CHAPTER 6

Plan-Formation and the Frame Problem

In the plan-formation problem we are given an initial state, a goal state, and a set of actions which transform one state into another. The problem is to construct a plan, consisting of an appropriate sequence of actions, which transforms the initial state into the goal state.

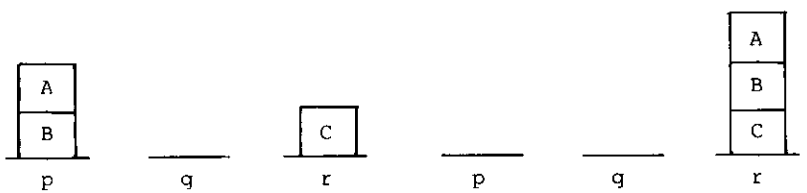
The plan-formation problem is identical, therefore, to the state-space problem. The n-tuple representation of state-space problems is not feasible, however, when the number n of individuals is large or unknown. In this chapter, we investigate a version of the binary representation of state space problems.

The use of logic, in both the n-ary and binary representations, runs into the frame problem: how to deal with the fact that almost all statements which hold true of a given state continue to hold after an action has been performed. It has often been assumed that such facts cannot be expressed naturally in logic and cannot be used efficiently.

The supposed inadequacies of logic have led to the development of special systems, such as STRIPS [Fikes and Nilsson 1971] and PLANNER [Hewitt 1969], specifically intended to deal with the frame problem. We shall argue that an equally satisfactory treatment of the frame problem can be obtained in logic: by using terms to name statements and by using the frame axiom, which describes the statements which continue to hold after an action has been performed, top-down rather than bottom-up.

Plan-formation and the blocks world

We shall consider the simple blocks world plan-formation problem [Sacerdoti 1977] in detail. There are three manipulatable blocks A, B and C and three unmanipulatable places p, q and r. The location of objects in the initial and goal states is illustrated below:



There is a single action

trans(x,y,z)

which transfers x from y to z. The action can be performed in a given state if

x is manipulatable,
x and z are clear,
x is on y, and
x is different from z.

The new statement that

x is on z and
y is clear

holds true of the new state which results when the action has been performed. All statements which held in the previous state, except that

x is on y and
z is clear,

continue to hold in the new state.

In general, an action is defined by specifying its preconditions and postconditions. Preconditions are statements which must hold in a state before an action can be performed; whereas postconditions are statements which hold in the new state after the action has been performed. Postconditions are of two kinds: new statements which are added to the description of the new state and old statements which continue to hold from the previous state. The old statements are described by means of a frame axiom which expresses that all statements which held in the old state, except for those explicitly stated as exceptions to be deleted, continue to hold in the new state. The explicit specification for every action of preconditions, added statements and deleted statements is due to STRIPS.

A clausal representation of the blocks world problem

In this formulation, both states and statements are regarded as individuals and are represented by means of terms. That a statement x holds true in a state y is represented by a binary relationship

Holds(x,y).

States are named by constant symbols or by composite terms. It is convenient to let the constant symbol 0 name the initial state and to let the term

result(u,v)

name the state which results from applying the action u to the state v.

The representation of statements by means of terms is discussed in Chapter 12 concerned with formalising part of the meta-language. Here it is sufficient to let the term

```
on(x,y)
```

name the statement that x is on y and

```
clear(x)
```

that x is clear. An alternative representation, in which the term

```
atom(x,y)
```

names the atomic formula with predicate symbol x and list of arguments y, is more flexible but not necessary here.

In the following clauses

```
Poss(x)     expresses that state x is possible,
Manip(x)                   object x is manipulatable,
Diff(x,y)                  x is different from y.
```

```
Initial state ∅   (1)  Poss(∅) <-
                  (2)  Holds(on(A,B), ∅) <-
                  (3)  Holds(on(B,p), ∅) <-
                  (4)  Holds(on(C,r), ∅) <-
                  (5)  Holds(clear(A), ∅) <-
                  (6)  Holds(clear(q), ∅) <-
                  (7)  Holds(clear(C), ∅) <-
State-independent
assertions        (8)  Manip(A) <-
                  (9)  Manip(B) <-
                 (10)  Manip(C) <-

Goal state       (11)  <- Holds(on(A,B),w), Holds(on(B,C),w),
                          Holds(on(C,r),w), Poss(w)

State space and
preconditions    (12)  Poss(result(trans(x,y,z),w)) <- Poss(w),
                                Manip(x), Diff(x,z), Holds(clear(x),w),
                                Holds(clear(z),w), Holds(on(x,y),w)

Added statements (13)  Holds(on(x,z),  result(trans(x,y,z), w)) <-
                 (14)  Holds(clear(y), result(trans(x,y,z), w)) <-

Frame axiom and deleted
statements       (15)  Holds(u, result(trans(x,y,z), w)) <-
                                          Holds(u,w), Diff(u, on(x,y)),
                                          Diff(u, clear(z))
```

Clauses (1)-(6) describe the initial state, whereas clauses (7)-(10) describe the state independent facts about the manipulatability of blocks and clause (11) describes the goal state. The remaining clauses describe the action of transferring an object from one location to another. Clause (12) defines the structure of the state-space search space. It expresses

the preconditions which need to hold before an action can be applied to a possible state in order to produce a new one. Clauses (13) and (14) express the postconditions which are added by the action, whereas (15) expresses those which hold in the new state because they held in the previous state and were not destroyed by the action.

The relationship Diff(s,t) holds, for variable-free terms s and t, when s and t are syntactically distinct. It is useful to imagine that clauses (1)-(15) are supplemented by infinitely many clauses of the form

$$\text{Diff}(s,t) \leftarrow$$

for every pair of terms s and t which do not match. Equivalently, the same relation can be defined by the axioms

$$\text{Diff}(f(x_1,\ldots,x_m),\ g(y_1,\ldots,y_n)) \leftarrow$$

for every pair of distinct function symbols f and g, including the cases $m = \emptyset$ and $n = \emptyset$ when f and g are constant symbols, and

$$\text{Diff}(f(x_1,\ldots,x_m),\ f(y_1,\ldots,y_m)) \leftarrow \text{Diff}(x_i,y_i)$$

for every function symbol f and for every argument i of f, excluding the case $m = \emptyset$ when f is a constant symbol. In practice, it is more efficient to define Diff as the negation of identity

$$\text{Diff}(x,y) \leftarrow \text{not-}(x = y)$$
$$x = x \leftarrow$$

and to determine that not-(x = y) holds by showing that x = y fails to hold. Such an interpretation of negation as failure and its relationship to the normal interpretation of negation has been studied by Clark [1978] and is discussed in Chapter 11 which is concerned with definitions expressed in terms of "if-and-only-if".

This formulation of the plan-formation problem is similar to the one employed by Green [1969b], based upon proposals of McCarthy and Hayes [McCarthy and Hayes 1969]. It differs from their formulations, however, in its use of the Holds relation. They add an extra state parameter to relations instead, writing, for example, On(x,y,w) to express that x is on y in state w and Clear(x,w) that x is clear in w. The treatment of statements as individuals, which is implied by the use of the Holds relation, can be regarded as a formalisation of part of the meta-language. The advantages of using logic as its own meta-language are discussed later in Chapter 12. Here it suffices to note that treating sentences as individuals avoids that part of the frame problem which is concerned with expressing the frame axiom. Instead of employing a separate frame axiom for every relation, writing, for example,

$$\text{On}(u, v, \text{result}(\text{trans}(x,y,z), w)) \leftarrow \text{On}(u,v,w), \text{Diff}(u,x)$$
$$\text{Clear}(u, \text{result}(\text{trans}(x,y,z), w)) \leftarrow \text{Clear}(u,w), \text{Diff}(u,z)$$

it suffices to employ a single frame axiom

$$\text{Holds}(u, \text{result}(v,w)) \leftarrow \text{Holds}(u,w), \text{Preserves}(v,u)$$

where Preserves(v,u) expresses that the action v preserves the truth of

statement u. The use of the Preserves relation separates the frame axiom from the specification of the statements which are deleted by individual actions. In the case of the trans-action:

```
Preserves(trans(x,y,z), u) <- Diff(u, on(x,y)),
                              Diff(u, clear(z))
```

As we shall see in the next chapter, clause (15), which combines the frame axiom and the specification of the deleted statements, can be obtained by macro-processing the procedure call to the relation Preserves. Macro-processing executes procedure calls at compile time before problems are given, rather than at run time during the course of trying to solve them. It can be regarded as a form of middle-out reasoning, which in turn is a special case of the resolution rule [Robinson 1965a]. Resolution also generalises top-down and bottom-up inference and applies to non-Horn clauses as well.

It is useful to classify relations into two kinds: primitive relations, which are independent of other relations, and defined relations, which can be defined in terms of the primitives. In the blocks world, the relationship which holds when one object is above another can be defined in terms of the primitive relationship which holds when one object is located directly on another.

```
Holds(above(x,y), w) <- Holds(on(x,y), w)
Holds(above(x,y), w) <- Holds(above(x,z), w),
                        Holds(above(z,y), w)
```

It suffices to specify added and deleted statements only for primitive relations. The effect of actions on defined relationships is determined by their effect on primitive relationships and by the definition of the defined relations in terms of the primitives. The classification of relations and its use in plan-formation was introduced with STRIPS.

We have treated the On and Clear relations as primitive. It would be more natural, however, to define the Clear relation in terms of the On relation:

```
Holds(clear(y), w) <- for all x not-Holds(on(x,y), w)
```

We shall discuss this possibility in Chapter 11, which investigates if-and-only-if definitions and the interpretation of negation as failure.

The logic of the blocks world problem is separate from its use. Clauses can be used either top-down or bottom-up. They can also be used in a mixture of directions. If the state space axiom (12) is used bottom-up, then the problem-solver reasons forward from the initial state, deriving new states from old ones, until the goal state is generated. If the axiom is used top-down, then the problem-solver reasons backward from the goal-state, until the initial state is generated.

The second part of the frame problem arises when the frame axiom (15) is used bottom-up to derive, from an assertion that a given statement holds in a given state, a new assertion that the same statement holds in a following state. For more realistic plan-formation tasks than the blocks world problem, a typical state needs to be described by a large number of assertions, many of which are unrelated to the problem at hand.

In such situations it is not computationally feasible to use the frame axiom bottom-up to copy preserved facts from state to state.

Both PLANNER and STRIPS deal with the frame problem by abandoning the frame axiom and using special-purpose procedures instead. Similar results can be obtained by retaining the frame axiom but interpreting it top-down:

To determine whether a statement u holds in a state result(v,w)

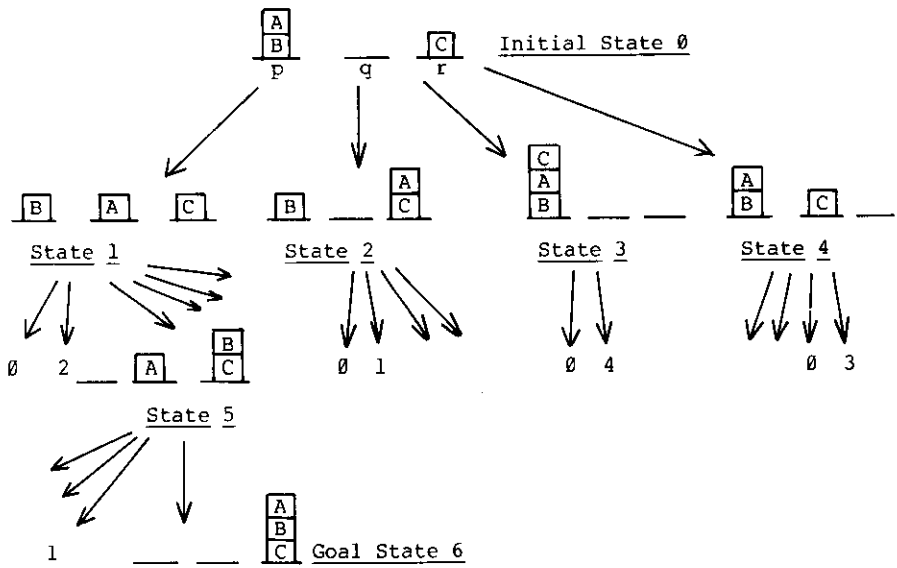
(i) show u is added by v,
(ii) alternatively, if u is not deleted by v,
determine whether u holds in the previous state w.

Changing the direction of execution of the frame axiom exemplifies the general strategy of improving an algorithm by improving its control without changing its logic.

We shall illustrate the different solutions determined for the blocks world problem by using the state space and frame axioms in different directions.

Bottom-up execution of the state space axiom (12)

The following illustration displays part or the search space of states determined by executing (12) bottom-up.



Distinct nodes represent distinct states. However, distinct states labelled by the same number are characterised by the same statements. In this case, the redundancy arises because it is never useful to pick up the same object twice in a row.

The assertions which are generated by bottom-up execution of the state space axiom describe the search space of states illustrated above and are independent of the direction of execution of the frame axiom.

Bottom-up execution of the frame axiom (15)

The following assertions, concerning states which belong to the solution path, are generated by bottom-up execution of the frame axiom.

```
Holds(on(B,p), 1)  <-    Holds(on(C,r), 1)  <-
Holds(on(A,q), 5)  <-    Holds(on(C,r), 5)  <-
Holds(on(B,C), 6)  <-    Holds(on(C,r), 6)  <-
Holds(clear(A), 1) <-    Holds(clear(C), 1) <-
Holds(clear(B), 5) <-    Holds(clear(A), 5) <-
Holds(clear(A), 6) <-    Holds(clear(p), 6) <-
```

The additional assertions

```
Holds(on(A,q), 1)  <-    Holds(clear(B), 1) <-
Holds(on(B,C), 5)  <-    Holds(clear(p), 5) <-
Holds(on(A,B), 6)  <-    Holds(clear(q), 6) <-
```

which are needed for a complete description of the same states are instances of the clauses (13) and (14) which specify the statements added by the trans-action. As in the previous illustration,

```
1   abbreviates  result(trans(A,B,q), ∅),
5                result(trans(B,p,C), 1),
6                result(trans(A,q,B), 5).
```

In the general case, a search strategy might need to generate many assertions concerning states which are not relevant to the solution as well as assertions such as

```
Holds(on(B,p), result(trans(A,C,B), ∅)) <-
Holds(on(B,p), result(trans(B,q,C), ∅)) <-
Holds(on(B,p), result(trans(B,B,B), ∅)) <-
```

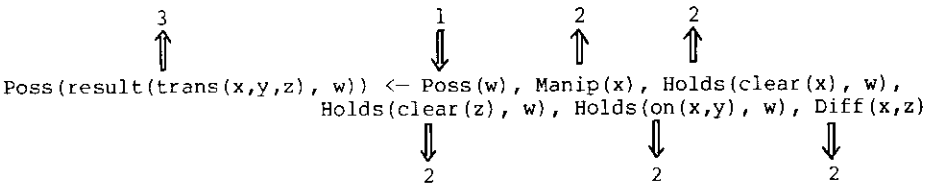
which describe impossible states. The generation of such undesirable assertions is avoided if the frame axiom is used top-down. It can also be avoided when the frame axiom is used bottom-up by adding the extra condition

```
Poss(result(trans(x,y,z), w))
```

to the frame axiom.

Mixed top-down and bottom-up execution of the frame axiom

Top-down execution of the frame axiom may be combined with bottom-up execution of the state space exiom. This can be pictured in arrow notation:



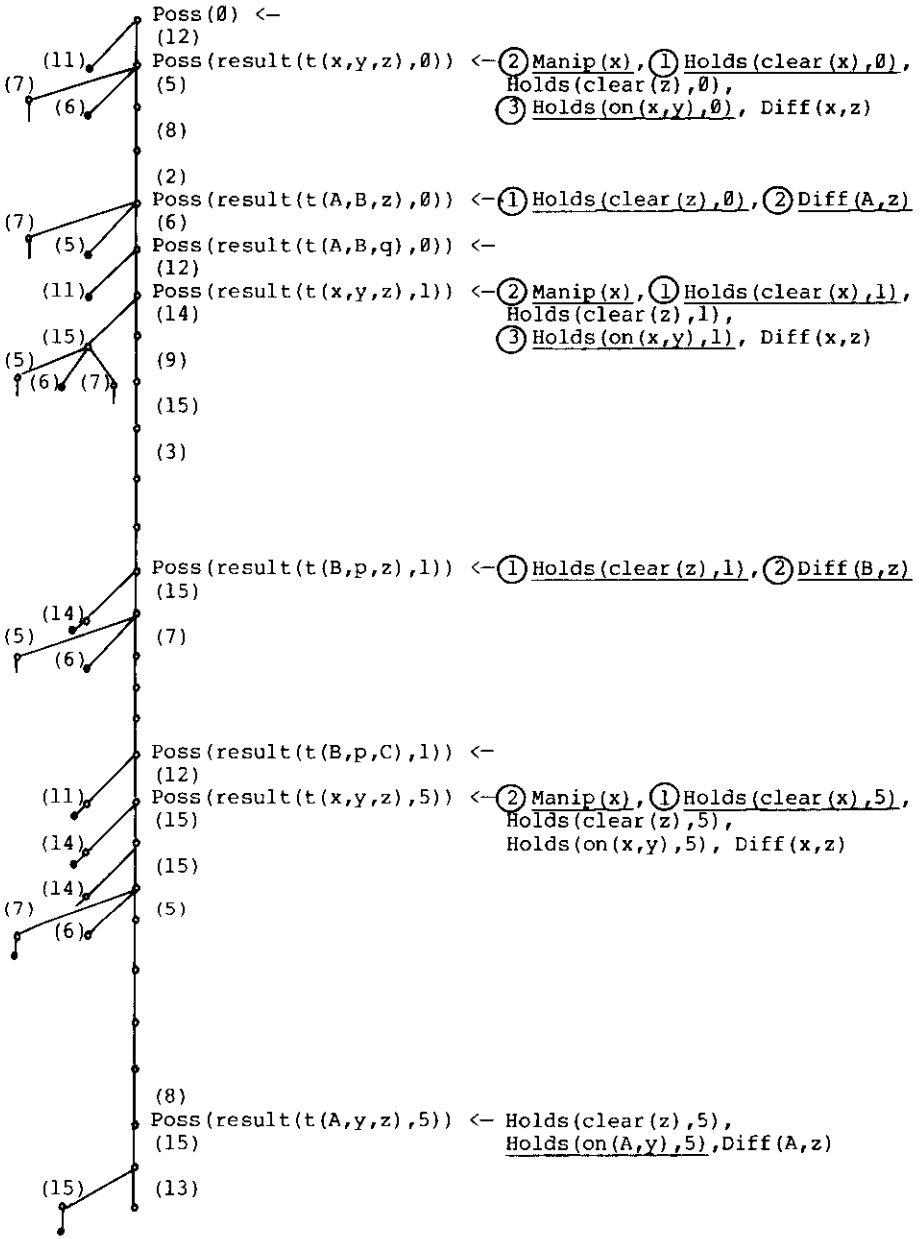
It can be simulated by top-down execution alone. It suffices to rewrite clauses (1), (11) and (12) using a predicate symbol Nposs which is the negation of Poss. Clauses (1), (11) and (12) become (1'), (11') and (12') respectively.

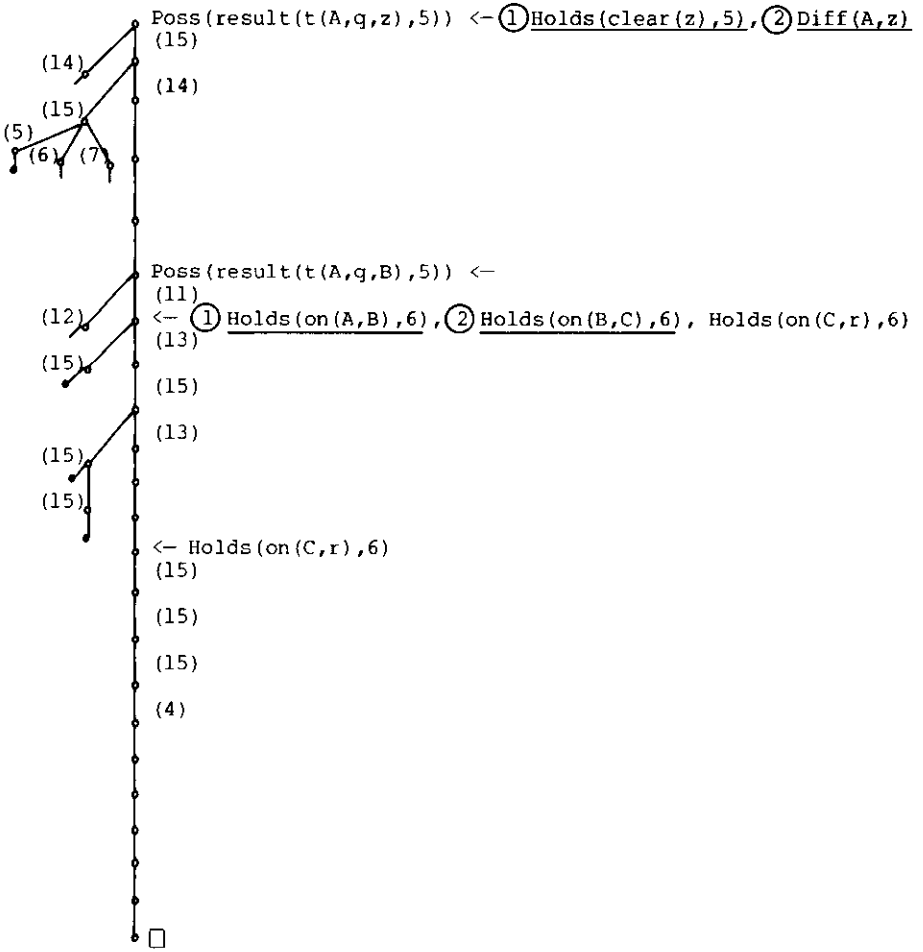
```
(1')        <- Nposs(Ø)
(11')       Nposs(w) <- Holds(on(A,B), w), Holds(on(B,C), w),
                        Holds(on(C,r), w)
(12')       Nposs(w) <- Nposs(result(trans(x,y,z), w)), Manip(x),
                        Holds(clear(x), w), Holds(clear(z), w),
                        Holds(on(x,y), w), Diff(x,z)
```

The renaming of predicate symbols, of the kind involved in rewriting clauses (1), (11) and (12), has been investigated by Meltzer [1966] and will be considered again in the next chapter.

A small part of the search space is illustrated below. The mixed top-down, bottom-up execution strategy is equivalent to pure top-down execution using clauses (1'), (11') and (12') instead of (1), (11) and (12). All arcs which diverge from the solution path are illustrated. Nodes which are labelled by clauses containing unsolvable subgoals are darkened to indicate that they are terminal failure nodes. The circled numbers preceding underlined atoms indicate the order in which they or their descendants are selected. Unlabelled arcs indicate execution of procedure calls containing the predicate symbol Diff. Some nodes are left unlabelled in order to suppress distracting details. t(x,y,z) abbreviates trans(x,y,z).

Notice that many alternatives to the solution path fail after only a few steps. The alternatives which do not fail correspond to genuine alternative actions in the search space of states.


Poss(∅) <-
(12)
(11)
Poss(result(t(x,y,z),∅)) <-(2)Manip(x),(1)Holds(clear(x),∅),
(7)
(5)
Holds(clear(z),∅),
(6)
(3)Holds(on(x,y),∅), Diff(x,z)
(8)
(2)
Poss(result(t(A,B,z),∅)) <-(1)Holds(clear(z),∅),(2)Diff(A,z)
(7)
(6)
(5)
Poss(result(t(A,B,q),∅)) <-
(12)
(11)
Poss(result(t(x,y,z),1)) <-(2)Manip(x),(1)Holds(clear(x),1),
(14)
Holds(clear(z),1),
(15)
(3)Holds(on(x,y),1), Diff(x,z)
(5)
(9)
(6)
(7)
(15)
(3)
Poss(result(t(B,p,z),1)) <-(1)Holds(clear(z),1),(2)Diff(B,z)
(15)
(14)
(5)
(7)
(6)
Poss(result(t(B,p,C),1)) <-
(12)
(11)
Poss(result(t(x,y,z),5)) <-(2)Manip(x),(1)Holds(clear(x),5),
(15)
Holds(clear(z),5),
(14)
Holds(on(x,y),5), Diff(x,z)
(15)
(14)
(7)
(5)
(6)
(8)
Poss(result(t(A,y,z),5)) <- Holds(clear(z),5),
(15)
Holds(on(A,y),5),Diff(A,z)
(15)
(13)



The eventual failure of the alternative attempts to solve the subgoals

```
Holds(on(A,y),5)
Holds(on(A,B),6)  and
Holds(on(B,C),6)
```

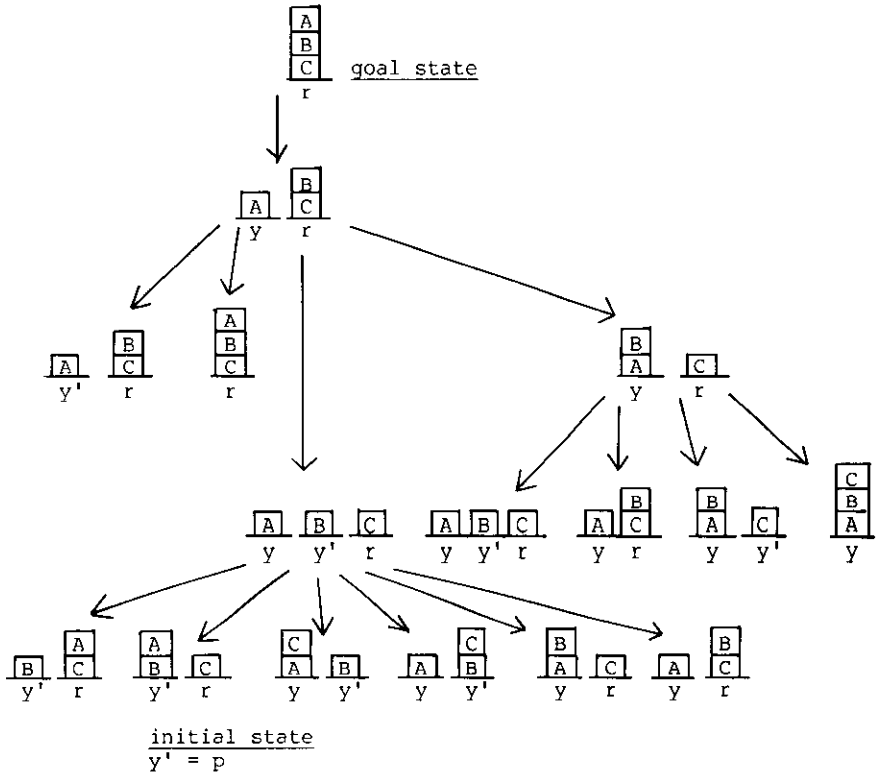
can be hastened by strengthening the restrictions on the frame axiom. The more restrictive version of the frame axiom

```
Holds(u, result(trans(x,y,z), w)) <- Holds(u,w),
                                     Diff(u, on(x,v)),
                                     Diff(u, clear(z)),
                                     Diff(u, clear(y))
```

in particular, fails immediately whenever one of the clauses (13) or (14) succeeds.

Top-down execution of the state space and frame axioms

Part of the search space of states determined by executing the state space axiom top-down is illustrated below. As in the case where the state space axiom is executed bottom-up, redundancy arises when the same object is picked up twice in succession. The variables y and y' name locations which have not yet been determined.



In the following solution all clauses are executed top-down. Subgoals are considered breadth-first and left to right in the order in which they are written. Duplicate subgoals are deleted. To save space, steps involving the solution of subgoals containing the predicate symbol Diff are not illustrated.

```
    <- Holds(on(A,B),w), Holds(on(B,C),w),
13     Holds(on(C,r),w), Poss(w)
15
15      w = result(trans(A,y,B),w1)
12
    <- Holds(on(B,C),w1), Holds(on(C,r),w1), Poss(w1),
13     Manip(A), Holds(clear(A),w1), Holds(clear(B),w1),
15     Holds(on(A,y),w1), Diff(A,B)
12
 8
15      w1 = result(trans(B,y',C),w2)
15
15
    <- Holds(on(C,r),w2), Poss(w2), Manip(B),
15     Holds(clear(B),w2), Holds(clear(C),w2),
12     Holds(on(B,y'),w2), Diff(B,C), Holds(clear(A),w2),
 9     Holds(clear(B),w2), Holds(on(A,y),w2)
14
15
15      w2 = result(trans(A,B,y),w3)
15
13
    <- Holds(on(C,r),w3), Poss(w3), Manip(A),
 4     Holds(clear(A),w3), Holds(clear(y),w3),
 1     Holds(on(A,B),w3), Diff(A,y), Holds(clear(C),w3),
 8     Holds(on(B,y'),w3), Holds(clear(A),w3)
 5
 6
 2      w3 = ∅   y = q   y' = p
 7
 3
       □
```

Applications of plan-formation

The principal application of plan-formation has been the construction of plans for robot-like machinery. Plan-formation has also been applied to the automatic construction of programs from specifications. The description of the input and the output states constitutes a specification of a program. The definition of the preconditions and of the statements added and deleted by actions expresses the semantics of the machine operations. A plan consists of a sequence of machine operations and represents a program. More elaborate systems of plan-formation include procedures for constructing plans with conditional statements, loops and other operations. Horn clause plan-formation programs written by Warren [1974, 1976] and Moss [1977] have been applied to program construction.

An application of plan-formation to the synthesis of organic compounds was developed by Fogel, while a high school student, at Imperial College during the summer of 1977. Chemical compounds, like states in plan-formation, can be described by assertions concerning the objects (atoms and bonds) which belong to them. The statement that

bond b of strength s holds between the
atoms a_1 and a_2 in the compound c

can be expressed by a single n-ary relationship

Bond(b,s,a_1,a_2,c) <-

or by several binary relationships:

b has strength s <-
b bonds a_1 <-
b bonds a_2 <-
b belongs to c <-

An initial compound functions as an initial state and a goal compound as a goal state. Chemical reactions are actions which transform one compound into another. They are defined by specifying (1) the preconditions which must hold before a reaction can take place, (2) the new bonds which the reaction introduces and (3) the old bonds which the reaction destroys. A frame axiom states that bonds which are not destroyed by a reaction are preserved by it. Both the program written by Moss and the one written by Fogel were implemented as Horn clause programs and run on a PROLOG-like system developed at Imperial College.

Programs for drug analysis have been written in PROLOG at the Ministry of Heavy Industry in Budapest [Futo, Darvas and Szeredi 1978]. These use relational data structures similar to those in the organic synthesis program. Because many of the properties of a given drug may be unknown, the drug analysis programs employ binary rather than n-ary relations. The programs have led to useful discoveries concerning previously unknown drug interactions and concerning inconsistencies in descriptions of drugs in the pharmaceutical literature.

Limitations

The approach taken in this chapter stores information about the initial state explicitly and uses the frame axiom to compute information about later states. It can be argued that this is unnatural and potentially inefficient. The alternative, when using depth-first search and reasoning forward from the initial state, is to store the current state explicitly and to compute information about earlier states. The two approaches are intuitively equivalent. The problem of formally explaining and justifying the equivalence, however, has still to be solved.

The treatment of plans as sequences of actions is another limitation, which creates redundancies when actions do not interact and can be performed in parallel. Performing the actions in sequence produces the same results redundantly in any sequence. Systems for generating plans which are partially ordered collections of actions have been described by Sacerdoti [1975] and Tate [1974]. A Horn clause program which generates partially ordered plans has also been written in PROLOG by Warren. A survey of plan-formation systems and a comparison with the one presented in this chapter has been made by Waldinger [1977].

Exercises

1) Formulate an n-tuple representation of the blocks world problem. Let State(x,y,z) hold when it is possible for block A to be on x, B on y and C on z simultaneously. Compare problem-solving strategies for the n-tuple representation with those for the binary representation of the problem.

2) Reformulate the water container problem investigated in Chapter 4 as a plan-formation problem using the binary representation investigated in this chapter. Compare the problem-solving strategies needed for efficient solution of the problem in both the n-ary and binary representations.

3) The assigment statement of conventional programming languages can be regarded as an action which transforms one state of a computer into another. The new state

assign(u,v,w)

differs from the preceding state w in that the location u contains v.

Assume that A, B and C are locations and that in the initial state Ø they contain a, b and c respectively. The problem is to find a state in which the initial values of A and B are interchanged.

Formulate and solve the problem as a plan-formation task.