In Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)
Phoenix, AZ, USA

Metaphysics of Planning Domain Descriptions

Siddharth Srivastava[1] and **Stuart Russell**[2] and **Alessandro Pinto**[1]

[1] United Technologies Research Center, Berkeley, CA 94705
[2] Computer Science Division, University of California, Berkeley CA 94720

Abstract

STRIPS-like languages (SLLs) have fostered immense advances in automated planning. In practice, SLLs are used to express highly abstract versions of real-world planning problems, leading to more concise models and faster solution times. Unfortunately, as we show in the paper, simple ways of abstracting solvable real-world problems may lead to SLL models that are unsolvable, SLL models whose solutions are incorrect with respect to the real-world problem, or models that are inexpressible in SLLs. There is some evidence that such limitations have restricted the applicability of AI planning technology in the real world, as is apparent in the case of task and motion planning in robotics. We show that the situation can be ameliorated by a combination of increased expressive power—for example, allowing angelic nondeterminism in action effects—and new kinds of algorithmic approaches designed to produce correct solutions from initially incorrect or non-Markovian abstract models.

1 Introduction

The need for using abstract models of systems for computational advantages is well appreciated in the planning literature. The vast majority of ongoing research in planning is focused on abstract models specified in STRIPS-like languages (SLLs), e.g. STRIPS, PDDL, SAS, etc.. In these languages, actions are defined using a set of effects that prescribe the atoms (e.g. variable assignments) that should be added to a state and those that should deleted from the state on which the action is applied. The use of a common representation language has led to immense progress in planners that can solve SLL problems, particularly through methods for automated synthesis of heuristic functions from domain specifications (Bonet and Geffner 2000; Hoffmann and Nebel 2001; Helmert, Haslum, and Hoffmann 2007).

Most researchers assume that domain experts intuitively formulate an abstraction of their systems into an SLL. For instance, the blocks-world model abstracts a situation where the robot needs to achieve a certain configuration of blocks on a table. Each "pickup" action in this model is implemented in the real world by a sequence of physical robot arm and gripper movements, each of which is implemented by lower-level control inputs to motors controlling various joints, each of which in turn is implemented by a specific pattern of voltages being applied, and so on. The SLL description of "pickup", on the other hand, mentions only a few logical preconditions and effects: if the gripper is empty and the block does not have anything on top of it, the gripper will end up holding the block.

However, abstractions of solvable problems can result in models that are unsolvable and also models whose solutions cannot actually be executed. The standard SLL specification of blocks world illustrates this by permitting solutions that cannot be executed. This is because the discrete SLL model cannot express real preconditions for picking up a block, which include geometric constraints resulting from collision avoidance, even in the absence of stacking.[1] Thus, a "pickup" action in a plan computed using an SLL model may not be executable because all paths between the robot's current base pose and the target block are obstructed. Although there do exist real-world scenarios for which the standard blocks-world specification is valid, they are extremely unusual: the robot should be either infinitely tall or attached to the ceiling, which in turn should be infinitely high and the robot should be able to pick objects using a suction/magnetic gripper. In this sense, one could argue that the true blocks-world problem has never been addressed by research on planning for SLLs. More generally, the problem of computing feasible plans using models that are imprecise abstractions of real-world problems has not received much research attention.

In this paper, we investigate the relationship between models in SLLs and the real-world systems that they are meant to abstract. In order to discuss the adequacy of SLLs for abstract models, we utilize the principles of metaphysics: in order to examine how the process of doing "physics" (aka, writing the rules of a domain) relates to the universe (aka, the real-world scenario) it purports to describe, one may as-

[1] The set of blocks obstructing the pickup of a block depends on several continuous variables such as the poses of the robot's base and the target block, the geometries of the robot's arm and the block, and finally, action arguments including the grasping pose and the trajectory to be used for a pickup; similar considerations are required when determining the obstructions "added" by a putdown action.

sume a particular kind of universe. Here, we begin by assuming a universe that is describable using an SLL at the concrete level, and show their inadequacy even under this assumption. The problems addressed in this paper arise because the process of abstraction tends to make action models imprecise (this is orthogonal to the problem of handling errors in modeling when the model has an accurate expression in the SLL (Nguyen and Kambhampati 2014)). In order to plan using the resulting imprecise action models, we need to *express imprecision without inducing incorrectness*. Inability to do so hinders progress and reduces the utility of the field, as evidenced by literature in task and motion planning that focuses on making solutions obtained using SLL models usable in real systems and in more accurate simulation models (Alami et al. 1998; Volpe et al. 2001; Plaku and Hager 2010; Hauser 2010).

We consider two main abstraction operations: removing predicates and removing action arguments (Sec. 2). These operations capture are sufficient for expressing a broad range of problems in SLLs, including tasks in robotics. We identify the conditions under which such abstractions satisfy fundamental, desirable properties such as the Markov property (Sec. 3.1). In order to correctly express abstract models, we draw upon two concepts from hierarchical planning (Marthi, Russell, and Wolfe 2007): angelic non-determinism in action effects 4, and the expression of super-sets or sub-sets of the truly possible action effects (Sec 4.2). Finally, we show that using information about the abstraction process together with representations that are imprecise (but not incorrect), allows the development of approaches for solving problems that don't have correct abstractions in SLLs (Sec 5).

2 Abstraction Framework

We focus on deterministic problems in this paper, and use a compact representation of conditional effects that can be compiled into SLLs. Let c and e be conjunctions of atoms. The conditional effect $c \rightarrow e$ indicates that if c holds in the state where the action is applied, positive literals in e are added to the state and negative literals in e are removed from it. Quantifiers can be used to compactly express conjunctions in this representation. The conditional effect $\top \rightarrow e$ is written as e. We illustrate our notation and the key ideas of abstraction using a running example.

Example 1 We illustrate the effects of abstraction on a simple model that is assumed to be accurate (Fig. 1(a)). We use the variables b_i, l_i and d to denote a block, a location, and a direction (left or right), respectively. In this problem, pickup and place actions require a direction of approach as an argument. In a state where both sides of a block are free, if it is picked up from the left, it's right portion remains free after the pickup and vice versa. Similarly, if a block is placed at a location l_1 from the left, the gripper ends up at the left of l_1. Suppose the original specification (a) is accurate. If an abstraction process drops the in_gripper predicate, we get the representation (b), where the most accurate description of place(b_1, l_1, d) depends on whether or not it followed pickup(b_1): the place action affects a block's location iff it is followed by a pickup action on that block. In this way,

a) "Original" specification:

pickup(b_1, l_1, d):

empty$(gripper)$		$\rightarrow$ in_gripper(b_1).
equals$(d, left)$, at(b_1, l_1), free$(b_1, left)$	gripper_at(d, l_1), empty$(gripper)$,	$\rightarrow \neg$ free$(b_1, left)$.
equals$(d, right)$, at(b_1, l_1), free$(b_1, right)$	gripper_at(d, l_1), empty$(gripper)$,	$\rightarrow \neg$ free$(b_1, right)$.

place(b_1, l_1, d):

in_gripper(b_1)	$\rightarrow$ at(b_1, l_1), $\neg$ in_gripper(b_1).
equals$(d, left)$	$\rightarrow$ gripper_at$(left, l_1)$.
equals$(d, right)$	$\rightarrow$ gripper_at$(right, l_1)$.

b) If the abstraction drops *in_gripper* in (a) we get a non-deterministic, pseudo-Markovian *place* action:

place(b_1, l_1, d):

$\top$	$\rightarrow$*ND*{at(b_1, l_1); $\emptyset$}
equals$(d, left)$	$\rightarrow$ gripper_at$(left, l_1)$
equals$(d, right)$	$\rightarrow$ gripper_at$(right, l_1)$

c) If the abstraction drops *equals* in (a) we get an operator with angelic choice:

pickup(b_1, l_1, d):

empty$(gripper)$	$\rightarrow$ in_gripper(b_1).
AngelicND{ gripper_at(d, l_1), at(b_1, l_1), empty$(gripper)$, free$(b_1, left)$	$\rightarrow \neg$ free$(b_1, left)$;
gripper_at(d, l_1), at(b_1, l_1), empty$(gripper)$free$(b_1, right)$	$\rightarrow \neg$ free$(b_1, right)$}

place(b_1, l_1, d):

in_gripper(b_1)	$\rightarrow$	at(b_1, l_1), $\neg$ in_gripper(b_1),
$\top$	$\rightarrow$	*AngelicND*{gripper_at$(left, l_1)$; gripper_at$(right, l_1)$}

Figure 1: Effects of abstraction on a model specification

this level of abstraction is not truly Markovian (Eg. 2 provides a more formal description using Def. 3). The description in Fig. 1(b) uses non-determinism to describe all possible effects of place. The operator *ND(η_1; ...; η_k)* denotes the non-deterministic selection of one of the conditional effects η_i. However this can make the model incomplete w.r.t. the existence of a contingent solution since no operation is guaranteed to change the location of a block. Clearly, this is undesirable.

On the other hand, if the abstraction process drops equals (Fig. 1(c)), the agent can choose arguments to place so as to satisfy the premise of either conditional effect in the real action specification. This can be expressed using *angelic* non-determinism indicating that the agent can choose to obtain a desired effect (the operator *AngelicND*) with syntax similar to the *ND* operator). This allows us to preserve solvability when expressing an abstracted model. The use of angelic non-determinism is discussed in Sec. 4.1.

Definition 1 Let X and S be sets such that $|S| \preceq |X|$. An **abstraction** from X to S is defined by a surjective function $f : X \rightarrow S$. For any $s \in S$, the **concretization function** $\gamma_f(s) = \{x \in X : f(x) = s\}$ denotes the **set of states**

represented by the abstract state s. For a set $C \subseteq X$, $[C]_f$ denotes the smallest set of abstract states representing C.

When considering abstractions from X to S, we refer to X as the set of "concrete" states and S as the set of "abstract" states. Each abstract state $s \in S$ can be written as $[x]_f$ for any state x such that $f(x) = s$, since the sets of concrete states corresponding to distinct abstract states are disjoint. For convenience we denote the abstract state space constructed by applying f on a set of states X as $[X]_f$. We abbreviate $c \in \gamma_f(s)$ as $c \in s$ when the abstraction function is clear from context. *Predicate abstraction* is an example of an abstraction function employed in situations where states are represented using logical structures and the abstraction projects out some of the predicates. Another example is the substitution abstraction, where a formula, whenever true in a state, is replaced with a term.

Definition 2 A deterministic **transition system** $T = \langle X, A\rangle$ consists of a set of states X and a set of actions A, where each a is a function from X to X. The **abstraction of an action** $a_i \in A$ w.r.t an abstraction function f is denoted as $[a_i]_f$; for any abstract state $s \in S$, $[a_i]_f(s) := [\{a_i(c) : c \in s\}]_f$. Given an abstraction function f, **the abstraction of a transition system** $T = \langle X, A\rangle$ is defined as $[T]_f = \langle [X]_f, \{[a]_f : a \in A\}\rangle$.

Unless otherwise noted, concrete transition systems used in this paper are deterministic. However, the abstraction of an action may be non-deterministic (mapping an abstract state to a set of abstract states) even when the original action is deterministic. We abbreviate the composition of abstract actions, $[a_1](\ldots([a_k](s))\ldots)$ as $[a_1]\ldots[a_k](s)$. We will drop the subscript f unless it is clear from the context.

3 Properties of Abstractions

Intuitively, an abstraction f is Markovian iff the abstract effect of an abstract action on an abstract state doesn't depend on the path through which the abstract state was reached.

Definition 3 An abstraction f is a **Markovian abstraction** of a transition system T defined over a set of states S and a set of actions A iff for every sequence of instantiated actions $a_1, \ldots, a_k \in A$, and abstract state $s \in [S]_f$, $[a_1 \ldots a_k]_f(s) = [a_1]_f \ldots [a_k]_f(s)$.

Example 2 Returning to the example in Fig. 1, the sequence of actions pickup(b_1, l_1, d_1);place(b_1, l_2, d_2) has the unique effects at(b_1, l_2) and gripper_at(d_2, l_2). Hence, this composition of actions can be specified as follows in an abstraction that drops the in-gripper predicate:

[pickup[(b_1, l_1, d_1); place(b_1, l_2, d_2)]:
  equals$(d_2, left)$ $\rightarrow$ at(b_1, l_2), gripper_at$(left, l_2)$
  equals$(d_2, right)$ $\rightarrow$ at(b_1, l_2), gripper_at$(right, l_2)$

[pickup(b_1, l_1, d_1)] has the same description as pickup in Fig. 1(a), except for the absence of the atom in_gripper(b_1). On any abstract state s, application of the abstraction action [pickup(b_1, l_1, d_1)] followed by applicaton of the abstract action [place(b_1, l_2, d_2)] (defined in Fig. 1(b)), results in two possible outcomes corresponding to the non-deterministic effects for the second action: either the block was not in the gripper and it remains at l_1, or it was in the gripper and moves to l_2. This is because the abstract state after picking up b_1 does not indicate whether or not b_1 is in the gripper. Thus, we have a situation where the composition of abstract actions is different from the abstraction of their composition, which implies that the abstraction that drops in_gripper is not Markovian.

3.1 Markovian Abstractions

A particularly desirable type of abstraction is one where every concrete member of s witnesses every abstract transition caused by $[a]$ from s.

Definition 4 An abstraction is a **forall-exists abstraction** iff for every $s' \in [a_1](s)$, for every $c \in s$, there exists a $c' \in a_1(c)$ such that $c' \in s'$.

In other words, the concretization of the abstract action's result always has something in common with the action's result on a concretization: $\gamma([a_1](s)) \cap a_1(\gamma(s)) \neq 0$. Bäckström and Jonsson (2013) consider such abstractions for the case of atomic state representations. In this paper we focus on predicate abstractions that result in forall-exists abstractions, and the relationship of these abstractions with Markovian abstractions. Forall-exists abstractions can also be seen as *simulation relations*.

Definition 5 (Milner 1971) A relation $R \subseteq S \times S$ is a **simulation relation** iff for all $(s, c) \in R$, for all $a \in A$, and for all $s' \in S$ such that $s \in a(s')$, there exists a $c' \in S$ such that $c \in a(c')$ and $(s', c') \in R$.

Theorem 1 *Forall-exists abstractions are simulation relations.*

PROOF Define the transition system $T'(S, A)$ where S is the union of the abstract states and the concrete states. The set of actions A only contains the set of concrete action symbols. For each $a \in A$, and abstract state s, $a(s) := [a](s)$ Let $R \subseteq S \times S$ be defined as follows: $(s, c) \in R$ iff $c \in \gamma(s)$. The result then follows from the definition of forall-exists abstractions and simulation relations. □

Theorem 2 *Forall-exists abstractions are Markovian.*

PROOF By the forall-exists property, in every sequence of abstract states generated by $[a_1][a_2]\ldots[a_k](s)$ has a witness thread of concrete transitions. Thus, for every result state in $[a_1]\ldots[a_k](s)$, there is at least one concrete member reachable from an element of $\gamma(s)$ via $a_1 \ldots a_k$. Thus, $[a_1 \ldots a_k](s) = [a_1 \ldots a_k(\gamma(s))]$ must include all result states in $[a_1]\ldots[a_k](s)$. The other direction always holds because $[a_1]\ldots[a_k](s)$ cannot be smaller than $[a_1 \ldots a_k](s)$. □

We now show that a special class of predicate abstractions generate forall-exists abstractions.

Definition 6 A predicate abstraction is **precondition preserving** if it doesn't drop any predicate that is used in the precondition of an action.

Lemma 1 *Every precondition-preserving abstraction of a deterministic transition system will be deterministic.*

PROOF Consider the application of an action on each member of an abstract state. They all satisfy the same set of preconditions, so the effects of the concrete action add and delete the same sets of predicates on each, even when there are conditional effects. The subsequent abstraction retains only the abstraction predicates which will be the same on all states because the same deltas were applied on the same initial set of abstracted predicates. Thus the set of abstract states capturing the result will be a singleton. □

Theorem 3 *Every precondition-preserving abstraction is a forall-exists abstraction.*

PROOF Since the abstraction is precondition preserving, a_1's preconditions do or do not hold in all members of s. In either case, all members represented by the initial state get carried to the same result states. □

Thus, precondition-preserving abstractions are Markovian.

3.2 Non-Markovian Abstractions

In this section we derive conditions under which an abstraction will be non-Markovian. A *non-Markovian residue* constitutes the evidence for an abstraction's being non-Markovian. It is a set of states that does not reach any member of an abstract state that the abstract transition system "thinks" is included in the result of an action application.

Definition 7 Let X be a set of states, f be an abstraction function, $s_1, s_2 \in [X]_f$, and a be an action in a transition system T defined using X such that $s_2 \in [a](s_1)$. The set $C_{s_1,s_2,a,f,T} = \{c : c \in s_1 \wedge a(c) \cap s_2 = \emptyset\}$ when non-empty, is referred to as the **non-Markovian residue** of s_1, s_2, a, f and T.

The non-Markovian residue therefore contains concrete states that have no post-image in one of the resulting abstract states for an abstract action.

Definition 8 A non-Markovian residue $C_{s_1,s_2,a,f,T}$ is **reachable in** T iff there is an abstract state $s \in [T]_f$ and a sequence of concrete actions α such that $\alpha(\gamma(s)) \subseteq C_{s_1,s_2,a,f,T}$.

Lemma 2 *If a transition system T has a reachable NMR w.r.t. an abstraction function f, then f is not a Markovian abstraction for T.*

PROOF Let the NMR be $C_{s_1,s_2,a,f,T}$, and let $\alpha(\gamma(s)) = C_{s_1,s_2,a,f,T}$. $[a(\alpha(\gamma(s)))] \subseteq [a(C_{s_1,s_2,a,f,T})]$ does not include s_2 but $[a][\alpha(s)] = [a][C_{s_1,s_2,a,f,T}] = [a](s_1)$ includes s_2. Thus, the abstraction f is not Markovian. □

We now show the converse, that non-Markovian abstractions must present a reachable NMR.

Lemma 3 *If an abstraction f for a transition system defined by a set of states X and a set of actions A is not Markovian, then there is a sequence of actions $\alpha \in A^*$ and an abstract state $s_1 \in [X]$ such that $\alpha(\gamma(s)) \cap \gamma(s_1)$ is an NMR.*

PROOF Suppose the abstraction is non-Markovian. Let $a_1, \ldots, a_k \in A$ be the smallest sequence such that $[a_1, \ldots, a_k](s) \neq [a_1](\ldots([a_k](s))$. Thus, there must exist an s_2 such that $s_2 \notin [a_1 \ldots a_k](s)$ but $s_2 \in [a_1] \ldots [a_k](s)$. (Note that any abstract state contained in $[a_1 \ldots a_k](s)$ will be contained in $[a_1] \ldots [a_k](s)$, because the abstraction can only add additional states to the result of an action application on an abstract state). Let s_1 be a member of $[a_2] \ldots [a_k](s)$ such that $[a_1](s_1)$ includes s_2, and let $C_1 = a_2 \ldots a_k(\gamma(s))$. Since $a_1 \ldots a_k$ is the smallest non-Markovian sequence, $[a_2 \ldots a_k](s) = [a_2] \ldots [a_k](s)$. Thus, $s_1 \in [a_2 \ldots a_k](s)$. Since $[a_2 \ldots a_k](s) = [a_2 \ldots a_k(\gamma(s))]$ by definition, this implies that $s_1 \in [C_1]$. Now $a_1(C_1) \cap s_2 = \emptyset$. If that was not true, $[a_1 \ldots a_k](s) = [a_1 \ldots a_k(\gamma(s))]$ would contain s_2 and we get a contradiction. Let $R = \gamma(s_1) \cap C_1$. R is non-empty because $s_1 \in [C_1]$; R is reachable from s because $C_1 = a_2 \ldots a_k(\gamma(s))$. Now $a_1(R)$ does not intersect with $\gamma(s_2)$ and R is a subset of $\gamma(s_1)$, therefore R is contained in the NMR of s_1, s_2, a_1, f and T, and T has a reachable NMR. □

The lemmas above provide necessary and sufficient conditions for non-Markovian abstractions.

Theorem 4 *An abstraction f of a transition system T is non-Markovian iff T has a reachable NMR w.r.t. f.*

4 Abstraction of Implicit Transition Systems

In general it is not feasible to express planning problems as explicit transition systems and one uses SLLs to describe transition systems implicitly by defining an initial state and a set of parameterized action operations. Our results about abstraction from the previous sections carry over to SLL representations. In addition, abstractions of SLL representations of transition systems introduce considerations about accuracy that do not arise for explicit transition systems.

Continuing with the principles of metaphysics described in the introduction, we consider abstractions of concrete transition systems that are expressed using SLLs. Since the operation of dropping action arguments necessitates dropping predicates, we focus on abstractions that drop predicates. To simplify the presentation we make the following assumptions without loss of generality: each predicate occurs with the same arguments in the action description. Different argument versions are considered to be different predicates for the purpose of the transformation. This imposition of uniformity effectively allows us to treat each occurrence of a predicate in the operator specification as a proposition.

In an SLL representation, suppose the predicate p is dropped in the abstraction. Then, for each conditional effect of an action, consider the following transformations:

T-ND1 if p occurs in the precondition or in the premise of a conditional effect, the effect e is replaced by the non-deterministic effect $ND\{e, \emptyset\}$, denoting that the operator may or may not take effect depending on the value of the dropped predicate.

T-ND2 if p occurs in the effect e, e is replaced by e' that has only the non-p components of e.

Applying these transformations yields abstracted actions in the sense that $a(\gamma(s)) \subseteq \gamma([a(s)])$. This is easy to verify since the abstract transition system only loses information w.r.t p, and each abstract state without p represents sets of

concrete states with each grounding of p set to true or false. This leads to the disjunction in action effects in T-ND1.

4.1 Angelic Non-Determinism

The transformations **T-ND1** and **T-ND2** use non-determinism to capture sets of reachable states. The conventional (demonic) semantics of non-determinism, however, are sometimes incorrect for such abstractions. Consider the example in Fig. 1. Using the syntactic transformation rules state above, dropping the *equals* predicate in the original specification (Fig. 1(a)) would result in:
placeND(b_1, l_1, d):

in_gripper(b_1)	$\rightarrow$	at(b_1, l_1), $\neg$ in_gripper(b_1).
$\top$	$\rightarrow$	*ND*{gripper_at$(left, l_1)$;$\emptyset$}, *ND*{gripper_at$(right, l_1)$;$\emptyset$}.

In this formulation, no contingent plan can be guaranteed to achieve the goal at$(b_1, l_1) \wedge$ gripper_at$(left, l_1)$ because gripper_at is provided only by placeND in the model. This results in an incorrect model because the goal *is* achievable even in the abstract transition system: regardless of the state in which place is applied, the agent can choose the argument d so as to achieve the desired version of gripper_at. In order to express imprecision without making the model incorrect, we can use *angelic* non-determinism operators to capture such abstractions. Angelic non-determinism in an action effect specifies the variations that are necessarily achievable by the agent as opposed to those that are not in the agent's control. E.g., recall Fig. 1(c):
place(b_1, l_1, d):

in_gripper(b_1)	$\rightarrow$	at(b_1, l_1), $\neg$ in_gripper(b_1).
$\top$	$\rightarrow$	*AngelicND*{ gripper_at$(left, l_1)$; gripper_at$(right, l_1)$}

We formalize this transformation as follows. Recall that we rename predicates if necessary, to ensure each predicate occurs in the set of conditional effect rules of an operator with a unique set and ordering of variables. Suppose a predicate $p(\bar{x})$ occurs in a set of conditional effect rules E. Let $\Sigma_{\bar{x} \rightarrow U}$ be the set of all instantiations of the variables in $\bar{x}$ to objects U. We then define the syntactic abstraction of E w.r.t. p, a set of states S, and U as follows:

$$[E]_{(p(\bar{x}),S,U)} = \textit{AngelicND}\{\textit{ND}\{E[\sigma, p]_s : s \in S\} : \sigma \in \Sigma_{\bar{x} \rightarrow U}\}$$

where $E[\sigma, p]_s$ is the version of E where σ is applied and all occurrences of p in the conditional premises are replaced by their truth values in s. In the rest of the paper we omit the arguments $\bar{x}$ from the subscript unless required for clarity. This leads to the following syntactic rule:

T-Angelic When S and U are known, replace the set of effects E with $[E]_{(p,S,U)}$

The angelic operator appears because the substitution σ is in the agent's control. This transformation expresses a tighter abstraction of reachable states than T-ND1 and T-ND2. T-Angelic generalizes the transformations T-ND1 and T-ND2. Indeed, T-ND1 and T-ND2 are obtained when AngelicND in the definition of $[E]_{(p,S,U)}$ is replaced by the less precise ND and $\{E[\sigma, p]_s : s \in S\}$ is replaced by the set of all versions of E obtained by substituting all possible evaluations of p in the premises of conditional rules in E. We can now define the transformation of the transition system that results from predicate abstraction.

Definition 9 Let P be a planning problem with a set of objects U; C be its state space; f be an abstraction that drops the predicate p and $[C]_f$ be the abstract state space produced by applying f on C. Let a be an action with the effect description E then $\forall s \in [C]_f$, we define the **angelic representation** of $[a(s)]$ as $[E]_{(p,\gamma(s),U)}$.

The angelic abstraction $[a]_{p_1,\ldots,p_k}$ of an action a produced by dropping an ordered set of predicates $p_1, \ldots, p_k$ is defined by treating the angelic abstraction of each predicate as a distinct abstraction function and composing the resulting transformations in order $[\ldots [a(s)]_{p_1} \ldots]_{p_k}$, where s is a state in the abstract state space without $p_1, \ldots, p_k$. Computation of optimal orderings of abstractions is a subject for future work. Note that the representation in Def. 9 may not be the most accurate possible representation of the abstract transition system, but it guaranteed to be an over-approximation: $\forall c \in s, a(c) \in \gamma([a(s)]_{p_1,\ldots,p_k})$.

The definition of $[E]_{(p,S,U)}$ implies that when p is a static fluent, then $p[\sigma]$ has the same truth value in all states and T-Angelic is independent of S. This is precisely what occurs in Fig. 1(c). To see this, note that *AngelicND*$\{p \rightarrow q \wedge w, p \rightarrow q \wedge v\}$ is equivalent to $p \rightarrow q \wedge$ *AngelicND*$\{w, v\}$. We formalize the arguments above as follows.

Theorem 5 *If the predicate p is a static fluent w.r.t. S, then $[E]_{(p,S,U)}$ introduces necessarily angelic non-determinism.*

The premise of this result is a sufficient, but not necessary condition for obtaining purely angelic non-determinism. We say that a predicate p occurring in the premises of conditional effects for an action a is **in the agent's control** in a w.r.t. a set of states S if in every $s \in S$, for every literal form of p used in a conditional premise in a, there exists an assignment of the arguments of a which makes that literal form true. In other words, in every state action arguments can be chosen to satisfy the form of p used in the premise of any desired conditional rule. In such cases also the effect of dropping p can be expressed as an angelic choice among the possible effects obtained for each evaluation of p because the agent can achieve each evaluation in every state. However, the form of $E_{(p,S,U)}$ written above doesn't directly produce such a purely angelic effect. This is indicative of non-trivial distributive properties of angelic and demonic non-deterministic operators.

4.2 When Angelic Representations are Infeasible

In some situations abstraction can result in too many angelic choices corresponding to different combinations of truth values for the dropped predicates. For such cases, we need an intermediate representation that is not as intractable to compute as the one generated by T-Angelic, but avoids the incorrectness resulting from a demonic non-deterministic operator. Consider a more realistic model where the description of the act of placing an object uses a geometric predicate to determine whether or not an obstruction is introduced:

pickup-clear(b_1, l_1, d):

at(b_1, l_1), empty($gripper$), ($\forall b_2 \neg$obstructs(b_2, b_1, d))	→	in_gripper(b_1), $\neg$at(b_1, l_1), $\forall b_2, d \neg$obstructs(b_1, b_2, d).

place-clear(b_1, l_1, o_1):

in_gripper(b_1), open(l_1)	→	at(b_1, l_1), $\neg$in_gripper(b_1), $\neg$open(l_1).
in_gripper(b_1), open(l_1), shape_obstructs(b_1, l_1, o_1, b_2, l_2), relative_direction(l_1, l_2, d)	→	obstructs(b_1, b_2, d).

Note that pickup-clear requires a block to be unobstructed. This specification uses new variables: b_2 denotes an arbitrary block; and l_2 denotes an arbitrary location for a block. Unbound variables denote an implicit conjunction of the rules in which they occur, over all of their possible instantiations. The geometric predicate shape_obstructs(b_1, l_1, o_1, b_2, l_2) is true iff placing b_1 at l_1 in the orientation o_1 will obstruct the gripper from picking b_2 at l_2 from the direction of l_1. Expressing states with such predicates would require expensive geometric computations for all possible groundings, and it is therefore desirable to abstract such predicates away. Angelic non-determinism could be used to express the possible combinations of obstructs predicates introduced when shape_obstructs and relative_direction are dropped. However, in order to do so, one must compute the truth values of these predicates for every choice of b_2, l_1 and l_2, which can be computationally expensive (and infeasible when l_1 or l_2 range over high-dimensional poses in the configuration space of a real robot). On the other hand, a non-deterministic representation would have no contingent solutions due to cyclic obstructions resulting from non-deterministic choices.

We therefore need an intermediate representation that is not as intractable to compute as the one generated by T-Angelic, that remains imprecise but is not incorrect. This can be achieved by indicating that the effects of an abstract action on certain predicates are deterministic, but undetermined in the current abstraction:

place-clear-un(b_1, l_1):

in_gripper(b_1), open(l_1)	→	at(b_1, l_1), $\neg$in_gripper(b_1), $\neg$open(l_1), $\widehat{+}${obstructs}.

The $\widehat{+}$ indicates that this action *may* add some obstructs facts. Retaining this information is useful: since the undetermined effects are annotated and are deterministic, custom reasoning tools can be used to determine the exact effects. Such an action model is *sound* in the sense of theorem proving: the portion outside $\widehat{+}$ asserts only the properties that are guaranteed to be achieved by the action.

5 Planning Modulo Abstractions

We now consider the problem of planning in the presence of abstractions. Early work in this direction (Sacerdoti 1974; Knoblock 1991) addresses special cases of this problem, where abstraction hierarchies can be assumed to satisfy certain properties that aid reasoning algorithms. More recent approaches use abstractions in heuristics for guiding search algorithms that operate on transition systems described in SLLs (e.g., (Helmert, Haslum, and Hoffmann 2007)). In order to develop general algorithms for planning across a pair of concrete and abstract models, we can draw upon the literature on SAT modulo theories (SMT), which deals with the similar problem of pairing SAT solvers with reasoning engines for theories whose translations into SAT would be intractable or impossible. The input formula for an SMT solver can include literals that represent atoms from a theory (e.g. $a() + 2 < b()$ is an atom that belongs to the theory of arithmetic). The basic $DPLL(T)$ algorithm (Nieuwenhuis, Oliveras, and Tinelli 2006) used by modern SMT solvers proceeds as follows. A SAT solver is used to search for a satisfying model of the input formula without any constraints on the atoms that are actually constrained by T. If no model is found, the formula is unsatisfiable. If a model M is found, a decision procedure for T (T-solver) is used to decide if M is T-consistent. If it is, the problem is solved. Otherwise, T-solver provides a lemma precluding M, which is then added to the theory. The process is then repeated until the SAT solver finds a T-consistent model or proves unsatisfiability.

We build upon the general principles of SMT to use any pair of planning algorithms where one operates on an abstraction of another's theory, as follows. The algorithm that uses the abstract model representation to generate high-level plans (partial models) plays the role of the SAT solver. Let each action take the form $a(\bar{x}, \bar{y})$, where $\bar{y}$ are the arguments to be dropped, and P be the predicates that are abstracted. Each high-level plan produced by the search algorithm corresponds to a formula of the form $\exists_{\bar{y}_1,\ldots,\bar{y}_n} \varphi_{s_0} \wedge a_1(\bar{o}_1, \bar{y}_1, 1) \wedge \varphi_{s_1} \wedge \ldots \wedge a_n(\bar{o}_n, \bar{y}_n n) \wedge \varphi_{goal_n}$, where the last action-arguments are timesteps, $\bar{y}_i$ are the dropped action arguments, $\bar{o}_i$ are instantiations of constants to the remaining arguments, and φ_{s_i} are formulas representing intermediate states and assignments for predicates in P, and φ_{goal_n} is a formula asserting that the goal is achieved at time n. Any algorithm that determines low-level feasibility of such plans and produces high-level lemmas in case of infeasibility can be used in place of the T-solver to mimic the SMT process.

There is ample evidence for the viability of this paradigm. In fact, the paradigm described above unifies several approaches in the literature on task and motion planning. (Gregory et al. 2012) develop a planning-modulo-theories approach that factors models into pieces that come from specific theories and uses callouts to low-level theories during search. (Erdem et al. 2011) explore various implementations of SMT style architectures, while using an ASP solver for high-level reasoning with a discretized space of possible action arguments. The approach presented by (Srivastava et al. 2014) is also similar to the SMT design outlined above; they identify a class of abstracted, imprecise models where high-level plans can be computed efficiently using classical planners. While these approaches have been focused on task and motion planning, the underlying problems being addressed are various forms of planning across abstractions and the analysis presented in this paper lends directly to expanding the scope and scalability of this methodology.

6 Conclusions

We presented an analysis of representational abstractions for planning problem specifications and proved several results categorizing abstraction mechanisms that exhibit desirable properties such as the Markov property. We showed that expressing a large class of solvable real-world problems in SLLs results in unsolvable or incorrect models, and presented methods for overcoming these limitations. We also showed that together with information about the abstraction process, such models can be utilized in solving an entirely new class of problems that were not expressible in SLLs. Our analysis presents several directions for future work, including refinement of the planning paradigm and methods for searching in the space of abstract representations.

7 Acknowledgements

The funding provided by the United Technologies Research Center is greatly appreciated. Opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the United Technologies Research Center.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *IJRR* 17:315–337.

Bäckström, C., and Jonsson, P. 2013. Bridging the gap between refinement and heuristics in abstraction. In *Proc. AAAI*.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. ICAPS*.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Proc. ICRA*.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning modulo theories: Extending the planning paradigm. In *Proc. ICAPS*.

Hauser, K. 2010. Task planning with continuous actions and nondeterministic motion planning queries. In *Proc. of AAAI Workshop on Bridging the Gap between Task and Motion Planning*.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *Proc. ICAPS*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Knoblock, C. A. 1991. Search reduction in hierarchical problem solving. In *Proc. AAAI*.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Angelic semantics for high-level actions. In *Proc. ICAPS*.

Milner, R. 1971. An algebraic definition of simulation between programs. In *Proc. IJCAI*.

Nguyen, T., and Kambhampati, S. 2014. A heuristic approach to planning with incomplete strips action models. In *Proc. ICAPS*.

Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2006. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM (JACM)* 53(6):937–977.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Proc. ICRA*.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial intelligence* 5(2):115–135.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Proc. ICRA*.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; and Das, H. 2001. The CLARAty architecture for robotic autonomy. In *Proc. of IEEE Aerospace Conference*.