SRI International

USING CAUSAL RULES IN PLANNING

Technical Note 410R

July 1987

By: David E. Wilkins
Senior Computer Scientist

Representation and Reasoning Program
Artificial Intelligence Center
Computer and Information Sciences Division

**APPROVED FOR PUBLIC RELEASE:
DISTRIBUTION UNLIMITED**

The research reported here is supported by the Air Force Office of Scientific Research, under Contract No. F49620-85-K-0001.

SRI International®

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

Abstract

Reasoning about actions necessarily involves tracking the truth of assertions about the world over time. The SIPE planning system retains the efficiency of the STRIPS assumption for this while enhancing expressive power by allowing the specification of a causal theory. Separation of knowledge about causality from knowledge about actions relieves operators of much of their representational burden and allows them to be applicable in a wide range of contexts. The implementation of causal rules is described, together with examples and evaluations of the system's expressive power and efficiency.

The research reported here is supported by Air Force Office of Scientific Research Contract F49620-85-K-0001.

1 Planning and Causality

The central problem in planning, from our point of view, is determining how a complex world description is affected by an event that occurs in the world described, so that a system can reason about the world both as it was before the event and as it will be after the event. (Some researchers call this the frame problem, though others define that term differently.) This is what makes planning inherently difficult, and what distinguishes it from similar problems that do not require this problem to be addressed. For example, many scheduling problems require constraints to be satisfied so that schedules can be correctly filled out, but do not require that the system reason about how the world changes as scheduled events occur. Scheduling problems can be very difficult, but they are simpler than problems that also require the planning problem to be addressed.

In the SIPE planning system [11,12], the solution to this problem is based on the STRIPS assumption, which is also used by many seminal planners such as STRIPS [3], NOAH [8], and NONLIN [9]. The (strict) STRIPS assumption is that no predicate will change its truth value when an event takes place unless the event explicitly lists that predicate on its add or delete lists. As stated, this strict assumption adversely affects the specification of operators (i.e., a planner's representation of actions or events), making them awkward or impossible to describe, especially as domains grow more complex.

SIPE alleviates problems caused by the strict STRIPS assumption through its use of constraints, resources, and domain rules. Domain rules, which are the subject of this paper, allow effects of an event to be deduced without being mentioned in add or delete lists. They can effectively represent a *causal theory* of the domain, similar to that advocated by Dean [2]. By allowing knowledge of cause-and-effect relations to be specified independently of the operators, both the operators and the planning process are simplified. Since conditional effects are deduced, operators are applicable over a much wider range of situations. Domain rules alter the STRIPS assumption in only a minor way, permitting the same basic algorithm

to determine the truth of a proposition. While such an extension to planning systems seems obvious, there are many problems in implementing it [10], especially if an expressive formalism is used to specify the rules. Our domain rules permit certain types of quantifiers and access to different world states, both of which would be problematic in previous STRIPS-assumption planners.

Domain rules allow expression of domain constraints within a world state, and permit access to different world states. Rules which do the former are generally called *state rules*, and rules which do the latter are generally called *causal rules*. These are discussed in more detail later. By accessing different world states, the system can react to *changes* between two states, thus permitting deduced effects concerning what happened during an action even though these effects might not be implied by the final world state. This provides a fairly powerful and expressive formalism for improving operator specification.

The ability to reason about changes between two states is crucial. Consider the problem of sliding boxes to the left or right on a table (whose depth is the same as that of the boxes). If some action slides a box from the left edge of the table to the right edge, then any intervening boxes will have been pushed off the table. Suppose the operator describing such an action states, as its only effect, that the moved box is located at the right side of the table after the action. Rules that can only access the final state would find nothing wrong with a second box being located in the middle of the table after the action — there are no inconsistencies. A causal rule in SIPE, however, could notice that the transition between the two states would involve the block in the middle and deduce that this block is pushed off the table.

One of the primary aims of our research on planning systems is to strike a balance between epistemological and heuristic adequacy. We retain as much expressive power as is practical, yet make enough restricting assumptions so that a viable, efficient implementation can still be realized. Domain rules are critical to the expressive power of the system. They maintain strict control over the deductive process while being represented in a fairly rich formalism

that includes constraints, conjunction, access to different world states, a strictly limited form of disjunction, and limited forms of both existential and universal quantifiers. As described here, this capability significantly expands upon the system's previously described deductive capability [11]. Justifications for its expressiveness and efficiency are presented. All results reported here have been fully implemented and tested.

2 A Motivating Example

A simple block-world example introduces SIPE's domain rules and shows the limitations of the strict STRIPS assumption. Consider the standard block world, with the small extension that some big blocks are large enough to support more than one small block. A strict STRIPS-assumption PUTON operator that represents moving block A from X to Y has an ADD list that includes $(On\ A\ Y)$ and $(Clear\ X)$. In our extended block world, $(Clear\ X)$ may or may not be true, depending upon whether X supports blocks other than A. The solution in strict STRIPS-assumption systems is to have two operators, one representing the move of a block that is the only block on its support, and the second representing the move of a block that is one of many on top of another block. This presupposes the ability to express and test the condition of two different blocks being on another block (which may be represented by an unbound variable at the time of the test), which not all planning systems can do.

The solution in SIPE is a single PUTON operator that lists $(On\ A\ Y)$ as its only effect, a causal rule that deduces $\neg(On\ A\ X)$, and a state rule that deduces $(Clear\ X)$ when A is the only block on X. The system is easily capable of expressing such rules, which are shown in Figure 1. These domain rules alleviate every operator in the system of the responsibility of deleting On predicates and adding Clear predicates, thus simplifying the description of many operators.

A domain rule is applied whenever an action being inserted in a plan has an effect which

CAUSAL-RULE: Not-On
ARGUMENTS: Object1,Object2,Object3;
TRIGGER: (On Object1 Object2);
PRECONDITION: (On Object1 Object3);
EFFECTS: ¬(On Object1 Object3);

STATE-RULE: Deduce-clear
ARGUMENTS: Object1,Object2, Object3 Class Existential;
TRIGGER: ¬(On Object1 Object2);
PRECONDITION: ¬(On Object3 Object2);
EFFECTS: (Clear Object2);

Figure 1: SIPE Domain Rules

matches its trigger. The trigger is matched in the current state (i.e., the world state that exists after the action is executed), while the precondition is matched in the previous state (before the action). Thus, in the Not-On causal rule, *Object*3 is bound to the support of *Block*1 before it was moved to *Object*2, and the deduced effect of $\neg On(Object1\ Object3)$ is added to the effects of the action. This deduced effect will then match the trigger of the Deduce-Clear state rule whose precondition is matched in the current state. Because *Object*4 is constrained to be in the existential class (scoping rules interpret the second precondition predicate as $\neg\exists Object2.On(Object2\ Block1)$ — see next section), the precondition will match (and $Clear(Block1)$ will be deduced) if there is currently no object on *Block*1. [1]

As more complex domains are represented, it becomes crucially important to use causal theories so that operators do not have to encode such knowledge. The simplifies the operators

[1] Note that Deduce-Clear could also have been written as a causal rule with the same trigger as the Not-On rule, but it is more clearly expressed here as a state rule. In one of our earlier papers [11], we used this causal-rule approach to Deduce-Clear. However, causal rules had not been implemented and the system had only deductive operators which were a restricted form of the current state rules. The deductive operator in our previous paper accessed previous states through the current state by relying on the order of deductions and on certain properties of the matching algorithm. Not surprisingly, this was rather obscure, and the semantics of the current domain rules are much clearer.

and allows them to be applicable in a wide range of contexts since the causal theory will deduce conditional effects. This can reduce exponentially the number of operators required by the strict STRIPS assumption which needs a distinct operator for every different context in which the operator may be used (where one context is *different* from another if it requires different predicates to appear in either the add or delete lists). If an action can affect N predicates, a strict STRIPS assumption might require 2^N operators to represent it, while only one operator and N domain rules are needed in SIPE. Furthermore, these domain rules will in general be used by other operators. Thus a second action that might change the same N predicates would require one operator and no new domain rules in SIPE, while it would require another 2^N operators using the strict STRIPS assumption.

3 Domain Rules in SIPE

Recently, SIPE's deductive capability has been completely redesigned to enable the implementation of more powerful and useful causal theories. By using domain rules as described below, rather than providing a full-fledged logic for deduction, we maintain strict control over the deductive process, thus preventing a combinatorial explosion. At the same time, domain rules provide a fairly rich formalism for deducing (possibly conditional) effects of an action because they can include constraints, conjunction, access to different world states, a strictly limited form of disjunction, and limited forms of both existential and universal quantifiers. The limitations on disjunction and quantifiers are described in Sections 3.1 and 3.2. The use of constraints in a partial matching algorithm, crucial for expressive power and efficiency, is briefly discussed in Section 3.3. Section 3.4 presents problems with this design and their solutions.

SIPE domain rules have triggers, preconditions, conditions, and effects. The trigger controls rule application because it must match an effect of the node of current concern before the rule can be applied. (A node is the planner's representation of a planned action

in the given situation and permits the causal rules to trigger on these recursively. This process continues until no effects are deduced that were not already true, thus computing the deductive closure of the causal rules. This process is then repeated for the state rules, initially firing them on all node-specified effects and all effects deduced from causal rules. In this way the deductive closure of all domain rules is computed while maintaining control of the deductive process and preventing deductive loops.

The causal rules are applied first because, using Georgeff's proposed view of causality [4], one might view a causal rule as representing another action that is caused by the original one (and, for our purposes, occurs simultaneously). Considering the disappearance of a moved block from its original position as a separate event caused by the move is reasonable in a world where some objects may leave copies of themselves behind when they move. Once the causal rules are used to determine all the simultaneously occurring events caused by the current event, the state rules then compute the domain constraints that must be true of all these events.

Note that there is no difference between causal rules and state rules other than their order of applicability – they have identical expressive power. Each domain rule must be declared as either a state rule or a causal rule when it is defined. In all domains yet implemented in SIPE, domain rules never have a precondition (only a condition) while causal rules always have a precondition (and perhaps also a condition). Thus the causal rules are reacting to changes between states, while domain rules are enforcing constraints within a state. These limitations are not enforced as definitions of causal and state rules in order to provide more power and flexibility.

3.1 Disjunction and Existential Quantifiers

The details of the system's matching algorithm and representation determine restrictions on quantifiers and disjunction. Disjunction and existential quantifiers are not permitted in the

and includes the action, its effects, and, implicitly, the world state that holds after the action.) If the precondition and condition of a rule hold, the effects of the rule can be added as effects of the node. The trigger, precondition, and condition are matched exactly like any other formula during the planning process, taking advantage of the system's efficient reasoning about what is currently true. Triggers and conditions are always matched in the (current) world state of this node, while preconditions are matched in the previous state.

More formally, given a causal rule with a trigger τ, a precondition ϕ, a condition χ, and effects ψ, the following formula (which borrows notation from the situation calculus) describes its meaning for all events and states:

$$\forall e, s \;.\; occurs(e, s) \;\wedge\; holds(\tau, result(e, s)) \;\wedge\; \neg holds(\tau, s) \wedge\; holds(\phi, s) \wedge \\ holds(\chi, result(e, s)) \;\supset\; holds(\psi, result(e, s))$$

Note that domain rules do not trigger on every formula in $result(e\ s)$, only on ones that were not true before e occurred. This is efficient in SIPE because possible triggers are simply the effects of a node. Assuming an event occurs in state s1 which results in state s2, the Not-On causal rule would be expressed more formally as follows (abbreviating object as obj):

$$\forall Obj1, Obj2, Obj3 \;.\; holds(On(Obj1, Obj2), s2) \;\wedge\; \neg holds(On(Obj1, Obj2), s1) \\ \wedge\; holds(On(Obj1, Obj3), s1) \;\supset\; holds(\neg On(Obj1, Obj3), s2)$$

All deductions that can be made are performed when a new node is inserted in the plan. The deduced effects are recorded and the system can then proceed as if all the effects had been listed in the operator. Because deductions are not attempted at other points in the planning process, SIPE can always use the STRIPS-assumption algorithm of regressing through actions, although the comparison of predicates at each point is more complicated because of constraints and quantifiers.

Domain rules are divided into causal rules and state rules, with the former being applied first. Initially, all causal rules whose trigger matches a node-specified effect are applied, thereby producing an additional set of [deduced] effects for that node. After all such rules have been applied, the system determines which newly deduced effects were not already true

world model or in the effects of actions, for obvious reasons. The form *(OR pred1 pred2 . . . predN)* is permitted wherever a predicate is expected in a trigger or precondition, but the semantics are that the system will use only the first predicate that matches. Thus, if pred1 has some possible matches, they are the only matches that will ever be considered. This has proven useful in practice without introducing additional complexity.

SIPE makes a closed world assumption, assuming that if an unnegated predicate is not given in the world model, then its negation is true. (Thus the user does not have to axiomatize the enormous number of formulas that are not true in his domain.) This leads to restrictions on the use of quantifiers. Formulas of the form $\exists x.P(x)$ and $\forall x.\neg P(x)$ are relatively easy to compute since they involve searching for occurrences of unnegated predicates. Existential variables can appear only in preconditions (see Figure 1 for an example), and are used to represent both these forms. In negated predicates, the existential quantifier is scoped (by definition) within the negation, yielding a formula equivalent to the previous universally quantified formula. The scope of each existential variable is local to the predicate in which it occurs.

Determining the truth of a negated predicate, $\neg P$, with an existential variable is somewhat complex. For each predicate occurrence P that might (with proper variable instantiations) disprove it, the system must find a matching negated occurrence that follows P in the plan. This may involve temporarily assuming that two variables codesignate. In this way, the Deduce-Clear state rule in Figure 1 can deduce that a block originally supporting N blocks will be clear only after N actions remove blocks.

3.2 Universal Quantifiers

Universal variables are permitted in domain rules. As expected, a universal variable in the effects of a node means that the effects are true for all objects that match the variable (taking into account the constraints on the variable). However, universal variables in preconditions

are treated like any other variable. If X is a variable in P, a precondition, it means that only instantiations of X for which P is true will be considered hereafter (i.e., $x \mid P(x)$). (In particular, X being universal does not mean $\forall x.P(x)$). During matching of such a precondition, the system will generate constraints enabling the variable to match all and only those objects for which the precondition is true. This ability to form subsets is useful and powerful, and exploits the system's representation and algorithms for the purpose of representing a number of assertions compactly as a single predicate with a universal variable. A primary advantage of this approach is the gain in efficiency that is achieved by matching a predicate only once (for all future matches that regress back to this node), instead of having to match it once for each possible match of the universal variable. [2]

The mobile robot domain provides an example of the use of universals. In one solution, we keep track of all objects that are next to the robot. The causal rule shown in Figure 2 eliminates the *Nextto* predicates that are no longer true after the robot has moved. When the precondition predicate (*Nextto Robot*1 *Object*2) is matched, it will constrain the variable *Object*2 to match only those objects that were next to the robot before it moved. The condition predicate will further constrain *Object*2 to not be any object that is adjacent to the new location of the robot. Thus the constraints on the universal *Object*2 ensure it will match exactly those objects that the robot was next to before it moved but was not next to afterwards. This effectively picks out the subset of objects that interests us and allows their efficient representation.

Note that the constraints on *Object*2 may refer to variables instead of actual objects. For

[2] Previously, SIPE did not permit a variable marked universal to appear in a precondition. Allowing them in preconditions entailed a major change in the matching algorithm used to determine the truth of a predicate (because constraints are now posted on universals). Before this extension, a universal variable always matched exactly, so the matcher had to look no further for possible matches. With constraints, universals are now only potential matches. Therefore, the matching process must therefore continue to collect other potential matches and let the system generate additional constraints upon the possible ways the predicate could be made true.

CAUSAL-RULE: No-longer-nextto
ARGUMENTS: Robot1, Location1,
    Object2 CLASS UNIVERSAL;
TRIGGER: (At Robot1 Location1);
PRECONDITION: (Nextto Robot1 Object2);
CONDITION: ¬(Adjacent-Loc Object2 Location1);
EFFECTS: ¬(Nextto Robot1 Object2);

Figure 2: Causal Rule for Updating Nextto

example, the constraints may specify that *Object2* must match one of a set of N planning variables. Some of these variables may not yet be instantiated, but eventually they will be. Perhaps they will be instantiated to N different objects, or they may all be instantiated to the same object. There is no matching problem because the constraints on each of the N planning variables specify all relevant information (e.g., which variables and objects these N planning variables are or aren't identical to).

The further planning of actions occurring either earlier or later will not affect the validity of the universal variable in the Nextto predicate that occurs in the deduced effects of the node. This is true by virtue of the place where the predicate is recorded in the plan, because the matching algorithm regresses back through a plan searching for effects that can match a given formula. Suppose N1 is one of the N planning variables that will match the universal variable. If a later action specifies (*Nextto Flakey N1*) as an effect, this latter predicate will always be matched to a formula before the predicate with the universal variable. Thus, the appropriate relationship between N1 and the corresponding variable in the formula being matched will already be determined and will not be affected by any subsequent attempt to match the same formula with the predicate containing the universal variable. Further planning of an action before the one containing the Nextto predicate as an effect will be done at the next lower [hierarchical] planning level. In this case, SIPE recomputes all deductions that follow (at the next planning level). Thus, a new universal variable will appear with constraints that have been properly calculated for the new situation.

3.3 Constraints and Partial Matching

Much of the power of domain rules comes from SIPE's ability to reason about constraints. PRED constraints are posted by the system to ensure that preconditions of domain rules will be true. They specify a disjunctive set where each element of the set is a list of possible instantiations for variables (which will assure that a predicate is made true). For example, to ensure that $P(x\ y\ z)$ holds, a PRED constraint will be posted on each of the three variables. In general, the constraint contains a disjunctive set with elements $(x_1\ y_1\ z_1)$ through $(x_n\ y_n\ z_n)$. To satisfy the constraint, there must be some i such that the three variables $(x\ y\ z)$ can codesignate with $(x_i\ y_i\ z_i)$, respectively.

System efficiency depends upon the recently modified algorithm for matching two variables with constraints, which is summarized briefly here. When matching two variables, PRED constraints force the system to determine if a set of instantiations will be compatible after these two variables are matched. In the above example, if the system is trying to match the variable x with the object A, it must find an $(x_i\ y_i\ z_i)$ in the PRED constraint on x such that x_i and A can codesignate. It should then check that the other members of this set (y_i and z_i in this case) can codesignate with their corresponding variables. Checking every member of the set in this way would propagate matches throughout the system as variables are matched recursively. In the worst case, every constraint in the system might be checked each time a variable is matched. This is computationally unacceptable. Furthermore, loops may be created because the system may recursively match the two original variables again.

To avoid the above problems, the matching algorithm does a complete check of constraints only at the top level of recursive calls to the matcher. At lower levels of the recursion, the system still checks all constraints, but this time recurses on PRED constraints only for variables that are instantiated (effectively assuming the uninstantiated variables will be acceptable). This approach avoids both the problem of needing to check for loops and of the matching becoming prohibitively expensive. This matching algorithm has proven satisfactory

in practice. It matches exactly the variables one would intuitively expect from looking at two levels of the constraint network. The idea of checking the instantiated variables at the lower levels of recursion is critical for achieving our level of performance, because it is frequently the case that most variables in a problem are instantiated. Although no invalid matches have been detected in practice, the consequences of an invalid match would generally be the unnecessary searching of a portion of the search space that did not contain a solution (as the problem will be discovered at the end of each planning level when the global constraint satisfaction problem is solved). This would not be a catastrophe, although earlier versions of the matching algorithm that did permit invalid matches caused problems in the replanning algorithm. SIPE's algorithm is similar in spirit to that used by Allen for maintaining temporal relations [1]. Allen's algorithm guarantees only consistency between three-node subnetworks of the overall constraint network. The algorithm therefore permits inconsistent labelings of a network, but he still considers it the best practical solution, since it avoids an exponential search while producing useful results.

3.4 Problems

The reader may have noticed two problems in the foregoing scheme. The first problem occurs while repeatedly applying the domain rules to produce their deductive closure: a later rule might deduce a predicate that negates a predicate deduced by a previous rule. Which of these conflicting deductions should the system allow to stand as an effect? SIPE's default is to accept the first deduction and ignore the second. However, there is a situation, first observed in the mobile robot domain, in which it is desirable to deduce effects that would be conflicting if they were to be matched directly against each other but become a valid, nonconflicting representation of effects when they are recorded in the order they are deduced. This situation involves the deduction of a predicate with a universal variable that negates particular nonuniversal instances of the predicate that have already been deduced.

It is permitted by the system because it appears to be of general utility (see below).

The matching algorithm matches formulas against effects in the order they are listed. Thus, one can initially deduce (*On Object*1 *Object*2), where the objects are not universals (they may be variables that are not yet instantiated but eventually will be), and later deduce ¬(*On Object*1 *Object*3), where *Object*3 is universal. These deductions will be recorded in the given order, which effectively encodes the fact that *Object*1 is on *Object*2 only. More precisely, any formula of the form (*On Object*1 *X*) will match positively if X can be constrained to be the same as *Object*2, and negatively in all other cases.

The second drawback of this design is more serious. SIPE may have to instantiate variables to match the precondition of a domain rule. However, it may not be desirable to do this since the instantiation so forced may prevent a solution to the problem from being found. For example, suppose Deduce-Clear was written to deduce that *Block*2 is clear instead of *Object*2. This is an acceptable description of the domain since the table is the only non-block object and it is always clear. (As written, Deduce-Clear simply never matches with *Object*2 being the table.) But now suppose the problem is to achieve *On*(*Redblock*1 *Blueblock*1) where the two blocks are left as variables. After planning an action to move *Redblock*1, the system might apply the block2-Deduce-Clear and, to match its precondition, constrain *Redblock*1 to be a block that was on another block. (As explained below, SIPE would *not* add such a constraint in this situation.) This would prevent *Redblock*1 from matching a block that was on the table, and later planning may discover that only a block from the table can solve the problem.

As this example shows, care must be taken in writing causal rules so as to avoid such undesirable behavior. There are a few tools that help the user get around this problem. He can choose whether or not to permit (in all cases) the forcing of instantiations by the application of causal rules. The default in SIPE, which has been used in all its applications, is a useful compromise. The system will constrain variables in an attempt to match a causal

rule, but only when the two variables are initially constrained to be of the same class. If a causal rule requires further specification of a variable's class, it will fail: it is assumed the user did not intend the deduction in this case. Thus the user can control the forcing of instantiations by the classes used in causal rules. For example, the block2-Deduce-Clear fails in SIPE whenever it must constrain a variable representing an object to be a block because the classes are different. (It may match later in the planning after the object variable is further specified to be a block.) Deduce-Clear, as shown in figure 1, is appropriate for both blocks and tables and does not force variables towards either. However, the user might want to force things to be moved off of blocks in order to clear them (a good heuristic for many block-world problems). This can be done in SIPE by inserting "class blocks" after *Object2* in Deduce-Clear. *Object2* would then be constrained to be in both the block and object classes, and the system would constrain variables of either class in an attempt to apply Deduce-Clear. This shows the flexibility our scheme provides the user, but care is required.

4 Using Causal Theories in a Mobile Robot Domain

Recently a simple indoor mobile-robot world was encoded in the SIPE planning system. A brief look at one of the nodes in a plan from this domain will show the importance of causal theories in the use of the planner. The domain consists of five rooms connected by a hallway in the Artificial Intelligence Center at SRI International, the robot itself (Flakey), and various objects. The rooms were divided into 35 symbolic locations, and the initial world is described by 222 predicate instances. The description of possible actions in SIPE includes 25 action-describing operators and 25 domain rules (5 of which are causal rules). The operators use four levels of abstraction in the planning process. The planner produces primitive plans that provide actual commands for controlling the robot's motors. These commands are executable by SRI's mobile robot, although the SIPE plans currently assume perfect execution of the commands in order to work. Thus, SIPE is used to control the robot

simulator until lower level routines are developed for reporting (or correcting) errors during execution of commands.

It would have been impractical to represent this domain without a causal theory. A typical problem in this domain was to deliver an object to an agent (in this case, a bagel to Leslie), which requires planning a path to retrieve the bagel and then a path to deliver it to Leslie. The prominence of the causal theory in the planning process is indicated by the fact that 73% of the CPU time spent on this problem was spent on deducing effects. The node in Figure 3 is from a plan for this problem. The only effects listed in operators for this action of going through a door are that Flakey is now at Loc11 and in Leslie's office. All of the following effects were deduced from domain rules (described in the order they appear): the rye bagel (which Flakey is holding) is now also at Loc11, Flakey is now next-to some subset of objects, Flakey is not next-to any other object, Flakey is not at any other location, Flakey now occupies an adjacent location to some subset of locations (the members of the subset are specified by constraints on the universal variable), Flakey does not occupy an adjacent location to any other location, the bagel is not at any other location, the bagel occupies an adjacent location to some subset of locations, the bagel is no longer adjacent to Loc11, the bagel is in Leslie's office, and neither Flakey nor the bagel is in the hallway anymore.

5 Expressive Power and Efficiency

By expressing knowledge of causality independently of the operators, operators become transparent, modular, and simpler, and the planning process becomes more efficient. SIPE is capable of expressing more powerful conditions than other STRIPS-assumption planners because of its constraints, quantified variables, and ability to access different world states. The domain rules themselves have considerable expressive power and can differentiate contexts that can not be differentiated in NOAH-like systems.

PROCESS: P6948
ACTION: Go-Thru-Door;
EFFECTS: (At Flakey Loc11),(Inroom Flakey Leslie-Office);
PURPOSE: (Deliver1 Leslie Rye-Bagel Flakey);
DEDUCE: (At Rye-Bagel Loc11),
(Nextto Flakey Object2-N6797) N6797 universal,
¬(Nextto Flakey Object2-N6802) N6802 universal,
¬(At Flakey Location2-N6782) N6782 universal,
(Adjacent-Loc Flakey Location3-N6788) N6788 universal,
¬(Adjacent-Loc Flakey Location4-N6790) N6790 universal,
¬(At Rye-Bagel Location2-N6816) N6816 universal,
(Adjacent-Loc Rye-Bagel Location3-N6822) N6822 universal,
¬(Adjacent-Loc Rye-Bagel Loc11),
(Inroom Rye-Bagel Leslie-Office),
¬(Inroom Flakey Jhall), ¬(Inroom Rye-Bagel Jhall);

Figure 3: Node in Robot Plan

While this approach is more expressive than using the strict STRIPS assumption, it still has many limitations. It depends upon the closed world assumption as well as certain and perfect knowledge of the world (although the latter can be relaxed somewhat). Events are considered to be discrete and instantaneous, and time is represented in a fairly simple manner. Despite its simple temporal representation, SIPE can specify that two actions be nonoverlapping without ordering them (by having them in parallel with a resource conflict between them). This cannot be represented in some temporal reasoning systems [1]. Nevertheless, many domains cannot be represented. Approaches such as general frame axioms [7] and circumscription [6] provide greater expressiveness for reasoning about the effects of actions on the world, but they suffer from lack of an efficient implementation, and possibly other problems such as unintended models [5].

The efficiency of reasoning with causal rules depends on the basic representations used in SIPE and the partial matching algorithm described above. Much of the efficiency in the representation is based on the closed-world assumption. Domain rules use the same operator syntax as do other SIPE operators and use the same algorithm for determining the truth of

a formula. The system assumes that no change has occurred in the world unless that change is listed as an effect of an action or can be deduced from the effects listed.

This approach has many computational advantages over a complete first-order logic for deduction (which would be used to provide more expressive power). With an expressive logic there is generally a need to specify axioms to deduce that all things not mentioned have stayed the same (unless the STRIPS assumption or something similar is employed). There can also be problems with the introduction of unintended models, or the computation of all possible effects an action might have. SIPE avoids these problems through its simplifying assumptions. Of course, these limiting assumptions restrict the domains that can be represented.

6 Conclusion

SIPE provides a powerful formalism for representing domains by allowing the specification of a causal theory. Separation of knowledge about causality from knowledge about actions relieves operators of much of their representational burden and allows them to be applicable in a wide range of contexts. The power is provided by rules that may contain constraints, the accessing of different world states, partial matching, disjunction, existential quantification, and universal quantification. The formalism remains efficient by appropriately (and sometimes severely) limiting the interpretations of the above concepts. The system has been tested on several domains, including a mobile robot domain, and has performed well.

Acknowledgments

Michael Georgeff greatly influenced the ideas expressed in this paper.

References

[1] Allen, J., "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, November 1983, v. 26 n. 11, pp. 832-843.

[2] Dean, T., "Temporal Imagery: An Approach to Reasoning about Time for Planning and Problem Solving", Technical Report 433, Yale University Computer Science Department, 1985.

[3] Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence 2*, 1971, pp. 189-208.

[4] Georgeff, M. P., "Actions, Processes, and Causality", *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, Timberline Lodge, Timberline, Oregon, 1987.

[5] Hanks, S., and McDermott, D., "Default Reasoning, Nonmonotonic Logics, and the Frame Problem", *Proceedings AAAI-86*, Philadelphia, Pennsylvania, 1986, pp. 328-333.

[6] McCarthy, J., "Circumscription: A Nonmonotonic Inference Rule", *Artificial Intelligence 13*, 1980, pp. 27-40.

[7] McDermott, D., "A Temporal Logic for Reasoning about Processes and Plans", *Cognitive Science 6*, pp. 101-155.

[8] Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977.

[9] Tate, A., "Generating Project Networks", *Proceedings IJCAI-77*, Cambridge, Massachusetts, 1977, pp. 888-893.

[10] Waldinger, R., "Achieving Several Goals Simultaneously", in *Readings in Artificial Intelligence*, Nilsson and Webber, eds., Tioga Publishing, Palo Alto, California, 1981, pp. 250–271.

[11] Wilkins, D., "Domain-independent Planning: Representation and Plan Generation", *Artificial Intelligence 22*, April 1984, pp. 269-301.

[12] Wilkins, D., "Recovering from Execution Errors in SIPE", *Computation Intelligence 1*, February 1985, pp. 33-45.