# Abstraction in Situation Calculus Action Theories

**Bita Banihashemi**
York University
Toronto, Canada
bita@cse.yorku.ca

**Giuseppe De Giacomo**
Sapienza Università di Roma
Roma, Italy
degiacomo@dis.uniroma1.it

**Yves Lespérance**
York University
Toronto, Canada
lesperan@cse.yorku.ca

**Abstract**

We develop a general framework for *agent abstraction* based on the situation calculus and the ConGolog agent programming language. We assume that we have a high-level specification and a low-level specification of the agent, both represented as basic action theories. A *refinement mapping* specifies how each high-level action is implemented by a low-level ConGolog program and how each high-level fluent can be translated into a low-level formula. We define a notion of *sound abstraction* between such action theories in terms of the existence of a suitable bisimulation between their respective models. Sound abstractions have many useful properties that ensure that we can reason about the agent's actions (e.g., executability, projection, and planning) at the abstract level, and refine and concretely execute them at the low level. We also characterize the notion of *complete abstraction* where all actions (including exogenous ones) that the high level thinks can happen can in fact occur at the low level.

## 1 Introduction

Intelligent agents often operate in complex domains and have complex behaviors. Reasoning about such agents and even describing their behavior can be difficult. One way to cope with this is to use *abstraction* (Saitta and Zucker 2013). In essence, this involves developing an abstract model of the agent/domain that suppresses less important details. The abstract model allows us to reason more easily about the agent's possible behaviors and to provide high-level explanations of the agent's behavior. To efficiently solve a complex reasoning problem, e.g. planning, one may first try to find a solution in the abstract model, and then use this abstract solution as a template to guide the search for a solution in the concrete model. Systems developed using abstractions are typically more robust to change, as adjustments to more detailed levels may leave the abstract levels unchanged.

In this paper, we develop a general framework for *agent abstraction* based on the situation calculus (SitCalc) (McCarthy and Hayes 1969; Reiter 2001) and the ConGolog (De Giacomo, Lespérance, and Levesque 2000) agent programming language. We assume that one has a high-level/abstract action theory, a low-level/concrete action theory, and a *refinement mapping* between the two. The mapping associates each high-level primitive action to a (possibly nondeterministic) ConGolog program defined over the low-level action theory that "implements it". Moreover, it maps each high-level fluent to a state formula in the low-level language that characterizes the concrete conditions under which it holds.

In this setting, we define a notion of a high-level theory being a *sound abstraction* of a low-level theory under a given refinement mapping. The formalization involves the existence of a suitable bisimulation relation (Milner 1971; 1989) between models of the low-level and high-level theories. With a sound abstraction, whenever the high-level theory *entails* that a sequence of actions is executable and achieves a certain condition, then the low level must also entail that there exist an executable refinement of the sequence such that the "translated" condition holds afterwards. Moreover, whenever the low level thinks that a refinement of a high-level action (perhaps involving exogenous actions) can occur (i.e., its executability is satisfiable), then the high level does also. Thus, sound abstractions can be used to perform effectively several forms of reasoning about action, such as planning, agent monitoring, and generating high-level explanations of low-level behavior. We also provide a proof-theoretic characterization that gives us the basis for automatically verifying that we have a sound abstraction.

In addition, we define a dual notion of *complete abstraction* where whenever the low-level theory *entails* that some refinement of a sequence of high-level actions is executable and achieves a "translated" high-level condition, then the high level also *entails* that the action sequence is executable and the condition holds afterwards. Moreover, whenever the high level thinks that an action can occur (i.e., its executability is satisfiable), then there exists a refinement of the action that the low level thinks can happen as well.

Many different approaches to abstraction have been proposed in a variety of settings such as planning (Sacerdoti 1974), automated reasoning (Giunchiglia and Walsh 1992), model checking (Clarke, Grumberg, and Long 1994), and data integration (Lenzerini 2002). Most of these do not deal with dynamic domains. Previous work on hierarchical planning generally makes major simplifying assumptions (Nau, Ghallab, and Traverso 2016). In contrast, our approach deals with agents represented in an expressive first-order framework. We later discuss related work in more details.

## 2 Preliminaries

The *situation calculus* is a well known predicate logic language for representing and reasoning about dynamically changing worlds. Within the language, one can formulate action theories that describe how the world changes as a result of actions (Reiter 2001). We assume that there is a *finite number of action types* $\mathcal{A}$. Moreover, we assume that the terms of object sort are in fact a countably infinite set $\mathcal{N}$ of standard names for which we have the unique name assumption and domain closure. For simplicity, and w.l.o.g., we assume that there are no functions other than constants and no non-fluent predicates. As a result a basic action theory (BAT) $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent, (second-order, or SO) axioms of the situation calculus ($\Sigma$); (first-order, or FO) precondition axioms stating when actions can be legally performed ($\mathcal{D}_{poss}$); (FO) successor state axioms (SSAs) describing how fluents change between situations ($\mathcal{D}_{ssa}$); (FO) unique name axioms for actions and (FO) domain closure on action types ($\mathcal{D}_{ca}$); (SO) unique name axioms and domain closure for object constants ($\mathcal{D}_{coa}$); and (FO) axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). A special predicate $Poss(a, s)$ is used to state that action $a$ is executable in situation $s$; precondition axioms in $\mathcal{D}_{poss}$ characterize this predicate. The abbreviation $Executable(s)$ means that every action performed in reaching situation $s$ was possible in the situation in which it occurred. In turn, successor state axioms encode the causal laws of the world being modeled; they replace the so-called effect axioms and provide a solution to the frame problem.

To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called *high-level programming languages* have been defined. Here we concentrate on (a fragment of) ConGolog that includes the following constructs:

$$\delta ::= \alpha \mid \varphi? \mid \delta_1;\delta_2 \mid \delta_1|\delta_2 \mid \pi x.\delta \mid \delta^* \mid \delta_1\|\delta_2$$

Above, $\alpha$ is an action term, possibly with parameters, and $\varphi$ is a situation-suppressed formula, i.e., a formula with all situation arguments in fluents suppressed. We denote by $\varphi[s]$ the formula obtained from $\varphi$ by restoring the situation argument $s$ into all fluents in $\varphi$. Program $\delta_1|\delta_2$ allows for the nondeterministic choice between programs $\delta_1$ and $\delta_2$, while $\pi x.\delta$ executes program $\delta$ for *some* nondeterministic choice of a legal binding for variable $x$ (observe that such a choice is, in general, unbounded). $\delta^*$ performs $\delta$ zero or more times. Program $\delta_1\|\delta_2$ expresses the concurrent execution (interpreted as interleaving) of programs $\delta_1$ and $\delta_2$.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates (De Giacomo, Lespérance, and Levesque 2000): *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program $\delta$ in situation $s$ may lead to situation $s'$ with $\delta'$ remaining to be executed; and *(ii)* $Final(\delta, s)$, which holds if program $\delta$ may legally terminate in situation $s$. The definitions of $Trans$ and $Final$ we use are as in (De Giacomo, Lespérance, and Pearce 2010), where the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Predicate $Do(\delta, s, s')$ means that program $\delta$, when executed starting in situation $s$, has as a legal terminating situation $s'$, and is defined as $Do(\delta, s, s') \doteq \exists\delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$ where *Trans*$^*$ denotes the reflexive transitive closure of *Trans*. In the rest, we use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

## 3 Refinement Mappings

Suppose that we have a basic action theory $\mathcal{D}_l$ and another basic action theory $\mathcal{D}_h$. We would like to characterize whether $\mathcal{D}_h$ is a reasonable abstraction of $\mathcal{D}_l$. Here, we consider $\mathcal{D}_l$ as representing the *low-level* (or *concrete*)(LL) action theory/agent and $\mathcal{D}_h$ the *high-level* (or *abstract*)(HL) action theory/agent. We assume that $\mathcal{D}_h$ (resp. $\mathcal{D}_l$) involves a finite set of primitive action types $\mathcal{A}_h$ (resp. $\mathcal{A}_l$) and a finite set of primitive fluent predicates $\mathcal{F}_h$ (resp. $\mathcal{F}_l$). For simplicity, we assume that $\mathcal{D}_h$ and $\mathcal{D}_l$, share no domain specific symbols except for the set of standard names for objects $\mathcal{N}$.

We want to relate expressions in the language of $\mathcal{D}_h$ and expressions in the language of $\mathcal{D}_l$. We say that a function $m$ is a *refinement mapping* from $\mathcal{D}_h$ to $\mathcal{D}_l$ if and only if:

1. for every high-level primitive action type $A$ in $\mathcal{A}_h$, $m(A(\vec{x})) = \delta_A(\vec{x})$, where $\delta_A(\vec{x})$ is a ConGolog program over the low-level theory $\mathcal{D}_l$ whose only free variables are $\vec{x}$, the parameters of the high-level action type; intuitively, $\delta_A(\vec{x})$ represents how the high-level action $A(\vec{x})$ can be implemented at the low level; since we use programs to specify the action sequences the agent may perform, we require that $\delta_A(\vec{x})$ be situation-determined, i.e., the remaining program is always uniquely determined by the situation (De Giacomo, Lespérance, and Muise 2012);
2. for every high-level primitive fluent $F(\vec{x})$ (situation-suppressed) in $\mathcal{F}_h$, $m(F(\vec{x})) = \phi_F(\vec{x})$, where $\phi_F(\vec{x})$ is a situation-suppressed formula over the language of $\mathcal{D}_l$, and the only free variables are $\vec{x}$, the object parameters of the high-level fluent; intuitively $\phi_F(\vec{x})$ represents the *low-level condition* under which $F(\vec{x})$ holds in a situation.

Note that we can map a fluent in the high-level theory to a fluent in the low-level theory, i.e., $m(F_h(\vec{x})) = F_l(\vec{x})$, which effectively amounts to having the low-level fluent be present in the high-level theory. Similarly, one can include low-level actions in the high-level theory.

**Example** For our running example, we use a simple logistics domain. There is a shipment with ID 123 that is initially at a warehouse ($W$), and needs to be delivered to a Cafe ($Cf$), along a network of roads shown in Figure 1 (Warehouse and cafe images are from freedesignfile.com).

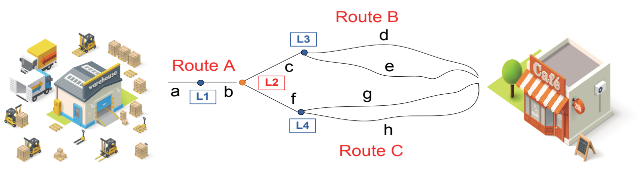

Figure 1: Transport Logistics Example

**High-Level BAT** $\mathcal{D}_h^{eg}$ At the high level, we abstract over navigation and delivery procedure details. We have actions that represent choices of major routes and delivering a shipment. $\mathcal{D}_h^{eg}$ includes the following precondition axioms (throughout the paper, we assume that free variables are universally quantified from the outside):

$$Poss(takeRoute(sID, r, o, d), s) \equiv o \neq d \wedge At_{HL}(sID, o, s) \\ \wedge CnRoute(r, o, d, s) \wedge (r = Rt_B \supset \neg Priority(sID, s))$$

$$Poss(deliver(sID), s) \equiv \exists l.Dest(sID, l, s) \wedge At_{HL}(sID, l, s)$$

The action $takeRoute(sID, r, o, d)$ can be performed to take shipment with ID $sID$ from origin location $o$ to destination location $d$ via route $r$ (see Figure 1), and is executable when the shipment is initially at $o$ and route $r$ connects $o$ to $d$; moreover, priority shipments cannot be sent by route $Rt_B$ (note that we refer to route X in Figure 1 as $Rt_X$). Action $deliver(sID)$ can be performed to deliver shipment $sID$ and is executable when $sID$ is at its destination.

The high-level BAT also includes the following SSAs:

$$At_{HL}(sID, l, do(a, s)) \equiv \exists l', r.a = takeRoute(sID, r, l', l) \vee \\ At_{HL}(sID, l, s) \wedge \forall l', r.a \neq takeRoute(sID, r, l, l')$$

$$Delivered(sID, do(a, s)) \equiv \\ a = deliver(sID) \vee Delivered(sID, s)$$

For the other fluents, we have SSAs specifying that they are unaffected by any action.

$\mathcal{D}_h^{eg}$ also contains the following initial state axioms:

$$Dest(123, Cf, S_0),\ At_{HL}(123, W, S_0), \\ CnRoute(Rt_A, W, L2, S_0),\ CnRoute(Rt_B, L2, Cf, S_0), \\ CnRoute(Rt_C, L2, Cf, S_0)$$

Note that it is not known whether 123 is a priority shipment.

**Low-Level BAT** $\mathcal{D}_l^{eg}$ At the low level, we model navigation and delivery in a more detailed way. The agent has a more detailed map with more locations and roads between them. He also takes road closures into account. Performing delivery involves unloading the shipment and getting a signature. The low-level BAT $\mathcal{D}_l^{eg}$ includes the following action precondition axioms:

$$Poss(takeRoad(sID, t, o, d), s) \equiv o \neq d \wedge \\ At_{LL}(sID, o, s) \wedge CnRoad(t, o, d, s) \wedge \neg Closed(t, s) \wedge \\ (d = L3 \supset \neg(BadWeather(s) \vee Express(sID, s)))$$

$$Poss(unload(sID), s) \equiv \exists l.Dest(sID, l, s) \wedge At_{LL}(sID, l, s)$$

$$Poss(getSignature(sID), s) \equiv Unloaded(sID, s)$$

Thus, the action $takeRoad(sID, t, o, d)$, where the agent takes shipment $sID$ from origin location $o$ to destination $d$ via road $t$, is executable provided that $t$ connects $o$ to $d$, $sID$ is at $o$, and $t$ is not closed; moreover, a road to $L3$ cannot be taken if the weather is bad or $sID$ is an express shipment as this would likely violate quality of service requirements.

The low-level BAT includes the following SSAs:

$$Unloaded(sID, do(a, s)) \equiv \\ a = unload(sID) \vee Unloaded(sID, s)$$

$$Signed(sID, do(a, s)) \equiv \\ a = getSignature(sID) \vee Signed(sID, s)$$

The SSA for $At_{LL}$ is like the one for $At_{HL}$ with $takeRoute$ replaced by $takeRoad$. For the other fluents, we have SSAs specifying that they are unaffected by any actions. Note that we could easily include exogenous actions for road closures and change in weather, new shipment orders, etc.

$\mathcal{D}_l^{eg}$ also contains the following initial state axioms:

$$\neg BadWeather(S_0),\ Closed(r, S_0) \equiv r = Rd_e, \\ Express(123, S_0),\ Dest(123, Cf, S_0),\ At_{LL}(123, W, S_0)$$

together with a complete specification of $CnRoad$ and $CnRoute$. We refer to road x in Figure 1 as $Rd_x$.

**Refinement Mapping** $m^{eg}$ We specify the relationship between the high-level and low-level BATs through the following refinement mapping $m^{eg}$:

$$m^{eg}(takeRoute(sID, r, o, d)) = \\ (r = Rt_A \wedge CnRoute(Rt_A, o, d))?; \\ \pi t.takeRoad(sID, t, o, L1); \pi t'.takeRoad(sID, t', L1, d) \mid \\ (r = Rt_B \wedge CnRoute(Rt_B, o, d))?; \\ \pi t.takeRoad(sID, t, o, L3); \pi t'.takeRoad(sID, t', L3, d) \mid \\ (r = Rt_C \wedge CnRoute(Rt_C, o, d))?; \\ \pi t.takeRoad(sID, t, o, L4); \pi t'.takeRoad(sID, t', L4, d)$$

$$m^{eg}(deliver(sID)) = unload(sID); getSignature(sID)$$
$$m^{eg}(Priority(sID)) = BadWeather \vee Express(sID)$$
$$m^{eg}(Delivered(sID)) = Unloaded(sID) \wedge Signed(sID)$$
$$m^{eg}(At_{HL}(sID, loc)) = At_{LL}(sID, loc)$$
$$m^{eg}(CnRoute(r, o, d)) = CnRoute(r, o, d)$$
$$m^{eg}(Dest(sID, l)) = Dest(sID, l)$$

Thus, taking route $Rt_A$ involves first taking a road from the origin $o$ to $L1$ and then taking another road from $L1$ to the destination $d$. For the other two routes, the refinement mapping is similar except a different intermediate location must be reached. Note that we could easily write programs to specify refinements for more complex routes, e.g., that take a sequence of roads from $o$ to $d$ going through intermediate locations belonging to a given set. We refine the high-level fluent $Priority(sID)$ to the condition where either the weather is bad or the shipment is express.

## 4 $m$-Bisimulation

To relate high-level and low-level models/theories, we resort to a suitable notion of bisimulation. Let $M_h$ be a model of the high-level BAT $\mathcal{D}_h$, $M_l$ a model of the low-level BAT $\mathcal{D}_l$, and $m$ a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$.

We first define a local condition for the bisimulation. We say that situation $s_h$ in $M_h$ is $m$-isomorphic to situation $s_l$ in $M_l$, written $s_h \sim_m^{M_h, M_l} s_l$, if and only if

> $M_h, v[s/s_h] \models F(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(F(\vec{x}))[s]$ for every high-level primitive fluent $F(\vec{x})$ in $\mathcal{F}_h$ and every variable assignment $v$ ($v[x/e]$ stands for the assignment that is like $v$ except that $x$ is mapped to $e$).

A relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ (where $\Delta_S^M$ stands for the situation domain of $M$) is an *$m$-bisimulation relation between $M_h$ and $M_l$* if $\langle s_h, s_l \rangle \in B$ implies that:

1. $s_h \sim_m^{M_h, M_l} s_l$, i.e., $s_h$ in $M_h$ is $m$-isomorphic to situation $s_l$ in $M_l$;
2. for every high-level primitive action type $A$ in $\mathcal{A}_h$, if there exists $s_h'$ such that $M_h, v[s/s_h, s'/s_h'] \models Poss(A(\vec{x}), s) \wedge s' = do(A(\vec{x}), s)$, then there exists $s_l'$ such that $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s, s')$ and $\langle s_h', s_l' \rangle \in B$;

3. for every high-level primitive action type $A$ in $\mathcal{A}_h$, if there exists $s'_l$ such that $M_l, v[s/s_l, s'/s'_l] \models Do(m(A(\vec{x})), s, s')$, then there exists $s'_h$ such that $M_h, v[s/s_h, s'/s'_h] \models Poss(A(\vec{x}), s) \land s' = do(A(\vec{x}), s)$ and $\langle s'_h, s'_l \rangle \in B$.

We say that $M_h$ *is bisimilar to* $M_l$ *relative to refinement mapping* $m$*, written* $M_h \sim_m M_l$, if and only if there exists an $m$-bisimulation relation $B$ between $M_h$ and $M_l$ such that $\langle S_0^{M_h}, S_0^{M_l} \rangle \in B$.

Given these definitions, we immediately get the following results. First, we can show that $m$-isomorphic situations satisfy the same high-level situation-suppressed formulas:

**Lemma 1** *If* $s_h \sim_m^{M_h, M_l} s_l$*, then for any high-level situation-suppressed formula* $\phi$*, we have that:*

$$M_h, v[s/s_h] \models \phi[s] \text{ if and only if } M_l, v[s/s_l] \models m(\phi)[s].$$

Note that $m(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula $\phi$ by $m(F(\vec{x}))$. For proofs of all our results, see (Banihashemi, De Giacomo, and Lespérance 2016).

Given this, it is straightforward to show that in $m$-bisimilar models, the same sequences of high-level actions are executable, and that in the resulting situations, the same high-level situation-suppressed formulas hold:

**Theorem 2** *If* $M_h \sim_m M_l$*, then for any sequence of ground high-level actions* $\vec{\alpha}$ *and any high-level situation-suppressed formula* $\phi$*, we have that*

$$M_l \models \exists s' Do(m(\vec{\alpha}), S_0, s') \land m(\phi)[s'] \quad \text{if and only if} \quad M_h \models Executable(do(\vec{\alpha}, S_0)) \land \phi[do(\vec{\alpha}, S_0)].$$

Note that $m(\alpha_1, \ldots, \alpha_n) \doteq m(\alpha_1); \ldots; m(\alpha_n)$ for $n \geq 1$ and $m(\epsilon) \doteq nil$.

## 5 Sound Abstraction

To ensure that the high-level theory is consistent with the low-level theory and mapping $m$, we require that for every model of the low-level theory, there is an $m$-bisimilar structure that is a model of the high-level theory.

We say that $\mathcal{D}_h$ is a *sound abstraction of* $\mathcal{D}_l$ *relative to refinement mapping* $m$ if and only if, for all models $M_l$ of $\mathcal{D}_l$, there exists a model $M_h$ of $\mathcal{D}_h$ such that $M_h \sim_m M_l$.

**Example** Returning to our example of Sec. 3, it is straightforward to show that it involves a high-level theory $\mathcal{D}_h^{eg}$ that is a sound abstraction of the low-level theory $\mathcal{D}_l^{eg}$ relative to the mapping $m^{eg}$. We discuss how we prove this later.

Sound abstractions have many interesting and useful properties. First, from the definition of sound abstraction and Theorem 2, we immediately get the following result:

**Corollary 3** *Suppose that* $\mathcal{D}_h$ *is a sound abstraction of* $\mathcal{D}_l$ *relative to mapping* $m$*. Then for any sequence of ground high-level actions* $\vec{\alpha}$ *and for any high-level situation-suppressed formula* $\phi$*, if* $\mathcal{D}_l \cup \mathcal{C} \cup \{\exists s. Do(m(\vec{\alpha}), S_0, s) \land m(\phi)[s]\}$ *is satisfiable, then* $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \land \phi[do(\vec{\alpha}, S_0)]\}$ *is also satisfiable. In particular, if* $\mathcal{D}_l \cup \mathcal{C} \cup \{\exists s. Do(m(\vec{\alpha}), S_0, s)\}$ *is satisfiable, then* $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\}$ *is also satisfiable.*

Thus if the low-level agent/theory thinks that a refinement of $\vec{\alpha}$ (perhaps involving exogenous actions) may occur (with $m(\phi)$ holding afterwards), the high-level agent/theory also thinks that $\vec{\alpha}$ may occur (with $\phi$ holding afterwards). If such a refinement actually occurs it will thus be consistent with the high-level theory.

We can also show that if the high-level theory entails that some sequence of high-level actions $\vec{\alpha}$ is executable, and that in the resulting situation, a situation-suppressed formula $\phi$ holds, then the low-level theory must also entail that some refinement of $\vec{\alpha}$ is executable and that in the resulting situation $m(\phi)$ holds:

**Theorem 4** *Suppose that* $\mathcal{D}_h$ *is a sound abstraction of* $\mathcal{D}_l$ *relative to mapping* $m$*. Then for any ground high-level action sequence* $\vec{\alpha}$ *and for any high-level situation-suppressed formula* $\phi$*, if* $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \land \phi[do(\vec{\alpha}, S_0)]$*, then* $\mathcal{D}_l \cup \mathcal{C} \models \exists s. Do(m(\vec{\alpha}), S_0, s) \land m(\phi)[s]$.

We can immediately relate the above result to *planning*. In the situation calculus, the planning problem is usually defined as follows (Reiter 2001):

> Given a BAT $\mathcal{D}$, and a situation-suppressed goal formula $\phi$, find a ground action sequence $\vec{a}$ such that $\mathcal{D} \models Executable(do(\vec{a}, S_0)) \land \phi[do(\vec{a}, S_0)]$.

Thus, Theorem 4 means that if we can find a plan $\vec{\alpha}$ to achieve a goal $\phi$ at the high level, i.e, $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \land \phi[do(\vec{\alpha}, S_0)]$, then it follows that there exists a refinement of $\vec{\alpha}$ that achieves $\phi$ at the low level, i.e., $\mathcal{D}_l \cup \mathcal{C} \models \exists s. Do(m(\vec{\alpha}), S_0, s) \land m(\phi)[s]$. However, note that the refinement could in general be different from model to model. But if, in addition, we have complete information at the low level, i.e., a single model for $\mathcal{D}_l$, then, since we have standard names for objects and actions, we can always obtain a plan to achieve the goal $\phi$ by finding a refinement in this way, i.e., there exists a ground low-level action sequence $\vec{a}$ such that $\mathcal{D}_l \cup \mathcal{C} \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, s)) \land m(\phi)[do(\vec{a}, s)]$. The search space of refinements of $\vec{\alpha}$ would typically be much smaller than the space of all low-level action sequences, thus yielding important efficiency benefits.

We can also show that if $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ with respect to a mapping, then the different sequences of low-level actions that are refinements of a given high-level primitive action sequence all have the same effects on the high-level fluents, and more generally on high-level situation-suppressed formulas, i.e., from the high-level perspective they are deterministic:

**Corollary 5** *If* $\mathcal{D}_h$ *is a sound abstraction of* $\mathcal{D}_l$ *relative to mapping* $m$*, then for any sequence of ground high-level actions* $\vec{\alpha}$ *and for any high-level situation-suppressed formula* $\phi$*, we have that*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s \forall s'. Do(m(\vec{\alpha}), S_0, s) \land Do(m(\vec{\alpha}), S_0, s') \supset (m(\phi)[s] \equiv m(\phi)[s'])$$

An immediate consequence of the above is the following:

**Corollary 6** *If* $\mathcal{D}_h$ *is a sound abstraction of* $\mathcal{D}_l$ *relative to mapping* $m$*, then for any sequence of ground high-level actions* $\vec{\alpha}$ *and for any high-level situation-suppressed formula* $\phi$*, we have that*

$$\mathcal{D}_l \cup \mathcal{C} \models (\exists s. Do(m(\vec{\alpha}), S_0, s) \land m(\phi)[s]) \supset (\forall s. Do(m(\vec{\alpha}), S_0, s) \supset m(\phi)[s])$$

It is also easy to show that if some refinement of the sequence of high-level actions $\vec{\alpha}\beta$ is executable, then there exists a refinement of $\beta$ that is executable after executing any refinement of $\vec{\alpha}$:

**Theorem 7** *If $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping $m$, then for any sequence of ground high-level actions $\vec{\alpha}$ and for any ground high-level action $\beta$, we have that*

$$\begin{aligned}\mathcal{D}_l \cup \mathcal{C} \models\ & \exists s. Do(m(\vec{\alpha}\beta), S_0, s) \supset \\ & (\forall s. Do(m(\vec{\alpha}), S_0, s) \supset \exists s'. Do(m(\beta), s, s'))\end{aligned}$$

Notice that this applies to all prefixes of $\vec{\alpha}$, so using Corollary 6 as well, we immediately get that:

**Corollary 8** *Suppose that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping $m$. Then for any ground high-level action sequence $\alpha_1, \dots, \alpha_n$, and for any high-level situation-suppressed formula $\phi$, then we have that:*

$$\begin{aligned}\mathcal{D}_l \cup \mathcal{C} \models\ & (\exists s. Do(m(\alpha_1, \dots, \alpha_n), S_0, s) \wedge m(\phi)[s]) \supset \\ & ((\forall s. Do(m(\alpha_1, \dots, \alpha_n), S_0, s) \supset m(\phi)[s]) \wedge \\ & (\exists s. Do(m(\alpha_1), S_0, s)) \wedge \\ & \textstyle\bigwedge_{2 \le i \le n} (\forall s. Do(m(\alpha_1, \dots, \alpha_{i-1}), S_0, s) \supset \\ & \qquad \exists s'. Do(m(\alpha_i), s, s')))\end{aligned}$$

These results mean that if a ground high-level action sequence achieves a high-level condition $\phi$, we can choose refinements of the actions in the sequence independently and be certain to obtain a refinement of the complete sequence that achieves $\phi$. We can exploit this in planning to gain even more efficiency. If we can find a plan $\alpha_1, \dots, \alpha_n$ to achieve a goal $\phi$ at the high level, then there exists a refinement of $\alpha_1, \dots, \alpha_n$ that achieves $\phi$ at the low level, and we can obtain it by finding refinements of the high-level actions $\alpha_i$ for $i : 1 \le i \le n$ one by one, without ever having to backtrack. The search space would typically be exponentially smaller in the length of the high-level plan $n$. If we have complete information at the low level, then we can always obtain a refined plan to achieve $\phi$ in this way.

**Example** Returning to our running example, we can show that the action sequence $\vec{\alpha} = [takeRoute(123, Rt_A, W, L2),\ takeRoute(123, Rt_C, L2, Cf), deliver(123)]$ is a valid high-level plan to achieve the goal $\phi_g = Delivered(123)$ of having delivered shipment 123, i.e., $\mathcal{D}_h^{eg} \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi_g[do(\vec{\alpha}, S_0)]$. Since $\mathcal{D}_h^{eg}$ is a sound abstraction of the low-level theory $\mathcal{D}_l^{eg}$ relative to the mapping $m^{eg}$, we know that we can find a refinement of the high-level plan $\vec{\alpha}$ that achieves the refinement of the goal $m^{eg}(\phi_g) = Unloaded(123) \wedge Signed(123)$. In fact, we can show that $\mathcal{D}_l^{eg} \cup \mathcal{C} \models Do(m^{eg}(\vec{\alpha}), S_0, do(\vec{a}\vec{b}\vec{c}, S_0)) \wedge m^{eg}(\phi_g)[do(\vec{a}\vec{b}\vec{c}, S_0)]$ for $\vec{a} = [takeRoad(123, Rd_a, W, L1), takeRoad(123, Rd_b, L1, L2)]$, $\vec{b} = [takeRoad(123, Rd_f, L2, L4), takeRoad(123, Rd_g, L4, Cf)]$, and $\vec{c} = [unload(123), getSignature(123)]$.

Now, let us define some low-level programs that characterize the refinements of high-level action/action sequences:

$any1hl \doteq |_{A_i \in \mathcal{A}_h}\ \pi\vec{x}. m(A_i(\vec{x}))$ , do any HL primitive action,

$anyseqhl \doteq any1hl^*$ i.e., do any sequence of HL actions.

How does one verify that one has a sound abstraction? The following yields a straightforward method for this:

**Theorem 9** *$\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to mapping $m$ if and only if*

**(a)** $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa} \models m(\phi)$*, for all* $\phi \in D^h_{S_0}$,

**(b)** $\mathcal{D}^l \cup \mathcal{C} \models \forall s. Do(anyseqhl, S_0, s) \supset$
$\quad \bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}. (m(\phi^{Poss}_{A_i}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$,

**(c)** $\mathcal{D}^l \cup \mathcal{C} \models \forall s. Do(anyseqhl, S_0, s) \supset$
$\quad \bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'. (Do(m(A_i(\vec{x})), s, s') \supset$
$\qquad \bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y} (m(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

*where $\phi^{Poss}_{A_i}(\vec{x})$ is the right hand side (RHS) of the precondition axiom for action $A_i(\vec{x})$, and $\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x})$ is the RHS of the successor state axiom for $F_i$ instantiated with action $A_i(\vec{x})$ where action terms have been eliminated using $\mathcal{D}^h_{ca}$.*

The above essentially gives us a "proof theoretic" characterization of sound abstraction. Conditions (a), (b), and (c) are all properties that standard verification techniques can deal with. The theorem also means that if $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ wrt $m$, then $\mathcal{D}^l$ must entail the mapped high-level successor state axioms and entail that the mapped conditions for a high-level action to be executable (from the precondition axioms of $\mathcal{D}^h$) correctly capture the executability conditions of their refinements (these conditions must hold after any sequence of refinements of high-level actions, i.e., in any situation $s$ where $Do(anyseqhl, S_0, s)$ holds).

**Example** Returning to our running example, it is straightforward to show that it involves a high-level theory $\mathcal{D}_h^{eg}$ that is a sound abstraction of the low-level theory $\mathcal{D}_l^{eg}$ relative to the mapping $m^{eg}$. $\mathcal{D}_l^{S_0}$ entails the "translation" of all the facts about the high-level fluents $CnRoute$, $Dest$ and $At_{HL}$ that are in $\mathcal{D}_h^{S_0}$. Moreover, $\mathcal{D}_l^{eg}$ entails that the mapping of the preconditions of the high-level actions $deliver$ and $takeRoute$ correctly capture the executability conditions of their refinements. $\mathcal{D}_l^{eg}$ also entails the mapped high-level successor state axiom for fluent $At_{HL}$ and action $takeRoute$ and similarly for $Delivered$ and action $deliver$ (other actions have no effects). Thus, $\mathcal{D}_h^{eg}$ is a sound abstraction of $\mathcal{D}_l^{eg}$ relative to $m^{eg}$.

## 6 Complete Abstraction

When we have a sound abstraction $\mathcal{D}_h$ of a low-level theory $\mathcal{D}_l$ wrt a mapping $m$, the high-level theory $\mathcal{D}_h$'s conclusions are always sound wrt the more refined theory $\mathcal{D}_l$, but $\mathcal{D}_h$ may have less information than $\mathcal{D}_l$ regarding high-level actions and conditions. $\mathcal{D}_h$ may consider it possible that a high-level action sequence is executable (and achieves a goal) when $\mathcal{D}_l$ knows it is not. The low-level theory may know/entail that a refinement of a high-level action sequence achieves a goal without the high level knowing/entailing it. We can define a stronger notion of abstraction that ensures that the high-level theory knows everything that the low-level theory knows about high-level actions and conditions.

We say that $\mathcal{D}_h$ is a *complete abstraction of $\mathcal{D}_l$ relative to refinement mapping $m$* if and only if, for all models $M_h$ of $\mathcal{D}_h$, there exists a model $M_l$ of $\mathcal{D}_l$ such that $M_l \sim_m M_h$.

From the definition of complete abstraction and Theorem 2, we immediately get the following converses of Corollary 3 and Theorem 4:

**Corollary 10** *Suppose that $\mathcal{D}_h$ is a complete abstraction of $\mathcal{D}_l$ relative to $m$. Then for any sequence of ground high-level actions $\vec{\alpha}$ and for any high-level situation-suppressed formula $\phi$, if $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]\}$ is satisfiable, then $\mathcal{D}_l \cup \mathcal{C} \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]\}$ is satisfiable. In particular, if $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\}$ is satisfiable, $\mathcal{D}_l \cup \mathcal{C} \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s)\}$ is satisfiable.*

**Theorem 11** *Suppose that $\mathcal{D}_h$ is a complete abstraction of $\mathcal{D}_l$ relative to mapping $m$. Then for any ground high-level action sequence $\vec{\alpha}$ and any high-level situation-suppressed formula $\phi$, if $\mathcal{D}_l \cup \mathcal{C} \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$, then $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$.*

Thus when we have a high-level theory $\mathcal{D}_h$ that is a complete abstraction of a low-level theory $\mathcal{D}_l$ wrt a mapping $m$, if $\mathcal{D}_l$ knows/entails that some refinement of a high-level action sequence $\vec{\alpha}$ achieves a high-level goal $\phi$, then $\mathcal{D}_h$ knows/entails that $\vec{\alpha}$ achieves $\phi$, i.e., we can find all high-level plans to achieve high-level goals using $\mathcal{D}_h$.

Note that with complete abstraction alone, we don't get Corollary 5, as $\mathcal{D}_l \cup \mathcal{C}$ may have models that are not $m$-bisimilar to any model of $\mathcal{D}_h$ and where different refinements of a high-level action yield different truth-values for $m(F)$, for some high-level fluent $F$.

We also say that $\mathcal{D}_h$ is a *sound and complete abstraction of $\mathcal{D}_l$ relative to refinement mapping $m$* if and only if $\mathcal{D}_h$ is both a sound and a complete abstraction of $\mathcal{D}_l$ relative to $m$.

**Example** Returning to our running example, the high-level theory does not know whether shipment 123 is high priority, i.e., $\mathcal{D}_h^{eg} \not\models Priority(123)[S_0]$ and $\mathcal{D}_h^{eg} \not\models \neg Priority(123)[S_0]$, but the low-level theory knows that it is, i.e., $\mathcal{D}_l^{eg} \models m^{eg}(Priority(123))[S_0]$. Thus $\mathcal{D}_h^{eg}$ has a model where $\neg Priority(123)[S_0]$ holds that is not $m^{eg}$-bisimilar to any model of $\mathcal{D}_l^{eg}$, and thus $\mathcal{D}_h^{eg}$ is a sound abstraction of $\mathcal{D}_l^{eg}$ wrt $m^{eg}$, but not a complete abstraction. For instance, the high-level theory considers it possible that the shipment can be delivered by taking route A and then route B, i.e., $\mathcal{D}_h^{eg} \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi_g[do(\vec{\alpha}, S_0)]\}$ is satisfiable for $\vec{\alpha} = [takeRoute(123, Rt_A, W, L2), takeRoute(123, Rt_B, L2, Cf), deliver(123)]$ and $\phi_g = Delivered(123)$. But the low-level theory knows that $\vec{\alpha}$ cannot be refined to an executable low-level plan, i.e., $\mathcal{D}_l^{eg} \cup \mathcal{C} \models \neg\exists s.Do(m^{eg}(\vec{\alpha}), S_0, s)$. If we add $Priority(123)[S_0]$ and a complete specification of $CnRoute$ to $\mathcal{D}_h^{eg}$, then it becomes a sound and complete abstraction of $\mathcal{D}_l^{eg}$ wrt $m^{eg}$. The plan $\vec{\alpha}$ is now ruled out as $\mathcal{D}_h^{eg} \cup \{Priority(123, S_0)\} \cup \{Executable(do(\vec{\alpha}, S_0))\}$ is not satisfiable.

The following results characterize complete abstractions:

**Theorem 12** *If $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to mapping $m$, then $\mathcal{D}^h$ is also a complete abstraction of $\mathcal{D}^l$ wrt mapping $m$ if and only if for every model $M_h$ of $\mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ca} \cup \mathcal{D}^h_{coa}$, there exists a model $M_l$ of $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa}$ such that $S_0^{M_h} \sim_m^{M_h, M_l} S_0^{M_l}$.*

**Theorem 13** *$\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ relative to mapping $m$ iff for every model $M_h$ of $\mathcal{D}^h$, there exists a model $M_l$ of $\mathcal{D}^l \cup \mathcal{C}$ such that $S_0^{M_h} \sim_m^{M_h, M_l} S_0^{M_l}$ and*
*$M_l \models \forall s.Do(anyseqhl, S_0, s) \supset$*
*$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$*
*and $M_l \models \forall s.Do(anyseqhl, S_0, s) \supset$*
*$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$*
*$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i, A_i}^{ssa}(\vec{y}, \vec{x}))[s] \quad \equiv \quad m(F_i(\vec{y}))[s']))$,*
*where $\phi_{A_i}^{Poss}(\vec{x})$ and $\phi_{F_i, A_i}^{ssa}(\vec{y}, \vec{x})$ are as in Theorem 9.*

## 7 Monitoring and Explanation

A refinement mapping $m$ from a high-level action theory $\mathcal{D}_h$ to a low-level action theory $\mathcal{D}_l$ tells us what are the refinements of high-level actions into executions of low-level programs. In some application contexts, one is interested in tracking/monitoring what the low-level agent is doing and describing it in abstract terms, e.g., to a client or manager. If we have a ground low-level situation term $S_l$ such that $\mathcal{D}_l \cup \{Executable(S_l)\}$ is satisfiable, and $\mathcal{D}_l \cup \{Do(m(\vec{\alpha}), S_0, S_l)\}$ is satisfiable, then the ground high-level action sequence $\vec{\alpha}$ is a possible way of describing in abstract terms what has occurred in getting to situation $S_l$. If $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\}$ is also satisfiable (it must be if $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt $m$), then one can also answer high-level queries about what may hold in the resulting situation, i.e., whether $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi(do(\vec{\alpha}, S_0))\}$ is satisfiable, and what must hold in such a resulting situation, i.e., whether $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\} \models \phi(do(\vec{\alpha}, S_0))$. One can also answer queries about what high-level action $\beta$ might occur next, i.e., whether $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}\beta, S_0))\}$ is satisfiable.

In general, there may be several distinct ground high-level action sequences $\vec{\alpha}$ that match a ground low-level situation term $S_l$; even if we have complete information and a single model $M_l$ of $\mathcal{D}_l \cup \mathcal{C}$, there may be several distinct $\vec{\alpha}$'s such that $M_l \models Do(m(\vec{\alpha}), S_0, S_l)$.[1]

In many contexts, this would be counterintuitive and we would like to be able to map a sequence of low-level actions performed by the low-level agent back into a *unique* abstract high-level action sequence it refines, i.e., we would like to define an inverse mapping *function* $m^{-1}$. Let's see how we can do this. First, we introduce the abbreviation $lp_m(s, s')$, meaning that $s'$ is a largest prefix of $s$ that can be produced by executing a sequence of high-level actions:

$$\begin{aligned} lp_m(s, s') \doteq\ & Do(anyseqhl, S_0, s') \wedge\ s' \leq s \wedge \\ & \forall s''.(s' < s'' \leq s \supset \neg Do(anyseqhl, S_0, s'')) \end{aligned}$$

We can show that the relation $lp_m(s, s')$ is actually a total function:

**Theorem 14** *For any refinement mapping $m$ from $\mathcal{D}_h$ to $\mathcal{D}_l$, we have that:*

1. *$D_l \cup \mathcal{C} \models \forall s. \exists s'.lp_m(s, s')$,*
2. *$D_l \cup \mathcal{C} \models \forall s \forall s_1 \forall s_2.lp_m(s, s_1) \wedge lp_m(s, s_2) \supset s_1 = s_2$.*

Given this result, we can introduce the notation $lp_m(s) = s'$ to stand for $lp_m(s, s')$.

[1]For example, suppose that we have two high level actions $A$ and $B$ with $m(A) = (C \mid D)$ and $m(B) = (D \mid E)$. Then the low-level situation $do(D, S_0)$ is a refinement of both $A$ and $B$ (assuming all actions are always executable).

To be able to map a low-level action sequence back to a unique high-level action sequence that produced it, we need the following assumptions:

**Assumption 1** *For any distinct ground high-level action terms $\alpha$ and $\alpha'$ we have that:*

**(a)** $\mathcal{D}_l \cup \mathcal{C} \models Do(m(\alpha), s, s') \supset \neg\exists\delta.Trans^*(m(\alpha'), s, \delta, s')$

**(b)** $\mathcal{D}_l \cup \mathcal{C} \models Do(m(\alpha), s, s') \supset \neg\exists a \exists\delta.Trans^*(m(\alpha), s, \delta, do(a, s'))$

**(c)** $\mathcal{D}_l \cup \mathcal{C} \models Do(m(\alpha), s, s') \supset s < s'$

Part $(a)$ ensures that different high-level primitive actions have disjoint sets of refinements; $(b)$ ensures that once a refinement of a high-level primitive action is complete, it cannot be extended further; and $(c)$ ensures that a refinement of a high-level primitive action will produce at least one low-level action. Together, these three conditions ensure that if we have a low-level action sequence that can be produced by executing some high-level action sequence, there is a unique high-level action sequence that can produce it:

**Theorem 15** *Suppose that we have a refinement mapping $m$ from $\mathcal{D}_h$ to $\mathcal{D}_l$ and that Assumption 1 holds. Let $M_l$ be a model of $\mathcal{D}_l \cup \mathcal{C}$. Then for any ground situation terms $S_s$ and $S_e$ such that $M_l \models Do(anyseqhl, S_s, S_e)$, there exists a unique ground high-level action sequence $\vec{\alpha}$ such that $M_l \models Do(m(\vec{\alpha}), S_s, S_e)$.*

Since in any model $M_l$ of $\mathcal{D}_l \cup \mathcal{C}$, for any ground situation term $S$, there is a unique largest prefix of $S$ that can be produced by executing a sequence of high-level actions, $S' = lp_m(S)$, and for any such $S'$, there is a unique ground high-level action sequence $\vec{\alpha}$ that can produce it, we can view $\vec{\alpha}$ as the value of the inverse mapping $m^{-1}$ for $S$ in $M_l$. For this, let us introduce the following notation:

$$m^{-1}_{M_l}(S) = \vec{\alpha} \doteq M_l \models \exists s'.lp_m(S) = s' \wedge Do(m(\vec{\alpha}), S_0, s')$$

where $m$ is a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$ and Assumption 1 holds, $M_l$ is a model of $\mathcal{D}_l \cup \mathcal{C}$, $S$ is a ground low-level situation term, and $\vec{\alpha}$ is a ground high-level action sequence.

Assumption 1 however does not ensure that any low-level situation $S$ can in fact be generated by executing a refinement of some high-level action sequence; if it cannot, than the inverse mapping will not return a complete matching high-level action sequence (e.g., we might have $m^{-1}_{M_l}(S) = \epsilon$). We can introduce an additional assumption that rules this out:[2]

**Assumption 2**
$D_l \cup \mathcal{C} \models \forall s.Executable(s) \supset \exists\delta.Trans^*(anyseqhl, S_0, \delta, s)$

With this additional assumption, we can show that for any executable low-level situation $s$, what remains after the largest prefix that can be produced by executing a sequence of high-level actions, i.e, the actions in the interval between $s'$ and $s$ where $lp_m(s, s')$, can be generated by some (not yet complete) refinement of a high-level primitive action:

**Theorem 16** *If $m$ is a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$ and Assumption 2 holds, then we have that:*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'.Executable(s) \wedge lp_m(s, s') \supset \exists\delta.Trans^*(any1hl, s', \delta, s)$$

**Example** Going back to the example of Section 3, assume that we have complete information at the low level and a single model $M_l$ of $\mathcal{D}_l^{eg}$, and suppose that the sequence of (executable) low-level actions $\vec{a} = [takeRoad(123, Rd_a, W, L1), takeRoad(123, Rd_b, L1, L2)]$ has occurred. The inverse mapping allows us to conclude that the high-level action $\alpha = takeRoute(123, Rt_A, W, L2)$ has occurred, since $m^{-1}_{M_L}(do(\vec{a}, S_0)) = \alpha$.[3] Since $\mathcal{D}_h \models At_{HL}(123, L2, do(\alpha, S_0))$, we can also conclude that shipment 123 is now at location $L2$. As well, since $\mathcal{D}_h \cup \{Poss(takeRoute(123, Rt_B, L2, Cf), do(\alpha, S_0))\}$ is satisfiable, we can conclude that high-level action $takeRoute(123, Rt_B, L2, Cf)$ might occur next. Analogously, we can also conclude that high-level action $takeRoute(123, Rt_C, L2, Cf)$ might occur next.

## 8 Discussion

In AI, (Giunchiglia and Walsh 1992) formalize abstraction as *syntactic* mappings between formulas of a concrete and a more abstract representation, while (Nayak and Levy 1995) present a *semantic* theory of abstraction. These approaches formalize abstraction of *static* logical theories, while our work focuses on abstraction of *dynamic* domains. In planning, several notions of abstraction have been investigated. These include *precondition-elimination abstraction*, first introduced in the context of ABSTRIPS (Sacerdoti 1974); *Hierarchical Task Networks* (HTNs) (e.g. (Erol, Hendler, and Nau 1996)), which abstract over a set of (non-primitive) tasks; and *macro operators* (e.g. (Korf 1987)), which represent meta-actions built from a sequence of action steps. (Aguas, Celorrio, and Jonsson 2016) proposed hierarchical finite state controllers for planning. Encodings of HTNs in ConGolog with enhanced features like exogenous actions and online executions (Gabaldon 2002) and preferred plans (Sohrabi and McIlraith 2008) have also been studied. (McIlraith and Fadel 2002) and (Baier and McIlraith 2006) instead, investigate planning with *complex actions* (a form of macro actions) specified as Golog (Levesque et al. 1997) programs. While these approaches focus on improving the efficiency of planning, our work provides a generic framework which can have applications in many areas. Moreover, (Gabaldon 2002; Sohrabi and McIlraith 2008) use a single BAT, and (McIlraith and Fadel 2002; Baier and McIlraith 2006) compile the abstracted actions

[2]One might prefer a weaker version of Assumption 2. For instance, one could write a program specifying the low level agent's possible behaviors and require that situations reachable by executing this program can be generated by executing a refinement of some high-level action sequence. We discuss the use of programs to specify possible agent behaviors in the conclusion.

[3]If we do not have complete information at the low level, $m^{-1}_{M}(\vec{a})$ may be different for different models $M$ of $\mathcal{D}_l$. To do high level tracking/monitoring in such cases, we need to consider all the possible mappings or impose additional restrictions to ensure that there is a unique mapping. We leave this problem for future work.

into a new BAT that contains both the original and abstracted actions. Also, they only deal with deterministic complex actions and do not provide abstraction for fluents. Moreover, our approach provides a refinement mapping (similar to that of Global-As-View in data integration (Lenzerini 2002)) between an abstract BAT and a concrete BAT.

In this paper, we proposed a general framework for agent abstraction based on the situation calculus and ConGolog. For simplicity, we focused on a single layer of abstraction, but the framework supports extending the hierarchy to more levels. Our approach can also support the use of ConGolog programs to specify the possible behaviors of the agent at both the high and low level, as we can follow (De Giacomo et al. 2016) and "compile" the program into the BAT $\mathcal{D}$ to get a new BAT $\mathcal{D}'$ whose executable situations are exactly those that can be reached by executing the program. We also identified a set of BAT constraints that ensure that for any low-level action sequence, there is a unique high-level action sequence that it refines. This is useful for providing high-level explanations of behavior and monitoring.

In future work, we will explore how using different types of mappings and BATs from various sources that yield sound/complete abstractions can support system evolvability. Moreover, we will investigate methodologies for designing abstract agents/theories and refinement mappings wrt given objectives, as well as automated synthesis techniques to support this. We will also explore how agent abstraction can be used in verification, where there is some related work (Mo, Li, and Liu 2016; Belardinelli, Lomuscio, and Michaliszyn 2016). Extending our agent abstraction framework to accommodate sensing and online executions is another direction for future research.

## Acknowledgments

We acknowledge the support of Sapienza 2015 project "Immersive Cognitive Environments" and the National Science and Engineering Research Council of Canada.

## References

Aguas, J. S.; Celorrio, S. J.; and Jonsson, A. 2016. Hierarchical finite state controllers for generalized planning. In *Proc. of the 25th International Joint Conference on Artificial Intelligence*, 3235–3241.

Baier, J. A., and McIlraith, S. A. 2006. On planning with programs that sense. In *Proc. of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, 492–502.

Banihashemi, B.; De Giacomo, G.; and Lespérance, Y. 2016. Abstraction in situation calculus action theories - Extended version. Technical Report EECS-2016-04, York University.

Belardinelli, F.; Lomuscio, A.; and Michaliszyn, J. 2016. Agent-based refinement for predicate abstraction of multi-agent systems. In *Proc. of the 22nd European Conference on Artificial Intelligence*, 286–294.

Clarke, E. M.; Grumberg, O.; and Long, D. E. 1994. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* 16(5):1512–1542.

De Giacomo, G.; Lespérance, Y.; Patrizi, F.; and Sardiña, S. 2016. Verifying ConGolog programs on bounded situation calculus theories. In *Proc. of the 13th AAAI Conference on Artificial Intelligence*, 950–9568.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In *Proc. of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 1031–1038.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation calculus based programs for representing and reasoning about game structures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 12th International Conference*.

Erol, K.; Hendler, J. A.; and Nau, D. S. 1996. Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence* 18(1):69–93.

Gabaldon, A. 2002. Programming hierarchical task networks in the situation calculus. In *AIPS02 Workshop on On-line Planning and Scheduling*.

Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. *Artificial Intelligence* 5(2):323–389.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Lenzerini, M. 2002. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 233–246.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programing* 31(1-3):59–83.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence* 4:463–502.

McIlraith, S. A., and Fadel, R. 2002. Planning with complex actions. In *Proc. of the 9th International Workshop on Non-Monotonic Reasoning*, 356–364.

Milner, R. 1971. An algebraic definition of simulation between programs. In *Proc. of the 2nd International Joint Conference on Artificial Intelligence*, 481–489.

Milner, R. 1989. *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

Mo, P.; Li, N.; and Liu, Y. 2016. Automatic verification of Golog programs via predicate abstraction. In *Proc. of the 22nd European Conference on Artificial Intelligence*, 760–768.

Nau, D.; Ghallab, M.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Nayak, P. P., and Levy, A. Y. 1995. A semantic theory of abstractions. In *Proc. of the 14th International Joint Conference on Artificial Intelligence*, 196–203.

Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

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

Saitta, L., and Zucker, J.-D. 2013. *Abstraction in Artificial Intelligence and Complex Systems*. Springer-Verlag New York.

Sohrabi, S., and McIlraith, S. A. 2008. On planning with preferences in HTN. In *Proc. of the 12th International Workshop on Non-Monotonic Reasoning*, 241–248.