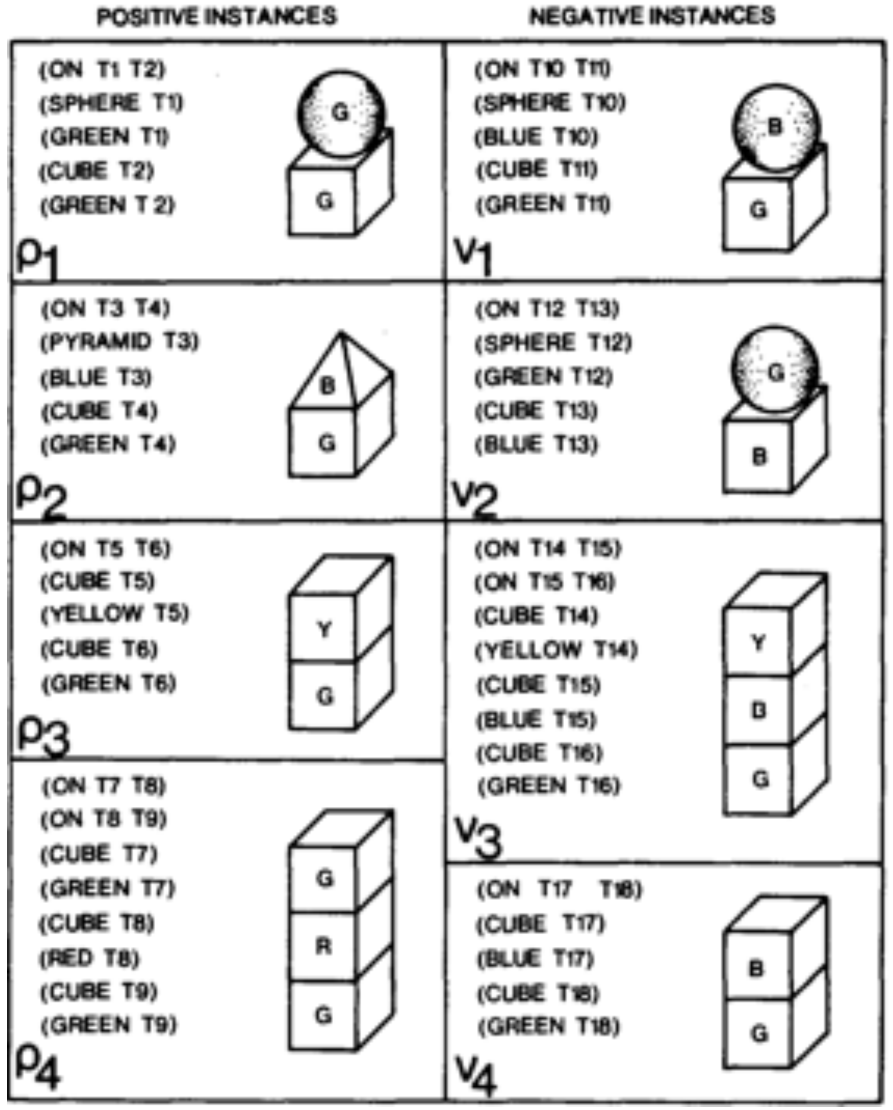
Problems with Incremental Learning

- 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).

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

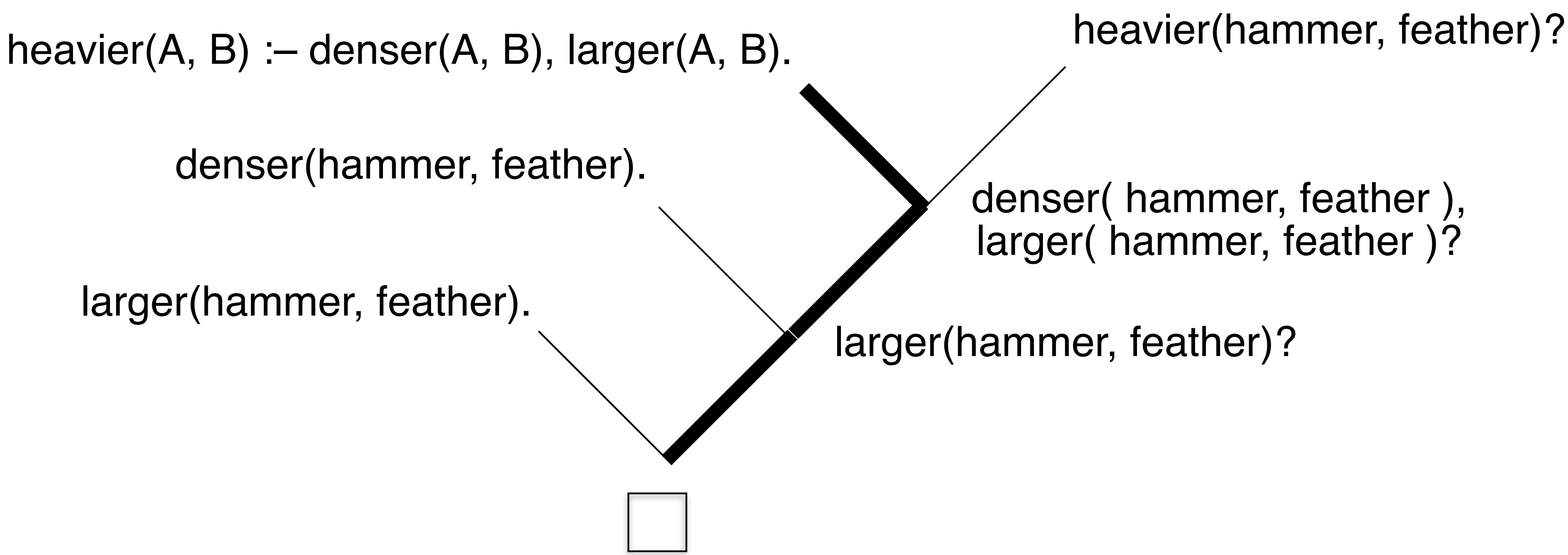
- Use a measure of compaction to guide search.
- More than one compaction operator applicable at any time.
- A measure is applied to each rule to determine which one will result in the greatest compaction.
- The measure of compaction is the reduction in the number of symbols in the set of clauses after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

Repairing Theories: MIS

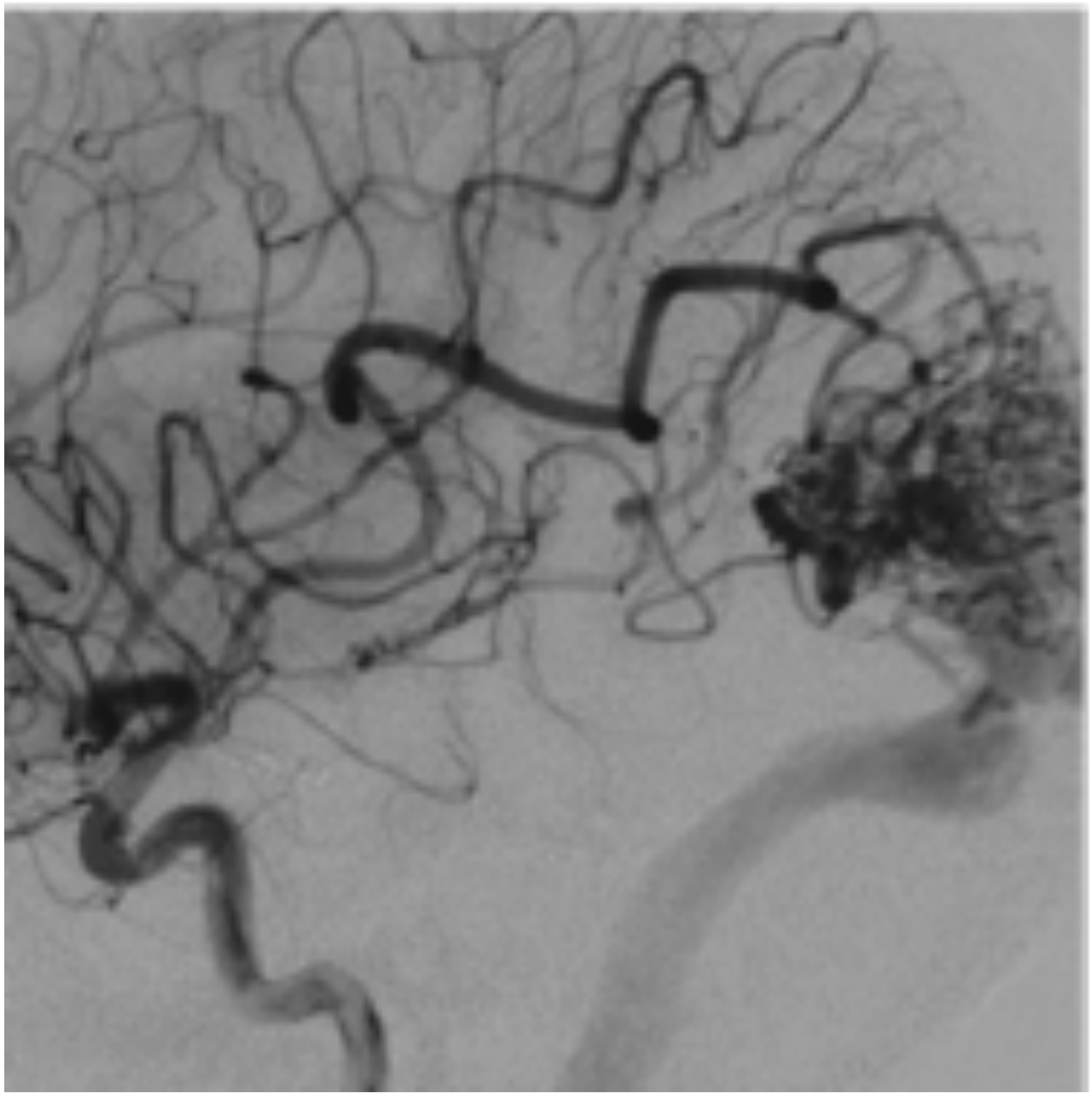
```
Set the theory T to { }
repeat
        Examine the next example
        repeat
                while the theory T is too general do
                        Specialise it by applying
                        contradiction backtracing and remove
                        from T  the refuted hypothesis
                while the theory is too specific do
                        Generalise it by adding to T
                        refinements of previously refuted
                        hypotheses
        until the conjecture T is neither too general nor too
                specific with respect to the known facts
        Output T
forever
```

Contradiction Backtracing

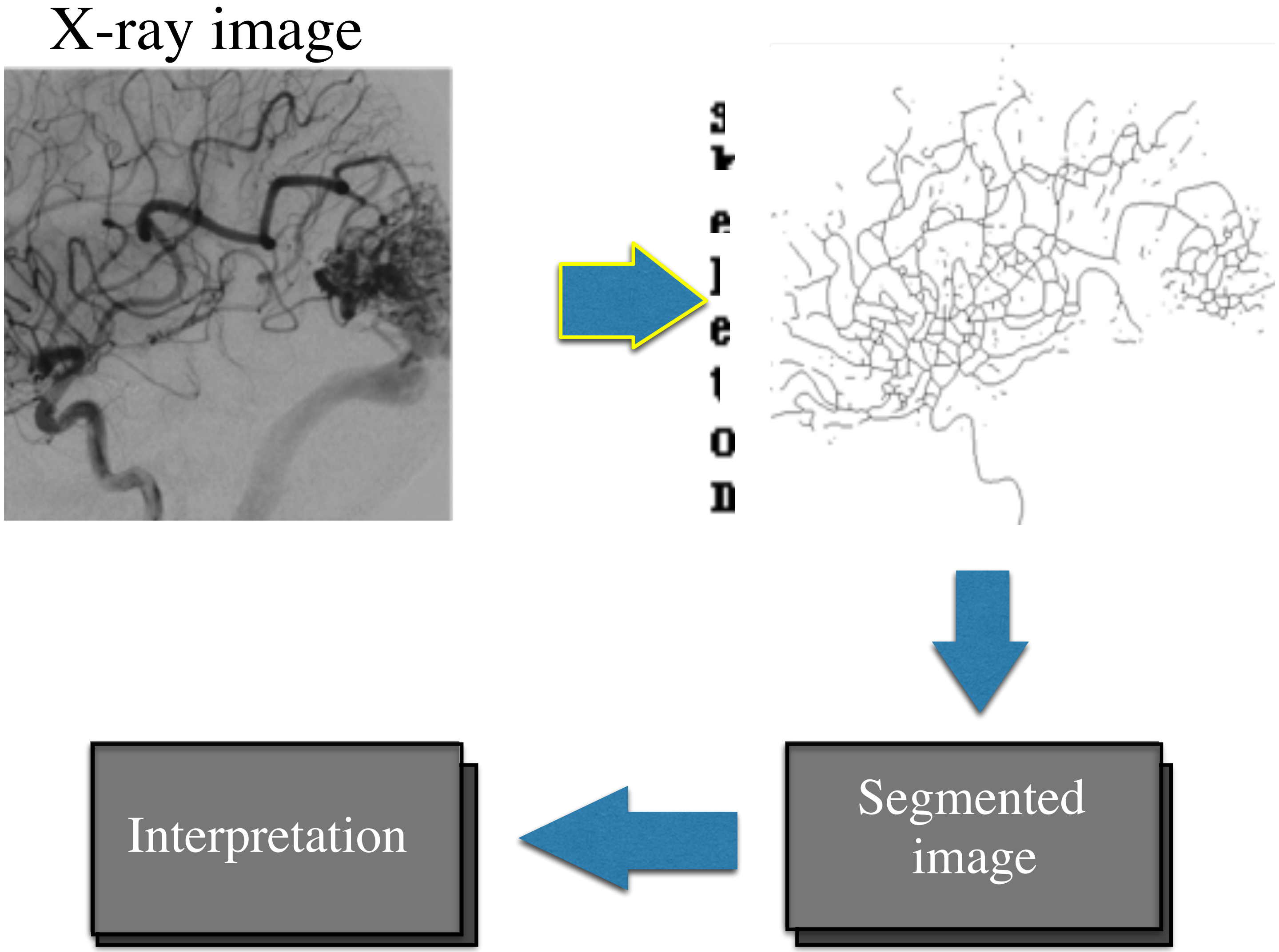
- If a clause is too general, it may recognise instances that it should not.
- Backtracing retreats along proof tree, testing each clause to determine if it is in error.



What is this?



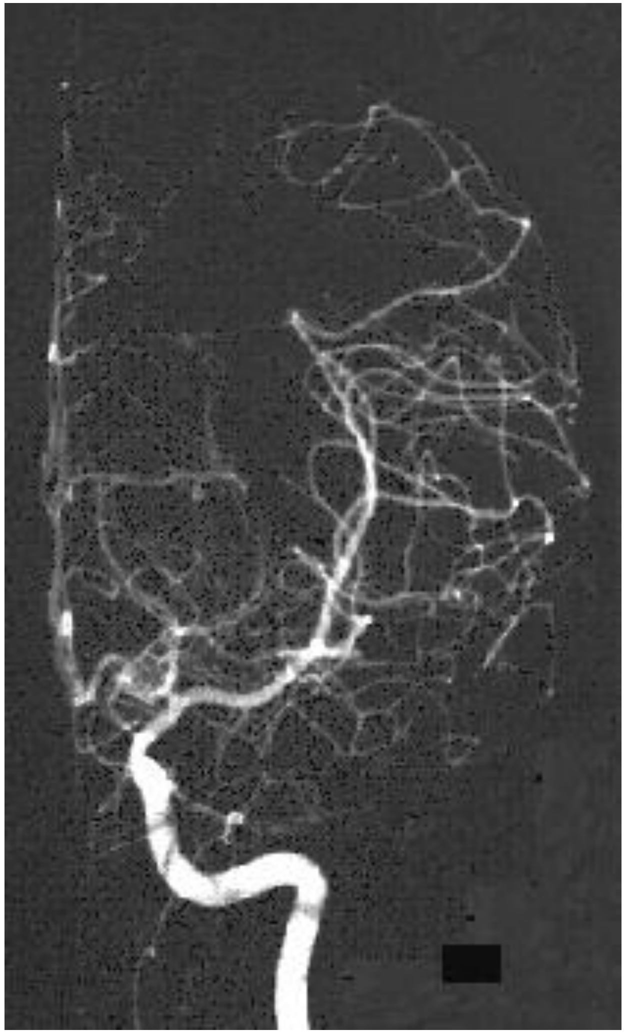
X-ray Angiography



Interpreting Images

- Grey-scale x-ray image thresholded to obtain a black-and-white image.
- Black-and-white image skeletonised to reduce thick vessels to lines only a single pixel wide.
- Skeleton traced to join pixels into segments of blood vessels.
- Segmented skeleton used to guide further processing of grey-scale image to obtain diameters and intensity values of each blood vessel segment.

Stages of processing



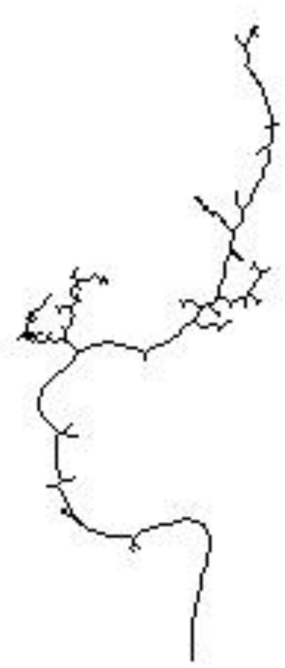
Original X-ray

After thresholding

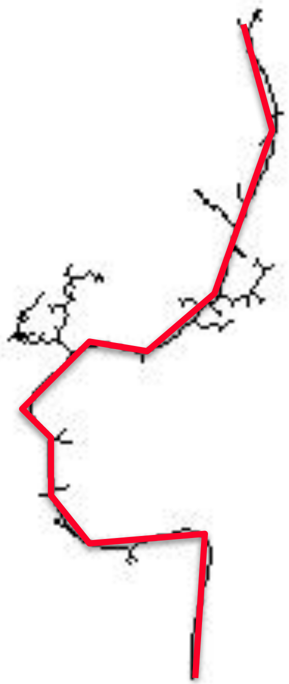
After Thinning

Stages of Processing

After Tracing



Polygonal Approximation



Learning

- Much knowledge exists in text books and anatomical atlases.
- But, there is an enormous range of variations in human anatomy.
- Learning can capture variations.
- Only have small number of labelled examples.
- Example require complex relational descriptions.

Output of low-level processing

```
internal_carotid_artery(mb1, 1).
segment(1, mb1, n, 40, 130, [2]).
segment(2, mb1, w, 40, 144, [3]).
segment(3, mb1, nw, 35, 135, [4, 5]).
segment(4, mb1, n, 40, 50, [6, 7]).
segment(6, mb1, ne, 20, 170, [8, 9]).
segment(5, mb1b1, e, 10, 100, []).
segment(7, mb1b2, w, 5, 125, []).
segment(8, mb1b3, e, 18, 90, []).
segment(9, mb1b4, n, 15, 100, []).
```

Background Knowledge

- Need to augment raw data with background knowledge about
  - turns
  - branches
  - Intensities

Saturated Example

```
internal_carotid_artery(mb1, mb1_1) :-
        segment(mb1_1, mb1, n, 40, 130, [mb1_2]),
        segment(mb1_2, mb1, w, 40, 144, [mb1_3]),
        segment(mb1_3, mb1, nw, 35, 135, [mb1_4, mb1_5]),
        segment(mb1_4, mb1, n, 40, 50, [mb1_6, mb1_7]),
        segment(mb1_5, mb1b1, e, 10, 100, []),
        segment(mb1_6, mb1, ne, 20, 170, [mb1_8, mb1_9]),
        segment(mb1_7, mb1b2, w, 5, 125, []),
        segment(mb1_8, mb1b3, e, 18, 90, []),
        segment(mb1_9, mb1b4, n, 15, 100, []),
        max(diameter, mb1, mb1_4, 40),
        max(intensity, mb1, mb1_6, 170),
        min(diameter, mb1, mb1_6, 20),
        min(intensity, mb1, mb1_4, 50),
        left_turn(mb1, mb1_1, mb1_2),
        right_turn(mb1, mb1_2, mb1_3),
        right_turn(mb1, mb1_3, mb1_4),
        right_turn(mb1, mb1_4, mb1_6),
        left_branch(mb1, mb1_4, mb1_7),
        left_branch(mb1, mb1_6, mb1_9),
        right_branch(mb1, mb1_3, mb1_5),
        right_branch(mb1, mb1_6, mb1_8),
        left_turns(mb1, 1),
        right_turns(mb1, 3),
        left_branches(mb1, 2),
        right_branches(mb1, 2).
```

A Small Sample

- 10 X-ray images of anterior-posterior view of Internal Carotid Artery
- 11 negative examples from images of other vessels and other views.
- Including background knowledge to recognise
  - left turns and right turns in blood vessel
  - left and right branches to other blood vessels
  - number of left and right turns
  - number of left and right branches

Least General Generalisation

```
internal_carotid_artery(_0, _1) :-
        segment(_1, _0, n, _2, _3, [_4 | _5]),
        left_turns(_0, _6),
        right_turns(_0, _7),
        left_branches(_0, 2),
        right_branches(_0, 2).
```