10a Machine Learning: Symbol-based

10.0 Introduction

10.1 A Framework for Symbol-based Learning

10.2 Version Space Search

10.3 The ID3 Decision Tree Induction Algorithm

10.4 Inductive Bias and Learnability

10.5 Knowledge and Learning

10.6 Unsupervised Learning

10.7 Reinforcement Learning

10.8 Epilogue and References

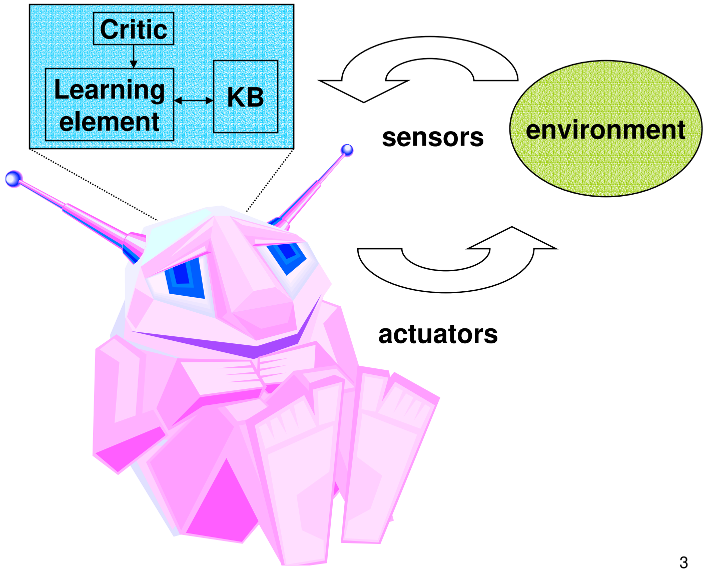
10.9 Exercises

Additional references for the slides:
Jean-Claude Latombe's CS121 slides:
robotics.stanford.edu/~latombe/cs121

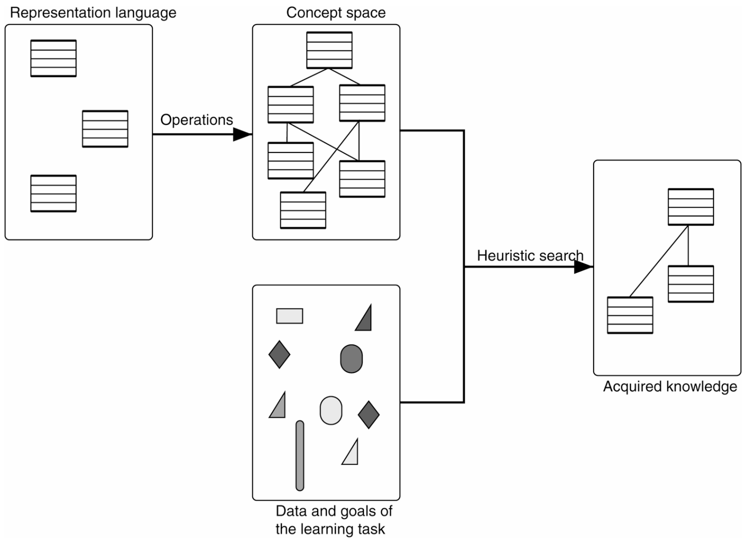
Chapter Objectives

- Learn about several "paradigms" of symbol-based learning
- Learn about the issues in implementing and using learning algorithms
- The agent model: can learn, i.e., can use prior experience to perform better in the future

A learning agent



A general model of the learning process



A learning game with playing cards

I would like to show what a *full house* is. I give you examples which are/are not full houses:

6♦ 6♠ 6♥ 9♣ 9♥	is a full house
6♦ 6♠ 6♥ 6♣ 9♥	is not a full house
3♣ 3♥ 3♣ 6♦ 6♠	is a full house
1♣ 1♥ 1♣ 6♦ 6♠	is a full house
Q♣ Q♥ Q♣ 6♦ 6♠	is a full house
1♦ 2♠ 3♥ 4♣ 5♥	is not a full house
1♦ 1♠ 3♥ 4♣ 5♥	is not a full house
1♦ 1♠ 1♥ 4♣ 5♥	is not a full house
1♦ 1♠ 1♥ 4♣ 4♥	is a full house

A learning game with playing cards

If you haven't guessed already, a *full house* is three of a kind and a pair of another kind.

6♦ 6♠ 6♥ 9♣ 9♥	is a full house
6♦ 6♠ 6♥ 6♣ 9♥	is not a full house
3♣ 3♥ 3♣ 6♦ 6♠	is a full house
1♣ 1♥ 1♣ 6♦ 6♠	is a full house
Q♣ Q♥ Q♣ 6♦ 6♠	is a full house
1♦ 2♠ 3♥ 4♣ 5♥	is not a full house
1♦ 1♠ 3♥ 4♣ 5♥	is not a full house
1♦ 1♠ 1♥ 4♣ 5♥	is not a full house
1♦ 1♠ 1♥ 4♣ 4♥	is a full house

Intuitively,

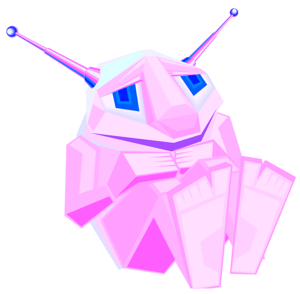
I'm asking you to describe a *set*. This set is the *concept* I want you to *learn*.

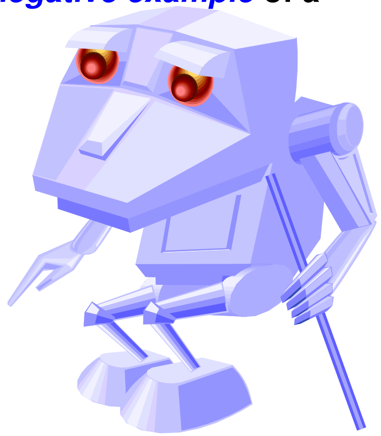
This is called *inductive learning*, i.e., learning a generalization from a set of examples.

Concept learning is a typical inductive learning problem: given examples of some concept, such as "cat," "soybean disease," or "good stock investment," we attempt to infer a definition that will allow the learner to correctly recognize future instances of that concept.

Supervised learning

This is called *supervised learning* because we assume that there is a *teacher* who classified the training data: the learner is told whether an instance is a *positive* or *negative example* of a target concept.





Supervised learning – the question

This definition might seem counter intuitive. If the teacher knows the concept, why doesn't s/he tell us directly and save us all the work?

Supervised learning – the answer

The teacher only knows the classification, the learner has to find out what the classification is. Imagine an online store: there is a lot of data concerning whether a customer returns to the store. The information is there in terms of attributes and whether they come back or not. However, it is up to the learning system to characterize the concept, e.g,

If a customer bought more than 4 books, s/he will return.

If a customer spent more than $50, s/he will return.

Rewarded card example

- Deck of cards, with each card designated by [r,s], its rank and suit, and some cards "rewarded"
- Background knowledge in the KB:
  $((r=1) \vee \ldots \vee (r=10)) \Leftrightarrow NUM(r)$
  $((r=J) \vee (r=Q) \vee (r=K)) \Leftrightarrow FACE(r)$
  $((s=S) \vee (s=C)) \Leftrightarrow BLACK(s)$
  $((s=D) \vee (s=H)) \Leftrightarrow RED(s)$
- Training set:
  $REWARD([4,C]) \wedge REWARD([7,C]) \wedge REWARD([2,S]) \wedge \neg REWARD([5,H]) \wedge \neg REWARD([J,S])$

Rewarded card example

Training set:
$REWARD([4,C]) \wedge REWARD([7,C]) \wedge REWARD([2,S]) \wedge \neg REWARD([5,H]) \wedge \neg REWARD([J,S])$

Card	In the target set?
4 ♣	yes
7 ♣	yes
2 ♠	yes
5 ♥	no
J ♠	no

Possible *inductive hypothesis*, h,:
$h = (NUM(r) \wedge BLACK(s) \Leftrightarrow REWARD([r,s]))$

Learning a predicate

- Set E of objects (e.g., cards, drinking cups, writing instruments)
- ***Goal predicate*** **CONCEPT (X), where X is an object in E, that takes the value True or False (e.g., REWARD, MUG, PENCIL, BALL)**
- ***Observable predicates*** **A(X), B(X), … (e.g., NUM, RED, HAS-HANDLE, HAS-ERASER)**
- ***Training set*****: values of CONCEPT for some combinations of values of the observable predicates**
- **Find a representation of CONCEPT of the form**

  $\text{CONCEPT}(X) \Leftrightarrow A(X) \wedge (B(X) \vee C(X))$

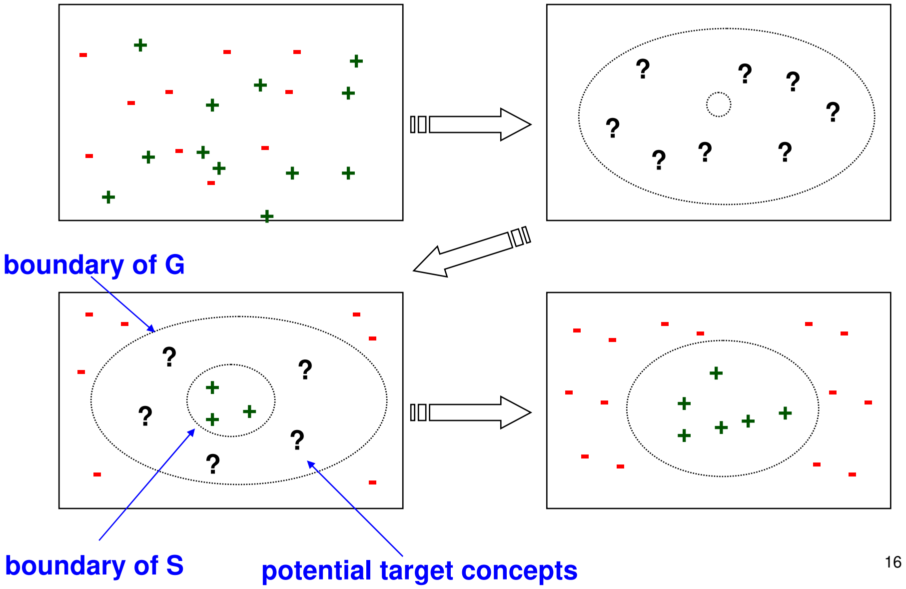
How can we do this?

- **Go with the most general hypothesis possible:**
  "any card is a rewarded card"
  This will cover all the positive examples, but will not be able to eliminate any negative examples.
- **Go with the most specific hypothesis possible:**
  "the rewarded cards are 4 ♣, 7 ♣, 2 ♠"
  This will correctly sort all the examples in the training set, but it is overly specific, will not be able to sort any new examples.
- **But the above two are good starting points.**

Version space algorithm

- **What we want to do is start with the most general and specific hypotheses, and**
  when we see a positive example, we minimally generalize the most specific hypothesis
  when we see a negative example, we minimally specialize the most general hypothesis
- **When the most general hypothesis and the most specific hypothesis are the same, the algorithm has *converged*, this is the target concept**

Pictorially



Hypothesis space

- When we shrink G, or enlarge S, we are essentially conducting a search in the *hypothesis space*
- A hypothesis is any sentence h of the form
  $CONCEPT(X) \Leftrightarrow A(X) \wedge (B(X) \vee C(X))$
  where, the right hand side is built with observable predicates
- The set of all hypotheses is called the *hypothesis space*, or *H*
- A hypothesis h agrees with an example if it gives the correct value of CONCEPT

Size of the hypothesis space

- n observable predicates
- 2^n entries in the truth table
- A hypothesis is any subset of observable predicates with the associated truth tables: so there are 2^(2^n) hypotheses to choose from:

BIG! 2^{2^n}

- n=6 $\Rightarrow$ 2 ^ 64 = 1.8 x 10 ^ 19

BIG!

- Generate-and-test won't work.

Simplified Representation for the card problem

For simplicity, we represent a concept by rs, with:

- r = a, n, f, 1, …, 10, j, q, k
- s = a, b, r, ♣, ♠, ♦, ♥

For example:

- n♠ represents:
  NUM(r) ∧ (s=♠) ⇔ REWARD([r,s])
- aa represents:
  ANY-RANK(r) ∧ ANY-SUIT(s) ⇔ REWARD([r,s])

Extension of an hypothesis

The *extension* of an hypothesis h is the set of objects that verifies h.

For instance,

the extension of f♠ is: {j♠, q♠, k♠}, and

the extension of aa is the set of all cards.

More general/specific relation

Let h1 and h2 be two hypotheses in H

h1 is more general than h2 iff the extension of h1 is a proper superset of the extension of h2

For instance,

aa is more general than f♦,

f♥ is more general than q♥,

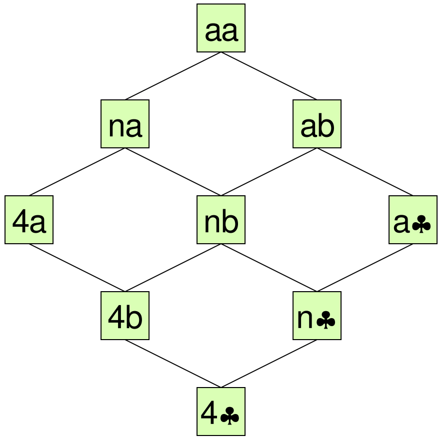
fr and nr are not comparable

More general/specific relation (cont'd)

The inverse of the "more general" relation is the "more specific" relation

The "more general" relation defines a partial ordering on the hypotheses in H

A subset of the partial order for cards



G-Boundary / S-Boundary of V

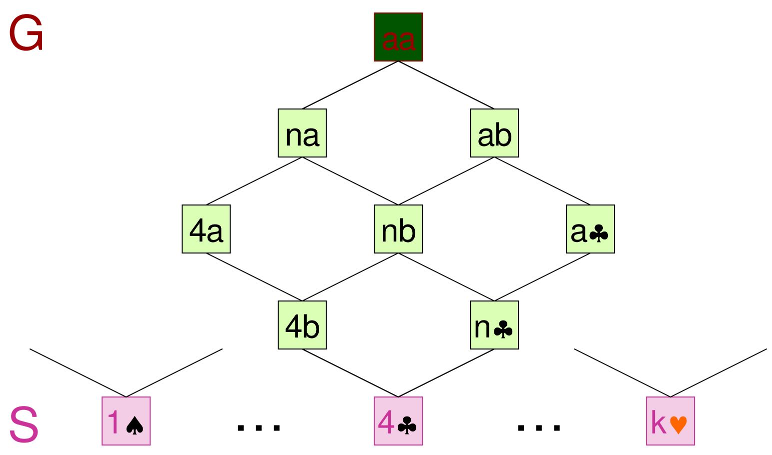
An hypothesis in V is ***most general*** iff no hypothesis in V is more general

G-boundary G of V: Set of most general hypotheses in V

An hypothesis in V is ***most specific*** iff no hypothesis in V is more general

S-boundary S of V: Set of most specific hypotheses in V

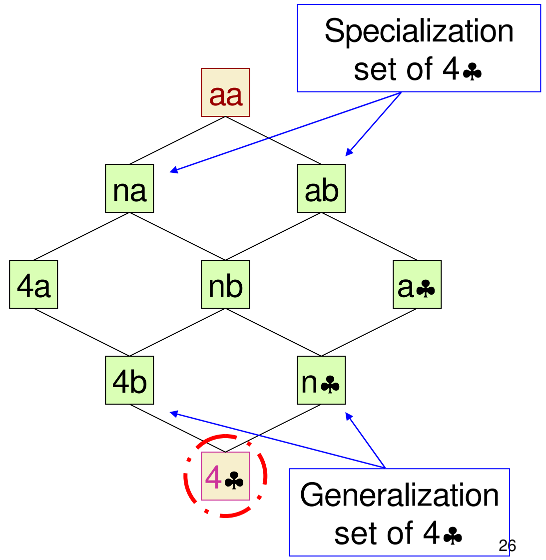
Example: The starting hypothesis space



4♣ is a positive example

We replace every hypothesis in S whose extension does not contain 4♣ by its generalization set

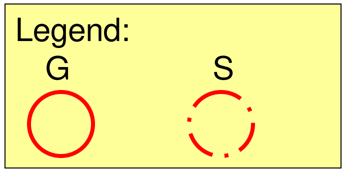
The generalization set of a hypothesis h is the set of the hypotheses that are immediately more general than h

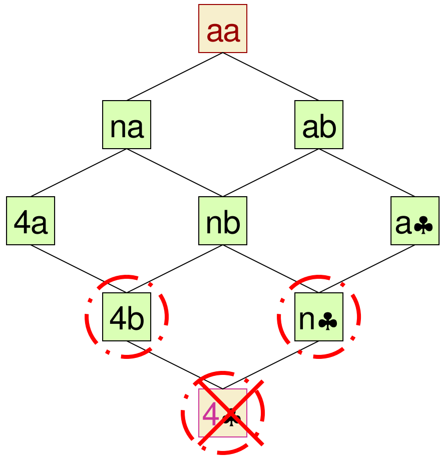


7♣ is the next positive example

Minimally generalize the most specific hypothesis set

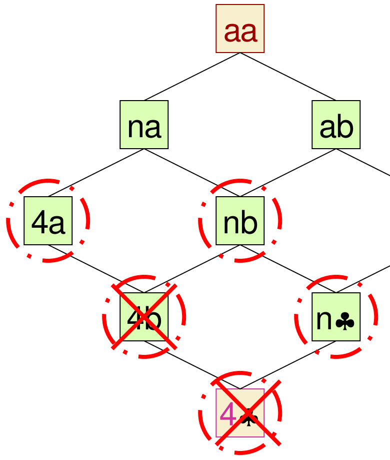
We replace every hypothesis in S whose extension does not contain 7♣ by its generalization set





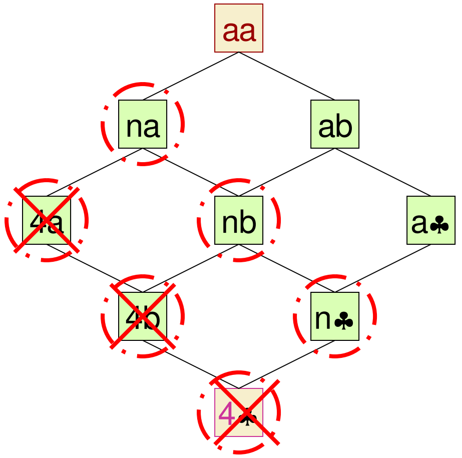
7♣ is positive(cont'd)

Minimally generalize the most specific hypothesis set



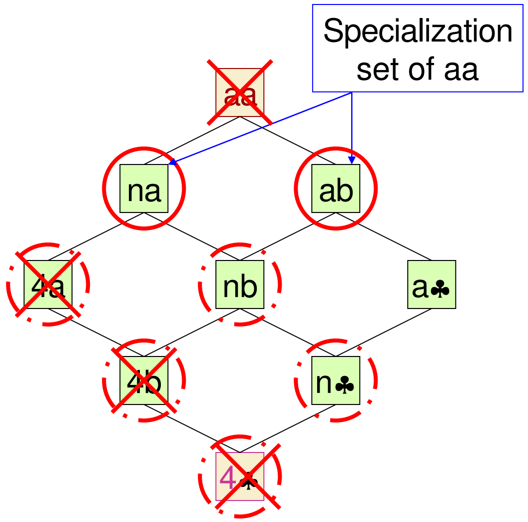
7♣ is positive (cont'd)

Minimally generalize the most specific hypothesis set



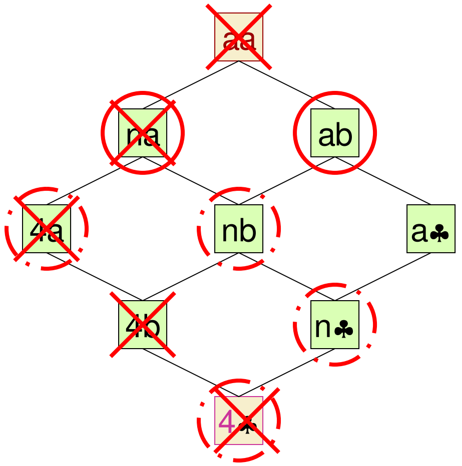
5♥ is a negative example

Minimally specialize the most general hypothesis set



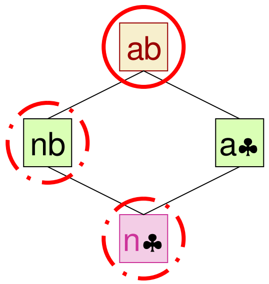
5♥ is negative(cont'd)

Minimally specialize the most general hypothesis set



After 3 examples (2 positive,1 negative)

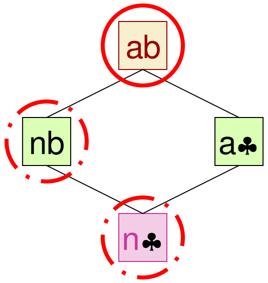
G and S, and all hypotheses in between form exactly the version space



1. If an hypothesis between G and S disagreed with an example x, then an hypothesis G or S would also disagree with x, hence would have been removed

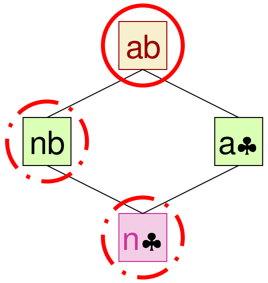
After 3 examples (2 positive,1 negative)

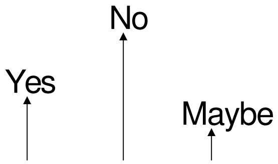
G and S, and all hypotheses in between form exactly the version space



2. If there were an hypothesis not in this set which agreed with all examples, then it would have to be either no more specific than any member of G – but then it would be in G – or no more general than some member of S – but then it would be in S

At this stage

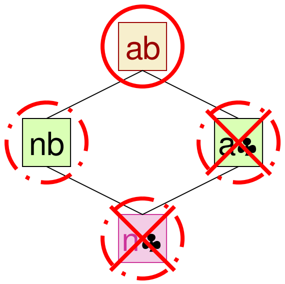




Do 8♣, 6♦, j♠ satisfy CONCEPT?

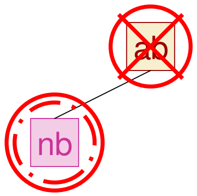
2♠ is the next positive example

Minimally generalize the most specific hypothesis set



j♠ is the next negative example

Minimally specialize the most general hypothesis set



Result

+ 4♣ 7♣ 2♠

– 5♥ j♠

nb

NUM(r) $\wedge$ BLACK(s) $\Leftrightarrow$ REWARD([r,s])

The version space algorithm

Begin

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

For each example x

If x is positive, then
(G,S) $\leftarrow$ POSITIVE-UPDATE(G,S,x)

else
(G,S) $\leftarrow$ NEGATIVE-UPDATE(G,S,x)

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 the positive instances and none of the negative instances

End

The version space algorithm (cont'd)

POSITIVE-UPDATE(G,S,p)

Begin

Delete all members of G that fail to match p

For every s $\in$ S, if s does not match p, replace s with its most specific generalizations that match p;

Delete from S any hypothesis that is more general than some other hypothesis in S;

Delete from S any hypothesis that is neither more specific than nor equal to a hypothesis in G; (different than the textbook)

End;

The version space algorithm (cont'd)

NEGATIVE-UPDATE(G,S,n)

Begin

Delete all members of S that match n

For every g $\in$ G, that matches n, replace g with its most general specializations that do not match n;

Delete from G any hypothesis that is more specific than some other hypothesis in G;

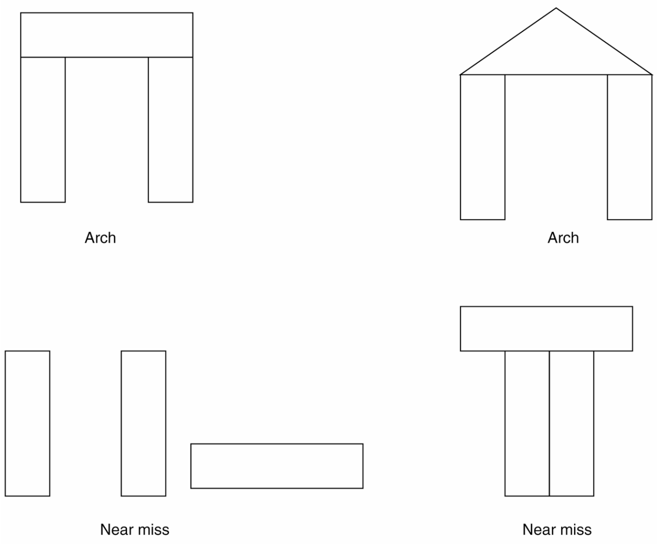
Delete from G any hypothesis that is neither more general nor equal to hypothesis in S; (different than the textbook)

End;

Comments on Version Space Learning (VSL)

- It is a bi-directional search. One direction is specific to general and is driven by positive instances. The other direction is general to specific and is driven by negative instances.
- It is an ***incremental learning algorithm***. The examples do not have to be given all at once (as opposed to learning decision trees.) The version space is meaningful even before it converges.
- The order of examples matters for the speed of convergence
- As is, cannot tolerate noise (misclassified examples), the version space might collapse

Examples and near misses for the concept “arch”



More on generalization operators

- Replacing constants with variables. For example,

  color (ball,red)
  generalizes to
  color (X,red)

- Dropping conditions from a conjunctive expression. For example,

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

More on generalization operators (cont’d)

- Adding a disjunct to an expression. For example,

  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))

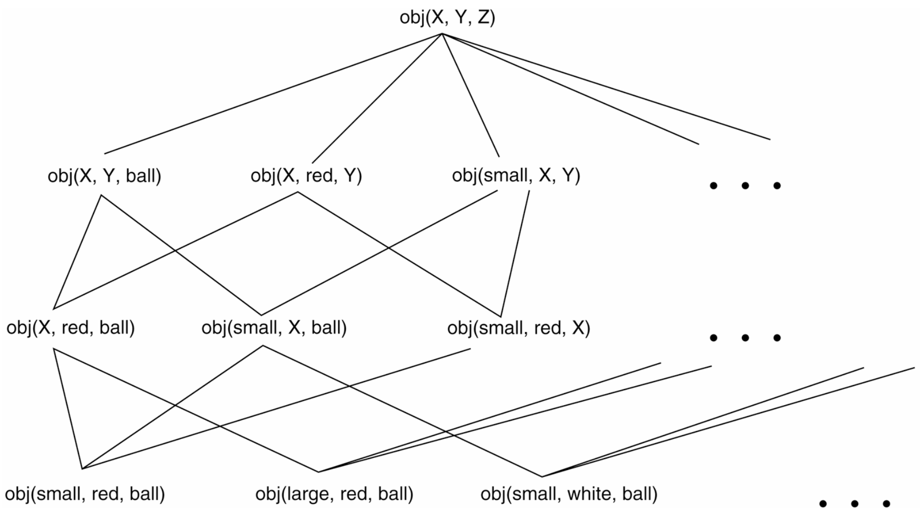
- Replacing a property with its parent in a class hierarchy. If we know that primary_color is a superclass of red, then

  color (X, red)
  generalizes to
  color (X, primary_color)

Another example

- sizes = {large, small}
- colors = {red, white, blue}
- shapes = {sphere, brick, cube}
- object (size, color, shape)
- If the target concept is a “red ball,” then size should not matter, color should be red, and shape should be sphere
- If the target concept is “ball,” then size or color should not matter, shape should be sphere.

A portion of the concept space



Learning the concept of a “red ball”

G : { obj (X, Y, Z)}
S : { }

positive: obj (small, red, sphere)

G: { obj (X, Y, Z)}
S : { obj (small, red, sphere) }

negative: obj (small, blue, sphere)

G: { obj (large, Y, Z), obj (X, red, Z), obj (X, white, sphere)
obj (X,Y, brick), obj (X, Y, cube) }
S: { obj (small, red, sphere) }
delete from G every hypothesis that is neither more general than nor equal to a hypothesis in S

G: {obj (X, red, Z) }
S: { obj (small, red, sphere) }

Learning the concept of a “red ball” (cont’d)

G: { obj (X, red, Z) }
S: { obj (small, red, sphere) }

positive: obj (large, red, sphere)

G: { obj (X, red, Z)}
S : { obj (X, red, sphere) }

negative: obj (large, red, cube)

G: { obj (small, red, Z), obj (X, red, sphere),
obj (X, red, brick)}
S: { obj (X, red, sphere) }
delete from G every hypothesis that is neither more general than nor equal to a hypothesis in S

G: {obj (X, red, sphere) }
S: { obj (X, red, sphere) } <u>converged to a single concept</u>

LEX: a program that learns heuristics

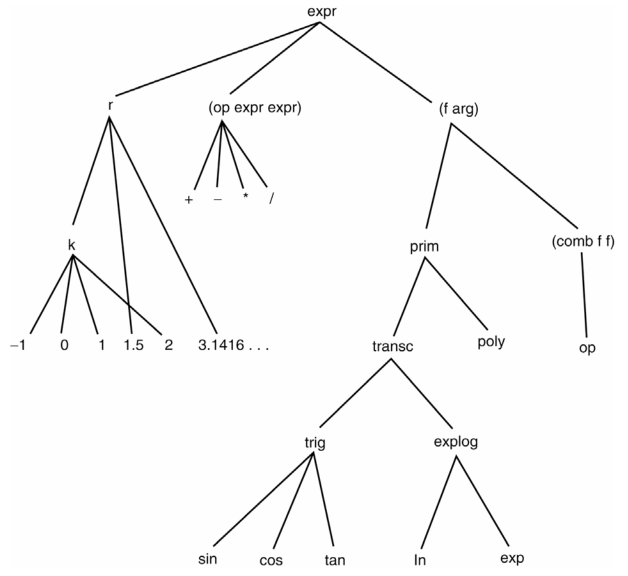
- Learns heuristics for symbolic integration problems
- Typical transformations used in performing integration include

  OP1: $\int r\, f(x)\, dx \rightarrow r \int f(x)\, dx$
  OP2: $\int u\, dv \rightarrow uv - \int v\, du$
  OP3: $1 * f(x) \rightarrow f(x)$
  OP4: $\int (f_1(x) + f_2(x))\, dx \rightarrow \int f_1(x)\, dx + \int f_2(x)\, dx$

- A heuristic tells when an operator is particularly useful:

  If a problem state matches $\int x$ transcendental$(x)\, dx$
  then apply OP2 with bindings
  $u = x$
  $dv =$ transcendental $(x)\, dx$

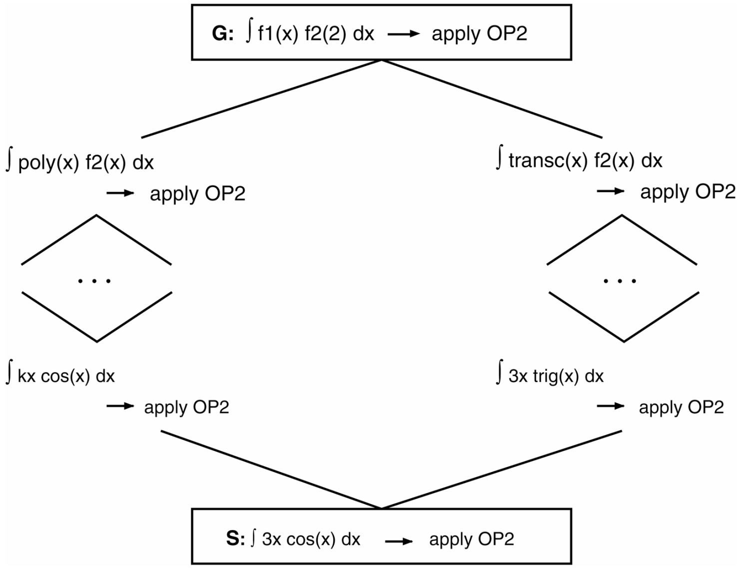
A portion of LEX's hierarchy of symbols



The overall architecture

- A ***generalizer*** that uses candidate elimination to find heuristics
- A ***problem solver*** that produces positive and negative heuristics from a problem trace
- A ***critic*** that produces positive and negative instances from a problem traces (the ***credit assignment problem***)
- A ***problem generator*** that produces new candidate problems

A version space for OP2 (Mitchell et al.,1983)



Comments on LEX

• The evolving heuristics are not guaranteed to be admissible. The solution path found by the problem solver may not actually be a shortest path solution.

• The problem generator is the least developed part of the program.

• Empirical studies:
before: 5 problems solved in an average of 200 steps
train with 12 problems
after: 5 problems solved in an average of 20 steps

More comments on VSL

• Still lots of research going on

• Uses breadth-first search which might be inefficient:

- might need to use ***beam-search*** to prune hypotheses from G and S if they grow excessively
- another alternative is to use ***inductive-bias*** and restrict the concept language

• How to address the noise problem?
Maintain several G and S sets.