Inheritance in Automated Planning

Josh D. Tenenberg*
University of Rochester
Computer Science Department
Rochester, New York 14627
josh@cs.rochester.edu

Abstract

The use of abstraction in planning is explored in order to simplify the reasoning task of an automated agent. An extension of inheritance (ISA) hierarchies from object classes to arbitrary object relations and to the actions of an agent serves to partition a planning system into distinct levels. The problem of maintaining the truth values of assertions at different levels of representation is addressed by stating the precise relationship between the different levels. This gives rise to the following correspondence between solutions: for each abstract solution, there exist *isomorphic* specializations at each lower level. Thus, the presence of abstract solutions strongly constrains the size of the original search space.

Introduction

In order to bring about desired states of the world, agents must compose actions in a purposeful fashion over a sustained duration of time. In artificial intelligence, the traditional view of planning is as a mental model of activity, where the agent's actions are represented as operations upon data structures that encode world state. Agents can thus search through representations of world states for those that satisfy their goals, rather than the costlier operation of searching within the real world itself. Unfortunately, these mental search spaces typically grow as an exponent of plan length. To combat this problem, researchers in planning have attempted to encode problems at different levels of abstraction [Alterman, 1987; Kautz, 1987; Knoblock, 1988; Nau, 1987; Sacerdoti, 1974; Yang and Nau, 1988]. Few such researchers have dealt effectively with the resulting problems of constructing and maintaining several coherent views of the world, and of specifying the relationship between these different views. This is especially difficult given that planning systems manage propositions whose truth values change over time. This paper summarizes research which formalizes one particular type of abstraction, and explores some of the properties and consequences of its use. The abstraction is a generalization of inheritance (ISA) hierarchies [Brachman, 1979; Hendrix, 1979]. The novelty of this approach emerges from two main sources: 1) the use of inheritance orthogonally throughout a planning system, from inheritance of object types, to relations between object types, to actions that effect object types; 2) the precise specification of the relation between levels, which entails the relation holding between solutions derived at these different levels.

Abstract representations typically differ from lower level representations by distinguishing between those aspects of a domain which can be considered details, and those of primary importance. In the described research, the grouping of object classes into superclasses is used as the basis for making this distinction. An abstract class is characterized by the features common to all of its members; details are taken to be those features that distinguish one subclass from another. For example, `Bottles` and `Cups` can be considered abstractly as `Containers`. Common features include the ability of both to hold liquid and to be poured from; distinguishing features include that `Bottles` have narrow necks and `Cups` have wide mouths. Object classes are used as the basis for an inheritance structuring on relations between objects, and actions applied to objects. Therefore, abstract actions effect relations between elements of abstract object classes. For example, one might abstract the predicate `InBottle(WineX, BottleY)` to `InContainer(LiquidX, ContainerY)`,

*From *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* From *Proceedings of the First International Conference on Principles of Representation and Reasoning, 1989.* This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441–5700, and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract Number F30602–85–C–0008 which supports the Northeast Artificial Intelligence Consortium (NAIC).

and abstract the operator `pourBottle(BottleY)` to `pourContainer(ContainerY)`. Abstract plans are specialized by choosing, for each abstract step, a concrete step that achieves the desired effect over a smaller class of objects.

Abstracting First-Order Theories

Predicate Mappings

An extended STRIPS-style language [Fikes and Nilsson, 1971] is used as the base representation. Actions are viewed as operations on sets of sentences denoting world states. Before considering the abstraction of actions, the abstraction of these world states will first be explored. All theorems are stated without proof; such proofs can be found in [Tenenberg, 1988].

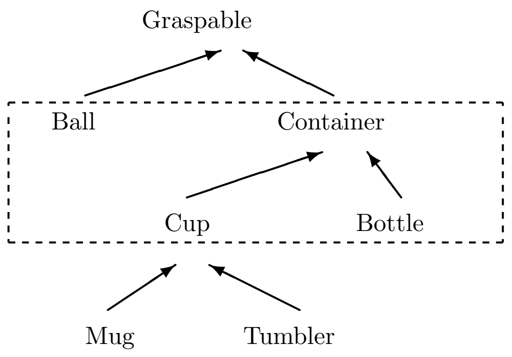

Figure 1: Inheritance Hierarchy

Figure 1 shows a fragment of a standard ISA-hierarchy (ignoring for the moment the dotted box). The nodes denote sets of objects, and the arcs denote the subset relation. This relation gives rise to inheritance, for if some property P is true of all elements of a set named by node Q, then by definition, P will be true of all elements of each descendant of Q. For example, anything true of all `Containers` is true of all `Bottles`.

In the described research, the specification of which object classes are grouped together (e.g., `Bottles` and `Cups`) is made meta-linguistically by using a *predicate-mapping function* which renames predicates between a concrete and an abstract level first-order language. Placing this function at the meta-level rather than embedding it implicitly at the object level allows comparison between different abstract class/subclass choices.

No claims are made that, for instance, `Bottles` are an inherently concrete level concept, and `Containers` are inherently abstract, merely that they are concrete and abstract with respect to one another. As pictured in Figure 1, inheritance hierarchies can contain several levels. The formalism that will be provided focuses on only two levels (termed the *concrete* and *abstract*), such as that in the dotted box, but is trivially extendible to arbitrary levels, as will subsequently be seen.

Formally, predicate mappings are functions that map predicate symbols from one first-order language to another. Given two sets of predicate symbols R and R', $f : R \overset{\text{onto}}{\mapsto} R'$ is a predicate mapping, where f is not necessarily $1-1$. This is extended to a mapping between two first-order languages, $f : L \overset{\text{onto}}{\mapsto} L'$, where the predicates are the only symbols that possibly distinguish L and L', and all non-predicate symbols map to themselves under f. Two or more concrete predicates mapping to the same abstract predicate will be called *analogues.* We interpret $f^{-1}(\psi)$ in the obvious way – as the *set* of concrete symbols that maps to ψ under f.

Model Abstraction

Consider the formal semantic models for two languages[1], L and $f(L) = L'$. Bearing in mind our intuitive notion of inheritance, when might we want to say that a model $\mathcal{M}'$ for L' is an abstraction of a model $\mathcal{M}$ for L?

A reasonable definition is when both models have the same objects in their universes, the interpretations assigned to the non-predicate symbols are identical, and all tuples of the interpretation of an abstract predicate P' in $\mathcal{M}'$ are exactly those tuples of the interpretation of all predicates P in $\mathcal{M}$ that map to P' under f. This is stated formally as follows.

Definition 1 *Let $\mathcal{M} = \langle U, G\rangle$ and $\mathcal{M}' = \langle U', G'\rangle$ be models of languages L and L' respectively, (where U is the universe and G the interpretation function) and $f : L \mapsto L'$ be a predicate mapping function. $\mathcal{M}'$ is* the abstract model of $\mathcal{M}$ through f *(that is, $\mathcal{M}'$ is $\text{AM}_f(\mathcal{M})$) if and only if by definition*

1. $U' = U$,

2. $G'(\psi') = G(\psi')$,
for all non-predicate symbols $\psi' \in L'$, and

3. $G'(R') = \bigcup_{R\in f^{-1}(R')} G(R)$,
for all symbols $R' \in \text{Pred}_{L'}$.

Note that neither exceptions nor inductions are captured by the abstraction/specialization relationship between models. If `Cup` maps through f to `Container`, then it is required, without exception, that *every* `Cup`

[1]The symbol $Pred_L$ will be used to denote the predicate symbols of language L.

```
S:
           Bottle(x) ⊃ Graspable(x)
           Cup(x) ⊃ Graspable(x)
           Bottle(x) ⊃ MadeOfGlass(x)
           Cup(x) ⊃ MadeOfCeramic(x)
           Bottle(A) ∨ Cup(A)
           ¬Bottle(B)
           ¬Cup(B)
f:
           f(Bottle) = f(Cup) = Container,
           f(Graspable) = Graspable,
           f(MadeOfGlass) = Breakable
           f(MadeOfCeramic) = Breakable
f(S):
           Container(x) ⊃ Graspable(x)
           Container(x) ⊃ Breakable(x)
           Container(A)
           ¬Container(B)
```

Figure 2: Predicate Mapping of Clause Set

is a `Container` in the corresponding models. In addition, each object taken to be a `Container` must be an element of the extension of a predicate that maps to `Container` (in the corresponding models). No inductions are made that allow objects to be `Containers` that were not one of the known specializations of `Container`.

Theory Abstraction

The predicate mapping on languages is extended to a mapping on sets of sentences in these languages (which will be taken to be in clause form[2] [Robinson, 1965]) in the obvious way. Intuitively, this amounts to rewriting the original expression and replacing all predicate symbols by their image under f. By definition, we take $f() =$, for every abstraction mapping f. An example of a clause set rewritten under a predicate mapping function is provided in Figure 2. Note that $Pred_L$, $Pred'_L$ need not be disjoint.

Having considered the abstraction relation between models, we can similarly consider the abstraction relation between axiomitizations. Given languages L and $f(L) = L'$, when do want to consider a clause set S' in L' to be an abstraction of a clause set S in L? When models for the respective clause sets are in the previously defined abstraction relationship:

Definition 2 *Let S and S' be sets of clauses in L and L', respectively, and let $f : L \mapsto L'$ be a predicate mapping function. S' is an* abstract clause set of S through f *(that is, S' is* $\textsc{acs}_f$ *of S) if and only if for every model $\mathcal{M}$ that satisfies S, $\textsc{am}_f(\mathcal{M})$ satisfies S'.*

A view of this is provided in Figure 3.

$$\begin{tikzcd}
M' \arrow[r, "\text{models}"'] & S' & \text{Abstract} \\
M \arrow[u, "AM_f"] \arrow[r, "\text{models}"] & S \arrow[u, "ACS_f"'] & \text{Concrete} \\
\text{Semantic Models} & \text{Clause Sets} &
\end{tikzcd}$$

Figure 3: Relationship of $\textsc{acs}_f$ to $\textsc{am}_f$

One might reasonably inquire, for a given abstraction mapping f and clause set S, if there exists an S' which is an abstract clause set of S through f? This question is answered affirmatively in the following section, where not only is such an S' shown to exist, but a constructive definition of the strongest such S' is provided.

Theory Mappings

Suppose we have an axiomitization in the concrete language that encodes our knowledge about the objects and relations in the world. In addition, we also have a predicate mapping function that indicates the categories of objects and relations that we consider analogous. We would like to construct a weaker axiomitization in the abstract language such that

1. it is faithful to the original axiomitization, in that no statements true in the abstract theory would be falsified at the concrete level,
2. it contains no contradictions, assuming that the concrete axiomitization is consistent,
3. it includes abstract assertions that hold of all specializations,
4. it preserves the abstract model property between the abstract and concrete theories with respect to the predicate mapping, as defined above.

[2]For simplicity, we will take clauses to be disjunctions of *distinct* literals. That is, no literal will appear more than once in any clause.

Note that f itself does not satisfy this criteria, since it results in too many clauses at the abstract level, which can result in inconsistency:

$$\texttt{Bottle(A)}, \neg\texttt{Cup(A)} \mapsto_f \\ \texttt{Container(A)}, \neg\texttt{Container(A)}.$$

Definition 4, however, satisfies this criteria. Intuitively, the abstract level never includes axioms whose specializations distinguish between the analogous predicates. Before presenting this definition, provability ($\vdash$) must first be defined.

Definition 3 *A (refutation) proof of C from clause set S is a directed binary tree $T = \langle V, E \rangle$, where labels the root, leaf nodes are labeled either by elements of S or by $\neg C$ in clause form, and there is an edge $\langle v, w \rangle \in V$ if and only if the label of v is the resolvent under full resolution [Robinson, 1965] from the label of w and some other clause. If there exists a refutation proof of C from S, then C is* provable *from S, denoted $S \vdash C$.*

Note that under this definition, $S \vdash C$ if and only if $S \models C$, since full resolution is a sound and refutation complete inference rule. In addition, the same clause can label more than one node of the graph. In particular, this will occur if the clause is used more than once in a proof.

In the following, for any clause C, $|C|$ denotes the number of literals in C, $neg(C)$ denotes the disjunction of the negative literals of C (or if there are none), and $pos(C)$ denotes the disjunction of the positive literals of C (or if there are none).

Definition 4 (Abs Clause Mapping)
$Abs_f(S) = \{C' |$

> *for every $N \in f^{-1}(neg(C'))$ having*
> *$|neg(C')|$ distinct literals,*
> *there exists $P \in f^{-1}(pos(C'))$ such that*
> *$S \vdash N \vee P.\}$*

In the degenerate case where C' has no negative literals, the membership condition for C' is not trivially satisfied. Rather, $neg(C')$ is defined as , and by definition $f^{-1}() = $. Therefore, if C' has no negative literals then there exists a unique $N \in f^{-1}()$ having no literals, namely itself, and it is required that there exist a $P \in f^{-1}(pos(C'))$ such that $S \vdash \vee P$. Put simply, if C' has no negative literals, there must exist a $P \in f^{-1}(pos(C'))$ such that $S \vdash P$. For example,

$$\texttt{Bottle(A)} \vee \texttt{Cup(A)} \mapsto_f \texttt{Container(A)}.$$

The case where there are no positive literals in C' is similar; for every $N \in f^{-1}(neg(C'))$ having $|neg(C')|$ literals, it must be that $S \vdash N$. For example,

$$\neg\texttt{Bottle(B)}, \neg\texttt{Cup(B)} \mapsto_f \neg\texttt{Container(B)}.$$

If C' has neither negative nor positive literals, that is, $C' =$, then $S \vdash$. Therefore, if S is inconsistent, so is $Abs_f(S)$. An example of a clause set mapping under Abs_f is the deductive closure of $f(S)$ from the simple example of Figure 2.

The following theorems and corollaries all hold, proofs of which can be found in [Tenenberg, 1988].

Theorem 1 *S' is* ACS$_f$ *of S if and only if $S' \subseteq Abs_f(S)$.*

Corollary 2 *If $S' \subseteq Abs_f(S)$ then S' is inconsistent only if S is inconsistent.*

Corollary 3 *If $Abs_f(S)$ is consistent then S is consistent.*

Corollary 4 *$DC(Abs_f(S)) = Abs_f(S)$, where DC denotes deductive closure.*

Corollary 5 *$Abs_f(S)$ is finite if and only if S is the empty clause set.*

Corollary 6 *There is no effective procedure for constructing $Abs_f(S)$, for arbitrary S and f.*

Since every theory S' which is an abstract clause set of S must be a subset of Abs_f(S), Theorem 1 states that Abs_f is as strong as possible. That is, one could not augment $Abs_f(S)$ by adding non-theorems, such that the resulting clause set is ACS$_f$ of S. As a consequence, however, Abs_f is not practical to use, due to the infinite size of the abstract clause sets[3] and the non-existence of an effective procedure for computing it. However, by Theorem 1, since each subset of Abs_f is also ACS$_f$ of S, one can consider subsets of Abs_f that do not have its computational problems but that satisfy the principles given at the beginning of Section 2.4. We demonstrate one such subset below, which additionally has a useful proof-theoretic property.

Definition 5 (MembAbs Clause Mapping)
$MembAbs_f(S) = \{C' |$

> *for every $N \in f^{-1}(neg(C'))$ having*
> *$|neg(C')|$ distinct literals,*
> *there exists $P \in f^{-1}(pos(C'))$ such that*
> *$N \vee P \in S\}$*

The only difference between $MembAbs_f$ and Abs_f is that specialized clauses must be *elements of* the original clause set in $MembAbs_f$, whereas they can be *derivable from* the original clause set in Abs_f. This results in the following Lemma:

Lemma 7 *If S is a set of atoms, then $MembAbs_f(S) = f(S)$.*

[3]Although it is my (as yet unproven) belief that for finite S, there exists finite S' having the same deductive closure as $Abs_f(S)$.

$MembAbs_f$ is computable in the worst case in time quadratic in the size of S, and by Lemma 7, linear in the best case. Note that Corollaries 3, 4, 5, and 6 are not true of $MembAbs_f$.

In addition to the model-theoretic properties associated with $MembAbs_f$ stated above, there is the following important proof-theoretic property. The case where the original clause set, S, is Horn[4] will be stated since it will be required in Section 4, with the more general case of unrestricted clause sets provided in [Tenenberg, 1988].

Theorem 8 *Let S be a Horn clause set in language L, $f : L \mapsto L'$ be a predicate mapping and G' be an atom in L'. If T' is a proof of G' from $MembAbs_f(S)$, then there exists a proof T of G from S such that T is isomorphic to T', and f is a renaming of labels between the nodes in T and their isomorphic images in T'.*

This theorem demonstrates that finding an abstract solution (proof) provides a strong constraint on finding a solution to the original problem, since the only concrete level proofs that need to be pursued are those that exhibit the isomorphism. That is, given an abstract proof, the concrete level search space has been reduced from the set of all proofs to the set of isomorphisms of T'.

STRIPS

Thus far, abstraction mappings have been explored with respect to first-order theories. This will be extended to planning systems using a formalization of the STRIPS language, which represents actions as operations on first-order knowledge bases. STRIPS has been used extensively in planning systems [Chapman, 1985; Sacerdoti, 1977; Waldinger, 1977; Wilkins, 1983]. One of the few attempts at a rigorous formalization is provided by [Lifschitz, 1986], from which much of the formalization to be presented has been adapted[5] Although most planning researchers are conversant in STRIPS, I provide a detailed description, since there are several variants described in the literature, with some non-trivial differences between them, and because a crisp formalization allows for a precise statement of the main definitions and theorems. For those familiar with STRIPS, the balance of this section can be skipped, but the following aspects of my definition should be noted:

1. the inferring of *secondary effects* is permitted through the use of a set of *static axioms*, which are elements of each world representation (*situation*), and hence never deleted,
2. there is a suitable description of the set of initial situations that characterize the problem space,
3. all elements of the precondition, add, and delete lists are atomic.

In the original system [Fikes and Nilsson, 1971], STRIPS is both a planning language *and* a stack-based control structure for searching the plan space. In the following, all references to STRIPS refer only to the language component. In STRIPS, the world is viewed as being in a particular state at each moment, and only through the actions of the agent can the world change state. The state of the world is represented by a set of sentences in a first-order language, which will be called a *situation*, and the actions of the agent are represented by *operators*. Associated with each operator is a set of preconditions specifying the conditions that must be satisfied by a situation in order for the operator to apply. The effects of each operator are represented by the deletion and addition of sentences to the situations in which they apply. STRIPS can thus be viewed as a knowledge base (KB) manager, where the effects of an action are realized as operations on the KB. In particular, suppose one has situation S_0 and action o, with the associated sentences A_o to be added, and D_o to be deleted. The new situation resulting from applying o to S_0 is $(S_0 \setminus D_o) \cup A_o$, that is, the old situation minus the deleted sentences, plus the added sentences. By virtue of this syntactic operation, those sentences not deleted continue to hold in the new situation, (the so-called *strips assumption* [Waldinger, 1977]), without the necessity of a separate axiom and inference step for each such sentence, as is typically required in *situation calculus* approaches [McCarthy and Hayes, 1969]. The STRIPS assumption, then, provides a simple approach to handling the *frame problem* [McCarthy and Hayes, 1969].

Since some propositions about the world might inferentially depend upon others, this affects the form in which the world state is encoded so as to facilitate the change in truth value that might occur as the result of applying an action. For instance, suppose that in a typical blocks world scene, there is a stack of blocks. If the top block is removed from the stack, then the fact

[4]A *Horn clause* is a clause in which at most one literal is unnegated. A *Horn clause set* is a clause set in which each clause is Horn.

[5]The main differences are that I have extended Lifschitz's formalization in 3 significant ways. First, I consider a STRIPS system as defining an entire problem space, not just a single problem instance. Second, I provide a more specific semantics in [Tenenberg, 1988] that associates a set of models with each situation, which provides an interpretation of all of symbols of the logic, not just the truth values of the sentences. And third, I provide a syntactic condition that is necessary and sufficient for the existence of a `Strips`-model for a given STRIPS system.

that it is no longer above the remaining blocks in the stack must be reflected in the axioms used to represent the world. If `Above` is encoded explicitly in the knowledge base, then the operator associated with removing a block must specify the deletion of all of the appropriate `Above` relationships from the KB. Alternatively, one could store only the `On` relationships in the KB, along with an axiom of the form:

$$\forall \texttt{x,y.} \quad \texttt{On(x,y)} \supset \texttt{Above(x,y)}$$

which allows one to infer all of the `Above` relationships. In this way, only the single `On` relation between the top block and the one just below it need be changed as a result of the remove action.

This approach will be generalized to all of the object relationships encoded in the system, and will be reflected in the description of the syntax of STRIPS systems. A predicate will be either *primary*, meaning that it is not dependent upon or derivable from any others, or *secondary*, meaning that it is derivable from the primary relations.

A STRIPS system, Σ is a quintuple (L, E, O, K, σ), where L is a first-order language, E is a subset of the predicates of L (E being the primary predicates), O is a set of operator schemata, K is a non-empty set of clauses in language L (the static axioms), and σ is a set whose elements are sets of ground atoms (the problem space). There are additional constraints on K and σ discussed below.

The set K is taken to be Horn in the balance of this paper. Theorem 9 relies upon the absence of disjunction at the concrete level. Set K includes those clauses which hold in every situation. It is required that every $P \in E$ can occur only in negated literals of clauses in K, that is, as the antecedent of a Horn clause. In essence, this ensures that primary atoms are never derivable in any situation except trivially through membership.

The set E_ϕ is defined as the set of ground atoms formed from predicates in E:

$$E_\phi = \{P(x_1, \ldots, x_n) | P \in E, \\ \text{and } x_1, \ldots, x_n \text{ are ground terms}\}.$$

Each element of σ is composed from atoms in E_ϕ.

An *operator schema name* is an expression which has the form

$$\mathit{operator}(\mathit{arg}_1, \mathit{arg}_2, \ldots, \mathit{arg}_n)$$

where *operator* is a symbol not in L, and the arg_i are all variables of L. Associated with each operator schema name o is an operator schema *description* (P_o, D_o, A_o), referred to as the preconditions, deletes, and adds, which are sets of atoms. The atoms in D_o and A_o must be atoms from E, which insures that only atomic expressions formed from primary predicates are explicitly managed by the KB operations. The only variables occurring in (P_o, D_o, A_o) are the arg_i of the associated operator schema. An operator schema name together with its description will be called an operator schema. If ϕ is a substitution of terms for variables, and o is an operator schema, then $o\phi$ is understood in the standard way as the substitution of each variable in o by its pairing under ϕ. We take the set O_ϕ to be the set of operator schemata in O under all *ground* substitutions:

$$O_\phi = \{o\phi | o \in O \text{ and } \phi \text{ is a ground substitution}\}.$$

Operator schemata will never occur in expressions partially instantiated. An *operator* is an operator schema fully instantiated by ground terms. Likewise for operator description and operator name. When the context is clear, the term *operator* will often informally be used to refer to just the operator schema name under a particular substitution.

A *dynamic situation* is a subset of E_ϕ, denoting those assertions of a situation that might change truth value as a result of applying an action. A *problem* is a pair $\rho = (T_0, G)$, where T_0 is a dynamic situation called the *initial* dynamic situation, and G, the *goal*, is a sentence of L. Unless specified otherwise, G is taken to be a set (conjunction) of atoms.

A *plan* is a finite sequence of operator names (fully instantiated), with the *length* of the plan being the number of operator names in the sequence. The length n plan $\omega = \langle o_1, \ldots, o_n \rangle$ defines a sequence of dynamic situations $T_0, T_1, \ldots, T_n$, where T_0 is the initial dynamic situation, and

$$T_i = (T_{i-1} \setminus D_{o_i}) \cup A_{o_i}, 1 \leq i \leq n.$$

That is, T_i is the dynamic situation T_{i-1} without the deletes of action o_i but including the adds of action o_i. This in turn defines a sequence of situations $S_0, S_1, \ldots, S_n$ composed of the dynamic situations unioned with the static axioms, i.e.,

$$S_i = T_i \cup K, 0 \leq i \leq n.$$

An operator o is *applicable* in situation S if $S \vdash P_o$. The plan ω is *accepted in* Σ *with respect to* S_0, under the condition that

$$S_i \vdash P_{a_{i+1}}, 1 \leq i < n.$$

where S_i is defined as above. That is, a plan is accepted if each operator is applicable from the situation in which it is applied. S_n, the final situation achieved by executing plan ω from initial situation S_0, is called the *result* and is denoted $Result(\omega, S_0)$. The null plan, denoted $\langle\rangle$, will be considered a plan of length zero accepted in Σ with respect to every situation, where $Result(\langle\rangle, S_0) = S_0$. A situation S_i is *accessible* from S_0 if and only if there exists a plan ω that is accepted in Σ with respect to S_0 and $Result(\omega, S_0) = S_i$. The initial situation is accessible to itself by the existence of the null plan. Plan ω *solves* problem $\rho = (T_0, G)$ if

$$Result(\omega, T_0 \cup K) \vdash G.$$

A problem is *solvable* if there exists a plan that solves it.

The dynamic situations are considered changing parameters of the system. There are only some dynamic situations that we will want to consider as possible values for the changing parameters. These are characterized by the set σ of the quintuple defining Σ. We take σ_K to be the set of elements of σ, each unioned with K:

$$\sigma_K = \{T \cup K | T \in \sigma\}.$$

It is required of all STRIPS systems that σ_K be closed under operator application. That is, for all $S \in \sigma_K$ and all $o \in O_\phi$ applicable in S, $Result(\langle o\rangle, S) \in \sigma_K$. Under this definition, σ_K defines the problem space. That is, all initial situations are drawn from σ_K, and all accessible situations from every $S \in \sigma_K$ are themselves elements of σ_K. For the balance of this paper, the Σ associated with the concrete level a STRIPS system will be taken as defined in Figure 4.

Definition 6 *A* STRIPS *system* $\Sigma=(L, E, O, K, \sigma)$ *is* consistent *if and only if for every situation $S \in \sigma_K$, every situation accessible from S is (logically) consistent.*

A simple example of a STRIPS system for the standard blocks world is given in Figure 4. Note that the static axioms serve primarily as *integrity constraints* [Reiter, 1978]. Since any goal is trivially derivable in an inconsistent situation, one would prefer never to generate such situations through the use of applicable operators. This is particularly important with abstraction, since the abstract solution will be used in constraining the concrete level search. Thus, being misled by inconsistencies to believe that the abstract goal is solved might be excessively costly. There is additionally a semantic motivation for maintaining the consistency of STRIPS systems as one ascends the abstraction hierarchy. A semantics for STRIPS is provided in [Tenenberg, 1988], which demonstrates the existence of a STRIPS-model if and only if the corresponding STRIPS system is consistent.

L :

Constants: $\mathtt{C}_1, \mathtt{C}_2, \ldots$
Variables: `w, x, y, z`
Predicates: `On`, `Clear`, $\neq$

E :

`On`, `Clear`

K :

$\neg$`On(TABLE,x)`
$\neg$`On(x,x)`
`On(x,y)` $\supset \neg$`Clear(y)`
`On(x,y)` $\wedge\, y \neq z \supset \neg$`On(x,z)`
`On(x,y)` $\wedge\, y \neq$ `TABLE` $\wedge\, x \neq z \supset \neg$`On(z,y)`
$\{\mathtt{C}_i \neq \mathtt{C}_j | i, j \in \mathbf{N} \text{ and } i \neq j\}$

O :

```
stack(x,y)
  P: On(x,TABLE), Clear(y), Clear(x),
     x ≠ y
  D: On(x,TABLE), Clear(y)
  A: On(x,y)

unstack(x,y)
  P: On(x,y), Clear(x)
  D: On(x,y)
  A: On(x,TABLE), Clear(y)
```

σ :

$\{\mathtt{T} | \mathtt{K} \cup \mathtt{T} \text{ is consistent and } \mathtt{T} \subseteq E_\phi\}$

Figure 4: Example STRIPS System

Abstract STRIPS Systems

The intent of abstracting STRIPS systems is similar to that of abstracting first-order theories—to enable the agent to perform search within a smaller space. This requires abstracting all elements of the quintuple that define the STRIPS system, the most difficult of which is the operator set. The intuition associated with abstracting the operators is that actions performed on analogous objects for analogous purposes can be considered the same at the abstract level. These actions will be determined to be analogous based upon a mapping and comparison of the preconditions, deletes, and adds of the actions under the given predicate mapping function.

The STRIPS system resulting from abstracting all operators will have the *downward-solution property.* That is, for every abstract plan ω' solving abstract goal G', there exists some specialization ω of ω', and G of G' such that ω solves G.

Let $\Sigma = (L, E, O, K, \sigma)$ be a STRIPS system and $f : L \mapsto L'$ be a predicate mapping function. The abstract system $\Sigma' = (L', E', O', K', \sigma')$ associated with Σ is itself a STRIPS system, where $K' = MembAbs_f(K)$,

$$\sigma' = \{f(T) | T \in \sigma\},$$

$$E' = \{f(P) | P \in E\},$$

under the restriction that all predicates mapping to the same symbol are either all primary, or all secondary.

The predicate mapping function, f, can be extended to a mapping on operator schemata by assigning to each schema name an abstract name, and mapping each atom of the description to its image under f. For example, the concrete operator `microWave` maps to the abstract operator `cook`, where f maps the predicates in the corresponding positions on the precondition, add, and delete lists:

```
microWave(x,y,z)
  P: In(x,y), InMicroWave(y,z), Raw(x),
  D: Raw(x),
  A: MicroWaved(x)

cook(x,y,z)
  P: In(x,y), InCooker(y,z), Raw(x),
  D: Raw(x),
  A: Cooked(x)
```

Given operator schema $o =$ *opname:* (P, D, A), define

$$f(o) = f(\mathit{opname}) : (f(P), f(D), f(A)).$$

Likewise for instantiated operator schemata. In addition, define

$$f(O) = \{f(o) | o \in O\}.$$

This mapping is extended to plans, so that if $\omega = \langle o_1, \ldots, o_n \rangle$ then $f(\omega) = \langle f(o_1), \ldots, f(o_n) \rangle$.

Only a subset of $f(O)$ will be used in the abstract level system—in particular, those concrete operators that do not distinguish between the analogous predicates. This is similar to how previously, given first-order theory S, only the subset $MembAbs_f(S)$ of $f(S)$ was used for performing abstract level inference. For instance, in order to map `PourFromBottle` to `PourFromContainer` it is required that `PourFromCup` also exist at the concrete level, assuming that cups and bottles are the only predicates that map to container. Abstracting an operator o from Σ to Σ' requires that there exists in Σ a complete set of analogues of o with respect to the objects over which o is applied.

The set of abstract level operator schemata, $O' \in \Sigma'$, are defined below. It is assumed that all corresponding variables in analogous operator schemata in O have been named identically.

Definition 7 (Operator Abstraction)
$O' = \{o' | o' \in f(O)$ *and there exists* $Q \subseteq O$ *such that*

1. *for each* $o \in Q$, $f(o) = o'$
2. *for each* $P \in f^{-1}(P_{o'})$ *such that each element of* P *is atomic, there exists* $o \in Q$ *such that* $P = P_o$,
3. *for each* $o \in Q$, *for each ground substitution* ϕ, *for each element* $d_i \in D_{o\phi}$, *for each atomic* $d_j \in f^{-1}(f(d_i))$, *either* $d_j \in D_{o\phi}$ *or* $K \cup P_{o\phi} \vdash \neg d_j\}$.

Informally, for each $o' \in O'$ there exists a subset Q of the concrete operator schemata, all of which are analogous and map to o' (under the uniform variable assignments). In addition each specialization of the precondition list of o' is the precondition list for some element of Q. That is, Q is a *complete* set of analogues. And for each element of Q, for every deleted atom, every analogous atom either co-occurs on the delete list, or its negation is derivable whenever the preconditions of this operator hold. This last requirement is imposed in order to insure a correspondence between the abstract and concrete levels, since otherwise, if two analogous propositions hold in some situation at the concrete level and only one is deleted by an operator o, then in the corresponding abstract situation, the abstraction of both propositions will be deleted by the abstraction of o.

A concrete level problem is abstracted as $\rho' = (T', G') = (f(T), f(G))$. Since the abstraction of a STRIPS system is itself a STRIPS system, definitions of legal situation, plan, consistency, etc., hold for the abstract level. Figure 5 illustrates how several operators in a concrete level kitchen domain can map to the same operator at the abstract level (assuming the appropriate predicate mapping function). Note that several different predicates have been abstracted in this example: the cutting surface, the tool, the type of food, and the type of cutting.

Downward-Solution Property

Given the abstract STRIPS system defined above, the abstract level will correspond to the concrete level in a precise way. Each abstract plan is specializable by an isomorphic concrete plan such that each intermediate abstract situation is $MembAbs_f$ of the corresponding concrete situation. In addition, each abstract level precondition proof is specializable by an isomorphic concrete level precondition proof for the corresponding operator. Thus, for every abstract level inference, there exists a set of isomorphic images that defines the space of specializations.

Theorem 9 *Let* $\Sigma' = (L', E', O', K', \sigma')$ *be an abstraction through* f *of the consistent* STRIPS *system* $\Sigma = (L, E, O, K, \sigma)$, *and let* $\omega' = \langle o'_1, \ldots, o'_n \rangle$ *solve* $\rho = (T'_0, G')$ *for some* $T'_0 \in \sigma'$. *For every* $T_0 \in \sigma$ *such that*

Concrete:
```
  sliceMushroomWithSlicer(x,y,z)
    P: On(x,z), Countertop(z), Slicer(y),
       Mushroom(x), Held(y), Whole(x)
    D: Whole(x),
    A: Sliced(x)

  sliceMushroomWithChefKnife(x,y,z)
    P: On(x,z), CuttingBoard(z), ChefKnife(y),
       Mushroom(x), Held(y), Whole(x)
    D: Whole(x),
    A: Sliced(x)

       ⋮

  diceHamWithCleaver(x,y,z)
    P: On(x,z), CuttingBoard(z), Cleaver(y),
       Ham(x), Held(y), Whole(x)
    D: Whole(x),
    A: Diced(x)
```

Abstract:
```
  makeInPieces(x,y,z)
    P: On(x,z), CuttingSurface(z), Knife(y),
       Food(x), Held(y), Whole(x)
    D: Whole(x),
    A: InPieces(x)
```

Figure 5: Operator Abstraction

$f(T_0) = T'_0$, there exists a plan $\omega = \langle o_1, \ldots, o_n \rangle$ accepted at the concrete level such that

1. $f(\omega) = \omega'$,
2. $MembAbs_f(\text{Result}(\langle o_1, \ldots, o_m \rangle, T_0 \cup K)) = \text{Result}(\langle o'_1, \ldots, o'_m \rangle, T'_0 \cup K'), 1 \leq m \leq n$,
3. *There exists $G \in f^{-1}(G')$ such that*

   $\text{Result}(\omega, T_0 \cup K) \vdash G$,
4. *For each proof W' of preconditions $P_{o'_{m+1}}$ from*

   $\text{Result}(\langle o'_1, \ldots, o'_m \rangle, T'_0 \cup K')$,

   there exists proof W of preconditions $P_{o_{m+1}}$ from

   $\text{Result}(\langle o_1, \ldots, o_m \rangle, T_0 \cup K)$

   such that W and W' are related as in Theorem 8, $0 \leq m < n$.

Figure 6 provides an example of the above theorem under a standard interpretation of the predicates and operators and a suitable predicate mapping.

Concrete	**Abstract**
Goal:	*Goal:*
`Diced(A)`	`InPieces(A)`
`Baked(A)`	`Cooked(A)`
Plan:	*Plan:*
`openFridge(F)`	`openFoodSource(F)`
`getFromIn(A,F)`	`getFromIn(A,F)`
`placeOn(A,C)`	`placeOn(A,C)`
`getFronOn(K,C)`	`getFromOn(K,C)`
`sliceHamWKnife(A,K,C)`	`makeInPieces(A,K,C)`
`placeOn(K,C)`	`placeOn(K,C)`
`getFromSurface(A,C)`	`getFromSurface(A,C)`
`putInPot(A,P)`	`putInVessel(A,P)`
`putInMW(P,M)`	`putInCooker(P,M)`
`microWave(A,P,M)`	`cook(A,P,M)`

Figure 6: Plan Abstraction

Note that although the stated theorems are between only two levels, the results extend to additional levels, since the abstract level is itself a STRIPS systems that can be abstracted in precisely the same fashion.

Rather than searching in the original problem space, then, a problem can be abstracted, search can be pursued in the abstract space, and an abstract solution, if found, can be specialized under the constraints of the above correspondence.

Note that the converse of the downward solution property does not hold – there might exist problems that are solvable at the concrete level for which there exists no abstract solution. In particular, these will be problems that rely upon distinguishing features of analogous concrete level classes, for example, a problem requiring the larger size of a conventional oven as opposed to a microwave.

There is, therefore, a delicate balance between the generality of the abstract level and its usefulness. One must trade-off the potential gains of search within increasingly simple spaces against the fewer problems that are solvable within these spaces. I leave it for future work to develop strategies that can *choose* predicate mappings for which this trade-off is optimized.

Related Research

There have been several recent suggestions focused upon abstracting actions, such as that of [Nau, 1987; Anderson and Farley, 1988; Alterman, 1987; Kautz, 1987]. All of these researchers propose operators that inherit preconditions and effects, and yet none consider abstract operators to actually *be* operations on an abstract representation of the domain. Therefore, none of their proposals enable the composition of operators, and hence the completion of an entire plan, at the abstract level. Without the ability to compose abstract operators, the meaning of such abstract operators is called into question, as well as the nature of their contribution to the eventual construction of a concrete level plan.

There are several potential benefit of the approach detailed within this paper that enables abstract level plans to be constructed. First, abstract plans allow a form of least commitment, since they do not require specialization of the operations until after the entire abstract level plan has been obtained. For instance, this might allow the specification of which kind of container used in a plan to be pushed closer toward execution time This is reminiscent of Stefik's [1981] constraint posting. Another advantage is that disjunctive information which might stymie a low-level planner need not prevent the formation of an abstract plan. For instance, if object A is either a bottle or a cup, planners that require disambiguation before choosing an action will not be able to proceed, while the planner described above could continue construction of an abstract plan.

Future Directions

One type of abstraction that has been considered in other work involves the use of operator composition, typified by MACROPS [Fikes *et al.*, 1972]. In this extension of STRIPS, sequences of operators were parametized by replacing constants by variables, and then stored for possible reuse to solve a subsequent problem. The drawback with this approach is that as the number of stored plans increases, the likelihood that a particular one will apply to a given problem decreases. In addition, the cost of searching the plan library for an appropriate plan grows prohibitively. It was clear to these researchers that there was not sufficient indexical structure to the library:

> The source of this difficulty is that the level of detail in a MACROP description does not match the level of planning at which the MACROP is to be used. ... It may be necessary to consider more sophisticated abstraction schemes involving alteration of predicate meanings or creation of new predicates.

This suggests that it is difficult to exploit the abstraction inherent in composing operators if one has no capability for specifying inheritance types of relationships between the operators. We believe that the predicate abstraction given here provides just such a sophisticated abstraction mechanism.

For instance, in the kitchen-world example, the saving of the specialized plan that was developed might be of little utility, since the same tools and resources might not be available in a subsequent similar problem. In addition, there is a non-trivial overhead cost associated with determining if a saved plan is directly applicable. On the other hand, the abstract plan may be of sufficient generality so that its frequent use offsets the cost associated with saving it.

Abstraction also appears useful for plan adaptation or repair. That is, if one has a particular plan that is being executed, errors or unforeseen events might occur at runtime–the chef's knife is not available, for instance, or the microwave oven is not functioning. In this case, then, the hierarchical structure of the plan provides a basis for quickly repairing the problem online. This type of computation can be found in [Alterman, 1987] who uses a script-based [Schank and Abelson, 1977] problem representation. When a plan step fails due to the inability to satisfy the precondition of an operator, an analogous operator is attempted that has the same abstract effect, but that has a different specialized precondition. This would amount to using a cleaver as a substitute for the missing chef knife.

Conclusion

This research has centered around the principle that in any reasoning system using multiple levels of representation, there should be a precise correspondence between the different levels. In addition, the manner in which inference at one level can guide inference at another level should be made explicit. Such a specification for planning systems has been hindered by the difficulty of managing propositions at two different levels each of whose truth values might change over time. This paper addresses these problems by extending the notion of inheritance from object classes, to relations on object classes, to actions over object classes. A model-theoretic semantics is presented for abstracting first-order theories that generalizes ISA-hierarchies, through the use of a *predicate mapping* function. This mapping is then extended to STRIPS systems. A powerful inferential relationship between levels is shown to hold – abstract plan solutions to problems can be specialized by choosing a specialization of each abstract plan step, and thus concrete solutions that are not isomorphic in this fashion need never be explored.

Acknowledgements

I am indebted to Dana Ballard for his encouragement and critical insights in his role as my dissertation advisor. In addition, I would like to thank Leo Hartman, Jay Weber, James Allen and Lenhart Schubert for the numerous readings of drafts and fruitful discussions.

References

[Alterman, 1987] Alterman, Richard 1987. Issues in adaptive planning. Technical Report 304, University of California at Berkeley.

[Anderson and Farley, 1988] Anderson, J. and Farley, A. 1988. Plan abstraction based on operator generalization. In *Proceedings of the 7th AAAI.*

[Brachman, 1979] Brachman, Ron 1979. On the epistemological status of semantic networks. In *Associative Networks.* Findler, Nicholas (ed.), Academic Press.

[Chapman, 1985] Chapman, David 1985. Planning for conjunctive goals. AI Technical Report 802, Massachusetts Institute of Technology.

[Fikes and Nilsson, 1971] Fikes, Richard and Nilsson, Nils 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

[Fikes *et al.*, 1972] Fikes, Richard; Hart, Peter; and Nilsson, Nils 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

[Hendrix, 1979] Hendrix, Gary 1979. Encoding knowledge in partitioned networks. In *Associative Networks.* Findler, Nicholas (ed.), Academic Press.

[Kautz, 1987] Kautz, Henry 1987. *A Formal Theory of Plan Recognition.* Ph.D. Dissertation, University of Rochester, Department of Computer Science, Rochester, New York.

[Knoblock, 1988] Knoblock, Craig 1988. Automatically generating abstractions for planning. In *Proceedings of the First International Workshop in Change of Representation and Inductive Bias.* 53–65.

[Lifschitz, 1986] Lifschitz, Vladimir 1986. On the semantics of strips. In *Proceedings of the Workshop on Reasoning about Actions and Plans*, Timberline, Oregon.

[McCarthy and Hayes, 1969] McCarthy, John and Hayes, Patrick 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4.* Edinburgh University Press.

[Nau, 1987] Nau, Dana 1987. Hierarchical abstraction for process planning. In *Proceedings of Second International Conference in Applications of Artificial Intelligence in Engineering.*

[Reiter, 1978] Reiter, Raymond 1978. On closed world data bases. In *Logic and Data Bases.* Gallaire and Minker (eds.), Plenum Publishing Corporation.

[Robinson, 1965] Robinson, J. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM* 12(1):23–41.

[Sacerdoti, 1974] Sacerdoti, Earl 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

[Sacerdoti, 1977] Sacerdoti, Earl 1977. *A Structure for Plans and Behavior.* American Elsevier.

[Schank and Abelson, 1977] Schank, R. and Abelson, R. 1977. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates.

[Stefik, 1981] Stefik, Mark 1981. Planning with constraints. *Artificial Intelligence* 16(2):111–140.

[Tenenberg, 1988] Tenenberg, Josh 1988. *Abstraction in Planning.* Ph.D. Dissertation, University of Rochester, Dept. of Computer Science, Rochester, NY.

[Waldinger, 1977] Waldinger, R. 1977. Achieving several goals simultaneously. In *Machine Intelligence 8.* Elcock and Michie (eds.), Ellis Horwood.

[Wilkins, 1983] Wilkins, David 1983. Representation in a domain-independent planner. In *Proceedings of the 8th IJCAI.*

[Yang and Nau, 1988] Yang, Qiang and Nau, Dana 1988. The formal specification of task reduction schemas for hierarchical planning. *University of Maryland.*