3.4 INTRODUCTION TO CONCEPT LEARNING

Concept learning is a learning strategy of acquiring abstract knowledge or inferring a general concept or deriving a category from the given training samples. It is a process of abstraction and generalization from the data.

Concept learning helps to classify an object that has a set of common, relevant features. Thus, it helps a learner compare and contrast categories based on the similarity and association of positive and negative instances in the training data to classify an object. The learner tries to simplify by observing the common features from the training samples and then apply this simplified model to the future samples. This task is also known as learning from experience.

Each concept or category obtained by learning is a Boolean valued function which takes a true or false value. For example, humans can identify different kinds of animals based on common relevant features and categorize all animals based on specific sets of features. The special features that distinguish one animal from another can be called as a concept. This way of learning categories for object and to recognize new instances of those categories is called as concept learning. It is formally defined as inferring a Boolean valued function by processing training instances.

Concept learning requires three things:

1. Input – Training dataset which is a set of training instances, each labeled with the name of a concept or category to which it belongs. Use this past experience to train and build the model.
2. Output – Target concept or Target function f. It is a mapping function $f(x)$ from input x to output y. It is to determine the specific features or common features to identify an object. In other words, it is to find the hypothesis to determine the target concept. For e.g., the specific set of features to identify an elephant from all animals.
3. Test – New instances to test the learned model.

Formally, Concept learning is defined as–"Given a set of hypotheses, the learner searches through the hypothesis space to identify the best hypothesis that matches the target concept".

Consider the following set of training instances shown in Table 3.1.

Table 3.1: Sample Training Instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

Here, in this set of training instances, the independent attributes considered are 'Horns', 'Tail', 'Tusks', 'Paws', 'Fur', 'Color', 'Hooves' and 'Size'. The dependent attribute is 'Elephant'. The target concept is to identify the animal to be an Elephant.

Let us now take this example and understand further the concept of hypothesis.

Target Concept: Predict the type of animal - For example –'Elephant'.

3.4.1 Representation of a Hypothesis

A *hypothesis* '*h*' approximates a target function '*f*' to represent the relationship between the independent attributes and the dependent attribute of the training instances. The hypothesis is the predicted approximate model that best maps the inputs to outputs. Each hypothesis is represented as a conjunction of attribute conditions in the antecedent part.

For example, (Tail = Short) ∧ (Color = Black)....

The set of hypothesis in the search space is called as hypotheses. Hypotheses are the plural form of hypothesis. Generally '*H*' is used to represent the hypotheses and '*h*' is used to represent a candidate hypothesis.

Each attribute condition is the constraint on the attribute which is represented as attribute-value pair. In the antecedent of an attribute condition of a hypothesis, each attribute can take value as either '?' or 'φ' or can hold a single value.

- "?" denotes that the attribute can take any value [e.g., Color = ?]
- "φ" denotes that the attribute cannot take any value, i.e., it represents a null value [e.g., Horns = φ]
- Single value denotes a specific single value from acceptable values of the attribute, i.e., the attribute 'Tail' can take a value as 'short' [e.g., Tail = Short]

For example, a hypothesis '*h*' will look like,

	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size
h =	<No	?	Yes	?	?	Black	No	Medium>

Given a test instance *x*, we say $h(x) = 1$, if the test instance *x* satisfies this hypothesis *h*.

The training dataset given above has 5 training instances with 8 independent attributes and one dependent attribute. Here, the different hypotheses that can be predicted for the target concept are,

	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size
h =	<No	?	Yes	?	?	Black	No	Medium>
				(or)				
h =	<No	?	Yes	?	?	Black	No	Big>

The task is to predict the best hypothesis for the target concept (an elephant). The most general hypothesis can allow any value for each of the attribute.

It is represented as:

<?, ?, ?, ?, ?, ?, ?, ?>. This hypothesis indicates that any animal can be an elephant.

The most specific hypothesis will not allow any value for each of the attribute <φ, φ, φ, φ, φ, φ, φ, φ>. This hypothesis indicates that no animal can be an elephant.

The target concept mentioned in this example is to identify the conjunction of specific features from the training instances to correctly identify an elephant.

Example 3.1: Explain Concept Learning Task of an Elephant from the dataset given in Table 3.1.

Given,

Input: 5 instances each with 8 attributes

Target concept/function 'c': Elephant → {Yes, No}

Hypotheses H: Set of hypothesis each with conjunctions of literals as propositions [i.e., each literal is represented as an attribute-value pair]

Solution: The hypothesis 'h' for the concept learning task of an Elephant is given as:

h = <No Short Yes ? ? Black No ?>

This hypothesis h is expressed in propositional logic form as below:

(Horns = No) ∧ (Tail = Short) ∧ (Tusks = Yes) ∧ (Paws = ?) ∧ (Fur = ?) ∧ (Color = Black) ∧ (Hooves = No) ∧ (Size = ?)

Output: Learn the hypothesis 'h' to predict an 'Elephant' such that for a given test instance x,

$$h(x) = c(x)$$

This hypothesis produced is also called as concept description which is a model that can be used to classify subsequent instances.

Thus, concept learning can also be called as *Inductive Learning* that tries to induce a general function from specific training instances. This way of learning a hypothesis that can produce an approximate target function with a sufficiently large set of training instances can also approximately classify other unobserved instances and is called as inductive learning hypothesis. We can only determine an approximate target function because it is very difficult to find an exact target function with the observed training instances. That is why a hypothesis is an approximate target function that best maps the inputs to outputs.

3.4.2 Hypothesis Space

Hypothesis space is the set of all possible hypotheses that approximates the target function f. In other words, the set of all possible approximations of the target function can be defined as hypothesis space. From this set of hypotheses in the hypothesis space, a machine learning algorithm would determine the best possible hypothesis that would best describe the target function or best fit the outputs. Generally, a hypothesis representation language represents a larger hypothesis space. Every machine learning algorithm would represent the hypothesis space in a different manner about the function that maps the input variables to output variables. For example, a regression algorithm represents the hypothesis space as a linear function whereas a decision tree algorithm represents the hypothesis space as a tree.

The set of hypotheses that can be generated by a learning algorithm can be further reduced by specifying a language bias.

The subset of hypothesis space that is consistent with all-observed training instances is called as **Version Space.** Version space represents the only hypotheses that are used for the classification.

For example, each of the attribute given in the Table 3.1 has the following possible set of values.

Horns - Yes, No

Tail - Long, Short

Tusks - Yes, No

Paws - Yes, No

Fur - Yes, No

Color - Brown, Black, White

Hooves - Yes, No

Size - Medium, Big

Considering these values for each of the attribute, there are $(2 \times 2 \times 2 \times 2 \times 2 \times 3 \times 2 \times 2) = 384$ distinct instances covering all the 5 instances in the training dataset.

So, we can generate $(4 \times 4 \times 4 \times 4 \times 4 \times 5 \times 4 \times 4) = 81{,}920$ distinct hypotheses when including two more values [?, φ] for each of the attribute. However, any hypothesis containing one or more φ symbols represents the empty set of instances; that is, it classifies every instance as negative instance. Therefore, there will be $(3 \times 3 \times 3 \times 3 \times 3 \times 4 \times 3 \times 3 + 1) = 8{,}749$ distinct hypotheses by including only '?' for each of the attribute and one hypothesis representing the empty set of instances. Thus, the hypothesis space is much larger and hence we need efficient learning algorithms to search for the best hypothesis from the set of hypotheses.

Hypothesis ordering is also important wherein the hypotheses are ordered from the most specific one to the most general one in order to restrict searching the hypothesis space exhaustively.

3.4.3 Heuristic Space Search

Heuristic search is a search strategy that finds an optimized hypothesis/solution to a problem by iteratively improving the hypothesis/solution based on a given heuristic function or a cost measure. Heuristic search methods will generate a possible hypothesis that can be a solution in

the hypothesis space or a path from the initial state. This hypothesis will be tested with the target function or the goal state to see if it is a real solution. If the tested hypothesis is a real solution, then it will be selected. This method generally increases the efficiency because it is guaranteed to find a better hypothesis but may not be the best hypothesis. It is useful for solving tough problems which could not solved by any other method. The typical example problem solved by heuristic search is the travelling salesman problem.

Several commonly used heuristic search methods are hill climbing methods, constraint satisfaction problems, best-first search, simulated-annealing, A* algorithm, and genetic algorithms.

3.4.4 Generalization and Specialization

In order to understand about how we construct this concept hierarchy, let us apply this general principle of generalization/specialization relation. By generalization of the most specific hypothesis and by specialization of the most general hypothesis, the hypothesis space can be searched for an approximate hypothesis that matches all positive instances but does not match any negative instance.

Searching the Hypothesis Space

There are two ways of learning the hypothesis, consistent with all training instances from the large hypothesis space.

1. Specialization – General to Specific learning
2. Generalization – Specific to General learning

Scan for information on *'Additional Examples on Generalization and Specialization'*

Generalization – Specific to General Learning This learning methodology will search through the hypothesis space for an approximate hypothesis by generalizing the most specific hypothesis.

Example 3.2: Consider the training instances shown in Table 3.1 and illustrate Specific to General Learning.

Solution: We will start from all false or the most specific hypothesis to determine the most restrictive specialization. Consider only the positive instances and generalize the most specific hypothesis. Ignore the negative instances.

This learning is illustrated as follows:

The most specific hypothesis is taken now, which will not classify any instance to true.

h = <φ φ φ φ φ φ φ φ>

Read the first instance $I1$, to generalize the hypothesis h so that this positive instance can be classified by the hypothesis $h1$.

$I1$:	No	Short	Yes	No	No	Black	No	Big	**Yes (Positive instance)**
$h1$ =	<No	Short	Yes	No	No	Black	No	Big>	

When reading the second instance *I*2, it is a negative instance, so ignore it.

*I*2:	Yes	Short	No	No	No	Brown	Yes	Medium	**No (Negative instance)**
*h*2 = <No	Short	Yes	No	No	Black	No	Big>		

Similarly, when reading the third instance *I*3, it is a positive instance so generalize *h*2 to *h*3 to accommodate it. The resulting *h*3 is generalized.

*I*3:	No	Short	Yes	No	No	Black	No	Medium	**Yes (Positive instance)**
*h*3 = <No	Short	Yes	No	No	Black	No	?>		

Ignore *I*4 since it is a negative instance.

*I*4:	No	Long	No	Yes	Yes	White	No	Medium	**No (Negative instance)**
*h*4 = <No	Short	Yes	No	No	Black	No	?>		

When reading the fifth instance *I*5, *h*4 is further generalized to *h*5.

*I*5:	No	Short	Yes	Yes	Yes	Black	No	Big	**Yes (Positive instance)**
*h*5 = <No	Short	Yes	?	?	Black	No	?>		

Now, after observing all the positive instances, an approximate hypothesis *h*5 is generated which can now classify any subsequent positive instance to true.

Specialization – General to Specific Learning This learning methodology will search through the hypothesis space for an approximate hypothesis by specializing the most general hypothesis.

Example 3.3: Illustrate learning by Specialization – General to Specific Learning for the data instances shown in Table 3.1.

Solution: Start from the most general hypothesis which will make true all positive and negative instances.

Initially,

h = <?	?	?	?	?	?	?	?>	

h is more general to classify all instances to true.

*I*1:	No	Short	Yes	No	No	Black	No	Big	**Yes (Positive instance)**
*h*1 = <?	?	?	?	?	?	?	?>		

*I*2 :	Yes	Short	No	No	No	Brown	Yes	Medium	**No (Negative instance)**
*h*2 = <No	?	?	?	?	?	?	?>		
	<?	?	Yes	?	?	?	?	?>	
	<?	?	?	?	?	Black	?	?>	
	<?	?	?	?	?	?	No	?>	
	<?	?	?	?	?	?	?	Big>	

*h*2 imposes constraints so that it will not classify a negative instance to true.

*I*3:	No	Short	Yes	No	No	Black	No	Medium	**Yes (Positive instance)**
*h*3 = <No	?	?	?	?	?	?	?>		
	<?	?	Yes	?	?	?	?	?>	

	<?	?	?	?	?	Black	?	?>
	<?	?	?	?	?	?	No	?>
	<?	?	?	?	?	?	?	Big>
*I*4:	No	Long	No	Yes	Yes	White	No	Medium **No (Negative instance)**
*h*4 =	<?	?	Yes	?	?	?	?	?>
	<?	?	?	?	?	Black	?	?>
	<?	?	?	?	?	?	?	Big>

Remove any hypothesis inconsistent with this negative instance.

*I*5:	No	Short	Yes	Yes	Yes	Black	No	Big **Yes (Positive instance)**
*h*5 =	<?	?	Yes	?	?	?	?	?>
	<?	?	?	?	?	Black	?	?>
	<?	?	?	?	?	?	?	Big>

Thus, *h*5 is the hypothesis space generated which will classify the positive instances to true and negative instances to false.

3.4.5 Hypothesis Space Search by Find-S Algorithm

Find-S algorithm is guaranteed to converge to the most specific hypothesis in ***H*** that is consistent with the positive instances in the training dataset. Obviously, it will also be consistent with the negative instances. Thus, this algorithm considers only the positive instances and eliminates negative instances while generating the hypothesis. It initially starts with the most specific hypothesis.

Algorithm 3.1: Find-S

Input: Positive instances in the Training dataset

Output: Hypothesis '*h*'

1. Initialize '*h*' to the most specific hypothesis.

   $h = <\varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \ldots.>$

2. Generalize the initial hypothesis for the first positive instance [Since '*h*' is more specific].
3. For each subsequent instances:

   If it is a positive instance,

   Check for each attribute value in the instance with the hypothesis '*h*'.

   If the attribute value is the same as the hypothesis value, then do nothing.

   Else if the attribute value is different than the hypothesis value, change it to '?' in '*h*'.

   Else if it is a negative instance,

   Ignore it.

Example 3.4: Consider the training dataset of 4 instances shown in Table 3.2. It contains the details of the performance of students and their likelihood of getting a job offer or not in their final semester. Apply the Find-S algorithm.

Table 3.2: Training Dataset

CGPA	Interactiveness	Practical Knowledge	Communication Skills	Logical Thinking	Interest	Job Offer
≥9	Yes	Excellent	Good	Fast	Yes	Yes
≥9	Yes	Good	Good	Fast	Yes	Yes
≥8	No	Good	Good	Fast	No	No
≥9	Yes	Good	Good	Slow	No	Yes

Solution:

Step 1: Initialize '*h*' to the most specific hypothesis. There are 6 attributes, so for each attribute, we initially fill 'φ' in the initial hypothesis '*h*'.

h = <φ φ φ φ φ φ>

Step 2: Generalize the initial hypothesis for the first positive instance. *I*1 is a positive instance, so generalize the most specific hypothesis '*h*' to include this positive instance. Hence,

*I*1: ≥9 Yes Excellent Good Fast Yes **Positive instance**

h = <≥9 Yes Excellent Good Fast Yes>

Step 3: Scan the next instance *I*2, since *I*2 is a positive instance. Generalize '*h*' to include positive instance *I*2. For each of the non-matching attribute value in '*h*' put a '?' to include this positive instance. The third attribute value is mismatching in '*h*' with *I*2, so put a '?'.

*I*2: ≥9 Yes Good Good Fast Yes **Positive instance**

h = <≥9 Yes ? Good Fast Yes>

Now,scan *I*3. Since it is a negative instance, ignore it. Hence, the hypothesis remains the same without any change after scanning *I*3.

I3: ≥8 No Good Good Fast No **Negative instance**

h = <≥9 Yes ? Good Fast Yes>

Now scan *I*4. Since it is a positive instance, check for mismatch in the hypothesis '*h*' with *I*4. The 5th and 6th attribute value are mismatching, so add '?' to those attributes in '*h*'.

I4: ≥9 Yes Good Good Slow No **Positive instance**

h = <≥9 Yes ? Good ? ?>

Now, the final hypothesis generated with Find-S algorithm is:

h = <≥9 Yes ? Good ? ?>

It includes all positive instances and obviously ignores any negative instance.

Limitations of Find-S Algorithm

1. Find-S algorithm tries to find a hypothesis that is consistent with positive instances, ignoring all negative instances. As long as the training dataset is consistent, the hypothesis found by this algorithm may be consistent.
2. The algorithm finds only one unique hypothesis, wherein there may be many other hypotheses that are consistent with the training dataset.

3. Many times, the training dataset may contain some errors; hence such inconsistent data instances can mislead this algorithm in determining the consistent hypothesis since it ignores negative instances.

Hence, it is necessary to find the set of hypotheses that are consistent with the training data including the negative examples. To overcome the limitations of Find-S algorithm, Candidate Elimination algorithm was proposed to output the set of all hypotheses consistent with the training dataset.

3.4.6 Version Spaces

The version space contains the subset of hypotheses from the hypothesis space that is consistent with all training instances in the training dataset.

Scan for information on *'Additional Examples on Version Spaces'*

List-Then-Eliminate Algorithm

The principle idea of this learning algorithm is to initialize the version space to contain all hypotheses and then eliminate any hypothesis that is found inconsistent with any training instances. Initially, the algorithm starts with a version space to contain all hypotheses scanning each training instance. The hypotheses that are inconsistent with the training instance are eliminated. Finally, the algorithm outputs the list of remaining hypotheses that are all consistent.

Algorithm 3.2: List-Then-Eliminate

Input: Version Space – a list of all hypotheses

Output: Set of consistent hypotheses

1. Initialize the version space with a list of hypotheses.
2. For each training instance,
   - remove from version space any hypothesis that is inconsistent.

This algorithm works fine if the hypothesis space is finite but practically it is difficult to deploy this algorithm. Hence, a variation of this idea is introduced in the Candidate Elimination algorithm.

Version Spaces and the Candidate Elimination Algorithm

Version space learning is to generate all consistent hypotheses around. This algorithm computes the version space by the combination of the two cases namely,

- Specific to General learning – Generalize S to include the positive example
- General to Specific learning – Specialize G to exclude the negative example

Using the Candidate Elimination algorithm, we can compute the version space containing all (and only those) hypotheses from H that are consistent with the given observed sequence of training instances. The algorithm defines two boundaries called *'general boundary'* which is a set of all hypotheses that are the most general and *'specific boundary'* which is a set of all hypotheses that are the most specific. Thus, the algorithm limits the version space to contain only those hypotheses that are most general and most specific. Thus, it provides a compact representation of List-then algorithm.

Algorithm 3.3: Candidate Elimination

Input: Set of instances in the Training dataset

Output: Hypothesis G and S

1. Initialize G, to the maximally general hypotheses.
2. Initialize S, to the maximally specific hypotheses.
   - Generalize the initial hypothesis for the first positive instance.
3. For each subsequent new training instance,
   - If the instance is **positive**,
     - Generalize S to include the positive instance,
       - Check the attribute value of the positive instance and S,
         - If the attribute value of positive instance and S are different, fill that field value with '?'.
         - If the attribute value of positive instance and S are same, then do no change.
     - Prune G to exclude all inconsistent hypotheses in G with the positive instance.
   - If the instance is **negative,**
     - Specialize G to exclude the negative instance,
       - Add to G all minimal specializations to exclude the negative example and be consistent with S.
         - If the attribute value of S and the negative instance are different, then fill that attribute value with S value.
         - If the attribute value of S and negative instance are same, no need to update 'G' and fill that attribute value with '?'.
     - Remove from S all inconsistent hypotheses with the negative instance.

Generating Positive Hypothesis 'S' If it is a positive example, refine S to include the positive instance. We need to generalize S to include the positive instance. The hypothesis is the conjunction of 'S' and positive instance. When generalizing, for the first positive instance, add to S all minimal generalizations such that S is filled with attribute values of the positive instance. For the subsequent positive instances scanned, check the attribute value of the positive instance and S obtained in the

previous iteration. If the attribute values of positive instance and S are different, fill that field value with a '?'. If the attribute values of positive instance and S are same, no change is required.

If it is a negative instance, it skips.

Generating Negative Hypothesis 'G' If it is a negative instance, refine G to exclude the negative instance. Then, prune G to exclude all inconsistent hypotheses in G with the positive instance. The idea is to add to G all minimal specializations to exclude the negative instance and be consistent with the positive instance. Negative hypothesis indicates general hypothesis.

If the attribute values of positive and negative instances are different, then fill that field with positive instance value so that the hypothesis does not classify that negative instance as true. If the attribute values of positive and negative instances are same, then no need to update 'G' and fill that attribute value with a '?'.

Generating Version Space – [Consistent Hypothesis] We need to take the combination of sets in 'G' and check that with 'S'. When the combined set fields are matched with fields in 'S', then only that is included in the version space as consistent hypothesis.

Example 3.4: Consider the same set of instances from the training dataset shown in Table 3.3 and generate version space as consistent hypothesis.

Solution:

Step 1: Initialize 'G' boundary to the maximally general hypotheses,

G = <? ? ? ? ? ?>

Step 2: Initialize 'S' boundary to the maximally specific hypothesis. There are 6 attributes, so for each attribute, we initially fill 'φ' in the hypothesis 'S'.

S = <φ φ φ φ φ φ>

Generalize the initial hypothesis for the first positive instance. I1 is a positive instance; so generalize the most specific hypothesis 'S' to include this positive instance. Hence,

I1:	≥9	Yes	Excellent	Good	Fast	Yes	**Positive instance**
S1 = <	≥9	Yes	Excellent	Good	Fast	Yes>	
G1 = <?		?	?	?	?	?>	

Step 3:

Iteration 1

Scan the next instance I2. Since I2 is a positive instance, generalize 'S1' to include positive instance I2. For each of the non-matching attribute value in 'S1', put a '?' to include this positive instance. The third attribute value is mismatching in 'S1' with I2, so put a '?'.

I2:	≥9	Yes	Good	Good	Fast	Yes	**Positive instance**
S2 = <	≥9	Yes	?	Good	Fast	Yes>	

Prune G1 to exclude all inconsistent hypotheses with the positive instance. Since G1 is consistent with this positive instance, there is no change. The resulting G2 is,

G2 = <? ? ? ? ? ?>

Iteration 2

Now Scan *I*3,

*I*3: ≥8 No Good Good Fast No **Negative instance**

Since it is a negative instance, specialize *G*2 to exclude the negative example but stay consistent with *S*2. Generate hypothesis for each of the non-matching attribute value in *S*2 and fill with the attribute value of *S*2. In those generated hypotheses, for all matching attribute values, put a '?'. The first, second and 6th attribute values do not match, hence '3' hypotheses are generated in *G*3.

There is no inconsistent hypothesis in *S*2 with the negative instance, hence *S*3 remains the same.

*G*3 = < ≥9 ? ? ? ? ?>

<? Yes ? ? ? ?>

<? ? ? ? ? Yes>

*S*3 = < ≥9 Yes ? Good Fast Yes>

Iteration 3

Now Scan *I*4. Since it is a positive instance, check for mismatch in the hypothesis '*S*3' with *I*4. The 5th and 6th attribute value are mismatching, so add '?' to those attributes in '*S*4'.

*I*4: ≥9 Yes Good Good Slow No **Positive instance**

*S*4 = < ≥9 Yes ? Good ? ?>

Prune *G*3 to exclude all inconsistent hypotheses with the positive instance *I*4.

*G*3 = < ≥9 ? ? ? ? ?>

<? Yes ? ? ? ?>

<? ? ? ? ? Yes> **Inconsistent**

Since the third hypothesis in *G*3 is inconsistent with this positive instance, remove the third one. The resulting *G*4 is,

*G*4 = < ≥9 ? ? ? ? ?>

<? Yes ? ? ? ?>

Using the two boundary sets, *S*4 and *G*4, the version space is converged to contain the set of consistent hypotheses.

The final version space is,

< ≥9 Yes ? ? ? ?>

< ≥9 ? ? Good ? ?>

<? Yes ? Good ? ?>

Thus, the algorithm finds the version space to contain only those hypotheses that are most general and most specific.

The diagrammatic representation of deriving the version space is shown in Figure 3.2.

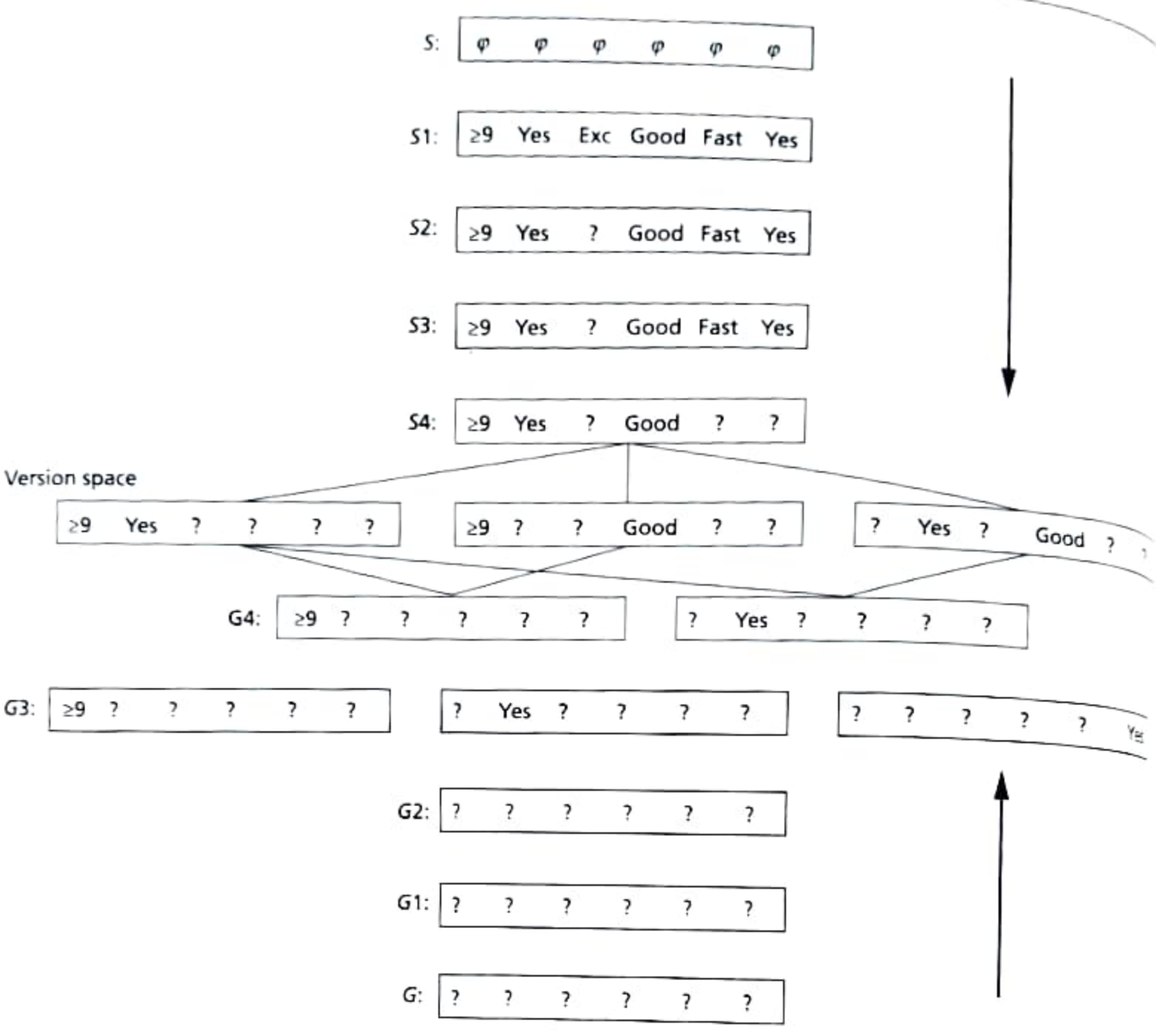

Figure 3.2: Deriving the Version Space