SRI International

How to Clear a Block: A Theory of Plans

Technical Note 397

December 1986

By: Zohar Manna
Computer Science Department
Stanford University

Richard Waldinger
Artificial Intelligence Center
Computer and Information Sciences Division

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

This research was supported by the National Science Foundation under Grants DCR-82-14523 and DCR-85-12356, by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-85-0383, by the Office of Naval Research under Contract N00014-84-C-0706, by United States Army Research under Contract DAJA-45-84-C-0040, and by a contract from the International Business Machines Corporation.

Preliminary versions of parts of this paper were presented at the *Eighth International Conference on Automated Deduction,* Oxford, England, July 1986, and the *Workshop on Planning and Reasoning about Actions,* Timberline, Oregon, July 1986.

SRI International®
333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

ABSTRACT

Problems in commonsense and robot planning are approached by methods adapted from program synthesis research; planning is regarded as an application of automated deduction. To support this approach, we introduce a variant of situational logic, called *plan theory*, in which plans are explicit objects.

A machine-oriented deductive-tableau inference system is adapted to plan theory. Equations and equivalences of the theory are built into a unification algorithm for the system. Frame axioms are built into the resolution rule.

Special attention is paid to the derivation of conditional and recursive plans. Inductive proofs of theorems for even the simplest planning problems, such as clearing a block, have been found to require challenging generalizations.

This research was supported by the National Science Foundation under Grants DCR-82-14523 and DCR-85-12356, by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, by the United States Air Force Office of Scientific Research under Contract AFOSR-85-0383, by the Office of Naval Research under Contract N00014-84-C-0706, by United States Army Research under Contract DAJA-45-84-C-0040, and by a contract from the International Business Machines Corporation.

Preliminary versions of parts of this paper were presented at the *Eighth International Conference on Automated Deduction*, Oxford, England, July 1986, and the *Workshop on Planning and Reasoning about Actions*, Timberline, Oregon, July 1986.

1. INTRODUCTION

For many years, the authors have been working on *program synthesis*, the automated derivation of a computer program to meet a given specification. We have settled on a deductive approach to this problem, in which program derivation is regarded as a task in theorem proving (Manna and Waldinger [80], [85a]). To construct a program, we prove a theorem that establishes the existence of an output meeting the specified conditions. The proof is restricted to be constructive, in that it must describe a computational method for finding the output. This method becomes the basis for the program we extract from the proof.

For the most part, we have focused on the synthesis of *applicative* programs, which yield an output but produce no side effects. We are now interested in adapting our deductive approach to the synthesis of *imperative* programs, which may alter data structures or produce other side effects.

Plans are closely analogous to imperative programs, in that actions may be regarded as computer instructions, tests as conditional branches, and the world as a huge data structure. This analogy suggests that techniques for the synthesis of imperative programs may carry over into the planning domain. Conversely, we may anticipate that insights we develop by looking at a relatively simple planning domain, such as the blocks world, would then carry over to program synthesis in a more complex domain, involving array assignments, destructive list operations, and other alterations of data structures.

Consider the problem of clearing a given block, where we are not told whether the block is already clear or, if not, how many blocks are above it. Assume that we are in a blocks world in which blocks are all the same size, so that only one block can fit directly on top of another, and in which the robot arm may lift only one block at a time. Then we might expect a planning system to produce the following program:

$$makeclear(a) \Leftarrow \begin{cases} \textit{if } clear(a) \\ \textit{then } \Lambda \\ \textit{else } makeclear(hat(a)); \\ \quad\quad put(hat(a), table). \end{cases}$$

In other words, to clear a given block a (the *argument*), first determine whether it is already clear. If not, clear the block that is on top of block a, and then put that block on the table. Here Λ is the empty sequence of instructions, corresponding to no action at all, and $hat(a)$ is the block directly on a, if one exists. The action $put(u, v)$ places the block u on top of the object v.

Note that the *makeclear* program requires a conditional (*if-then-else*) and a recursive call to *makeclear* itself. Planning systems have often attempted to avoid constructing plans using these constructs by dealing with completely known worlds. Had we known exactly how many blocks were to be on top of block a, for example, we could have produced a

plan with no conditionals and no recursion. Once we begin to deal with an uncertain environment, we are forced to introduce some constructs for testing and for repetition.

A fundamental difficulty in applying a theorem-proving approach to plan construction is that the meaning of an expression in a plan depends on the situation, whereas in ordinary logic the meaning of an expression does not change. Thus, the block designated by $hat(a)$ or the truth-value designated by $clear(a)$ may change from one state to the next. The traditional approach to circumventing this difficulty relies on a *situational logic*, i.e., one in which we can refer explicitly to situations or states of the world.

2. THE TROUBLE WITH SITUATIONAL LOGIC

In this section, we describe conventional situational logic and point out some of its deficiencies when applied to planning. These deficiencies motivate the introduction of our own version of situational logic, called "plan theory."

Conventional Situational Logic

Situational logic was introduced into the literature of computer science by McCarthy [63]. A variant of this logic was incorporated into the planning system QA3 (Green [69]). In the QA3 logic, function and predicate symbols whose values might change were given state arguments. Thus, rather than speaking about $hat(x)$ or $clear(x)$, we introduce the *situational function* symbol $hat'(w,x)$ and the *situational predicate* symbol $Clear(w,x)$, each of which is given an explicit state argument w; for example, $hat'(w,x)$ is the block on top of block x in state w. Actions are represented as functions that yield states; for example, $put'(w,x,y)$ is the state obtained from state w by putting block x on object y.

Facts about the world may be represented as axioms in situational logic. For example, the fact that the hat of an unclear block is on top of the block is expressed by the axiom

$$\begin{array}{l} \textit{if not } Clear(w,x) \\ \textit{then } On(w,\ hat'(w,x),\ x). \end{array}$$

Actions can also be described by situational-logic axioms. For example, the fact that after block x has been put on the table, block x is indeed on the table is expressed by the axiom

$$\begin{array}{l} \textit{if } Clear(w,x) \\ \textit{then } On(put'(w,x,table),\ x,\ table). \end{array}$$

In a conventional situational logic, such as the QA3 logic, to construct a plan that will meet a specified condition, one proves the existence of a state in which the condition is true. More precisely, let us suppose that the condition is of the form $\mathcal{Q}[s_0,a,z]$, where

s_0 is the initial state, a the argument or input parameter, and z the final state. Then the theorem to be proved is

$$(\forall s_0)(\forall a)(\exists z)\mathcal{Q}[s_0,\ a,\ z].$$

For example, the plan to clear a block is constructed by proving the theorem

$$(\forall s_0)(\forall a)(\exists z)\,Clear(z, a).$$

From a situational-logic proof of this theorem, using techniques for the synthesis of applicative programs, one can extract the program

$$makeclear'(s_0, a) \Leftarrow \begin{cases} \textit{if } Clear(s_0, a) \\ \textit{then } s_0 \\ \textit{else let } s_1 \textit{ be } makeclear'(s_0,\ hat'(s_0, a)) \textit{ in} \\ \quad put'(s_1,\ hat'(s_1, a),\ table). \end{cases}$$

This program closely resembles the *makeclear* program we proposed initially, except that it invokes situational operators, which contain explicit state arguments.

Executable and Nonexecutable Plans

It would seem that, by regarding plans as state-producing functions, we can treat an imperative program as a special kind of applicative program and use the same synthesis methods for both. In other words, we can perhaps extract programs from situational-logic proofs and regard these programs as plans. Unfortunately, there are some programs we can extract from proofs in this formulation of situational logic that cannot be regarded as plans.

For example, consider the problem illustrated in Figure 1. The monkey is presented with two boxes and is informed that one box contains a banana and the other a bomb, but he is not told which. His goal is to get the banana, but if he goes anywhere near the bomb it will explode. As stated, the problem should have no solution. However, if we formulate the problem in conventional situational logic, we can prove the appropriate theorem,

$$(\forall s_0)(\exists z)\,Hasbanana(z).$$

The "program" we extract from one proof of this theorem is

$$getbanana(s_0) \Leftarrow \begin{cases} \textit{if } Hasbanana(goto'(s_0, a)) \\ \textit{then } goto'(s_0, a) \\ \textit{else } goto'(s_0, b)\ . \end{cases}$$

According to this plan, the monkey should ask whether, if it were to go to box a, it would get the banana? If so, it should go to box a; otherwise, it should go to box b. We cannot execute this "plan" because it allows the monkey to consider whether a given

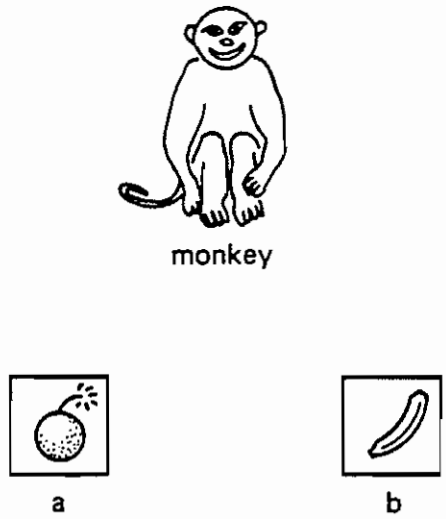


Fig. 1: *The Monkey, the Banana, and the Bomb*

proposition *Hasbanana* is true in a hypothetical state $goto'(s_0, a)$, which is different from the current state s_0.

We would like to restrict the proofs in situational logic to be constructive, in the sense that the programs we extract should correspond to executable plans. This kind of consideration has influenced the design of our version of situational logic, called *plan theory.*

3. PLAN THEORY

In plan theory we have two classes of expressions. The *static* (or *situational*) *expressions* denote particular objects, states, and truth-values. For example, the static expressions $hat'(s, b)$, $Clear(s, b)$, and $put'(s, b, c)$ denote a particular block, truth-value, and state, respectively (where b and c denote blocks and s denotes a state). We shall also introduce corresponding *fluent terms*, which will not denote any particular object, truth-value, or state, but which will *designate* such elements with respect to a given state. For example, the fluent terms

$$hat(d), \quad clear(d), \quad \text{and} \quad put(d, \hat{d})$$

will only designate a block, truth-value, or state, respectively, with respect to a given state (where d and $\hat{d}$ are themselves fluent terms that designate blocks).

Fluent terms themselves do not refer to any state explicitly. To see what element a

fluent term e designates with respect to a given state s, we apply a *linkage operator* to s and e, obtaining a static expression. We use one of three linkage operators,

$$s{:}e, \quad s :: e, \quad \text{or} \quad s;e \ ,$$

depending on whether e designates an object, truth-value, or state, respectively. For example, the static expressions

$$s{:}hat(d) \quad s :: clear(d), \quad \text{and} \quad s;put(d, \hat{d})$$

will indeed denote a particular block, truth-value, and state, respectively.

While we shall retain static expressions as specification and proof constructs, we shall restrict our proofs to be constructive in the sense that the programs we extract from them will contain no static expressions, but only fluent terms. Because fluent terms do not refer to states explicitly, this means that the knowledge of the agent will be restricted to the implicit current state; it will be unable to tell what, say, the hat of a given block is in a hypothetical or future state. In this way, we ensure that the programs we extract may be executed as plans. Nonplans, such as the *getbanana* "program" mentioned above, will be excluded.

Now let us describe plan theory in more detail.

Elements of Plan Theory

Plan theory is a theory in first-order predicate logic that admits several sorts of terms.

- The *static* (*situational*) *terms*, or *s-terms*, denote a particular element. They include
  - *object* s-terms, which denote an object, such as a block or the table.
  - *state* s-terms, which denote a state.

For example, $hat'(s, b)$ is an object s-term and $put'(s, b, c)$ is a state s-term, if s is a state s-term and b and c are object s-terms.

- The *static* (*situational*) *sentences*, or *s-sentences*, denote a particular truth-value.

For example, $Clear(s, b)$ is an s-sentence, if s is a state s-term and b an object s-term .

- The *fluent terms*, or *f-terms*, only designate an element with respect to a given state. They include
  - *object* f-terms, which designate an object with respect to a given state.
  - *propositional* f-terms, which designate a truth-value with respect to a given state.
  - *plan* f-terms, which designate a state with respect to a given state.

For example, $hat(d)$, $clear(d)$, and $put(d, \hat{d})$ are object, propositional, and plan f-terms, respectively. The plan f-constant Λ denotes the empty plan.

Object f-terms denote *object fluents*, propositional f-terms denote *propositional fluents*, and plan f-terms denote *plans*. We may think of object fluents, propositional fluents, and plans as functions mapping states into objects, truth-values, and states, respectively. Syntactically, however, they are denoted by terms, not function symbols. To determine what elements these terms designate with respect to a given state, we invoke the *in* function ":", the *in* relation "::", and the execution function ";" .

The *in* Function ":"

If s is a state s-term and e an object f-term,

$$s{:}e$$

is an object s-term denoting the object *designated by e in state s*. For example, $s_0{:}hat(d)$ denotes the object designated by the object f-term $hat(d)$ in state s_0.

In general, we shall introduce object f-function symbols $f(u_1, \ldots, u_n)$ and object s-function symbols $f'(w, x_1, \ldots, x_n)$ together, where f takes object fluents $u_1, \ldots, u_n$ as arguments and yields an object fluent, while f' takes a state w and objects $x_1, \ldots, x_n$ as arguments and yields an object. The two symbols are linked in each case by the *object linkage* axiom

$$w{:}f(u_1, \ldots, u_n) \;=\; f'(w,\; w{:}u_1,\; \ldots,\; w{:}u_n) \qquad (\textit{object linkage})$$

(Implicitly, variables in axioms are universally quantified. For simplicity, we omit sort conditions such as $state(w)$ from the axioms.)

For example, corresponding to the object f-function $hat(u)$, which yields a block fluent, we have an object s-function $hat'(w, x)$, which yields a fixed block. The appropriate instance of the *object linkage* axiom is

$$w{:}hat(u) \;=\; hat'(w, w{:}u).$$

Thus $s{:}hat(d)$ denotes the block on top of block $s{:}d$ in state s. (This is not necessarily the same as the block on top of $s{:}d$ in some other state s'.)

The *in* Relation "::"

The *in* relation :: is analogous to the *in* function :. If s is a state s-term and e a propositional f-term,

$$s :: e$$

is a proposition denoting the truth-value *designated by e in state s*. For example, $s_0 :: clear(d)$ denotes the truth-value designated by the propositional f-term $clear(d)$ in state s_0.

In general, we shall also introduce propositional f-function symbols $r(u_1, \ldots, u_n)$ and s-predicate symbols $R(w, x_1, \ldots, x_n)$ together, with the convention that r takes object fluents $u_1, \ldots, u_n$ as arguments and yields a propositional fluent, while R takes a state w and objects $x_1, \ldots, x_n$ as arguments and yields a truth-value. The two symbols are linked in each case by the *propositional-linkage* axiom

$$w :: r(u_1, \ldots, u_n) \equiv R(w, w{:}u_1, \ldots, w{:}u_n) \qquad \textit{(propositional linkage)}$$

For example, corresponding to the propositional f-function $clear(u)$, which yields a propositional fluent, we have an actual relation $Clear(w, x)$, which yields a truth-value. The instance of the *propositional-linkage* axiom that relates them is

$$w :: clear(u) \equiv Clear(w, w{:}u).$$

Thus $s :: clear(d)$ is true if the block $s{:}d$ is clear in state s.

The Execution Function ";"

If s is a state s-term and p a plan f-term,

$$s;p$$

is a state s-term denoting the state obtained by *executing plan p in state s*. For example, $s; put(d, \hat{d})$ is the state obtained by putting block d on object $\hat{d}$ in state s.

In general, we shall introduce plan f-function symbols $g(u_1, \ldots, u_n)$ and state s-function symbols $g'(w, x_1, \ldots, x_n)$ together, where g takes object fluents $u_1, \ldots, u_n$ as arguments and yields a plan, while g' takes a state w and objects $x_1, \ldots, x_n$ as arguments and yields a new state. The two symbols are linked in each case by the *plan linkage* axiom

$$w;g(u_1, \ldots, u_n) = g'(w, w{:}u_1, \ldots, w{:}u_n) \qquad \textit{(plan linkage)}$$

For example, corresponding to the plan f-function $put(u, v)$, which takes object fluents u and v as arguments and produces a plan, we have a state s-function $put'(w, x, y)$, which takes a state w and the actual objects x and y as arguments and produces a new state. The appropriate instance of the *plan linkage* axiom is

$$w;put(u, v) = put'(w, w{:}u, w{:}v).$$

The empty plan Λ is taken to be a right identity under the execution function; that is,

$$w;\Lambda = w \qquad \textit{(empty plan)}$$

for all states w.

Rigid Designator

Certain fluent constants (f-constants) are to denote the same object regardless of the state. For example, we may assume that the constants *table* and *banana* always denote

the same objects. In this case, we shall identify the object fluent with the corresponding fixed object.

An object f-constant u is a *rigid designator* if

$$w{:}u = u \qquad (\textit{rigid designator})$$

for all states w.

For example, the fact that *table* is a rigid designator is expressed by the axiom

$$w{:}table = table$$

for all states w. In the derivation of a plan, we shall assume that our argument (or input parameter) a is a rigid designator. On the other hand, some f-constants, such as *here*, *the-highest-block*, or *the-president*, are likely not to be rigid designators.

The Composition Function ";;"

We introduce a notion of composing plans.

If p_1 and p_2 are plan f-terms, $p_1;;p_2$ is the *composition* of p_1 and p_2.

Executing $p_1;;p_2$ is the same as executing first p_1 and then p_2. This is expressed by the *plan composition* axiom

$$w;(p_1;;p_2) = (w;p_1);p_2 \qquad (\textit{plan composition})$$

for all states w and plans p_1 and p_2. Normally we shall ignore the distinction between the composition function ;; and the execution function ; , writing ; for both and relying on context to make the meaning clear.

Composition is assumed to be associative; that is

$$(p_1;;p_2);;p_3 = p_1;;(p_2;;p_3) \qquad (\textit{associativity})$$

for all plans p_1, p_2, and p_3. For this reason, we may write $p_1;;p_2;;p_3$ without parentheses.

The empty plan Λ is taken to be the identity under composition, that is,

$$\Lambda;;p = p;;\Lambda = p \qquad (\textit{identity})$$

for all plans p.

Specifying Facts and Actions

As in conventional situational logic, facts about the world may be expressed as plan theory axioms. For example, the principal property of the *hat* function is expressed by the *hat* axiom

$$\begin{array}{l} \textit{if } not\, Clear(w, y) \\ \textit{then } On(w,\ hat'(w, y),\ y) \end{array} \qquad (\textit{hat})$$

for all states w and blocks y. (As usual, for simplicity, we omit sort conditions such as $state(w)$ from the antecedent of the axiom.) In other words, if block y is not clear, its hat is directly on top of it. (If y is clear, its hat is a "nonexistent" object, not a block.) It follows, if we take y to be $w{:}v$ and apply the *propositional* and *object linkage* axioms, that

$$\begin{array}{l} \mathit{if}\ \mathit{not}\,(w :: \mathit{clear}(v)) \\ \mathit{then}\ w :: \mathit{on}(\mathit{hat}(v),\, v). \end{array}$$

for all states w and block fluents v. Other axioms are necessary for expressing other properties of the *hat* function.

The effects of actions may also be described by plan theory axioms. For example, the primary effect of putting a block on the table may be expressed by the ***put-table-on*** axiom

$$\begin{array}{ll} \mathit{if}\ \mathit{Clear}(w,\, x) & \\ \mathit{then}\ \mathit{On}(\mathit{put}'(w, x, \mathit{table}),\, x,\, \mathit{table}) & (\textit{put-table-on}) \end{array}$$

for all states w and blocks x. The axiom says that after a block has been put on the table, the block will indeed be on the table, provided that it was clear beforehand. (The effects of attempting to move an unclear block are not specified and are therefore unpredictable.) It follows, if we take x to be $w{:}u$ and apply the *linkage* axioms plus the rigidity of the designator *table*, that

$$\begin{array}{l} \mathit{if}\ w :: \mathit{clear}(u) \\ \mathit{then}\ \mathit{On}(w{;}\mathit{put}(u, \mathit{table}),\, w{:}u,\, \mathit{table}) \end{array}$$

for all states w and block fluents u.

Note that, in the consequent of the above property, we cannot conclude that

$$(w{;}\mathit{put}(u, \mathit{table})) :: \mathit{on}(u, \mathit{table}),$$

that is, that after putting u on the table, u will be on the table. This is because u is a fluent and we have no way of knowing that it will designate the same block in state $w;\mathit{put}(u, \mathit{table})$ that it did in state w. For example, if u is taken to be $\mathit{hat}(a)$, the property allows us to conclude that, if $s_0 :: \mathit{clear}(\mathit{hat}(a))$, then

$$\mathit{On}(s_0{;}\mathit{put}(\mathit{hat}(a), \mathit{table}),\ s_0{:}\mathit{hat}(a),\ \mathit{table}).$$

In other words, the block that was on block a initially is on the table after execution of the plan step. On the other hand, we cannot conclude that

$$(s_0{;}\mathit{put}(\mathit{hat}(a), \mathit{table})) :: \mathit{on}(\mathit{hat}(a),\, \mathit{table}),$$

that is, that $\mathit{hat}(a)$ is on the table after the plan step has been executed. In fact, in this state, a is clear and $\mathit{hat}(a)$ no longer designates a block.

Plan Formation

To construct a plan for achieving a condition $Q[s_0, a, z]$, where s_0 is the initial state, a the input object, and z the final state, we prove the theorem

$$(\forall s_0)(\forall a)(\exists z_1)Q[s_0,\ a,\ s_0;z_1].$$

Here z_1 is a plan variable. In other words, we show, for any initial state s_0 and input object a, the existence of a plan z_1 such that, if we are in state s_0 and execute plan z_1, we obtain a state in which the specified condition Q is true. A program producing the desired plan is extracted from the proof of this theorem. Informally, we often speak of this program as a plan itself, although in fact it computes a function that only produces a plan when it is applied to an argument.

Note that, in the QA3 version of situational logic, one proves instead the theorem

$$(\forall s_0)(\forall a)(\exists z)Q[s_0,\ a,\ z].$$

The phrasing of the theorem in plan theory ensures that the final state z can indeed be obtained from s_0 by the execution of a plan z_1. For example, the plan for clearing a block is constructed by proving the theorem

$$(\forall s_0)(\forall a)(\exists z_1)[Clear(s_0;z_1,\ a)].$$

In other words, the block a is to be clear after execution of the desired plan z_1 in the initial state s_0.

In the balance of this paper, we present a machine-oriented deductive system for plan theory in which we can prove such theorems and derive the corresponding plans at the same time. We shall use the proof of the above theorem, together with the concomitant derivation of the *makeclear* plan, as a continuing example.

4. THE PLAN-THEORY DEDUCTIVE SYSTEM

To support the synthesis of applicative programs, we developed a *deductive-tableau* theorem-proving system (Manna and Waldinger [80], [85a]), which combines nonclausal resolution, well-founded induction, and conditional term rewriting within a single framework. In this paper, we carry the system over into plan theory. Although a full introduction to the deductive-tableau system is not possible here, we describe just enough to make this paper self-contained.

Deductive Tableaux

The fundamental structure of the system, the *deductive tableau*, is a set of rows, each of which contains a plan theory sentence, either an *assertion* or a *goal*, and an optional

term, the *plan entry.* We can assume that the sentences are quantifier-free. Let us forget about the plan entry for a moment.

Under a given interpretation, a tableau is *true* whenever the following condition holds:

> If all instances of each of the assertions are true,
> then some instance of at least one of the goals is true.

Thus, variables in assertions have tacit universal quantification, while variables in goals have tacit existential quantification. In a given theory, a tableau is *valid* if it is true under all models for the theory.

To prove a given sentence valid, we remove its quantifiers (by skolemization) and enter it as the initial goal in a tableau. Any other valid sentences of the theory that we are willing to assume may be entered into the tableau as assertions. The resulting tableau is valid if and only if the given sentence is valid.

The deduction rules add new rows to the tableau without altering its validity; in particular, if the new tableau is valid, so is the original tableau. The deductive process continues until we derive as a goal the propositional constant *true*, which is always true, or until we derive as an assertion the propositional constant *false*, which is always false. The tableau is then automatically valid; hence the original sentence is too.

In deriving a plan $f(a)$, we prove a theorem of form

$$(\forall s_0)(\forall a)(\exists z_1)\mathcal{Q}[s_0,\ a,\ s_0;z_1].$$

In skolemizing this, we obtain the sentence

$$\mathcal{Q}[s_0,\ a,\ s_0;z_1],$$

where s_0 and a are skolem constants and z_1 is a variable. (Since this sentence is a theorem or goal to be proved, its existentially quantified variables remain variables, while its universally quantified variables become skolem constants or functions. The intuition is that we are free to choose values for the existentially quantified variables, whereas the values for the universally quantified variables are imposed on us. The situation is precisely the opposite for axioms or assertions.)

To prove this theorem, we establish the validity of the initial tableau

assertions	goals	plan: $s_0;f(a)$
	$\mathcal{Q}[s_0,\ a,\ s_0;z_1]$	$s_0;z_1$

For example, the initial tableau for the *makeclear* derivation is

assertions	goals	plan: $s_0;makeclear(a)$
	1. $Clear(s_0;z_1,\ a)$	$s_0;z_1$

Certain valid sentences of plan theory, such as the axioms for blocks-world actions, would be included as assertions.

Plan Entry

Note that the initial tableau includes a plan entry $s_0;z_1$. The plan entry is the mechanism for extracting a plan from a proof of the given theorem. Throughout the derivation, we maintain the following *correctness* property:

> For any model of the theory, and for any goal [or assertion] in the tableau,
> if some instance of the goal is true [assertion is false],
> then the corresponding instance $s_0;t$ of the plan entry (if any)
> will satisfy the *specified condition* $\mathcal{Q}[s_0,\ a,\ s_0;t]$.

In other words, executing the plan t produces a state $s_0;t$ that satisfies the specified condition. The initial goal already satisfies the property in a trivial way, since it is the same as the specified condition. Each of the deduction rules of our system preserves this correctness property, as well as the validity of the tableau.

If a goal [or assertion] has no plan entry, this means that any plan will satisfy the specified condition if some instance of that goal is true [assertion is false]. In other words, we do not care what happens in that case.

Basic Properties

It may be evident that there is a duality between assertions and goals; namely, in a given theory,

a tableau that contains an assertion $\mathcal{A}$ is valid
if and only if
the tableau that contains instead the goal $(not\,\mathcal{A})$, with the same plan entry, is valid.

On the other hand,

a tableau that contains a goal $\mathcal{G}$ is valid
if and only if
the tableau that contains instead the assertion $(not\,\mathcal{G})$, with the same plan entry, is valid.

This means that we could shift all the goals into the assertion column simply by negating them, thereby obtaining a refutation procedure; the plan entries and the correctness

properties would be unchanged. (This is done in conventional resolution theorem-proving systems.) Or we could shift all the assertions into the goal column by negating them. Nevertheless, the distinction between assertions and goals has intuitive significance, so we retain it in our exposition.

Two other properties of tableaux are useful. First, the variables of any row in the tableau are dummies and may be renamed systematically without changing the tableau's validity or correctness. Second, we may add to a tableau any instance of any of its rows, preserving the validity and correctness.

Primitive Plans

We want to restrict our proofs to be sufficiently constructive so that the plans we extract can be executed. For this purpose, we distinguish between *primitive* symbols, which we know how to execute, and *nonprimitive* symbols, which we do not. For example, we regard the function symbols : and *hat'* and the predicate symbols :: and *Clear* as nonprimitive, because we do not want to admit them into our plans. On the other hand, we regard the f-function symbols *hat* and *clear* as primitive.

In deriving a plan, we shall maintain the *primitivity* property, namely, that the final segment t of the plan entry $s_0;t$ for any assertion or goal of the tableau shall be composed entirely of primitive symbols. Otherwise the new row is discarded.

Extracting the Plan

As we have mentioned, the deductive process continues until we derive either the final goal *true* or the final assertion *false*. At this point, the proof is complete and we may extract the plan

$$f(a) \Leftarrow t,$$

where $s_0;t$ is the plan entry associated with the final row.

This is because we have maintained the correctness property that the plan entry of any goal [or assertion] must satisfy the specified condition $Q[s_0, a, s_0;t]$ when that goal [or assertion] is true [or false]. Since the truth symbol *true* is always true and the truth symbol *false* always false, the plan entry $s_0;t$ will always satisfy the specified condition. We know also that the extracted plan will be executable, because we have maintained the primitivity property, which requires that the plan term t be expressed exclusively in terms of primitive symbols. (Should the final plan still contain variables, these may be replaced by any primitive terms.)

In the next section we begin to introduce the deduction rules of our system, emphasizing those that need to be adapted for plan theory or that play a major role in plan derivations.

5. FORMATION OF CONDITIONALS

The resolution rule accounts for the introduction of conditionals, or tests, into the derived plan and also is important for ordinary reasoning. Because a special adaptation of the rule is necessary to form conditionals in plan theory without introducing the nonprimitive predicate symbol :: into the plan, we first consider applications of the rule that do not form conditionals.

The Resolution Rule: Ground Version

We begin by disregarding the plan entries and considering the ground version, in which there are no variables. We describe the rule in a tableau notation.

assertions	goals
$\mathcal{F}[\mathcal{P}]$	
$\mathcal{G}[\mathcal{P}]$	
$\mathcal{F}[\mathit{true}]$ *or* $\mathcal{G}[\mathit{false}]$	

More precisely, if our tableau contains two assertions, $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$, which share a common subsentence $\mathcal{P}$, we may replace all occurrences of $\mathcal{P}$ in $\mathcal{F}[\mathcal{P}]$ with *true*, replace all occurrences of $\mathcal{P}$ in $\mathcal{G}[\mathcal{P}]$ with *false*, take the disjunction of the results, and (after propositional simplification) add it to the tableau as a new assertion.

The rationale for this rule is as follows. We suppose that $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$ are true under a given model, and show that ($\mathcal{F}[\mathit{true}]$ *or* $\mathcal{G}[\mathit{false}]$) is then also true. We distinguish between two cases. In the case in which $\mathcal{P}$ is true, because $\mathcal{F}[\mathcal{P}]$ is true, its equivalent $\mathcal{F}[\mathit{true}]$ is true. On the other hand, in the case in which $\mathcal{P}$ is false, because $\mathcal{G}[\mathcal{P}]$ is true, its equivalent $\mathcal{G}[\mathit{false}]$ is true. In either case, the disjunction ($\mathcal{F}[\mathit{true}]$ *or* $\mathcal{G}[\mathit{false}]$) is true.

Note that the rule is asymmetric in its treatment of $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}]$. In fact, it can be restricted according to the "polarity" of the occurrences of $\mathcal{P}$, the common subsentence. We may require that some occurrence of $\mathcal{P}$ in $\mathcal{F}[\mathcal{P}]$ be of *negative polarity* (i.e., it must be within the scope of an odd number of implicit or explicit negations) and that some occurrence of $\mathcal{P}$ in $\mathcal{G}[\mathcal{P}]$ be of *positive polarity* (i.e., it must be within the scope of an even number of implicit or explicit negations). The antecedent of an implication is considered to be within the scope of an implicit negation. Thus, in applying the rule between two assertions

$$(\mathit{if}\ P\ \mathit{then}\ Q) \quad \text{and} \quad (P\ \mathit{or}\ R),$$

the role of $\mathcal{F}[\mathcal{P}]$ must be played by ($\mathit{if}\ P^{-}\ \mathit{then}\ Q$), in which P has negative polarity, and the role of $\mathcal{G}[\mathcal{P}]$ by ($P^{+}\ \mathit{or}\ R$), in which P has positive polarity, yielding the new assertion

$$(\mathit{if}\ \mathit{true}\ \mathit{then}\ Q)\ \mathit{or}\ (\mathit{false}\ \mathit{or}\ R),$$

that is, after propositional simplification, $(Q \text{ or } R)$. Reversing the roles of the two assertions yields the trivial assertion *true*, which is of no value in the proof. This strategy has been shown by Murray [82] to retain completeness for first-order logic.

If only one of the goals has a plan entry, the new goal is given the same plan entry. (The case in which both goals have plan entries requires the introduction of a conditional plan and is treated separately.)

We have applied the rule between two assertions but, by duality, the rule can just as well be applied between two goals or between an assertion and a goal. In these cases, a new goal is introduced, which is a conjunction rather than a disjunction. In applying the polarity strategy, each goal must be considered to be within the scope of an implicit negation.

We assume that all the sentences in a tableau are subjected to full propositional simplification. Rules such as

$$\mathcal{P} \text{ and } true \rightarrow \mathcal{P}$$
$$\mathcal{P} \text{ and } \mathcal{P} \rightarrow \mathcal{P}$$
$$not(not\,\mathcal{P}) \rightarrow \mathcal{P}$$

are applied repeatedly wherever possible before an assertion or goal is entered. Simplification is always necessary when the resolution rule is applied.

The Resolution Rule: General Version

We have up to now been considering the ground case, in which the sentences have no variables. In the general case, the rule may be expressed as follows:

assertions	goals
$\mathcal{F}[\mathcal{P}]$	
$\mathcal{G}[\mathcal{P}']$	
$\mathcal{F}\theta[true] \text{ or } \mathcal{G}\theta[false]$	

More precisely, let us suppose that our tableau contains two assertions $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}']$, which have been renamed so that they have no variables in common. The subsentences $\mathcal{P}$ and $\mathcal{P}'$ are not necessarily identical, but they are unifiable, with a most-general unifier θ; thus $\mathcal{P}\theta = \mathcal{P}'\theta$. Then we may apply θ to $\mathcal{F}[\mathcal{P}]$ and $\mathcal{G}[\mathcal{P}']$, replace all occurrences of $\mathcal{P}\theta$ in $(\mathcal{G}[\mathcal{P}])\theta$ with *true*, replace all occurrences of $\mathcal{P}'\theta$ in $(\mathcal{G}[\mathcal{P}'])\theta$ with *false*, take the disjunction of the results, and (after propositional simplification) add it to our tableau as a new assertion. In other words, after applying the most-general unifier θ, we use the ground version of the rule. If exactly one of the rows has a plan entry t, the appropriate

instance $t\theta$ of that entry is inherited by the new row. If it turns out that $t\theta$ contains nonprimitive symbols, the new row is discarded to maintain the primitivity property.

In general, there may be several unifiable subsentences $\mathcal{P}_1, \mathcal{P}_2, \ldots$ in $\mathcal{F}$ and several unifiable subsentences $\mathcal{P}'_1, \mathcal{P}'_2, \ldots$ in $\mathcal{G}$. The substitution θ must then be a most-general unifier for all these sentences.

Equational Unification

Typically our knowledge of the world is represented by assertions in the tableau. It is possible, however, to build certain of the equations and equivalences of a theory into an equational-unification algorithm (Fay [79]; see also Hullot [80], Martelli and Rossi [86]), so they need not be included among the assertions. Properties of plan theory may be represented in this way, including the *linkage*, *rigidity*, and *composition* axioms.

For example, consider the sentences

$$\mathit{Clear}(s_0;z_1,\ a) \quad \text{and} \quad \mathit{Clear}\,(\mathit{put}'(w,x,\mathit{table}),\ y).$$

Regarded as expressions in pure first-order logic, these sentences are not unifiable, because the function symbols ; and put' are distinct. Suppose we apply the substitution

$$\{y \leftarrow a,\ w \leftarrow s_0,\ x \leftarrow s_0{:}u,\ z_1 \leftarrow \mathit{put}(u,\mathit{table})\}.$$

Then we obtain the sentences

$$\mathit{Clear}(s_0;\mathit{put}(u,\mathit{table}),\ a) \quad \text{and} \quad \mathit{Clear}(\mathit{put}'(s_0,\ s_0{:}u,\ \mathit{table}),\ a),$$

respectively. These are distinct sentences, but in plan theory we have

$$\begin{aligned}\mathit{Clear}(s_0;\mathit{put}(u,\mathit{table}),\ a) \ &\equiv\ \mathit{Clear}(\mathit{put}'(s_0,\ s_0{:}u,\ s_0{:}\mathit{table}),\ a)\\ &\qquad\text{(by the } \textit{plan linkage} \text{ axiom)}\\ &\equiv\ \mathit{Clear}(\mathit{put}'(s_0,\ s_0{:}u,\ \mathit{table}),\ a)\\ &\qquad\text{(by the rigidity of the designator } \mathit{table}).\end{aligned}$$

In short, by applying the substitution we have obtained sentences equivalent in plan theory. This substitution is returned by the equational-unification algorithm. We shall say that the two sentences have been unified *invoking* the two properties cited.

Most-general equational unifiers are not unique. For example, consider the substitution

$$\{y \leftarrow a,\ w \leftarrow s_0;z_2,\ x \leftarrow (s_0;z_2){:}u,\ z_1 \leftarrow z_2;\mathit{put}(u,\mathit{table})\}.$$

Applying this substitution to the same two sentences, we obtain

$$\mathit{Clear}(s_0;(z_2;\mathit{put}(u,\mathit{table})),\ a)$$

and

$$Clear(put'(s_0;z_2,\ (s_0;z_2):u,\ table),\ a),$$

respectively. But

$$\begin{aligned} Clear\,(s_0;(z_2;put(u,table)),\ a) &\equiv Clear\,((s_0;z_2);put(u,table),\ a) \\ &\quad \text{(by the } \textit{plan composition} \text{ axiom)} \\ &\equiv Clear\,(put'(s_0;z_2,\ (s_0;z_2):u,\ (s_0;z_2):table),\ a) \\ &\quad \text{(by the } \textit{plan linkage} \text{ axiom)} \\ &\equiv Clear\,(put'(s_0;z_2,\ (s_0;z_2):u,\ table),\ a) \\ &\quad \text{(by the rigidity of the designator } table). \end{aligned}$$

In general, the equational-unification algorithm may yield an infinite stream of most-general unifiers. We obtain a different resolvent for each of these substitutions.

Examples

Let us illustrate the resolution rule with an example from the *makeclear* derivation.

Example (*resolution*). Suppose our tableau contains the initial goal

assertions	goals	plan: $s_0;makeclear(a)$
	1. $\boxed{Clear(s_0;z_1,\ a)}^-$	$s_0;z_1$

and the *put-table-clear* axiom

if $On(w,x,y)$ *and* $Clear(w,x)$ *then* $\boxed{Clear\,(put'(w,x,table),\ y)}^+$		

The axiom asserts that, after a block has been put on the table, the block underneath it is clear.

As we have seen above, the two boxed subsentences are equationally unifiable in the blocks-world theory. One of the most-general unifiers is

$$\{y \leftarrow a,\ w \leftarrow s_0;z_2,\ x \leftarrow (s_0;z_2):u,\ z_1 \leftarrow z_2;put(u,table)\}.$$

The polarity of the boxed subsentences is indicated by their annotation. (The goal is negative because goals are within the scope of an implicit negation.) Let us apply the

resolution rule, taking $\mathcal{P}$ and $\mathcal{P}'$ to be the boxed subsentences and θ to be the above unifier. Recall that, according to the duality property, we can shift the assertion into the goal column by negating it. We obtain

	true *and* *not* (*if* $On(s_0;z_2, (s_0;z_2):u, a)$ *and* $Clear(s_0;z_2, (s_0;z_2):u)$ *then false*)	$s_0; z_2; put(u, table)$

which simplifies propositionally to

	2. $On(s_0;z_2, (s_0;z_2):u, a)$ *and* $Clear(s_0;z_2, (s_0;z_2):u)$	$s_0; z_2; put(u, table)$

In other words, if after execution of some plan z_2, some block u is on block a but is itself clear, we can achieve our specified condition by first executing plan z_2 and then putting block u on the table. ⌟

To present another step of the *makeclear* derivation, we give a further example of branch-free resolution.

Example (*resolution*). The boxed subsentence of the new goal,

	2. $\boxed{On(s_0;z_2, (s_0;z_2):u, a)}^-$ *and* $Clear(s_0;z_2, (s_0;z_2):u)$	$s_0;z_2; put(u, table)$

unifies equationally with the boxed subsentence of the *hat* axiom,

if not $Clear(w, y)$ *then* $\boxed{On(w, hat'(w, y), y)}^+$		

with a most-general unifier

$$\{y \leftarrow a,\ u \leftarrow hat(a),\ w \leftarrow s_0;z_2\}.$$

The equational-unification algorithm here invokes the equalities

$$(s_0;z_2):hat(a) \ = \ hat'(s_0;z_2, (s_0;z_2):a),$$

which is an instance of the *object linkage* axiom, and

$$(s_0;z_2):a \;=\; a,$$

which is a consequence of the rigidity of the input parameter a. Applying the resolution rule, we obtain (after propositional simplification)

	3. $Clear(s_0;z_2,\ (s_0;z_2):hat(a))$ *and* *not* $Clear(s_0;z_2,\ a)$	$s_0;z_2;$ $put(hat(a), table)$

In other words, if, after execution of some plan step z_2, the block a is not clear but the block $hat(a)$ is, we can achieve our specified condition by first executing plan z_2 and then putting $hat(a)$ on the table. ∎

Resolution with Conditional Formation

In applying the resolution rule between two rows, both of which have plan entries, we must generate a conditional plan entry. If we applied the ordinary resolution rule in such a case, we would be forced to introduce tests that contain the predicate symbol :: . We would have no way of executing the resulting nonprimitive plans. To avoid introducing nonprimitives into the plan entry, we employ the following resolution rule. We present the ground version of the rule as it applies to two goals:

assertions	goals	plan: $s_0;f(a)$
	$\mathcal{F}[s :: p]$	$s;e_1$
	$\mathcal{G}[s :: p]$	$s;e_2$
	$\mathcal{F}[true]$ *and* $\mathcal{G}[false]$	$s;\begin{pmatrix} if\ p \\ then\ e_1 \\ else\ e_2 \end{pmatrix}$

In other words, suppose our tableau contains two goals, both of which refer to the truth of the same propositional fluent p in a common state s. Suppose further that s is an initial segment of the plan entries for each of the two goals. Then we can introduce the same new goal as the previous branch-free version of the rule. The plan entry associated with this goal has as its initial segment the common state s of the given plan entries. Its final segment is a conditional whose test is the matching propositional fluent p and whose *then*-clause and *else*-clause are the final segments e_1 and e_2, respectively, of the given plans.

The rationale for this plan entry is as follows. We suppose that the new goal

$(\mathcal{F}[true]\ and\ \mathcal{G}[false])$ is true and show that the associated plan entry satisfies the specified condition.

We distinguish between two cases. In the case in which $s::p$ is true, because the conjunct $\mathcal{F}[true]$ is true, the given goal $\mathcal{F}[s::p]$ is also true, and hence the associated plan entry $s;e_1$ satisfies the specified condition. In this case, the conditional plan

$$s\,;(if\ p\ then\ e_1\ \ else\ e_2)$$

will also satisfy the condition because, when executed in state s, the result of the test of p will be true.

Similarly, in the case in which $s::p$ is false, the given goal $\mathcal{G}[s::p]$ is true, the associated plan entry $s;e_2$ satisfies the specified condition, and the conditional plan will also satisfy the condition. Thus, in either case the conditional plan satisfies the specified condition.

Of course, the rule applies to assertions as well as to goals. The polarity strategy may be imposed as before. We have given the ground version of the rule; in the general version, in which the rows may have variables, we first apply a most-general unifier of the subsentences $s::p$ and $s'::p'$, after renaming as necessary; we then use the ground version of the rule.

We illustrate this with an example.

Example (*resolution with conditional formation*). Suppose our tableau contains the two goals

	goals	plan: $s_0;makeclear(a)$
	$\boxed{(s_0;z_1)::clear(a)}^{-}$	$s_0;z_1$
	$not\ \boxed{(s_0;\Lambda)::clear(a)}^{+}$	$s_0;\Lambda;\ makeclear\big(hat(a)\big);$ $put\big(hat(a),table\big)$

The boxed subsentences are unifiable, with a most-general unifier $\{z_1 \leftarrow \Lambda\}$. The unified subsentences both refer to the truth of the same propositional fluent $clear(a)$ in a common state, the state $s_0;\Lambda$. The state s_0 is an initial segment for the plan entries of each of the given goals. Therefore we can apply the resolution rule to obtain (after propositional simplification)

	$true$	$s_0;\Lambda;\ \begin{pmatrix} if\ clear(a) \\ then\ \Lambda \\ else\ makeclear\big(hat(a)\big); \\ \quad put\big(hat(a),table\big) \end{pmatrix}$

Using equational unification, we can take advantage of properties of plan theory in applying the resolution rule. For instance, we could apply the rule in this example if our two goals were

$$Clear(s_0;z_1,\ a)$$

and

$$not\ (s_0 :: clear(a))$$

to obtain the same result. (The first is our goal 1.) This could be the final step of a *makeclear* derivation. ∎

Let us remark that we could formulate a resolution rule without the restriction that the common state be an initial segment of the plan entries. If these entries were s_1' and s_2', the plan entry for the derived goal could be taken to be

$$if\ s :: p\ \ then\ \ s_1'\ \ else\ \ s_2'.$$

The unrestricted rule does preserve the validity and correctness of the tableau. However, because the new plan entry contains the nonprimitive symbol ::, the row would have to be discarded immediately. This is why we are forced to restrict the rule.

Theory Resolution Rule

We have seen that we can build equations and equivalences of a theory into the resolution rule by using an equational-unification algorithm. Stickel [85] has introduced a further extension of the resolution rule that enables it to behave as if nonequational properties of the theory were built in, so that they may be invoked as required. We introduce a simplified version of Stickel's rule here. (The actual version is more general.)

We consider the ground case and ignore plan entries for the moment. Let us suppose that $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$ is a valid sentence of the theory. Then the *theory resolution rule, invoking the property* $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$, is as follows:

assertions	goals
	$\mathcal{F}[\mathcal{P}]$
	$\mathcal{G}[\mathcal{Q}]$
	$not\ \mathcal{H}[false,\ true]$ *and* $\mathcal{F}[true]$ *and* $\mathcal{G}[false]$

According to the polarity strategy, we may assume that some occurrence of $\mathcal{P}$ is positive in $\mathcal{H}$, that some occurrence of $\mathcal{Q}$ is negative in $\mathcal{H}$, that some occurrence of $\mathcal{P}$ in $\mathcal{F}$ is negative in the tableau, and that some occurrence of $\mathcal{Q}$ in $\mathcal{G}$ is positive in the tableau; otherwise, other cases of the rule apply.

The soundness of the rule is evident, for we can derive an equivalent goal by two applications of the ordinary resolution rule if we introduce the valid sentence $\mathcal{H}[\mathcal{P}, \mathcal{Q}]$ as an assertion. The strategic benefit of the theory resolution rule is that, if $\mathcal{H}$ is built into the rule, it is invoked only when needed, while if it is represented as an assertion, it may have numerous irrelevant consequences.

We have presented the rule as it applies to two goals. By duality, the rule can just as well be applied to two assertions or to an assertion and a goal. Also, we have presented only the ground version of the rule. To apply the general version, we first rename so that the given rows $\mathcal{F}$ and $\mathcal{G}$ and the sentence $\mathcal{H}$ will have no variables in common. We then apply a most-general unifier θ that allows the ground version of the rule to be applied to $\mathcal{F}\theta$ and $\mathcal{G}\theta$, invoking $\mathcal{H}\theta$.

Example (*theory resolution rule*). Suppose we have incorporated into the theory resolution rule the sentence

$$\begin{array}{lll} & \textit{if } \textit{Clear}(w, x) & \\ \mathcal{H}: & \textit{then if } \boxed{\textit{Red}(w, y)}^{-} & \\ & \quad \textit{then } \boxed{\textit{Red}(\textit{put}'(w, x, \textit{table}), y)}^{+}, & \end{array}$$

which is assumed to be valid in our theory. (In other words, a red object will remain red after a block has been put on the table.)

Suppose our tableau contains the rows

assertions	goals
	$\mathcal{F}:$ $\boxed{\textit{Red}(\textit{put}'(s_0, b, \textit{table}), a)}^{-}$
$\mathcal{G}:$ $\boxed{\textit{Red}(s_0, a)}^{+}$	

(In other words, we know that block a is red in state s_0, and we would like to show that a is still red after block b has been put on the table.)

The boxed subsentences of these rows unify with the correspondingly boxed subsentences of the sentence $\mathcal{H}$. The unifying substitution is

$$\theta: \{y \leftarrow a, x \leftarrow b, w \leftarrow s_0\}.$$

Therefore we may apply the theory resolution rule, invoking the above property $\mathcal{H}$. After the application of θ, the singly boxed subsentences play the role of $\mathcal{P}$, while the doubly boxed subsentences play the role of $\mathcal{Q}$. We obtain

	$not \begin{pmatrix} if\ Clear(s_0,\ b) \\ then\ if\ true \\ \quad then\ false \end{pmatrix}$ *and* *true and not false*	

which simplifies to

	$Clear(s_0,\ b)$	

(In other words, it suffices to show that block b is clear in the initial state s_0.)

The treatment of the plan entries is analogous to that for the ordinary resolution rules. If both given rows have plan entries, the rule is restricted and a conditional plan is introduced. We assume that an equational-unification algorithm is employed. Thus the rule may also invoke built-in equations and equivalences of the theory in its search for a unifying substitution. For example, $\mathcal{F}$ above could be $Red(s_0;put(b, table),\ a)$ if b and *table* are rigid designators.

The Frame Problem

One obstacle to employing a situational logic is the so-called *frame problem* (see McCarthy and Hayes [69], Kowalski [79]). In addition to specifying what relations are changed by a given action, it is also necessary to provide *frame axioms* that state explicitly what relations are left unchanged.

For instance, we have provided the *put-table-on* axiom, which states that, after a block has been put on the table, that block is indeed on the table. This may be regarded as a *primary axiom* for the action. We must also provide an associated *put-table-on* frame axiom, which states that the positions of other blocks remain unchanged by the action, namely,

$$\begin{array}{l} if\ Clear(w,\ x)\ and\ not(x = y) \\ then\ if\ On(w,\ y,\ \hat{y}) \\ \qquad then\ On(put'(w, x, table),\ y,\ \hat{y}) \end{array}$$

for all states w, blocks x and y, and objects $\hat{y}$. If we admit other relations into our theory, we must provide additional frame axioms indicating that these relations are unchanged

by the action, if indeed they are. For example, we might require a *red* frame axiom

$$\begin{array}{l} \textit{if } \mathit{Clear}(w, x) \\ \textit{then if } \mathit{Red}(w, y) \\ \qquad \textit{then } \mathit{Red}(\mathit{put}'(w, x, \mathit{table}),\ y) \end{array}$$

(if block y is red before the action, it is red afterwards) and so forth.

It is clear that, in any rich theory, a large number of axioms must be introduced to describe each action. If these axioms are expressed as assertions in our tableau, the effect on the search space could be disastrous. For instance, suppose our goal is actually $\mathit{Red}(s_0;z_1, a)$, to make block a red. We can perfectly well apply the resolution rule to this goal and the above *red* frame axiom, obtaining the suggestion that putting some block x on the table may help us make block a red, if only it is red beforehand.

Aside from the strategic intrusiveness of the frame axioms, it seems fundamentally wrong for a formalism to force us to spell out each one individually. We would like to be able to give only the primary axioms for an action, and then say that all other relations remain unchanged, unless a change is implied by these axioms. Although this approach is intuitively clear, the technical obstacles to pursuing it appear formidable. One possibility is to apply McCarthy's circumscription principle (see Lifschitz [85]) or some other form of "nonmonotonic" reasoning.

We henceforth assume that the necessary frame axioms have been constructed, perhaps by some circumscription-like mechanism. Rather than introduce these axioms as assertions in the tableau, let us allow them and their consequences to be invoked by the theory resolution rule.

Example (*frame axiom*). Suppose we have developed a goal

assertions	goals	plan
	$\boxed{On(s_0;put(a, table),\ b,\ \hat{b})}^{-}$	$s_0; put(a, table)$

and an assertion

$\boxed{\boxed{On(s_0,\ b,\ \hat{b})}}^{+}$		

In other words, we know that block b is on object $\hat{b}$ initially and would like to show that it is still on $\hat{b}$ after block a is put on the table.

We cannot unify these sentences. However, the sentences do unify equationally with the correspondingly boxed subsentences of the *put-table-on* frame axiom

$$\begin{array}{l} \textit{if } Clear(w,\, x) \textit{ and } not(x = y) \\ \textit{then if } \boxed{\boxed{On(w,\, y,\, \hat{y})}}^{-} \\ \qquad \textit{then } \boxed{On(put'(w, x, table),\, y,\, \hat{y})}^{+} . \end{array}$$

In other words, if block y is on object $\hat{y}$ in a given state, it is still on $\hat{y}$ after block x has been put on the table, provided that block x is clear in the given state and that blocks x and y are distinct.

The unifying substitution is

$$\{y \leftarrow b,\ \hat{y} \leftarrow \hat{b},\ w \leftarrow s_0,\ x \leftarrow a\}.$$

The equational-unification algorithm invokes the property

$$s_0{:}put(a, table) \;=\; put'(s_0,\, s_0{:}a,\, s_0{:}table),$$

which is an instance of the *plan linkage* axiom, and the rigidity of the designators a and *table*. Therefore we may apply the theory resolution rule, invoking the *put-table-on* frame axiom, to get

	$Clear(s_0,\, a)$ *and* $not(a = b)$	$s_0;\, put(a, table)$

In other words, it suffices to show that block a is clear initially and that blocks a and b are distinct. ⌟

By building the frame axioms and their consequences into the theory resolution rule, we have avoided the explosion of the search space that results if they are introduced into the tableau as assertions.

Resolution with Equality Matching

Sometimes in an attempt to apply the resolution rule, two subsentences will fail to unify completely but will "nearly" unify; that is, all but certain pairs of subterms will unify. In such cases, instead of abandoning the attempt altogether, it may be advantageous to go ahead and apply the rule but impose certain conditions upon the conclusion. This is the effect of applying the *resolution rule with equality matching.*

In its simplest (ground) version, the rule may be expressed as follows:

assertions	goals
	$\mathcal{F}[\mathcal{P}\langle s\rangle]$
	$\mathcal{G}[\mathcal{P}\langle t\rangle]$
	$s = t$ *and* $\mathcal{F}[\mathit{true}]$ *and* $\mathcal{G}[\mathit{false}]$

Here $\mathcal{P}\langle s\rangle$ and $\mathcal{P}\langle t\rangle$ are identical except that certain occurrences of s in $\mathcal{P}\langle s\rangle$ are replaced by t in $\mathcal{P}\langle t\rangle$. If they were completely identical, we could apply the ordinary resolution rule to obtain the new goal $(\mathcal{F}[\mathit{true}]$ *and* $\mathcal{G}[\mathit{false}])$. Instead, we obtain this goal with the additional conjunct $s = t$. The treatment of the plan entry is analogous to that for the original resolution rule.

Our rule is a nonclausal version of the E-resolution rule (Morris [69]) or the RUE-resolution rule (Digricoli and Harrison [86]). In Manna and Waldinger [86], we generalize the rule to allow more than one pair of mismatched terms and to employ reflexive binary relations other than equality, but we shall not require these extensions here.

In the nonground version, in which the sentences may contain variables, we apply a substitution to the given rows and then apply the ground version of the rule to the results. The substitution is the outcome of an abortive attempt to unify the subsentences. We shall see that, for a given pair of sentences, the substitution we employ and the pair of mismatched subterms we obtain are not necessarily unique. Some of the strategic aspects of choosing the substitution and term pair are discussed by Digricoli and Harrison [86].

Example (*resolution with equality matching*). Suppose our tableau contains the goal

	$\boxed{\mathit{Clear}(s_0;z_2,\ (s_0;z_2):\mathit{hat}(a))}^{-}$ *and* $Q(z_2)$	$s_0;\ z_2;$ $\mathit{put}(\mathit{hat}(a), \mathit{table})$

and the assertion

if $R(w,\ u)$ *then* $\boxed{\mathit{Clear}(w;\mathit{makeclear}(u),\ w:u)}^{+}$		

The two boxed subsentences are not unifiable. However, if we apply the substitution

$$\{u \leftarrow \mathit{hat}(a),\ w \leftarrow s_0;z_2\},$$

we obtain the sentences

$$\mathit{Clear}(s_0;z_2,\ (s_0;z_2):\mathit{hat}(a))$$

and

$$Clear\big((s_0;z_2);makeclear\big(hat(a)\big),\ (s_0;z_2):hat(a)\big).$$

Our mismatched terms are then

$$s_0;z_2 \quad \text{and} \quad (s_0;z_2);makeclear\big(hat(a)\big).$$

The conclusion of the rule is then (before simplification)

	$s_0;z_2 = (s_0;z_2);makeclear(hat(a))$ *and* *true and* $Q(z_2)$ *and* *not* (*if* $R(s_0;z_2,\ hat(a))$ *then false*)	$s_0;\ z_2;$ $put(hat(a),\ table)$

On the other hand, if we apply the substitution

$$\{w \leftarrow s_0,\ z_2 \leftarrow makeclear(u)\},$$

the boxed subsentences become

$$Clear\big(s_0;makeclear(u),\ (s_0;makeclear(u)):\ hat(a)\big)$$

and

$$Clear\big(s_0;makeclear(u),\ s_0:u\big).$$

Our mismatched terms are then

$$(s_0;makeclear(u)):hat(a) \quad \text{and} \quad s_0:u,$$

and the conclusion of the rule (after simplification this time) is then

	$(s_0;makeclear(u)):hat(a) = s_0:u$ *and* $Q(makeclear(u))$ *and* $R(s_0,u)$	$s_0;\ makeclear(u);$ $put\big(hat(a),table\big)$

In applying resolution with equality matching, we have altered an ordinary unification algorithm to return mismatched terms instead of failing. If we alter instead an equational-unification algorithm, we can invoke properties of our plan theory in our search for near-unifiers.

6. FORMATION OF RECURSION

The mathematical-induction rule accounts for the introduction of the basic repetitive construct — recursion — into the plan being derived. We employ well-founded induction,

i.e., induction over a well-founded relation; this is a single, very general rule that applies to many subject domains.

The Mathematical-Induction Rule

A well-founded relation $\prec_\alpha$ is one that admits no infinite decreasing sequences, i.e., sequences $x_1, x_2, x_3, \ldots$ such that

$$x_1 \succ_\alpha x_2 \quad \text{and} \quad x_2 \succ_\alpha x_3 \quad \text{and} \quad \ldots.$$

For instance the less-than relation $<$ is well-founded in the theory of nonnegative integers but not in the theory of real numbers. A well-founded relation need not be transitive.

The instance of the *well-founded induction rule* we require can be expressed as follows (the general rule is notationally more complex):

Suppose that our initial tableau is

assertions	goals	plan: $s_0;f(a)$
	$\mathcal{Q}[s_0,\ a,\ s_0;z_1]$	$s_0;z_1$

In other words, we are trying to construct a program f that, for a given input a, yields a plan $f(a) = z_1$ satisfying our condition $\mathcal{Q}[s_0,\ a,\ s_0;z_1]$. According to the well-founded induction rule, we may prove this under the induction hypothesis that, for a given state w and input u, the program f will yield a plan $f(u)$ satisfying the condition $\mathcal{Q}[w, w{:}u,\ w;f(u)]$, provided that the input $w{:}u$ is less than the original input $s_0{:}a$, that is, a, with respect to some well-founded relation. More precisely, we may add to our tableau, as a new assertion, the induction hypothesis

if $\langle w,\ w{:}u\rangle \prec_\alpha \langle s_0,\ a\rangle$ *then* $\mathcal{Q}[w,\ w{:}u,\ w;f(u)]$		

Here w and u are both variables, and $\prec_\alpha$ is actually a well-founded relation on pairs of states and objects. The relation $\prec_\alpha$ is arbitrary; its selection may be deferred until later in the proof.

Example (*well-founded induction rule*). The initial tableau in the *makeclear* derivation is

assertions	goals	plan: $s_0;makeclear(a)$
	1. $Clear(s_0;z_1,\ a)$	$s_0;z_1$

By application of the well-founded induction rule, we may add to our tableau the new assertion

if $\langle w, w{:}u\rangle \prec_\alpha \langle s_0, a\rangle$ *then* $Clear(w;makeclear(u), w{:}u)$		

In other words, we may assume inductively that the *makeclear* program will yield a plan $makeclear(u)$ that satisfies the specified condition for any input u in any state w, provided that the state-block pair $\langle w, w{:}u\rangle$ is less than the pair $\langle s_0, a\rangle$ with respect to some well-founded relation $\prec_\alpha$. ⌟

Use of the induction hypothesis in the proof may account for the introduction of a recursive call into the derived program.

Example (*formation of recursive calls*). In the *makeclear* derivation, we have obtained the goal

	3. $\boxed{Clear(s_0;z_2,\ (s_0;z_2){:}hat(a))}^-$ *and* *not* $Clear(s_0;z_2, a)$	$s_0; z_2;$ $put(hat(a), table)$

The boxed subsentence "nearly" unifies with the boxed subsentence of our induction hypothesis,

if $\langle w, w{:}u\rangle \prec_\alpha \langle s_0, a\rangle$ *then* $\boxed{Clear(w;makeclear(u), w{:}u)}^+$		

If we take the substitution to be

$$\{w \leftarrow s_0,\ z_2 \leftarrow makeclear(u)\},$$

the mismatched subterms are

$$(s_0;makeclear(u)){:}hat(a) \quad \text{and} \quad s_0{:}u.$$

We obtain the new goal

	4. $(s_0;makeclear(u)){:}hat(a) = s_0{:}u$ *and* *not* $Clear(s_0;makeclear(u), a)$ *and* $\langle s_0, s_0{:}u\rangle \prec_\alpha \langle s_0, a\rangle$	$s_0; makeclear(u);$ $put(hat(a), table)$

Other substitutions are possible, resulting in other new goals. ⌟

Note that, at this stage of the derivation, a recursive call $makeclear(u)$ has been introduced into the plan entry for the new goal 4. The condition $\langle s_0, s_0{:}u\rangle \prec_\alpha \langle s_0, a\rangle$ in the goal ensures that this recursive call will not contribute to nontermination. Any nonterminating computation involves an infinite sequence of nested recursive calls $makeclear(a)$, $makeclear(u)$, $makeclear(u')$, ... From any such sequence we can construct an infinite decreasing sequence of pairs $\langle s_0, a\rangle$, $\langle s_0, s_0{:}u\rangle$, $\langle s_0, s_0{:}u'\rangle$, ..., which is contrary to the well-foundedness of $\prec_\alpha$.

The Choice of a Well-founded Relation

Although the well-founded induction principle is the same from one theory to the next, each theory has its own well-founded relations. We actually take well-founded relations to be objects in each theory and regard the expression $x \prec_\alpha y$ as a notation for a three-place relation $\prec(\alpha, x, y)$, where α is a variable that ranges over well-founded relations.

For the blocks-world theory, one relation of particular importance is the *on* relation, which holds if one block is directly on top of another. In a given state, this relation is well-founded because we assume that towers of blocks cannot be infinite. More precisely, for each state w, we define the well-founded relation $\prec_{on_w}$ by the following *on-relation* axiom:

$$x \prec_{on_w} y \quad \equiv \quad On(w, x, y) \qquad (on\ relation)$$

(Note that for each state w we obtain a different relation $\prec_{on_w}$.) This relation has the *hat* property

$(*)$ *if* $not(w :: clear(v))$
then $w{:}hat(v) \prec_{on_w} w{:}v$.

The *on* relation $\prec_{on_w}$ applies to blocks, but the desired relation $\prec_\alpha$ in goal 4 applies to state-block pairs. However, for any well-founded relation $\prec_\beta$, there exists a corresponding well-founded *second-projection* relation $\prec_{\pi_2(\beta)}$ on pairs, defined by the following *second-projection* axiom:

$$\langle x_1, x_2\rangle \prec_{\pi_2(\beta)} \langle y_1, y_2\rangle \quad \equiv \quad x_2 \prec_\beta y_2 \qquad (second\ projection)$$

In other words, two pairs are related by the *second-projection* relation $\prec_{\pi_2(\beta)}$ if their second components are related by $\prec_\beta$. As usual we omit the sort conditions, but here β is a variable that ranges over well-founded relations. (Of course, there is a *first-projection* axiom also, but the second projection is the one we shall use.)

By applying rules of the system to the above properties, we may reduce our most recent goal

	4. $(s_0;makeclear(u)){:}hat(a) = s_0{:}u$ *and* *not* $Clear(s_0;makeclear(u), a)$ *and* $\langle s_0, s_0{:}u\rangle \prec_\alpha \langle s_0, a\rangle$	$s_0; makeclear(u);$ $put(hat(a), table)$

to obtain, by the *second-projection* axiom, taking α to be $\pi_2(\beta)$,

	5. $(s_0;makeclear(u)):hat(a) = s_0:u$ *and* $not\ Clear(s_0;makeclear(u),\ a)$ *and* $s_0:u \prec_\beta a$	$s_0;\ makeclear(u);$ $put(hat(a), table)$

and then, by the above *hat* property $(*)$, taking β to be on_{s_0},

	6. $(s_0;makeclear(hat(a))):hat(a)$ $= s_0:hat(a)$ *and* $not\ Clear(s_0;makeclear(hat(a)),\ a)$ *and* $not\ (s_0 :: clear(a))$.	$s_0;\ makeclear(hat(a));$ $put(hat(a), table)$

Through these steps, the well-founded relation $\prec_\alpha$ on state-block pairs is chosen to be $\prec_{\pi_2(on_{s_0})}$, the second projection of the *on* relation in the initial state s_0.

At this stage, we have completed the derivation of the entire *else*-branch of the *makeclear* program.

The Need for Generalization

One might believe that the derivation is nearly complete; all that remains is to dispense with the first two conjuncts of our goal 6,

(†) $\quad (s_0;makeclear(hat(a))):hat(a) = s_0:hat(a)$

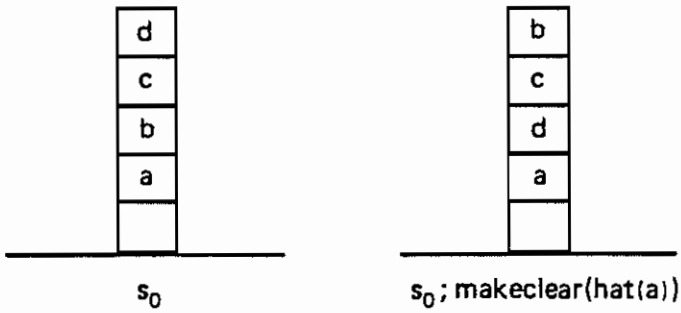
and

(‡) $\quad not\ Clear(s_0;makeclear(hat(a)),\ a).$

(The third conjunct, $not(s_0 :: clear(a))$, will then be eliminated by resolution with the initial goal 1, resulting in the introduction of the conditional construct into the final plan.) In fact, closer examination of the above two conditions indicates that they are not so straightforward.

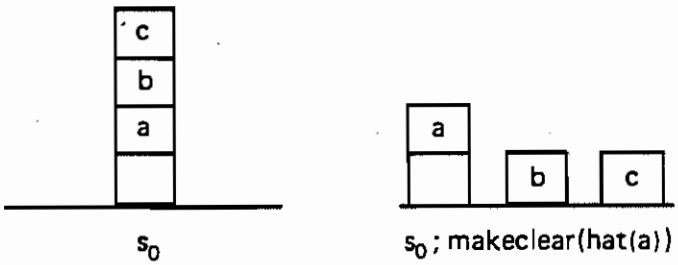
The first condition (†) requires that, after $hat(a)$ has been cleared, the value of $hat(a)$ should be the same as it was before. In other words, we must show that the *makeclear* program we are constructing will not move $hat(a)$ in the process of clearing it. In fact, the program does not move $hat(a)$, but nothing in its specification forces it to be so well-behaved. If *makeclear* were trying to be economical with table space, it might clear $hat(a)$

by putting underneath it all the blocks that were previously on top of it, as illustrated below:



Here a hypothetical *makeclear* program has cleared $hat(a)$, that is, b, by putting c and d underneath b. The subsequent value of $hat(a)$ is d, not b, which is contrary to the condition. An attempt to put $hat(a)$ on the table will then lead to unpredictable results because d is not clear.

The second condition (‡) of the goal requires that, in the process of clearing $hat(a)$, we do not inadvertently clear a. Again the program we are constructing will not do this, but there is nothing in the specification that prevents an over ambitious *makeclear* program from clearing a or any other block when it was asked only to clear $hat(a)$, as illustrated below:



Attempting to move $hat(a)$ will then lead to unpredictable results because $hat(a)$ is not a block.

The only knowledge we have about *makeclear* is that given in our induction hypothesis, which depends in turn on what is required by our specification. We have not specified what $makeclear(a)$ does to blocks underneath its input parameter a or elsewhere on the table. Thus it is actually impossible to prove the two conditions.

In proving a given theorem by induction, it is often necessary to prove a stronger, more general theorem, so as to have the benefit of a stronger induction hypothesis. Such strengthening is mentioned by Polya [57] (see also Manna and Waldinger [85b]) and is

done automatically by the system of Boyer and Moore [79]. By analogy, in constructing a program to meet a given specification, it is often necessary to impose a stronger specification, so as to have the benefit of more powerful recursive calls.

This turns out to be the case with the *makeclear* problem; the program must be constructed to meet not the given specification, but the following stronger one:

$$(\forall s_0)(\forall a)(\exists z_1)\left[\begin{array}{l} Clear(s_0;z_1,\ a)\ \ and \\ (\forall g)\left[\begin{array}{l} if\ \ Over(s_0,\ a,\ g) \\ then\ \ not\, Clear(s_0;z_1,\ g)\ \ and \\ \qquad hat'(s_0;z_1,\ g) = hat'(s_0,\ g) \end{array}\right] \end{array}\right].$$

(Here $Over(w, x, y)$ holds if block x is directly or indirectly supported by object y in state w.) In other words, in clearing block a, we do not clear any block g that is underneath a, nor do we change the hat of any such block g. In short, the relative positions of all the blocks underneath a remain unchanged. This theorem gives us an induction hypothesis strong enough to show that, in clearing $hat(a)$, or $hat\big(hat(a)\big)$, or $hat\big(hat\big(hat(a)\big)\big)$, or ..., we do not move $hat(a)$ itself. The induction hypothesis is also strong enough to enable us to prove the new condition in the theorem.

With human intuition, it may not be difficult to formulate such strengthened theorems. But the strengthening required by this problem seems to be beyond the capabilities of the Boyer-Moore system or other current theorem provers.

Although we do not know exactly how the condition could be strengthened automatically, let us suppose that it can be done. In this case, we must "edit" the derivation by adding the new condition as a conjunct in the initial goal, to obtain

	goals	plan: $s_0;makeclear(a)$
	1". $\boxed{Clear(s_0;z_1,\ a)}^{-}$ *and* $\left[\begin{array}{l} if\ \ Over(s_0,\ a,\ g'(z_1)) \\ then\ \ not\, Clear\big(s_0;z_1,\ g'(z_1)\big) \\ \qquad and\ \ hat'\big(s_0;z_1,\ g'(z_1)\big) \\ \qquad\qquad = hat'\big(s_0,\ g'(z_1)\big) \end{array}\right]$	$s_0;z_1$

Here $g'(z_1)$ is a skolem function obtained by removing the quantifier $(\forall g)$ from the given goal. In presenting the derivation, we shall drop the argument of this function and write g throughout.

We attempt to mimic the original derivation, applying the same sequence of rules to the altered goals.

For example, in the original derivation we applied the resolution rule to goal 1 and the *put-table-clear* axiom

if $On(w,x,y)$ *and* $Clear(w,x)$ *then* $\boxed{Clear(put'(w,x,table),\ y)}^{+}$		

In the altered derivation, we apply the resolution rule to the altered goal 1* and this axiom, to obtain

	2*. $On(s_0;z_2,\ (s_0;z_2):u,\ a)$ *and* $Clear(s_0;z_2,\ (s_0;z_2):u)$ *and* $\left[\begin{array}{l} \textit{if } Over(s_0,\ a,\ g) \\ \textit{then not } Clear(s_0;z_2;put(u,table),\ g) \\ \quad \textit{and } hat'(s_0;z_2;put(u,table),\ g) \\ \quad\quad = hat'(s_0,\ g) \end{array}\right]$	$s_0;\ z_2;$ $put(u,table)$

This goal is the same as goal 2 except for the addition of a third conjunct.

We proceed by mimicking the remaining steps of the original derivation. We allow ourselves to interpose additional steps as necessary. Although the induction hypothesis is now strong enough to establish the two troublesome conditions in our original derivation, additional deductive steps must be introduced to handle the new conjunct in our goal. These steps do not affect the final program.

Ultimately we derive the goal

	$not(s_0 :: clear(a))$	$s_0;makeclear(hat(a));$ $put(hat(a),\ table)$

As we have seen, we can apply the resolution rule to our initial goal 1 and this one, to obtain the final goal

	true	$s_0;\ \left(\begin{array}{l} \textit{if } clear(a) \\ \textit{then } \Lambda \\ \textit{else } makeclear(hat(a)); \\ \quad put(hat(a),\ table) \end{array}\right)$

From this goal we extract the plan

$$makeclear(a) \Leftarrow \begin{cases} \textit{if } clear(a) \\ \textit{then } \Lambda \\ \textit{else } makeclear(hat(a)); \\ \quad put(hat(a),\ table). \end{cases}$$

7. DISCUSSION

In this section we touch on some matters we have not treated in this paper.

Comparison with Human Planning

The reader may have been struck by the complexity of the reasoning required by the *makeclear* derivation, as contrasted with the apparent simplicity of the original planning problem. In fact the most difficult parts of the proof are involved not with generating the plan itself, but with proving that it meets the specified conditions successfully. We might speculate that human beings never completely prove the correctness of the plans they develop, relying instead on their ability to draw plausible inferences and to replan at any time if trouble arises. By a process of successive debugging, the HACKER system of Sussman [73] developed a plan similar to our *makeclear* plan, but it never demonstrated the plan's correctness. (It also relied on somewhat higher-level knowledge.) While imprecise inference may be necessary for planning applications, fully rigorous theorem proving seems better-suited to more conventional program synthesis.

The Problem of Strategic Control

Many people believe that a theorem-proving approach is inadequate for planning because a general-purpose theorem prover will never be able to compete with a system whose strategies are designed especially for problem solving. Although we have not yet dealt with the strategic question, we propose to overlay a general-purpose theorem prover with a special strategic component for planning. For example, the WARPLAN system (Warren [74]) might be regarded as a situational-logic theorem prover equipped with a strategy that enables it to imitate the STRIPS planning system (Fikes and Nilsson [71]). We speculate that, in the same way, a theorem prover could be induced to mimic any dedicated planning system, given the requisite strategic component.

ACKNOWLEDGMENTS

The authors would like to thank Martin Abadi, Tom Henzinger, Peter Ladkin, Vladimir Lifschitz, John McCarthy, and Jonathan Traugott for reading this manuscript, discussing its content, and suggesting improvements. Thanks also to Dag Mellgren and Mark Stickel, for assistance in applying their implementations of two equational-unification algorithms, and to Evelyn Eldridge-Diaz, for her patience in TEXing many versions of the manuscript.

REFERENCES

Boyer and Moore [79]
R. S. Boyer and J S. Moore, *A Computational Logic*, Academic Press, Orlando, Fla., 1979.

Digricoli and Harrison [86]
V. J. Digricoli and M. C. Harrison, Equality-based binary resolution, *Journal of the ACM*, Vol. 33, No. 2, April 1986, pp. 253–289.

Fay [79]
M. Fay, First-order unification in an equational theory, *Proceedings of the Fourth Workshop on Automated Deduction*, Austin, Texas, Feb. 1979, pp. 161–167.

Fikes and Nilsson [71]
R. E. Fikes and N. J. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, Vol. 2, No. 3–4, Winter 1971, pp. 189–208.

Green [69]
C. C. Green, Application of theorem proving to problem solving, *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, D.C., May 1969, pp. 219–239.

Hullot [80]
J.-M. Hullot, Canonical forms and unification, *Proceedings of the Fifth Conference on Automated Deduction*, Les Arcs, France, July 1980, pp. 318–334.

Kowalski [79]
R. Kowalski, *Logic for Problem Solving*, North-Holland, New York, N.Y., 1979.

Lifschitz [85]
V. Lifschitz, Circumscription in the blocks world, unpublished report, Stanford University, Stanford, Calif., Dec. 1985.

McCarthy [63]
J. McCarthy, Situations, actions, and causal laws, technical report, Stanford University, Stanford, Calif., 1963. Reprinted in *Semantic Information Processing* (Marvin Minsky, editor), MIT Press, Cambridge, Mass., 1968, pp. 410–417.

McCarthy and Hayes [69]
J. McCarthy and P. Hayes, Some philosophical problems from the standpoint of artificial intelligence, *Machine Intelligence 4* (B. Meltzer and D. Michie, editors), American Elsevier, New York, N.Y., 1969, pp. 463–502.

Manna and Waldinger [80]
Z. Manna and R. Waldinger, A deductive approach to program synthesis, *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, Jan. 1980, pp. 90–121.

Manna and Waldinger [85a]
Z. Manna and R. Waldinger, The origin of the binary-search paradigm, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, Calif., Aug. 1985, pp. 222–224. Also in *Science of Computer Programming* (to appear).

Manna and Waldinger [85b]
Z. Manna and R. Waldinger, *The Logical Basis for Computer Programming*, Vol. 1: *Deductive Reasoning*, Addison-Wesley, Reading, Mass., 1985.

Manna and Waldinger [86]
Z. Manna and R. Waldinger, Special relations in automated deduction, *Journal of the ACM*, Vol. 33, No. 1, Jan. 1986, pp. 1–60.

Martelli and Rossi [86]
A. Martelli and G. Rossi, An algorithm for unification in equational theories, *Proceedings of the Third Symposium on Logic Programming*, Salt Lake City, Utah, Sept. 1986.

Morris [69]
J. B. Morris, E-resolution: Extension of resolution to include the equality relation, *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, D.C., May 1969, pp. 287–294.

Murray [82]
N. V. Murray, Completely nonclausal theorem proving, *Artificial Intelligence*, Vol. 18, No. 1, 1982, pp. 67–85.

Polya [57]
G. Polya, *How to Solve It*, Doubleday and Company, Garden City, N.Y., 1957.

Stickel [85]
M. E. Stickel, Automated deduction by theory resolution, *Journal of Automated Reasoning*, Vol. 1, No. 4, 1985, pp. 333–355.

Sussman [73]
G. J. Sussman, *A Computational Model of Skill Acquisition*, Ph.D. thesis, MIT, Cambridge, Mass., 1973.

Waldinger and Lee [69]
R. J. Waldinger and R. C. T. Lee, PROW: A step toward automatic program writing, *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, D.C., May 1969, pp. 241–252.

Warren [74]
D. H. D. Warren, WARPLAN: A system for generating plans, technical report, University of Edinburgh, Edinburgh, Scotland, 1974.