Reasoning about Locations in Theory and Practice

Karen L. Myers David E. Wilkins

Artificial Intelligence Center

SRI International

333 Ravenswood Ave.

Menlo Park, CA 94025

myers@ai.sri.com wilkins@ai.sri.com

415-859-4833 415-859-2057

February 7, 1997

Abstract

Locational reasoning plays an important role in many applications of AI problem-solving systems, yet has remained a relatively unexplored area of research. This paper addresses both theoretical and practical issues relevant to reasoning about locations. We define several theories of location designed for use in various settings, along with a sound and complete belief revision calculus for each that maintains a STRIPS-style database of locational facts. Techniques for the efficient operationalization of the belief revision rules in planning frameworks are presented. These techniques were developed during application of the location theories to several large-scale planning tasks within the SIPE–2 planning framework.

Keywords: locational reasoning, belief revision, generative planning, hierarchical task networks, experimental AI

1 Introduction

Much progress has been made in recent years in the development of knowledge-based problem-solving systems that can solve tasks of practical interest. Planners, reactive controllers, and schedulers, for example, are being applied successfully to increasingly larger and more sophisticated problems [1, 11, 23, 34, 29]. The research contributions in these efforts have emphasized the technologies themselves; for example, developing generalized frameworks and more efficient problem-solving algorithms. In contrast, relatively little effort has been devoted to the development of well-grounded theories for the underlying *domain knowledge* required by such systems. Certainly, there have been advances in knowledge representation for numerous areas; however, work on representations generally ignores computational issues. Principled representations of domain knowledge with good computational properties are essential for nontrivial applications. Nevertheless, representational theories driven by practical considerations remain under-explored for many critical topics.

This paper presents a commonsense theory for *locational reasoning* that is motivated by both theoretical and practical concerns. Our approach deviates from much of the work on commonsense theories in two ways. First, we make no attempt to provide a *universal* theory that covers all eventualities (as, for example, was the motivation for Hayes' work on liquids [15]). Instead, we define a *practical* theory focused on those aspects of locational reasoning that are necessary and sufficient for a broad class of meaningful applications. While a more in-depth treatment of locational reasoning is essential for certain domains, we believe that the theory provided here is epistemically adequate for many problems of interest. Second, we are motivated by the goal of applying our theory within implemented problem-solving systems. For this reason, we present techniques for the efficient operationalization of our formal theory, with emphasis on its use in planning systems.

Our theory models a location as a point in a predefined space of places, which can span multiple levels of abstraction. The locations of objects change as a result of discrete movement operations. Objects are immutable in the sense that their basic structure does not change. However, they can be comprised of a finite set of predefined components, each of which can be moved independently. While not universal, this model of locations is useful for a broad range of applications that require basic reasoning about objects whose positions change over time. Examples include directed navigation for autonomous vehicles (such as mobile robots), transportation planning, military operations, production line scheduling, and real-time tracking.

There has been no previous work to date of which we are aware on locational theories of the sort described in this paper. In the knowledge representation community, locational reasoning is overshadowed by the more general topic of *spatial reasoning* (for an overview, consult [8]; more

recent work is described by [4, 26, 17]). Spatial theories differ from our locational theory in several regards. Theories for spatial reasoning focus on topics such as the articulation and axiomization of notions of shape, spatial occupancy, and qualitative spatial relationships (such as *near* and *somewhere*). These concepts are beyond the scope of what is required for basic locational reasoning. Most spatial theories lack notions inherent to our theory of locations, such as partitioning of objects and abstraction hierarchies for locations. There have been few efforts to incorporate reasoning about movement in spatial settings, as noted by [28], whereas movement is integral to our theory. Finally, work on spatial reasoning has for the most part ignored computational issues; thus, while many of the spatial theories are expressively powerful, they are not amenable to efficient reasoning.

There has been success in defining representation theories with good computational properties for other domains of general interest. One example is the development of formalisms, semantics, and algorithms for taxonomic reasoning systems. A second example is the work on temporal reasoning, initally undertaken by Allen [2] but extensively developed by others in follow-on efforts. The work on *specialist reasoners* [27] has examined several domains, including time, colors, and taxonomies. Our work provides comparable theoretical and practical foundations for the task of reasoning about locations, with particular emphasis on applications to planning systems.

The paper begins with the presentation of a basic model of location, which is progressively enriched to include notions such as multiple abstraction levels and aggregation. For each case we present a formal theory and discuss the situations to which it applies. Next, we define a belief revision framework along with a provably correct set of belief revision rules for maintaining a STRIPS-style database in accordance with the constraints of each theory.

The second half of the paper presents techniques for operationalizing the belief revision rules in order to provide efficient reasoning about locations. The emphasis is on planning systems, although many of the ideas apply more generally to other problem-solving frameworks. For the sake of concreteness, we describe these techniques in the context of a specific planning framework, namely SIPE–2 (System for Interactive Planning and Execution) [31, 32, 33].[1] Experimental results are provided that show how the proposed operationalization improves efficiency substantially over a direct translation of the belief revision rules. The experimental results are based on problems from two domains that make extensive use of locational reasoning, namely planning of military operations [34], and planning the actions of a mobile robot [31].

[1] SIPE–2 is a trademark of SRI International.

2 Theories of Location

The formal theories of location are defined using a first-order language $\mathcal{L}$. The vocabulary of $\mathcal{L}$ contains (among others) the predicate:

$$\mathrm{AT}(x, l) \qquad \text{object } x \text{ is at location } l$$

Location variables are represented using $l, l_1, l_2, \ldots$ while object variables are represented using $x, y, z, x_1, y_1, z_1, \ldots$ We further assume the use of a predicate $=$ (written using infix notation for the sake of clarity) with the standard interpretation for equality [35]. Other predicates will be introduced as required.

To simplify the presentation, we assume that locational information is to be maintained for all objects in our domain; in other words, it is meaningful to associate a location with all objects.[2]

2.1 A Basic Theory of Location

The characteristic properties of location for our basic theory are captured by the following two constraints.

Every object has a location.
No object has more than one location.

The constraints are captured formally by the axioms

$$\forall x\ \exists l.\ \mathrm{AT}(x, l) \tag{1}$$

$$\forall x\ \forall l_1\ \forall l_2.\ \mathrm{AT}(x, l_1) \wedge \mathrm{AT}(x, l_2)\ \supset\ l_1 = l_2 \tag{2}$$

We refer to this pair of axioms as the *basic theory of locations*, written as $\mathcal{T}_L^0$.

2.2 Uniqueness of Locatee

For certain classes of locations, the following constraint should be added to the basic theory:

At most one object can be at a given location.

[2] This assumption does not hold in general (*e.g.*, consider abstract concepts such as numbers) but is easily relaxed. One solution is to use a typed logic that includes a type *locatable-object*. Alternatively, a monadic predicate could be defined whose extension consists of all objects for which locations are definable; all location axioms would be relativized to objects that satisfy this introduced predicate.

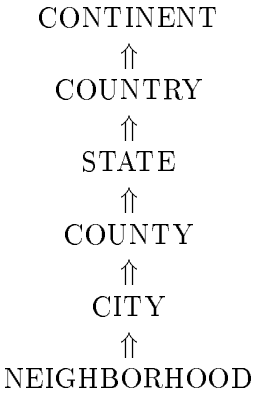

Figure 1: Geopolitical Place Abstraction

The constraint is necessary when reasoning about parking spaces or seat assignments. It is also typically adopted for blocks world problems. The constraint is not part of a general theory of locations because it is inappropriate for locations on a large scale: for example, a city, state or country usually contains more than one person.

Extending the basic theory to incorporate uniqueness of locatee requires the additional axiom:

$$\forall l\ \forall x\ \forall y.\ \mathrm{AT}(x,l) \wedge \mathrm{AT}(y,l)\ \supset\ x = y \tag{3}$$

2.3 Place Abstractions

The two theories presented above assume that all locations are at the same level of specificity. More generally, reasoning about places must be done at multiple levels of abstraction. The geopolitical place abstraction hierarchy depicted in Figure 1, which might be used in transportation planning, provides an example. Similarly, the description of an office building for a robot that must plan navigational routes might contain multiple levels of abstraction, as shown in Figure 2. This second example illustrates how multiple types of locations can share a single level of abstraction (for example, ROOM, CORRIDOR, and JUNCTION). Multiple levels of abstraction have been shown to be valuable in terms of more efficient planning: for example, when planning a trip from Palo Alto to Boston, the search can be greatly constrained by first deciding how to get from California to Massachusetts rather than immediately starting to plan low-level details [16].

The following statements characterize a revision of the basic theory of locations to include place abstractions. The first two statements are generalizations of the defining principles from the basic theory. The third expresses the *containment principle*.

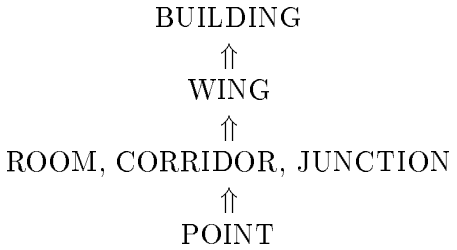

Figure 2: Place Abstraction for Office Buildings

Every object has a location at every level of abstraction.
No object can be at more than one location within a single level of abstraction.
When an object is at a location contained within another location,
the object is also at the containing location.

Formalization of the theory of locations with multiple levels of abstraction requires the introduction of the predicate WITHIN and function LEVEL:

$\text{WITHIN}(l_1, l_2)$	location l_1 is directly contained within location l_2
$\text{LEVEL}(l)$	the abstraction level of location l

By *directly contained* in the characterization of $\text{WITHIN}(l_1, l_2)$, we mean that l_1 must be at the level of abstraction immediately below that of l_2.

The requisite axioms are:

$$\forall x\ \forall l\ \exists l'.\ \text{LEVEL}(l) = \text{LEVEL}(l')\ \wedge\ \text{AT}(x, l') \tag{4}$$

$$\forall x\ \forall l_1\ \forall l_2.\ \text{AT}(x, l_1) \wedge \text{AT}(x, l_2) \wedge\ l_1 \neq l_2\ \supset\ \text{LEVEL}(l_1) \neq \text{LEVEL}(l_2) \tag{5}$$

$$\forall x\ \forall l_1\ \forall l_2.\ \text{AT}(x, l_1) \wedge \text{WITHIN}(l_1, l_2)\ \supset\ \text{AT}(x, l_2) \tag{6}$$

We refer to the collection of axioms (4) – (6) as the *theory of hierarchical locations*, and denote it by $\mathcal{T}_L^H$.

It is possible to dispense with the LEVEL function in the above formalization by employing AT predicates at multiple levels of abstraction instead. For example, the geopolitical abstraction hierarchy would require predicates AT-NEIGHBORHOOD, AT-CITY, AT-COUNTY, AT-STATE, AT-COUNTRY, and AT-CONTINENT. Using AT– predicates for each level of abstraction leads to many more number of axioms: a version of axiom (5) would be required for each level, and axioms

in the spirit of (6) would be required to link each level with the level immediately above it in the hierarchy. Since the original formalization is domain-independent and is also more perspicuous, we make use of it throughout this paper. However, we do consider the computational properties of the two alternatives in Section 6.4.4.

2.4 Aggregation

To this point, our theories of location have assumed that objects are indivisible. We now consider an object that can be partitioned into a finite set of discrete subobjects, each of which can be moved independently. We call a partitionable object of this sort an *aggregate*. An army, whose units may be stationed at different sites, provides a good example.

To simplify the presentation, we assume that there is only a single level of abstraction for locations throughout this subsection; the following subsection describes a theory that combines aggregation with location abstractions.

Our theory of location for aggregates is defined by the following characteristics:

All non-aggregate objects have a location.
When all subparts of an object are at a location, the object is at that location.
When all subparts of an object are not co-located, the object has no location.

The first property states that non-partitionable objects always have a location. The second property dictates that the location of an aggregate object is l when its components are all located at l. The third property, which we call the *co-location requirement for aggregates*, is somewhat more controversial. This property applies to many of the objects typically reasoned about in planning domains, which motivates its inclusion here. Armies with their independent units provides one example; cargo that must be partitioned into independently moved shipments is a second example. We note that in contrast to other models of splitting objects [9], an aggregate in our theory continues to exist when its subparts are not co-located; only the *location* of the aggregate object ceases to be defined.

The co-location requirement is not universal. Consider, for example, a woodpile, whose location would persist despite the removal of some proper subset of the logs (up to a certain point). When modeling domains that contain aggregate objects of this type, the scope of the co-location requirement should be restricted to exclude such objects.

The axiomatization of the aggregation properties makes use of the following predicate:

$$\text{SUBPART}(x_1, x) \quad x_1 \text{ is an independently movable subobject of } x$$

SUBPART denotes 1-step rather than transitive decomposition: while an army may be partitioned into brigade subparts and each brigade may be further partitioned into unit subparts, units would not be subparts of the army. We use the term *ancestor/descendant* to refer to the transitive closure upward/downward of the SUBPART relation.

The axiomatic statement of the three aggregation properties is as follows:

$$\forall x \;\; (\neg\exists x_1 \; \text{SUBPART}(x_1, x)) \;\vee\; \exists l \; \text{AT}(x, l) \tag{7}$$

$$\forall x \; l. \; (\forall x_1 \; \neg\text{SUBPART}(x_1, x) \vee \text{AT}(x_1, l)) \;\supset\; \text{AT}(x, l) \tag{8}$$

$$\begin{aligned} \forall x \; x_1 \; x_2 \; l_1 \; l_2. \; \text{AT}(x_1, l_1) \;\wedge\; \text{AT}(x_2, l_2) \;\wedge\; l_1 \neq l_2 \wedge \\ \text{SUBPART}(x_1, x) \wedge \text{SUBPART}(x_2, x) \quad \supset \quad \neg\exists l. \; \text{AT}(x, l) \end{aligned} \tag{9}$$

The axioms (7) – (9) along with (2) comprise the *theory of locations for aggregates*, denoted by $\mathcal{T}_L^A$. (To exclude the co-location requirement for aggregates, axiom (9) should be omitted.)

The following is a direct consequence of axioms (8) and (9):

$$\forall x \; x_1 \; l. \; \text{AT}(x, l) \wedge \text{SUBPART}(x_1, x) \;\supset\; \text{AT}(x_1, l) \tag{10}$$

This formula states that when an aggregate object is at a location, then so are all of its subparts.

2.5 The Full Theory

Finally, we define a full theory of location that encompasses both location hierarchies and aggregation. The full theory does not consist of $\mathcal{T}_L^H \cup \mathcal{T}_L^A$. In particular, the constraint (4) from $\mathcal{T}_L^H$ that requires all objects to have a location at each level of abstraction conflicts with (9) from $\mathcal{T}_L^A$. The axiom (4) is omitted, leaving the weaker existence constraints (7) and (9). In addition, the *co-location requirement* for aggregates must be relativized to individual levels of abstraction:

> *When all subparts of an object are not co-located at a given abstraction level, the object has no location at that abstraction level.*

The axiom (9) is modified as follows to correspond to this relativized principle:

$$\begin{aligned} \forall x \; y_1 \; y_2 \; l_1 \; l_2. \; \text{AT}(y_1, l_1) \wedge \text{AT}(y_2, l_2) \wedge \text{LEVEL}(l_1) = \text{LEVEL}(l_2) \wedge \; l_1 \neq l_2 \wedge \\ \text{SUBPART}(y_1, x) \wedge \text{SUBPART}(y_2, x) \supset \; \neg\exists l. \; \text{AT}(x, l) \wedge \text{LEVEL}(l) = \text{LEVEL}(l_1) \end{aligned} \tag{11}$$

We refer to the full theory using the symbol $\mathcal{T}^L$.

3 Movement

The theories defined in Section 2 focus exclusively on location information for stationary objects. In this section, we extend the theories to include the effects of movement.

3.1 Assumptions

As has been well-documented in the literature, formalizing action is a challenging endeavor. Complexity arises in the form of the frame [20], ramification [10, 13], and qualification [19] problems. We employ several assumptions in this paper for simplifying the task of reasoning about move actions.

First, we assume that move actions always succeed when their preconditions are satisfied. This assumption enables us to focus on the issues of maintaining locational information rather than the many subtle problems that relate to execution failure [14].

Second, we assume a single, discrete move action at each time step. We note, however, that it is straightforward to generalize our formalism to support parallel move actions provided that the parallel actions are *non-interfering*; however, we refrain from doing so in order to keep the presentation simple. By non-interfering, we mean that that the executability of one action is not impacted by the simultaneous execution of a second action (*e.g.*, trying to move two objects to the same location at one time, when only one object can occupy that location). Our implementation of the location theories using SIPE–2 operators (discussed in Section 6.3.1) supports parallelism with this restriction. Allowing parallel move activities that can interfere raises a number of difficult technical and ontological problems (including the ramification problem) which we do not wish to address. Many planning domains do not require the use of potentially interfering parallel move operations. In planning military operations, for example, the movement of troops, cargo and machinery is highly parallel but the simultaneously planned actions are non-interfering,

Finally, we do not provide movement theories that contain the uniqueness of locatee axiom (3). The use of this axiom requires domain-specific information to characterize the effects of movement. For certain applications (e.g., putting cars in parking spaces, seat assignments), the appropriate behavior is for a move action to be classified as unexecutable whenever the destination of the move is not vacant. However, in domains where the movement of an action can lead to the clearing of a target destination in a predictable manner (e.g., the object currently at the target location may get crushed or pushed to a new location), it is necessary to customize the effects of movement operators in accord with domain characteristics. Again, we wish to concentrate on issue of reasoning about locations, not general problems in reasoning about actions.

3.2 Movement Theories

Our basic principles relating movement to location are:

> *An object will be at a given location when moved there.*
> *Unmoved objects remain undisturbed.*

Our formalization of these principles makes use of the situation calculus [20] for the first-order language $\mathcal{L}^{\mathcal{M}}$. This language generalizes $\mathcal{L}$ from the previous section in that it employs situation variables $s, s_1, s_2, \ldots$ and replaces the predicate $\mathrm{AT}(x, l)$ by a fluent of the form $\mathrm{AT}(x, l, s)$. In addition, $\mathcal{L}^{\mathcal{M}}$ contains the action fluent $\mathrm{MOVE}(x, l, s)$, which serves as the sole transition function between situations.

For a single level of place abstraction, the movement-location relationship is captured by the *relocation axiom* (12) and *inertia axiom* (13):

$$\forall x\ l\ s.\ \mathrm{AT}(x, l, \mathrm{MOVE}(x, l, s)) \tag{12}$$

$$\forall x\ y\ l_x\ l_y\ s.\ \mathrm{AT}(y, l_y, s) \wedge x \neq y\ \supset\ \mathrm{AT}(y, l_y, \mathrm{MOVE}(x, l_x, s)) \tag{13}$$

To accommodate aggregates, the axiom (13) must be refined to exclude subparts since they necessarily move with their superparts:

$$\forall x\ x_1\ l_1\ l_2\ s.\ \mathrm{AT}(x_1, l_1, s) \wedge x \neq x_1 \wedge \neg\mathrm{SUBPART}(x_1, x)\ \supset\ \mathrm{AT}(x_1, l_1, \mathrm{MOVE}(x_1, l_1, s)) \tag{14}$$

Note that the movement of subparts with their aggregate need not be explicitly axiomatized, as this movement is a consequence of axiom (7).

In planning domains, it is often the case that movements are specified at a certain level of abstraction without committing to particular lower-level locations. For instance, a robot may plan to navigate to a specific room without identifying a specific coordinate pair within the room. For this reason, we introduce the constraint of *locational indeterminacy*:

> *After moving an object at a given level of abstraction, the object could be at any location at lower levels of abstraction that is contained within the higher-level location.*

Let $\text{WITHIN-N}(l_1, l_2)$ denote the relation that (l_1, l_2) is in the transitive closure of WITHIN. Define $<$ to be a partial order on abstraction levels such that $\mathrm{LEVEL}(l_0) < \mathrm{LEVEL}(l_1)$ when l_0 is at a lower level of abstraction than l_1. The constraint of *locational indeterminacy* can be expressed as follows:

$$\forall x\ y\ l\ l_0\ s.\ \text{LEVEL}(l_0) < \text{LEVEL}(l)\ \supset \tag{15}$$
$$\exists l_1\ \ \text{LEVEL}(l_1) = \text{LEVEL}(l_0)\ \wedge\ \text{WITHIN-N}(l_1, l) \wedge\ \text{AT}(x, l_1, \text{MOVE}(x, l, s))$$

This axiom states that for every location l_0 whose abstraction level is below that of the location to which an object was moved, there is some location at the same level as l_0 where the object resides. As was the case with aggregates, there is no need for a corresponding 'upward' axiom to account for the position of encompassing locations higher in the abstraction hierarchy; these changes are covered by the location axiom (5).

To complement the theories of location defined above, we define four separate theories of movement. The basic theory of movement, denoted by $\mathcal{T}_M^0$, consists of a situationalized version of $\mathcal{T}_L^0$ plus (12) and (13).[3] The theory of movement for aggregates, denoted by $\mathcal{T}_M^A$, consists of a situationalized version of $\mathcal{T}_L^A$ with (12) and (14). The theory of movement for hierarchical locations consists of a situationalized version of $\mathcal{T}_L^H$ along with (12), (13) and (15). The full theory of movement, denoted by $\mathcal{T}^M$, consists of a situationalized version of $\mathcal{T}^L$ along with (12), (14), and (15).

It is interesting to note that we have formalized our theories of movement compactly in a first-order language, without recourse to any of the complex nonmonotonic reasoning schemes often deemed essential for reasoning about action. While it is true that one can readily construct situations that lie beyond the scope of the theories presented here, our experiences in the domains of robot navigation and military operations planning indicate that the basic theories suffice for many practical problems.

4 Belief Revision

We now present a syntax for belief revision rules that can be used to maintain a STRIPS-style database of unnegated propositional facts. In the following section, we provide sets of belief revision rules within this framework that operationalize the theories of location presented above.

Our belief revision framework supports both the addition and retraction of facts, initiated by either operators (in the case of a planner) or sensors (in the case of a situated reactive system). We have chosen to use a belief revision calculus for describing our treatment of locational reasoning rather than a specific planning formalism since the ideas apply to the more general case.

[3] By situationalized, we mean that instances of the 2-ary predicate $\text{AT}(x, l)$ has been replaced by the fluent version $\text{AT}(x, l, s)$ and universally quantified situation variables have been introduced, as appropriate. For example, $\forall s\ x\ \exists l.\ \text{AT}(x, l, s)$ constitutes a situationized version of (1).

4.1 Belief Revision Framework

Belief revision rules (or *BR-rules*) are defined as follows. Let VARS($\{L_1, \ldots, L_k\}$) denote the set of variables appearing in the literals $L_1, \ldots, L_k$.

Definition 1 (Belief Revision Rules) Belief revision rules *have the form:*

$$P_1, \ \ldots \ P_k, \ T^+ \ \rightarrow \ A_1{}^+, \ \ldots \ A_m{}^+, \ R_1{}^-, \ \ldots \ R_n{}^- \qquad \textit{(Insertion Rule)}$$

$$P_1, \ \ldots \ P_k, \ T^- \ \rightarrow \ A_1{}^+, \ \ldots \ A_m{}^+, \ R_1{}^-, \ \ldots \ R_n{}^- \qquad \textit{(Retraction Rule)}$$

where $m, n, k \geq 0$ *and* T,P_i,A_i,R_i *are positive literals and*

$$\mathit{VARS}(\{P_1, \ldots \ P_k, A_1, \ldots, A_m, R_1, \ldots, R_n\} \subseteq \mathit{VARS}(\{T\}) \ .$$

T *is referred to as the* trigger fact, *being* positive *for the insertion and* negative *for the retraction rule. The* P_i *are called the* gating *conditions for the rules.*

Explanation of the intended semantics of a BR-rule requires some notation for substitutions and unification. We write $L : \sigma$ to denote the literal L under substitution σ, and $\{L_1, \ldots, L_k\} : \sigma$ to denote the set $\{L_1 : \sigma, \ldots, L_k : \sigma\}$. The notation $\beta : \sigma$ denotes the instantiation of a BR-rule for the substitution σ.

BR-rules are intended to be used to modify a database in response to the specification of a ground literal serving as an *update*. The operational semantics for the insertion rule is as follows: if S is a positive ground literal that matches trigger T for the substitution σ (*i.e.*, $S = T : \sigma$), then when S is added to a database containing (among others) the literals $P_1 : \sigma, \ \ldots \ P_k : \sigma$, the literals in $\{A_1 : \sigma, \ \ldots \ A_m : \sigma\}$ (referred to as the *add list*) should be added and the literals $\{R_1 : \sigma, \ \ldots \ R_n : \sigma\}$ (referred to as the *delete list*) removed. For the retraction rule, when a ground literal S that matches T for a substitution σ is removed from a database containing literals $P_1 : \sigma, \ \ldots \ P_k : \sigma$, the literals $A_1 : \sigma, \ \ldots \ A_m : \sigma$ should be added and the literals $R_1 : \sigma, \ \ldots \ R_n : \sigma$ removed. The constraint on VARS($\{T\}$) in Definition 1 ensures that for any substitution σ for which $S = T : \sigma$, all variables in a BR-rule are bound by σ.

This behavior is captured by the following definitions for applicability and execution of belief revision rules. All updates are assumed to be positive ground literals, with the notation S^+ indicating the addition of the literal S, and S^- indicating the removal of S.

Definition 2 (BR-rule Applicability) *Let* $\mathcal{D}$ *be a propositional database and* S *be a positive ground literal. An insertion rule*

$$P_1, \ \ldots \ P_k, \ T^+ \ \rightarrow \ A_1{}^+, \ \ldots \ A_m{}^+, \ R_1{}^-, \ \ldots \ R_n{}^-$$

is applicable *for an update S^+ when $S \notin \mathcal{D}$, $S = T : \sigma$ for some substitution σ, and $\{P_1, \ldots P_k\} : \sigma \subseteq \mathcal{D}$. A retraction rule*

$$P_1, \ldots P_k, T^- \rightarrow A_1{}^+, \ldots A_m{}^+, R_1{}^-, \ldots R_n{}^-$$

is applicable *for update S^- when $S \in \mathcal{D}$, $S = T : \sigma$ for some substitution σ, and $\{P_1, \ldots P_k\} : \sigma \subseteq \mathcal{D}$. We denote the set of instantiated BR-rules that apply for an update U by $APPLIES(U, \mathcal{R})$. The* effects *for an instantiated BR-rule $\beta : \sigma$ are denoted by $EFFECTS(\beta, \sigma)$ and defined to be the list $[A_1{}^+ : \sigma, \ldots A_m{}^+ : \sigma, R_1{}^- : \sigma, \ldots R_n{}^-\sigma]$.*

Execution of a set of BR-rules for a given update proceeds in breadth-first fashion: the update is first applied to the database, and then the updates in the effects of the applicable rules for the update are in turn processed. This cycle repeats until no further updates apply. the formal definition of BR-rule execution below makes use of the following list notation: $[U;\mathcal{U}]$ denotes a list of updates whose head is U and tail $\mathcal{U}$, $[]$ denotes the empty list, and the operators $+$ and $\sum$ represent list concatenation.

Definition 3 (BR-rule Execution) *Let $\mathcal{R}$ be a collection of BR-rules, $\mathcal{D}$ be a propositional database, $\mathcal{U}$ a list of updates, and S a ground proposition.* BR-rule execution *for $\mathcal{D}$, $\mathcal{R}$, and $\mathcal{U}$ is denoted by $\mathcal{D} \otimes \mathcal{U}$ and defined as follows:*

$$\mathcal{D} \otimes [] = \mathcal{D}$$

$$\mathcal{D} \otimes [S^+ \; ; \; \mathcal{U}] = (\mathcal{D} \cup \{S\}) \; \otimes \; (\mathcal{U} + \sum_{\beta:\sigma \in APPLIES(T^+, \mathcal{R})} EFFECTS(\beta : \sigma) \;)$$

$$\mathcal{D} \otimes [S^- \; ; \; \mathcal{U}] = (\mathcal{D} - \{S\}) \; \otimes \; (\mathcal{U} + \sum_{\beta:\sigma \in APPLIES(T^-, \mathcal{R})} EFFECTS(\beta : \sigma) \;)$$

We adopt the shorthand $\mathcal{D} \otimes U$ when referring to execution of BR-rules for a singleton update U. Note that given the restrictions that all updates are ground literals, and that that $\text{VARS}(\{P_1, \ldots P_k, A_1, \ldots, A_m, R_1, \ldots, R_n\} \subseteq \text{VARS}(\{T\})$ holds for BR-rules (as noted in Definition 1), it is straightforward to show that application of a BR-rule maps a database of ground literals to a new database that also contains only ground literals.

In the above definition, the summation of the effects for applicable BR-rules assigns an arbitrary order. In general, the sequencing of updates is not commutative. For the collections of BR-rules considered in this paper, however, the order is irrelevant.

4.2 Properties of BR-Systems

We are interested in defining a system of BR-rules that 'correctly implements' our theories of location and movement. In particular, each database generated by the BR-framework should correspond to an individual situation as modeled by the situation theory. In this section, we define the two properties that are important for validation purposes: *soundness* and *completeness* with respect to a situation theory.

The BR-framework is useful for modeling what we refer to as *Markov theories* of the situation calculus. Any fluent in a Markov theory is either a literal, or an axiom of the form

$$B_1[s] \ldots B_n[s] \supset C_1[s'] \ldots C_n[s']$$

where s' is the successor state of s. Thus, a Markov theory permits axioms such as (13), which links a state to its (unique) successor, but not

$$\forall y\ s.\ \mathrm{ONLIGHT}(y, s) \supset \mathrm{ONLIGHT}(y, \mathrm{TOGGLE}(y, \mathrm{TOGGLE}(y, s)))$$

which refers to a state more than one action distant. The Markov property is relevant for our purposes because the BR-framework maps a database and update to a single successor database. We restrict attention to Markov theories of the situation calculus throughout this document.

To establish that a given BR-system $\mathcal{R}$ correctly implements a Markov situation theory, it is necessary to define the correspondences between the theory and the databases generated by $\mathcal{R}$. A database represents a single state of the world and does not contain situational information; however, situational information is explicit in the fluents of the situation calculus. For fluents, we assume that the situational variables are always the last argument to a predicate or function (for example, $\mathrm{AT}(x, y, s)$), and that there is a corresponding situationless predicate or function in the language of the BR-system (for example, $\mathrm{AT}(x, y)$). To connect this pair, we introduce the operator *Sitn* that maps a situation-free predicate or function expression $\phi(t_1, \ldots, t_k)$ to a corresponding fluent; that is,

$$\mathit{Sitn}(\phi(t_1, \ldots, t_k), s) = \phi(t_1, \ldots, t_k, s)\ .$$

We also employ $\mathcal{S}$ to situationalize a set of predicates whose predicate symbols are drawn from some set $\mathcal{B}$:

$$\mathcal{S}(\Phi, \mathcal{B}, s) = \{P(t_1, \ldots, t_k, s)\ \mid\ P(t_1, \ldots, t_k) \in \Phi \text{ and } P \in \mathcal{B}\}\ .$$

For simplicity, we assume that the vocabularies of $\mathcal{R}$ and the theory are identical, except for the fluents of the situation theory and their situationless counterparts in $\mathcal{R}$. We use the symbol Q to

denote the set of predicate symbols used to define the fluents and their $\mathcal{R}$ counterparts. For our theories of location, Q contains only the symbol AT. As a slight abuse of terminology, we will refer to the set Q as the fluents used in a particular theory built from $\mathcal{L}^{\mathcal{M}}$.

The operator *Sitn* can be used to relate the semantics of individual databases generated by a BR-system with the corresponding situation theory. In particular, the semantics of a given database $\mathcal{D}$ used to model some specific situation for a situational theory $\mathcal{T}$ is characterized as:

$$\mathcal{T} \cup \mathcal{S}(\mathcal{D}, Q, s_0)$$

where s_0 is an introduced situation constant. The transition from one situation to another by means of an action in the situation calculus is modeled by the execution of an update in the BR-framework that adds or retracts propositions to the database corresponding to the direct effects of the action. Thus, each action in the situation theory necessarily maps to a set of updates. We let the function $\mathcal{F}$ denote this map and represent the state that results from execution of an action A in a situation s by $A(s)$.

Soundness and completeness mean that the belief rule system does the 'right thing' for a given theory in that it causes all (by completeness) and only all (by soundness) fluent consequences of the situation theory to be represented by their corresponding nonfluent projections in the database. To formalize these concepts, we first introduce the notions of *relative consistency* and *relative closure* for a database with respect to a theory.

Definition 4 (Relative Consistency) *Let $\mathcal{T}$ be a theory with fluents Q. A database $\mathcal{D}$ is* consistent relative to *$\mathcal{T}$ for Q iff for any introduced situation s_0 not in $\mathcal{T}$ the collection of formulas*

$$\mathcal{T} \cup \mathcal{S}(\mathcal{D}, Q, s_0)$$

is consistent.

Definition 5 (Relative Closure) *Let $\mathcal{T}$ be a theory with fluents Q. A database $\mathcal{D}$ is* closed relative to *$\mathcal{T}$ for Q iff for any introduced situation s_0 not in $\mathcal{T}$, when*

$$\mathcal{T} \cup \mathcal{S}(\mathcal{D}, Q, s_0) \models \mathcal{S}(p(t_1, \ldots, t_k), s_0)$$

for $p \in Q$ then $p(t_1, \ldots, t_k) \in \mathcal{D}$.

The combination of relative consistency and relative closure ensure that a database corresponds to a correct characterization of the fluents (relative to Q) in a possible situation. Using these concepts, soundness and completeness are defined as follows.

Definition 6 (Soundness of a Belief Rule System) *Let $\mathcal{T}$ be a theory with fluents Q and let $\mathcal{F}$ be an action map. A belief rule system $\mathcal{R}$ is* sound *with respect to $\mathcal{T}$, Q, and $\mathcal{F}$ iff for every database $\mathcal{D}$ that is closed and consistent relative to $\mathcal{T}$ for Q, when $p(t_1, \ldots, t_k) \in \mathcal{D} \otimes \mathcal{F}(A)$ for an action A then for any introduced situation s_0*

$$\mathcal{T} \cup \mathcal{S}(\mathcal{D}, Q, s_0) \models \mathrm{Sitn}(p(t_1, \ldots, t_k), A(s_0)) \ .$$

Definition 7 (Completeness of a Belief Rule System) *Let $\mathcal{T}$ be a theory with fluents Q and let $\mathcal{F}$ be an action map. A belief rule system $\mathcal{R}$ is* complete *with respect to $\mathcal{T}$, Q, and $\mathcal{F}$ iff for every database $\mathcal{D}$ that is closed and consistent relative to $\mathcal{T}$ for Q, when*

$$\mathcal{T} \cup \mathcal{S}(\mathcal{D}, Q, s_0) \models \mathrm{Sitn}(p(t_1, \ldots, t_k), A(s_0))$$

for action A, introduced situation s_0, and $p \in Q$, then $p(t_1, \ldots, t_k) \in \mathcal{D} \otimes \mathcal{F}(A)$.

4.3 Implementing BR-Rules

The belief revision rules can be translated directly into SIPE–2 deductive operators for use in planning applications. We describe one translation in subsection 6.3.3. Translations to planning operators in other frameworks are feasible, provided that the operators have some kind of *fact-invoked* or *event-based* capability to implement the triggers in BR-rules.

BR-rules are also useful in other classes of problem-solving systems. For instance, the rules can be used by a reactive execution system that operates in dynamic environments to maintain a database of facts that model the current state of the world. Once such system is the Procedural Reasoning System (PRS) [12].[4] The belief revision rules defined above are directly translatable to PRS *knowledge areas* (called *KAs*), which correspond to operator representations in traditional planning systems. We have used KA translations of the BR-rules for locational reasoning presented in the following section, as part of a reactive system designed to execute military deployment plans.

5 Reasoning about Locations and Movement

We now define sets of belief revision rules that operationalize the locational theories from Section 3. Each set of rules is proven sound and complete for its respective theory, relative to the fluent $\mathrm{AT}(x, l, s)$ and an action map $\mathcal{F}^m$ that translates instances of the fluent $\mathrm{MOVE}(x, l, s)$ to the corresponding update $\mathrm{AT}(x, l)^+$. All proofs can be found in the Appendix.

[4] PRS is a trademark of SRI International.

The Basic Case

The basic theory $\mathcal{T}_M^0$ is captured by the single rule:

$$\text{AT}(x, l_1),\ \text{AT}(x, l_2)^+ \quad \rightarrow \quad \text{AT}(x, l_1)^- \tag{16}$$

We use the notation $\mathcal{R}^0$ to denote the singleton set containing the rule (16). The following proposition validates the correctness of $\mathcal{R}^0$ for implementing $\mathcal{T}_M^0$.

Proposition 1 *The BR-system $\mathcal{R}^0$ is sound and complete with respect to the theory $\mathcal{T}_M^0$ for the fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

Place Abstractions

We define $\mathcal{R}^H$, the set of BR-rules for the theory of hierarchical locations $\mathcal{T}_M^H$, to be the following three rules:

$$\text{AT}(x, l_1),\ \text{LEVEL}(l_1) = \text{LEVEL}(l_2),\ \text{AT}(x, l_2)^+ \quad \rightarrow \quad \text{AT}(x, l_1)^- \tag{17}$$

$$\text{WITHIN}(l_1, l_2),\ \text{AT}(x, l_1)^+ \quad \rightarrow \quad \text{AT}(x, l_2)^+ \tag{18}$$

$$\text{WITHIN}(l_1, l_2),\ \text{AT}(x, l_2)^- \quad \rightarrow \quad \text{AT}(x, l_1)^- \tag{19}$$

We refer to any collection of axioms defined in terms of the predicates WITHIN and LEVEL that describes an individual place abstraction tree as an *abstraction theory*. The correctness of $\mathcal{R}^H$ for implementing $\mathcal{T}_M^H$ is established by the following result.

Proposition 2 *Let $\mathcal{H}$ be an abstraction theory. The BR-system $\mathcal{R}^H$ is sound and complete with respect to the theory $\mathcal{T}_M^H \cup \mathcal{H}$ for fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

Aggregate Objects

Reasoning about the movement of aggregate objects requires the rule (16) defined for the basic theory. In addition, the following rule handles the basic movement of an aggregate's subparts:

$$\text{SUBPART}(x_1, x),\ \text{AT}(x, l)^+ \quad \rightarrow \quad \text{AT}(x_1, l)^+ \tag{20}$$

Splitting of aggregates is captured by the rule:

$$\text{SUBPART}(x_1, x),\ \text{AT}(x, l),\ \text{AT}(x_1, l)^- \quad \rightarrow \quad \text{AT}(x, l)^- \tag{21}$$

Joins are somewhat more complex. For each aggregate with k subparts, a collection of k rules is required to cover all possible joins (*i.e.*, each subpart could be moved to complete the join). Below is a schema for an individual aggregate, where the predicate $\mathrm{UNA}_k(x_1, \ldots x_k)$ denotes a unique names constraint on its arguments: that is, each x_i is bound to a term that denotes a distinct object.

$$\begin{aligned} &\mathrm{SUBPART}(x_1, x)\ , \mathrm{AT}(x_1, l),\ \ldots, \\ &\quad \mathrm{SUBPART}(x_{k-1}, x),\ \mathrm{AT}(x_{k-1}, l),\ \mathrm{UNA}_k(x_1, \ldots x_k), \\ &\qquad\qquad \mathrm{SUBPART}(x_k, x),\ \mathrm{AT}(x_k, l)^+ \quad \rightarrow \quad \mathrm{AT}(x, l)^+ \end{aligned} \tag{22}$$

We define $\mathcal{R}^A$ to be the collection of rules (16) – (21) along with the instantiations of the schema (22). For applications where the co-location requirement for aggregates is not appropriate, (21) and (22) should be omitted.

We refer to any collection of axioms based on the predicate SUBPART that describes an individual aggregation hierarchy and contains appropriate unique names axioms as an *aggregation theory*. The correctness of $\mathcal{R}^A$ for implementing $\mathcal{T}^A_M$ is assured by the following result.

Proposition 3 *Let $\mathcal{A}$ be an aggregation theory. The BR-system $\mathcal{R}^A$ is sound and complete with respect to the theory $\mathcal{T}^A_M \cup \mathcal{A}$ for the fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

The Full Theory

Define the BR-system $\mathcal{R}^*$ to consist of the BR-rules (17) – (19) along with (20), (21) and the instantiated schemas of (22). The rules in $\mathcal{R}^*$ correspond to the combination of the BR-rules for place abstractions and aggregate objects, omitting rule (16) from $\mathcal{R}^A$. The omitted rule served to remove all previous AT facts for an object when a new AT fact is added to the database. The rule is too strong when multiple levels of place abstractions are used. Instead, the rule (17) provides the removal of previous AT facts at the same level of abstraction as the update.

Proposition 4 *Let $\mathcal{H}$ be an abstraction theory and $\mathcal{A}$ be an aggregation theory. The BR-system $\mathcal{R}^*$ is sound and complete with respect to the theory $\mathcal{T}^M \cup \mathcal{H} \cup \mathcal{A}$ for the fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

It is important to note that the soundness and completeness results are relative to $\{\mathrm{AT}(x, l, s)\}$, not $\{\mathrm{AT}(x, l, s), \neg\mathrm{AT}(x, l, s)\}$. In general, there will be a large number of negated AT propositions that hold for a given situation; thus, it is impractical to expect that the database contain all such negated facts. The closed-world assumption

Assume $\neg AT(A, L)$ *when* $AT(A, L)$ *is not in the database.*

can be used to obtain appropriate negative instances of the predicate AT, but only when there is a single level of abstraction.

Applying the full closed-world assumption when there are multiple levels of abstraction leads to problems. The source of the difficulties is the possibility of moving objects at high levels of abstraction without specifying specific locations at lower levels, as was the case with the example of planning to move a robot to a new room without specifying a specific coordinate within that room. After such a planned move, there would be no explicit location available for the robot at the coordinate level. The full closed-world assumption would thus allow the unwarranted conclusion of $\neg$AT(*robot*, l) for every coordinate l. For multiple levels of abstraction, the following relativized version of the assumption must be used:

Assume $\neg AT(A, L)$ *for any location* L *whose abstraction level is at or above that of a location* L' *for which* $AT(A, L')$ *is in the database.*

6 Practical Issues

To this point, we have defined a collection of formal theories of location and movement, and presented classes of belief revision rules that correctly implement those theories. However, direct implementations of these rules can be inefficient for large-scale applications. This section describes implementation techniques that enable more efficient reasoning about locations, and experimental results that validate the effectiveness of these techniques for reducing solution times in practice.

Our focus is on implementation techniques for planning systems, although many of the ideas apply more generally to other classes of problem-solvers. More specifically, we are interested in deducing situation-dependent locational facts in world states determined by a partial-order plan that contains planning variables with constraints. The computational problems of reasoning about locations are particularly severe in planning systems that generate partial-order plans (determining whether a given predicates holds in a world-state of a plan is NP-hard in the worst case [6]), yet such plans are often required for interesting problems. Furthermore, in least-commitment planners, a planning variable that represents a location may have a complex set of constraints that can make unifying location variables expensive.

Although one could define a specialized subsystem to keep track of locational information, our goal is to work within the paradigm of planning itself. In particular, we wish to have locational reasoning performed by the component of the planner that reasons about the effects of actions. Our

model assumes a set of deductive rules that encode both direct and indirect effects of actions on locations, along with a general-purpose reasoning engine for deducing information about locations and revising deductions in response to new information. Locational reasoning in this framework amounts to performing deductions and retractions for locational predicates in individual world-states of a plan, in response to changes precipitated by planning operations (such as the introduction of an action with specified effects).

SIPE–2, a hierarchical task network (HTN) style planning system, is used to illustrate the implementation techniques. However, the techniques are not specific to SIPE–2 or HTN planning; they apply to any partial-order, least-commitment planner with sufficiently powerful deductive capabilities. SIPE–2 is also used for the experimental evaluation of the methods. Several characteristics make SIPE–2 a good planning system for evaluating the locational reasoning techniques. It is one of the few AI planning systems that can solve large, realistic problems, and the techniques described here contribute significantly to that ability. SIPE–2 was developed with heuristic adequacy as a high priority, and has been employed by numerous users to solve a range of interesting problems over a twelve-year period. Example applications include planning the movement of aircraft on a carrier deck, travel planning, construction tasks, and the problem of producing products from raw materials on process lines under production and resource constraints, robot navigation and military operations. During those efforts, the system has been refined to make it highly efficient. Thus, demonstrating that a particular technique can improve plan generation time by a nontrivial amount for a given problem is significant.

Section 6.1 describes interactions between locational reasoning and planning. Section 6.2 summarizes the problems used in our experiments. Section 6.3 presents a brief overview of the SIPE–2 representation of domain knowledge along with a systematic method for translating BR-rules into this representation. Finally, Section 6.4 presents the improved implementation techniques along with the experimental results that illustrate their effectiveness.

6.1 Locational Reasoning During Planning

As noted above, we want to use the deductive machinery within the planner itself to perform locational reasoning, rather than defining a specialized locational reasoning system. One key reason for doing so is to enable modeling of complex interactions between locational and non-locational predicates, which would be difficult to support with a distinct module for locational reasoning [22].

The location of an object at a given state in an evolving plan can be impacted by various operations of the planner. One such operation is the insertion of a new goal or action into a plan. In particular, the goal or action may directly change the location of an object (*e.g.*, a MOVE

Problem	# Actions	# Abstraction Levels
Robot	12	2
Robot-with-1	12	2
Robot-with-5	12	2
Mil-Small-1+	47	1 – 2
Mil-Small-4	47	4
Mil-Big-1+	188	1 – 2
Mil-Big-4	188	4

Table 1: Charcteristics of Plans used for Experimentation

action). Furthermore, for any world state in the plan that is ordered after the new action, all location facts must be rededuced because the new action may invalidate assumptions that were made during conclusions previously drawn from locational reasoning. For example, if an action is added at the beginning of a plan to have a robot pick up a key, then all subsequent actions that move the robot should also change the location of the key. The addition of an ordering constraint between two actions requires similar recalculation of locations in later actions (*i.e.*, the ordering constraint can be viewed as resulting in the insertion of certain actions before others).

Instantiation of a planning variable can also force recalculation of locations for dependent actions. For example, suppose the plan specifies that a robot will pick up an as of yet undetermined object. Once the object is bound to, say a specific key, the new location of the key must be reflected in all actions and world-states in which the robot moved while continuing to hold this object. Furthermore, there may be additional key-specific deductions that should now be triggered.

6.2 Experimentation Domains

We have evaluated our techniques for implementing locational reasoning by generating plans for two domains: mobile robot navigation [31] and military operations [34]. This section summarizes the characteristics of these domains that are relevant to the experimental evaluation. A listing of the problems is provided in Table 1, which is further explained below.

The robot problem involves generating a plan for a robot to retrieve objects from one room and deliver them to another room. Thus, both the robot and objects that it might grasp can change their locations. Two levels of abstraction for locations are used: the planner first plans to move objects from one room to another, then refines this plan to include moves between grid locations within each room. The generated plan (referred to as *Robot*) contains 12 primitive actions, 10 of which involve changes of location (*e.g.*, robot movement, object grasping). Certain primitive actions (*e.g.*, opening a door) do not affect the location of any object.

For actions that involve movement of the robot, the planner must correctly deduce that both the robot and any objects it is carrying are at the new location (and not at the old location). These deductions must be done for both abstraction levels (rooms and grid locations). In addition, our implementation deduces for each moved object the set of objects to which it is adjacent, thus further stressing location reasoning.

In addition to *Robot*, two other plans are used for evaluation, namely *Robot-with-1* and *Robot-with-5*. They involve the same basic task but are extended so that the robot holds 1 or 5 additional objects, respectively. These objects change location whenever the robot moves, thus invoking additional locational reasoning. While the robot planning problems are not large, they are beyond the capabilities of many current AI planning systems.

The problems from the military domain involve generating employment plans for countering specific enemy courses of action, and expanding deployment plans for moving the relevant combat forces, supporting forces, and their equipment and supplies to their target destinations in time for the successful completion of a mission. Four levels of abstraction for locations are used; in increasing order, they are *place*, *sector*, *region*, and *territory*. The domain contains approximately 100 planning operators to encode relevant military actions. There are approximately 500 objects/individuals with 15-20 properties per object. About 300 of the domain objects can change location, while the remaining objects are the locations themselves. Over 2200 predicates are used to model the initial world and persistent domain constraints. Planning variables that represent locations or forces can have many possible instantiations and constraints (e.g., a location variable may have constraints on terrain type, capacity of airport, runway length, transit approval, nearness to another location, etc.).

Two basic problems were used from the military domain for the experiments, representing scenarios of differing scope (in terms of degrees of enemy threat) for the employment/deployment planning process described above. The smaller problem (referred to as *Mil-Small-1+*) results in a plan with 47 actions, 45 of which lead to changes of location for troops and equipment. Counting non-action nodes (e.g., nodes that encode plan assumptions and rationale as needed for plan modification), the final plan network contains 95 total nodes and 15 parallel branchings. The larger problem (referred to as *Mil-Big-1+*) is similar but involves the deployment of additional forces. It results in a plan containing 188 actions, 169 of which lead to changes in location. There are 381 total nodes in the final plan network and 72 parallel branchings. These plans are highly parallel, with up to 17 simultaneous activities.

The military problems can be solved using only the two lowest abstraction levels in the domain, *place* and *sector* (*e.g.*, no *region* or *territory* level operations are required). Furthermore, only limited

reasoning about the *sector* level is required (for about a third of the actions). In order to test the effects of the number of abstraction levels on our locational reasoning techniques, we considered two versions of *Mil-Small-1+* and *Mil-Big-1+*. The *Mil-Small-1+* and *Mil-Big-1+* versions use the minimal $1-2$ abstraction levels required to solve the problems while the *Mil-Small-4* and *Mil-Big-4* versions employ complete locational reasoning for all four abstraction levels. (We eliminated higher-level locational reasoning in the *Mil-Small-1+* and *Mil-Big-1+* problems by removing from the domain those predicates that specify containment relationships between location levels that were unnecessary for solving the problems.)

6.3 Overview of SIPE-2

Some knowledge of SIPE–2 is necessary to understand the implementation techniques presented in this section. SIPE–2 is a domain-independent, Hierarchical Task Network (HTN) style partial-order planning system. It provides a formalism for describing actions as *operators* and utilizes knowledge encoded in this formalism to generate plans for achieving goals. Given an arbitrary initial situation and initial task network expressed as a partially-ordered set of goals, the system either automatically or under interactive control repeatedly expands and refines a plan, generating a novel sequence of actions that responds precisely to the situation at hand.

Each plan expansion is done by applying appropriate operators, which add new actions to the plan. At first the new actions are abstract, but eventually directly executable actions are added for every plan step. After some amount of plan expansion, SIPE–2 runs planning algorithms that detect problems (e.g., two parallel actions that interfere with each other). These problems are corrected by plan refinements that do not add new actions, but may order actions, instantiate actions, or remove actions and replace them by the goal they were intended to solve.

SIPE–2's formalism supports reasoning about resources, the posting of constraints on planning variables, and the description of a deductive causal theory to represent and reason about the effects of actions in different world states. This last capability is of particular importance in this paper since our theories of locations are implemented as causal theories in the planner. The causal theories are applied every time new actions are inserted, both to the new actions and to any actions following them. In addition, the causal theories often need to be applied to a portion of the plan after plan refinements that correct detected problems.

The following subsections describe SIPE–2 operators and deductive rules, and provide a domain-independent translation of BR-rules into them.

Operator: Deploy-airforce
Purpose: (deployed airforce1 airfield2 end-time1)
Precondition: (at airforce1 location1)
(near airfield1 location1) (near seaport1 location1)
(partition-force airforce1 cargobyair1 cargobysea1)
(transit-approval airfield2)(transit-approval seaport2)
(near seaport2 airfield2)
(route-aloc airfield1 airfield2 air-loc1)
(route-sloc seaport1 seaport2 sea-loc1)
Plot:
Process
Action: split-aggregate
Effects: (not (at airforce1 location1))
(at cargobyair1 location1) (at cargobysea1 location1)
Parallel
Branch 1:
Goal: (at cargobyair1 airfield1)
Goal: (at cargobyair1 airfield2)
Branch 2:
Goal: (at cargobysea1 seaport1)
Goal: (at cargobysea1 seaport2)
Goal: (at cargobysea1 airfield2)
End Parallel
Process
Action: join-aggregate
Effects: (at airforce1 airfield2)
(not (at cargobyair1 airfield2)) (not (at cargobysea1 airfield2))
End Plot End Operator

Figure 3: SIPE–2 Operator for Deploying an Air Force

6.3.1 SIPE-2 Operators

Operators are used to represent the actions that the system can perform in a given domain, at varying levels of abstraction. The primary representational task of an operator is to describe how the world changes as a direct consequence of executing the action that it represents. Many of these effects are not explicitly listed in the operators from which the plan was produced, but are deduced during plan generation, as described in Section 6.3.2. SIPE–2 makes the assumption that the world stays the same except for the effects associated with each action in its representation of the plan.

Features of operators relevant to locational reasoning will be described while discussing the operator Deploy-airforce in Figure 3. This operator represents the action of deploying the cargo of an airforce unit to a particular airfield. It describes the movement of an aggregate object (using a technique discussed in Section 6.4.3) through several locations in a partially ordered subplan, contains effects that trigger the deductive location rules, and introduces variables for locations that will accumulate constraints.

The *purpose* of an operator denotes the goals to which the operator can be applied. The *precondition* must be true in the world state before the operator can be applied. The precondition in Figure 3 requires the initial position of the air force to be known, and introduces variables for intermediate seaports and airfields with transit approval that are on known routes to the destination. These variables contain constraints (e.g., airfield2 must have transit approval), and may therefore affect locational reasoning.

The *plot* of an operator provides a partially ordered sequence of actions and goals for performing the higher-level action represented by the operator. Applying an operator involves using its plot as a template for generating nodes to insert in the plan. In Figure 3, the plot consists of moving an aggregate object — the airforce is split into subparts which are moved in parallel by air and by sea (via intermediate locations) to the destination and finally aggregated when they have all reached the destination. Plots can contain both *goal* nodes, which require a certain predicate to be achieved, and *process* nodes, which require a specific operator or primitive action to be applied. For the example operator, the plot uses process nodes for splitting and aggregating the airforce, and uses goal nodes to achieve changes in the locations of the subparts.

6.3.2 SIPE-2 Deductive Rules

In SIPE–2, a *deductive causal theory* of the domain is used to infer context-dependent effects of actions from the effects explicitly listed with the action. The explicitly listed effects apply to every situation in which the operator might be used, while the deduced effects may be conditioned on the situation. In nontrivial applications, the deductive causal theory is used to represent many of the effects of actions, including most of the locational effects. In the example operator of Figure 3, the effects of moving a subpart to a new location will be deduced by the causal theory from the AT goal predicates. In particular, the deduced effects will include the fact that the unit is not at its previous location and its location at other abstraction levels has changed.

The deductive causal theory consists of *deductive rules*, which are similar to operators with no plot. A sample deductive rule is shown in Figure 4. Deductive rules allow the specification of domain constraints within a world state, as well as permitting access to the previous world state. Operators that encode the former are called *state rules*, while rules that allow the latter are called *causal rules*. By accessing both the current and previous world states, the system can react to *changes* between two states, thus permitting effects concerning what happened during an action to be deduced even though these effects might not be implied by the final world state.

To access both the previous and current world states, causal rules have both a *precondition* and a *condition*. The condition is matched in the world state after the action, while the precondition

is matched in the world state before the action. Thus, state rules may have a condition but not a precondition, while causal rules may have both. Two other slots are needed on a deductive rule: *trigger* and *effects*. A deductive rule is applied whenever an action being inserted in a plan has an effect that matches the predicate in the trigger. The addition of deduced effects to an action also triggers deductive rules. If the precondition and condition of a triggered rule are both satisfied, then the effects of the rule are added as deduced effects of the action.

The ability to deduce context-dependent effects of an action increases modularity and eliminates unnecessary duplication, since the deductive knowledge is consolidated in one place (as opposed to scattering this knowledge throughout many operators). Deducing effects can also reduce greatly the number of operators required to model a domain. Without this ability, there must be a distinct version of an operator for every possible set of context-dependent effects that the action modeled by the operator can have (depending on the situation in which it is applied). For example, let us consider a robot that can carry a maximum of N objects. With deduced effects, one move operator suffices and the locations of all carried objects can be deduced. Without deduced effects, $N + 1$ move operators would be needed: one for moving with no objects, one for moving with one object, etc. As more complex domains are represented, it is critical to deduce effects that are conditional on the current situation.

Most AI planners, including O-Plan2 [30], the SNLP-based planners [18] and TWEAK [7], do not deduce context-dependent effects. SIPE–2, Pednault's ADL [24], and UCPOP [25] (which is based on ADL) are exceptions. Pednault provides a good description of the complexity of deducing situation-dependent effects in a partial-order planning system [24], and concludes by stating, "Further research is needed to determine the restrictions necessary to produce reasonable levels of performance for applications of interest." Here, we present techniques that produce reasonable levels of performance for applications that involve locational reasoning.

6.3.3 Translating BR-Rules into SIPE-2 Deductive Rules

The belief revision rules from Section 4 can be translated directly into SIPE–2 causal rules. The gating conditions $P_1, \ldots, P_k$ translate to the *precondition* slot, the triggers translate to the *trigger* slot, the *add list* translates to the *effects* slot while the *delete list* is negated and added to the *effects* slot. The only exception to the above translation arises with equality predicates, which are treated as constraints in SIPE–2. Thus, equality predicates in the gating conditions are translated to constraints on members of the *arguments* slot.

As an example, the BR-rule

$$\mathrm{AT}(robot_1, region_1),\ region_1 \neq region_2,\ \mathrm{AT}(robot_1, region_2)^+ \rightarrow \mathrm{AT}(robot_1, region_1)^-$$

Causal-Rule: Move-Robot
Arguments: $robot_1$, $region_2$, $region_1$ *is not* $region_2$
Trigger: (*at* $robot_1$ $region_2$)
Precondition: (*at* $robot_1$ $region_1$)
Effects: $\neg$(*at* $robot_1$ $region_1$)

Figure 4: SIPE–2 causal rule for removing outdated AT facts

translates to the SIPE–2 causal rule shown in Figure 4.

Translated SIPE–2 rules differ from the original BR-rules in that they add explicit negated AT facts while the BR-rules simply retract AT facts that no longer hold. The SIPE–2 rules preserve the same positive AT facts as the BR-rules though, as required for the soundness and completeness results of Section 5.

This translation provides a generic, domain-independent compilation of the belief revision rules into SIPE–2 deductive rules. Specialized translations, some of which make use of domain-specific constraints, can produce operators with markedly improved computational properties. Examples of such specialized translations for the belief rules defined for reasoning about locations are discussed in the following section.

6.4 Improving Computational Efficiency

Experience in applying SIPE–2 to a number of different applications has revealed that a direct implementation of the belief revision rules for locational reasoning is inefficient. This observation prompted the development of several techniques that significantly increased the efficiency of reasoning about locations during planning. We summarize these techniques into four lessons on the topics of (1) functional facts and universal variables, (2) instantiating variables, (3) reasoning about aggregates, and (4) abstraction levels.

6.4.1 Lesson 1: Functional Predicates and Universal Variables

In the basic and aggregate theories, $\mathcal{T}_M^0$ and $\mathcal{T}_M^A$, an object can have at most one location per situation. When there are multiple abstraction levels for locations, an object can have at most one location for a given level of abstraction. The BR-rules described above maintain these constraints by explicitly removing prior AT facts for an object when a new AT fact is added. For example, the causal rule in Figure 4 ensures that a given object has at most one location at the abstraction level *region*.

The precondition in the causal rule of Figure 4 requires determination of the previous location

of an object. As noted above, determining such facts for partial-order plans is NP-hard. Thus, such rules can contribute greatly to reasoning time for a planner. Significant efficiency gains can be made by taking advantage of the fact that locations are *functional* with respect to an object, i.e., at any given time, an object can be at only one location for a given abstraction level. More generally, a predicate is said to be *functional* in some subset of its arguments if there is only one set of bindings for the remaining arguments for which the predicate is satisfied. For example, suppose the predicate *READING(gauge32 12:00 11)* represents the fact that pressure gauge 32 had a reading of 11 at noon. The *READING* predicate is functional in its first two arguments — given the gauge and the time, there is exactly one pressure reading.

Functional predicates have been used in other problem-solving systems (as described at the end of this section) for reasoning about many kinds of knowledge. Here, we describe why they are of particular importance to reasoning about locations in planning systems, and present experimental data that quantify the efficiency gains.

Analysis The use of functional predicates for reasoning about locations provides three significant advantages in planning systems: (1) deduction of the effects of movement actions is more efficient, (2) planning algorithms that use these deduced effects are more efficient, and (3) the encoding of domain knowledge is simpler. These advantages are described briefly here.

When the functionality property for locations is not exploited, the planner must explicitly record the fact that an object is no longer at its previous location after a move operation at that level of abstraction or above. When functionality is exploited, the object can only be at the location mentioned in the functional predicate. Thus while deducing effects, the prior location of a moved object does not have to be determined (as is done by the precondition of the causal rule in Figure 4). This approach saves N NP-hard queries for every move of an object, where N is the number of abstraction levels for locations.

The functionality property provides computational benefits to other portions of the planning process. Since planners reason about many possible future world states, they generally represent a new state as a set of additive effects that incrementally change the previous state. An algorithm, which we refer to as the *truth criterion*, is used to determine whether a particular predicate is true in a state given a partially ordered set of such effects. Many components of a planning system invoke the truth criterion to determine whether a predicate holds (e.g., to determine the applicability of an operator, or to identify conflicts in a plan.) The truth criterion can exploit the fact that a functional predicate completely determines the truth of any query when the functional arguments are given. For example, any query about an object's location is completely determined by the most

State-Rule: at-region
Arguments: $object_1$, $region_2$, $region_1$ *is not* $region_2$ *class universal*
Trigger: (*at* $object_1$ $region_2$)
Effects: $\neg$(*at* $object_1$ $region_1$)

Figure 5: SIPE–2 Deductive Rule With a Universal Variable

recent fact giving the object's location. Therefore, the truth criterion does not need to process the effects of actions that occur before the last state in which the object's location was changed. In contrast, the truth criterion must process the effects of all actions in the plan in order to determine the set of all objects at a particular location.

Without functional predicates, deducing that a moved object is no longer at its previous location can be problematic when different parts of the plan are at different levels of planning abstraction [31, Chapter 4]. There are various ways to work around such problems, generally involving modifications to operators to perform book-keeping for coordinating planning abstraction levels. In contrast, the problem never arises with functional predicates since they determine an object's location independent of earlier actions in the plan. Thus, functional predicates enable a simpler encoding of the domain, because methods to coordinate abstraction levels are not needed.

In summary, because functional predicates determine an object's location independent of earlier actions in the plan, knowledge is easier to encode, deducing that an object is not at its previous location is unnecessary, and all planning algorithms querying an object's location can ignore any effects that precede the assertion of a functional predicate specifying an object's location. The resultant savings are proportional to the number of relevant queries and the size of the plan. While the specifics of the implementation of the truth criterion will affect the savings accrued, the ability to ignore earlier parts of the plan can provide significant savings in most partial-order planners.

Efficiency gains in practice Functional predicates are implemented in SIPE–2 using deductive rules whose effects make use of universal variables that are constrained to be distinct from the functional value. The state rule in Figure 5 is the functional predicate version of the causal rule in Figure 4. If an action has the effect AT(*Robot* *region*1), SIPE–2 would trigger this state-rule and deduce the effect $\neg$AT(*Robot* x), where x matches all objects of class *region* different from *region1*. In nearly every application in which SIPE–2 has been used, functional predicates have proven valuable. The military domain employs the state rule from Figure 5, along with similar rules for the other location abstraction levels (for reasons described in Section 6.4.4). The robot domain has similar rules for its two levels of abstraction.

Problem	With FPs	Without FPs	Ratio
Robot	1.4	4.5	3.2
Robot-with-1	1.8	7.1	3.9
Robot-with-5	2.2	16.7	7.6
Mil-Small-1+	0.9	7.8	8.7
Mil-Big-1+	4.5	38.6	8.6

Table 2: Deduction Times (in seconds) with and without Functional Predicates (FPs)

Problem	With FPs	Without FPs	Ratio
Robot	2.6	6.0	2.3
Robot-with-1	3.2	8.4	2.8
Robot-with-5	3.6	18.3	5.1
Mil-Small-1+	12.2	22.3	1.8
Mil-Big-1+	75.5	133.1	1.8

Table 3: Plan Generation Times (in seconds) with and without Functional Predicates (FPs)

Tables 2 and 3 show the efficiency gains from exploiting the functional properties of locations in these domains.[5] Table 2 shows the time spent deducing the effects of actions during generation of plans for various problems,[6] while Table 3 shows the total time for plan generation. In both tables, the second column shows the times when functional predicates were employed while the third column shows the times when the original causal rules with preconditions were employed (as in Figure 4). The last column shows the ratio of the latter to the former.

The effects in Table 2 are striking. For the robot problems (representing a fairly simple test case), deduction takes 3.2 to 7.6 times as long when functionality is not exploited. For the military problems (representing a more sophisticated and demanding test), deduction takes almost 9 times as long in the single abstraction-level case. Table 3 shows that plan generation takes from 2 to 5 times as long in the robot problems, and almost twice as long in the military problems.

It is instructive to examine the effects of functional predicates on planning algorithms other than deduction. Because of its size and complexity, the large military problem *Mil-Big-1+* is the best example to consider. The two tables shows that, without functional predicates, deduction time increases by 34.1 seconds, while plan generation time increases by 57.6 seconds. Thus 23.5 extra seconds are consumed by non-deductive components of the planning process when functionality is

[5] All computation times in this paper are user run time in seconds for SIPE–2 running in Lucid Common Lisp. The times in Tables 2 and 3 are for a Sun Sparc 20; all other times are for a Sun Sparc 1+.

[6] These times include deduction of all effects, including non-locational facts.

not exploited (accounting for roughly 31% of the increased plan generation time).

Functional predicates in other systems Our results show that functional predicates can make locational reasoning significantly more efficient in partial-order planners. Most formally defined AI planners, including SNLP [18] and TWEAK [7], do not support functional predicates. (UCPOP [25] is an exception — its universally quantified effects can be used to represent functional predicates.) By contrast, problem-solving systems that were built for real-world problems often employ techniques for efficiently reasoning about functional predicates. PRS [21], a reactive control system, allows predicates to be declared *functional* in certain arguments with the deductive subsystem of PRS enforcing the constraint implicitly. Functionality is used frequently in PRS applications. O-Plan2 [30] encodes functional predicates using an explicit functional syntax (for example, AT($Bill$) = $Whitehouse$). The advantages of functional predicates for reasoning about mathematics in O-Plan2 have been described [5].

Universal variables Functional predicates are one use of the capabilities that universal variables provided in SIPE–2. More generally, any subset of the arguments to a predicate can be universal variables, and constraints can be added to a universally quantified variable to designate a certain set of objects that can be used in the effects of actions. The former is not needed for locational reasoning, but the latter can be useful. For example, an object is at a location at each abstraction level at any one time. By constraining the location variable to a certain class, SIPE–2 makes the variable (and thus the functionality relationship) apply to only one abstraction level. Since an object can have one location for each abstraction level, PRS and O-Plan2 can only use functional predicates when there is a single level of abstraction (unless different predicate names are used for each level, as described in Section 6.4.4).

Universal variables can also be used to simplify reasoning about sets of objects. In the robot domain, universal variables (with suitable constraints) can be used to represent the set of objects held by the robot when it moves. These objects would have their new locations deduced by a single effect based on this universal variable. This approach was used for the robot experiments listed in Table 3. Thus, the plan generation time when the robot held five objects increased only slightly over the case for one object (see lines 2 and 3 of Table 3). The same number of deduced effects are present in both problems, although there are five times as many constraints on the universal variables when the robot holds five objects. Without using universal variables to represent sets, a planner would have to deduce five location predicates when the robot held five objects (and, without functional predicates, five additional negated location predicates to remove the prior locations).

6.4.2 Lesson 2: Instantiation of Variables

Many AI planners use a *least-commitment* approach to variable binding, meaning that they delay the instantiation of a planning variable until as late as possible (i.e., until sufficient constraints accumulate to identify a single satisficing value, or until an instantiation must be made). Early instantiation can result in a poor choice that leads to low-quality solutions, exploration of an unnecessarily large search space, or failure to find any solution at all. Conversely, uninstantiated variables increase the computational complexity of any reasoning system, as they introduce the costly problems of general-purpose unification and constraint satisfaction. Uninstantiated variables are particularly expensive for planning systems [7, 31].

In the military operations domain, the cost of uninstantiated variables is high: planning variables can have dozens of constraints and hundreds of possible instantiations, particularly for locations to which objects might be moving. For example, a variable representing a destination airfield in a plan may have constraints on the capacity of the airport, runway length, transit approval, proximity to other locations, and availability of maintenance and fuel. Each of these constraints might have a large number of potentially satisficing objects, making constraint satisfaction and variable unification expensive. Uninstantiated variables can increase substantially the computation time for many parts of the planning process, including the deduction of locational information.

Two methods can be used to reduce the computational costs that stem from uninstantiated variables: (1) instantiate certain variables early, and (2) minimize reasoning with constraints that contain uninstantiated variables. Effective application of these methods requires domain-specific knowledge as to when they can be used without adversely affecting the quality of the solution found by the planner. In this section, we describe the application of these methods for locational reasoning in the military domain, and show the resulting efficiency gains.

In the military domain, certain variables can be instantiated immediately without affecting solution quality. For example, an acceptable deployment force can be chosen as soon as the nature of the enemy threat is recognized; in contrast, the transportation route should be left uninstantiated until further constraints are accumulated (e.g., the type of airplanes available might affect the choice of destination airport). Similarly, reasoning with certain constraints can sometimes be delayed until all relevant variables are instantiated. As an example, the planner sometimes needs to compute the temporal duration of a deployment to a location that is not yet determined. To do so, it is necessary to consider all possible location values when computing the numerical bounds on the duration. A move from a location with M possible instantiations to one with N possible instantiations requires M times N computations in the worst case. If the bounds on the duration are critical to generating

a good plan, then they should be computed. However, if it is known that acceptable plans can be generated without considering these bounds, then large efficiency gains can be made by delaying their computation until the target destination is determined.

To take advantage of such insights, a planning framework must support the representation of domain-specific knowledge about when to instantiate particular variables, and when to postpone certain constraint analyses. SIPE–2 permits specification of a set of variables for each operator that must be instantiated immediately after application of the operator. SIPE–2 also provides a global list of functions for processing numerical quantities that will not have their bounds computed until all relevant variables have been instantiated. Such numerical functions play an important role in real-world problems.

Experimental Results Table 4 shows the efficiency gains when these domain-specific techniques are used for the *Mil-Small-1+* and *Mil-Big-1+* problems. In particular, it illustrates the utility of (a) forcing the instantiation of variables representing deployment units and their initial locations in operators that introduce a movement of troops, and (b) blocking the computation of bounds on temporal durations for move operators when there are uninstantiated arguments. Forced instantiations were considered for two cases: all unit variables (army, navy and air units) and army units only. The tests summarized in Table 4 employed the functionality property for locations, as discussed in the previous section.

By forcing instantiation of all unit variables and their initial locations, and computing temporal durations only for fully instantiated move operators, the 47-action plan was generated in approximately one minute, with acceptable choices of objects and locations. In that case, all 47 actions were affected by the decision to instantiate unit and location variables.[7] When only army variables and their starting locations were instantiated, planning time basically quadrupled, with 28 of the 47 actions affected. When bounds on durations were computed for uninstantiated locations, planning time approximately tripled. One would expect that the combination of these two techniques would be multiplicative since bounds would be computed on fewer uninstantiated locations. The final line of Table 4 supports this hypothesis, since planning took 10-14 times longer when only army units were instantiated early and reasoning with uninstantiated location variables was permitted.

When functional representations for predicates are used (as described in Section 6.4.1), one would expect that the cost of deducing locational facts would not be greatly affected by uninstantiated variables, because the deductive process does not need to instantiate the moved object or

[7]Saying that an action was *affected by* the instantiation process means that an early instantiation was made for some variable of that action. This does not necessarily mean that the resultant action differs from the corresponding action in the plan generated without early instantiations of variables.

Variables to Instantiate	Use Uninstantiated Duration Constraints	*Mil-Small-1+ (seconds)*	*Mil-Small-4 (seconds)*	Actions Affected
army, navy, air	no	49	80	47/47
army	no	215	291	28/47
army, navy, air	yes	164	202	18/47
army	yes	711	800	10/47

Table 4: Plan Generation Times (in seconds) with and without (1) Early Instantiation of Variables and (2) Reasoning with Uninstantiated Constraints

the location, or even compute restrictions on the instantiations implied by the constraints. Similar domain-specific knowledge should improve computational properties in any least-commitment planner when solution quality permits. We also note that the usefulness of instantiating variables and avoiding analysis of constraints on uninstantiated variables is not limited to locational reasoning. This hypothesis is supported by measurements taken of the time spent doing deductions in the *Mil-Small-1+* problem with uninstantiated variables (i.e., line 2 of Table 4): of the 215 seconds for plan generation, only 6.4 seconds were spent on deductions. However, the cost of using locational information elsewhere in the planner is greatly increased by having uninstantiated variables for both objects being moved and the locations to which they are moved, as shown by the quadrupling of plan generation time compared to the improved method. For instance, unifying two locational variables takes longer, which in turn increases computation time for tasks such as testing a precondition or checking for conflicts in a plan.

Eager vs Delayed Commitment The method of instantiating variables early, which provided significant computational savings for the two military problems, contrasts with the approach of *delayed commitment* to variable bindings advocated by others. Advocates of the delayed commitment approach argue that unnecessary backtracking can be eliminated by postponing choices until as late as possible. In particular, Yang and Chan cite experimental results (based on an extension to the SNLP planner implemented by Barret and Weld [3]) showing that delayed commitment can improve planning performance substantially in certain cases [36].

A key issue for determining whether to use eager or delayed variable binding is the density of the solution space: the military problems used for the experiments reported here have dense solution spaces (although the choice of instantiations can affect solution quality). In addition, the computation of bounds on temporal durations of troop movements required for the military problem has a relatively high cost. In contrast, many of the test cases in [36] have few solutions.

It remains an open question to determine where on the spectrum most real-world problems lie, although in our experiences high-density solution spaces are more common.

6.4.3 Lesson 3: Reasoning Efficiently with Aggregates

The belief revision rules for aggregates provide the means to maintain explicit location information for objects and their subparts. Unfortunately, application of these rules (even for small numbers of subparts) can be computationally expensive. Two factors contribute to the expense. One is the number of locational facts that need to be maintained when an aggregate with many descendants is moved. This cost is magnified when multiple levels of abstraction are used. The second factor is the cost of checking the belief revision rules for splitting and joining aggregates: although the rules may apply infrequently, it can be expensive to validate their inapplicability (especially for the join rule).

The first problem can be overcome for planning systems by making slight modifications to planning operators. Note that when the subparts of an aggregate are all co-located, there is no need to store their individual locations, as this information can be derived from the position of the aggregate itself. Locational information for subparts of an aggregate need be maintained only when the subparts are not co-located. Once a locational fact for a subpart is generated, the location will be updated by the belief revision rules independently of its parent and fellow subparts. By restricting the maintenance of explicit locational facts to objects that are not co-located with their immediate parent, it becomes possible to eliminate the belief update rules (20) and (21).

Operationalizing this strategy requires only a small change to the planning operators. Let $\text{PARENT}(x_i, x)$ denote the relationship that x is a direct parent of x_i in the aggregation hierarchy. In the original scheme, operators made use of facts of the form $\text{AT}(x_i, l)$ to bind location variables. In the new approach, it is also necessary to consider the relationship defined by $\text{PARENT}(x_i, x) \wedge \text{AT}(x, l)$. For example, Figure 6 illustrates a SIPE–2 operator for moving a force from a city to an airfield. This operator employs the conjunctive precondition

```
(parent movable1 force1)
(OR (AT movable1 urban1) (AT force1 urban1))
```

which first tries to bind the current location of the object `movable1` by looking for an `AT` fact for the object and, if one does not exist, looking for an `AT` fact for its parent.

While individual movement actions will instigate the splitting of aggregates, the new approach requires a slight modification to the join rule. In particular, the schemas of (22) must be extended

Operator: move-from-urban-to-airfield
Arguments: movable1,airfield1,urban1,force1,
duration1 is (ground-duration urban1 airfield1);
Purpose: (at movable1 airfield1)
Precondition: (parent movable1 force1)
(or (at movable1 urban1) (at force1 urban1))
Plot:
Process
Action: move-ground
Arguments: movable1,urban1,airfield1;
Duration: duration1;
Effects: (at movable1 airfield1)
End Plot End Operator

Figure 6: Movement Operator that Inherits Location from Parent

to remove the location facts for subparts when a join occurs:

$$\begin{aligned} \text{SUBPART}(x_1,x),\ \text{AT}(x_1,l),\ \ldots,\ \text{SUBPART}(x_{k-1},x),\ \text{AT}(x_{k-1},l), & \qquad (23)\\ \text{UNA}_k(x_1,\ldots x_k), \text{SUBPART}(x_k,x),\ \text{AT}(x_k,l)^+ & \rightarrow \\ \text{AT}(x,l)^+,\ \text{AT}(x_1,l)^-,\ \ldots \text{AT}(x_k,l)^- & \end{aligned}$$

The second problem, the cost of checking applicability of the split and join belief revision rules, can be overcome in many cases by having planning operators assume responsibility for splits and joins, rather than leaving those responsibilities to deductive rules. This method requires a plan operator that independently moves subparts to explicitly note their dispersal and recomposition. In particular, a join involves both recording the position of the newly composed aggregate and eliminating explicit locational facts for all subparts (as was the case with (23)) above. When applicable, this scheme eliminates the need for the rule schema (22), resulting in substantial computational savings (as described below). This method applies to domains where aggregates are split into subparts to achieve some fixed set of goals and regrouped once those goals are achieved. (For example, the transporting of cargo and groups of people has this property: cargo or groups of people may be split into subparts to be moved to target locations using different methods.)

The SIPE–2 operator `Deploy-airforce` in Figure 3 uses the above technique to explicitly manage the splitting and merging of aggregates. This operator describes how to deploy the cargo for an airforce unit to a particular airfield. Deployment involves shipping a portion of cargo by sea and the remainder by air to a target destination. The actual routes are determined by the predicates `route-aloc` and `route-sloc` in the precondition. The variables `cargobysea1` and `cargobyair1`

Problem	PF	# Nodes	Aggregation		No Aggregation		Ratio
			total	*per node*	*total*	*per node*	
Agg-Mil-Small-1+	2	47	59	1.26	159	3.38	2.69
	3	58	89	1.53	324	5.59	3.64
	4	69	125	1.81	480	6.96	3.84
Agg-Mil-Small-4	2	47	86	1.83	196	4.17	2.28
	3	58	127	2.19	394	6.79	3.10
	4	69	167	2.42	571	8.28	3.42

Table 5: Plan Generation Times (in seconds) with and without the Aggregation Methods, for various Partition Factors (PFs).

constitute a partition of the airforce into subparts. The processes labeled `split-aggregate` and `join-aggregate` in the plot perform explicit maintenance of subpart/aggregate locations. In particular, the split-aggregate process removes the location for the airforce and asserts locations for the subparts. The join-aggregate process removes the locations for the subparts once they have regrouped at the target destination and concludes the fact that the aggregate is at the target destination. The original formulation of the `Deploy-airforce` operator was similar, but omitted the processes `split-aggregate` and `join-aggregate`.

By using the techniques of (a) keeping location information only for subparts not co-located with their parent force, and (b) splitting and joining aggregates in the operators instead of in the deductive rules, reasoning about aggregates in the military operations planning problem can be made much more efficient. Table 5 summarizes experimental results that support this claim. The table contrasts plan-generation times for various problems in the military domain when the original translations of the belief revision rules are used, with the case where the two aggregation techniques from this section were employed. The problems used are similar to the *Mil-Small-1+* and *Mil-Small-4* problems described above, but with a slight modification so that forces could be subdivided into varying numbers of aggregates. These problems are named *Agg-Mil-Small-1+* and *Agg-Mil-Small-4*, respectively. We refer to the value of k as the *partition factor* (PF) for a given problem. In particular, each army/navy/air force was split into k subforces, one of which in turn was partioned into k subforces, where k was set to 2, 3, or 4. Planning operators were modified slightly to force the explicit movement of all subforces. The original *Mil-Small-1+* problem had a PF of 2; the plans for PF values 3 and 4 contained 58 and 69 actions, respectively.

The data shows that without the techniques described in this section, planning time more than doubles in all cases, and almost quadruples in some cases. Furthermore, the efficiency gains increase as the PF increases. Given the marked speedup for the limited number of aggregates used in the

test cases, it is clear that the aggregation techniques will be critical in domains with deep or bushy aggregation trees.

We note that the method presented in this section moves information from deductive rules to operators. Above, it was argued that such a transfer is generally undesirable. The reasons cited were the need to duplicate the effects of the deductive rules in multiple operators and the possible need to increase the number of operators, corresponding to the application of different deductive rules in different situations. Implementing the techniques of this section requires adding the split and join effects explicitly to all operators that required the partitioning of aggregates. However, since the location rules were not highly context-dependent, the second consideration was not an issue. Given our experimental results, the trade-off between computationally efficiency and representational perspicuity seems to argue for embedding splits and joins in operators rather than defining them as separate deductive rules.

6.4.4 Lesson 4: Explicit Separation of Abstraction Levels

The theory $\mathcal{T}_M^H$ presents a formalization of locations and movement for objects whose locations are defined at multiple levels of abstraction. This theory was defined using a single AT predicate, with the function LEVEL providing the means to distinguish abstraction levels of places. As noted in Section 2.3, an equivalent formalization exists based on the use of a separate AT predicate for each level of abstraction. With this alternative approach, abstraction levels are explicitly embedded in the location predicates themselves. The explicit separation of abstraction levels eliminates the need to directly reason about the abstraction levels of objects. The explicit separation also enables simple specification of actions that are appropriate for moving objects at one abstraction level but not at another. For example, an operator that moves only from room to room could be expressed using a goal based on a predicate AT-ROOM.

The choice of single *vs* multiple predicates for representing locations at varying levels of abstractions should not significantly influence planning efficiency since the number of deductions is the same in both cases, only the method of representing and triggering the deductive rules changes. The use of separate abstraction levels could provide some computational savings, however, by providing better control of the triggering of deductions. With a single AT predicate, a movement at the lowest level of abstraction can trigger deductions at all higher levels of abstraction. However, deductions at the higher levels also deduce AT predicates that in turn may retrigger the same deductive rules at the lower level. This looping cycle must be broken; the cycle can stop when predicates that are already true are rededuced, but the computation of the last round of already-true deductions can be avoided by keeping abstraction levels separate. This cost is likely to be trivial in reasoning

systems other than planners, and should not be high in planners that use functional predicates. However, without functional predicates, this cost could be significant since queries will be made over the partially ordered actions. With separate abstraction levels, there are different deductive rules for each abstraction level (with either different predicate names or different class-membership constraints), so only the necessary deductions are triggered.

The disadvantage of multiple AT predicates is the need for a larger number of rules for updating locational information. However, the rules are simple and have a one-time development cost when the locational theory is encoded. Weighing this cost against the computational savings described above will depend on the properties of the reasoning system being used to implement the locational theory. In particular, the computational properties will be important if the reasoning system is a partial-order planning system not using functional predicates for locations that is generating large plans with many unordered actions. There may be concerns about the perspicuity of the representation as the number of predicate names grows, but the technique of using class-membership constraints eases this problem.

6.5 Summary of Implementation Techniques

Several techniques have been described for implementing reasoning about locations in planning systems. These techniques can be employed to increase efficiency significantly over direct translations of the locational theories into planning operators. Such techniques are discovered only by implementing and using a planner, and analyzing how it can be made more efficient. Here, we summarize the lessons learned.

- The ability to represent functional predicates has proven valuable in a number of reasoning systems in areas as diverse as planning, reactive systems, and mathematical modeling. In the military domain, replacing the simple rules that explicitly delete location facts with rules that implement functional predicates cut deduction time by almost one-ninth, and overall planning time almost in half.

- In many domains, early instantiation of certain variables and postponing analysis of certain constraints on uninstantiated variables can significantly improve efficiency without adversely affecting solution quality. Application of these techniques in the military domain reduced planning time by a factor of 3 for postponing constraints, by a factor of 4 for instantiating some variables, and by a factor of 10–14 when the two were combined.

- Reasoning about the location of aggregate objects can be implemented more efficiently through the use of two techniques. The first is to maintain explicit location facts only for subparts

Techniques	Combinations			
Functional Predicates	X	X		
Ignore Uninstantiated Durations	X		X	
Instantiate Unit Variables	X	X	X	X
Planning Time *(seconds)*	80	202	171	285

Table 6: Plan Generation Times for *Mil-Small-4* using Various Combinations of Techniques

Techniques	Combinations			
Aggregation	X			
Functional Predicates	X	X		
Ignore Uninstantiated Durations	X	X	X	
Instantiate Unit Variables	X	X	X	X
Planning Time *(seconds)*	86	196	407	537

Table 7: Plan Generation Times for *Agg-Mil-Small-4* using Various Combinations of Techniques

that are not co-located with their parents. The second is to embed the splitting and joining of aggregates in the planning operators rather than using a deductive locational theory. On moderate-sized problems in the military domain, these techniques made plan generation faster by a factor of 3.

- Multiple predicate names or class-membership constraints on variables can be used in lieu of explicit representations of abstraction levels for objects and can produce computational advantages when reasoning over large, partially ordered plans. Separation of abstraction levels by these means also supports the encoding of actions that apply only at one level. The cost of doing so is an increased number of deductive rules.

Importantly, the computational savings obtained by these techniques are complementary in the sense that combinations of the techniques yield greater efficiency. Table 4 shows that the combination of instantiating variables and not computing bounds on possible values for uninstantiated variables (such as temporal durations) has a multiplicative savings. Tables 6 and 7 summarize experimental results for other combinations of techniques. Table 6 shows that the savings of functional predicates and computing durations only for instantiated location variables are additive: the 80 seconds required for *Mil-Small-4* when both techniques were used increased to 171 seconds without functional predicates, to 202 seconds when computing all durations, and to 285 seconds when neither technique was used. While it seems reasonable that these techniques would be cumulative

with the aggregation methods, the situation is complicated because the latter affects the amount of deduction being done. Table 7 shows that the savings are indeed cumulative in the *Mil-Small-4* problem in that not using functional predicates doubles planning time for both the original and improved aggregate methods (see also Table 5). As well, computing all durations adds an additional 2 minutes to the planning time.

Finally, we note that while the aggregate techniques presented here are particular to reasoning about locations, functional predicates, universal variables, and domain specific knowledge for instantiating variables can improve reasoning efficiency for many classes of knowledge.

7 Conclusion

AI planning theory will have more impact when it can address nontrivial problems. A significant component of achieving that goal is to develop representations for commonly used domain knowledge that are both correct and heuristically adequate. While much progress has been made in the development of planning systems, the representational and computational issues of formalizing knowledge remain severely under-explored.

In an attempt to partially fill the void, this paper has analyzed both the theoretical and computational issues involved in representing and reasoning about locational information for objects that can move or be moved. Locational information plays an important role in a broad range of AI applications, including planning and plan execution problems.

We presented a formal locational theory along with extensions for multiple abstraction levels and aggregation. We defined a belief revision framework and a provably correct set of belief revision rules that maintains a database in accordance with each theory. The complete theory is applicable to any domain in which the locations of objects change and is compatible for use in any problem-solving framework in which locational information must be maintained, including generative planners and plan execution systems. We also considered the practical application of the locational theories to large-scale planning tasks within the SIPE–2 framework. Based on our experiences with these planning tasks, we identified several implementation techniques that can be used to improve efficiency significantly over direct implementations of the theories.

Our locative theories are not intended to cover all eventualities. Rather, they are designed to be both epistemically and heuristically adequate for solving many classes of problems of practical interest. It is our hope that this paper will encourage similar efforts that combine analyses of theory and practice for widely used classes of domain knowledge.

Acknowledgements

The research reported in this paper was supported by the ARPA/Rome Laboratory Planning Initiative under Contract F30602-90-C-0086.

Appendix: Propositions and Proofs

The appendix contains proofs for the propositions from Section 4.

Proposition 1 *The BR-system $\mathcal{R}^0$ is sound and complete with respect to the theory $\mathcal{T}_M^0$ for the fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

Proof. Let $\mathcal{D}$ be a propositional database that is closed and consistent relative to $\mathcal{T}_M^0$ for AT, and let s_0 be an introduced situation constant. By virtue of the consistency and completeness relative to $\mathcal{T}_M^0$, there is exactly one AT fact in $\mathcal{D}$ for each object in the domain. Now, consider an arbitrary object O and location L. Application of the BR-rule (16) for an update $\mathrm{AT}(O, L)^+$ preserves the number of location facts in the database: while $\mathrm{AT}(O, L)$ replaces the previous location fact for O, all other AT facts remain unchanged and no new AT facts are added. The result is a database $\mathcal{D}' = \mathcal{D} \otimes \{\mathrm{AT}(O, L)^+\}$ where again there is precisely one AT fact per object.

To establish soundness, it is necessary to show that for any AT proposition $\mathrm{AT}(O', L')$ in $\mathcal{D}'$,

$$\mathcal{T}_M^0 \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0) \models \mathrm{AT}(O', L', \mathrm{MOVE}(O, L, s_0)) \tag{24}$$

holds. As noted above, there are two types of AT facts in $\mathcal{D}'$: the added fact $\mathrm{AT}(O, L)$, and facts $\mathrm{AT}(O_1, L_1)$ that were also in $\mathcal{D}$, where $O \neq O_1$. The fact $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ is an immediate consequence of the relocation axiom (12) in $\mathcal{T}_M^0$. For a fact $\mathrm{AT}(O_1, L_1) \in \mathcal{D}'$ that is also in $\mathcal{D}$, necessarily $\mathrm{AT}(O_1, L_1, s_0) \in \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$. Combining this fluent with the inertial axiom (13) yields
$\mathrm{AT}(O_1, L_1, \mathrm{MOVE}(O, L, s_0))$. The soundness of $\mathcal{R}^0$ for $\mathcal{T}_M^0$ immediately follows.

Proof of completeness requires only that when condition (24) holds, $\mathrm{AT}(O', L') \in \mathcal{D}'$. Examination of $\mathcal{T}_M^0$ shows that the only AT fluents in situation $\mathrm{MOVE}(O, L, s_0)$ derivable from $\mathcal{T}_M^0 \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$ are $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ and $\mathrm{AT}(O_1, L_1, s_0)$ where $\mathrm{AT}(O_1, L_1) \in \mathcal{D}'$ and $O_1 \neq O$. As noted above, the corresponding situation-less projections of these AT facts are contained in $\mathcal{D}'$, thus establishing completeness.

Proposition 2 *Let $\mathcal{H}$ be an abstraction theory. The BR-system $\mathcal{R}^H$ is sound and complete with respect to the theory $\mathcal{T}_M^H \cup \mathcal{H}$ for fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

Proof. Let $\mathcal{D}$ be a propositional database that is consistent and closed with respect to $\mathcal{T}_M^H \cup \mathcal{H}$ for AT, and let s_0 be an introduced situation constant. Consider an arbitrary object O and location L.

By the consistency and closure relative to $\mathcal{T}_M^H \cup \mathcal{H}$, there is an abstraction level k for each object such that the object has an assigned location for levels k and higher in $\mathcal{D}$ but no location at levels lower. Furthermore, there is at most one AT fact in $\mathcal{D}$ for an object at any given level (by the situationalized version of axiom (5)). The rules of $\mathcal{R}^H$ preserve these properties: for the update $\mathrm{AT}(O,L)^+$, $\mathrm{AT}(O,L)$ is added to the database as are location facts for all levels above L, while all previous location facts for O are removed. No AT facts for other objects are removed and no further AT facts are added.

To establish soundness, it is necessary to show that for any AT proposition $\mathrm{AT}(O',L')$ in $\mathcal{D}' = \mathcal{D} \otimes \mathrm{AT}(O,L)^+$,

$$\mathcal{T}_M^H \cup \mathcal{H} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0) \models \mathrm{AT}(O', L', \mathrm{MOVE}(O, L, s_0)). \tag{25}$$

$\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ is an immediate consequence of the relocation axiom (12) while AT facts for O at higher levels of abstraction subsequently follow by (4). From (13), for any object O_1 distinct from O,

$$\mathrm{AT}(O_1, L_1, s_0) \supset \mathrm{AT}(O_1, L_1, \mathrm{MOVE}(O, L, s_0)).$$

By these observations, all AT facts in $\mathcal{D} \otimes \{\mathrm{AT}(O,L)^+\}$ are valid consequences of $\mathcal{T}_M^0 \cup \mathcal{H} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$ and so $\mathcal{R}^H$ is sound for $\mathcal{T}_M^H \cup \mathcal{H}$.

Completeness amounts to showing that when condition (25) holds, $\mathrm{AT}(O',L') \in \mathcal{D}'$. Examination of $\mathcal{T}_M^H$ shows that the only AT fluents for situation $\mathrm{MOVE}(O, L, s_0)$ derivable from $\mathcal{T}_M^0 \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$ are $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ and
$\mathrm{AT}(O, L', \mathrm{MOVE}(O, L, s_0))$ where L' is above L in the abstraction hierarchy, and $\mathrm{AT}(O_1, L_1, s_0)$ where $\mathrm{AT}(O_1, L_1) \in \mathcal{D}'$ and $O_1 \neq O$. As noted above, the situationless projections of these AT facts are all in $\mathcal{D}'$. Since no additional instances of AT are entailed by $\mathcal{T}_M^0 \cup \mathcal{H} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$, $\mathcal{R}^H$ is complete for $\mathcal{T}_M^H \cup \mathcal{H}$.

Proposition 3 *Let $\mathcal{A}$ be an aggregation theory. The BR-system $\mathcal{R}^A$ is sound and complete with respect to the theory $\mathcal{T}_M^H \cup \mathcal{A}$ for the fluent $AT(x,l,s)$ and action map $\mathcal{F}^m$.*

Proof. Let $\mathcal{D}$ be a propositional database that is consistent and closed relative to $\mathcal{T}_M^A \cup \mathcal{A}$ for AT, and let s_0 be an introduced situation constant. Consider an arbitrary object O and location L.

The consistency and completeness of $\mathcal{D}$ relative to $\mathcal{T}_M^A \cup \mathcal{A}$ ensure that the following conditions hold for any object C and location P:

If $\mathrm{AT}(C, P) \in \mathcal{D}$ and C_i is a descendant of C then $\mathrm{AT}(C_i, P) \in \mathcal{D}$

(ii) If $\mathrm{AT}(C_i, P) \in \mathcal{D}$ and C is an ancestor of C_i then either $\mathrm{AT}(C, P) \in \mathcal{D}$ or there is some descendant C_j of C and some location $P_j \neq P$ such that $\mathrm{AT}(C_j, P_j) \in \mathcal{D}$

(iii) Each object has at most one location.

We now show that database $\mathcal{D}' = \mathcal{D} \otimes \mathrm{AT}(O, L)^+$ has the following characteristics:

$\mathrm{AT}(O, L) \in \mathcal{D}'$

(b) if O_i is a descendant of O then $\mathrm{AT}(O_i, L) \in \mathcal{D}'$

(c) if $\mathrm{AT}(O', L) \in \mathcal{D}'$ and O' is an ancestor of O then either $\mathrm{AT}(O', L) \in \mathcal{D}'$ or there is some descendant O_j of O and some location $L_j \neq L$ such that $\mathrm{AT}(O_j, L_j) \in \mathcal{D}'$

(d) Each object has at most one location

(e) All other AT facts are as in $\mathcal{D}$.

We do so by first showing that properties (a), (b), (d) and (e) hold for the update when only rules (16) and (20) are considered; similarly, (a), (c), (d) and (e) hold for the update when only rules (16), (21) and (22) are used. We then show that rules (16) – (22) can be combined to establish all conditions (a) – (e). We note that in the case where $\mathrm{AT}(O, L) \in \mathcal{D}$, the update has no effect and the claim trivially follows. Hence, we assume below that $\mathrm{AT}(O, L) \notin \mathcal{D}$.

Consider first the rules (16) and (20). By a simple inductive proof on the number of levels of descendants of O, we show that for update $\mathrm{AT}(O, L)^+$, conditions (a), (b), (d) and (e) hold. If O has no descendants, then the only applicable BR-rule is (16), which adds $\mathrm{AT}(O, L)$ to the database and removes any previous AT fact for O. Given properties (i) – (iii) for $\mathcal{D}$, it is easy to verify that (a), (b) (d), and (e) all hold. Now suppose O has $n + 1$ levels of descendants and let A be a descendant at level $n + 1$. Then there is a single nth-level descendant B of O for which $\mathrm{SUBPART}(A, B)$ holds. In terms of the order application of br-rules, the case with $n + 1$ descendants is equivalent to the case with n descendants up to the point of the (possible) addition of $\mathrm{AT}(B, L)$ to the database (that is, the $n + 1$st descendant is irrelevant until changes are made to the nth descendant). If $\mathrm{AT}(A, L)$ is in $\mathcal{D}$, then the rule applications are still identical. If $\mathrm{AT}(A, L) \notin \mathcal{D}$, then when $\mathrm{AT}(B, L)$ is added to the database both (16) and (20) are triggered in the $n + 1$ levels case but only (16) in the n level case. The rules (16) and (20) do not interact with each other: the former removes any an AT fact for B (if there was one) while the latter adds $\mathrm{AT}(A, L)$, which in turn triggers (16) to remove any previous AT fact for A. No further rules apply. Thus, the resultant database is equivalent to

the case for n levels of descendants, except that any previous AT fact for A has been replaced by $\mathrm{AT}(A, L)$. Using the inductive hypothesis, it is straightforward to check that conditions (a), (b), (d) and (e) hold, as claimed. From a more general perspective, the rules (16) and (20) have the effect of setting the unique location for all descendants of O to L, while not affecting any other AT facts.

Now consider the rules (16), (21) and (22). We show by induction on the number of levels of ancestors for O that (a), (c), (d) and (e) hold. For the base case, suppose O has no proper ancestors. In this case, the rule (16) applies but no others, resulting in $\mathcal{D}'$ being the same as $\mathcal{D}$ except that any AT fact for O has been removed and the new fact $\mathrm{AT}(O, L)$ added. In this case, $\mathcal{D}'$ satisfies the conditions (a), (c), (d), and (e). Now suppose the result holds for n ancestors and let B be the $n+1$st ancestor. As above, if $\mathrm{AT}(A, L) \in \mathcal{D}$ for some subpart A of B (that is, $\mathrm{SUBPART}(A, B)$ holds) then by (2), $\mathrm{AT}(B, L) \in \mathcal{D}$ and the update has no effect on AT facts for B. So suppose that the update $\mathrm{AT}(O, L)^+$ results in $\mathrm{AT}(A, L)$ being added to the database for some object A that is an n-level ancestor of O. There are two cases to consider. If the update $\mathrm{AT}(A, L)^+$ triggers the corresponding instance of the join rule (22), then $\mathrm{AT}(B, L)$ will be added, which in turn triggers (16). This rule would remove any previous AT fact for B. If the split rule (21) applies, the fact $\mathrm{AT}(B, L)$ is removed but no other changes occur. In both cases, it is straightforward to check that (a), (c), (d), and (e) hold. To generalize, rules (16), (21) and (22) have the effect of reseting the location of all ancestors of O without affecting any other AT facts.

We now prove that (a) – (e) hold when all the rules in $\mathcal{T}_M^A \cup \mathcal{A}$ are employed. As noted above, for an update that assigns a location to an object, the rules (16) and (20) reset the location of that object and all descendants without impacting any other AT facts. Furthermore, the rules (16), (21) and (22) reset the locations for the ancestors of the object without affecting descendants. This latter set of rules can trigger further applications of (16) and (20), which in turn cause the unique AT facts for certain descendants to be reset in accord with (b). However, the two rule sets do not interfere with each other. As such, application of the merged set of rules produces a database where all conditions (a) – (e) hold.

To establish soundness, it is necessary to show that for any AT proposition $\mathrm{AT}(O', L')$ in $\mathcal{D}' = \mathcal{D} \otimes \mathrm{AT}(O, L)^+$,

$$\mathcal{T}_M^A \cup \mathcal{A} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0) \models \mathrm{AT}(O', L', \mathrm{MOVE}(O, L, s_0)). \tag{26}$$

Consider the various AT facts in $\mathcal{D}'$, as detailed by (a) – (e). First, consider the AT facts for O along with its ancestors and descendants. $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ is an immediate consequence of the movement axiom (12). By (b), $\mathrm{AT}(O_i, L) \in \mathcal{D}'$ for any descendant O_i; the fact $\mathrm{AT}(O_i, L, \mathrm{MOVE}(O, L, s_0)$ follows from $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ and the situationalized version of (10). By (c), for any ancestor B of O, the fact $\mathrm{AT}(B, L) \in \mathcal{D}'$ only when all subparts of B are at L; that is, when $\mathrm{AT}(B_i, L) \in \mathcal{D}'$. The fact $\mathrm{AT}(B, L, \mathrm{MOVE}(O, L, s_0))$ then follows from the situationalized version of axiom (8). From (13), for any object O_1 not related by the SUBPART relation to O,

$$\mathrm{AT}(O_1, L_1, s_0) \supset \mathrm{AT}(O_1, L_1, \mathrm{MOVE}(O, L, s_0)).$$

By these observations, all AT facts in $\mathcal{D} \otimes \{\mathrm{AT}(O, L)^+\}$ are valid consequences of $\mathcal{T}_M^0 \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$, and the soundness of $\mathcal{R}^A$ for $\mathcal{T}_M^A \cup \mathcal{A}$ follows.

Completeness reduces to showing that when condition (26) holds, $\mathrm{AT}(O', L') \in \mathcal{D}'$. Examination of $\mathcal{T}_M^A$ shows that the only AT fluents for situation $\mathrm{MOVE}(O, L, s_0)$ derivable from $\mathcal{T}_M^A \cup \mathcal{A} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$ are $\mathrm{AT}(O, L, \mathrm{MOVE}(O, L, s_0))$ and $\mathrm{AT}(O_i, L, \mathrm{MOVE}(O, L, s_0))$ where O_i is a descendant of O, $\mathrm{AT}(B, L, s_0)$ where B is an ancestor of O whose descendants are all at L, and $\mathrm{AT}(C, M)$ where $\mathrm{AT}(C, M) \in \mathcal{D}'$ and C is not related to O. As noted above, the situationless projections of these AT facts are all in $\mathcal{D}'$. Since no additional instances of AT are entailed by $\mathcal{T}_M^A \cup \mathcal{A} \cup \mathcal{S}(\mathcal{D}, \{\mathrm{AT}\}, s_0)$, it follows that $\mathcal{R}^A$ is complete for $\mathcal{T}_M^A \cup \mathcal{A}$.

Proposition 4 *Let $\mathcal{H}$ be an abstraction theory and $\mathcal{A}$ be an aggregation theory. The BR-system $\mathcal{R}^*$ is sound and complete with respect to the theory $\mathcal{T}^M \cup \mathcal{H} \cup \mathcal{A}$ for the fluent $AT(x, l, s)$ and action map $\mathcal{F}^m$.*

Proof. The proof follows from Propositions 2 and 3 along with the observation that the BR-rules in $\mathcal{R}^*$ for maintaining locations for ancestors and descendants do not interfere with the BR-rules for maintaining locations given multiple levels of abstraction.

References

[1] J. M. Agosta and D. E. Wilkins. Using SIPE-2 to plan emergency response to marine oil spills. *IEEE Expert*, 11(6):6–8, December 1996.

[2] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23, 1984.

[3] A. Barrett and D. Weld. Partial order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, Department of Computer Science and Engineering, University of Washington, 1992.

[4] S. Borgo, N. Guarino, and C. Masolo. A pointless theory of space based on strong connection and congruence. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.

[5] A. Bundy. *Annual Review of Science*. Academic Press, 1983.

[6] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[7] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[8] E. Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann, San Mateo, Ca, 1990.

[9] E. Davis. The kinematics of cutting solid objects. *Annals of Mathematics and Artificial Intelligence*, 9:253–305, 1993.

[10] J. J. Finger. *Exploiting Constraints in Design Synthesis*. PhD thesis, Stanford University, 1987.

[11] T. Garvey and K. Myers. The intelligent information manager. Final Report SRI Project 8005, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[12] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989.

[13] M. L. Ginsberg and D. E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35, 1988.

[14] M. L. Ginsberg and D. E. Smith. Reasoning about action II: the qualification problem. *Artificial Intelligence*, 35, 1988.

[15] P. J. Hayes. Naive physics I: Ontology for liquids. In *Formal Theories of the Commonsense World*, pages 71–107. Ablex Publishing Corp., Norwood, New Jersey, 1985.

[16] R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1), 1987.

[17] O. Lemon. Semantical foundations of spatial logics. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR '96)*. Morgan Kaufmann Publishers, 1996.

[18] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 1991 National Conference on Artificial Intelligence*, pages 634–639, American Association for Artificial Intelligence, Menlo Park, CA, 1991.

[19] J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977.

[20] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, 1969.

[21] K. L. Myers. *User's Guide for the Procedural Reasoning System.* Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[22] K. L. Myers. Hybrid reasoning using universal attachment. *Artificial Intelligence*, 67:329–375, 1994.

[23] K. L. Myers. A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on AI Planning Systems.* AAAI Press, 1996.

[24] E. P. D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4(4):356–372, 1988.

[25] J. S. Penberthy and D. Weld. A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*. Morgan Kaufmann, 1992.

[26] D. A. Randell and A. G. Cohn. A spatial logic based on regions and connections. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*. Morgan Kaufmann, 1992.

[27] L. Schubert, M. Papalaskaris, and J. Taugher. Accelerating deductive inference: Special methods for taxonomies, colours and times. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge.* Springer-Verlag, 1987.

[28] M. Shanahan. Default reasoning about spatial occupancy. *Artificial Intelligence*, 74(1):147–164, 1995.

[29] S. F. Smith, O. Lassila, and M. Becker. Configurable, mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, 1996.

[30] A. Tate, B. Drabble, and R. Kirby. O-Plan2: An open architecture for command, planning and control. In M. Fox and M. Zweben, editors, *Knowledge Based Scheduling*. Morgan Kaufmann, 1994.

[31] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, 1988.

[32] D. E. Wilkins. Can AI planners solve practical problems? *Computational Intelligence*, 6(4):232–246, 1990.

[33] D. E. Wilkins. *Using the SIPE-2 Planning System: A Manual for Version 4.3*. Artificial Intelligence Center, Menlo Park, CA, August 1993.

[34] D. E. Wilkins and R. V. Desimone. Applying an AI planner to military operations planning. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[35] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, 1984.

[36] Q. Yang and A. Y. M. Chan. Delaying variable binding commitments in planning. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 182–187, 1994.