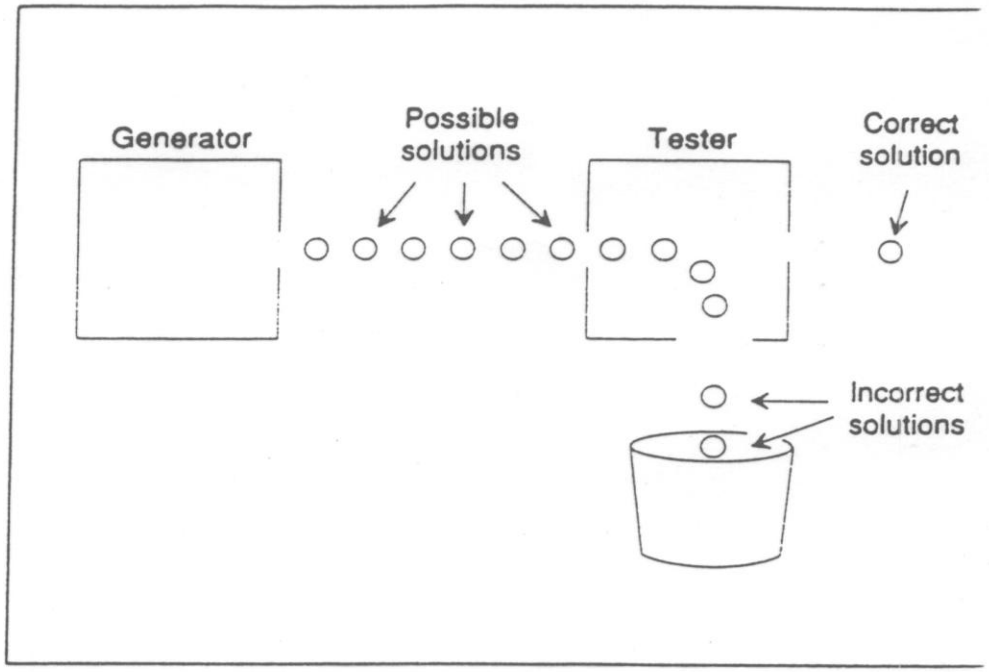
Problem Solver

- Problem Solving is the process of developing a sequence of actions to achieve a goal.
- This broad definition admits all goal directed AI programs to the ranks of problem solvers.
- Some more problem solving methods:
  - Generate and Test
  - Means-Ends Analysis
  - Problem Reduction
  - Planning
  - Decision Table
  - Decision Tree
- Few real problems can be solved by a single problem-solving method. Accordingly, it is often seen that problem-solving methods are working together.

Generate and Test



To perform generate and test,

- Until a satisfactory solution is found or no more candidate solutions can be generated,
  - Generate a candidate solution.
  - Test the candidate solution.
- If an acceptable solution is found, announce it; otherwise, announce failure.

Good Generators Are Complete, Nonredundant, and Informed

It is obvious that good generators have three properties:

- Good generators are complete: They eventually produce all possible solutions.
- Good generators are nonredundant: They never compromise efficiency by proposing the same solution twice.
- Good generators are informed: They use possibility-limiting information, restricting the solutions that they propose accordingly.

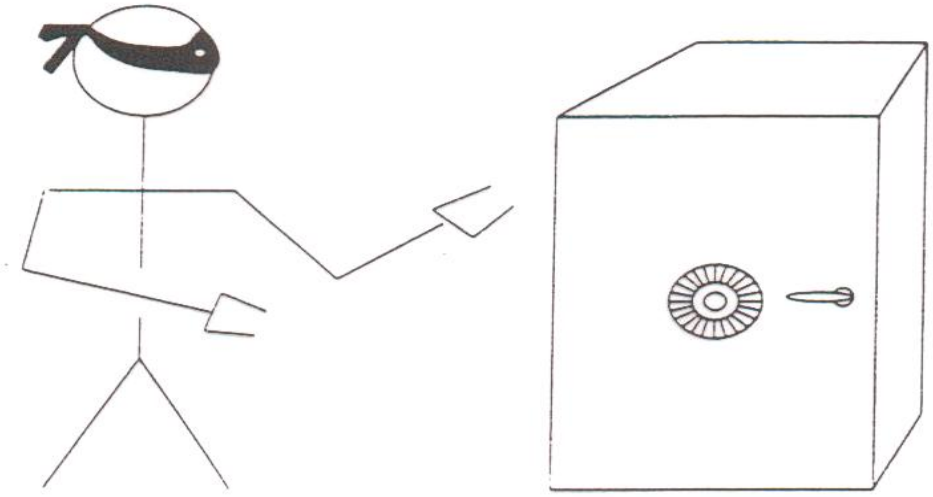
Informability is important, because otherwise there are often too many solutions to go through.

Generate-and-Test Systems Often Do Identification

To use the generate-and-test paradigm to identify, say, a tree, you can reach for a tree book, then thumb through it page by page, stopping when you find a picture that looks like the tree to be identified. Thumbing through the book is the generation procedure; matching the pictures to the tree is the testing procedure.

To use generate and test to burgle a three-number, two-digit safe, you can start with the combination 00-00-00, move to 00-00-01, and continue on through all possible combinations until the door opens. Of course, the counting is the generation procedure, and the twist of the safe handle is the testing procedure.

The burglar in figure 3.2 may take some time to crack the safe with this approach, however, for there are $100^3 = 1$ million combinations. At three per minute, figuring that he will have to go through half of the combinations, on average, to succeed, the job will take about 16 weeks, if he works 24 hours per day.



Means-Ends Analysis

The purpose of means-ends analysis is to identify a procedure (operator) for a selected transition which reduce the difference between the current state and the goal state.

The procedure selected is believed to reduce the difference, but there is no built-in mechanism preventing backward steps in the most general form of mens-ends analysis.

Figure 3.3 Means–ends analysis involves states and procedures for reducing differences between states. The current state and goal state are shown solid; other states, not yet encountered, are shown dotted.

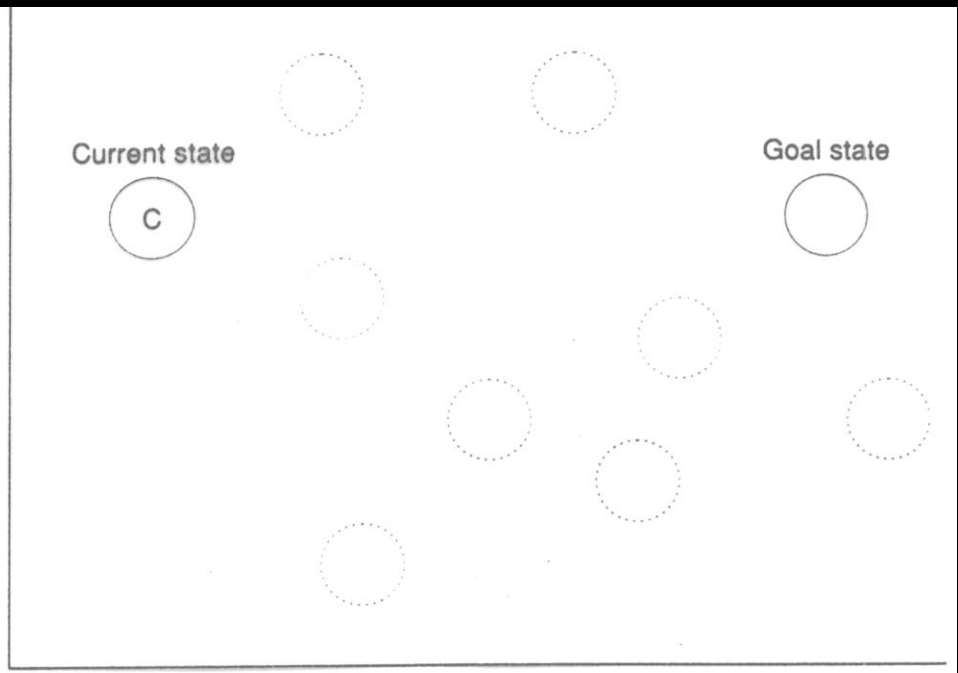
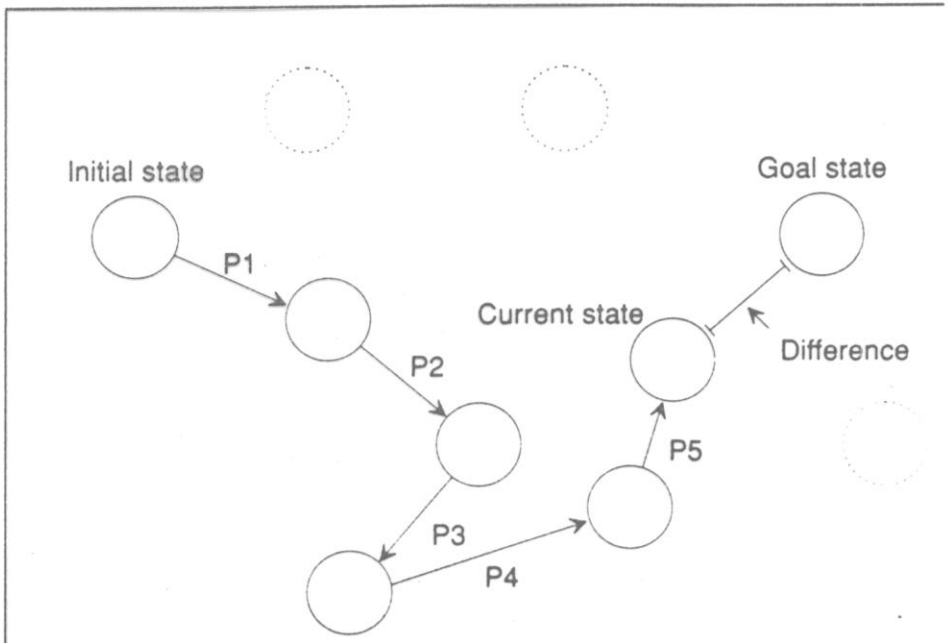


Figure 3.4 Means–ends analysis produces a path through state space. The current state, the goal state, and a description of their difference determine which procedure to try next. Note that the procedures are expected, but not guaranteed, to cause a transition to a state that is nearer the goal state than is the current state.



To perform means–ends analysis,

- Until the goal is reached or no more procedures are available,
  - Describe the current state, the goal state, and the difference between the two.
  - Use the difference between the current state and goal state, possibly with the description of the current state or goal state, to select a promising procedure.
  - Use the promising procedure and update the current state.
- If the goal is reached, announce success; otherwise, announce failure.

Difference-Procedure Tables Often Determine the Means

Whenever the description of the difference between the current state and the goal state is the key to which procedure to try next, a simple **difference-procedure table** may suffice to connect difference descriptions to preferred procedures.†

Consider, for example, a travel situation in which the problem is to find a way to get from one city to another. One traveler's preferences might link the preferred transportation procedure to the difference between states, described in terms of the distance between the cities involved, via the following difference-procedure table:

Distance	Airplane	Train	Car
More than 300 miles	√		
Between 100 and 300 miles		√	
Less than 100 miles			√

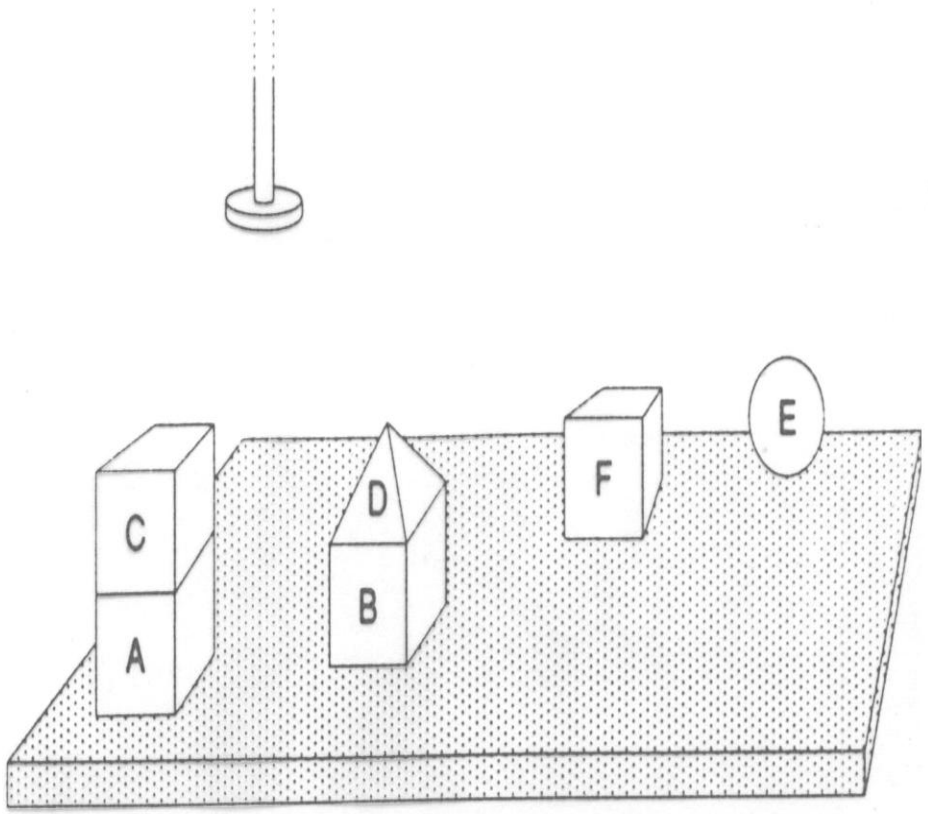
Problem Reduction

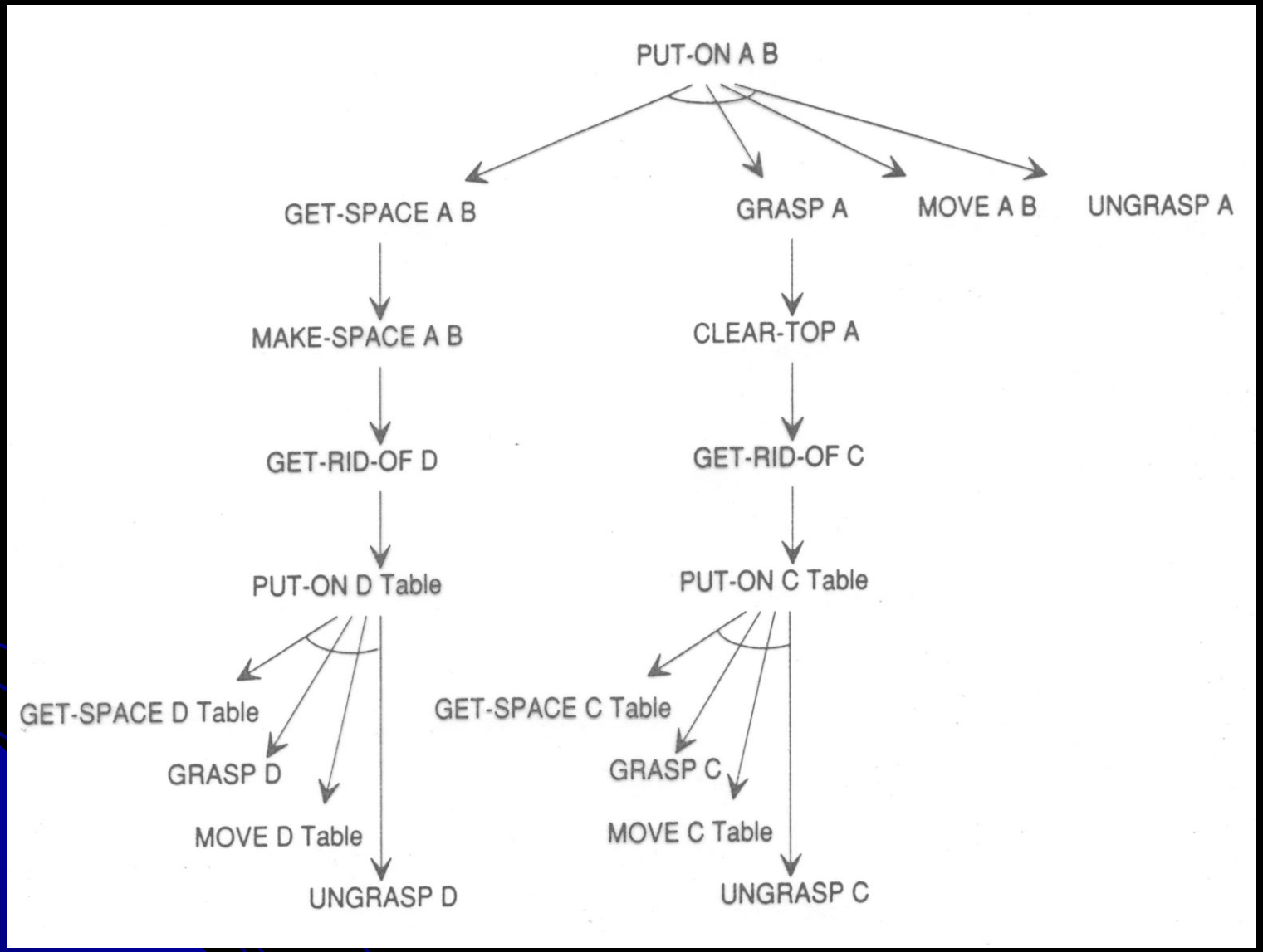
Problem reduction tries to convert difficult goals into easier-to-achieve subgoals. Each subgoal, in turn, may be divided into lower-level subgoals.

Goal tree is a kind of And-Or tree, in which nodes represent goals and branches indicate how you can achieve goals by solving one or more subgoals. Each node's children correspond to immediate subgoals; each node's parant corresponds to the immediate supergoal. Some goals are satisfied directly, they are called leaf goals.

Goal trees may be used to answer how and why questions.

Figure 3.5 MOVER is a procedure for planning motion sequences in the world of bricks, pyramids, balls, and a robot hand.




PUT-ON A B
GET-SPACE A B
GRASP A
MOVE A B
UNGRASP A
MAKE-SPACE A B
CLEAR-TOP A
GET-RID-OF D
GET-RID-OF C
PUT-ON D Table
PUT-ON C Table
GET-SPACE D Table
GET-SPACE C Table
GRASP D
GRASP C
MOVE D Table
MOVE C Table
UNGRASP D
UNGRASP C

To determine whether a goal has been achieved, you need a testing procedure. The key procedure, REDUCE, channels action into the REDUCE-AND and the REDUCE-OR procedures:

To determine, using REDUCE, whether a goal is achieved,

- Determine whether the goal is satisfied without recourse to subgoals:
  - If it is, announce that the goal is satisfied.
  - Otherwise, determine whether the goal corresponds to an And goal:
    - If it does, use the REDUCE-AND procedure to determine whether the goal is satisfied.
    - Otherwise, use the REDUCE-OR procedure to determine whether the goal is satisfied.

REDUCE uses two subprocedures: one deals with And goals, and the other deals with Or goals:

To determine, using REDUCE-AND, whether a goal has been satisfied,

- ▷ Use REDUCE on each immediate subgoal until there are no more subgoals, or until REDUCE finds a subgoal that is not satisfied.
- ▷ If REDUCE has found a subgoal that is not satisfied, announce that the goal is not satisfied; otherwise, announce that the goal is satisfied.

To determine, using REDUCE-OR, whether a goal has been satisfied,

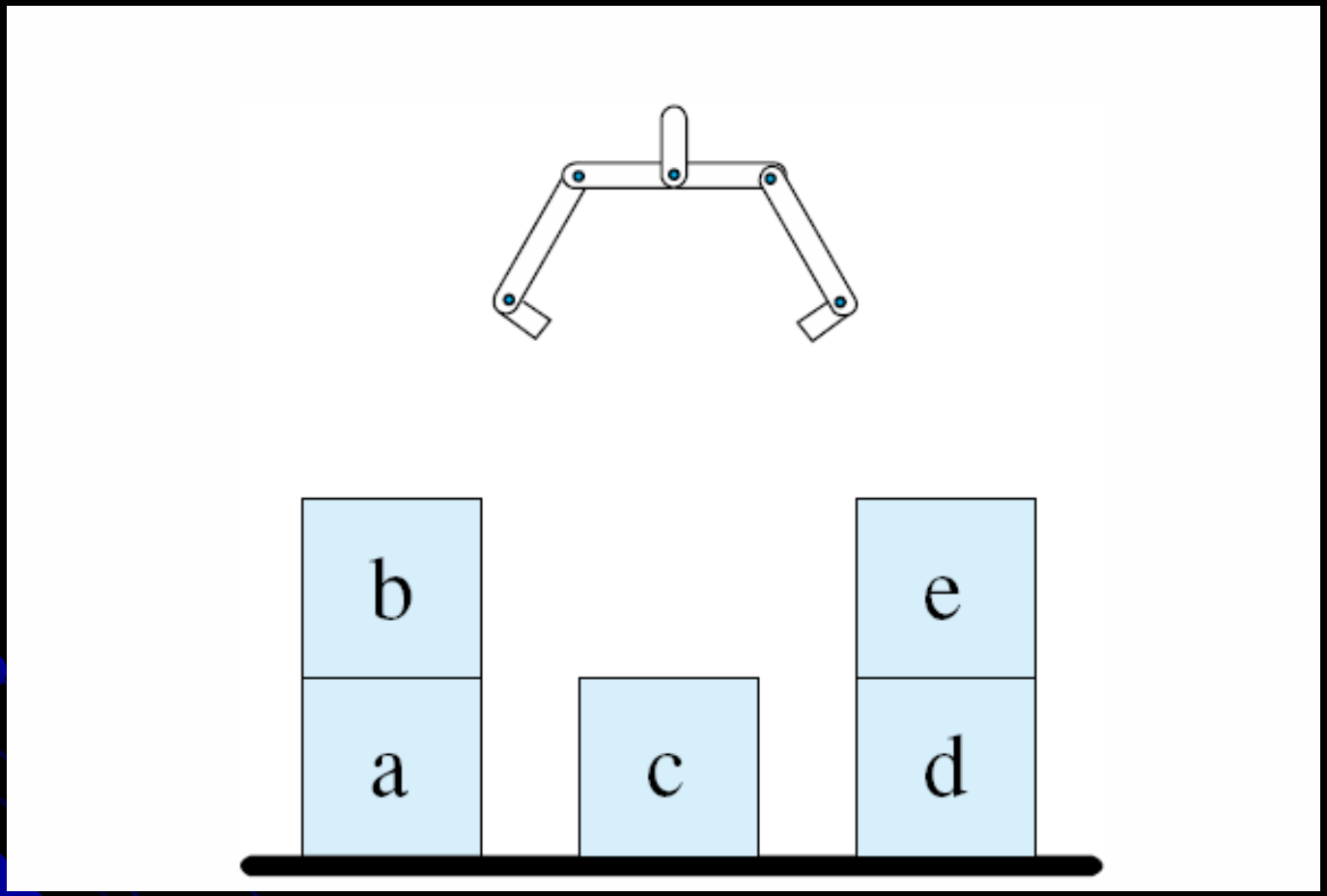
- ▷ Use REDUCE on each subgoal until REDUCE finds a subgoal that is satisfied.
- ▷ If REDUCE has found a subgoal that is satisfied, announce that the goal is satisfied; otherwise, announce that the goal is not satisfied.

With REDUCE, REDUCE-AND, and REDUCE-OR in hand, it is a simple matter to test an entire And–Or tree.

Planning

- Planning means deciding on a course of actions before acting.
- Failure to plan can result in less than optimal problem solving.
- Common-used approaches to planning:
  - Nonhierarchical planning
  - Hierarchical planning
  - Script-based planning
  - Opportunistic planning

Fig 8.18 The blocks world.



The blocks world of figure 8.18 may now be represented by the following set of predicates.

STATE 1

ontable(a).	on(b, a).	clear(b).
ontable(c).	on(e, d).	clear(c).
ontable(d).	gripping( ).	clear(e).

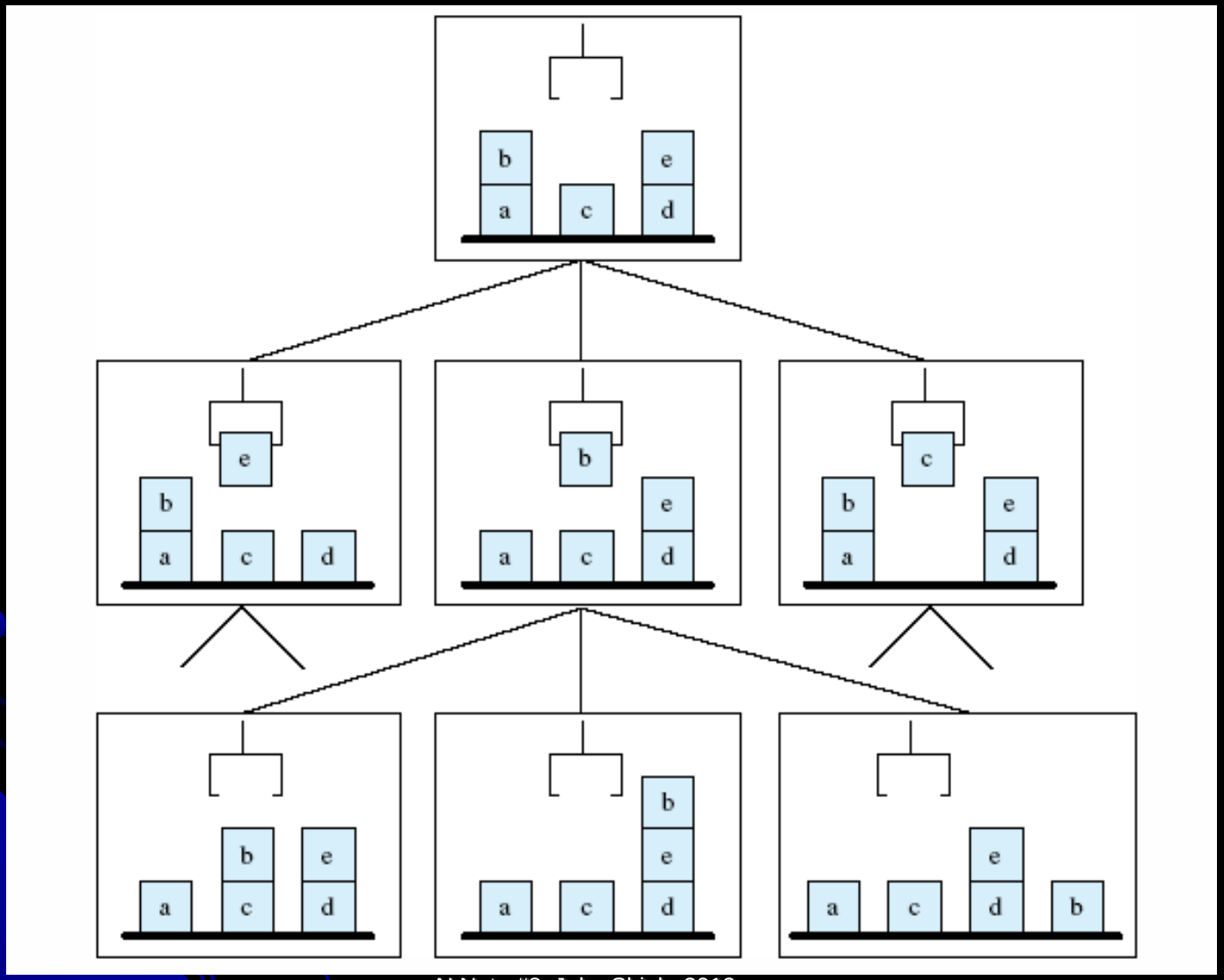
STATE 2

ontable(a).	on(b,a).	clear(b).
ontable(c).	clear(c).	clear(d).
ontable(d).	gripping(e).	clear(e).

A number of truth relations or rules for performance are created for the clear (X), ontable (X), and gripping ().

1. $(\forall X)\ (\text{clear}(X) \leftarrow \neg\ (\exists Y)\ (\text{on}(Y,X)))$
2. $(\forall Y)\ (\forall X) \neg\ (\text{on}(Y,X) \leftarrow \text{ontable}(Y))$
3. $(\forall Y)\ \text{gripping}(\ ) \leftrightarrow \neg\ (\text{gripping}(Y))$
4. $(\forall X)\ (\text{pickup}(X) \rightarrow (\text{gripping}(X) \leftarrow (\text{gripping}(\ ) \wedge \text{clear}(X) \wedge \text{ontable}(X))))$.
5. $(\forall X)\ (\text{putdown}(X) \rightarrow ((\text{gripping}(\ ) \wedge \text{ontable}(X) \wedge \text{clear}(X)) \leftarrow \text{gripping}(X)))$.
6. $(\forall X)\ (\forall Y)\ (\text{stack}(X,Y) \rightarrow ((\text{on}(X,Y) \wedge \text{gripping}(\ ) \wedge \text{clear}(X)) \leftarrow (\text{clear}(Y) \wedge \text{gripping}(X))))$.
7. $(\forall X)(\forall Y)\ (\text{unstack}(X,Y) \rightarrow ((\text{clear}(Y) \wedge \text{gripping}(X)) \leftarrow (\text{on}(X,Y) \wedge \text{clear}(X) \wedge \text{gripping}(\ ))))$.
8. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{unstack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X)))$.
9. $(\forall X)\ (\forall Y)\ (\forall Z)\ (\text{stack}(Y,Z) \rightarrow (\text{ontable}(X) \leftarrow \text{ontable}(X)))$.

Fig 8.19 Portion of the state space for a portion of the blocks world.



Using blocks example, the four operators pickup, putdown, stack, and unstack are represented as triples of descriptions.

pickup(X)	P: gripping( ) $\wedge$ clear(X) $\wedge$ ontable(X) A: gripping(X) D: ontable(X) $\wedge$ gripping( )
putdown(X)	P: gripping(X) A: ontable(X) $\wedge$ gripping( ) $\wedge$ clear(X) D: gripping(X)
stack(X,Y)	P: clear(Y) $\wedge$ gripping(X) A: on(X,Y) $\wedge$ gripping( ) $\wedge$ clear(X) D: clear(Y) $\wedge$ gripping(X)
unstack(X,Y)	P: clear(X) $\wedge$ gripping( ) $\wedge$ on(X,Y) A: gripping(X) $\wedge$ clear(Y) D: gripping( ) $\wedge$ on(X,Y)

Fig 8.20 Goal state for the blocks world.

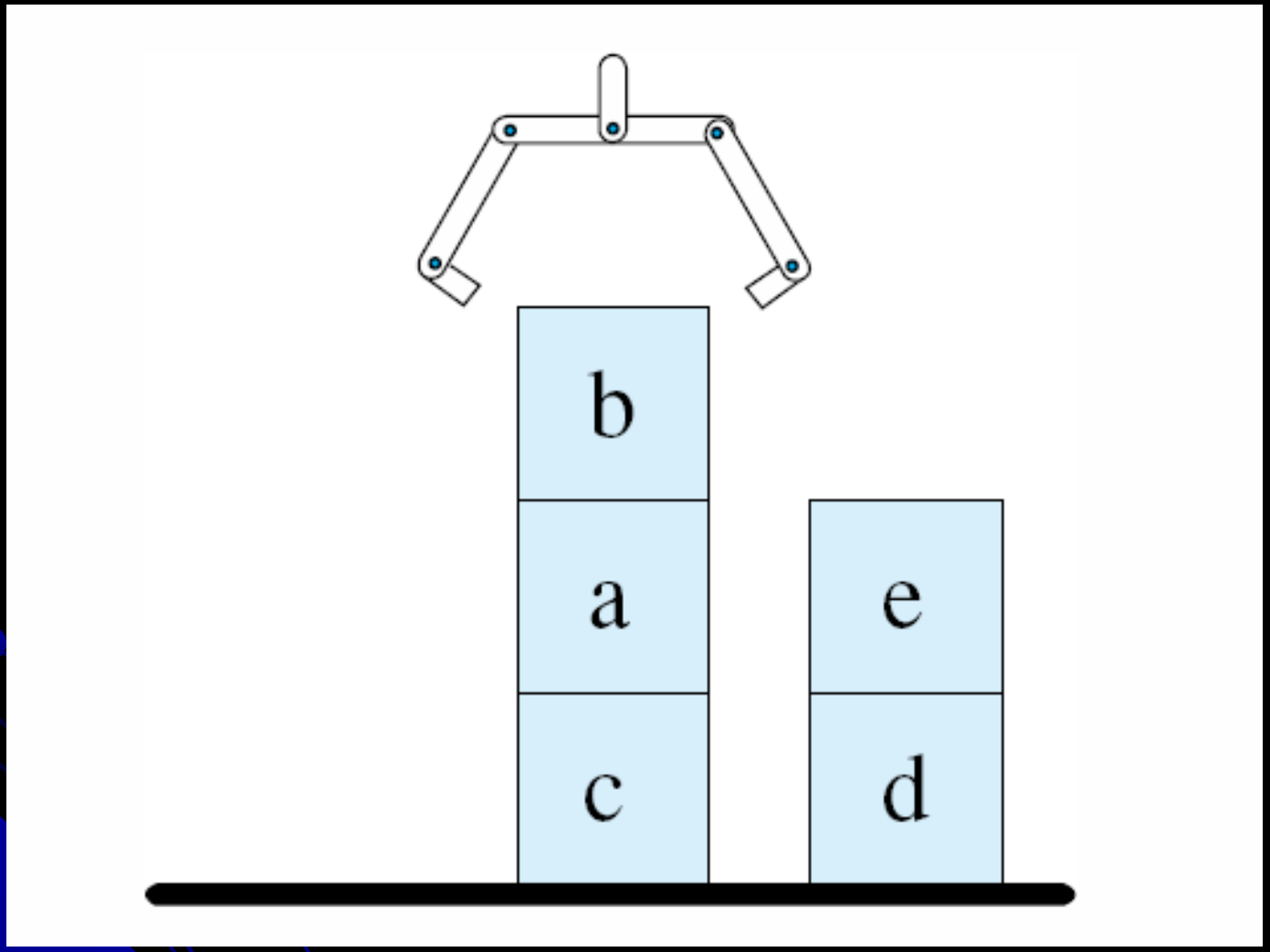
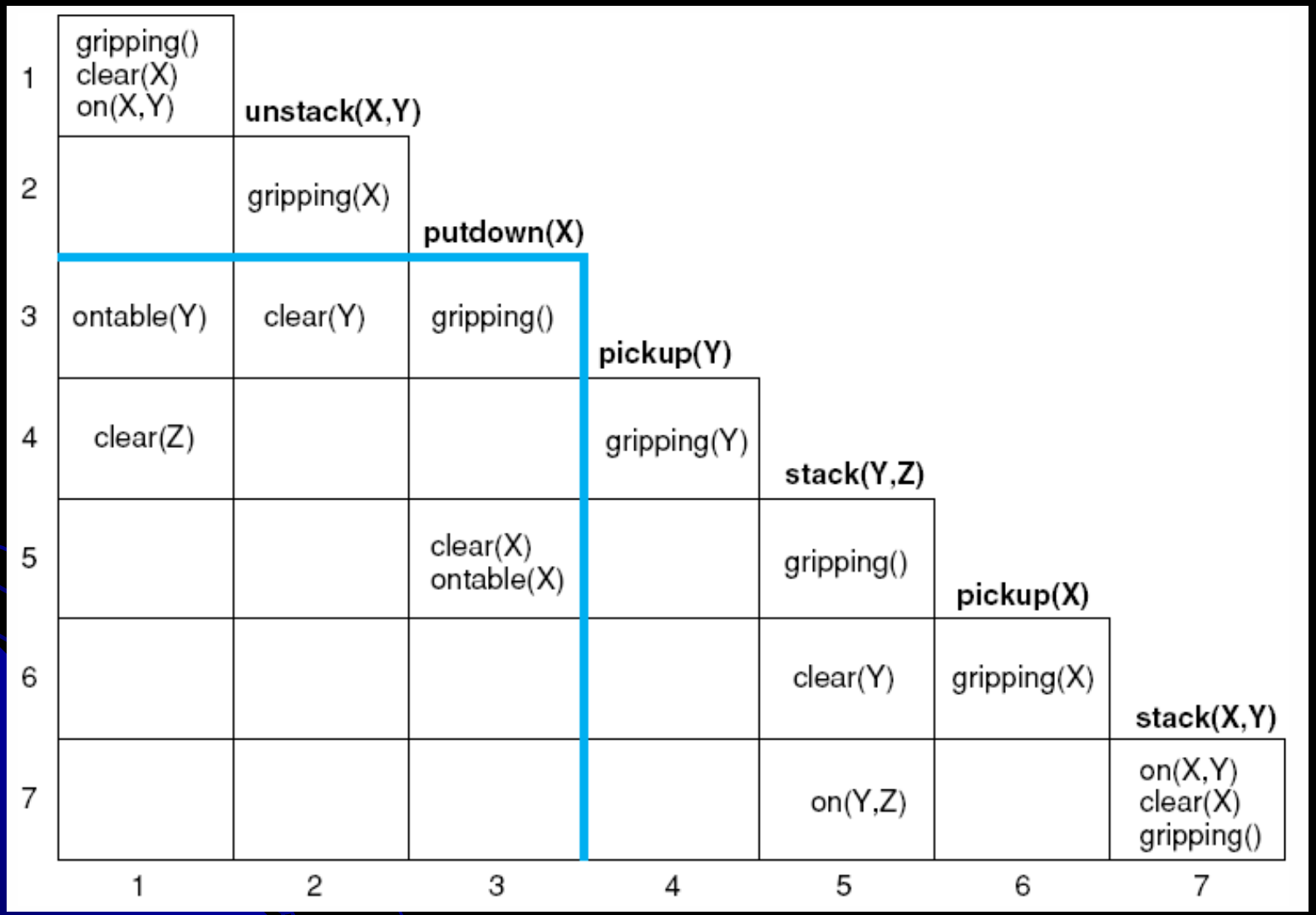


Fig 8.21 A triangle table, adapted from Nilsson (1971).

For goals of stack(X,Y) ^ Stack(Y,Z)



In a decision table, the decision rules are exhaustive in that they must cover every possible combination of conditions. Thus the creation of a decision table must concern its completeness, accuracy, redundancy, inconsistency, endless loops, and size. Some decision table languages, such as Detab/65, provide a check on the completeness and consistency of the design.

Decision tables are rarely used in the situations which can be clearly described in mathematical terms.

DECISION TABLE STRUCTURE

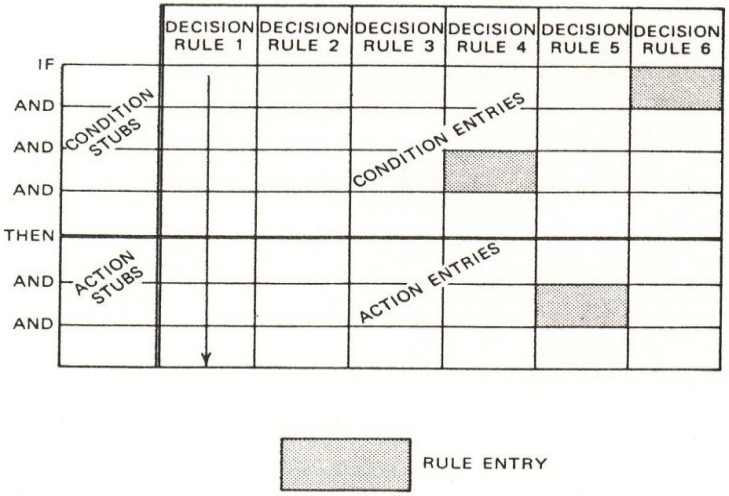


Figure 2-1 **Decision table structure.**

Limited Entry Rows

In the condition area (above the horizontal line) note the following:

"Y" prescribes that the condition in the stub must be satisfied;

"N" prescribes that the condition in the stub must not be satisfied;

"—" prescribes that it is immaterial whether the condition in the stub is satisfied or not.

In the action area (below the horizontal double line) note the following:

"X" prescribes the action in the stub that is to be executed if all the conditions of that rule are satisfied.

"–" prescribes that the action in the stub is to be ignored, whether or not all the conditions of that rule are satisfied.

The rules described in the decision table of Figure 2-2 are read as follows:

Decision Rule 1: If a customer's credit limit is satisfactory, approve his order.

Decision Rule 2: If the customer's credit limit is not satisfactory and his pay experience is favorable, approve his order.

Decision Rule 3: If the customer's credit limit is not satisfactory and his pay experience is not favorable and special clearance is obtained, approve his order.

Decision Rule 4: If the customer's credit limit is not satisfactory and his pay experience is not favorable and special clearance is not obtained, return his order to the sales department.

	DECISION RULE 1	DECISION RULE 2	DECISION RULE 3	DECISION RULE 4
CREDIT LIMIT IS SATISFACTORY	Y	N	N	N
PAY EXPERIENCE IS FAVORABLE	–	Y	N	N
SPECIAL CLEARANCE IS OBTAINED	–	–	Y	N
APPROVE ORDER	X	X	X	–
RETURN ORDER TO SALES	–	–	–	X

Figure 2-2 **Credit Policy.**

In a decision tree[2], each chance node is represented by a circle, and each decision node by a square. Events fan out from circles, and decisions fan out from squares. By convention, the tree extends from left to right. At the tips of the tree (the rightmost nodes) are outcomes that describe where a specific act-event sequence will lead.

Any path from left to right through the tree constitutes a scenario. Any path through a decision tree can be thought of as an alternating sequence of actions by a decision maker and events in the environment.

By specifying an (in principle) exhaustive set of scenarios consequent to each possible action, a decision tree facilitates evaluation of actions.

Decision trees are very efficient for selection and classification.

Another useful feature of decision trees is that they can be made self-learning. If the guess is wrong, a procedure can be called to query the user for a new, correct classification question and the answers to the "yes" and "no" responses. A new node, branches and leaves can then be dynamically created and added to the tree.

Problems suitable for decision trees are typified by two primary characteristics:

1. They provide the answer to a problem from a predetermined set of possible answers. Taxonomy and diagnosis problems generally meet this requirement.

   In general, decision trees will not work well for problems which must generate solutions in addition to selecting them.

2. The manner in which decision trees derive a solution is by reducing the set of possible solutions with a series of decisions or questions that prune the search space of a decision tree.

Decision trees

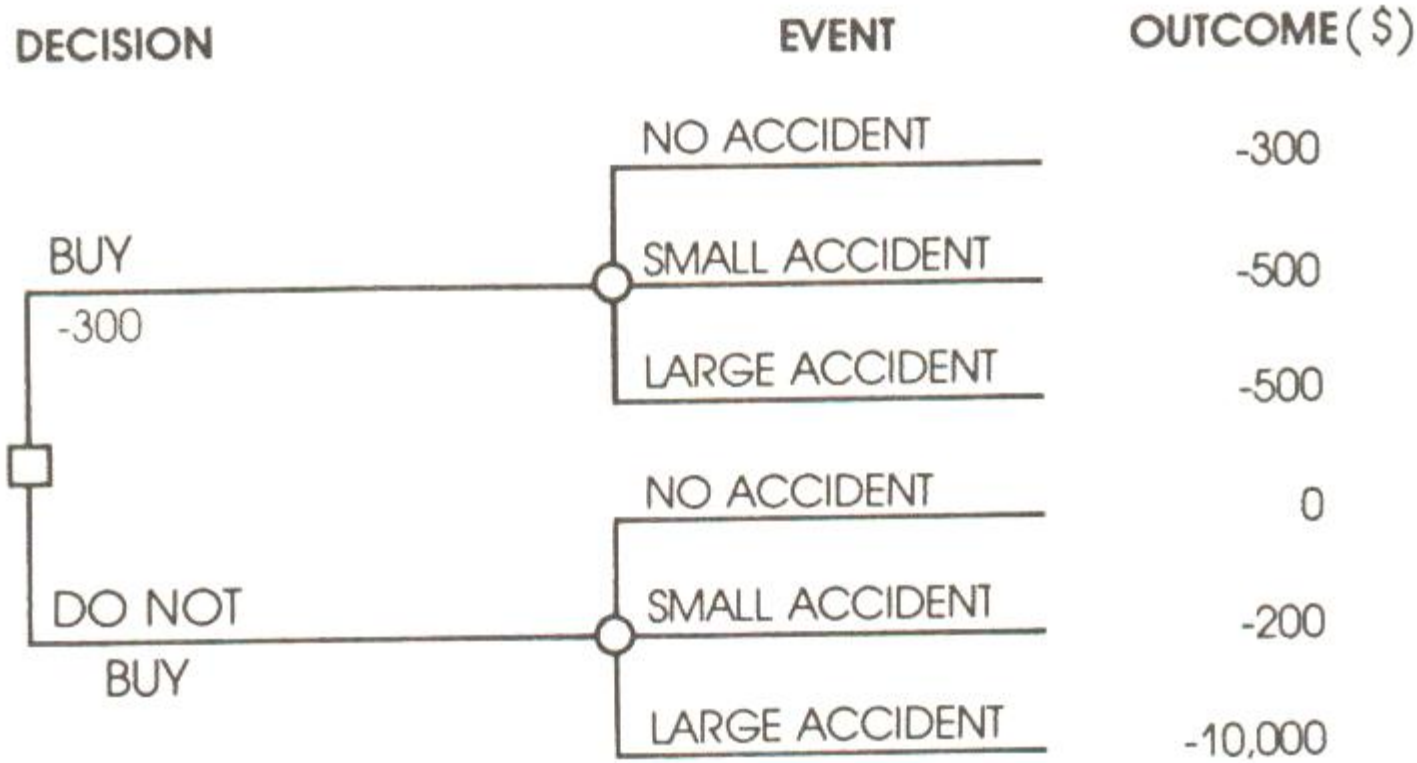


Figure 3.1. Decision tree for an insurance example.

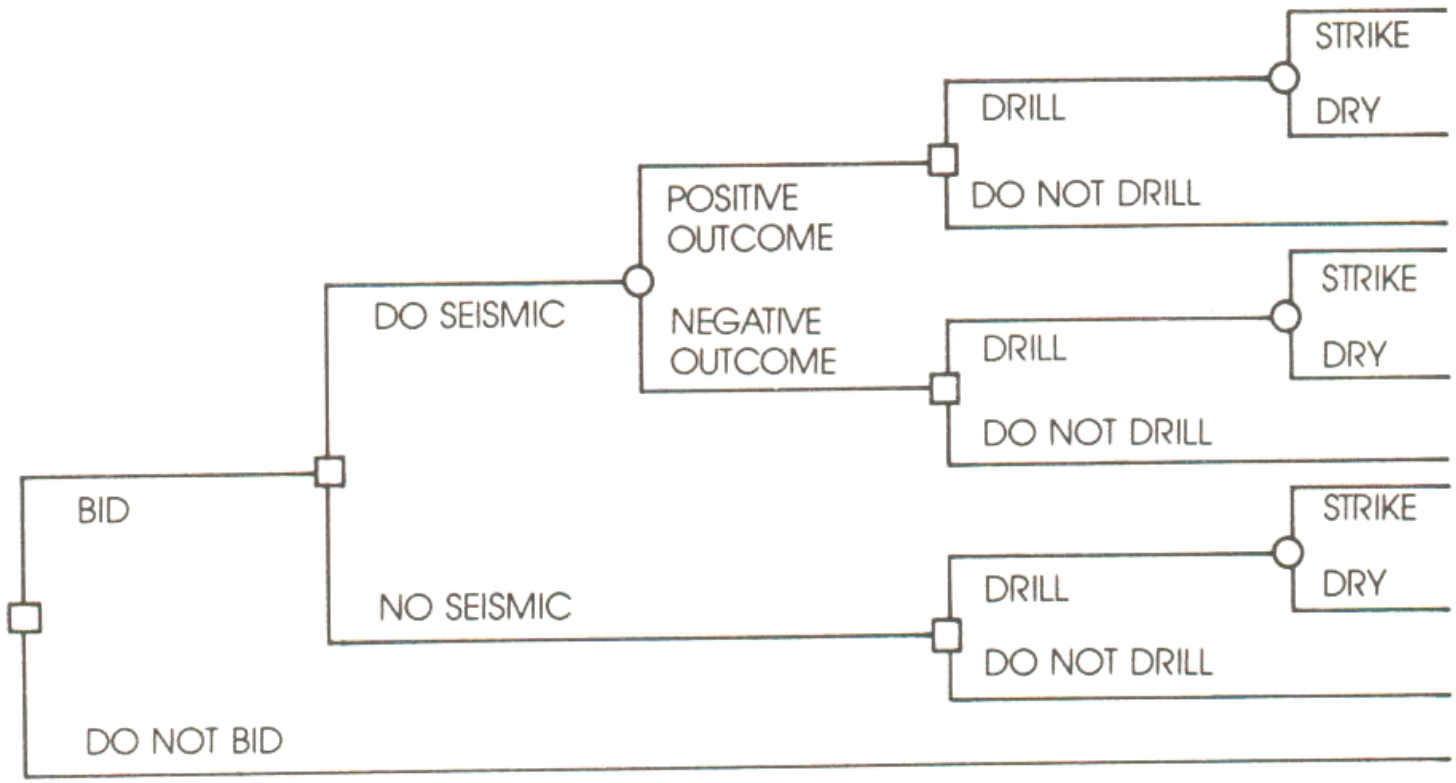


Figure 3.2. Decision tree for an oil-drilling example.