Sequential Decision Making for Intelligent Agents
Papers from the AAAI 2015 Fall Symposium

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 94709

Abstract

Domain models for sequential decision-making typically represent abstract versions of real-world systems. In practice, such abstract representations are compact, easy to maintain, and afford faster solution times. Unfortunately, as we show in the paper, simple ways of abstracting solvable real-world problems may lead to models whose solutions are incorrect with respect to the real-world problem. There is some evidence that such limitations have restricted the applicability of sequential decision-making 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 literature on sequential decision making. The benefit of hierarchical representations is also well understood (Parr and Russell 1998; Sutton, Precup, and Singh 1999; Dietterich 2000; Andre and Russell 2002). Approaches for exploiting hierarchical structures typically utilize temporal abstractions, where multiple actions, are grouped together into meaningful control structures or behaviors. On the other hand, state abstractions group states together into abstract states. Unlike temporal abstractions, state abstractions can preclude accurate representations of the value function. Such state abstractions are often implicit in discrete MDP/POMDP representations.

Consider a PR2 robot that needs to empty the dishwasher. We may construct a POMDP model for this domain with state variables for the current position of each dish using (x, y, z) coordinates in a discrete 3-dimensional grid. Assuming cameras are mounted on the manipulators, we may also construct a noisy observation function that models object identity and pose detectors. Uncertainty about the robot's localization and movement can be captured similarly. When the available sensors and actuators permit, getting a robot to empty the dishwasher with such a model seems to be a matter of selecting the right reward function and allocating sufficient time for computation. Unfortunately, this hypothesis turns out to be false.

This is because such problem representations implicitly ablate details such as the precise poses and geometries of the dishes, dishwasher racks, and robot arms; kinematic constraints on the feasibility of being able to access each dish from a given pose of the robot's base. As a result, an optimal policy for the stated POMDP model may attempt to pick up a plate in a state where a spatula obstructs all motion plans for grasping it. [1] When such considerations are ignored, even with perfect sensors and actuators, optimal high-level policies may be *unexecutable* in practice.

At first glance this situation seems to be remediable: decision-theoretic frameworks for sequential decision making incorporate support for noisy sensors and actuators which could potentially be used to capture the imprecision arising from model abstraction. In the dishwasher scenario, one may include the possibility that a pickup operation can fail, by expressing the effects of an action in an abstract state as a distribution over the set of all possible results under different settings of the ablated properties. However, this leads to two problems. First, one would have to determine all the truly possible result states in the abstract domain model. This requires a significant amount of reasoning with the continuous version of the model (considering all the truly possible trajectories and speeds) if it it is available, or using real world execution when it isn't. Second, even if this is done, one would end up dealing with all possible collections of fragments of dishes scattering all over the kitchen every time the robot moves its arm (the location, geometry, speed profile, and the trajectory of movement are abstracted out and all possibilities have to be considered). This is unfortunate because in practice, discretizations of even two dimensional problems that take into account object geometries and manipulator paths can result in state spaces that are too large to express or solve (Srivastava et al. 2014a). In other words, such techniques fail to achieve the underlying objective of utilizing abstracted models for efficient reasoning.

These observations motivate the need for rigorous analysis of abstraction functions that may be employed while

[1] The set of objects obstructing a pickup depends on several continuous variables including the poses of the robot's base and the target object, the geometries of the robot's arm and the object, 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 induced by a putdown action.

creating the model of an underlying system in a given mathematical language. For sequential decision making in particular, we would like such models to retain key properties such as a Markovian transition system and executability of the computed policies. Obviously, not all abstractions would satisfy such properties. The interesting question then becomes, are desirable abstractions always possible? If not, is there a way to exploit the benefits of hierarchical structure in situations where the required or useful abstractions violate some such properties? These questions about abstraction are particularly relevant today as advances in embedded systems and robotics provide new opportunities for sequential decision making techniques. As noted in the example above however, such systems typically require intractably large or continuous state spaces to be modeled accurately.

In this paper, we make three main contributions. First, we study abstraction functions and derive the conditions under which they satisfy certain desirable properties for sequential decision making. Second, we show that in many domains of interest these properties are violated for a large class of abstraction functions that are used implicitly by domain designers. Finally, we discuss how, in situations where the abstraction is not Markovian and does not preserve executability, knowledge of the abstraction function can be utilized to make the computation of solutions more efficient. Our work also sheds light on conditions under which the abstraction of a deterministic model may result in a model that is nondeterministic.

In order to study these core problems, we focus on situations where the underlying system is fully observable and deterministic. We ground our discussion to STRIPS like languages (SLLs), e.g., PDDL, SAS, STRIPS etc. SLLs express transition systems in a functional form by defining the changes on state variables that would result if the action was applied on a state that satisfied its preconditions. A number of extensions of such logic-based languages have been proposed for the efficient expression of decision-theoretic models (Sanner and Boutilier 2009; Sanner and Kersting 2010; Srivastava et al. 2014b).

We consider two main abstraction operations: removing predicates and removing action arguments (Sec.2). These operations capture transformations required 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. 2.1). 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 (Sec. 3), and the expression of super-sets or sub-sets of the truly possible action effects (Sec 3.2). Finally, we show that by using information about the abstraction process together with representations that are imprecise (but not incorrect), we can develop an approach for solving problems that don't have correct abstractions in SLLs (Sec 4).

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.

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 nondeterministic, pseudo-Markovian definition for the *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$(b_1, left)$ gripper_at(d, l_1), at(b_1, l_1), empty$(gripper)$free$(b_1, right)$	 $\rightarrow \neg$ free$(b_1, left)$; $\rightarrow \neg$ 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

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, this level of abstraction is not truly Markovian (Eg. 2 provides a more formal description using Def. 1). 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 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.

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 (the operator *AngelicND*) with syntax similar to the *ND* operator). In contrast to demonic non-determinism in action effects, where the environment "chooses" an outcome (and thus the agent must plan for all possible outcomes), angelic non-determinism over a set of possible outcomes is used to express actions for which the agent may use an appropriate implementation of an action, to select any of the desired outcomes.

The use of angelic non-determinism is discussed in Sec. 3.1.

Formally, we consider abstractions defined by a surjective abstraction function f that maps concrete states to abstract states. As a result, each abstract state s can be written as $[x]_f$ for any state x such that $f(x) = s$; the sets of concrete states corresponding to distinct abstract states are disjoint. We denote the set of states represented by an abstract state s using the notation $\gamma_f(s)$. For brevity, we abbreviate $c \in \gamma_f(s)$ as $c \in s$ when the abstraction function is clear from context. We denote the abstract state space constructed by applying f on a set of states X as $[X]_f$. Predicate abstraction is a special case where the function f projects out some propositions/predicates from the state. The substitution abstraction, where a formula, whenever true in a state, is replaced with a term, is also a special case of this abstraction.

Let T be a transition system defined over a set of states S and a set of actions A specified in some language. Let $[T]_f$ be an abstraction of T, obtained using a state abstraction f as follows. For any set of states C, let $[C]_f$ be the smallest set of abstract states capturing all of C. For each $a_i \in A$, let $[a_i]_f$ be the abstract representation of a_i: for any abstract state s, $[a_i](s) := [\{a_i(c) : c \in s\}]$. We define [] for sequences of actions in the same way, and abbreviate the composition of abstract actions, $[a_1](\ldots([a_k](s))\ldots)$ as $[a_1]\ldots[a_k](s)$. $[T]_f$ is the transition system over states $\{[c] : c$ is a concrete state in $T\}$ and actions $\{[a] : a$ is an action in $T\}$. We will drop the subscript f unless it is required for clarity.

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 1 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.

2.1 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 2 An abstraction is a **forall-exists abstraction** iff for every $s' \in [a_1](s)$, for every $c \in \gamma(s)$, there exists a $c' \in a_1(c)$ such that $c' \in \gamma(s')$.

In other words, the concretization of the abstract action's result always has something in common with the action's result on a concretization: $\gamma([a_1](s)) \cap a_1(\gamma(s)) \neq 0$. 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*. 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 We need to show that $[a_1 \ldots a_k](s) = [a_1] \ldots [a_k](s)$.

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

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

Definition 3 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. □

As a corollary, all precondition-preserving abstractions are Markovian.

2.2 Non-Markovian Abstractions

In this section we present conditions under which an abstraction will be non-Markovian. A *non-Markovian residue* constitutes in some sense, 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" should be included in the result of an action application.

Definition 4 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 $C_{s_1,s_2,a,f,T} = \{c : c \in s_1 \wedge a(c) \cap \gamma_f(s_2) = \emptyset\}$ is the **non-Markovian residue** of s_1, s_2, a, f and T.

The non-Markovian residue C therefore includes states that have no post-image under a in the abstract result state $s_2 \in [a](s_1)$: $a(C_{s_1,s_2,a,f,T}) \cap \gamma_f(s_2) = \emptyset$.

Definition 5 A non-Markovian residue $C_{s_1,s_2,a,f,T}$ is **reachable** iff there is an abstract state s and a sequence of concrete actions α such that $\alpha(\gamma(s)) \subseteq C_{s_1,s_2,a,f,T}$.

Lemma 2 *If an NMR $C_{s_1,s_2,a,f,T}$ is reachable, the abstraction f is not Markovian.*

PROOF 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. □

We now show that non-Markovian abstractions have a non-Markovian residue.

Theorem 4 *If an abstraction f for a state space X is not Markovian, then there is a sequence of actions α 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 $a_1, \dots, a_k$ be the smallest sequence such that $[a_1, \dots, a_k](s) \neq [a_1](\dots([a_k](s))$.

Thus, there must exist an s_2 such that $s_2 \notin [a_1 \dots a_k](s)$ but $s_2 \in [a_1]\dots[a_k](s)$. (Note that any abstract state contained in $[a_1 \dots a_k](s)$ will be contained in $[a_1]\dots[a_k](s)$, 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 $[a_2]\dots[a_k](s)$ such that $[a_1](s_1)$ includes s_2, and let $C_1 = a_2 \dots a_k(\gamma(s))$. Since $a_1 \dots a_k$ is the smallest non-Markovian sequence, $[a_2 \dots a_k](s) = [a_2]\dots[a_k](s)$. Thus, $s_1 \in [a_2 \dots a_k](s)$. Since $[a_2 \dots a_k](s) = [a_2 \dots a_k(\gamma(s))]$ by definition, this implies that $s_1 \in [C_1]$.

Now $a_1(C_1)$ does not contain any member of s_2. If it did, $[a_1 \dots a_k](s) = [a_1 \dots a_k(\gamma(s))]$ would contain s_2 and we get a contradiction. Let $R = \gamma(s_1) \cap C_1$. R is non empty because $s_1 \in [C_1]$. Now $a_1(R)$ does not intersect with $\gamma(s_2)$ and R is a subset of $\gamma(s_1)$. Therefore R is a non-Markovian residue. □

3 Syntactic Transformations for Abstraction

The preceding sections were focused on abstraction in explicitly specified transition systems. However, in general it is not feasible to express planning problems as explicit transition systems. In this section, we study abstractions of transition systems expressed implicitly 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.

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 nondeterministic 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.

The resulting action specification captures the abstracted action 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.

3.1 Angelic Non-Determinism

The transformations listed above 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. The following illustration is reproduced from 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 6 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 $p_1, \ldots, p_k$. The computation of optimal orderings of abstractions is beyond the scope of this paper and is left for future work. Note that the representation in Def. 6 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.

In concluding this section, we note that dropping a predicate can make some action arguments unnecessary. For instance, the d argument in *place* can be removed after dropping *equals* because it plays no role in the transformed description.

3.2 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))	$\rightarrow$	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)	$\rightarrow$	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)	$\rightarrow$	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)	$\rightarrow$	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.

4 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 accurate 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. Nieuwenhuis et al. describe several variations and methods for optimizing this basic procedure, which can be adapted to our setting.

For any time horizon H, the problem of planning using the accurate and the abstracted models can be reduced to the SMT problem using translations such as those carried out in SATPLAN (Kautz and Selman 1992). However, the general principles of SMT can be adapted to use any pair of reasoning processes in place of the SAT solver and the theory solver. In particular, a search algorithm that uses the abstract model representation to generate high-level plans (partial models) can play 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. A few practical considerations make it difficult to apply this exact process: (a) low-level theory solvers may be only *probabilistically* complete, as in the case of motion planning; (b) the number of possible truth values for

abstracted predicates can be intractable; (c) heuristic methods may be required for instantiation of abstracted action arguments; and finally, (d) incorporating lemmas in the high-level theory of planning can be difficult if a non-SAT planner is used.

Several approaches are possible for addressing these aspects. (a) and (c) can be addressed by daisy chaining the search for instantiations of dropped action arguments with increasing computation towards the determination of low-level feasibility. We conjecture that the resulting solver will be probabilistically complete if the space of feasible instantiations for every feasible high-level plan is of non-zero measure under a probabilistic instantiation process, and the low-level reasoner is probabilistically complete. (b) can be addressed by developing search algorithms that utilize the structure of abstract representations, e.g., by optimistically choosing preconditions that have to be true. (d) can be addressed by starting distinct threads for each instantiation and using revised models in each.

We already have evidence of viability for this paradigm in situations where the abstraction process is known, and leads to imprecise action models. (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. 2014a) 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.

5 Related Work and Conclusions

Physical systems such as those involving robots form a significant source of motivation for sequential decision making. Since such modeling such systems typically leads to large MDPs and POMDPs, the topic of utilizing hierarchies for computational efficiency has received significant attention. With few exceptions, most of the hierarchies used in such methods are temporal abstractions (Parr and Russell 1998; Sutton, Precup, and Singh 1999) or require that the abstraction does not project out any variables that are relevant for accurately expressing the value function (Dietterich 2000; Andre and Russell 2002). However, as we note above, several commonly used abstractions violate this condition. In particular, task and motion planning problems typically involve lossy state abstractions under which policy expressions suffer not just in terms of optimality but also in terms of basic executability.

The DRIPS framework (Haddawy, Doan, and Goodwin 1995) is perhaps the closest in spirit to the problems addressed in this paper. In that work, action abstractions are considered by merging together different conditional effects. Abstraction actions are represented using upper and lower bounds on the probabilities of different effects. However, state abstractions, which are the focus of the current paper, are not considered. Konidaris et al. (2014) propose an approach for computing a factored propositional transition model given an input continuous state space and subgoal option models. Such an approach could be combined with the abstraction techniques presented in this paper to obtain trade-offs between the computational cost of constructing high-level representations and the level of abstraction of the resulting transition model.

The methods developed in this paper could be extended to the stochastic setting by derive DRIPS-style probability ranges for abstract actions that are induced by state abstractions. An alternative approach would be to compute marginalizations w.r.t. the variables that are projected out. Our analysis of abstractions also presents several directions for future work, including refinement of the planning paradigm. The abstraction functions we studied can be used to construct a search space of abstract representations and automatically compute useful abstract models for a given objective.

In conclusion, 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 a paradigm for solving an entirely new class of problems that were not expressible in SLLs.

References

Andre, D., and Russell, S. J. 2002. State abstraction for programmable reinforcement learning agents. In *AAAI/IAAI*, 119–125.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

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 *ICRA*, 4575–4581.

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

Haddawy, P.; Doan, A.; and Goodwin, R. 1995. Efficient decision-theoretic planning: Techniques and empirical analysis. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proc. of the 10th European Conference on Artificial intelligence*, 359–363.

Knoblock, C. A. 1991. Search reduction in hierarchical problem solving. In *Proc. of the 9th National Conference on Artificial Intelligence*, 686–691.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2014. Constructing symbolic representations for high-level planning. In *Proc. AAAI*.

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

Parr, R., and Russell, S. 1998. Reinforcement learning with hierarchies of machines. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97.

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

Sanner, S., and Boutilier, C. 2009. Practical solution techniques for first-order MDPs. *Artificial Intelligence* 173(5-6):748–788.

Sanner, S., and Kersting, K. 2010. Symbolic dynamic programming for first-order POMDPs. In *Proc. AAAI*.

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

Srivastava, S.; Russell, S.; Ruan, P.; and Cheng, X. 2014b. First-order open-universe POMDPs. *Proc. UAI-14*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1):181–211.