Machine Learning

Part I

A Biswas

IIEST, Shibpur

Syllabus

Introduction

Learning Problems, Well-posed learning problems, Designing learning systems.

Concept Learning

Concept learning task, Inductive hypothesis, Ordering of Hypothesis, General-to-specific ordering of hypotheses. Version spaces, Inductive Bias.

Learning Rule Sets

Sequential Covering Algorithm, First Order Rules, Induction, First Order Resolution, Inverting Resolution.

Regression

Linear regression, Notion of cost function, Logistic Regression, Cost function for logistic regression, application of logistic regression to multi-class classification.

Continued …

Syllabus (continued)

Supervised Learning

Support Vector Machine, Decision tree Learning, Representation, Problems, Decision Tree Learning Algorithm, Attributes, Inductive Bias, Overfitting.

Bayes Theorem, Bayesian learning, Maximum likelihood, Least squared error hypothesis, Gradient Search, Naive Bayes classifier, Bayesian Network, Expectation Maximization Algorithm.

Unsupervised learning

Clustering, K-means clustering, hierarchical clustering.

Instance-Based Learning

k-Nearest Neighbour Algorithm, Radial Basis Function, Locally Weighted Regression, Locally Weighted Function.

Neural networks

Linear threshold units, Perceptrons, Multilayer networks and back-propagation, recurrent networks. Probabilistic Machine Learning, Maximum Likelihood Estimation.

Regularization, Preventing Overfitting, Ensemble Learning: Bagging and Boosting, Dimensionality reduction

Books

Machine learning by Tom M Mitchell

Pattern Recognition and Machine Learning by Christopher M Bishop

The Hundred-Page Machine Learning Book by Andriy Burkov

Machine Learning Yearning by Andrew Ng

Neural Networks and Deep Learning by Michael Nielsen

Definition Machine Learning

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed. (ArthurSamuel(1959))

Definition Machine Learning

Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Tom Mitchell(1998)

Definition Machine Learning

Examples of **Well-posed Learning Problem**:

A checkers learning problem:

T: playing checkers

P: percentage of games own

E: playing practice games against itself

Definition Machine Learning

Examples of <u>**Well-posed Learning Problem**</u>:

Handwriting recognition:

T: recognising and classifying handwritten words in image

P: percentage of words correctly classified

E: a database of handwritten words with given classifications

Definition Machine Learning

Examples of **Well-posed Learning Problem**:

Robot driving learning:

T: driving on public highways using vision sensors

P: average distance travelled before an error (identified by a human overseer)

E: a sequence of images and steering commands recorded while observing a human driver

Designing a learning system

1. Choosing the training experience
2. Choosing the target function
3. Choosing a representation of the target function
4. Choosing a function approximation algorithm
    A. Estimating training values
    B. Adjusting the weights
5. The final design

Designing a learning system

1. Choosing the training experience

To choose the type of training experience from which our system will learn.

The type of training experience available can have a significant impact on success or failure of the learner.

Designing a learning system

1. Choosing the training experience

…

One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.

Designing a learning system

1. Choosing the training experience

…

A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.

Designing a learning system

1. Choosing the training experience

…

A third important attribute of the training experience is how well it represents the distribution of examples over which the final system performance P must be measured.

Designing a learning system

1. Choosing the training experience
2. Choosing the target function

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

Generates the legal moves from any board state.

The program needs only to learn how to choose the best move from among these legal moves.

…

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

…

This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

To choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

…

Let us call this function CHOOSEMOVE and use the notation **ChooseMove** : B —> M to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

…

An alternative target function, is an **evaluation function** that assigns a numerical score to any given board state. Let us call this target function V and again use the notation $V : B \to \mathbb{R}$ to denote that V maps any legal board state from the set B to some real value $\mathbb{R}$.

Designing a learning system

2. Choosing the target function …

Checkers-playing program:

…

Let target function V to assign higher scores to better board states.

If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

Designing a learning system

2. Choosing the target function …

Checkers-playing program: …

This can be accomplished by generating the successor board state produced by every legal move, then using V to choose the best successor state and therefore the best legal move.

Designing a learning system

2. Choosing the target function …

In fact, we often expect learning algorithms to acquire only some approximation to the target function, and for this reason the process of learning the target function is often called function approximation.

Let us use the symbol $\hat{V}$ to refer to the function that is actually learned by our program, to distinguish it from the ideal target function V.

Designing a learning system

1. Choosing the training experience
2. Choosing the target function
3. Choosing a representation of the target function
4. Choosing a function approximation algorithm
    A. Estimating training values
    B. Adjusting the weights
5. The final design

Designing a learning system …

3. Choosing a representation of the target function

Choose a representation that the learning program will use to describe the function $\hat{V}$ that it will learn.

Many options:

1. From a large table
2. By using a collection of rules
3. By using an Artificial neural network

Designing a learning system …

3. Choosing a representation of the target function …

Need a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.

The more expressive the representation, the more training data the program will require.

Designing a learning system …

We have seen so far

the **original formulation** of the learning problem by

choosing a type of **training experience,**

a **target function** to be learned, and

a **representation for this target function**

Designing a learning system

4. Choosing a function approximation algorithm

    A. Estimating training values

    B. Adjusting the weights

To learn the target function $\hat{V}$ we require a set of training examples

Designing a learning system

4. Choosing a function approximation algorithm

    A. Estimating training values

: We require training examples that assign specific scores to specific board states.

End of the game: training value is obvious.

What about the many intermediate states?

Simple approach: $V_{train}(b) \leftarrow \hat{V}(successor(b))$

Designing a learning system

4. Choosing a function approximation algorithm

A. Estimating training values

In fact, under certain conditions the approach of iteratively estimating training values based on estimates of successor state values can be proven to converge toward perfect estimates of V_{train}.

Designing a learning system

4. Choosing a function approximation algorithm
    A. Estimating training values
    B. Adjusting the weights

Now, we need to specify the learning algorithm for choosing the weights w_i to **best fit** the set of training examples $\{ < b, V_{train}(b) > \}$.

What is best fit? …

Designing a learning system

4. Choosing a function approximation algorithm

B. Adjusting the weights

What is best fit? …

To minimize the squared error

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \ training\ examples} (V_{train}(b) - \hat{V}(b))^2$$

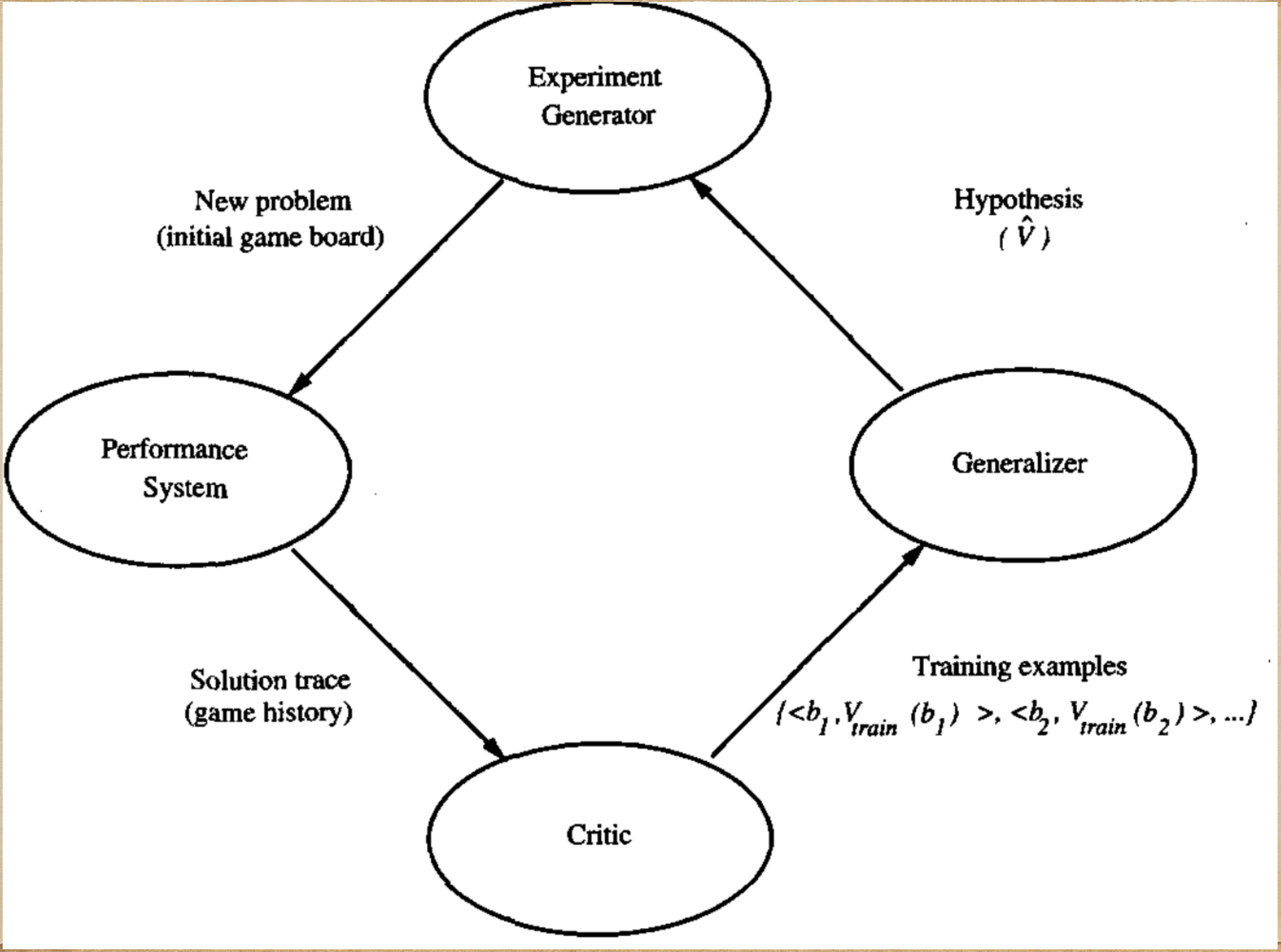
To seek the weights, or equivalently the $\hat{V}$, that minimize E for the observed training examples.

Designing a learning system

5. The final design

The final design can be naturally described by four distinct program modules that represent the central components in many learning systems

- Performance System
- Critic
- Generalizer
- Experiment Generator


Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples
$\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, ...\}$
Critic

Designing a learning system

5. The final design

Performance system:

The **Performance System** is the module that must solve the given performance task (playing checkers), by using the learned target function(s).

It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

Designing a learning system

5. The final design ...

The strategy used by the Performance System to select its next move at each step is determined by the learned $\hat{V}$ evaluation function.

Therefore, we expect its performance to improve as this evaluation function becomes increasingly accurate.

Designing a learning system

5. The final design

Critic:

The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate V_{trian} of the target function value.

Designing a learning system

5. The final design

Critic:

the Critic corresponds to the training rule

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

Designing a learning system

5. The final design

The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function.

It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

Designing a learning system

5. The final design

The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

Concept Learning and the general-to-specific ordering

The problem of inducing general functions from specific training examples is **central to learning.**

Concept learning: acquiring the definition of a general category given a sample of positive and negative training examples of the category.

Concept Learning and the general-to-specific ordering

Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.

In many cases this search can be efficiently organized by **taking advantage of a naturally occurring structure** over the hypothesis space-a general- to-specific ordering of hypotheses.

Concept Learning and the general-to-specific ordering

Acquiring general concepts from specific training examples.

People may learn general concepts or categories like bird, car, etc.

Concept Learning and the general-to-specific ordering

Each such concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute birds).

Alternatively, each concept can be thought of as a boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

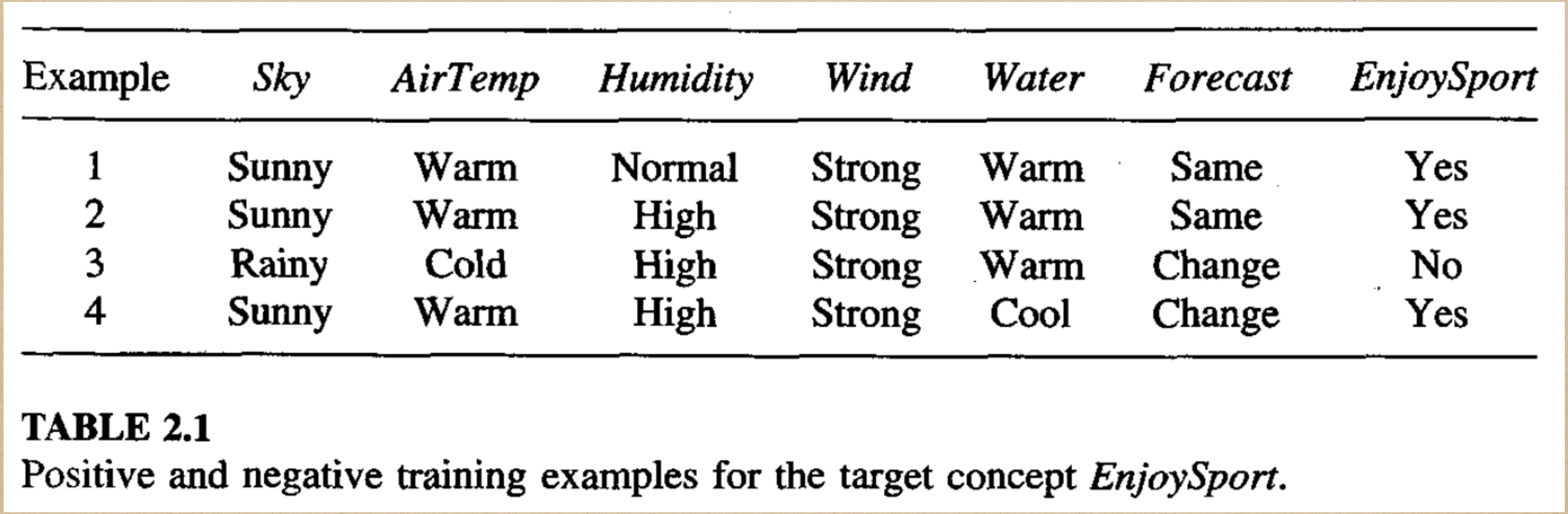
Concept Learning and the general-to-specific ordering

Each such concept can be viewed as describing some subset of objects or events defined over a larger set (e.g., the subset of animals that constitute birds).

Alternatively, each concept can be thought of as a boolean-valued function defined over this larger set (e.g., a function defined over all animals, whose value is true for birds and false for other animals).

A Concept Learning Task

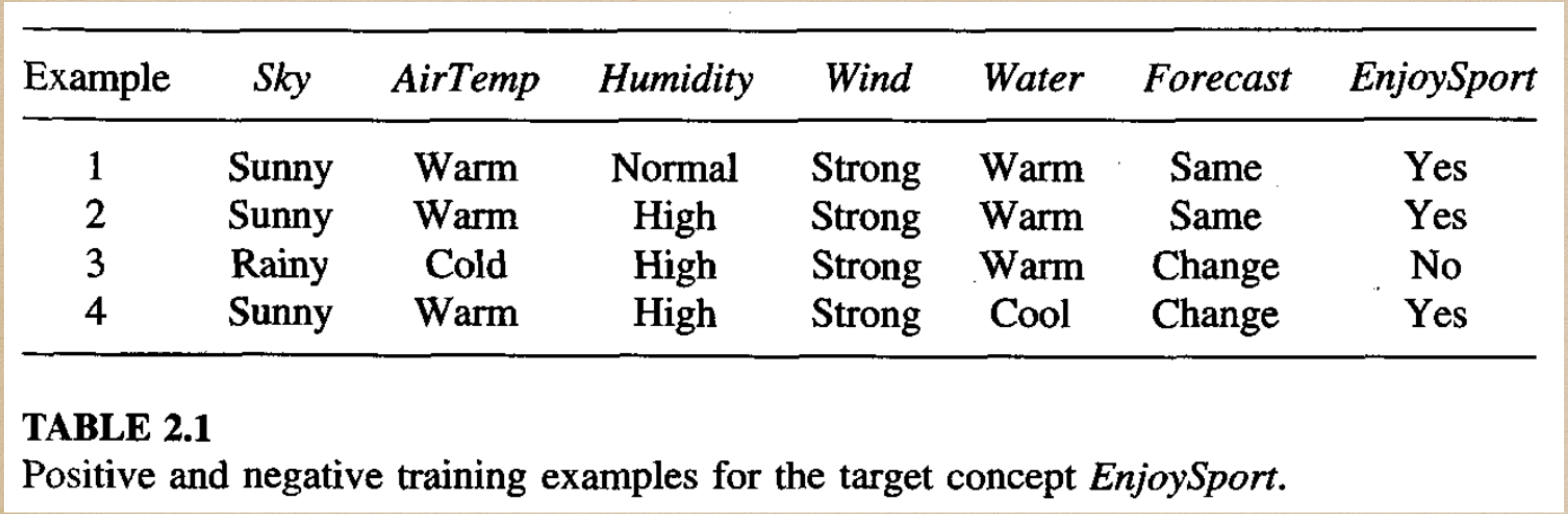
Example: task of learn- ing the target concept "days on which my friend Aldo enjoys his favorite water sport."



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

A Concept Learning Task

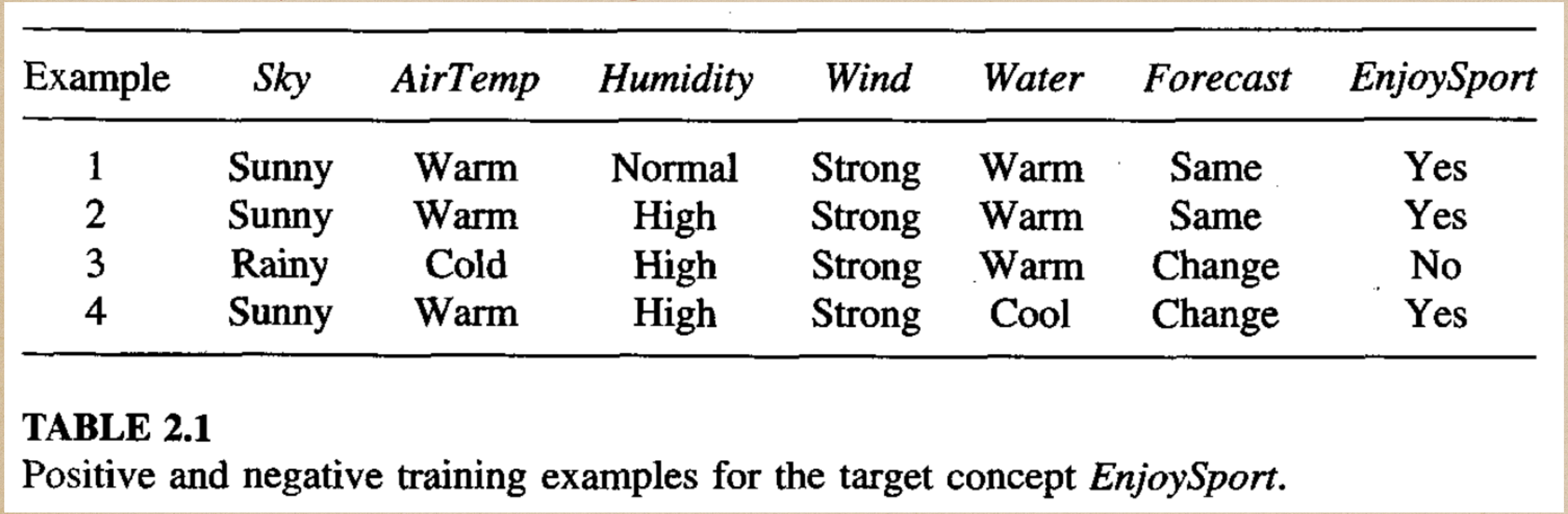


Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

Let each hypothesis be a vector of six constraints, specifying the values of the six attributes **Sky, AirTemp, Humidity, Wind, Water** and Forecast.

A Concept Learning Task



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

TABLE 2.1
Positive and negative training examples for the target concept *EnjoySport*.

The hypothesis that Aldo enjoys his favorite sport only on cold days with high humidity is represented by the expression $<?, Cold, High, ?, ?, ?>$

A Concept Learning Task

Any concept learning task can be described by

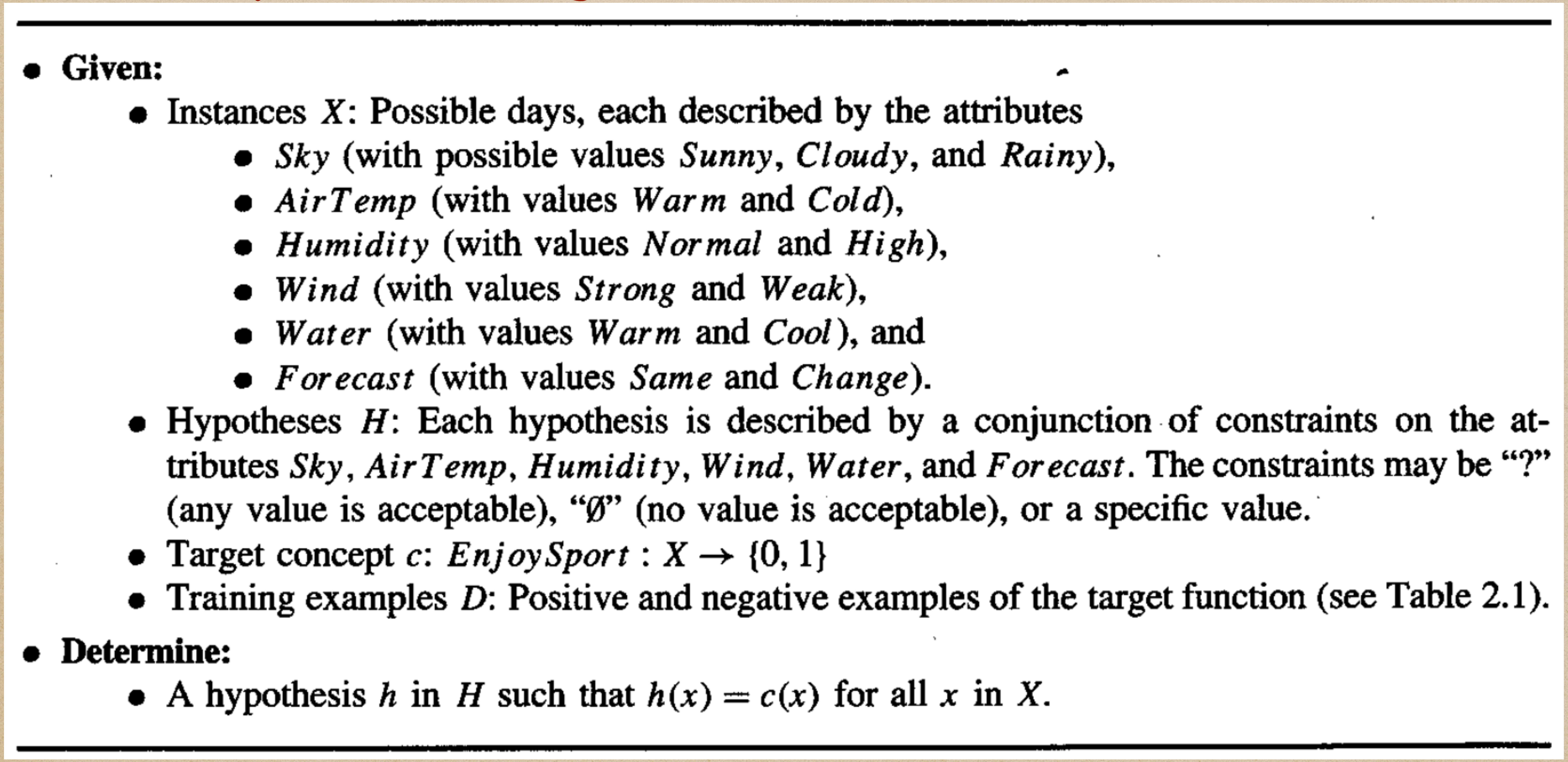
the set of instances over which the target function is defined,

the target function,

the set of candidate hypotheses considered by the learner, and

the set of available training examples.

A Concept Learning Task



- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Ø" (no value is acceptable), or a specific value.
  - Target concept c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Training examples D: Positive and negative examples of the target function (see Table 2.1).
- **Determine:**
  - A hypothesis h in H such that $h(x) = c(x)$ for all x in X.

A Concept Learning Task

While learning the target concept, the learner is presented with some training examples:

<an instance $x \in X$, c(x)>

c(x) = 1 —> positive example

c(x) = 0 —> negative example

<x,c(x)> is the training example.

A Concept Learning Task

Given a set of training examples of the target concept c, the problem faced by the learner is to hypothesize, or estimate, c.

Use the symbol

H = the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. choice of hypothesis representation.

A Concept Learning Task

In general, each hypothesis h in H represents a boolean-valued function defined over X, that is, $h : X \to \{0,1\}$.

The goal of the learner is to find a hypothesis h such that h(x)=c(x) for all $x \in X$.

The Inductive Learning Hypothesis

Notice that although the learning task is to determine a hypothesis h identical to the target concept c over the entire set of instances X, the only information available about c is its value over the training examples.

The 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.

CONCEPT LEARNING AS SEARCH

Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.

The goal of this search is to find the hypothesis that best fits the training examples.

CONCEPT LEARNING AS SEARCH

It is important to note that by selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.

CONCEPT LEARNING AS SEARCH

Consider the example **EnjoySport**:

Given that the attribute SKY has three possible values, and that **AirTemp, Humidity, Wind,Water**, and **Forecast** each have two possible values, the instance space X contains exactly 3.2.2.2.2.2=96 distinct instances.

CONCEPT LEARNING AS SEARCH

Consider the example **EnjoySport**: …

Similarly, there are **5.4.4.4.4.4=5120** syntactically distinct hypotheses within H.

Notice, however, that every hypothesis containing one or more empty symbols represents the empty set of instances; that is, it classifies every instance as negative.

CONCEPT LEARNING AS SEARCH

Therefore, the number of semantically distinct hypotheses is only 1 +(4 . 3 . 3 . 3 . 3 . 3) =973.

Our EnjoySport example is a very simple learning task, with a relatively small, finite hypothesis space. Most practical learning tasks involve much larger, sometimes infinite, hypothesis spaces.

CONCEPT LEARNING AS SEARCH

If we view learning as a search problem, then it is natural that our study of learning algorithms will examine different strategies for searching the hypothesis space.

We will be particularly interested in algorithms capable of efficiently searching very large or infinite hypothesis spaces, to find the hypotheses that best fit the training data.

General-to-Specific Ordering of Hypotheses

Many algorithms for concept learning organize the search through the hypothesis space by relying on a very useful structure that exists for any concept learning problem: a general-to-specific ordering of hypotheses.

Design learning algorithms that exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

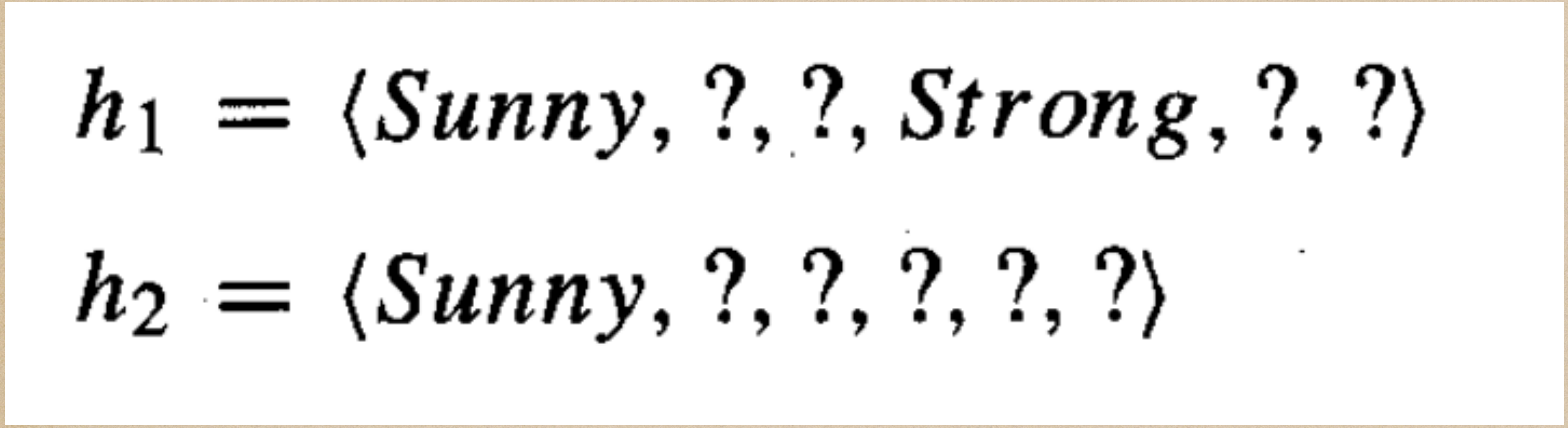
$$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$$

$$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$$

Consider the sets of instances that are classified positive by h1 and by h2. Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses



$$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$$

$$h_2 = \langle Sunny, ?, ?, ?, ?, ? \rangle$$

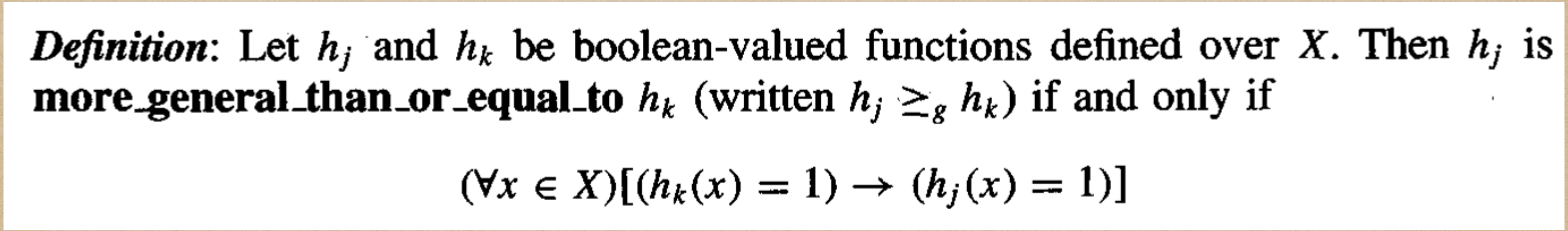
In fact, any instance classified positive by hl will also be classified positive by h2. Therefore, we say that h2 is more general than h1.

General-to-Specific Ordering of Hypotheses

This intuitive "more general than" relationship between hypotheses can be defined as: First, for any instance x inX and hypothesis h in H, we say that x satisfies h if and only if h(x)=1.

General-to-Specific Ordering of Hypotheses

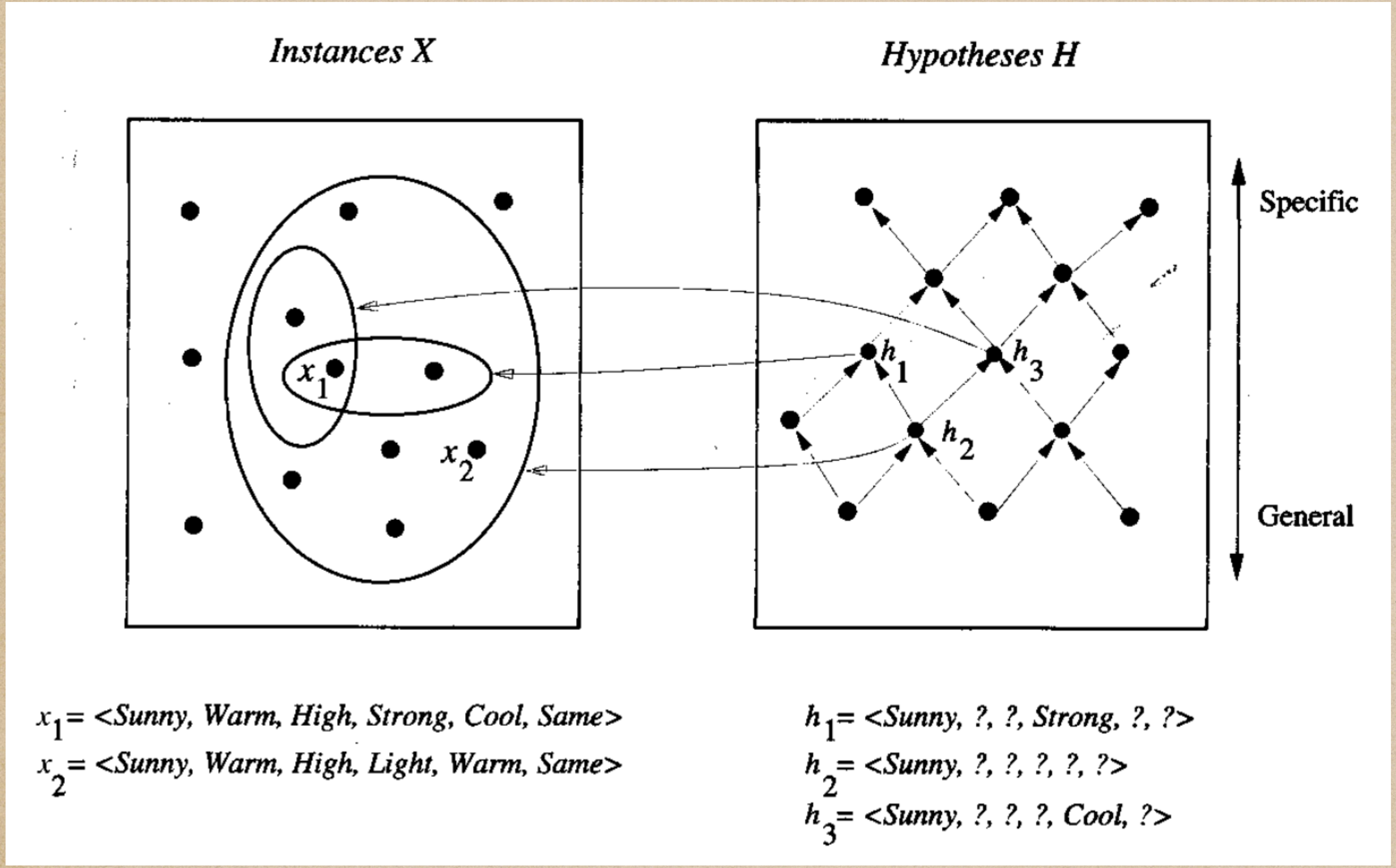
Now define the more-general-than_or.-equal relation in terms of the sets of instances that satisfy the two hypotheses: Given hypotheses hj and hk, hj is more-general-than-or-equal-to hk if and only if any instance that satisfies hk also satisfies hj.



Definition: Let h_j and h_k be boolean-valued functions defined over X. Then h_j is **more_general_than_or_equal_to** h_k (written $h_j \geq_g h_k$) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

General-to-Specific Ordering of Hypotheses

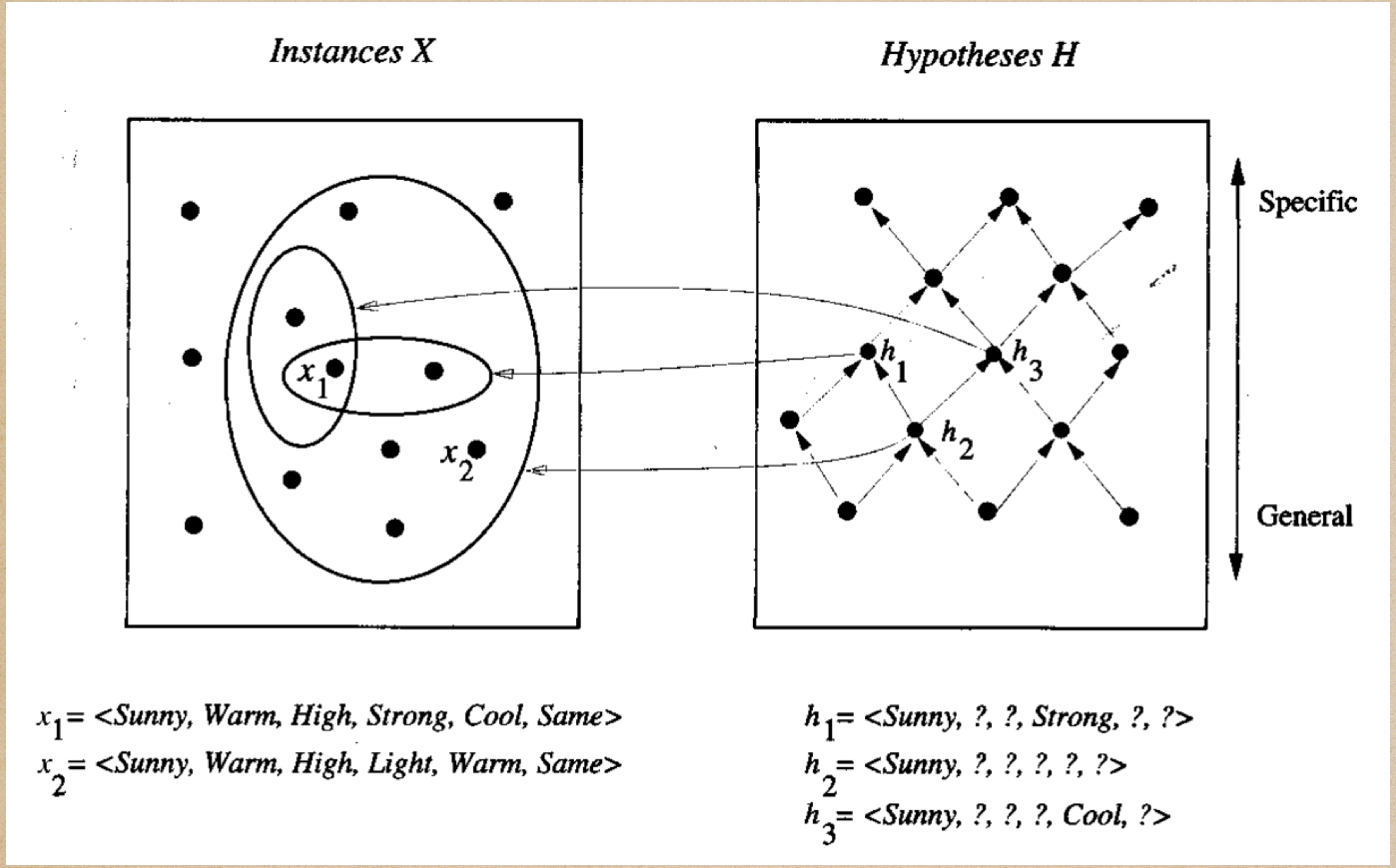


x_1 = <Sunny, Warm, High, Strong, Cool, Same>
x_2 = <Sunny, Warm, High, Light, Warm, Same>

h_1 = <Sunny, ?, ?, Strong, ?, ?>
h_2 = <Sunny, ?, ?, ?, ?, ?>
h_3 = <Sunny, ?, ?, ?, Cool, ?>

Left: the set $_x$ of all instances, Right: H, all hypotheses

General-to-Specific Ordering of Hypotheses

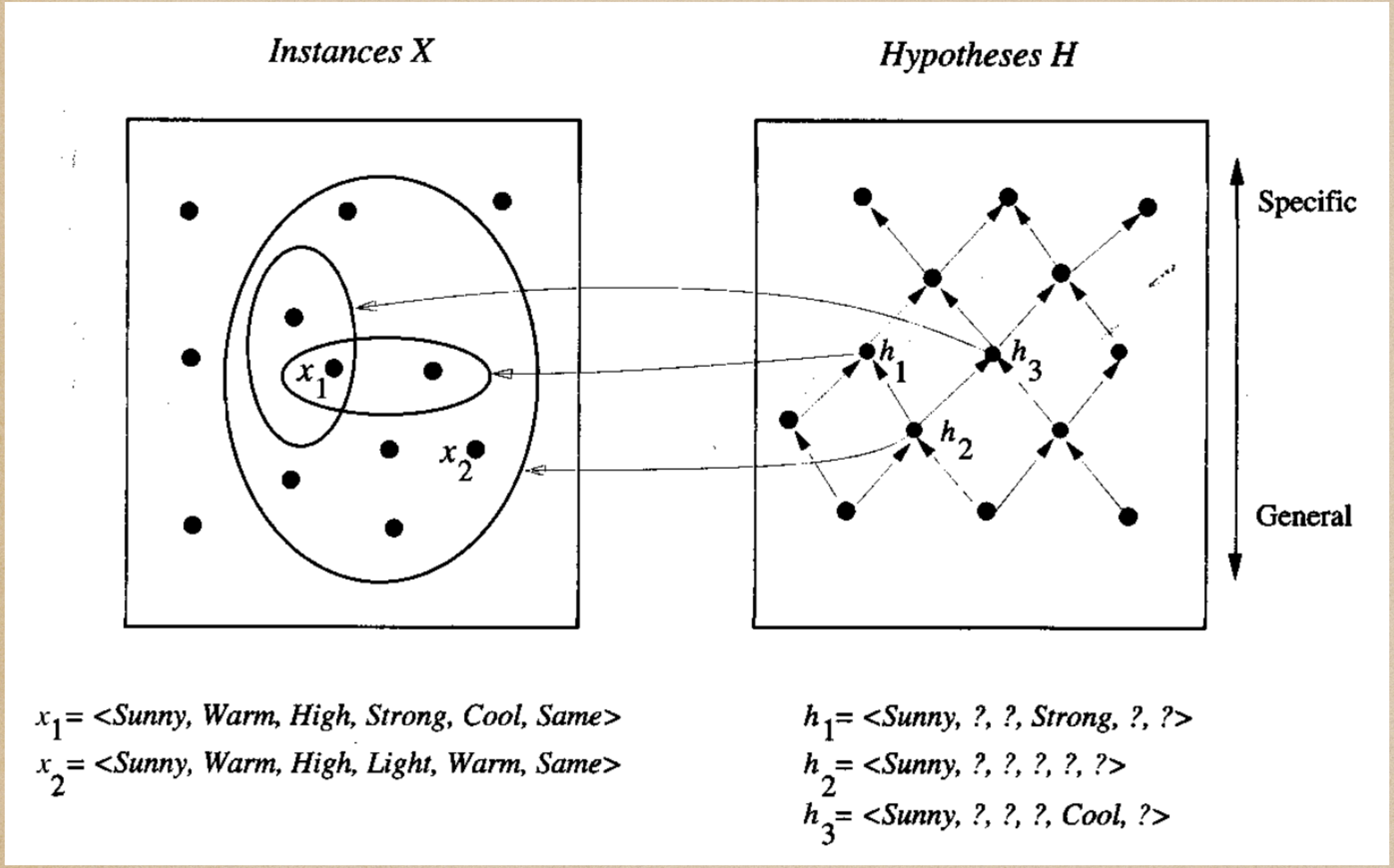


x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>

h_1= <Sunny, ?, ?, Strong, ?, ?>
h_2= <Sunny, ?, ?, ?, ?, ?>
h_3= <Sunny, ?, ?, ?, Cool, ?>

Each hypothesis corresponds to some subset of X - the subset of instances that it classifies positive.

General-to-Specific Ordering of Hypotheses

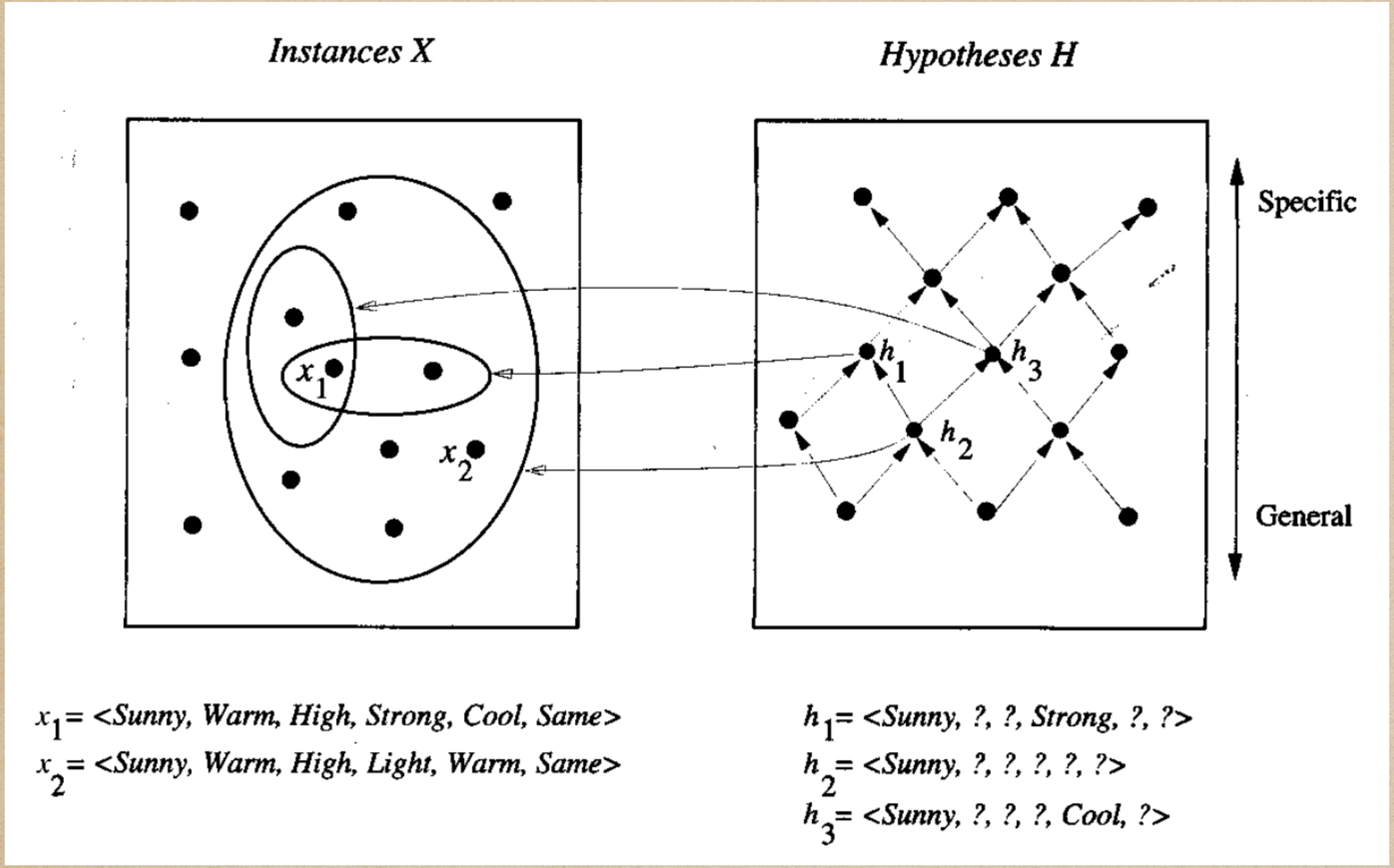


x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>

h_1= <Sunny, ?, ?, Strong, ?, ?>
h_2= <Sunny, ?, ?, ?, ?, ?>
h_3= <Sunny, ?, ?, ?, Cool, ?>

The arrow pointing toward the less general hypothesis.

General-to-Specific Ordering of Hypotheses



x_1= <*Sunny, Warm, High, Strong, Cool, Same*>
x_2= <*Sunny, Warm, High, Light, Warm, Same*>

h_1= <*Sunny, ?, ?, Strong, ?, ?*>
h_2= <*Sunny, ?, ?, ?, ?, ?*>
h_3= <*Sunny, ?, ?, ?, Cool, ?*>

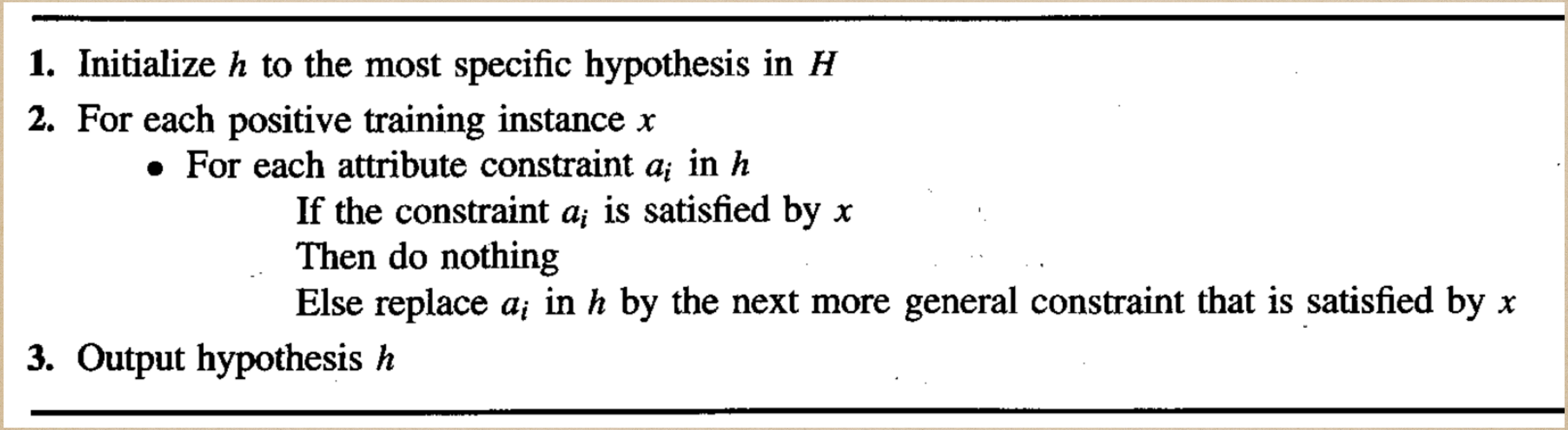
Note the subset of instances characterized by h2 subsumes the subset characterized by h1 hence h2 is more-general-than-equal-to h1.

FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

Begin with the most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.

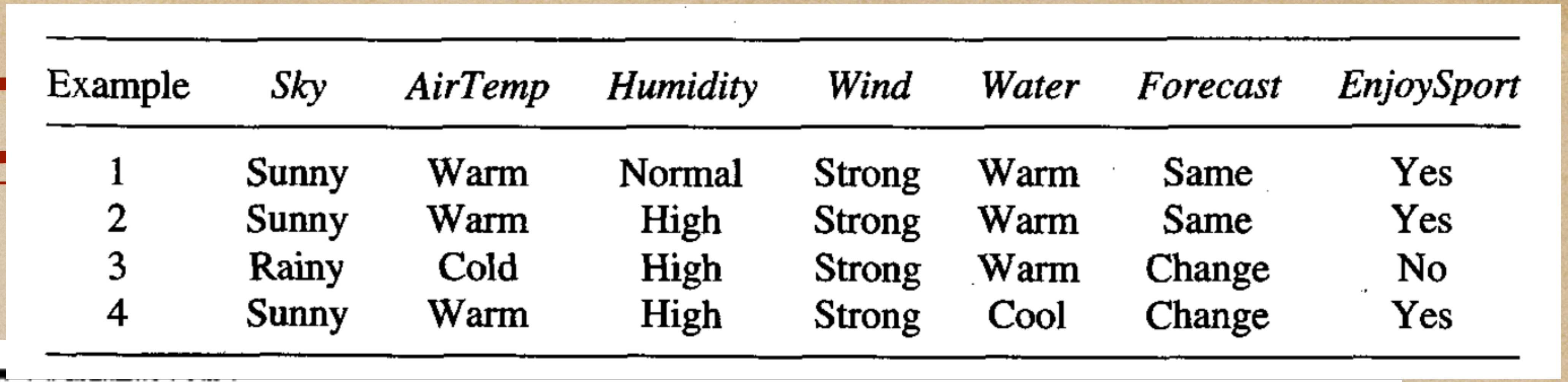
[A hypothesis "covers" a positive example if it correctly classifies the example as positive.]

FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

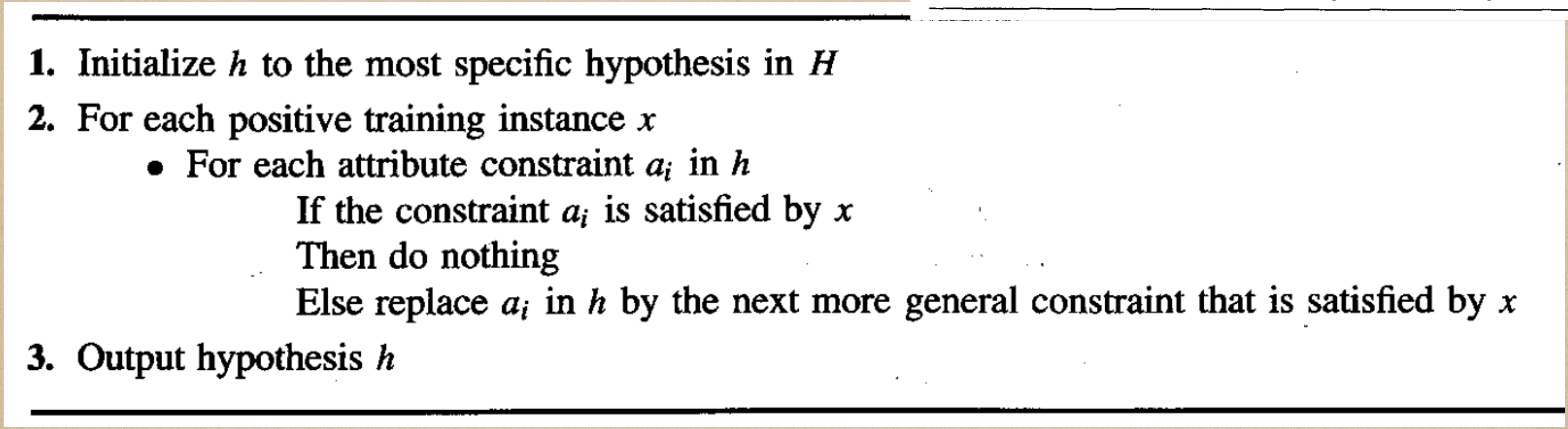


```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
       • For each attribute constraint $a_i$ in $h$
             If the constraint $a_i$ is satisfied by $x$
             Then do nothing
             Else replace $a_i$ in $h$ by the next more general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

FINDING A MAXIMALLY SPECIFI



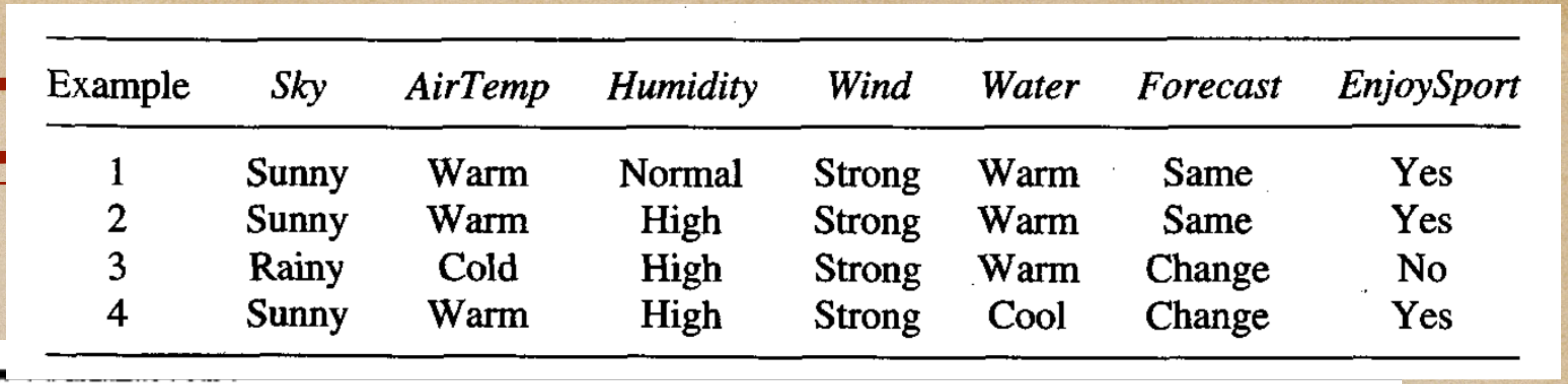
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



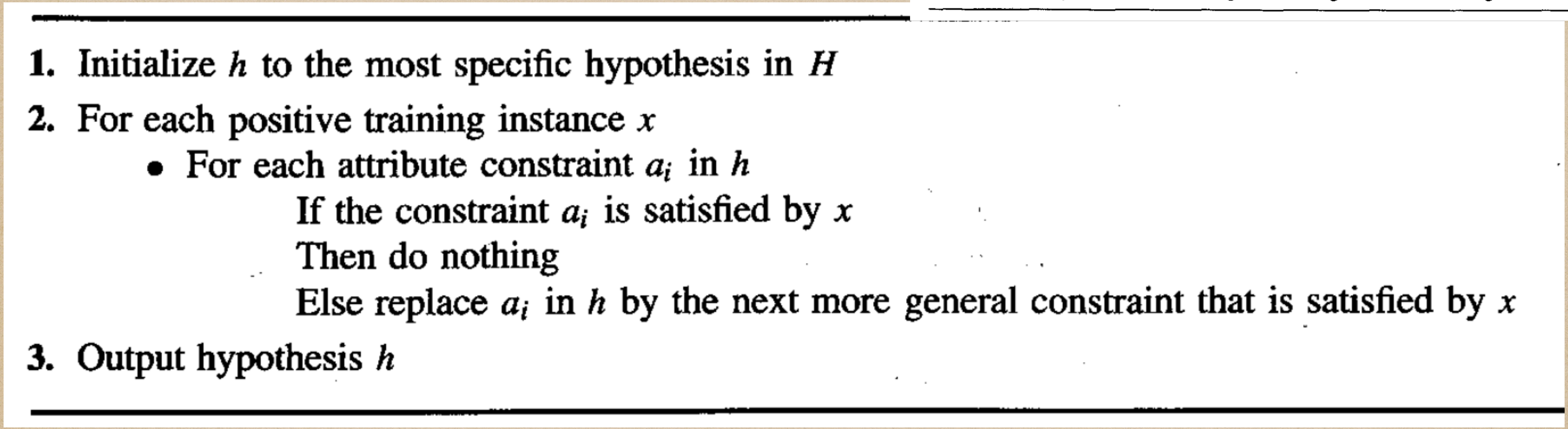
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
     • For each attribute constraint $a_i$ in $h$
           If the constraint $a_i$ is satisfied by $x$
           Then do nothing
           Else replace $a_i$ in $h$ by the next more general constraint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

Say $h \leftarrow < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$ is the specific hypothesis. From the first example above, h is too specific.

FINDING A MAXIMALLY SPECIFI



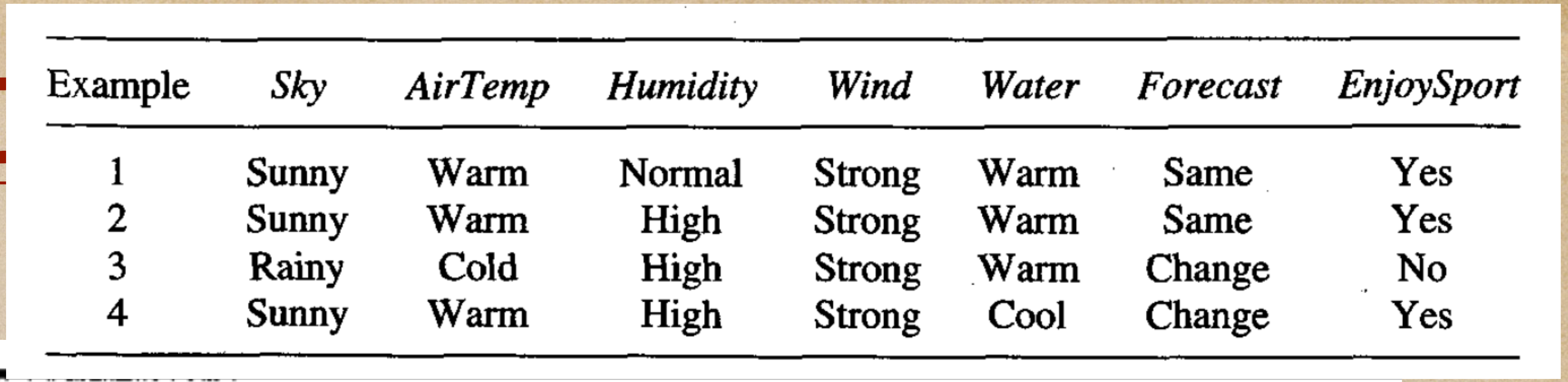
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



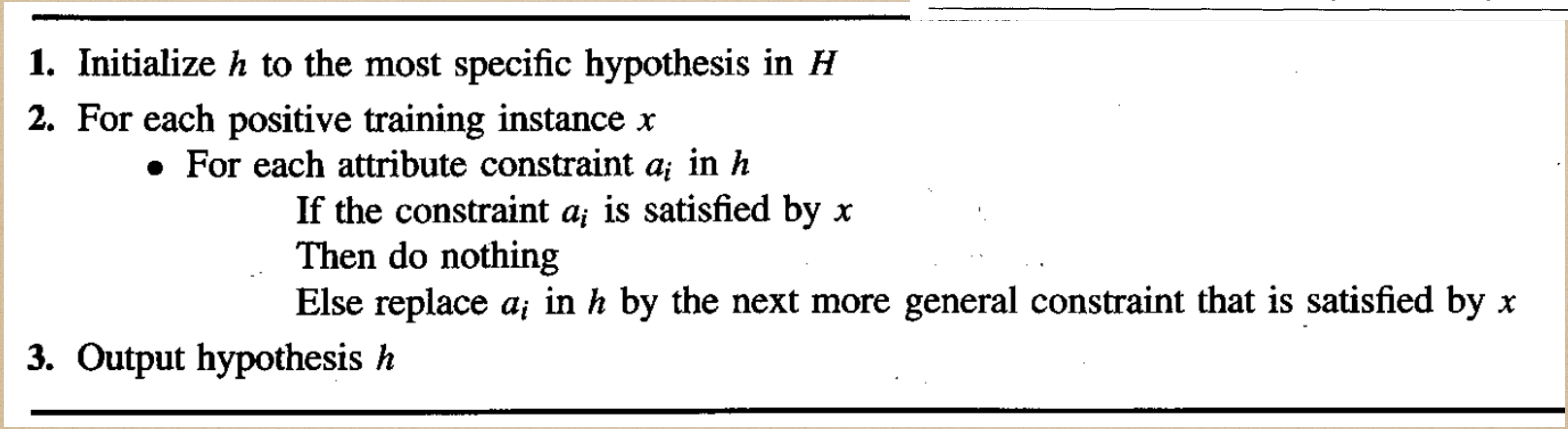
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
     • For each attribute constraint $a_i$ in $h$
           If the constraint $a_i$ is satisfied by $x$
           Then do nothing
           Else replace $a_i$ in $h$ by the next more general constraint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

$h \leftarrow < \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset >$ none of the "0" constraints in h are satisfied by this example, so each is replaced by the next more general constraint.

FINDING A MAXIMALLY SPECIFI

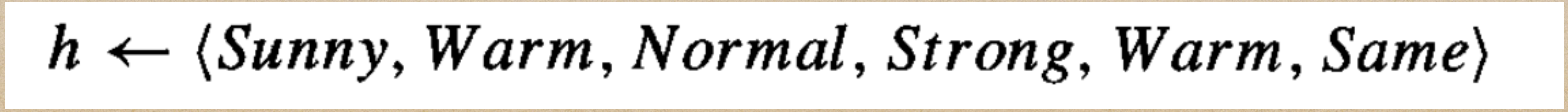


Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



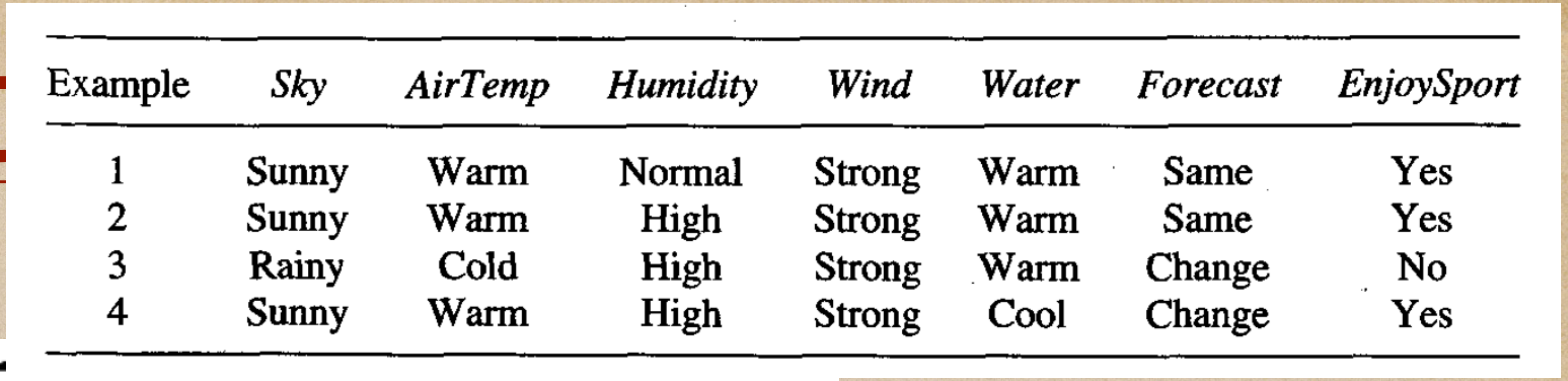
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
     • For each attribute constraint $a_i$ in $h$
           If the constraint $a_i$ is satisfied by $x$
           Then do nothing
           Else replace $a_i$ in $h$ by the next more general constraint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

… so each is replaced by the next more general constraint. namely, the attribute values for this training example

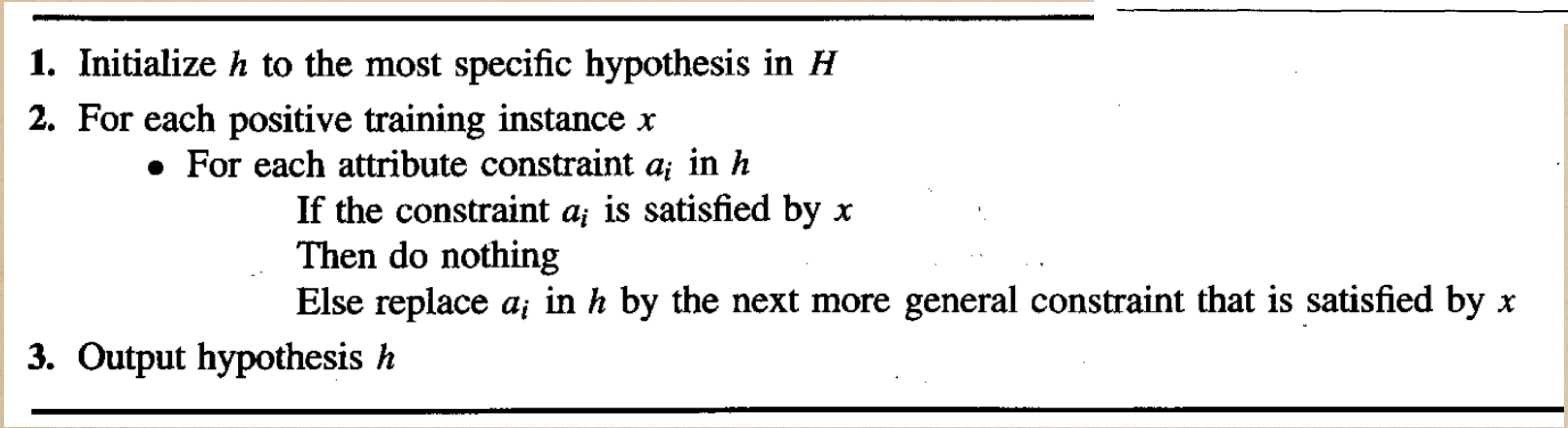


$$h \leftarrow \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$$

FINDING A MAXIMALLY SPECIFI



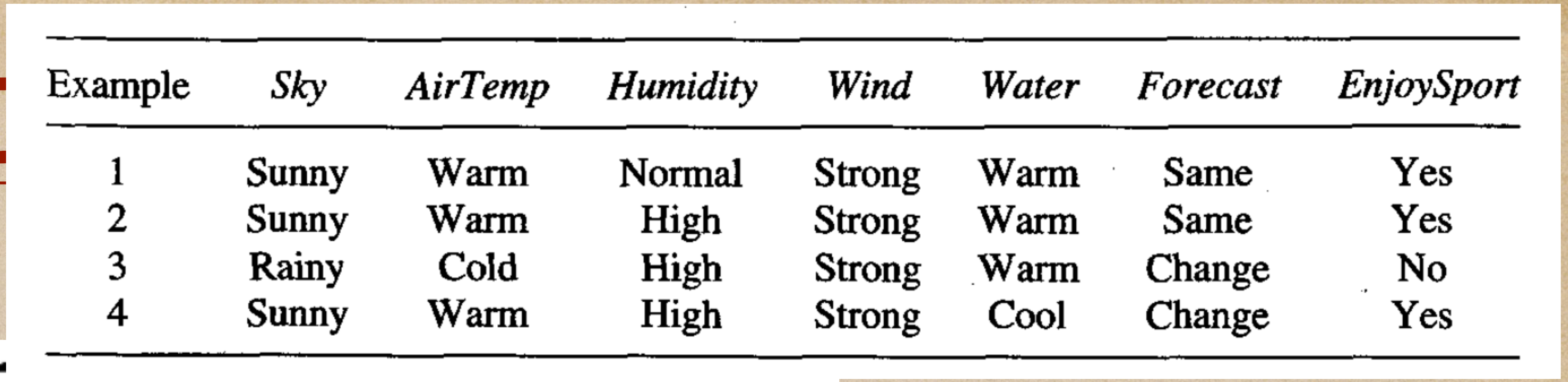
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



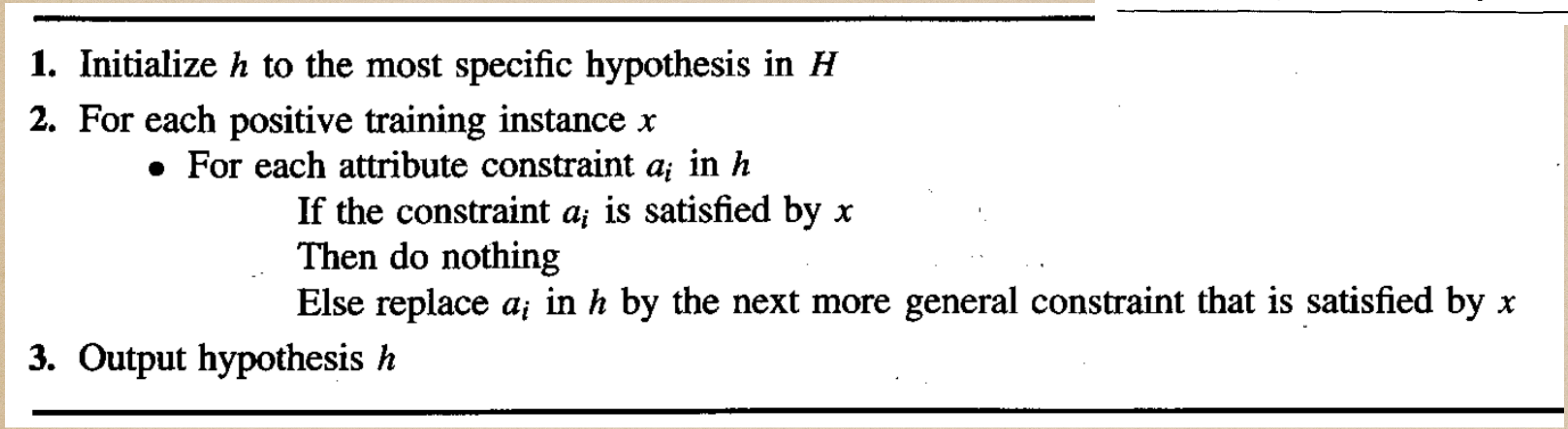
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
      • For each attribute constraint $a_i$ in $h$
            If the constraint $a_i$ is satisfied by $x$
            Then do nothing
            Else replace $a_i$ in $h$ by the next more general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

… Next, the second training example (also positive in this case) forces the algorithm to further generalize h, this time substituting a ?in place of any attribute value in h that is not satisfied by the new example.

FINDING A MAXIMALLY SPECIFI



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

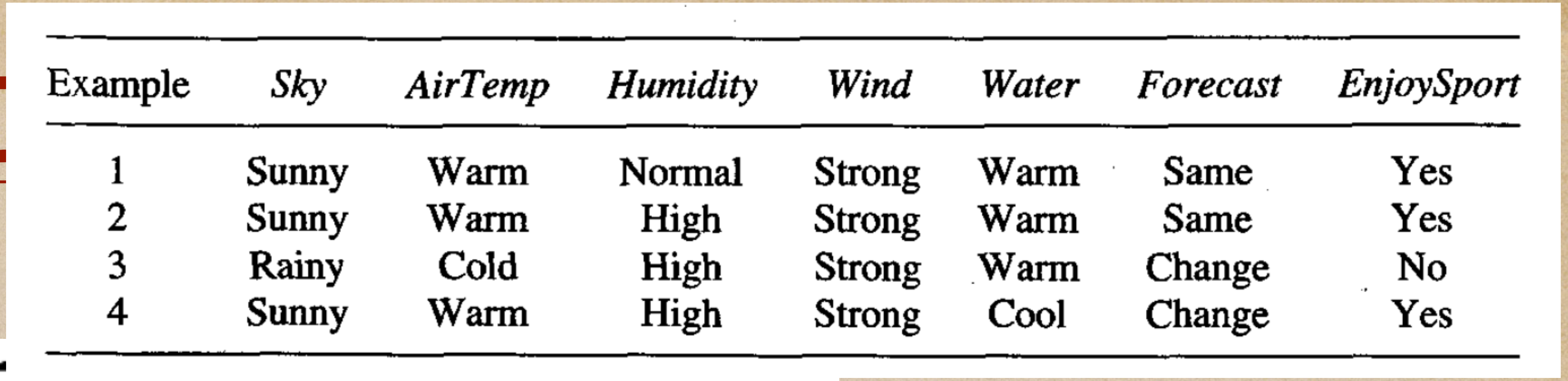


```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
       • For each attribute constraint $a_i$ in $h$
             If the constraint $a_i$ is satisfied by $x$
             Then do nothing
             Else replace $a_i$ in $h$ by the next more general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

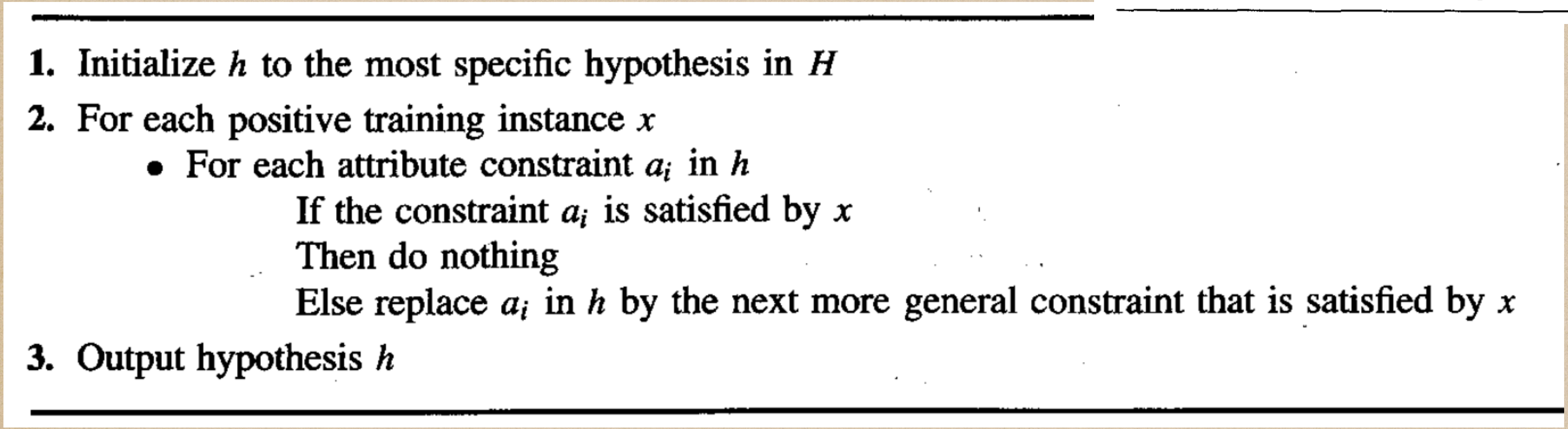
… $h \leftarrow \langle Sunny, Warm, ?, Strong, Warm, Same \rangle$ Third instance: negative, hence no changes. 4th instance

$$h \leftarrow \langle Sunny, Warm, ?, Strong, ?, ? \rangle$$

FINDING A MAXIMALLY SPECIFI



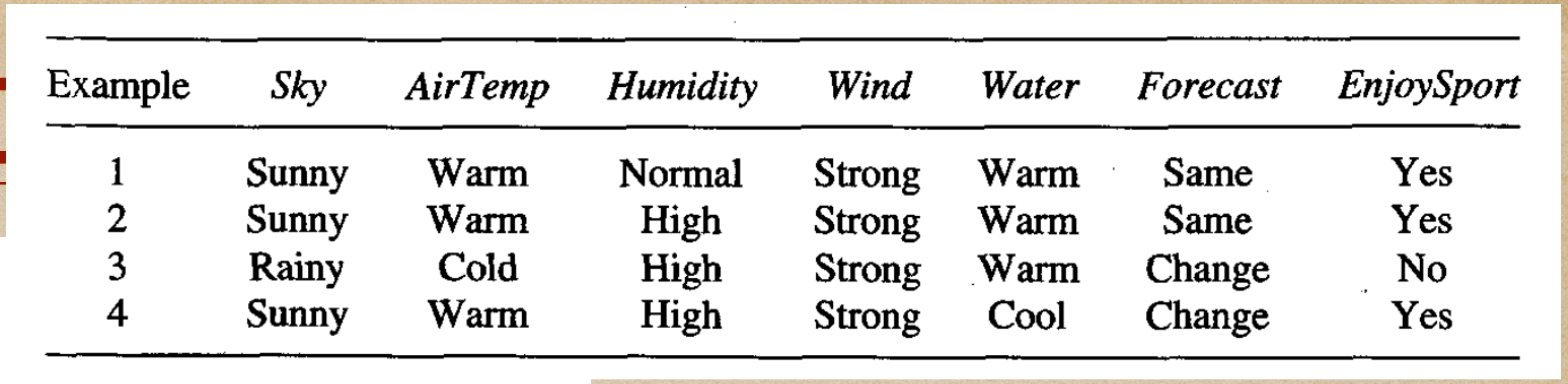
Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



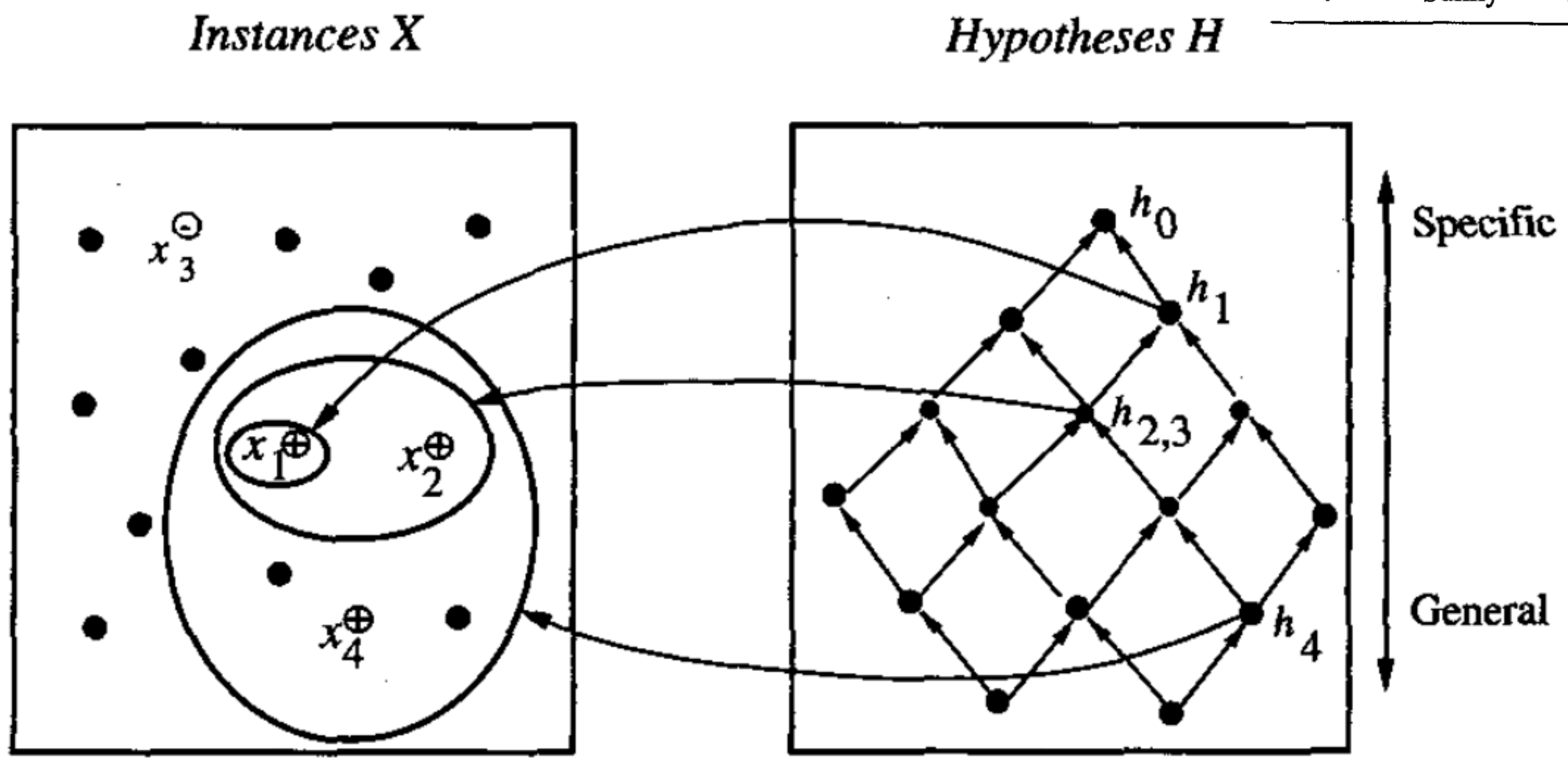
```
1. Initialize $h$ to the most specific hypothesis in $H$
2. For each positive training instance $x$
     • For each attribute constraint $a_i$ in $h$
         If the constraint $a_i$ is satisfied by $x$
         Then do nothing
         Else replace $a_i$ in $h$ by the next more general constraint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

… The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses along one chain of the partial ordering.

FINDING A MAXIMALLY SPECIFI



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



h_0 = <∅, ∅, ∅, ∅, ∅, ∅>

x_1 = *<Sunny Warm Normal Strong Warm Same>, +*	h_1 = *<Sunny Warm Normal Strong Warm Same>*
x_2 = *<Sunny Warm High Strong Warm Same>, +*	h_2 = *<Sunny Warm ? Strong Warm Same>*
x_3 = *<Rainy Cold High Strong Warm Change>, -*	h_3 = *<Sunny Warm ? Strong Warm Same>*
x_4 = *<Sunny Warm High Strong Cool Change>, +*	h_4 = *<Sunny Warm ? Strong ? ? >*