Representation in a Domain-Independent Planner

David E. Wilkins
Artificial Intelligence Center
SRI International
Menlo Park, California

ABSTRACT

The representation used in a domain-independent planning program that supports both automatic and interactive generation of hierarchical, partially ordered plans is described. An improved formalism for representing domains and actions is presented. The formalism makes extensive use of *constraints,* offers efficient methods for representing properties of objects that do not change over time, allows specification of plan rationale, allows specification of *resources* for efficiently detecting and remedying harmful parallel interactions, and provides the ability to express deductive rules for deducing the effects of actions.

1. Overview

The automation of planning in a computer program involves representing the world, representing actions and their effects on the world, reasoning about the effects of sequences of such actions, reasoning about the interaction of actions that are taking place concurrently, and controlling the search so that plans can be found with reasonable efficiency. Planners designed to work efficiently in a single problem domain, though desirable, often depend on the structure of that domain to such an extent that the underlying ideas cannot be readily used in other domains. This paper discusses domain-independent planners that are of particular interest, since they yield planning techniques that are applicable in many domains and provide a general planning capability. Such a commonsense planning capability is likely to require different techniques from those used by an expert planning in his particular domain of expertise, but it is nonetheless essential for people in their daily lives and for intelligent programs. Of course, a general planner should provide representations and methods for including domain specific knowledge and heuristics. This paper describes an implemented planning program that expands the core of domain-independent planning techniques as it builds on and extends such previous domain-independent planning systems as Sacerdoti's NOAH [4], Tate's NONLIN [7], Sridharan's PLANXIO [5], Vere's DEVISER [8], and SRI's STRIPS [1].

*The research reported here is supported by Air Force Office of Scientific Research Contract F4962O-79-C-0188.

We have designed and implemented (in INTERLISP) a system, SIPE (System for Interactive Planning and Execution Monitoring), that supports domain-independent planning. The program has produced correct parallel plans for problems in four different domains (the blocks world, cooking, aircraft operations, and a simple robotics assembly task). The system allows for hierarchical planning and parallel actions. Development of the basic planning system has led to several extensions of previous systems. These include the development of a perspicuous formalism for encoding descriptions of actions, the use of constraints to partially describe objects, the creation of mechanisms that permit concurrent exploration of alternative plans, the incorporation of heuristics for reasoning about resources, mechanisms that make it possible to perform simple deductions, and advanced abilities to reason about the interaction among parallel actions. SIPE can automatically generate plans, but, unlike its predecessors, SIPE is designed to also allow interaction with human users throughout the planning and plan execution processes, if this is desired. The user is able to watch and, when he wishes, guide and/or control the planning process.

In SIPE, a plan is a set of partially ordered goals and actions, which is composed by the system from operators (the system's description of actions that it may perform). By simply applying operators, plans that do not achieve the desired goal may sometimes be generated, so the system has *critics* that find potential problems and attempt to avert them. In particular, most of the reasoning about interactions between parallel actions is done by the critics. The plans are represented in procedural nets [4], primarily for graceful interaction between man and machine. Invariant properties of objects in the domain are represented in a tree-structured sort hierarchy, which allows inheritance of properties and the posting of constraints on the values of attributes of these objects. The relationships that change over time - and therefore all goals - are represented in a version of first-order predicate calculus that is typed and interacts with the knowledge in the sort hierarchy. Operators are represented in an easily understood formalism, developed by us,

in which the ability to post constraints on variables is a primary feature. Each of these parts of the system will be described in more detail later.

It should be noted here that, like most domain-independent planning systems (DEVISER being an exception), ours assumes discrete time, discrete states, and discrete operators. These assumptions are acceptable in many real-world domains, even though they are restrictive and prevent many real-world phenomena from being adequately represented. For example, sophisticated reasoning about time and modelling of dynamic processes are not possible within our present framework. Few artificial intelligence programs have addressed these problems (McDermott's recent work being a notable exception [2]).

The planning-representation problem involves representing the domain, goals, and operators. Operators are the system's representation of actions that may be performed in the domain or, in the hierarchical case, abstractions of actions that can be performed in the domain. An operator includes a description of how each action changes the state of the world. In a logical formalism such as Rosensebein's adaptation of dynamic logic to planning [3], the same representational formalism may be used for representing the domain, goals, and operators; however, in many planners more concerned with efficiency, including SIPE, there is a different representation for each. The goal is to have a rich enough representation so that many interesting domains ran be represented (an advantage of logical formalisms), but this must be measured against the ability of the system to deal with its representations efficiently during the planning process.

This paper describes SIPE's solution to the problem of representing the domain, goals, and operators. (Other aspects of the system are described in [11].) A central concern in designing a representation for a planning system is how to represent the effects an action has on the state of the world. This means that the frame problem [I] must be solved in an efficient manner. Since we intend that many domains will be encoded in the planning system, it is also necessary that the solution to the frame problem not be too cumbersome. For example, one does not want to have to write a large number of frame axioms for each new action that is defined.

2. Representation of Domain and Goals

The system provides for representation of domain objects and their invariant properties by nodes linked in a hierarchy. This permits SIPE to incorporate the advantages of frame-based systems (primarily efficiency), while retaining the power of the predicate calculus for representing properties that do vary. Invariant properties do not change as actions planned by the system are performed (e.g., the size of a frying pan does not change when you cook something in it). Each node can have attributes associated with it, and can inherit properties from other nodes in the hierarchy. The values of attributes may be numbers, pointers to other nodes, key words the system recognizes, or any arbitrary string (which can only be used by checking if it is equal to another such string). The attributes are an integral part of the system, since planning variables arc also nodes in the hierarchy and contain constraints on the values of attributes of possible instantiations. Constraints are an important part of the system and are discussed in considerable detail later. There are different node types for representing variables, objects, and classes, but these will not be discussed in detail here, since they are similar to those occurring in many representation formalisms - for example, semantic networks and UNITS[6].

A restricted form of first-order predicate calculus is used to represent properties of domain objects and the relationships among them that may change with the performance of actions; it is also therefore used to describe goals as well as the preconditions and effects of operators. Quantifiers are allowed whenever they can be handled efficiently. Universal quantifiers are always permitted in effects, and over negated predicates in preconditions. Existential quantifiers can occur in the preconditions of operators, but not in the effects. Disjunction is not allowed. These restrictions result from using "add lists" to solve the frame problem. (Why this is so is described in the next section). By representing the invariant properties of the domain separately, SIPE reduces the number of formulas in the system and makes deductions more efficient. There is currently no provision for creating or destroying objects as actions are executed, although in some domains this would be useful (e.g., after an omelet has been made, do the original three eggs still exist as objects?).

3. Representation of Operators

Operators representing actions the system may perform contain information about the objects that participate in the actions (represented as resources and arguments of the actions), what the actions are attempting to achieve (their goals), the effects of the actions when they are performed, and the conditions necessary before the actions can be performed (their preconditions). Before SIPE's representation is described in detail, some basic assumptions made by SIPE about the effects of actions need to be presented.

Determining the state of the world after actions have been performed (e.g., the planner must ascertain whether the intended goals have been achieved), involves solving the frame problem. Here we make what Waldinger [9] has called the STRIPS assumption, which is that all relations mentioned in the world model remain unchanged unless an action in the plan specifies that some relation has changed. In STRIPS, an action specifies that a relation has changed by mentioning it on an "addlist" or "deletelist". Alternatively, relations that change might be deduced from general frame axioms as long as the deduction is tightly controlled.

Muking this assumption imposes requirements on the formalism used for representing the domain, since it must support the STRIPS assumption. While the STRIPS assumption may be very limiting in the representation of rich domains, such as automatic programming, there are many domains of interest for which it causes no problems. For example, the fairly simple environments in which robot arms often operate appear adequately represent able in a system embodying the STRIPS assumption. SI PIC currently makes the closed-world assumption: any negated predicate is true unless the unnegated form of the predicate is explicitly given in the model or in the effects of an action that has been performed. This is not critical; the system could be changed to assume that a predicate's truth-value is unknown unless an explicit mention of the predicate is found in either negated or unnegated form. (Although in large domains, there may be an enormous number of predicates that are not true.) Deduction in SIPE does not violate the closed-world assumption; it is used only to deduce effects of an action when the action is added to a plan (thus sparing the operator that represents the action from having to specify these effects).

Many features combine to make SIPE's operator description language an improvement over operator descriptions in previous systems. These features will be presented by discussing the sample operator given in Figure 1, with subsections devoted to the more important features. The SIPE system has produced correct parallel plans for problems in four different domains, one of which is the blocks world (described in [4]) for which many domain-independent planning systems (e.g., NONLIN and NOAH) have presented solutions. To facilitate comparison with these systems, a PUTON operator for the blocks world in the SIPE formalism is shown in Figure 1.

The operator's effects, preconditions, and purpose are all encoded as first-order predicates on variables and objects in the domain. (In this case, BLOCK1 and OBJECTI are variables.) Negated predicates that occur in the effects of an operator essentially remove from the model a fact that was true before, but is no longer true.

```
OPERATOR: PUTON
ARGUMENTS: BLOCK 1, OBJECTI IS NOT BLOCK1;
PURPOSE: (ON BLOCK1 OBJECTI);
PLOT:
    PARALLEL
        BRANCH 1:
            GOALS: (CLEARTOP OBJECTI);
            ARGUMENTS: OBJECTI;
        BRANCH 2:
            GOALS: (CLEARTOP BLOCK 1);
            ARGUMENTS: BLOCKI;
    END PARALLEL

    PROCESS
    ACTION: PUTON.PRIMITIVE;
    ARGUMENTS: OBJECTI;
    RESOURCES: BLOCKI;
    EFFECTS: (ON BLOCKI OBJECTI);
END
```

Figure 1
A PUTON Operator in SIPE

Operators contain a plot that specifies how the action is to be performed in terms of actions and goals at either the current level or some lower level of the hierarchy. Like plans, plots are represented as procedural networks. When used by the planning system, the plot can be viewed as instructions for expanding a node in the procedural network to the next level. The plot of an operator can be described either in terms of *goals* to be achieved (i.e., a predicate to make true), or in terms of *processes* to be invoked (i.e., an action to be performed). (NOAH represented a process as a goal with only a single choice of action.) Encoding a step as a process implies that only the action it defines can be taken at that point, while encoding a step as a goal implies that any action can be taken that will achieve the goal. Another less explicit difference between encoding a step as a goal or as a process is whether the emphasis is on the situation to be achieved or the actual action being performed.

During planning, an operator is used to expand an already existing GOAL or PROCESS node in the procedural network to produce additional procedural network structure at the next level. For example, the PUTON operator might be applied to a GOAL node in a plan whose goal predicate is (ON A B). Operators may specify preconditions that must obtain in the world state before the operator can be applied. (The operator in Figure 1 has no precondition.) Operators contain lists of resources and arguments to be matched with the resources and arguments of the node being expanded. In our example, A and B in the GOAL node are matched with BLOCKI and OBJECTI in the PUTON operator when the operator is used to expand

the node. The plot of the operator is used as a template for generating two COAL nodes and one PROCESS node in the plan.

Operators in SIPE provide for posting of constraints on variables, specification of resources, and the use of deduction to determine the effects of actions. Each feature is described below in some detail. SIPE also provides the ability to explicitly represent the rationale behind each action (this is mentioned briefly below), and to apply the plots of operators iteratively to sets (this is not discussed in this paper).

3.1 SIFE's Constraint Language

SIPE's ability to construct partial descriptions of unspecified objects is one of its most important advances over previous domain-independent planning systems. This ability is important both for domain represention (e.g., objects with varying degrees of abstractness can be represented in the same formalism) and for finding solutions efficiently (since decisions can be delayed until partial descriptions provide more information). Almost no previous domain-independent planning systems have used this approach (e.g., NOAH cannot partially describe objects) so the constraints in SIPE will be documented in some detail.

Planning variables that do not yet have an instantiation ran be partially described by setting constraints on the possible values an instantiation might take. This allows instantiation of the variable to be delayed until it is forced or until as much information as possible has been accumulated, thus preventing incorrect choices from being made. Constraints may place restrictions on the properties of an object (e.g., requiring certain attribute values for it in the sort hierarchy), and also require that certain relationships exist between an object and other objects (e.g., predicates that must be satisfied in a certain world state). SIPE provides a general language for expressing these constraints on variable bindings so they can be encoded as part of t the operator. During planning, the system also generates constraints that are basted on interactions within a plan, propagates them to variables in related parts of the network, and finds variable bindings that satisfy all constraints.

The allowable constraints in SIPE on a variable V are listed below:

• CLASS. This constrains V to be in a specific class in the sort hierarchy. In SIPE's operator description language there is implicit typing based on the variable name; therefore, in the PUTON operator in Figure 1, the variable created for BLOCK1 has a CLASS constraint that requires the instantiation for the variable to be a member of the class BLOCKS. Similarly, the OBJECT1 variable has a CLASS constraint for class OBJECTS.

• NOT-CLASS. V must be instantiated so that it is not a member of a given class.

• PR ED. V must be instantiated so that a given predicate (in which V is an argument of the predicate), is true. This results in an explicit number of choices for V's instantiation, since all true facts are known (by the closed-world assumption).

• NOT-PRED. V must be instantiated so that a given predicate (in which V is an argument of the predicate), is not true.

• SAME. Y must be instantiated to the same object to which some other given variable is instantiated.

• NOT-SAME. Y must not be instantiated to the same object to which some other given variable is instantiated. In the PUTON operator in Figure 1, the phrase "IS NOT BLOCK 1" results in a NOT-SAME constraint being posted on both BLOCK 1 and OB.JECTI that requires they not be instantiated to the same thing. Thus, if SIPE is looking for a place to put block A, it will not choose A as the place to put it.

• INS TAN. V must be instantiated to a given object. This could be represented by using SAME applied to objects as well as variables (or using PRED with an EQ predicate), but instantiation is a basic function of the system and warrants its own constraint for a slight gain in efficiency.

• NOT-IN STAN. V must not be instantiated to a given object.

• OPTIONAI,-SAME. This is similar to SAME, but merely specifics a preference and is not binding. For example, one would prefer to conserve resources by making two variables be the same object, but, if this is not possible, then different objects are acceptable.

• OPTIONAL-NOT-SAME. This is similar to NOT-SAME, but is not binding. If SIPE notices that a conflict will occur between two parallel actions if two variables are instantiated to the same object, then it will post an OPTIONAL-NOT-SAME constraint on both variables. If it is possible to instantiate them differently, a conflict is avoided. If it is not, they may be made the same but the system will have to resolve the ensuing conflict (perhaps by not doing things in parallel).

• Any attribute name. This requires a specific value for a specific attribute of an object. For example, the PUTON operator could have specified "BLOCK 1 WITH COLOR RED". This would create a constraint on BLOCK 1 requiring the COLOR attribute (in the sort hierarchy) of any possible instantiation to have the value RED. For attributes with numerical values, "greater than" and "less than" can also be used. In planning

an airline schedule, for example, the operator used for cross-country flights might contain the following variable declaration: "PLANEI WITH RANGE GREATER THAN 3000"

Constraints add considerably to the complexity of the planner because they interact with all parts of the system. For example, to determine if a goal predicate is true, SIPE must verify whether it matches predicates that arc effects earlier in the plan. This may require matching variables that are arguments to the two predicates, which in turn involves determining whether the constraints on the two variables are compatible. In a similar way, constraints also interact with the deductive capability of the system (to be described later). Constraints also affect critics, since determining if two concurrent actions interart may depend on whether their constraints are compatible. SIPE must also solve a general constraint satisfaction problem with reasonable efficiency, though how to control the amount of processing spent on constraint satisfaction is an open and important question. SIPE's method of propagating and checking constraints is described in more detail in [11].

The use of constraints is a major advance over previous domain-independent, planning systems. NOAH, for example, would have to represent every property of an object as a predicate and then, to get variables properly instantiated, would have each such predicate as either a precondition of an operator or a goal in the plan. In SIPE an operator might declare a variable as "CARGOPLANEI WITH RANGE 3000" and the plan using this variable can assume it has the proper type of aircraft. In NOAH, goals similar to (CARGOPLANE X) and (RANGE X 3000) would have to be included in the operator and achieved as part of the plan. This makes both the operators and plans much longer and harder to use and understand. In addition to syntactic sugar, constraints in SIPE improve efficiency and expressibility. The OPTIONAL-SAME and OPTIONAL-NOT-SAME constraints used in resource reasoning cannot be expressed as goals or preconditions in a system like NOAH. The constraint satisfaction algorithm used in SIPE takes advantage of the fact that invariant properties of objects are stored directly in the sort hierarchy. The lookup of such properties in SIPE is much more efficient than the process of looking through the plan to determine which predicates are currently true, as would have to be done in systems like NOAH and NONLIN.

Some domain-dependent systems make use of constraints. Stefik's system [6], one of the few existing planning systems with the ability to construct partial descriptions of an object without identifying the object, operates in the domain of molecular genetics. Our system extends Stefik's approach in three ways. (1) We provide an explicit, general set of constraints that can be used in many domains. Stefik does not present a list of allowable constraints in his system; moreover, some of them that are mentioned seem specifically related to the genetics domain. (2) Constraints on variables can be evaluated before the variables are fully instantiated. For example, a set can be created that is constrainable to be only bolts, then to be longer than one inch and shorter than two inches, then to have hex heads. This set can be used in planning before its members are identified in the domain. (3) Partial descriptions can vary with the context, thus permitting simultaneous consideration of alternative plans involving the same unidentified objects (see [11]).

3.2 Resources

Parallelism is considered beneficial since optimal plans in many domains require it. (Two segments of a plan are in parallel if the partial ordering of the plan does not specify that one segment must be done before the other.) The approach used in SIPE, therefore, is to keep as much parallelism as possible and then to detect and respond to interactions between parallel branches of a plan. SIPE provides the ability to reason about *resources,* which is a powerful mechanism both for detecting parallel interactions and remedying them.

The formalism for representing operators in SIPE includes a means of specifying that some of the variables associated with an action or goal will actually serve as resources for that action or goal (e.g., BLOCK 1 is declared as a resource in the PUTON.PRIMITIVE action of the PUTON operator in Figure 1). Resources are to be employed during a particular action and then released, just as a frying pan is used while vegetables are being sauteed in it. Reasoning about resources is a common phenomenon. It is a useful way of representing many domains, a natural way for humans to think about problems, and, consequently, an important aid to interaction with the system.

SIPE has specialized knowledge for handling resources; declaration of a resource associated with an action is a way of saying that one precondition of the action is that the resource be available. Mechanisms in the planning system, as they allocate and deallocate resources, automatically check for resource conflicts and ensure that these availability preconditions will be satisfied.

Resources enable SIPE's operators and plans to be more succinct and easier to understand than similar operators and plans in domain-independent parallel planning systems, such as NOAH and NONLIN. In the latter systems, resource availability would hav? to be correctly axiomatized, checked, and updated in the preconditions and effects of the operators. It is not clear that it would be possible to do this so that the critics would recognize only the intended conflicts. If it were indeed possible,

the resource reasoning in S1PE would be much more efficient than such an axiomatization in the other systems. For example, in NOAH (he resolve-conflicts c/itic would eventually have to notice that posted "available-resource" effects are in conflict. This could be done only after the entire plan had been expanded and the critics applied. Even then, conflicts between uninstantiated variables might not be detected, since only in an attempt to instantiate them would an actual conflict arise. In SIPE it is known which resources an operator needs before it is applied, so conflicts can be detected even before the plan is expanded. This can result in choosing operators that do not produce conflicts, thereby pruning the search space. SIPE avoids not only the immediate incorrect operator expansion, but also both the entire expansion to the next level and the application of the critics after that. The savings can be considerable in domains that use resources heavily. SIPE can also detect conflicts between uninstantiated variables; if a plan requires two arms as resources and only one arm exists in the world, SIPE can detect this conflict even though the two arm variables have not been instantiated.

This section only summarizes the resource reasoning abilities provided by SIPE's representation. Details of how resources are actually implemented are given in [II] and [10]. These papers include examples of problems solved with and without reasoning about resources, as well as a general description of the parallel interaction problem and SIPE's solution to it.

3.3 Plan Rationale

SIPE provides more flexibility in specifying the rationale behind a plan than many domain-independent planners. (The rationale for an action in a plan is "why" the action is in the plan.) This is needed for determining how long a condition must be maintained, what changes in the world cause problems in the plau, and what the relationship is among different levels in the hierarchy. SIPE constructs links, both between the levels of a plan and within a level, that help express the rationale behind t he actions. The system has reasonable defaults for constructing these links, but also provides the flexibility for operators to specify how these links should be constructed (thus permitting a larger class of phenomena to be represented). This is described in more detail in [11].

3.4 Deductive Operators

In addition to operators describing actions, SIPE allows specification of deductive operators that deduce facts from the current world state. This provides a deductive capability that is useful, but nevertheless keeps deduction under control by severely restricting the deductions that can be made. As more complex domains are represented, it becomes increasingly important to deduce the effects of actions from axioms about the world, rather than explicitly representing these effects in operators. For example, the PUTON operator in Figure 1 lists only (ON BLOCK 1 OBJECTI) as an effect. It does not mention which objects are or are not now CLEARTOP, since that is deduced by deductive operators. Because deductive operators in SIPE may include both existential and universal quantifiers, they provide a rich formalism for deducing (possibly conditional) effects of an action. Effects that are deduced in SIPE are considered to be side effects. (Operators can also specify effects as either main effects or side effects.) Knowing which effects are merely side effects is important in handling parallel interactions.

Figure 2 shows one of the deductive operators in the SIPE blocks world for deducing CLEARTOP relationships. Deductive operators are written in the same formalism as other operators in SIPE, thus permitting the system to control deduction with the same mechanisms it uses to control the application of operators. This also allows constraints to be used and, as this example shows, they play a major role in SIPE's deductive capability.

All deductions that can be made are performed at the time an operator is expanded. The deduced effects are recorded in the procedural net, and the system can proceed just as if all the effects had been listed in the operator. Deductions are not attempted at other points in the planning process. Deductive operators have triggers for controlling their application. The DCLEAR operator in Figure 2 is applied when OBJECTI is placed on OBJECT2. Deductive operators have no instructions for expanding a node to a greater level of detail. Instead, if the precondition of a deductive operator holds, its effects can be added to the world model (in the same context in which the precondition matched) without changing the existing plan. This may "achieve" some goal in the plan (by deducing that it has already been achieved), thereby making it unnecessary to plan actions to achieve it. In Figure 2, matching the precondition will bind BLOCK3 to the block that OBJECTI was on before it moved to OBJECT2. Since OBJECT4 is constrained to be in the EXISTENTIAL class (see below) and is constrained to not be OBJECTI, the precondition will match (and CLEARTOP of BLOCK3 deduced) only if OBJECTI is the only object on BLOCK3 (just before moving OBJECTI to OBJECT2).

The method used for specifying variables as existentially quantified (i.e., constraining them to be in the EXISTENTIAL class) does not provide scoping information. Since only certain types of quantifiers are permitted for efficiency reasons, SIPE in-

```
DEDUCT1VE.OPERATOR: DCLEAR
ARGUMENTS: OBJECTI, 0BJECT2,
     BLOCK3 IS NOT OBJECT2,
     OBJECTI CLASS EXISTENTIAL IS NOT OBJECTI;
TRIGGER: (ON OBJECTI OBJECT2);
PRECONDITION: (ON OBJECTI BLOCK3),
     (NUT (ON OBJECT4 BLOCK3));
EFFECTS: (CLEAR BLOCK3);
```

Figure 2
A Deductive Operator in SIPE

terprets preconditions according to defaults that are somewhat non-standard. The scope of each EXISTENTIAL variable appearing as an argument in a predicate is local to that predicate. Each predicate effectively gets a different existential variable. In addition, negated predicates are interpreted as having the quantifier within the scope of the negation. Thus, the variable is effectively universally quantified for negated predicates. As an example, with *x* declared EXISTENTIAL, the precondition P*(1)A-Q[x)* is interpreted as 3 *x.P(x)* A -3 *x.Q(x)* (or equivalently, 3 1.*P{x)* A Vz. -» *Q(x)*). These restrictions make use of SIPE's representation (e.g., the fact that negated predicates are treated differently) to permit handling quantifiers efficiently.

Besides simplifying operators, deductive operators are important in many domains for their ability to represent conditional effects. In NOAH's blocks world, only one block may be on top of another; consequently, whenever a block is moved, the operator for the move action can be written to state explicitly the effect that the block underneath will be clear. In the more general case in which one large block might have many smaller blocks on top of it, there may or may not be another block on the block underneath, so the effects of the action must be conditional upon this. Since systems like NOAH and NONLIN must mention effects explicitly (universally or existentially quantified variables are not allowed in the description of effects), they cannot represent this more general case with a single move operator. These systems would need two move operators - one for the one-block-on-top case, another for the many-blocks-on-top case. Furthermore, the preconditions for separation of the cases would add an undesirable complication to the representation of the operators.

As the above example shows, SIPE's deductive operators allow certain quantifiers and are powerful enough to handle this case. Since SIPE can deduce all the clearing and unclearing effects that occur in the blocks world, the operators themselves do not need to represent them. As domains grow to include many operators, this becomes very convenient. Deductive operators provide a way to distinguish side effects, which can be important. By using deduction, more complicated blocks worlds can be represented more elegantly in SIPE than in previous domain-independent planners.

4. Performance of SIPE

SIPE has been tested in four different domains: the blocks world, cooking, aircraft operations, and a simple robotics assembly task. These domains do not have large branching factors or search spaces so that the automatic search can find solutions. The cooking domain was encoded to demonstrate resource reasoning. SIPE operators naturally represented requirements for frying pans and burners during the cooking of a dish. Problems such as cooking four dishes with three pans on two burners were handled efficiently by the resource reasoning mechanisms in SIPE. Handling a problem means producing plans for cooking as many dishes as possible in parallel, with enough serialization to get the task accomplished with the available resources. Such plans consisted of dozens of nodes in our simple cooking world.

The standard blocks world was encoded in SIPE, with some enrichments (e.g., more than one block could be on top of another). Use of deductive operators made the PUTON operator more readable. Resource reasoning enabled SIPE to quickly find and correct parallel interaction problems. A number of other problems involving properties of the blocks and quantifiers were also handled elegantly (making use of the constraints in SIPE). For example, the problem of getting *some* red block on top of *some* blue block is easily represented and solved. (SIPE will choose a red block and a blue block that are already clear, if such exist.)

5. Conclusion

SIPE's operator description language was designed to be perspicuous (to enable graceful interaction) while being more powerful than those found in previous domain-independent planners. Constraints, resources, and deductive operators all contribute to the power of the representation. Deductive operators allow quantified variables and can therefore be used to make fairly sophisticated deductions, thus eliminating the need to express effects in operators when they can be deduced. They are also useful in distinguishing main effects from side effects.

One of the most important features of SIPE is its ability to constrain the possible values of variables. It is well known that this enables more efficient planning, since choices can be delayed until information has been accumulated. Other advantages of constraints, however, are also critical. A key consideration is that constraints allow convenient expression of a much wider range of problems. Constraint satisfaction finds variable in-

stand ations efficiently by taking advantage of the fact that invariant properties of objects are encoded in the sort hierarchy. Constraints also help prevent harmful parallel interactions.

SIPE provides the ability to reason about resources which is important both for representation of domains and the handling of parallel interactions. Combined with the system's ability to post constraints, resource reasoning helps the system avoid many harmful interactions, helps it recognize sooner those interactions that do occur, and helps the system solve some of these interactions more quickly. SIPE'S handling of interactions is also improved by its ability to differentiate side effects and to correctly determine the rationale behind actions.

ACKNOWLEDGMENTS

Many people influenced the ideas expressed in this paper. Special thanks go to Ann Robinson who helped design and implement SIPE, and to Nils Nilsson, Mike Georgeff, and Stan Roscnschein for many enlightening discussions.

REFERENCES

1. Fikes, R., Hart, P., and Nilsson, N., "Learning and Executing (Generalized Robot Plans", *Readings in Artificial Intelligence,* Nilsson and Webber, ed., Tioga Publishing, Palo Alto, California, 1981, pp. 231 249.

2. McDermott, D., "A Temporal logic for Reasoning About Processes and Plans" , Cognitive Science, forthcoming.

3. Roseuscbein, S., "Plan Synthesis: A Logical Perspective", *Proceedings IJCAI 81,* Vancouver, British Columbia, 1981, pp. 331-337.

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

5. Sridharan, N., and Bresina, J., "Plan Formation in Large, Realistic Domains", *Proceedings CSCSI Conference,* Saskatoon, Saskatchewan, 1982, pp. 12-18.

6. Stefik, M., "Planning with Constraints", Report STAN-CS-80-784, Computer Science Dept., Stanford University, 1980.

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

8. Vere, S., "Planning in Time: Windows and Durations for Activities and Goals", Jet Propulsion Lab, Pasadena, California, November 1981.

9. Waldinger, R., "Achieving Several Goals Simultaneously", *Readings in Artificial Intelligence,* Nilsson and Webber, ed., Tioga Publishing, Palo Alto, California, 1981, pp. 250-271.

10. Wilkins, D. E., "Parallelism in Planning and Problem Solving: Reasoning About Resources", *Proceedings CSCSI Conference,* Saskatoon, Saskatchewan, 1982, pp. 1-7.

11. Wilkins, D. E., "Domain-independent Planning: Representation and Plan Generation", Technical Note 266R, SRI International Artificial Intelligence Center, Menlo Park, California, May 1983.