Symbol-based Machine Learning

- Learning is a major topic in AI and we explore three types of learning: symbol-based, connectionist, and genetic (or evolutionary)
- Herbert Simon once defined learning as:

any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population (Simon, 1983).

- This chapter concentrates on inductive learning, such as
  - Concept learning using data-driven generalization
  - Explanation learning
  - Supervised and unsupervised learning

The Data and Goals of the Learning Task

- Concept learning
  - A sequence of both positive and negative examples of a concept are given
  - The learner should generalize the concept so that when a new example is encountered, it can be correctly classified as positive or negative
- Explanation-based learning
  - Attempt to infer a general concept from a single example and some domain knowledge
- Unsupervised learning
  - Start with a set of unclassified examples and try to discover possible categorizations

Representation and Operations

- Representation of learned knowledge
  - From instances of particular balls (as conjuncts)

    size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)
    size(obj2, large) ∧ color(obj2, red) ∧ shape(obj2, round)

    and the general concept "ball" would be defined by:

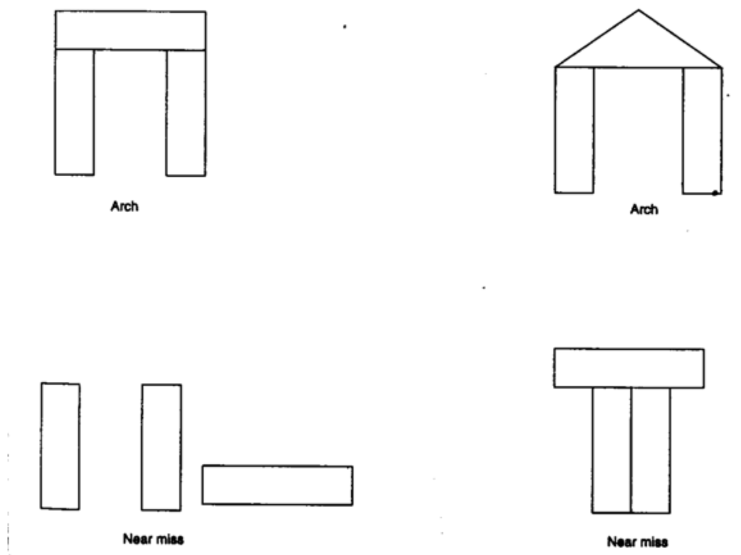
    size(X, Y) ∧ color(X, Z) ∧ shape(X, round)
- Operations
  - Usually the learner is trying to construct a generalization, heuristic rule or a plan
  - Operations include generalization or specialization, adjusting weights, or modifying the representation
  - As seen above, generalization may involve replacing constant values with variables

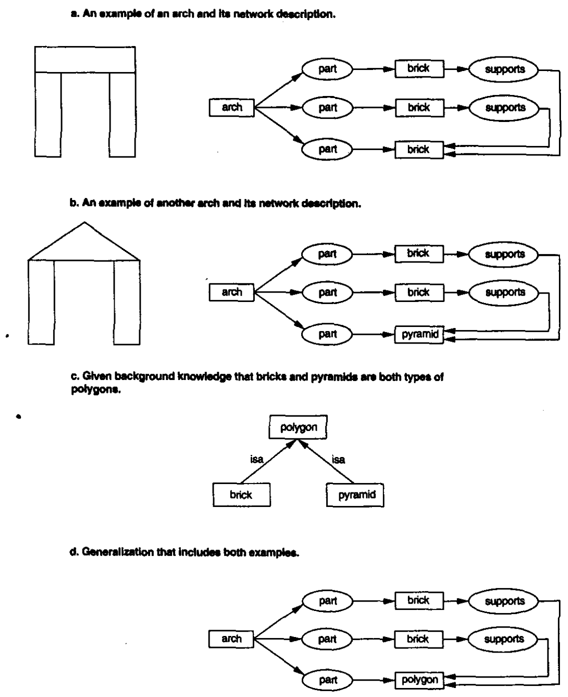
The Concept Space and Heuristic Search

- Concept space
  - Based on representation language and operations
  - The complexity of the concept space indicates the difficulty of a learning problem
- Heuristic search
  - The learner must search the concept space to find the desired concept
  - Available training data and heuristics can guide the search
  - For example, in generalization, values that are allowed to change and you still have a positive instance of the concept are candidate for substituting variables in place of values

Winston's Program to Learn about Arches

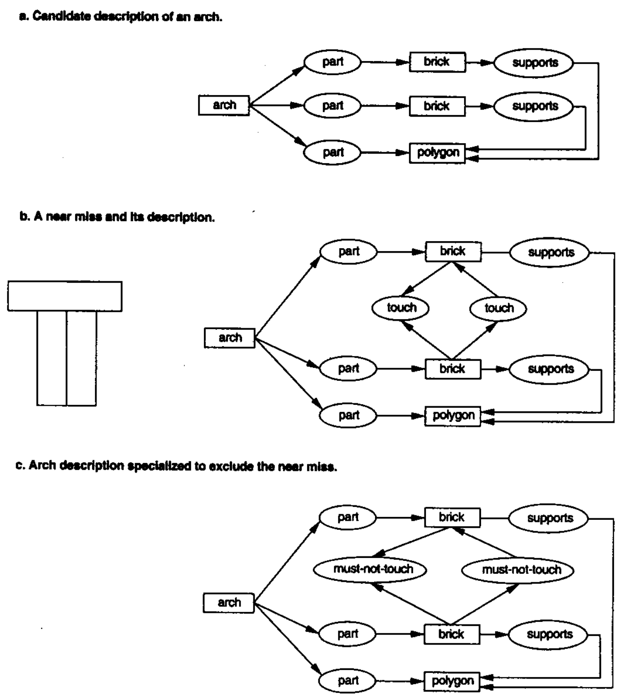
- Here are examples of arches and near misses




a. An example of an arch and its network description.
arch
part
part
part
brick
brick
brick
supports
supports
b. An example of another arch and its network description.
arch
part
part
part
brick
brick
pyramid
supports
supports
c. Given background knowledge that bricks and pyramids are both types of polygons.
polygon
isa
isa
brick
pyramid
d. Generalization that includes both examples.
arch
part
part
part
brick
brick
polygon
supports
supports

Generalization of the Lintel

Specialization to Eliminate a Near Miss



Summary of Winston's Arches Program

- Characteristics
  - The program is based on a hill climbing search using a sequence of examples
  - There is no backtracking, so it is very sensitive to the ordering of the examples
  - The quality of training examples is also important
- Important contributions
  - The operations of generalization and specialization
  - Use of data to guide the search through the concept space
  - The importance of training examples

Version Space Search

- Inductive learning in a concept space
  - Generalization operations impose an ordering on the concepts in the space
  - This ordering can be used to guide the search
- Replacing constants with variables

color(ball, red)

generalizes to

color(X, red)

- Dropping conditions from a conjunction

shape(X, round) $\wedge$ size(X, small) $\wedge$ color(X, red)

generalizes to

shape(X, round) $\wedge$ color(X, red)

More Generalization

- Adding a disjunct to an expression

shape(X, round) $\wedge$ size(X, small) $\wedge$ color(X, red)

generalizes to

shape(X, round) $\wedge$ size(X, small) $\wedge$ (color(X, red) $\vee$ color(X, blue))

- Replacing a property with its parent in a class hierarchy

color(X, red)

generalizes to

color(X, primary_color)

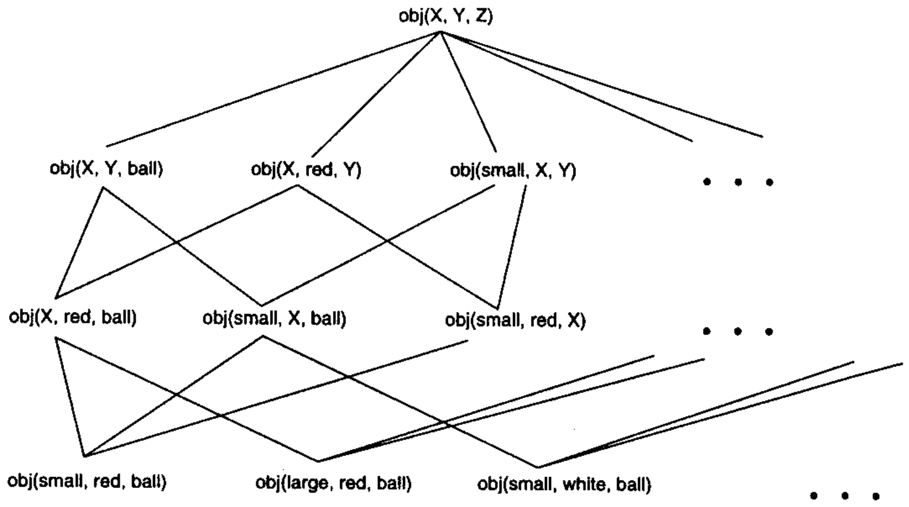
A Concept Covering Another Concept

- If concept p is more general than concept q, then p covers q
  - Let p(x) and q(x) be descriptions that classify objects as positive examples of a concept, so that p(x) → positive(x) and q(x) → positive(x)
  - p covers q if and only if q(x) → positive(x) is a logical consequence of p(x) → positive(x)
- Examples: color(X, Y) covers color(ball, Z) which in turn covers color(ball, red)
- Given a domain of objects with the following three properties and values

  size = {large, small}
  colors = {red, white, blue}
  shape = {ball, brick, cube}

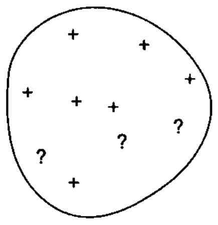
  The figure on the next slide shows the covering relations

A Concept Space

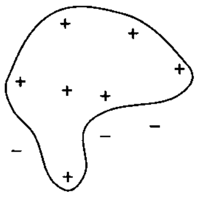


Avoiding Overgeneralization

- It is important to include negative instances of a concept to avoid overgeneralization



Concept induced from
positive examples only



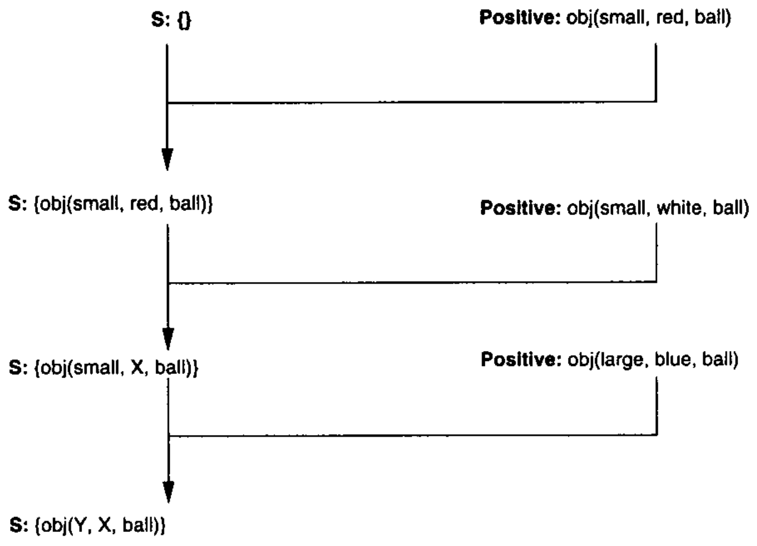
Concept induced from
positive and negative examples

- A concept is maximally specific if it covers all positive examples, no negative examples, and for any other concept c′ that covers the positive examples, c <= c′

Specific to General Search

```
Begin
Initialize S to the first positive training instance;
N is the set of all negative instances seen so far;

For each positive instance p
    Begin
    For every s ∈ S, if s does not match p, replace s with its most specfic
        generalizations that match p;
    Delete from S all hypotheses more general than some other hypothesis in S;
    Delete from S all hypotheses that match a previously observed negative
        instance in N;
    End;

For every negative instance n
    Begin
    Delete all members of S that match n;
    Add n to N to check future hypotheses for overgeneralization;
    End;
End
```

Specific to General Search for a "ball"

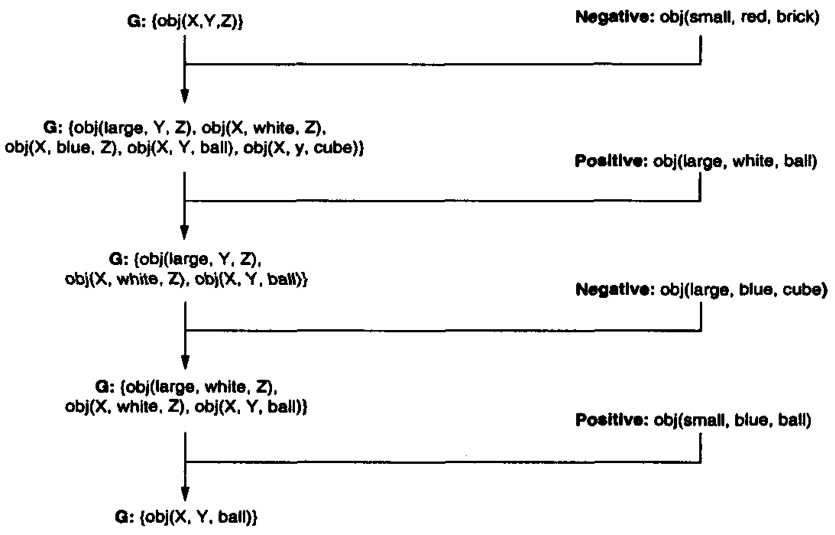

S: {}
Positive: obj(small, red, ball)
S: {obj(small, red, ball)}
Positive: obj(small, white, ball)
S: {obj(small, X, ball)}
Positive: obj(large, blue, ball)
S: {obj(Y, X, ball)}

Algorithm to find Maximally General Concepts

- A concept is maximally general if it covers all positive examples, no negative examples, and for any other concept c′ that covers the positive examples, c >= c′

```
Begin
Initialize G to contain the most general concept in the space;
P contains all positive examples seen so far;

For each negative instance n
    Begin
    For each g ∈ G that matches n, replace g with its most general specializations
        that do not match n;
    Delete from G all hypotheses more specific than some other hypothesis in G;
    Delete from G all hypotheses that fail to match some positive example in P;
    End;

For each positive instance p
    Begin
    Delete from G all hypotheses that fail to match p;
    Add p to P;
    End;
End
```

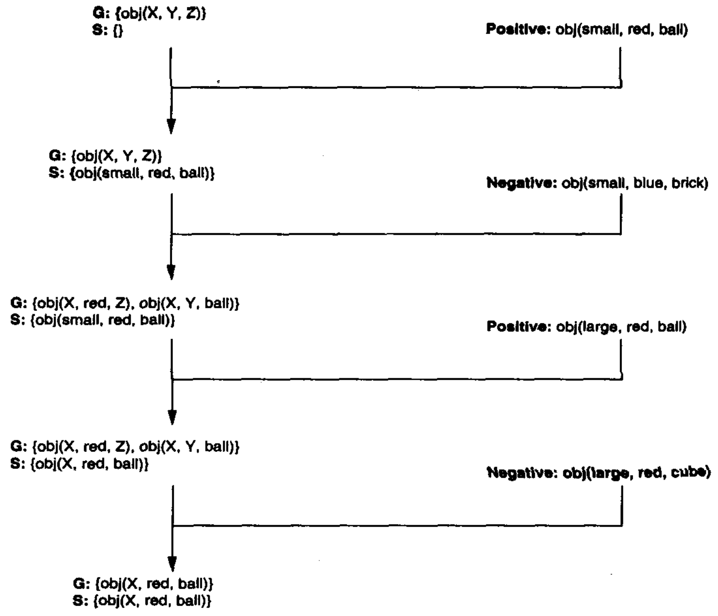
General to Specific Search for a "ball"


G: {obj(X,Y,Z)}
Negative: obj(small, red, brick)
G: {obj(large, Y, Z), obj(X, white, Z), obj(X, blue, Z), obj(X, Y, ball), obj(X, y, cube)}
Positive: obj(large, white, ball)
G: {obj(large, Y, Z), obj(X, white, Z), obj(X, Y, ball)}
Negative: obj(large, blue, cube)
G: {obj(large, white, Z), obj(X, white, Z), obj(X, Y, ball)}
Positive: obj(small, blue, ball)
G: {obj(X, Y, ball)}

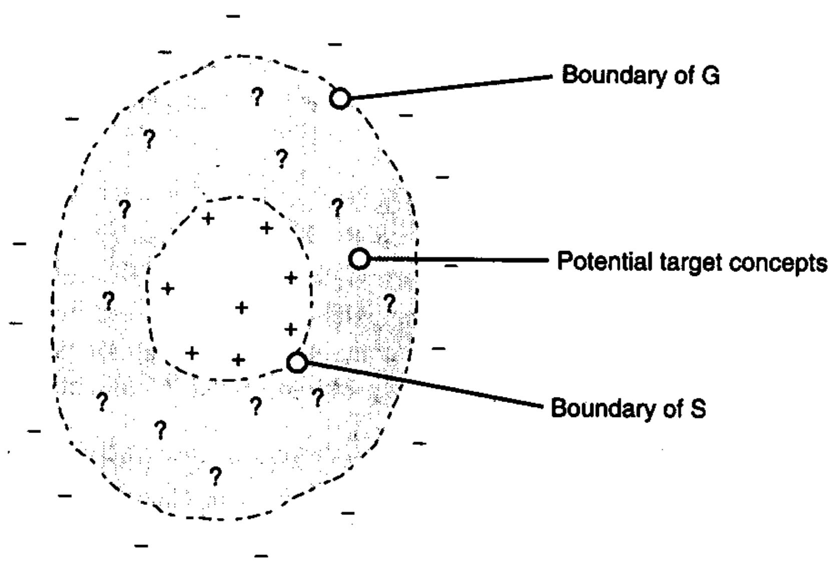
Candidate Elimination Algorithm

```
Begin
Initialize G to be the most general concept in the space;
Initialize S to the first positive training instance;

For each new positive instance p
    Begin
    Delete all members of G that fail to match p;
    For every s ∈ S, if s does not match p, replace s with its most specific
        generalizations that match p;
    Delete from S any hypothesis more general than some other hypothesis in S;
    Delete from S any hypothesis not more specific than some hypothesis in G;
    End;

For each new negative instance n
    Begin
    Delete all members of S that match n;
    For each g ∈ G that matches n, replace g with its most general specializations
        that do not match n;
    Delete from G any hypothesis more specific than some other hypothesis in G;
    Delete from G any hypothesis more specific than some hypothesis in S;
    End;

If G = S and both are singletons, then the algorithm has found a single concept that
    is consistent with all the data and the algorithm halts;
If G and S become empty, then there is no concept that covers all positive instances
    and none of the negative instances;
End
```

Candidate Elimination Algorithm learning a “red ball”


G: {obj(X, Y, Z)}
S: {}
Positive: obj(small, red, ball)
G: {obj(X, Y, Z)}
S: {obj(small, red, ball)}
Negative: obj(small, blue, brick)
G: {obj(X, red, Z), obj(X, Y, ball)}
S: {obj(small, red, ball)}
Positive: obj(large, red, ball)
G: {obj(X, red, Z), obj(X, Y, ball)}
S: {obj(X, red, ball)}
Negative: obj(large, red, cube)
G: {obj(X, red, ball)}
S: {obj(X, red, ball)}

Converging Boundaries of S and G

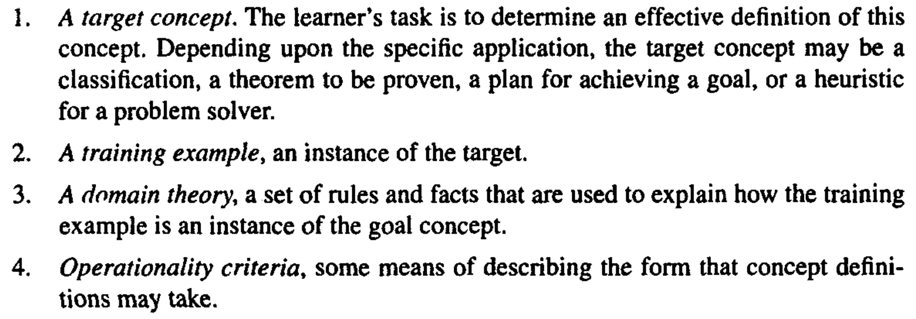


ID3 Decision Tree Induction Algorithm

- We have already discussed decision trees
- If you no longer have the handout, it is available at the instructor's website

Explanation Based Learning

- The basic components



1. *A target concept*. The learner's task is to determine an effective definition of this concept. Depending upon the specific application, the target concept may be a classification, a theorem to be proven, a plan for achieving a goal, or a heuristic for a problem solver.
2. *A training example*, an instance of the target.
3. *A domain theory*, a set of rules and facts that are used to explain how the training example is an instance of the goal concept.
4. *Operationality criteria*, some means of describing the form that concept definitions may take.

- An explanation for a training example is derived from the domain knowledge
- This explanation is then generalized to select relevant features and organize knowledge

A First Example

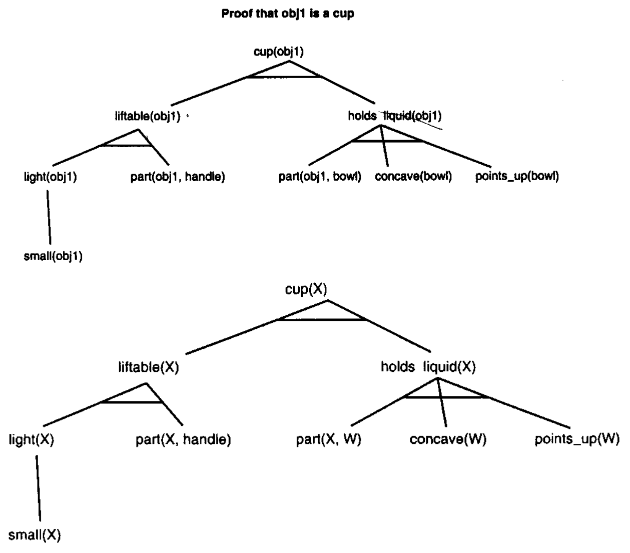
- The target is a rule for a cup premise(X) → cup(X)
- Here is the domain knowledge & the training example

liftable(X) ∧ holds_liquid(X) → cup(X)
part(Z, W) ∧ concave(W) ∧ points_up(W) → holds_liquid(Z)
light(Y) ∧ part(Y, handle) → liftable(Y)
small(A) → light(A)
made_of(A, feathers) → light(A)

cup(obj1)
small(obj1)
part(obj1, handle)
owns(bob, obj1)
part(obj1, bottom)
part(obj1, bowl)
points_up(bowl)
concave(bowl)
color(obj1, red)

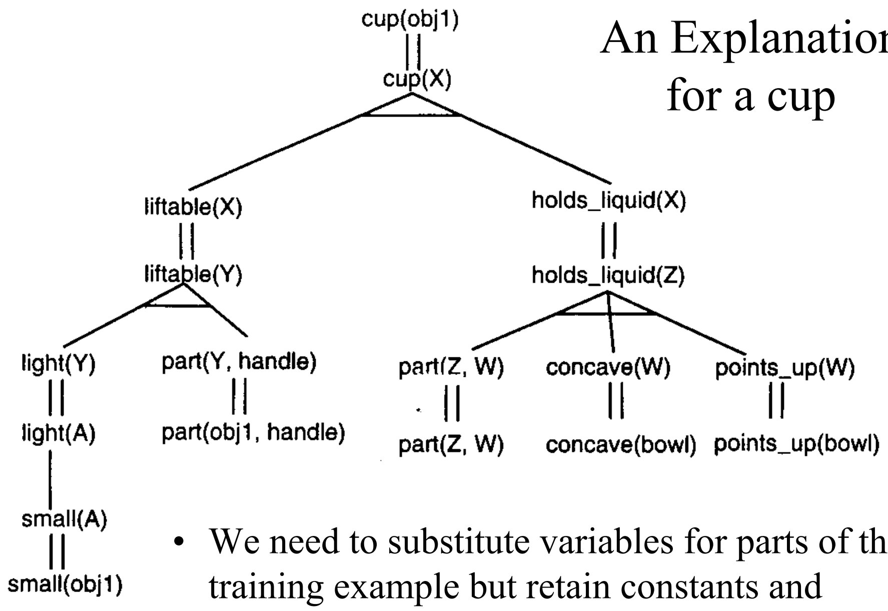
- Notice there is considerable "noise" in the training example
- A specific proof tree is then generalized
- This produces a general rule for an object being a cup

Specific and Generalized Proof



small(X) ∧ part(X, handle) ∧ part(X, W) ∧ concave(W) ∧ points_up(W) → cup(X).

An Explanation for a cup



- We need to substitute variables for parts of the training example but retain constants and constraints from domain theory

Constructing the Substitutions

```
if $e_1$ is in the premise of a domain rule and $e_2$ is the conclusion of a domain rule
then begin
  $T_s$  = the most general unifier of $e_1s_s$ and $e_2s_s$        % unify $e_1$ and $e_2$ under $s_s$
  $s_s$  = $s_sT_s$                                                  % update $s_s$ by composing it with $T_s$
  $T_g$  = the most general unifier of $e_1s_g$ and $e_2s_g$        % unify $e_1$ and $e_2$ under $s_g$
  $s_g$  = $s_gT_g$                                                  % update $s_g$ by composing it with $T_g$
end
if $e_1$ is in the premise of a domain rule and $e_2$ is a fact in the training instance
then begin                                                            % only update $s_s$
  $T_s$  = the most general unifier of $e_1s_s$ and $e_2s_s$        % unify $e_1$ and $e_2$ under $s_s$
  $s_s$  = $s_s T_s$                                                 % update $s_s$ by composing it with $T_s$
end
```

s_s = {obj1/X, obj1/Y, obj1/A, obj1/Z, bowl/W}

s_g = {X/Y, X/A, X/Z}

Benefits of EBL

1. Training examples often contain irrelevant information, such as the color of the cup in the preceding example. The domain theory allows the learner to select the relevant aspects of the training instance.
2. A given example may allow numerous possible generalizations, most of which are either useless, meaningless, or wrong. EBL forms generalizations that are known to be relevant to specific goals and that are guaranteed to be logically consistent with the domain theory.
3. By using domain knowledge EBL allows learning from a single training instance.
4. Construction of an explanation allows the learner to hypothesize unstated relationships between its goals and its experience, such as deducing a definition of a cup based on its structural properties.

Analogical Learning

- Reasoning by analogy tries to apply existing knowledge to new situations
- An example of water flow and current flow
  - Amperage is analogous to the flow of water
  - Voltage is analogous to the water pressure
  - It is known that the capacity of the pipe is critical to water flow, so there must be an analogous property for electricity
  - It turns out resistance is the analogous property

A Framework for Analogical Reasoning

1. *Retrieval.* Given a target problem, it is necessary to select a potential source analog. Problems in analogical retrieval include selecting those features of the target and source that increase the likelihood of retrieving a useful source analog and indexing knowledge according to those features. Generally, retrieval establishes the initial elements of an analogical mapping.
2. *Elaboration.* Once the source has been retrieved, it is often necessary to derive additional features and relations of the source. For example, it may be necessary to develop a specific problem-solving trace (or explanation) in the source domain as a basis for analogy with the target.
3. *Mapping and inference.* This stage involves developing the mapping of source attributes into the target domain. This involves both known similarities and analogical inferences.
4. *Justification.* Here we determine that the mapping is indeed valid. This stage may require modification of the mapping.
5. *Learning.* In this stage the acquired knowledge is stored in a form that will be useful in the future.

An Example

- The source domain

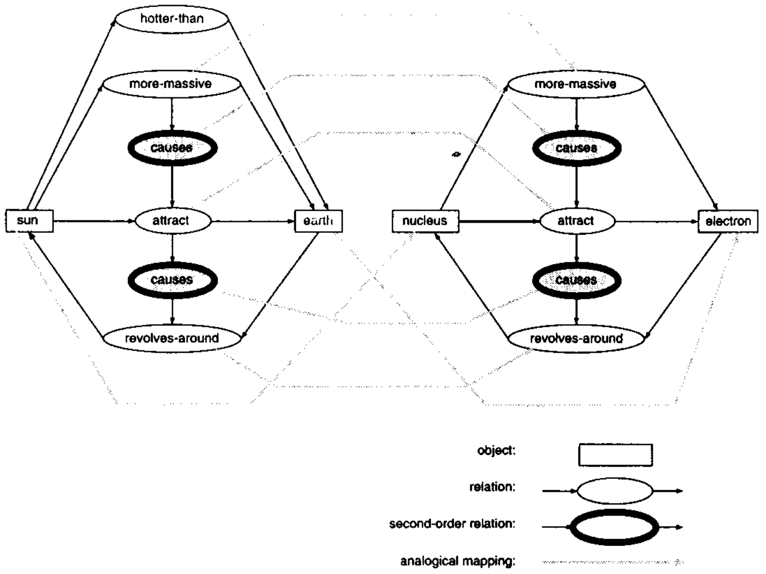
```
yellow(sun)
blue(earth)
hotter-than(sun, earth)
causes(more-massive(sun, earth), attract(sun, earth))
causes(attract(sun, earth), revolves-around(earth, sun))
```

- The goal is to explain

```
more-massive(nucleus, electron)
revolves-around(electron, nucleus)
```

- The mapping attempts to identify relevant factors and drop those factors that are not relevant, such as color and hotter-than above

An Analogical Mapping



Completing the Analogy

- The constraints lead to the mapping

  sun → nucleus
  earth → electron

- Extending the mapping leads to the inferences

  causes(more-massive(nucleus, electron), attract(nucleus, electron))
  causes(attract(nucleus, electron), revolves-around(electron, nucleus))

- There are many similarities between analogical learning presented here and case-based expert systems discussed earlier