Operator Construction: A Compact, Maintainable Representation for Real World Planners

Daniel M. Gaines
dmgaines@cs.uiuc.edu

Caroline C. Hayes
hayes@cs.uiuc.edu

Computer Science Department
University of Illinois at Urbana-Champaign and
Beckman Institute of Advanced Science and Technology

Abstract

AI planners typically use a set of generalized operator descriptions as a representation of the possible domain actions. These operators are then selected and instantiated to accomplish particular task goals. However, in complex domains it may be difficult to enumerate all of the operators, many of which may contain redundant information. Adapting the planner to related domains or to changes in the domain properties may require extensive modifications to update and maintain the operator descriptions. A more maintainable representation is required for complex domains. This paper proposes a technique called *Operator Construction* which can be applied to physical planning domains. In this approach, instead of representing the possible domain actions as a set of generalized operators, knowledge about domain objects that generate actions is used to generate operators. The result is better suited for complex domains because Operator Construction offers the advantage of ease of maintenance, and a more direct link between domain properties and possible actions. This technique has been used in a complex manufacturing planning domain.

Introduction

In this paper we present an alternative method for generating operator instantiations and representing the set of possible actions in physical domains, called *operator construction*. The advantages of this method are that it makes a direct connection between the properties of the domain objects that produce actions and the resulting possible operations. The result is a more maintainable and modifiable representation, which is particularly critical when constructing planners for complex, practical domains.

Many planners represent actions that can be taken in a particular domain as a set of generalized operators. These operators list a set of preconditions that must be true before the operator can be applied, and a set of effects that become true after its application. The planner's task is to find operators to accomplish a given set of goals, this is usually done by matching goals against the operators' listed effects. Specific operator instantiations are created by substituting values from the goal for the generalized operator's parameters.

However, there are many difficulties with this type of action representation. Problems of particular concern to this work are that:

1. The set of operators included in the planner may not actually cover the full range of actions that it is possible to take in the domain. In very large domains is very difficult for knowledge engineers to think of all the possible operators and to produce an adequate list of preconditions and effects.
2. Much redundant information may be repeated in many operators.
3. It may be very difficult to update and maintain such an operator description in complex domains. The same change may need to be made to many operators, and it is difficult to translate the effect of a change in the domain capabilities into generalized operator representations.

We address these problems with an alternate method for representing the information in generalized operators and generating operator instantiations called Operator Construction. Operator Construction is most applicable to complex, physical domains, such as building, transport or manufacturing. It may also be applicable to non-physical domains, but we have not yet investigated this issue. The main idea behind Operator Construction is that instead of using a set of generalized operator descriptions and matching goals to effects, an operator construction engine is given a set of descriptions of the action producing objects in the domain, and a specific plan goal. The operator generating engine combines the action producing objects to produce a set of operator instantiations that can satisfy the goal. These operator instantiations can then be passed on to a planner (Hayes 1996) which must select which operator instantiation is the most appropriate in the context of the plan as a whole, look for interactions between operators, and sequence them in to a plan (but not necessarily in that order).

The Operator Construction technique provides a means for representing and reasoning about knowledge that produces actions. The reasoning behind this is that actions that can be taken in a physical[1] domain result directly from the properties of the domain objects used to perform actions. For example, one's ability to create a set of decorative door moldings for one's house may depend on having the correct type of router bits to cut curved and beveled edges, and proper guides that allow the router to be driven in a straight line along the edge of the wood. Without the proper tools, the task may not be impossible, but it may become impracticable. For convenience, we will divide the domain objects into those that make actions possible (such as the router, router bits and guides) to be the *action generating objects* and *acted-upon objects* (such as the wood). Some objects may fall into both categories.

Work that is most closely related to the Operator Construction technique includes MOLGEN (Stefik 1981b, Stefik 1981a) and SIPE (Wilkins 1984). Figure 1 shows the distinction between operator construction and these related systems. Figure 1 (a) depicts the operator selection and instantiation process used by SIPE and MOLGEN. Like these techniques, Operator Construction uses hierarchies of domain objects and operators are produced by a process of successive refinement.

The difference lies in the fact that MOLGEN and SIPE represent operators explicitly in the form of an operator refinement hierarchy which lists all the possible operators at varying levels of detail, while Operator Construction is an operator hierarchy generator. It never explicitly represents this whole operator hierarchy. Instead, Operator Construction generates only those portions of the operator hierarchy that are applicable to the current goal. The property that makes operator construction easy to maintain is that by changing the object descriptions fed to the operator generator, the operator hierarchy produced also changes.

For example, if the action producing objects in a domain change (for example if your router bits wear out or you purchase a jig saw) then the possible actions and the results that can be produced also change. These changes can be directly represented in the Operator Construction method by changing the description of the action generating objects. The operator generator then does the work of translating those changes into specific operator instantiations. This is much simpler and more direct than attempting to infer the changes necessitated in a set of generalized operator descriptions when these physical equipment changes. We have implemented this method successfully in a complex manufacturing domain (Gaines, Castaño, & Hayes 1996), and have found it to be an effective and maintainable representation.

Contrast to SIPE and MOLGEN

Throughout this paper, the traveling domain is used as an example to illustrate our ideas. This section describes the traveling domain and then contrasts the Operator Construction technique to two planners, MOLGEN and SIPE, through the use of this example.

The Traveling Domain. In the traveling domain, the planner must decide the best means of traveling between two locations. A plan is represented by a route and a means of traveling between these two points. There are usually many ways in which one might travel given the host of transport vehicles and routes available. Many options and alternative should be considered to ensure a quality plan based on the factors of importance, such as cost, total travel time, safety, etc.

Contrast to Related Work. There are many properties of the planning methods used by MOLGEN (Stefik 1981b) and SIPE (Wilkins 1984, Wilkins 1988) that are closely related to the Operator Construction method. All use hierarchies of domain objects (similar to object and sort hierarchies) and operators are produced by a process of successive refinement. The difference lies in the fact that MOLGEN an SIPE explicitly represent operators in an operator refinement hierarchy which lists all the possible operators at varying levels of detail, while Operator Construction never explicitly represents this whole operator hierarchy. Instead, Operator Construction generates only those portions of the operator hierarchy that are applicable to the current goal. Additionally, the end result of the Operator Construction process is a set of applicable operator instantiations for a given goal, not just a single operator instantiation. It is the job of the planner to select which is the most appropriate operator instantiation.

SIPE uses a hierarchy of operator descriptions to describe the actions available in a domain at various levels of abstraction. A separate structure, called the *sort hierarchy*, is used to describe properties of objects which can be used as arguments to the operators. Figure 2 contains the hierarchy of operators that SIPE would use to represent the operators in this domain [2].

[1]This approach may also be applicable in abstract, non-physical domains, like algebraic manipulations, but we have not yet investigated these areas.

[2]This figure was adapted by the authors from SIPE datafiles. The authors have made efforts to accurately represent the structure. We hope we have not made

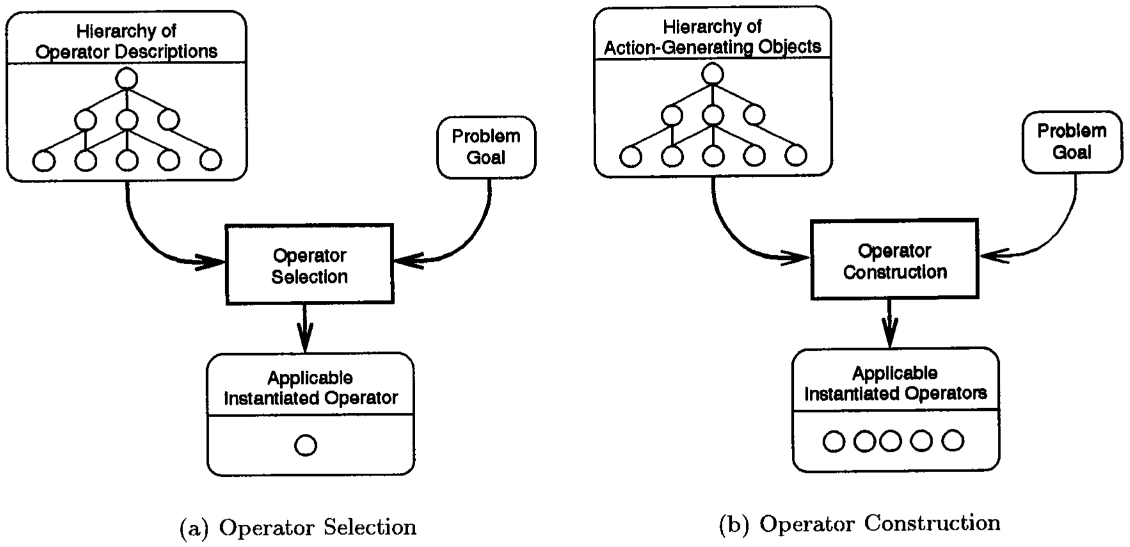
Hierarchy of Operator Descriptions
Problem Goal
Operator Selection
Applicable Instantiated Operator
(a) Operator Selection
Hierarchy of Action-Generating Objects
Problem Goal
Operator Construction
Applicable Instantiated Operators
(b) Operator Construction

Figure 1: The operator construction vs. operator selection process.

At the highest level is the abstract *Transport* operator, which has the effect of moving an object from one location to another if a route exists. More detailed operators are provided at the next level which describe traveling by land, sea or air. Each of these has its own special requirements in addition to the requirements for the abstract *Transport* operator. For example, both *Transport* and *Land* need a route, but the *Land* operator requires that route to be a roadway. Again, specialized versions of these operators are provided at the lowest level which describe specific means of each form of transportation. For example, when traveling by land, one has the choice of driving a car, taking a bus or riding on a train. Each of these operators has additional requirements on their applicability, e.g. it is not possible to take a train between two locations unless there are train tracks.

Operators are selected based on their ability to achieve a goal. Once selected, their parameters are instantiated with the particular objects in the goal. For example, given the goal of traveling from home to the grocery store, the *Drive-Car* operator could be chosen as its effect is to move from one location to another. The start and destination parameters would be replaced by home and grocery store, respectively.

MOLGEN uses a similar approach to represent a hierarchy of laboratory operations and objects. In MOLGEN, *Lab-Operator* is used to describe the most abstract operator. This is refined by a set of MARS (Merge, Amplify, React, Sort) operators which describe a class of laboratory operations for conducting experiments in molecular genetics. Just as SIPE has a sort hierarchy to describe the domain objects, MOLGEN has a laboratory object hierarchy to describe the objects that manipulated by the operators.

Of particular importance to the ease of maintenance issue is the fact that each operator in the hierarchy of Figure 2 copies information from its ancestors. In more complex domains, maintaining this structure can become quite difficult. In the next section, an alternative approach for representing information is provided. This approach provides a maintainable, compact structure which can be used to dynamically generate operator descriptions based on the objects' ability to accomplish goals.

The Operator Construction Technique

Operator Construction is different from SIPE and MOLGEN in that it does not use an operator hierarchy, only an object hierarchy. Furthermore, the object hierarchy does not represent all objects in the domain, just the objects which will produce actions, like cutting tools, clamping devices, etc. The object hierarchy is then used to generate the section of the operator hierarchy that can satisfy the current goal.

Operator Construction results in a set of instantiated operators, each describing an alternative means of achieving the goal. The important parts of Operator Construction are the hierarchy of action-generating ob-

any misinterpretations. The interested reader is referred to `http://www.ai.sri.com/~wilkins/` for the complete description.

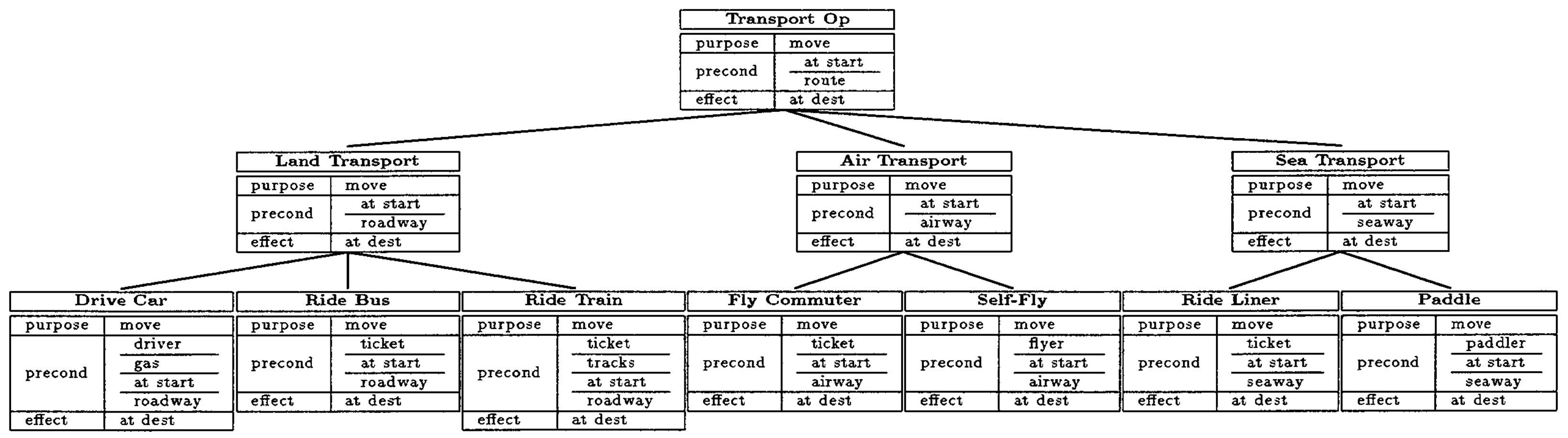

Figure 2: The SIPE operator refinement hierarchy for the traveling domain.

jects and Operator Construction engine which generates sets of instantiated operators to achieve given goals.

Hierarchy of Action-Generating Objects

In the Operator Construction technique, an *object refinement hierarchy* contains action-generating objects and their important properties such as the requirements needed to drive a car: roadways and the ability to drive. Nodes in a hierarchy describe what an object or class of objects can do, and requirements which must be true in order to allow the object to be used. The hierarchical structure reflects abstractions in the domain, allowing related objects to share common properties.

As an example, Figure 3 shows how the hierarchy for action producing objects in the traveling domain is represented. Objects at the top of the hierarchy represent abstractions of the objects below it. Common properties and requirements of all objects below are stored in the abstract objects. The object hierarchy includes classes of objects: *Transport Vehicles*, *Land Vehicles*, etc, as well as specific objects such as *Cars* and *Commuter Planes*. In the hierarchy, each transport vehicle has specific properties which make it distinct from the other vehicles. Some modes of transportation such as, such as *Drive Car*, *Self Fly* and *Paddle*, require special properties for their use. Others, such as *Ride Train*, place further restrictions on the type of route which they can take. All of the objects have different speeds at which they can travel. For simplicity's sake, many details have been left out of the example. One might wish to consider the cost of using each vehicle, the preferences of the traveler with respect to the quality of vehicle and other considerations. All of this information, pertaining to how to travel, can be added into the hierarchy. As properties in the domain change, for example when cars are able to drive themselves, the hierarchies can be easily updated.

The next section describes how operator hierarchies are constructed from the object refinement hierarchies. Following the description, an illustration of these ideas will be provided with an example from the traveling domain.

Operator Construction Engine

This section describes how Operator Construction uses the object refinement hierarchy to generate operator descriptions. Figure 4 shows the system diagram for Operator Construction. Operator Construction takes a goal, problem description and a set of object hierarchies, and begins constructing a set of operators to satisfy the goal in the context of the problem description. The objective is to identify which objects, or combinations of objects, in the hierarchy can be used to achieve the goal.

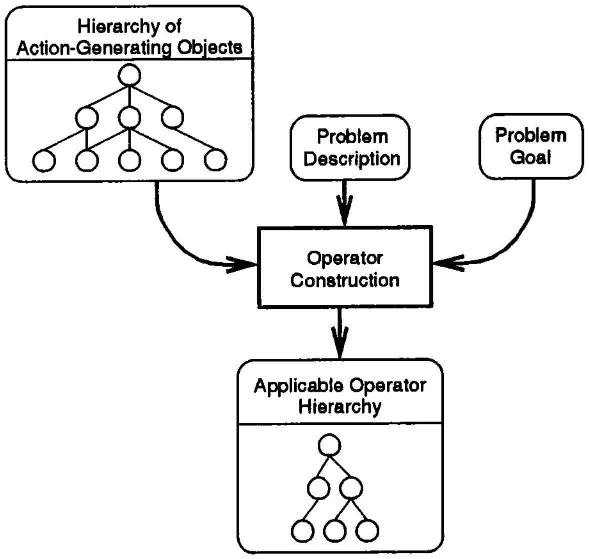
Hierarchy of Action-Generating Objects
Problem Description
Problem Goal
Operator Construction
Applicable Operator Hierarchy

Figure 4: Operator Construction system diagram.

Figure 5 contains an illustration of the Operator Construction process at work on a generalized object refinement hierarchy. Operator Construction starts by applying knowledge at the top level of the object hierarchy to the problem description. If the top level object can be used to solve the goal, then an initial abstract operator description is created to describe how that object will be used to accomplish that goal. Several preconditions and post conditions of the operator may be filled in as well. At each successive level of processing, the operator is refined: more pre and post conditions may be added and one abstract operator description may be split into several refined operators.

Figure 5 (b) shows the Operator Construction process at an intermediate stage. The dotted horizontal line indicates the current refinement level. The solid lines below this level are the paths still under consideration. The remaining paths have been pruned away.

```
(defun refine (problem node)
  (let ((C (children node)))
    (if (null? C)
      node
      (let ((methods '()))
        (list node
              (dolist (child C methods)
                      (when (applicable? problem child)
                            (setf methods
                                  (cons (refine problem child)
                                        methods))))))))
```

Figure 6: Refinement algorithm.

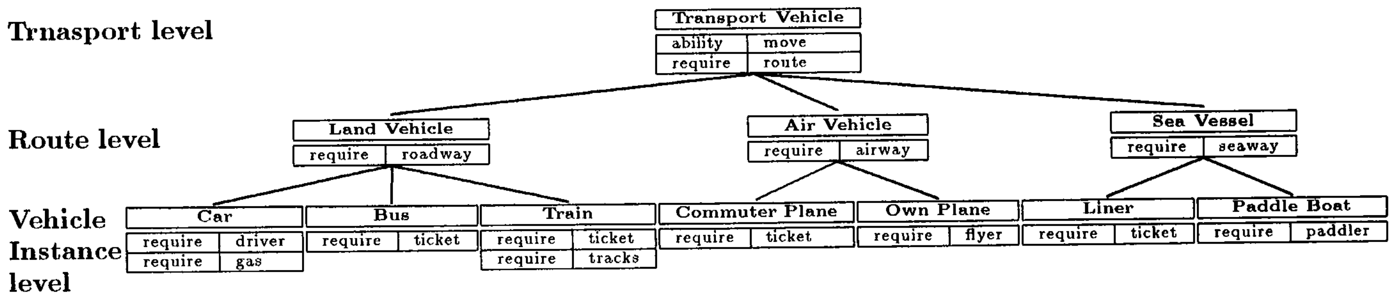

Figure 3: The operator construction hierarchy of action-generating objects for the traveling domain.

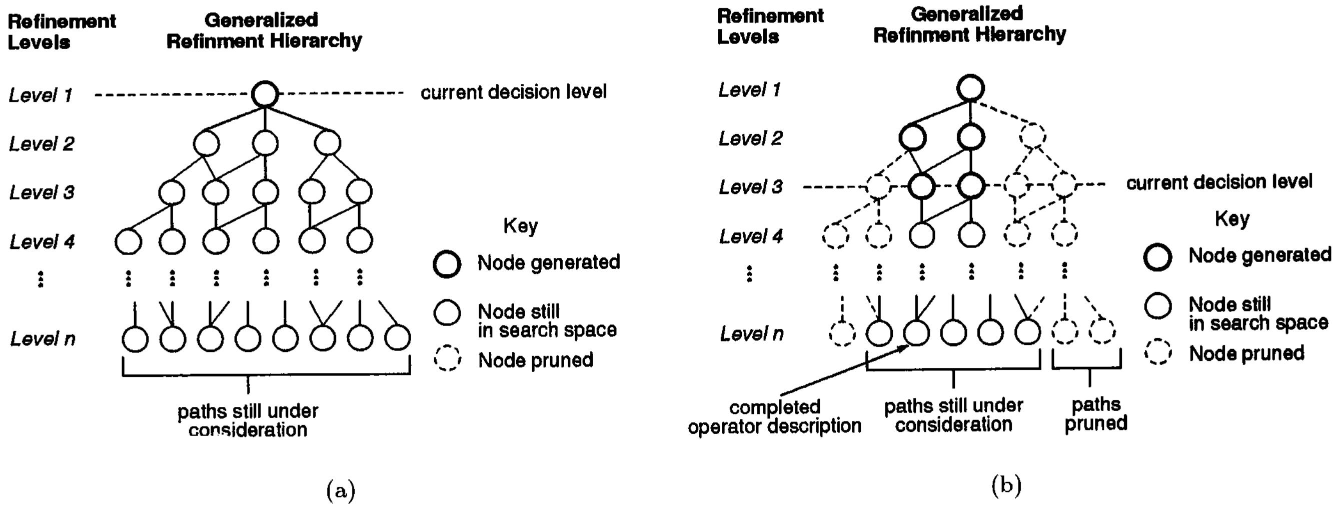

Figure 5: A generalized operator refinement hierarchy.

Figure 6 contains the refinement algorithm used by Operator Construction. `Refine` takes as input a node from an object refinement hierarchy and a problem statement. The algorithm uses the predicate `applicable?` to check the applicability of each child of the current node towards the given problem. The decision knowledge at the node is used to determine if the operator options represented by the current node apply to the goal. A refined operator is created for each applicable child. The refined operator may have more parameters specified or additional constraints placed on previously constrained parameters.

The result is a hierarchy of operators suitable for carrying out the task at each level of abstraction. This hierarchy is analogous to an instantiated portion of SIPE's operator refinement hierarchy, Figure 2, applicable towards the given goal. These operators can be passed back to the planner for consideration. The benefits are that the object refinement hierarchy is easier to maintain than operator descriptions and the planner's actions are not limited to the set of a priori enumeration of operator descriptions.

Example: Traveling to Portland

As an example of Operator Construction, consider the problem of deciding how to travel from Los Angeles to Portland. The Operator Construction process is given the object hierarchy in Figure 3 which describes the available transport objects and their properties. The problem description is shown in Figure 7.

The Operator Construction process begins at the Transport level (Figure 3. The object, *Transport* Vehicle, is able to achieve the goal of move, but it requires that a route of some type be present between the start and the destination. A transport operator is created to show that there is some abstract vehicle description that can accomplish the goal. The constructed operator description is shown in Figure 8. The

Destination	Portland
	has roadway
	has airway
	no seaway
Origin	Los Angeles
	has roadway
	has airway
	no seaway
Can Drive	true
Can Fly	false

Figure 7: Example problem description.

effect of the operator is that after its application, the traveler will be at Portland. The operator includes the precondition that to go from LA to Portland, one must be in LA.

Transport level

Transport	
precondition	at LA
vehicle	transport
effect	at Portland

Figure 8: Operator hierarchy at level 1.

The Operator Construction process proceeds to the Route level of the object hierarchy (Figure 3) and checks the type of routes available for traveling form LA to Portland in the problem description against the route requirements of the available vehicle types in the object hierarchy: *Land*, *Sea*, and *Air*. In this case, *Land* and *Air* are applicable, but since the example does not include a seaway, *Sea* vehicles are not applicable to the problem description. Figure 9 shows the result of Operator Construction after the Route level. The abstract operator of the Transport level is refined and split into two, slightly less abstract operators at the route level. No new preconditions or effects have been added to the generated operators, but the operator space has been pruned down to include only those vehicles under the *Land* and *Air* nodes.

Each operator is further refined as Operator Construction moves down to the third and final level of the object hierarchy.

At this level, the requirement of particular vehicles types, like cars, busses, etc., are tested to see if there are any which can be used in the given situation.

The *Land* operator from level 2 is refined by checking which of its children are applicable to this particular goal. *Car* is valid because it has no further constraints on the land route and the traveler is able to

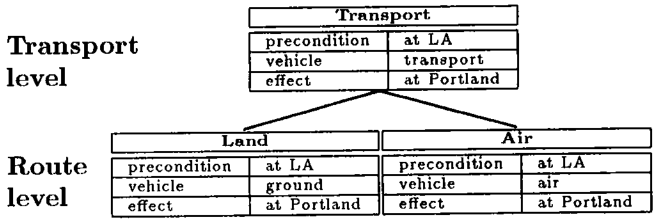

Figure 9: Operator hierarchy after level 2.

drive. Nothing in the problem description has been said about the availability of gas, so Operator Construction simply adds this as a precondition to using the generated *Car* operator in Figure 10. *Train* is not valid because there are no train tracks running from LA to Portland. *Bus* object is applicable, but the precondition of having a ticket is added to the constructed operator.

The *Air* operator is refined in a similar manner. The traveler in the example is not able to fly a plane, so *Own Plane* is not applicable. *Commuter Plane* is applicable, but has the added requirement that a ticket is needed.

Figure 10 contains the operators constructed at this level. The leaf nodes represent several possible alternative instantiated operators. The paths leading to these nodes represent the decisions made in constructing the operator.

Each resulting operator contains the required information, effects and preconditions, and can be inserted into more complex plans. The hierarchy generated in the example is, in fact, a portion of SIPE's operator refinement hierarchy shown in Figure 2, which has been instantiated for the particular goal of traveling from LA to Portland. The main difference is that the hierarchy generated in Figure 10 is instantiated for the particular problem. Start and destination locations have been replaced with LA and Portland. The generated operators have fewer preconditions than the SIPE operators. For example, there is no need to add a precondition that a roadway exists between LA and Portland in order to use the generated *Car* operator, since this operator description was created because there is a roadway connecting these two locations. If there had not been a roadway, this operator would not be created.

The main difference between the Operator Construction method and other hierarchical planners, is the way in which the knowledge about actions is stored. The traditional method is to store this knowledge in terms of an operator hierarchy. In Operator Construction, this knowledge is stored in terms of domain objects that produce actions. The advantage of storing this knowledge as objects instead of operators is that con-

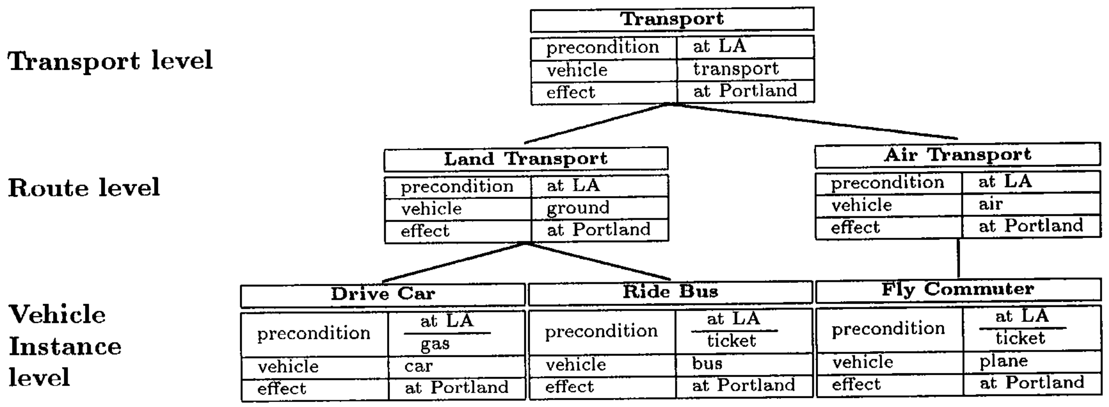

Figure 10: Operator refinement hierarchy for the traveling domain.

crete domain objects are much easier to describe and maintain than operators which are abstract concepts.

Operator Construction's Impact on the Planner

Ease of maintenance is an aspect of a planning system's efficiency which is often overlooked. Operator Construction provides an explicit link between the actions the planner can take and the objects in the domain which produce these actions. This makes Operator Construction more maintainable than approaches which use operator selection. With operator selection approaches such as SIPE and MOLGEN, changes in the domain may require extensive modifications to the operator descriptions. Because the operators may contain knowledge relevant to several domain objects, it may not always be obvious which modifications to make. In contrast to the operator selection approach, in Operator Construction, the knowledge about the altered objects can be easily updated by modifying the knowledge-refinement hierarchies. The Operator Construction process will generate operator descriptions based on this new knowledge.

Conclusions

This paper has presented Operator Construction, a technique which is especially applicable to complex physical domains. In this technique, operator instantiations are created by a process of successive refinement from descriptions of the domain properties. By linking objects with the constructed operator descriptions, the approach provides a means for reasoning about domain knowledge that produces action. It differs from closely related techniques, like those used in MOLGEN (Stefik 1981b) and SIPE (Wilkins 1984) in that Operator Construction does not contain any explicit representations of generalized operators, or operator refinement hierarchies. Instead, it generates only the instantiated portions of the operator hierarchy that are applicable to the given goal.

The reasoning behind the Operator Construction technique is that the actions and results that can be produced in a physical domain are a direct result of the properties of the objects that produce those actions. For example, the wooden shapes that can be generated in a particular wood shop are a direct result of the characteristics of the wood-cutting tools and machines in the shop. We feel a natural and maintainable way to represent actions is to represent the important properties of the domain objects that produce those actions and to use those descriptions to generate operator instantiations. This type of representation is very maintainable because it is easier to describe the change to the action producing objects, than it is to describe the changes to possible actions that result from changing the action producing objects. Instead of representing abstract operators, we have constructed an operator generator.

Advantages of this method include a maintainable representation, and a more direct link between domain properties and possible domain actions. This technique has been successfully implemented in a complex manufacturing domain. In ongoing work on a systems called MEDIATOR (Gaines, Castaño, & Hayes 1996) we are using a related technique to identify abstract problem goals embedded in a problem description.

Acknowledgments

Funding for this work was provided by NASA grant NAG-1-613. The traveling domain was adapted from Wilkins' example which can be found at `http://www.ai.sri.com/~wilkins/`.

References

Gaines, D. M.; Castaño, F.; and Hayes, C. C. 1996. MEDIATOR: Reconfigurable feature recognition for a maintainable, extendible CAD/CAPP integration. Submitted to *ASME Journal of Mechanical Design.*

Hayes, C. C. 1996. P3: A process planner for manufacturability analysis. *IEEE Transactions of Robotics and Automation, special issue on Assembly and Task Planning* 12(2).

Stefik, M. 1981a. Planning and meta-planning (MOLGEN part 2). *Artificial Intelligence* 16(2).

Stefik, M. 1981b. Planning with constraints (MOLGEN part 1). *Artificial Intelligence* 16(2).

Wilkins, D. E. 1984. Domain-independent planning: Representation and plan generation. *Artificial Intelligence* 22(3).

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Paradigm.* Morgan Kaufman.