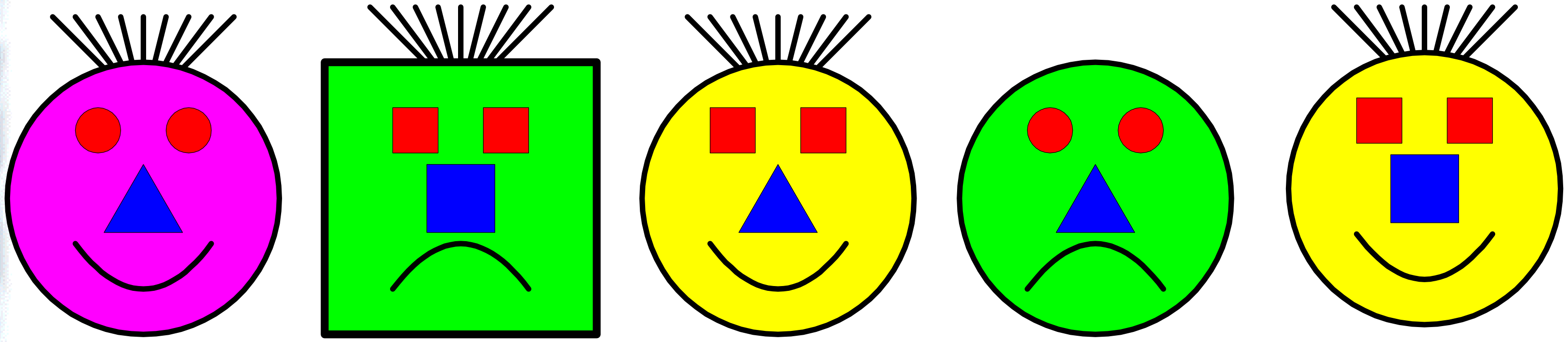
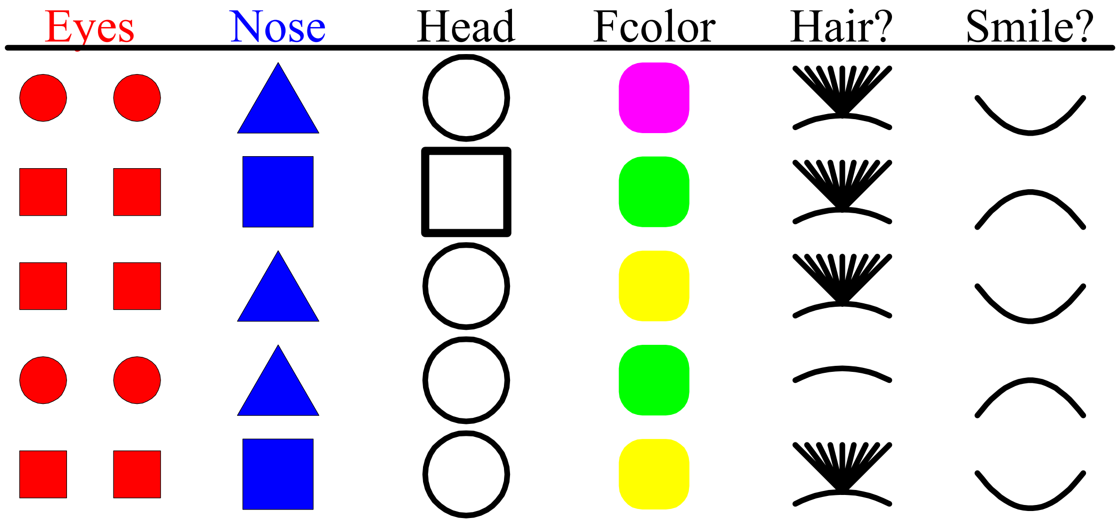
Concept Learning

- Learning from examples
- General-to-specific ordering over hypotheses
- Version Spaces and candidate elimination algorithm
- Picking new examples
- The need for inductive bias

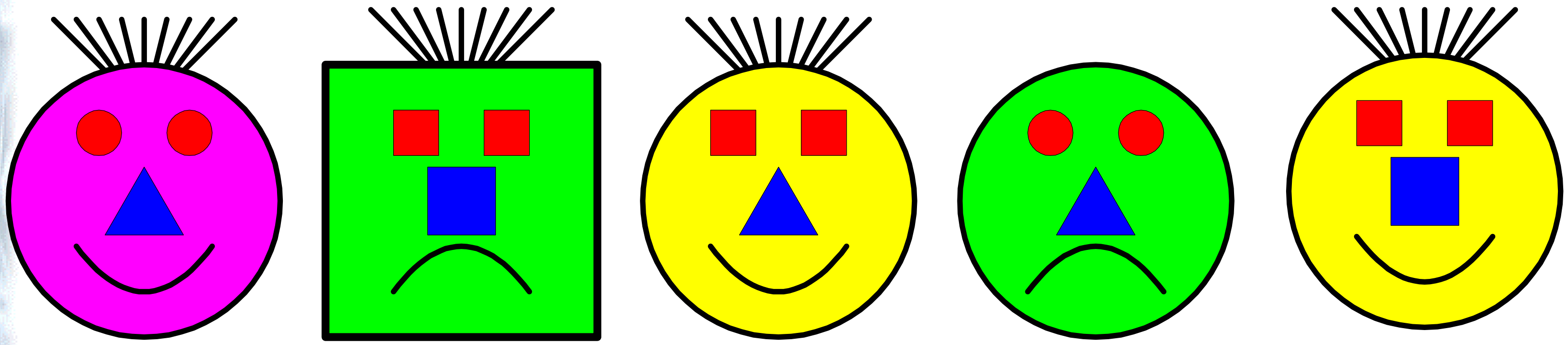
Some Examples for SmileyFaces





Eyes	Nose	Head	Fcolor	Hair?	Smile?
● ●	▲	○	🟪	⋎	◡
■ ■	■	□	🟩	⋎	◠
■ ■	▲	○	🟨	⋎	◡
● ●	▲	○	🟩	⌒	◠
■ ■	■	○	🟨	⋎	◡

Features from Computer View



Eyes	Nose	Head	Fcolor	Hair?	Smile?
Round	Triangle	Round	Purple	Yes	Yes
Square	Square	Square	Green	Yes	No
Square	Triangle	Round	Yellow	Yes	Yes
Round	Triangle	Round	Green	No	No
Square	Square	Round	Yellow	Yes	Yes

Representing Hypotheses

Many possible representations for hypotheses *h*

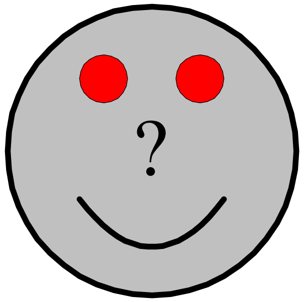
Idea: h as conjunctions of constraints on features

Each constraint can be:

- a specific value (e.g., *Nose* = *Square*)
- don't care (e.g., *Eyes = ?)*
- no value allowed (e.g., *Water=Ø*)

For example,

Eyes	Nose	Head	Fcolor	Hair?
<Round,	?,	Round,	?,	No>



Prototypical Concept Learning Task

Given:

- Instances *X*: Faces, each described by the attributes *Eyes*, *Nose*, *Head*, *Fcolor*, and *Hair?*
- Target function *c: Smile? : X* -> { no, yes }
- Hypotheses *H*: Conjunctions of literals such as

  <?,Square,Square,Yellow,?>
- Training examples *D*: Positive and negative examples of the target function

  $$< x_1, c(x_1) >, < x_2, c(x_2) >, \ldots, < x_m, c(x_m) >$$

Determine: a hypothesis *h* in *H* such that *h(x)=c(x)* for all *x* in *D*.

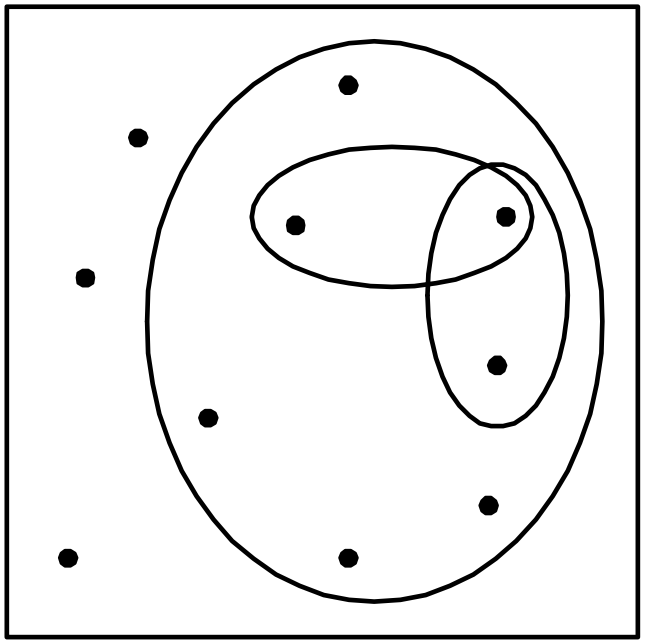
Inductive Learning Hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

- What are the implications?
- Is this reasonable?
- What (if any) are our alternatives?
- What about concept drift (what if our views/tastes change over time)?

Instances, Hypotheses, and More-General-Than

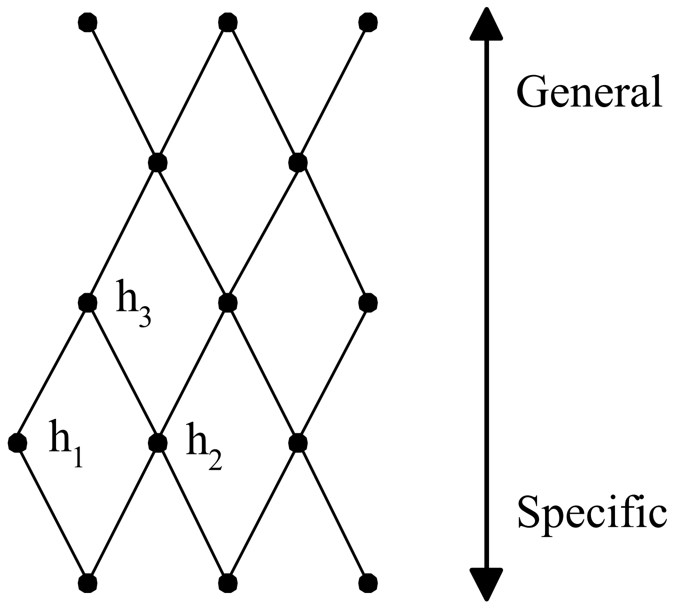
Instances X



x_1=<Round,Square,Square,Purple,Yes>

x_2=<Round,Square,Round,Green,Yes>

Hypotheses H



h_1=<Round,?,Square,?,?>

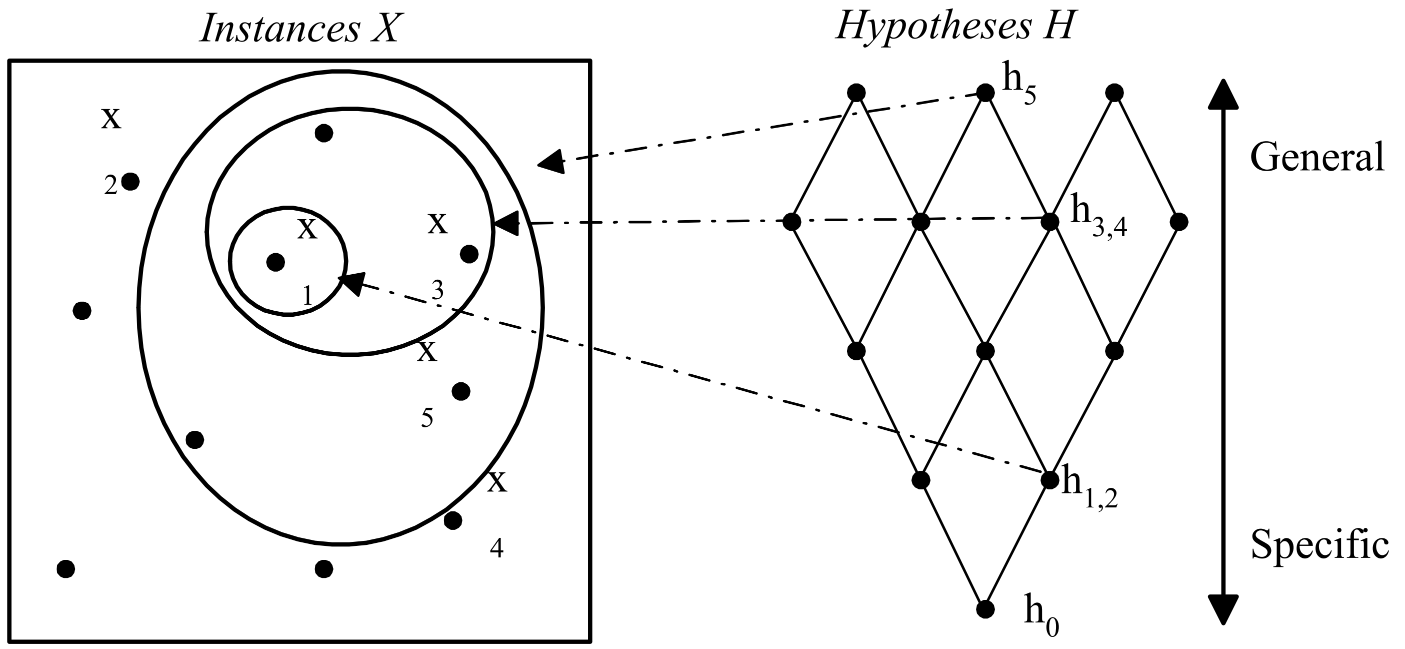
h_2=<Round,?,?,?,Yes>

h_3=<Round,?,?,?,?>

Find-S Algorithm

```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
   For each attribute constraint $a_i$ in $h$
      IF the constraint $a_i$ in $h$ is satisfied by $x$ THEN
         do nothing
      ELSE
         replace $a_i$ in h by next more general constraint satisfied by $x$
3. Output hypothesis $h$
```

Hypothesis Space Search by `Find-S`



h_0=<ϕ,ϕ,ϕ,ϕ,ϕ >

x_1=<Round,Triangle,Round,Purple,Yes> +	h_1=<Round,Triangle,Round,Purple,Yes>
x_2=<Square,Square,Square,Green,Yes> -	h_2=<Round,Triangle,Round,Purple,Yes>
x_3=<Square,Triangle,Round,Yellow,Yes> +	h_3=<?,Triangle,Round,?,Yes>
x_4=<Round,Triangle,Round,Green,No> -	h_4=<?,Triangle,Round,?,Yes>
x_5=<Square,Square,Round,Yellow,Yes> +	h_5=<?,?,Round,?,Yes>

Complaints about `Find-S`

- Cannot tell whether it has learned concept
- Cannot tell when training data inconsistent
- Picks a maximally specific *h* (why?)
- Depending on *H*, there might be several!

- How do we fix this?

The `List-Then-Eliminate` Algorithm

1. Set *VersionSpace* equal to a list containing every hypothesis in *H*
2. For each training example, <*x,c(x)*>

   remove from *VersionSpace* any hypothesis *h* for which *h(x) != c(x)*
3. Output the list of hypotheses in *VersionSpace*

- But is listing all hypotheses reasonable?
- How many different hypotheses in our simple problem?
  - How many not involving "?" terms?

Version Spaces

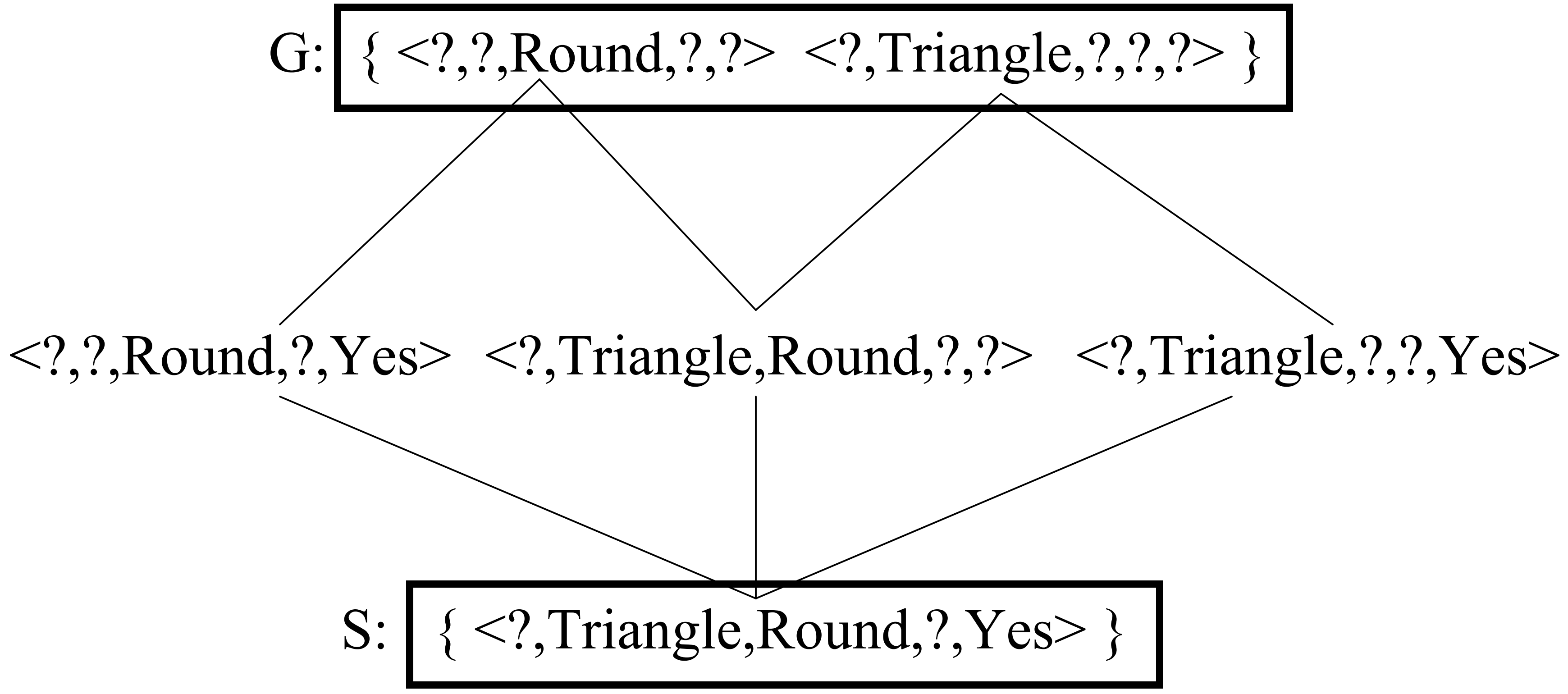
A hypothesis *h* is **consistent** with a set of training examples *D* of target concept *c* if and only if *h(x)=c(x)* for each training example in *D*.

$$Consistent(h, D) \equiv (\forall < x, c(x) > \in D)\, h(x) = c(x)$$

The **version space**, $VS_{H,D}$, with respect to hypothesis space *H* and training examples *D*, is the subset of hypotheses from *H* consistent with all training examples in *D*.

$$VS_{H,D} \equiv \{h \in H \mid Consistent(h, D)\}$$

Example Version Space



Representing Version Spaces

The **General boundary**, G, of version space $VS_{H,D}$ is the set of its maximally general members.

The **Specific boundary**, S, of version space $VS_{H,D}$ is the set of its maximally specific members.

Every member of the version space lies between these boundaries

$$VS_{H,D} = \{h \in H \mid (\exists s \in S)(\exists g \in G)(g \geq h \geq s)\}$$

where $x \geq y$ means x is more general or equal to y

Candidate Elimination Algorithm

G = maximally general hypotheses in *H*

S = maximally specific hypotheses in *H*

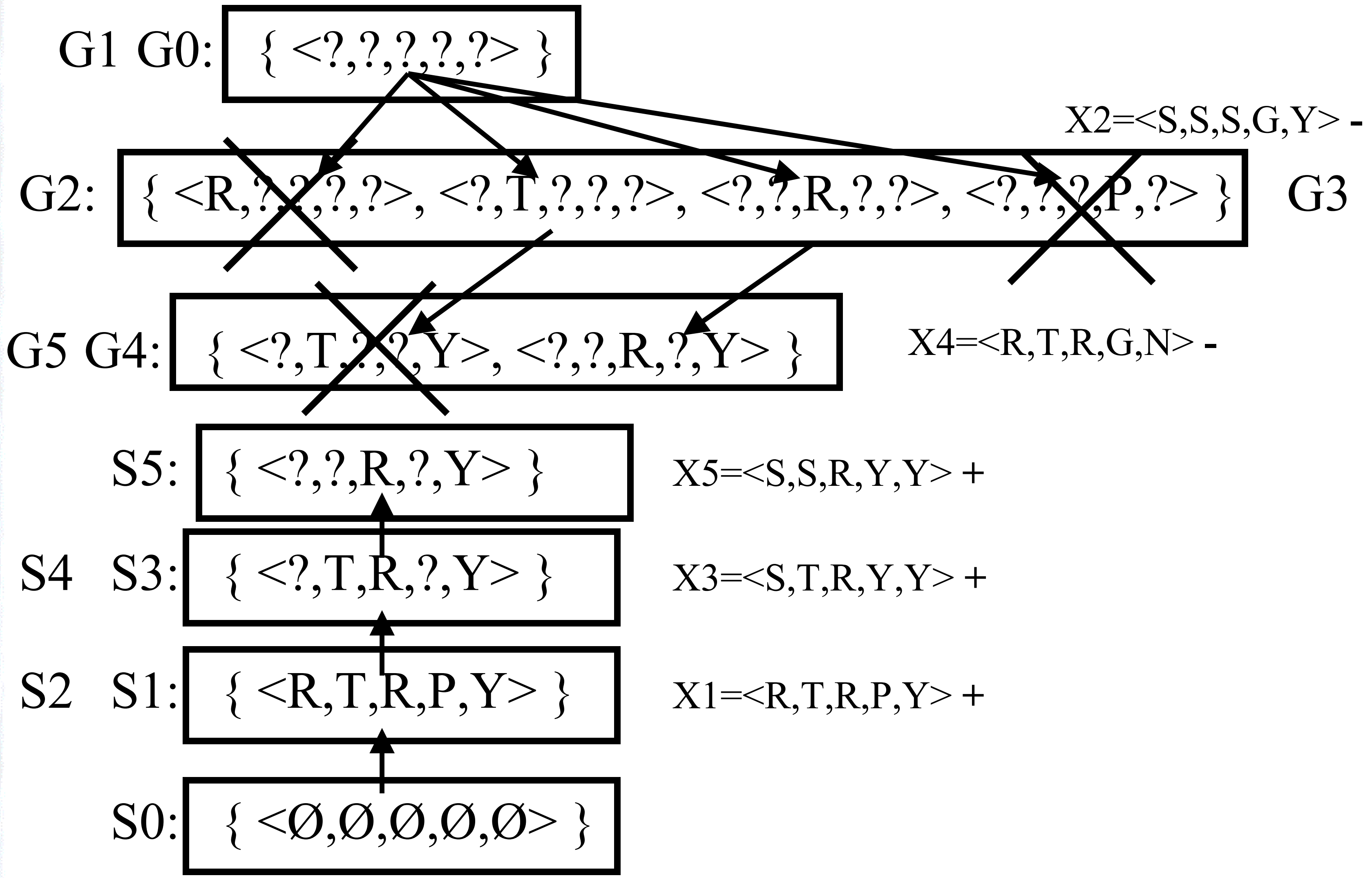
```
For each training example $d$, do
    If $d$ is a positive example
        Remove from G any hypothesis that does not include $d$
        For each hypothesis $s$ in S that does not include $d$
            Remove $s$ from S
            Add to S all minimal generalizations $h$ of $s$ such that
                1. $h$ includes $d$, and
                2. Some member of G is more general than $h$
            Remove from S any hypothesis that is more general
              than another hypothesis in S
```

Candidate Elimination Algorithm (cont)

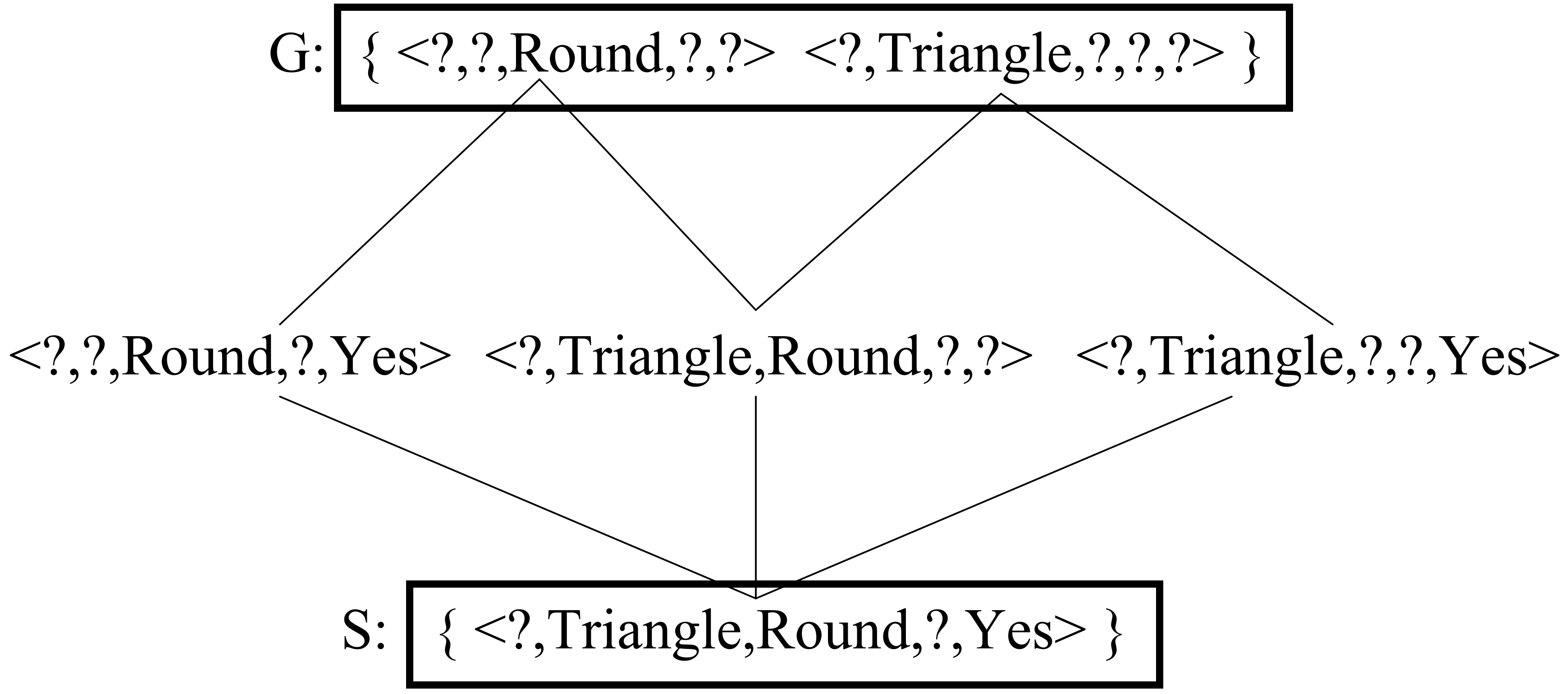
```
For each training example d, do (cont)
    If d is a negative example
        Remove from S any hypothesis that does include d
        For each hypothesis g in G that does include d
            Remove g from G
            Add to G all minimal generalizations h of g such that
                1. h does not include d, and
                2. Some member of S is more specific than h
            Remove from G any hypothesis that is less general
              than another hypothesis in G

If G or S ever becomes empty, data not consistent (with H)
```

Example Trace

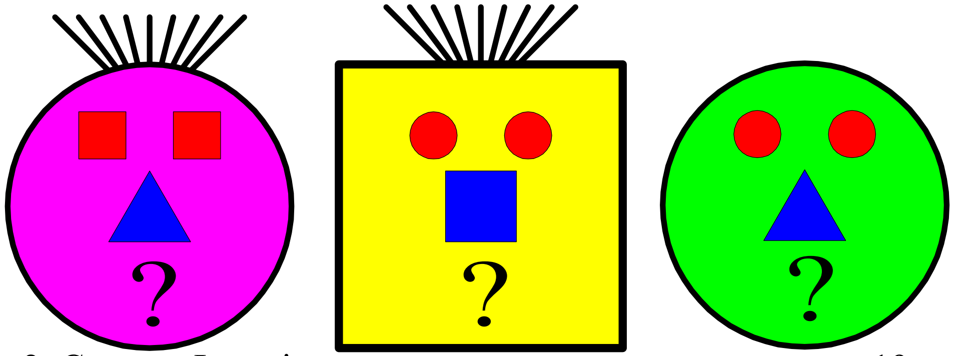


What Training Example Next?



How Should These Be Classified?

$$\begin{tikzcd}[column sep=small]
 & \text{G: } \{ <?,?,\text{Round},?,?> \arrow[dl, no head] \arrow[dr, no head] & & <?,\text{Triangle},?,?,?> \} \arrow[dl, no head] \arrow[dr, no head] & \\
<?,?,\text{Round},?,\text{Yes}> \arrow[drr, no head] & & <?,\text{Triangle},\text{Round},?,?> \arrow[d, no head] & & <?,\text{Triangle},?,?,\text{Yes}> \arrow[dll, no head] \\
 & & \text{S: } \{ <?,\text{Triangle},\text{Round},?,\text{Yes}> \} & &
\end{tikzcd}$$



What Justifies this Inductive Leap?

+ < Round, Triangle, Round, Purple, Yes >

+ < Square, Triangle, Round, Yellow, Yes >

S: < ?, Triangle, Round, ?, Yes >

Why believe we can classify the unseen?

< Square, Triangle, Round, Purple, Yes > ?

An UN-Biased Learner

Idea: Choose *H* that expresses every teachable concept (i.e., *H* is the power set of *X*)

Consider *H'* = disjunctions, conjunctions, negations over previous *H*.

For example:

$$< ?, Triangle, Round, ?, Yes > \vee < Square, Square, ?, Purple, ? >$$

What are S, G, in this case?

Inductive Bias

Consider

- concept learning algorithm L
- instances X, target concept c
- training examples $D_c = \{<x, c(x)>\}$
- let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on data D_c.

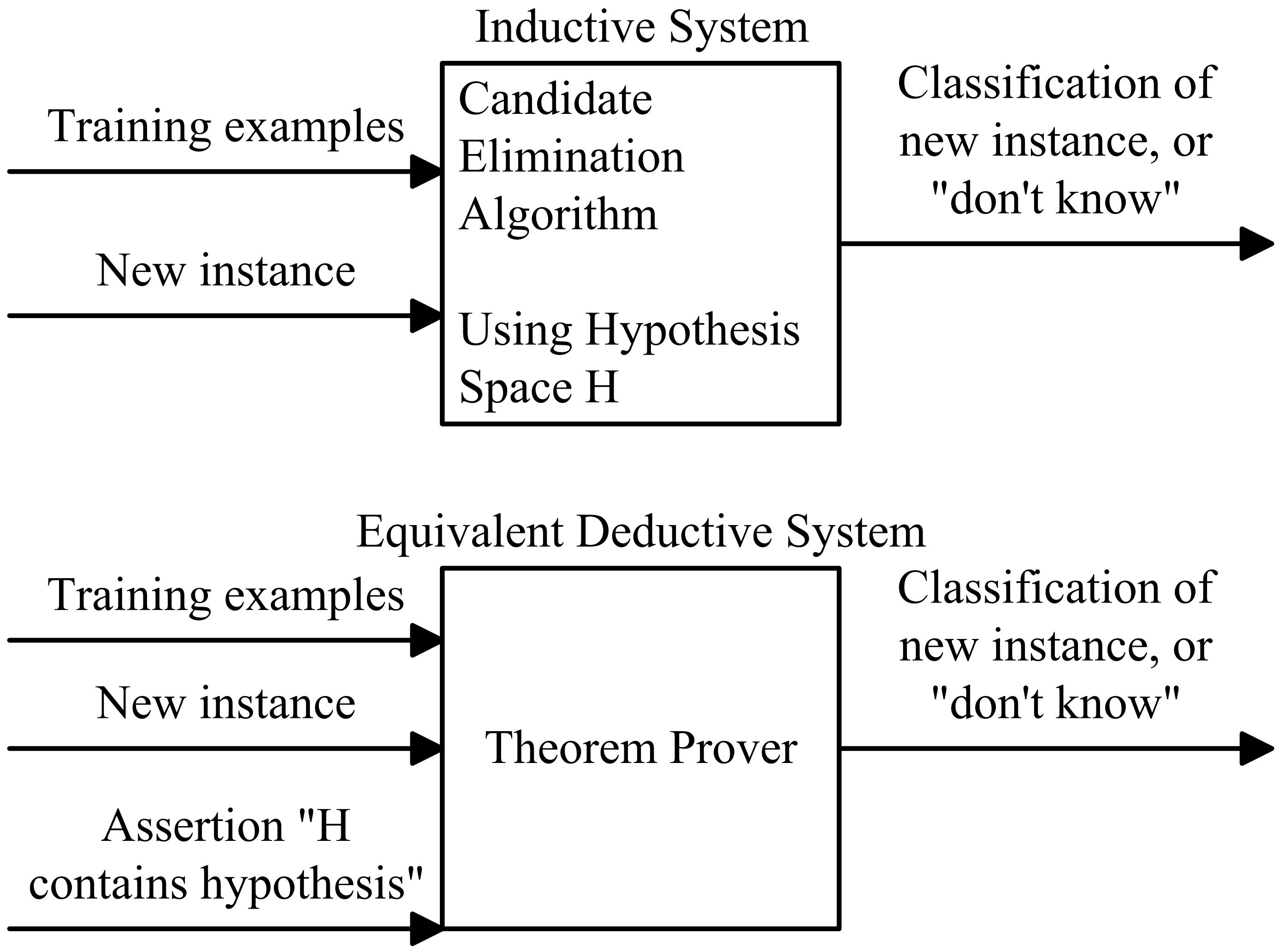
Definition:

The **inductive bias** of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

where A $\vdash$ B means A logically entails B

Inductive Systems and Equivalent Deductive Systems


Inductive System
Training examples
New instance
Candidate Elimination Algorithm
Using Hypothesis Space H
Classification of new instance, or "don't know"
Equivalent Deductive System
Training examples
New instance
Assertion "H contains hypothesis"
Theorem Prover
Classification of new instance, or "don't know"

Three Learners with Different Biases

1. *Rote learner*: store examples, classify new instance iff it matches previously observed example (don't know otherwise).

2. *Version space candidate elimination algorithm*.

3. *Find-S*

Summary Points

1. Concept learning as search through *H*
2. General-to-specific ordering over *H*
3. Version space candidate elimination algorithm
4. *S* and *G* boundaries characterize learner's uncertainty
5. Learner can generate useful queries
6. Inductive leaps possible only if learner is biased
7. Inductive learners can be modeled by equivalent deductive systems