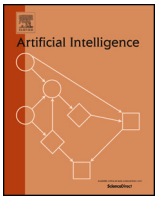
Artificial Intelligence 348 (2025) 104407

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Abstracting situation calculus action theories

Bita Banihashemi [a,*], Giuseppe De Giacomo [b], Yves Lespérance [c]

[a] IGDORE, Gothenburg, Sweden
[b] University of Oxford, Oxford, United Kingdom
[c] York University, Toronto, Canada

ARTICLE INFO

Keywords:
Knowledge representation and reasoning
Reasoning about action and change
Situation calculus
Abstraction
Autonomous agents

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. To facilitate verifying that one has a sound/complete abstraction relative to a mapping, we provide a set of necessary and sufficient conditions. Finally, we identify a set of basic action theory constraints that ensure that for any low-level action sequence, there is a unique high-level action sequence that it refines. This allows us to track/monitor what the low-level agent is doing and describe it in abstract terms (i.e., provide high-level explanations, for instance, to a client or manager).

1. Introduction

Autonomous 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* [77]. 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 [63,75] and the ConGolog [27] 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)

* Corresponding author.
E-mail addresses: bita.banihashemi@igdore.org (B. Banihashemi), giuseppe.degiacomo@cs.ox.ac.uk (G. De Giacomo), lesperan@eecs.yorku.ca (Y. Lespérance).

https://doi.org/10.1016/j.artint.2025.104407
Received 29 October 2024; Received in revised form 22 July 2025; Accepted 18 August 2025

Available online 1 September 2025
0004-3702/© 2025 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

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 [66,67] relative to the mapping between the models of the low-level and high-level theories. To have a sound abstraction, every model of the low-level theory must be bisimilar to some model of the high-level theory relative to the mapping. In essence, a high-level model is bisimilar to a low-level model relative to the mapping if all executions of high-level actions at the high level can be simulated at the low level by executions of their refinements and vice versa. 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 exists 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 as well. 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.

In addition, we define a dual notion of a high-level theory being a *complete abstraction* of a low-level theory under a refinement mapping. This requires that every model of the high-level theory must be bisimilar to some model of the low-level theory relative to the mapping. With a complete abstraction, we have that 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.

We also provide a set of necessary and sufficient conditions for having a sound and/or complete abstraction relative to a mapping. These essentially say that the low-level models ensure that the executability of refinements of high-level actions and their effects on high-level fluents are as specified by the high-level theory. These necessary and sufficient conditions can be used to verify that one has a sound/complete abstraction.

Finally, we identify a set of basic action theory constraints that ensure that for any low-level action sequence, there is a unique high-level action sequence that it refines. This allows us to track/monitor what the low-level agent is doing and describe it in abstract terms (i.e., provide high-level explanations [35]) e.g., to a client or manager. This can have applications in *Explainable AI*.

In the past, many different approaches to abstraction have been proposed in a variety of settings such as planning [76,36,51], automated reasoning [44,70], model checking [20], and data integration [53]. In [44] abstraction is formalized as syntactic mappings between formulas of a concrete and a more abstract first-order theories, while [70] present a semantic theory of abstraction. In model checking, abstraction techniques are widely used to deal with very large state spaces (e.g., [19]). In planning, several notions of abstraction have been investigated. These include precondition-elimination abstraction, first introduced in the context of ABSTRIPS [76], Hierarchical Task Networks (HTNs) (e.g., [36]), which abstract over a set of (non-primitive) tasks, macro operators (e.g., [51]), which represent meta-actions built from a sequence of action steps, and generalized planning (e.g., [56]) where planning solutions generalize over a set of planning instances. With the exception of work on abstraction in planning, these approaches do not deal with dynamic domains. Previous work on abstraction in planning focuses on the planning task and often incorporates important representational restrictions [40,41]. In contrast, our approach provides a generic framework that can be applied to different reasoning tasks in dynamic domains represented in an expressive first-order framework. We discuss related work in more details in Section 8.

This work revises and extends [6]. We add some new results (e.g., Lemma 5 and Corollary 22), provide proofs, include a comprehensive survey of related work, and discuss work that has extended and adapted our abstraction framework since publication in [6]. We have also expanded the preliminaries section and cover the ConGolog Semantics in the appendix.

The paper is organized as follows. In the next section, we review the basics of the situation calculus and ConGolog. Then in Section 3, we define a notion of refinement mapping between a high-level and a low-level basic action theory. Section 4 introduces our notion of bisimulation with respect to a mapping that relates models at the abstract and concrete levels. Then in Sections 5 and 6, we define the notions of sound and complete abstractions respectively, showing how they allow the abstract theory to be exploited in reasoning. In Section 7, we discuss how our framework can be used in monitoring what the low-level agent is doing and explaining it in abstract terms. This is followed by a discussion of related work in Section 8. In Section 9 we provide an overview of some of the adaptations and extensions to our abstraction framework. Finally in Section 10, we conclude by summarizing our main contributions and discussing future work.

2. Preliminaries

In this section, we provide an overview of the situation calculus [63,75], basic action theories [73,75], and the high-level programming language ConGolog [27].

The situation calculus and basic action theories. The *situation calculus* is a well known predicate logic language for representing and reasoning about dynamically changing worlds [63,75]. All changes to the world are the result of *actions*, which are terms in the language. A possible world history is represented by a term called a *situation*. The constant S_0 is used to denote the initial situation. Sequences of actions are built using the function symbol do, such that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates and functions whose value varies from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument. For example, we may have that $Door1$ is not open in the initial situation S_0, i.e., $\neg IsOpen(Door1, S_0)$ holds, but is open in the situation that results from doing the action $open(Door1)$ in S_0, i.e.,

$IsOpen(Door1, do(open(Door1), S_0)$ holds. $s \sqsubset s'$ means that s is a predecessor situation of s', and $s \sqsubseteq s'$ stands for $s = s' \vee s \sqsubset s'$. The abbreviation $do([a_1, \ldots, a_n], s)$ stands for $do(a_n, do(a_{n-1}, \ldots, do(a_1, s) \ldots))$; also for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$.

Within this language, one can formulate action theories that describe how the world changes as a result of the available actions. Here, we concentrate on *basic action theories* as proposed in [73,75]. We also 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.[1] 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 $\mathcal{D}$ is the union of the following disjoint sets of first-order (FO) and second-order (SO) axioms:

- $\mathcal{D}_{S_0}$: (FO) *initial situation description axioms* describing the initial configuration of the world (such a description may be complete or incomplete);
- $\mathcal{D}_{poss}$: (FO) *precondition axioms* of the form

  $$Poss(A(\vec{x}), s) \equiv \phi_A^{Poss}(\vec{x}, s),$$

  one per action type, stating the conditions $\phi_A^{Poss}(\vec{x}, s)$ under which an action $A(\vec{x})$ can be legally performed in situation s; these use a special predicate $Poss(a, s)$ meaning that action a is executable in situation s; $\phi_A(\vec{x}, s)$ is a formula of the situation calculus that is uniform in s[2];
- $\mathcal{D}_{ssa}$: (FO) *successor state axioms* of the form

  $$F(\vec{x}, do(a, s)) \equiv \phi_F^{ssa}(\vec{x}, a, s),$$

  one per fluent, describing how the fluent changes when an action is performed; the right-hand side (RHS) $\phi_F^{ssa}(\vec{x}, a, s)$ is again a situation calculus formula uniform in s; successor state axioms encode the causal laws of the world being modeled; they take the place of the so-called effect axioms and provide a solution to the frame problem;
- $\mathcal{D}_{ca}$: (FO) unique name axioms for actions and (FO) domain closure on action types;
- $\mathcal{D}_{coa}$: (SO) unique name axioms and domain closure for object constants in $\mathcal{N}$;
- Σ: (SO) foundational, domain independent, axioms of the situation calculus [73].

The abbreviation $Executable(s)$ is used to denote that every action performed in reaching situation s was possible in the situation in which it occurred. When executability of situations is taken into consideration, we use $<$ instead of $\sqsubset$ to indicate precedence on situations; consequently, $s \leq s'$ indicates that s' is a successor situation of s and that every action between s and s' is in fact executable.

A key feature of BATs is the existence of a sound and complete *regression mechanism* for answering queries about situations resulting from performing a sequence of actions [73,75]. In a nutshell, the regression operator $\mathcal{R}^*$ reduces a formula ϕ about a particular future situation to an equivalent formula $\mathcal{R}^*[\phi]$ about the initial situation S_0, essentially by substituting fluent relations with the right-hand side formula of their successor state axioms. Another key result about BATs is the relative satisfiability theorem [73,75]: $\mathcal{D}$ is *satisfiable* if and only if $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$ is satisfiable, the latter being a purely first-order theory (here, $\mathcal{D}_{una}$ is the set of unique names axioms for actions). This implies that we can check if a regressable formula ϕ is entailed by $\mathcal{D}$, by checking if its regression $\mathcal{R}^*[\phi]$ is entailed by $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$ only.

High-level programs. 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, starting with Golog [57], and subsequently ConGolog [27], an extension that supports concurrent processes. Here we concentrate on (a fragment of) ConGolog that includes the following constructs:

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

In the above, α is an action term, possibly with parameters, and φ 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 φ by restoring the situation argument s into all fluents in φ. The test action $\varphi?$ checks if condition φ holds in the current situation. The sequence of program δ_1 followed by program δ_2 is denoted by $\delta_1; \delta_2$. Program $\delta_1 | \delta_2$ allows for the nondeterministic choice between programs δ_1 and δ_2, while $\pi x.\delta$ executes program δ for *some* nondeterministic choice of a legal binding for variable x (observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times. Program $\delta_1 \| \delta_2$ expresses the concurrent execution (interpreted as interleaving) of programs δ_1 and δ_2. We also use *nil*, an abbreviation for $True?$, to represent the *empty program*, i.e., when nothing remains to be performed.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates [27]: *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining to be executed; and

[1] This makes it easier to relate the high-level and low-level action theories. One of the main consequences of assuming standard names is that quantification can be considered substitutionally; for instance, $\exists x.P(x)$ is true just in case $P(n)$ is true for some standard name n.

[2] A formula of the situation calculus is *uniform* in s if and only if it does not mention the predicates $Poss$ or $\sqsubset$, it does not quantify over variables of sort situation, it does not mention equality on situations, and whenever it mentions a term of sort situation in the situation argument position of a fluent, then that term is s [75].

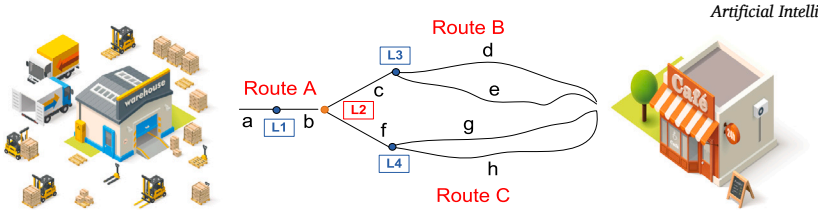


Fig. 1. Transport Logistics Example.

(ii) $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ we use are as in [32], where the test construct $\varphi?$ does not yield any transition, but is final when satisfied (see Appendix B for details). Predicate $Do(\delta, s, s')$ means that program δ, 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.

We say that ConGolog program δ is *situation-determined* (SD) in a situation s [29] if and only if for every sequence of transitions, the remaining program is determined by the resulting situation, i.e.,

$$SituationDetermined(\delta, s) \doteq \forall s', \delta', \delta''.$$
$$Trans^*(\delta, s, \delta', s') \wedge Trans^*(\delta, s, \delta'', s') \supset \delta' = \delta''.$$

For example, program $(a; b) \mid (a; c)$ is not SD, while $a; (b \mid c)$ is (assuming the actions involved are always executable). Thus, a (partial) execution of a SD program is uniquely determined by the sequence of actions it has produced.

3. Refinement mappings

In this section, we define a notion of refinement mapping between a high-level action theory and a low-level action theory. We also introduce a running example which illustrates the notion.

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* (LL) (or *concrete*) action theory/agent and $\mathcal{D}_h$ the *high-level* (HL) (or *abstract*) 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 $\mathcal{D}_l$.

Definition 1 *(Refinement Mapping).* 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;
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.

We can extend the mapping to an arbitrary high-level situation-suppressed formula ϕ by taking $m(\phi)$ to be the result of substituting every fluent $F(\vec{x})$ in ϕ by $m(F(\vec{x}))$. Also, we can extend the mapping to sequences of high-level actions by taking: $m(\alpha_1, \ldots, \alpha_n) \doteq m(\alpha_1); \ldots; m(\alpha_n)$ for $n \geq 1$ and $m(\epsilon) \doteq nil$.

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 Fig. 1.[3]

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.

First we have to specify the action preconditions. 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 Fig. 1), and is executable when the shipment is initially at o and route

[3] Warehouse and cafe images are from freedesignfile.com.

r connects o to d; moreover, priority shipments cannot be sent by route Rt_B (note that we refer to route X in Fig. 1 as Rt_X). Action $deliver(sID)$ can be performed to deliver shipment sID and is executable when sID is at its destination. To specify this, $\mathcal{D}_h^{eg}$ includes the following precondition axioms (throughout the paper, we assume that free variables are universally quantified from the outside):

$$Poss(takeRoute(sID, r, o, d), s) \equiv o \neq d \wedge At_{HL}(sID, o, s) \wedge CnRoute_{HL}(r, o, d, s) \wedge (r = Rt_B \supset \neg Priority(sID, s))$$
$$Poss(deliver(sID), s) \equiv \exists l. Dest_{HL}(sID, l, s) \wedge At_{HL}(sID, l, s).$$

$\mathcal{D}_h^{eg}$ also includes the following successor state axioms (SSAs) to specify how the domain state changes:

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

The first axiom says that shipment sID is at location l after performing action a in situation s if and only if, either action a was the $takeRoute$ action which moved the shipment from a previous location l' to location l via route r, or, the shipment was already at location l in situation s, and no $takeRoute$ action was performed that would move the shipment to a new location l' through any routes. The second axiom says that shipment sID is delivered after performing action a in situation s if and only if, either action a was the $deliver$ action which delivered the shipment, or, the shipment was already delivered. For the other fluents, we have SSAs specifying that they are unaffected by any action.

In the initial situation, the shipment 123 is at the warehouse, and the destination is the cafe. Route Rt_A connects the warehouse to location $L2$, and routes Rt_B and Rt_C connect location $L2$ to the cafe. Note that it is not known whether 123 is a priority shipment. Thus $\mathcal{D}_h^{eg}$ also contains the following initial state axioms:

$$Dest_{HL}(123, Cf, S_0), \quad At_{HL}(123, W, S_0),$$
$$CnRoute_{HL}(Rt_A, W, L2, S_0), \quad CnRoute_{HL}(Rt_B, L2, Cf, S_0),$$
$$CnRoute_{HL}(Rt_C, L2, Cf, S_0).$$

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.

Thus we have 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 action $unload(sID)$ to unload shipment sID can be performed if the agent is at the shipment's destination. Action $getSignature(sID)$ is executable if sID has been unloaded. To specify this, the low-level BAT $\mathcal{D}_l^{eg}$ includes the following action precondition axioms:

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

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:

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

together with a complete specification of $CnRoad$ and $CnRoute_{LL}$ as in Fig. 1. We refer to road x in the figure as Rd_x.

Refinement mapping m^{eg}. We specify the relationship between the high-level and low-level BATs through the refinement mapping m^{eg} given below. Under mapping m^{eg}, the high-level action of taking route Rt_A is refined to first taking a road from the origin o to location $L1$ 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 traversed. 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. The high-level action $deliver(sID)$ is mapped to a sequence of two low-level actions, first unload the shipment sID and then get

the recipient's signature for it. We refine the high-level fluent $Priority(sID)$ to the condition where either the weather is bad or the shipment is express. The high-level fluent $Delivered(sID)$ is mapped to a conjunction: the shipment has been unloaded and signed for.

$$
\begin{aligned}
&m^{eg}(takeRoute(sID, r, o, d)) = \\
&\quad (r = Rt_A \wedge CnRoute_{LL}(Rt_A, o, d))?; \\
&\quad \pi t.takeRoad(sID, t, o, L1); \pi t'.takeRoad(sID, t', L1, d) \mid \\
&\quad (r = Rt_B \wedge CnRoute_{LL}(Rt_B, o, d))?; \\
&\quad \pi t.takeRoad(sID, t, o, L3); \pi t'.takeRoad(sID, t', L3, d) \mid \\
&\quad (r = Rt_C \wedge CnRoute_{LL}(Rt_C, o, d))?; \\
&\quad \pi t.takeRoad(sID, t, o, L4); \pi t'.takeRoad(sID, t', L4, d) \\
&m^{eg}(deliver(sID)) = unload(sID); getSignature(sID) \\
&m^{eg}(Priority(sID)) = BadWeather \vee Express(sID) \\
&m^{eg}(Delivered(sID)) = Unloaded(sID) \wedge Signed(sID) \\
&m^{eg}(At_{HL}(sID, loc)) = At_{LL}(sID, loc) \\
&m^{eg}(CnRoute_{HL}(r, o, d)) = CnRoute_{LL}(r, o, d) \\
&m^{eg}(Dest_{HL}(sID, l)) = Dest_{LL}(sID, l). \quad \square
\end{aligned}
$$

4. *m*-bisimulation

In this section, we define our notion of bisimulation with respect to a refinement mapping, which relates models at the abstract and concrete levels. We also prove some basic properties of models that are bisimilar relative to a mapping. For some background on bisimulation, see Section 8.

Let M_h be a model of the high-level BAT $\mathcal{D}_h$, M_l a model of the low-level BAT $\mathcal{D}_l \cup C$, and m a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$.

We first define a local condition for the bisimulation.

Definition 2 *(m-isomorphism).* We say that situation s_h in M_h is m-isomorphic to situation s_l in M_l, written $s_h \simeq_m^{M_h, M_l} s_l$, if and only if

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

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), i.e., s_h and s_l interpret all high-level fluents the same.

Definition 3 *(m-bisimulation).* A relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ (where Δ_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. $s_h \simeq_m^{M_h, M_l} s_l$, 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 $\langle s_h', s_l' \rangle \in B$, i.e., if $A(\vec{x})$ is executable in the high-level model at s_h, then the program that implements $A(\vec{x})$ must be executable in the low-level model at s_l, and the resulting pair of situations s_h' and s_l' must be bisimilar;
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 $M_h, v[s/s_h, s'/s_h'] \models Poss(A(\vec{x}), s) \wedge s' = do(A(\vec{x}), s)$ and $\langle s_h', s_l' \rangle \in B$, i.e., if the program that implements $A(\vec{x})$ is executable in the low-level model at s_l, then $A(\vec{x})$ must be executable in the high-level model at s_h, and the resulting pair of situations s_h' and s_l' must be bisimilar.

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

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[4]:

Lemma 4. *If $s_h \simeq_m^{M_h, M_l} s_l$, then for any high-level situation-suppressed formula ϕ, we have that:*

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

[4] For proofs of all our results, see Appendix A.

We can also show that in m-bisimilar models, the same sequences of high-level actions are executable, and that the resulting situations are m-bisimilar:

Lemma 5. *If $M_h \sim_m M_l$, then for any sequence of high-level actions $\vec{\alpha}$, we have that*

if $M_l, v[s'/s_l] \models Do(m(\vec{\alpha}), S_0, s')$, then there exists s_h such that
$M_h, v[s'/s_h] \models s' = do(\vec{\alpha}, S_0) \wedge Executable(s')$ and $s_h \sim_m^{M_h, M_l} s_l$

and

if $M_h, v[s'/s_h] \models s_h = do(\vec{\alpha}, S_0) \wedge Executable(s_h)$,
then there exists s_l such that $M_l, v[s'/s_l] \models Do(m(\vec{\alpha}), S_0, s')$ and $s_h \sim_m^{M_h, M_l} s_l$.

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

Theorem 6. *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 ϕ, we have that*

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

5. Sound abstraction

This section defines our notion of a high-level BAT being a sound abstraction of a low-level BAT relative to a mapping, shows some useful properties of sound abstractions that facilitate reasoning at the high level and refining the results at the low level, and identifies necessary and sufficient conditions for having a sound abstraction.

When we have a high-level action theory $\mathcal{D}_h$ that is a sound abstraction of a low-level theory $\mathcal{D}_l$ with respect to a mapping m, it should be the case that the high-level theory $\mathcal{D}_h$'s conclusions about when high-level actions are executable and what effects they achieve are correct with respect to when their refinements are executable and what high-level effects they achieve at the low level. If we had this, it would allow us to reason with the high-level theory and get conclusions that are correct with respect to the low level. To get this, the high level theory $\mathcal{D}_h$ should be consistent with the low-level theory $\mathcal{D}_l$ regarding executions of high-level actions and their effects on high-level fluents relative to the mapping m. To ensure this, 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.

Definition 7 *(Sound Abstraction).* We say that $\mathcal{D}_h$ is a *sound abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for every model M_l of $\mathcal{D}_l \cup C$, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$.

Example Cont. Returning to our example of Section 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 6, we immediately get the following result, which states that 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 ϕ holding afterwards). If we observe that such a refinement actually occurs it will thus be consistent with the high-level theory.

Corollary 8. *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 ϕ, if $\mathcal{D}_l \cup C \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]\}$ is satisfiable, then $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]\}$ is also satisfiable. In particular, if $\mathcal{D}_l \cup C \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s)\}$ is satisfiable, then $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\}$ is also satisfiable.*

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 ϕ 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 9. *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 ϕ, if $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$, then $\mathcal{D}_l \cup C \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$.*

We can immediately relate the above result to *planning*. In the situation calculus, the planning problem is usually defined as follows [75]:

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

Thus, Theorem 9 means that if we can find a plan $\vec{\alpha}$ to achieve a goal ϕ at the high level, i.e., $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$, then it follows that there exists a refinement of $\vec{\alpha}$ that achieves ϕ at the low level, i.e., $\mathcal{D}_l \cup C \models \exists s. Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$. 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 and domain closure for objects and actions, we can always obtain a plan to achieve the goal ϕ by finding a refinement in this way, i.e., there exists a ground low-level action sequence $\vec{a}$ such that $\mathcal{D}_l \cup C \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, S_0)) \wedge m(\phi)[do(\vec{a}, S_0)]$. 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 10. *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 ϕ, we have that*

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

An immediate consequence of the above is the following:

Corollary 11. *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 ϕ, we have that*

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

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 β that is executable after executing any refinement of $\vec{\alpha}$:

Theorem 12. *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 β, we have that*

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

Notice that this applies to all prefixes of $\vec{\alpha}$, so using Corollary 11 as well, we immediately get that:

Corollary 13. *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 $\alpha_1, \ldots, \alpha_n$, and for any high-level situation-suppressed formula ϕ, then we have that:*

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

These results mean that if a ground high-level action sequence achieves a high-level condition ϕ, we can choose refinements of the actions in the sequence independently and be certain to obtain a refinement of the complete sequence that achieves $m(\phi)$. We can exploit this in planning to gain even more efficiency. If we can find a plan $\alpha_1, \ldots, \alpha_n$ to achieve a goal ϕ at the high level, then there exists a refinement of $\alpha_1, \ldots, \alpha_n$ that achieves $m(\phi)$ at the low level, and we can obtain it by finding refinements of the high-level actions α_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 $m(\phi)$ in this way.

Example Cont. In 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 $\mathcal{D}_l^{eg} \cup C \models Do(m^{eg}(\vec{\alpha}), S_0, do(\vec{a}\vec{b}\vec{c}, S_0)) \wedge m^{eg}(\phi_g)[do(\vec{a}\vec{b}\vec{c}, S_0)]$ 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:

$\text{ANY1HL} \doteq |_{A_i \in \mathcal{A}_h} \pi \vec{x}.m(A_i(\vec{x}))$
i.e., do any refinement of any one HL primitive action,

$\text{ANYSEQHL} \doteq \text{ANY1HL}^*$
i.e., do any sequence of refinements of HL actions.

How does one verify that one has a sound abstraction? The following result identifies necessary and sufficient conditions for having a sound abstraction:

Theorem 14. *$\mathcal{D}^h$ 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* $\phi \in \mathcal{D}^h_{S_0}$,
(b) $\mathcal{D}^l \cup C \models \forall s.Do(\textit{ANYSEQHL}, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi^{Poss}_{A_i}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$,
(c) $\mathcal{D}^l \cup C \models \forall s.Do(\textit{ANYSEQHL}, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi^{ssa}_{F_i,A_i}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

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 provides us with a way of showing that we have a sound abstraction by proving that properties (a), (b), and (c) are entailed by the low-level theory. That is, $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ with respect to m if and only if (a) the low-level initial state axioms together with the unique name and domain closure axioms $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa}$ entail the mapped high-level initial state axioms in $\mathcal{D}^h_{S_0}$, (b) the low-level theory entails that after any sequence of refinements of high-level actions (i.e., in any situation s where $Do(\text{ANYSEQHL}, S_0, s)$ holds), 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, and (c) the low-level theory entails that after any sequence of refinements of high-level actions, the mapped high-level successor state axioms capture how any refinement of a high-level action affects the mapped high-level fluents.

In the general first-order case, verifying that one has a sound abstraction is undecidable and requires using theorem proving techniques. One possible approach to doing this is to use Theorem 14 and try to verify properties (a), (b), and (c). Note that while (a) is a first-order property, (b) and (c) are essentially properties of programs. We need to show that the right hand side (RHS) holds in any situation produced by an arbitrary sequence of refinements of high-level actions. One can try to identify state constraints that hold for all such situations and use them to prove the RHS. In (b), the RHS essentially involves the weakest precondition for a refinement of the high-level action. Similarly in (c), the RHS essentially involves the weakest condition for a refinement of the high-level action to make the fluent hold afterwards (for the left to right direction, and similarly for the converse). There are well known techniques to try to compute these (see [61] for a formalization of regression over Golog programs).

A notable special case is when one has a *finite object domain* and *complete information about the initial state* in the low-level action theory, i.e., a unique model M^l of $\mathcal{D}^l$. Then one can use model checking techniques [4] to do the verification. One approach would be to use Theorem 14 and try to model check the three properties in the theorem. An alternative approach would be to use model checking techniques to directly check that the high-level and low-level models are m-bisimilar.

Returning to our running example, it is straightforward to show that it involves a high-level theory $\mathcal{D}^{eg}_h$ that is a sound abstraction of the low-level theory $\mathcal{D}^{eg}_l$ relative to the mapping m^{eg}:

Proposition 15. *$\mathcal{D}^{eg}_h$ is a sound abstraction of $\mathcal{D}^{eg}_l$ wrt m^{eg}.*

$\mathcal{D}^{S_0}_l$ entails the "translation" of all the facts about the high-level fluents $CnRoute_{HL}$, $Dest_{HL}$ and At_{HL} that are in $\mathcal{D}^{S_0}_h$. For instance, in the high level theory, we have that $\mathcal{D}^{eg}_h \models Dest_{HL}(123, Cf, S_0)$. We have that $Dest_{HL}(sID)$ is mapped to $Dest_{LL}(sID, l)$ and furthermore, in the low-level theory, we have that $\mathcal{D}^{eg}_l \models Dest_{LL}(123, Cf, S_0)$. Moreover, $\mathcal{D}^{eg}_l$ entails that the mapping of the preconditions of the high-level actions $deliver$ and $takeRoute$ correctly capture the executability conditions of their refinements. For example, high-level action $deliver(sID)$ is mapped to the program $unload(sID)$; $getSignature(sID)$ in the low-level theory. Also, the precondition of $deliver(sID)$ maps to $\exists l.Dest_{LL}(sID, l, s) \wedge At_{LL}(sID, l, s)$ which is the precondition for of $unload(sID)$ and this in turn ensures the precondition for action $getSignature(sID)$. $\mathcal{D}^{eg}_l$ also entails that the mapped high-level successor state axioms hold at the low level for refinements of high-level actions. For instance, consider the high-level fluent $Delivered(sID, s')$, which is only affected by action $deliver(sID)$. Given that $deliver(sID)$ is mapped to the program $unload(sID); getSignature(sID)$ and $Delivered(sID)$ maps to $Unloaded(sID) \wedge Signed(sID)$ in the low-level theory, we can use the successor state axioms of $Unloaded(sID, s')$ and $Signed(sID, s')$ to show that the result holds. For high-level fluent At_{HL}, which is only affected by the $takeRoute$ action, the proof is similar. Thus, $\mathcal{D}^{eg}_h$ is a sound abstraction of $\mathcal{D}^{eg}_l$ relative to m^{eg}. For more details, see the proof in Appendix A. □

6. Complete abstraction

This section defines a notion of a high-level BAT being a complete abstraction of a low-level BAT relative to a mapping, shows some useful properties of complete abstractions, and identifies necessary and sufficient conditions for having a complete abstraction.

When we have a sound abstraction $\mathcal{D}_h$ of a low-level theory $\mathcal{D}_l$ with respect to a mapping m, the high-level theory $\mathcal{D}_h$'s conclusions are always sound with respect to 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.

Definition 16 *(Complete Abstraction).* We say that $\mathcal{D}_h$ is a *complete abstraction of* $\mathcal{D}_l$ *relative to refinement mapping* m if and only if, for every model M_h of $\mathcal{D}_h$, there exists a model M_l of $\mathcal{D}_l \cup C$ such that $M_l \sim_m M_h$.

From the definition of complete abstraction and Theorem 6, we immediately get the following converses of Corollary 8 and Theorem 9:

Corollary 17. *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 ϕ, if $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]\}$ is satisfiable, then $\mathcal{D}_l \cup C \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]\}$ is satisfiable. In particular, if $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0))\}$ is satisfiable, then $\mathcal{D}_l \cup C \cup \{\exists s.Do(m(\vec{\alpha}), S_0, s)\}$ is satisfiable.*

Theorem 18. *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 ϕ, if $\mathcal{D}_l \cup C \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$, 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 $\mathcal{D}_l$ with respect to a mapping m, if $\mathcal{D}_l$ knows/entails that some refinement of a high-level action sequence $\vec{\alpha}$ achieves the refinement of a high-level goal ϕ, then $\mathcal{D}_h$ knows/entails that $\vec{\alpha}$ achieves ϕ. It follows that we can find all high-level plans to achieve high-level goals using $\mathcal{D}_h$.

Note as well that with a complete abstraction that is not a sound abstraction, we no longer get that high-level actions are deterministic at the low level with respect to high-level fluents, i.e., Corollary 10; this happens because $\mathcal{D}_l \cup C$ 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.

Complete abstractions can constrain the search space that is used in automated reasoning (e.g., planning) and thus speed up finding solutions to problems. But if we don't have soundness, we need to check that the result actually holds at the low level.

Definition 19 *(Sound and Complete Abstraction).* We 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 Cont. Returning to our running example, the high-level theory does not know whether shipment 123 is high priority, i.e., $\mathcal{D}_h^{eg} \nvDash Priority(123)[S_0]$ and $\mathcal{D}_h^{eg} \nvDash \neg Priority(123)[S_0]$, 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 $\mathcal{D}_l^{eg}$ with respect to 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., $\mathcal{D}_l^{eg} \cup C \models \neg\exists s.Do(m^{eg}(\vec{\alpha}), S_0, s)$. If we add $Priority(123)[S_0]$ and a complete specification of $CnRoute_{HL}$ to $\mathcal{D}_h^{eg}$, then it becomes a sound and complete abstraction of $\mathcal{D}_l^{eg}$ with respect to 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. □

We can show that if we have a sound abstraction relative to a mapping m, then the abstraction is also complete if and only if for every model of the initial situation at the high level, there is a m-isomorphic model at the low level, thus ensuring that the high-level knows as much as the low level about the high-level fluents in initial situation:

Theorem 20. *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 $\mathcal{D}^l$ with respect to 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 $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$.*

This result provides a method to verify that one has a sound and complete abstraction.

We also have the following result that characterizes complete (but not necessarily sound) abstractions:

Theorem 21. *$\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ relative to mapping m if and only if for every model M_h of $\mathcal{D}^h$, there exists a model M_l of $\mathcal{D}^l \cup C$ such that*

(a) $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$,
(b) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$,
(c) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

where $\phi_{A_i}^{Poss}(\vec{x})$ *and* $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ *are as in Theorem 14.*

The conditions in the theorem are necessary and sufficient to ensure that M_h and M_l are m-bisimilar; see the proof in Appendix A for details.

Example. Let $\mathcal{D}^h$ be

$$Poss(A, s) \equiv True$$
$$Poss(B, s) \equiv False$$
$$P(do(a, s)) \equiv a = A \vee P(s)$$
$$Q(do(a, s)) \equiv Q(s)$$
$$R(do(a, s)) \equiv R(s)$$
$$\neg P(S_0) \wedge \neg Q(S_0) \wedge \neg R(S_0),$$

and let $\mathcal{D}^l$ be

$$Poss(A, s) \equiv True$$
$$Poss(B, s) \equiv R(s)$$
$$P(do(a, s)) \equiv a = A \vee P(s)$$
$$Q(do(a, s)) \equiv Q(s)$$
$$R(do(a, s)) \equiv (a = A \wedge Q(s)) \vee R(s)$$
$$\neg P(S_0) \wedge \neg R(S_0),$$

with the mapping m being

$$m(A) = A, m(B) = B$$
$$m(P) = P, m(Q) = Q, m(R) = R.$$

Then, $\mathcal{D}^h$ is a complete but not sound abstraction of $\mathcal{D}^l$ wrt m. $\mathcal{D}^h$ has a single model (up to isomorphism) M_h where Q is false initially. We can show that $\mathcal{D}^l$ has a model M_l that is m-bisimilar to M_h. We have $M_l \models R(do(a, s)) \equiv R(s)$ because Q remains false in all situations in M_l. Thus R also remains false in all situations in M_l. It follows that $M_l \models Poss(B, s) \equiv False$. $\mathcal{D}^h$ is not a sound abstraction of $\mathcal{D}^l$ wrt m because $\mathcal{D}^l$ also has a model M_l' where Q is true initially, and $\mathcal{D}^h$ has no such model. We have that $M_l' \not\models R(do(a, s)) \equiv R(s)$ and $M_l' \not\models Poss(B, s) \equiv False$. □

For the special case where $\mathcal{D}_{S_0}^h$ is a complete theory, we also have the following result:

Corollary 22. *If* $\mathcal{D}_{S_0}^h$ *is a complete theory (i.e., for any situation suppressed formula* ϕ*, either* $\mathcal{D}_{S_0}^h \models \phi[S_0]$ *or* $\mathcal{D}_{S_0}^h \models \neg\phi[S_0]$*) and* $\mathcal{D}^l$ *is satisfiable, then if* $\mathcal{D}^h$ *is a sound abstraction of* $\mathcal{D}^l$ *with respect to* m*, then* $\mathcal{D}^h$ *is also a complete abstraction of* $\mathcal{D}^l$ *with respect to* m*.*

7. Monitoring and explanation

In this section, we identify constraints that allow a low-level action sequence to be mapped back into a unique high-level action sequence that it refines, which allows one to monitor the agent's low level behavior and describe it in abstract terms.

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 $\mathcal{D}_l$ with respect to 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 β 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 $\mathcal{D}_l \cup C$, i.e., we may have $M_l \models Do(m(\vec{\alpha}_1), S_0, S_l) \wedge Do(m(\vec{\alpha}_2), S_0, S_l)$ and $\mathcal{D}_h \models \alpha_1 \neq \alpha_2$.[5]

In many contexts, this would be counter-intuitive 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*:

$$lp_m(s, s') \doteq Do(\text{ANYSEQHL}, S_0, s') \wedge s' \leq s \wedge \\ \forall s''.(s' < s'' \leq s \supset \neg Do(\text{ANYSEQHL}, S_0, s'')).$$

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

Theorem 23. *For any refinement mapping m from $\mathcal{D}_h$ to $\mathcal{D}_l$, we have that:*

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

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

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

Constraint 1. *For any distinct ground high-level action terms α and α' we have that:*

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

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 24. *Suppose that we have a refinement mapping m from $\mathcal{D}_h$ to $\mathcal{D}_l$ and that Constraint 1 holds. Let M_l be a model of $\mathcal{D}_l \cup C$. Then for any ground situation terms S_s and S_e such that $M_l \models Do(\text{ANYSEQHL}, S_s, S_e)$, 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 $\mathcal{D}_l \cup C$, 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 Constraint 1 holds, M_l is a model of $\mathcal{D}_l \cup C$, S is a ground low-level situation term, and $\vec{\alpha}$ is a ground high-level action sequence.

Constraint 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, then 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 constraint that rules this out[6]:

Constraint 2.

$$D_l \cup C \models \forall s.Executable(s) \supset \exists\delta.Trans^*(\text{ANYSEQHL}, S_0, \delta, s).$$

[5] 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).

[6] One might prefer a weaker version of Constraint 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.

With this additional constraint, 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 25. *If m is a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$ and constraint 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 Cont. 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 $m_{M_l}^{-1}(do(\vec{a}, S_0)) = \alpha$.[7] Since $\mathcal{D}_h^{eg} \models At_{HL}(123, L2, do(\alpha, S_0))$, we can also conclude that shipment 123 is now at location $L2$. As well, since $\mathcal{D}_h^{eg} \cup \{Poss(takeRoute(123, Rt_B, L2, Cf), do(\alpha, S_0))\}$ 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. However, since $\mathcal{D}_h^{eg} \models \neg At_{HL}(123, Cf, do(\alpha, S_0))$, i.e., agent's current location is not the cafe, the high-level action $Deliver(123)$ is not executable and cannot occur next. □

8. Related work

In this section, we review related work in the areas of abstraction of static logical theories, planning, and model checking. We also briefly discuss previous work involving bisimulation.

Given the importance of abstraction in AI, it is unsurprising that there has been much previous work on the topic. Among logical theories of abstraction, Plaisted [74] is perhaps the first to propose a general theory of abstraction focused on theorem proving (and in particular on resolution) in a first-order language. He proposed a notion of abstraction as a mapping from a set of clauses B to a simpler (i.e., more abstract) set of clauses A that satisfied certain properties with respect to the resolution mechanism. Resolution proofs from B map onto (possibly) simpler resolution proofs in A. Plaisted introduced different types of syntactic abstraction mappings, including *renaming predicate and function symbols*, where several predicates (resp. functions) could be renamed to the same predicate (resp. function) in the abstract clause. Plaisted was aware that this approach could yield an abstract theory that produces inferences that are not sanctioned by the base theory,[8] and called this issue the "false proof" problem; in fact one might even create an inconsistent abstract theory from a consistent base theory.

Tenenberg [85] focused on *predicate mapping* abstractions, which can be considered as a special case of Plaisted's *renaming predicate and function symbols* mapping abstractions. To ensure that the abstract theory remains consistent, he proposed *restricted predicate mappings*, where in essence, the clauses from the base theory that distinguish the predicates being conflated are not mapped. More precisely, this is achieved by requiring that for each clause C in the abstract level, there exists a predicate (or clause) D in the concrete level clause set such that D maps into C and either C is a positive clause or for every D that maps to such C it is the case that D can be derived from the low-level clause set.

Giunchiglia and Walsh [44] consider abstractions as *syntactic* mappings between two representations of a problem modeled as axiomatic formal systems. A formal system is defined as a triple $\Sigma = (L, \Theta, \Delta)$, where Θ is a set of axioms in the language L, and Δ is the deductive machinery of Σ (i.e., set of inference rules). An abstraction $f : \Sigma_{base} \Rightarrow \Sigma_{abs}$ is defined as a mapping between formal systems Σ_{base} (i.e., the *ground* space) and Σ_{abs} (i.e., the *abstract* space), with languages L_{base} and L_{abs}, respectively, and an effective, total function $f_L : L_{base} \rightarrow L_{abs}$ which is referred to as the *mapping function*.

This approach classifies abstractions based on their *effects on provability*; that is, whether the set of theorems of the high-level theory $TH(\Sigma_{abs})$ are equal to, a subset, or a superset of the set of theorems of the low-level theory $TH(\Sigma_{base})$. An abstraction $f : \Sigma_{base} \Rightarrow \Sigma_{abs}$ is said to be (i) a theorem-increasing (TI) abstraction if and only if, for any well formed formula (wff) α, if $\alpha \in TH(\Sigma_{base})$ then $f_L(\alpha) \in TH(\Sigma_{abs})$, (ii) a theorem-decreasing (TD) abstraction if and only if, for any wff α, if $f_L(\alpha) \in TH(\Sigma_{abs})$ then $\alpha \in TH(\Sigma_{base})$, and (iii) a theorem-constant (TC) abstraction if it is both a TI and a TD abstraction. It is clear that our notions of sound abstraction, complete abstraction, and sound and complete abstraction are related to Giunchiglia and Walsh's notions of TD abstraction, TI abstraction, and TC abstraction respectively, as they have similar properties with respect to theorems/entailments.

Giunchiglia and Walsh [44] take certain subclasses of TI abstractions to be useful, especially in applications where the abstract space is used to help find a proof in the ground space. For instance, ABSTRIPS [76] uses a TI abstractions that admit false proofs and are used as heuristics to guide search to speed up problem solving. [44] reconstructed a number of previous approaches to abstraction in their framework and analyzed their properties; examples include planning with ABSTRIPS [76], common sense reasoning [46], as well as the predicate abstractions of Plaisted and Tenenberg.

Nayak and Levy [70] point out that while the syntactic theory of abstraction proposed in [44] captures important aspects of different types of abstractions, and enables their use in reasoning by theorem provers, such a theory fails to explicitly capture the

[7] If we do not have complete information at the low level, $m_M^{-1}(\vec{a})$ may be different for different models M of 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.

[8] For example, if the base theory contains $JapaneseCar(x) \supset Reliable(x)$ and $EuropeanCar(x) \supset Fast(s)$, and we rename both $JapaneseCar$ and $EuropeanCar$ to $ForeignCar$, we may unjustifiably infer that a $EuropeanCar$ instance is $Reliable$ (see [85,70]).

underlying justifications or assumptions that lead to the abstraction. Hence [70] proposed a *semantic theory* of abstraction which focuses on a *mapping between the models* of concrete and abstract theories. Abstraction is viewed as a two-step process: in the first step, the intended domain model is abstracted; and in the second step, a set of (abstract) formulas is constructed that capture the abstracted domain model. In general, an *abstraction mapping* is a function π that maps a model/interpretation M_{base} of a theory T_{base} in the base language L_{base} to an interpretation $\pi(M_{base})$ of the abstract language L_{abs}. For first-order languages, an abstraction mapping can be specified by giving formulas of the base language that define the universe and the denotation of predicate and function symbols of $\pi(M_{base})$ in terms of that in M_{base}: (i) a wff $\pi_\forall(x)$ which defines the universe of $\pi(M_{base})$ as the set of entities that satisfy $\pi_\forall(x)$ in M_{base}, (ii) a wff $\pi_R(x_1,\ldots,x_n)$ (with n free variables) for each n-ary abstract relation R, where the denotation of R in $\pi(M_{base})$ is the relation defined by $\pi_R(x_1,\ldots,x_n)$ in M_{base} (the set of entity n-tuples that satisfy it) restricted to universe of $\pi(M_{base})$, and (iii) similar wffs that specify the denotations of abstract function symbols and constants. Nayak and Levy also define the class of model increasing (MI) abstractions, which is a strict subset of TD abstractions and yield no false proofs. Moreover, as an MI abstraction can be weaker than the intended model-level abstraction, they also define the notion of *strongest* MI abstraction of a base theory. This would yield an abstract theory that *precisely* implements the intended model-level abstraction. Notice that Nayak and Levy's notion of MI abstraction is closely related to our notion of sound abstraction and their notion of strongest MI abstraction is related to our notion of sound and complete abstraction.

Ghidini and Giunchiglia [43] propose a semantic formalization of the notion of abstraction based on the Local Models Semantics [42]. They associate to each of the concrete and abstract languages a set of interpretations (i.e., context). The abstraction mapping is formalized as a *compatibility relation* that defines how meaning is preserved when moving from the concrete to abstract representations.

In Model theory [47,84], an interpretation of a structure A_1 in another structure A_2 (whose signature may be unrelated to A_1) is a well established notion that approximates the idea of representing A_1 inside A_2. Starting with A_2, it is possible to build A_1 by defining the domain D_1 of A_1 as well as all the labeled relations and functions of A_1 as relations definable in A_2 by certain formulas (with parameters). A further refinement involves finding a definable equivalence relation on D_1 and considering the domain of A_1 to be set of equivalence classes of this relation. A typical example is the interpretation of the group $\mathbb{Z}$ of integers in the structure $\mathbb{N}$ which consists of natural numbers $0, 1, 2, \ldots$ with labels for $0, 1$ and $+$. Interpreting structure A_1 in a structure A_2 results in the ability to translate every first-order statement about A_1 into a first-order statement about A_2; this means that all of A_1 can be read from A_2. If it is possible to generalize this notion to interpreting a family of models of a theory T_2, always using the same defining formulas, then the resulting structures will all be models of a theory T_1 that can be read from T_2 and the defining formulas. Then we can say that theory T_1 is interpreted in theory T_2. This also shows that T_1 is reducible to T_2.

Our work is clearly distinct from all of the approaches discussed above, which formalize abstraction of *static* logical theories. We instead focus on abstraction of *dynamic* domains. In our approach we have assumed that the high-level and the low-level have the same domain/universe; this is mainly for simplicity and we leave exploring object abstraction for future work. As mentioned, Nayak and Levy [70] handle object abstraction by introducing a special formula that designates which of the low-level objects exist in the high-level theory. Object abstraction is also considered in [46], with the help of an indistinguishability relation, which can be defined by means of a set of relevant predicates (i.e., subset of predicates of the theory that are considered to be relevant to the situation at hand). In this approach, objects x and y are indistinguishable if no relevant predicate distinguishes between them.

Abstraction techniques are widely used in model checking to deal with very large state spaces. A popular approach is counterexample-guided abstraction refinement (CEGAR) [19], which works by using automated techniques to iteratively construct and refine abstractions until a desired precision level is reached. The approach starts by computing an abstraction which is an upper approximation. In this setting, when a specification ([19] uses $ACTL^*$, a fragment of CTL^* which only permits universal quantification over paths, as specification language) holds in the abstract model, it also holds in the concrete model. However, when a counterexample is found in the abstract model, it must be checked to see whether it is reproducible at the concrete level, or it is spurious, and needs to be excluded by refining the abstraction. The initial abstraction is obtained by partitioning the domain of values of variable clusters into classes that are equivalent with respect to the satisfaction of atomic conditions that determine system transitions or appear in the specification and using one representative value from each equivalence class. When a spurious counterexample is obtained, the abstraction is refined by splitting an equivalence class to eliminate it. The approach was extended by Shoham and Grumberg [81] to verify arbitrary CTL specifications and avoid the need to check if a potential counterexample found in the abstract system also holds in the concrete system by using a three-valued semantics.

The abstractions studied in this paper are quite different. They are defined by the user rather than by some automatic technique. They involve new high-level fluents and actions that are meaningful to the users and can be used to express many high-level goals and tasks/programs of interest. They can be used to speed up planning and to give high-level explanations of system behavior.

There is also work showing decidability of model checking of temporal properties in infinite state/object domain systems where the "active domain", i.e., the set of objects that are in the extension of a predicate in the state, remains bounded [30]. Decidability is shown by constructing a finite abstract transition system that is bisimilar to the original infinite one. This kind of abstraction is also quite different from ours. First, it is not defined by a designer, but it is automatically computed by leveraging the properties of bounded theories. Second, it is essentially based on finding a finite number of proxies for real objects and then reusing the proxies, exploiting a locality principle that only objects in the current active domain and in the next state's active domain need to be distinguished.

In planning, several notions of abstraction have been investigated. Among the first approaches to abstraction in planning was *precondition elimination abstraction*, which was initially introduced in context of ABSTRIPS [76]. *Hierarchical Task Networks* (HTNs) (e.g., [36]), abstract over a set of (non-primitive) tasks. Encodings of HTNs in ConGolog with enhanced features like exogenous actions and online executions have been studied by Gabaldon [38]. In contrast to our approach, [38] uses a single BAT; also it does not provide abstraction for fluents.

Planning with *macro operators* (e.g., [51]), is another approach to abstraction which represents meta-actions built from a sequence of action steps. McIlraith and Fadel [65], and Baier and McIlraith [5] investigate planning with *complex actions* (a form of macro actions) specified as Golog programs. Differently from our approach, [65,5] compile the abstracted actions 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. Our approach provides a refinement mapping (similar to that of Global-As-View in data integration [53]) between an abstract BAT and a concrete BAT.

Generalized planning (e.g., [56,49,83,17]) studies the characterization and computation of planning solutions that can generalize over a set of planning instances. An influential work by Bonet and Geffner [17] proposed a notion of *abstract actions*, which abstract over concrete actions in different instances of a generalized planning problem, and capture the effect of actions on the common set of (Boolean and numerical) features. They show that if the abstraction is sound, then an abstract policy which is a solution in the abstract space, can always be instantiated to provide a solution in the original space. Other work in the area uses abstraction, for instance, Aguas et al. [2], which proposed hierarchical finite state controllers for generalized planning. Another related work is [60], which proposes to represent general strategies to solve a class of possibly infinitely many games that have similar structures by a finite state automaton encoded in a BAT with edges labeled by Golog programs. They then show how to use counterexample-guided local search for invariants to verify that the strategy is a solution. We discuss other generalized planning work that builds on our abstraction framework in the next section.

Hierarchical planning approaches mainly focus on improving planning efficiency. Generalized planning works often use abstractions but focus on obtaining general strategies that solve a class of planning problems. Our work instead provides a general framework for abstracting an agent's action repertory and how it affects the domain. It supports various forms of reasoning, not just planning. It can also be used to explain low-level behavior in high-level terms and can be applied in many ways.

Our notion of abstraction between action theories is based on the existence of a bisimulation relation relative to a refinement mapping between the theories' models. Bisimulation [66,67] is a core notion used in the study of concurrent processes in formal methods. Processes, often specified in some process calculus/algebra, can be modeled as labeled transition systems. A process simulates another if it can mimic, i.e., go through a matching sequence of transitions, for all executions of the latter. Processes are equivalent if they can simulate each other, i.e., are bisimilar. Bisimulation is also commonly used to analyze the relationship between logics. For instance, a key result in modal logic is van Benthem's characterization theorem, which essentially states that propositional modal logic is the fragment of first-order logic that is invariant under bisimulation [15,16]. Note that in our notion of bisimulation relative to a mapping m, a transition at the high level usually corresponds to several transitions at the low level and that the resulting situations need only be m-isomorphic.

9. Adaptations and extensions of the abstraction framework

Since it first appeared in [6], our abstraction framework has been adapted, used, and extended in various directions. We summarize such work in this section.

Some work has addressed the issue of automated verification and synthesis of abstractions. In [62], Luo et al. investigate the relationship between our account of agent abstraction and the well-known notion of *forgetting* in first-order/second-order logic. They also address the problem of *synthesizing a sound and complete abstraction given a low-level BAT and a refinement mapping*. They show that one can use forgetting (of low-level fluent and action symbols) to obtain second-order theory that is a sound and complete abstraction of a low-level BAT for a given mapping, provided that the mapping is *nondeterministic uniform (NDU)* with respect to the low-level BAT, i.e., such that different refinements of a high-level action sequence are indistinguishable in the high-level language.[9] In general, the result of this method may not have the form of a BAT. But it is also shown that if the mapping is "Markovian" with respect to the low-level BAT, i.e., such that the executability and effects of refinements of high-level actions only depend on the values of the high-level fluents in the current situation, whenever such a situation is the result of executing a refinement of high-level action sequence, then a sound and complete abstraction in the form of a generalized BAT, i.e., where the initial database, action preconditions and successor state descriptions can be second-order formulas, can be obtained. Finally, they show that in the propositional case, under the Markovian restriction, a sound and complete abstraction is always computable.

Luo [59] studies automated verification of the existence of a propositional agent abstraction given a low-level finite-state labeled transition system (LTS) and a refinement mapping that maps high-level atoms into low-level state formulas and high-level actions into loop-free low-level programs. Notions of sound/complete abstractions over such LTS are defined. Note that actions in the high-level LTS need not be deterministic as in our approach. Luo then shows that the propositional agent abstraction existence problem (for both deterministic and non-deterministic LTSs) can be reduced to a CTLK (an extension of CTL with epistemic operators [72]) model checking problem. The idea behind this is that low-level states reached by executions of the same high-level actions are epistemically indistinguishable for the high-level agent. Given the low-level LTS and mapping, an induced epistemic transition system (ETS) is obtained. It is shown that a propositional agent abstraction exists if this ETS satisfies certain properties. If the low-level LTS is represented symbolically as a propositional STRIPS planning domain, a symbolic induced ETS can also be obtained. However this symbolic ETS includes all the propositional variables that were present in the given low-level LTS, and so is it not fully abstract. The approach has been implemented in a system that uses the MCMAS [58] model checker. Experiments on several domains from

[9] Note that it follows our Theorem 20 that their NDU condition is satisfied whenever a sound abstraction BAT exists for the given low-level BAT and mapping.

classical planning show that the system can verify the existence of sound and complete propositional abstractions in reasonable time for medium size domain instances. Synthesizing abstract LTS/planning domains is left for future work.

As mentioned in the previous section, much work in *generalized planning* exploits abstraction to generate a (typically iterative) general plan/strategy that solves a set of similar planning problems. Inspired by [17], Cui et al. [24] extend our framework to develop a uniform abstraction framework for generalized planning. They consider abstractions that involve nondeterministic actions as in fully-observable nondeterministic (FOND) planning, and seek strong cyclic solutions that achieve the goal under a fairness assumption. They use Golog programs to represent non-deterministic actions. To deal with the issue of termination of generalized plans under fairness constraints, they use an extended situation calculus with infinite histories [79]. The framework is also extended with counting (following [52]) and transitive closure (following [50]). In this setting, a generalized planning problem (GPP) is formalized as a triple of a BAT, a trajectory constraint and a goal. They define notions of sound/complete abstractions with respect to a mapping m at the level of models by introducing notions of m-simulation and m-back simulation that are weaker than m-bisimulation.[10] They then use these to define notions of sound/complete abstractions (at the theory level) for GPPs. They show that if a high-level GPP is a sound abstraction of a low-level GPP, and a high-level program δ is a strong solution to the high-level GPP, then $m(\delta)$ is a strong solution to the low-level GPP, where $m(\delta)$ is obtained by replacing all high-level fluents and actions in δ by their mapped value. To be a strong solution to a GPP, a program δ must be guaranteed to achieve the goal under the trajectory constraint unless the execution aborts/blocks. Note that this is a rather weak notion of strong solution given that δ and $m(\delta)$ may be nondeterministic and have executions that block. A number of existing approaches to generalized planning such as [17] are formalized in the proposed abstraction framework and compared.

In later work, Cui et al. [23] extended this approach to support automatic verification that one has a sound abstraction for a GPP, where the abstraction model is a qualitative numerical planning (QNP) problem, a common approach. They assume that the QNP abstraction is "bounded", i.e., integer variables can only be incremented or decremented by one. To do this, they first obtain a "proof-theoretic" characterization of their notion of sound abstraction for GPPs (along the lines of our Theorem 14), and then identify a sufficient condition for having a sound abstraction when high-level action implementations are deterministic and loop-free. They then develop a sound abstraction verification system that checks the sufficient condition based on the SMT solver $Z3$. Experiments showed that the system was able to successfully verify hand-crafted QNP abstractions for several GPP domains in reasonable time.

Dong et al. [34] develops an automatic method for generating sound and complete bounded qualitative numeric planning (BQNP) abstractions for generalized planning from classical planning problem instances with baggable types. BQNP is a generalization of classical planning with non-negative real variables that can be increased or decreased by one. The refined solutions to the abstract problem are solutions to the generalized planning problem whose instances share the same bags with the given instance but the sizes of the bags might be different. The abstraction method works by introducing a counter for each bag of indistinguishable tuples of objects. One limitation is that BQNP is undecidable; a sound but incomplete solver has been developed. Their notion of sound and complete abstraction is taken from [24].

Fang et al. [37] develops an automated method for synthesizing a high-level action theory $\mathcal{D}_h$ that is a sound and complete abstraction for a given a low-level action theory $\mathcal{D}_l$ with respect to a restricted type of refinement mapping m under certain conditions. The method works for low-level BATs in a situation calculus extended with transitive closure and integer-valued fluents. The high-level theory is expressed in a propositional situation calculus with integer valued fluents and linear integer arithmetic formulas. The mapping is quite restricted in that a (parameter-less) high-level action must be mapped into a single closed guarded low-level action. The high-level abstract action theory can be obtained by syntactic manipulation. But some of the conditions for correctness are defined semantically. They use our definition of sound and complete abstraction, with some extensions to the classes of BATs involved.

In this paper, we have ignored the issue of sensing and knowledge acquisition. Banihashemi et al. [7] extend our abstraction framework to the case where the agent is executing *online* i.e., may acquire new knowledge while executing (e.g., by sensing) [33,78]. This means that the knowledge base that the agent uses in its reasoning needs to be updated during the execution. A sufficient property is identified which allows *sound abstraction to persist in online executions*. This property ensures that the low level has learned as much as the high level did when a refinement of the high-level action was performed, which ensures that every low-level model still has an m-bisimilar high-level model. Abstraction can also be exploited to support planning for agents that execute online. To achieve its goals, such an agent may need to select different courses of action depending on the results of sensing actions (or the occurrence of exogenous actions). This work adapts definition of ability to perform a task/achieve a goal (e.g., [69,55]) to its model of online executions. To be *able* to achieve a goal, an agent needs to have a *strategy* that ensures reaching the goal no matter how the environment behaves and how sensing turns out. Ability is similar to the concept of *conditional* or *contingent planning* [18,3,39]. The main result of this work shows that under some reasonable assumptions, if one has a sound abstraction and the agent *has a strategy by which it is able to achieve a goal at the high level, then one can refine it into a low-level strategy* by which the agent is able to achieve the refinement of the goal. Furthermore, the low-level strategy can be obtained piecewise/incrementally, by finding a refinement of each step of the high-level strategy individually. This makes reasoning about agents' abilities much easier.

Many agents operate in nondeterministic environments where the agent does not fully control the outcome of its actions (e.g., flipping a coin where the outcome may be heads or tails). In more recent work, [9,10] extend our framework to abstract the behavior of

[10] Such notions are interesting and allow one to get interesting results about the existence of plans in models, but m-bisimulation is a simpler and more intuitive notion. If one has perfect information and a single model, then one would expect to have m-bisimilar models at different levels of abstraction. If instead one has imperfect information and multiple models, but the information one has at the high and low levels is consistent, then our notion of sound (and possibly also complete) abstraction defined in terms of m-bisimulation seems sufficient to get the results that we want. More analysis of these issues would be worthwhile.

an agent that has nondeterministic actions based on nondeterministic situation calculus [26] action theories. In this setting, each high-level action is considered as being composed of an agent action and an (implicit) environment reaction. As a consequence, the agent action (without the environment reaction) is mapped into a low-level agent program that appropriately reflects the nondeterminism of the environment, and the complete high-level system action, including both the agent action and the environment reaction, is mapped into a low-level system program that also relates the high-level environment reaction to the low-level ones. A constraint is defined to ensure that the mapping is proper, i.e., that agent actions and system actions are mapped in a consistent way. This allows our notion of m-bisimulation to extend naturally to this new setting, which supports strategic reasoning and strategy synthesis, by allowing one to quantify separately on agent actions and environment reactions. It is shown that if the agent has a (strong FOND) plan/strategy to achieve a goal/complete a task at the abstract level, and it can always execute the nondeterministic abstract actions to completion at the concrete level even if it does not control their outcomes, then there exists a refinement of it that is a (strong FOND) plan/strategy to achieve the refinement of the goal/task at the concrete level. Our framework was also recently extended along similar lines to abstract agent behavior in multi-agent synchronous game structures and support verification of strategic properties of coalitions in [54]. Our notion of sound abstraction in the nondeterministic situation calculus has also been extended to support linear temporal logic (LTL) constraints and objectives in [28], which uses this to construct propositional abstractions to solve generalized planning problems.

Related to this is work on using BAT abstraction in supervisory control of agents. In [8], constraints on the agent's behavior are specified in the language of the abstract BAT. A high-level maximally permissive supervisor (MPS) is first obtained that enforces these constraints on the behavior of a high-level agent while leaving the agent with as much freedom to act/autonomy as possible. The task is then to synthesize an MPS for the concrete agent. To support mapping an abstract supervision specification to a concrete one, [8] extend the refinement mapping to map any situation-determined high-level program to a situation-determined low-level program that implements it (concurrency is allowed at the high level, but the implementations of high-level actions cannot be interleaved to maintain bisimulation). A low-level MPS based on the concrete specification is then obtained, and it is shown that the high-level MPS is the abstract version of the low-level MPS. Moreover, a hierarchically synthesized MPS may be obtained by taking the abstract MPS and refining its actions piecewise.

Belle [13] extends our abstraction approach to probabilistic relational models, assuming a weighted model counting reasoning infrastructure, and focusing on static models where there are no actions. He assumes that one defines a refinement mapping that associates each high-level atom to a (propositional) formula over the language of the low-level model/theory. Isomorphism between a high-level model and a low-level model relative to a mapping is defined essentially as in our account. This is then used to define a notion of weighted sound abstraction relative to a mapping between such models which ensures that high-level sentences that have a non-zero probability at the low level must also have a non-zero probability at the high level, and high-level sentences that have probability 1 at the high level must also have probability 1 at the low level. A weighted complete abstraction ensures the converse. Finally, a weighted exact abstraction is a weighted sound and complete abstraction where the probability of any high-level sentence is the same at the high level and low level. The paper also discusses techniques for automated synthesis of weighted abstractions.

Hofmann and Belle [48] extend this approach to support abstraction of probabilistic dynamical systems. The high-level and low-level action theories are defined in the DSG logic (which extends DS [14]), a first-order modal version of the situation calculus that supports probabilistic belief ($B(\alpha : r)$ means that α is believed with probability r), stochastic/noisy observations and actions, and Golog programs ($[\delta]\alpha$ means that α holds after every execution of program δ). Their notion of m-bisimulation relates epistemic states in the high-level and low-level theories; the base condition is epistemic m-isomorphism, which requires not only that the states be m-isomorphic, but also that the degrees of belief in high-level sentences be exactly the same. It is then shown that m-bisimilar states satisfy the same bounded (i.e., without the "always" operator) high-level DSG formulas. This is then extended to theories by defining notions of sound/complete abstraction as usual. The framework is used to show one can abstract away the stochastic aspects of the domain and obtain a non-probabilistic high-level domain theory while keeping guarantees that refinements of the high-level actions and plans achieve their goals. One limitation is that DSG does not support strategic reasoning: one cannot say that the agent has a strategy to execute a program to termination and achieve a goal no matter how the environment behaves, i.e., no matter how it selects sensing results and action outcomes. The requirement that degrees of belief in high-level sentences be exactly the same in epistemic m-isomorphism may also be too strict in some applications.

Applying abstraction to smart manufacturing, De Giacomo et al. [25] focus on manufacturing as a service in which manufacturing facilities "bid" to produce (previously unseen and complex) products. A facility, consists of a set of configurable manufacturing resources (e.g., robots, tools, etc.). Given a product recipe, which outlines the abstract (i.e., resource independent) tasks required to manufacture a product, a facility decides whether to bid for the product if it can synthesize, "on the fly", a process plan controller that delegates abstract manufacturing tasks in the supplied process recipe to the appropriate manufacturing resources. The operations in manufacturing processes are described by basic action theories, and the processes as ConGolog programs over such action theories. The facility basic action theory is obtained by combining the resources' basic action theories, and the facility process results from the concurrent, synchronous execution of the processes of each resource. Moreover, another basic action theory represents a common, resource independent information model of the data and objects that the recipes can manipulate and based on which they are designed. A set of mappings relate the common abstract resource-independent BAT with the resource-dependent facility BAT. Based on this representation, this work formalizes when a process recipe can be realized by facility. Two decidable cases of finite domains and bounded action theories are identified, for which techniques to actually synthesize the controller are provided.

10. Conclusion

In this paper, we proposed a general framework for agent abstraction based on the situation calculus and ConGolog. We defined a notion of a high-level basic action theory (BAT) being a *sound and/or complete abstraction* of a low-level BAT with respect to a given mapping between their respective languages. This formalization involved the existence of a *bisimulation relation relative to the mapping* between models of the abstract and concrete theories. Furthermore, we identified a set of necessary and sufficient conditions for checking if one has a sound and/or complete abstraction with respect to a mapping. We have shown that 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. Finally, we 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 monitoring and providing high-level explanations of behavior. Our framework is based on the situation calculus, which provides a first-order representation of the state and can model the data/objects that the agent interacts with.

In this work, 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 [31] and "compile" such a 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.

There are some limitations of the abstraction framework presented in this paper. Being based on the standard version of the situation calculus, it does not handle actions with nondeterministic effects, sensing and knowledge acquisition by the agent, as well as domains that involve multiple agents and imperfect information. But as mentioned in the previous section, there has already been work on addressing some of these limitations. Another limitation is that, for simplicity, we assume that the object domain is the same at the abstract and concrete levels and do not support object abstraction. It should be possible to adapt the approach of [70] to handle this. Another limitation is that our notion of sound abstraction is quite strict: all the high-level actions and fluents must be mapped and every model of the low level theory must be bisimilar to some model of the high-level theory relative to the mapping m. However, note that one can restrict attention to a narrow range of actions and situations at the high level and specify the mapping to embody these restrictions, for instance, by having strict conditions for refinements of high-level actions to be executable. Also, one need not have a complete abstraction, i.e., some high-level models need not have m-bisimilar low-level models and the high-level theory can have less information than the low-level one, even when restricting attention to high-level fluents and actions. It would be interesting to investigate less strict notions of abstraction, for instance, where the high-level theory does not accurately model the probability distribution over action outcomes that hold at the low level; see [13,48] for work along these lines.

A key topic for future work is how one can construct abstraction mappings and abstract action theories. This could be done in the context of designing a new agent system or to enhance an existing one, for instance to support monitoring, adaptation, and evolvability. Note that there may be many different abstractions of a concrete action theory, each of which may be useful for a different purpose. We need to identify how such requirements would be specified. Once we have such requirements, one could generate an abstract language/ontology, i.e., fluents and action types, and a mapping for them that satisfies the requirements. After that, one could check that there exists a high-level theory that is a sound and/or complete abstraction for such a mapping and a given low-level theory, and if so, synthesize one and verify that it is indeed a sound and/or complete abstraction. Many of these steps could be automated to a significant extent. But a human designer would likely need to be involved to refine/revise the requirements and help adjust the high-level language and mapping to ensure that a sound/complete abstraction is obtained. For example, if different refinements of a high-level action can produce situations that are not m-isomorphic, then we can either switch to more abstract high-level fluents to ensure the resulting situations are indeed m-isomorphic, or split the high-level action into several for which we do have m-isomorphic results. We plan to investigate methodologies and develop tools for this process. Note that in the above, we assumed that one wants to obtain an abstract action theory given a concrete one, i.e., obtain an abstraction bottom up. In other cases, one may already have the abstract action theory and be interested in obtaining a concrete one. In this case one could try to adapt well known formal methods techniques for refinement-based software development [1].

As mentioned in the previous section, there has been some work on checking whether a sound/complete abstraction exists for a given low-level BAT and mapping, and on synthesizing a high-level BAT when it does, focusing on the propositional case [62,59]. For verifying that a high-level BAT is a sound abstraction of a low-level BAT with respect to a mapping in the general infinite-states case, one can try to use general first-order and higher-order logic theorem proving techniques and tools as discussed in Section 5; for instance, one could build upon Shapiro's work [80]. Another possibility is to restrict attention to bounded basic action theories [30]. These are action theories where it is entailed that in all situations, the number of object tuples that belong to the extension of any fluent is bounded, although the object domain remains infinite and an infinite run may involve an infinite number of objects. In [30], it is shown that verifying a large class of temporal properties over bounded BATs is decidable. Finally, for the case where one has a finite object domain, one can use model checking techniques as discussed in Section 5.

Pre-trained large language models have also been used to help in defining planning domains [45,82]. Given this and the fact that they embed wide ranging world knowledge, it would seem promising to try to exploit them to help generate useful abstractions. Banihashemi et al. [11] explore the generation of an abstract planning domain and problem instance in the Planning Domain Definition Language (PDDL) [64] based on a concrete domain and problem instance and a purpose of abstraction using large language models. Note that for any domain and problem in the ADL [71] fragment of PDDL, we can obtain an equivalent BAT as shown in [21]; this BAT must have a finite object domain and the initial situation description must be complete, i.e., completely specify which tuples of objects are in the extension of every fluent initially. We plan on exploring the generation of useful abstractions using large language models.

As already mentioned, we have already extended our agent abstraction framework to deal with nondeterministic domains and contingent planning [9,10], as well as agents that perform sensing and acquire new knowledge at execution time [7]. Lespérance et al. [54] also extends the framework to multi-agent domains and supports strategic reasoning. Further work in these areas is indicated, in particular to support multiple environment models involving a range of contingencies. We would also like to explore how agent abstraction can be used in verification, where there is some related work [68,12].

CRediT authorship contribution statement

Bita Banihashemi: Writing – review & editing, Writing – original draft, Validation, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Giuseppe De Giacomo:** Writing – review & editing, Writing – original draft, Validation, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Yves Lespérance:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially supported by the UKRI Erlangen AI Hub on Mathematical and Computational Foundations of AI Grant Number EP/Y028872/1, the Natural Sciences and Engineering Research Council of Canada (NSERC) Grant Number RGPIN-2022-04565, and York University.

Appendix A. Proofs

A.1. m-bisimulation

Lemma 4. *If $s_h \simeq_m^{M_h,M_l} s_l$, then for any high-level situation-suppressed formula ϕ, we have that:*

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

Proof. By induction of the structure of ϕ, using the definition of m-isomorphic situations. □

Lemma 5. *If $M_h \sim_m M_l$, then for any sequence of high-level actions $\vec{\alpha}$, we have that*

if $M_l, v[s'/s_l] \models Do(m(\vec{\alpha}), S_0, s')$, then there exists s_h such that:
$M_h, v[s'/s_h] \models s_h = do(\vec{\alpha}, S_0) \wedge Executable(s_h)$ and $s_h \sim_m^{M_h,M_l} s_l$

and

if $M_h, v[s'/s_h] \models s_h = do(\vec{\alpha}, S_0) \wedge Executable(s_h)$
then there exists s_l such that $M_l, v[s'/s_l] \models Do(m(\vec{\alpha}), S_0, s')$ and $s_h \sim_m^{M_h,M_l} s_l$.

Proof. The result follows easily by induction on the length of α using the definition of m-bisimulation. □

Theorem 6. *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 ϕ, we have that:*

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

Proof. The result follows immediately from Lemma 4 and Lemma 5. □

A.2. Sound abstraction

Theorem 9. *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 ϕ, if $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$, then $\mathcal{D}_l \cup C \models \exists s. Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$.*

Proof. Assume that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m and that $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$. Take an arbitrary model M_l of $\mathcal{D}_l \cup C$. Since $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. Given our assumption,

it follows that $M_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$. It then follows by Theorem 6 that $M_l \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$. M_l was an arbitrarily chosen model of $\mathcal{D}_l \cup \mathcal{C}$, so the result follows. □

Corollary 10. *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 ϕ, we have that:*

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

Proof. By contradiction. Suppose that there exist M_l and v such that $M_l, v \models \mathcal{D}_l \cup \mathcal{C} \cup \{Do(m(\vec{\alpha}), S_0, s) \wedge Do(m(\vec{\alpha}), S_0, s') \wedge m(\phi)[s] \wedge \neg m(\phi)[s']\}$. Since $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping m, by Corollary 8, it follows that $\mathcal{D}_h \cup \{Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)] \wedge \neg\phi[do(\vec{\alpha}, S_0)]\}$ is satisfiable, a contradiction. □

Theorem 12. *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 β, we have that:*

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

Proof. Take an arbitrary model M_l of $\mathcal{D}_l \cup \mathcal{C}$ and valuation v and assume that $M_l, v \models \exists s.Do(m(\vec{\alpha}\beta), S_0, s)$. It follows that there exists s_l such that $M_l, v[s/s_l] \models Do(m(\vec{\alpha}), S_0, s) \wedge \exists s'.Do(m(\beta), s, s')$. Since $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. Then by Lemma 5, it follows that there exists a situation s_h such that $M_h, v[s/s_h] \models s = do(\vec{\alpha}, S_0) \wedge Executable(s) \wedge Poss(\beta, s)$. Take an arbitrary situation s'_l and suppose that $M_l, v[s/s'_l] \models Do(m(\vec{\alpha}), S_0, s)$. Then it follows by Lemma 5 that $s_h \sim_m^{M_h, M_l} s'_l$. Since we also have that $M_h \models Poss(\beta, do(\vec{\alpha}, S_0))$, it follows that $M_l, v[s/s'_l] \models \exists s'.Do(m(\beta), s, s')$. Since s'_l was chosen arbitrarily, it follows that $M_l, v \models \forall s.Do(m(\vec{\alpha}), S_0, s) \supset \exists s'.Do(m(\beta), s, s')$. □

To prove Theorem 14 (and Theorem 21), we start by showing some lemmas.

Lemma 26. *If $M_h \models \mathcal{D}^h$ for some high-level BAT $\mathcal{D}^h$ and $M_l \models \mathcal{D}^l \cup \mathcal{C}$ for some low-level BAT $\mathcal{D}^l$ and $M_h \sim_m M_l$ for some mapping m, then*

(a) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi^{Poss}_{A_i}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s')),$

(b) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s'])),$

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

Proof. By contradiction. Assume that M_h is a model of a high-level BAT $\mathcal{D}^h$ and M_l is a model of a low-level BAT $\mathcal{D}^l \cup \mathcal{C}$ and $M_h \sim_m M_l$. Suppose that condition (a) does not hold. Then there exists a ground high-level action sequence $\vec{\alpha}$, a ground low-level situation term S, and a ground high-level action $A_i(\vec{x})$ such that $M_l \models Do(m(\vec{\alpha}), S_0, S)$ and either (*) $M_l \models m(\phi^{Poss}_{A_i}(\vec{x}))[S]$ and $M_l \not\models \exists s'.Do(m(A_i(\vec{x})), S, s')$ or (**) $M_l \not\models m(\phi^{Poss}_{A_i}(\vec{x}))[S]$ and $M_l \models \exists s'.Do(m(A_i(\vec{x})), S, s')$. In case (*), by Theorem 6, since $M_h \sim_m M_l$, it follows that $M_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi^{Poss}_{A_i}(\vec{x})[do(\vec{\alpha}, S_0)]$. Since $M_h \models \mathcal{D}^h_{Poss}$, we must also have that $M_h \models Poss(A_i(\vec{x}), do(\vec{\alpha}, S_0))$, and thus that $M_h \models Executable(do([\vec{\alpha}, A_i(\vec{x})], S_0))$. Thus by Theorem 6, $M_l \models Do(m(\vec{\alpha}), S_0, S) \wedge \exists s' Do(m(A_i(\vec{x})), S, s')$, which contradicts (*). Case (**) can be shown to lead to a contradiction by a similar argument.
Now suppose that condition (b) does not hold. Then there exists a ground high level action sequence $\vec{\alpha}$, a ground high-level action $A_i(\vec{x})$, and ground low-level situation terms S and S' such that $M_l \models Do(m(\vec{\alpha}), S_0, S) \wedge Do(m(A_i(\vec{x})), S, S')$ and either (*) $M_l \models m(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[S]$ and $M_l \not\models m(F_i(\vec{y}))[S']$ or (**) $M_l \not\models m(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[S]$ and $M_l \models m(F_i(\vec{y}))[S']$. In case (*), by Theorem 6, since $M_h \sim_m M_l$, it follows that $M_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x})[do(\vec{\alpha}, S_0)] \wedge Poss(A_i(\vec{x}), do(\vec{\alpha}, S_0))$. Since $M_h \models \mathcal{D}^h_{ssa}$, we must also have that $M_h \models F_i(\vec{y})[do([\vec{\alpha}, A_i(\vec{x})], S_0)]$. Thus by Theorem 6, $M_l \models Do(m(\vec{\alpha}), S_0, S) \wedge Do(m(A_i(\vec{x})), S, S') \wedge m(F_i(\vec{y}))[S']$, which contradicts (*). Case (**) can be shown to lead to a contradiction by a similar argument. □

The above lemma implies 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.

We also prove another lemma:

Lemma 27. *Suppose that $M_h \models \mathcal{D}^h$ for some high-level BAT $\mathcal{D}^h$ and $M_l \models \mathcal{D}^l \cup \mathcal{C}$ for some low-level BAT $\mathcal{D}^l$ and m is a mapping between the two theories. Then if*

(a) $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$,
(b) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$ *and*
(c) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,
then $M_h \sim_m M_l$,
where $\phi_{A_i}^{Poss}(\vec{x})$ *is the right hand side of the precondition axiom for action* $A_i(\vec{x})$, *and* $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ *is the right hand side of the successor state axiom for* F_i *instantiated with action* $A_i(\vec{x})$ *where action terms have been eliminated using* D_{ca}^h.

Proof. Assume that the antecedent holds. Let us show that $M_h \sim_m M_l$. Let B be the relation over $\Delta_S^{M_h} \times \Delta_S^{M_l}$ such that

$\langle s_h, s_l \rangle \in B$
if and only if
there exists a ground high-level action sequence $\vec{\alpha}$
such that $M_l, v[s/s_l] \models Do(m(\vec{\alpha}), S_0, s)$
and $s_h = do(\vec{\alpha}, S_0)^{M_h}$.

Let us show that B is an m-bisimulation relation between M_h and M_l. We need to show that if $\langle s_h, s_l \rangle \in B$, then it satisfies the three conditions in the definition of m-bisimulation. We prove this by induction on n, the number of actions in s_h.
Base case $n = 0$: By the definition of B, $s_h = S_0^{M_h}$, $\alpha = \epsilon$, and $s_l = S_0^{M_l}$. By (a) we have that $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$, so condition 1 holds. By Lemma 4, it follows that $M_h, v[s/s_h] \models \phi_A^{Poss}(\vec{x})[s]$ if and only if $M_l, v[s/s_l] \models m(\phi_A^{Poss}(\vec{x}))[s]$ for any high-level primitive action type $A \in \mathcal{A}_h$. Thus by the action precondition axiom for A, $M_h, v[s/s_h] \models Poss(A(\vec{x}), s)$ if and only if $M_l, v[s/s_l] \models m(\phi_A^{Poss}(\vec{x}))[s]$. By condition (b), we have that $M_l, v[s/s_l] \models m(\phi_A^{Poss}(\vec{x}))[s]$ if and only if $M_l, v[s/s_l] \models \exists s'.Do(m(A(\vec{x})), s, s')$. Thus $M_h, v[s/s_h] \models Poss(A(\vec{x}), s)$ if and only if there exists s_l' such that $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s, s')$. By the way B is defined, $\langle do([\vec{\alpha}, A(\vec{x})], S_0)^{M_h,v}, s_l' \rangle \in B$ if and only if $M_l, v[s/s_l, s'/s_l'] \models Do(m(A(\vec{x})), s_l, s_l')$ (note that we have standard names and domain closure for objects and actions, so we can always ground $\vec{x}$). Thus conditions (2) and (3) hold for $\langle s_h, s_l \rangle = \langle S_0^{M_h}, S_0^{M_l} \rangle$.
Induction step: Assume that if $\langle s_h, s_l \rangle \in B$ and the number of actions in s_h is no greater than n, then $\langle s_h, s_l \rangle$ satisfies the three conditions in the definition of m-bisimulation. We have to show that this must also hold for any $\langle s_h, s_l \rangle \in B$ where s_h contains $n+1$ actions. First we show that condition 1 in the definition of m-bisimulation holds. If $\langle s_h, s_l \rangle \in B$ and s_h contains $n+1$ actions, then due to the way B is defined, there exist situations s_h' and s_l', a ground high-level action sequence $\vec{\alpha}$ of length n, and a ground high-level action $A(\vec{c})$, such that $s_h = do(A(\vec{c}), do(\vec{\alpha}, S_0))^{M_h}$, $s_h' = do(\vec{\alpha}, S_0)^{M_h}$, $M_l, v[s/s_l'] \models Do(m(\vec{\alpha}), S_0, s)$, and $\langle s_h', s_l' \rangle \in B$. s_h' contains n actions so by the induction hypothesis, $\langle s_h', s_l' \rangle$ satisfies the three conditions in the definition of m-bisimulation, in particular $s_h' \simeq_m^{M_h,M_l} s_l'$. By Lemma 4, it follows that $M_h, v[s/s_h'] \models \phi_{F,A}^{ssa}(\vec{y}, \vec{c})[s]$ if and only if $M_l, v[s/s_l'] \models m(\phi_{F,A}^{ssa}(\vec{y}, \vec{c}))[s]$ for any high-level fluent $F \in \mathcal{F}_h$. Thus by the successor state axiom for F, $M_h, v[s/s_h'] \models F(\vec{y}, do(A(\vec{c}), s))$ if and only if $M_l, v[s/s_l'] \models m(\phi_{F,A}^{ssa}(\vec{y}, \vec{c}))[s]$. By condition (c), we have that $M_l, v[s/s_l'] \models m(\phi_{F,A}^{ssa}(\vec{y}, \vec{c}))[s]$ if and only if $M_l, v[s/s_l] \models m(F(\vec{y}))[s]$. Thus $M_h, v[s/s_h'] \models F(\vec{y}, do(A(\vec{c}), s))$ if and only if $M_l, v[s/s_l] \models m(F(\vec{y}))[s]$. Therefore, $s_h \simeq_m^{M_h,M_l} s_l$, i.e., condition 1 in the definition of m-bisimulation holds.
We can show that $\langle s_h, s_l \rangle$, where s_h contains $n+1$ actions, satisfies conditions 2 and 3 in the definition of m-bisimulation, by exactly the same argument as in the base case. □

With these lemmas in hand, we can prove our main result:

Theorem 14. $\mathcal{D}^h$ *is a sound abstraction of* $\mathcal{D}^l$ *relative to mapping* m *if and only if*

(a) $\mathcal{D}_{S_0}^l \cup \mathcal{D}_{ca}^l \cup \mathcal{D}_{coa}^l \models m(\phi)$, *for all* $\phi \in \mathcal{D}_{S_0}^h$,
(b) $\mathcal{D}^l \cup \mathcal{C} \models \forall s.Do(ANYSEQHL, S_0 o, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$,
(c) $\mathcal{D}^l \cup \mathcal{C} \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

where $\phi_{A_i}^{Poss}(\vec{x})$ *is the right hand side (RHS) of the precondition axiom for action* $A_i(\vec{x})$, *and* $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ *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* $\mathcal{D}_{ca}^h$.

Proof. (⇒) By contradiction. Assume that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m. Suppose that condition (a) does not hold, i.e., there exists $\phi \in \mathcal{D}_{S_0}^h$ such that $\mathcal{D}_{S_0}^l \cup \mathcal{D}_{ca}^l \cup \mathcal{D}_{coa}^l \not\models m(\phi)$. Thus there exists a model M_l' of $\mathcal{D}_{S_0}^l \cup \mathcal{D}_{ca}^l \cup \mathcal{D}_{coa}^l$ such that $M_l' \not\models m(\phi)$. By the relative satisfiability theorem for basic action theories [73,75], this model can be extended to a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ such that

$M_l \nvDash m(\phi)$. Since $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. By Theorem 6, it follows that $M_h \nvDash \phi$. Thus $\mathcal{D}_h \nvDash D^h_{S_0}$, contradiction.

Now suppose that condition (b) does not hold. Then there exists a model M_l of $\mathcal{D}_l \cup C$ such that M_l falsifies condition (b). Since $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ wrt m, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. But then by Lemma 26, M_l must satisfy condition (b), contradiction.

We can prove that condition (c) must hold using Lemma 26 by the same argument as for condition (b).

($\Leftarrow$) Assume that conditions (a), (b), and (c) hold. Take a model M_l of $\mathcal{D}^l \cup C$. Let M_h be a model of the high-level language such that

- **(i)** M_h has the same object domain as M_l and interprets all object terms like M_l,
- **(ii)** $M_h \vDash \mathcal{D}^h_{ca}$,
- **(iii)** $M_h \vDash \Sigma$,
- **(iv)** $M_h, v \vDash F(\vec{x}, do(\vec{\alpha}, S_0))$ if and only if $M_l, v \vDash \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(F(\vec{x}))[s]$ for all fluents $F \in \mathcal{F}^h$ and all ground high-level action sequences $\vec{\alpha}$,
- **(v)** $M_h \vDash Poss(A(\vec{x}), do(\vec{\alpha}, S_0))$ if and only if $M_l \vDash \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge \exists s' Do(m(A(\vec{x})), s, s')$.

It follows immediately that $M_h \vDash \Sigma \cup \mathcal{D}^h_{ca} \cup \mathcal{D}^h_{coa}$. By condition (iv) above, we have that $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$. Thus by condition (a) and Lemma 4, we have that $M_h \vDash \mathcal{D}^h_{S_0}$. By condition (b) of the Theorem and conditions (iv) and (v) above, $M_h \vDash \mathcal{D}^h_{Poss}$. By condition (c) of the Theorem and condition (iv) above, $M_h \vDash \mathcal{D}^h_{ssa}$. Thus $M_h \vDash \mathcal{D}^h$.

Now M_h and M_l satisfy all the conditions for applying Lemma 27, by which it follows that $M_h \sim_m M_l$. □

Proposition 15. *$\mathcal{D}^{eg}_h$ is a sound abstraction of $\mathcal{D}^{eg}_l$ wrt m^{eg}.*

Proof. We prove this using Theorem 14.

(a) It is easy to see that $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa} \vDash m(\phi)$, for all $\phi \in D^h_{S_0}$.

(b) For the *deliver* high-level action, we need to show that:

$$\begin{aligned}&\mathcal{D}^l \cup C \vDash Do(\text{ANYSEQHL}, S_0, s) \supset \\ &\quad \forall sID.(m(\exists l.Dest_{HL}(sID, l, s) \wedge At_{HL}(sID, l, s)) \\ &\qquad \equiv \exists s' Do(m(deliver(sID)), s, s')),\end{aligned}$$

i.e.,

$$\begin{aligned}&\mathcal{D}^l \cup C \vDash Do(\text{ANYSEQHL}, S_0, s) \supset \forall sID. \\ &\quad (\exists l.Dest_{LL}(sID, l, s) \wedge At_{LL}(sID, l, s) \\ &\qquad \equiv \exists s' Do([unload(sID); getSignature(sID)], s, s')).\end{aligned}$$

It is easy to check that the latter holds as $\exists l.Dest_{LL}(sID, l, s) \wedge At_{LL}(sID, l, s)$ is the precondition of $unload(sID)$ and $unload(sID)$ ensures that the precondition of $getSignature(sID)$ holds.

For the *takeRoute* action, we need to show that:

$$\begin{aligned}&\mathcal{D}^l \cup C \vDash Do(\text{ANYSEQHL}, S_0, s) \supset \forall sID, r, o, d. \\ &\quad m(o \neq d \wedge At_{HL}(sID, o, s) \wedge CnRoute_{HL}(r, o, d, s) \wedge \\ &\qquad (r = Rt_B \supset \neg Priority(sID, s))) \\ &\qquad \equiv \exists s' Do(m(takeRoute(sID, r, o, d)), s, s'),\end{aligned}$$

i.e.,

$$\begin{aligned}&\mathcal{D}^l \cup C \vDash Do(\text{ANYSEQHL}, S_0, s) \supset \forall sID, r, o, d. \\ &\quad o \neq d \wedge At_{LL}(sID, o, s) \wedge CnRoute_{LL}(r, o, d, s) \wedge \\ &\qquad (r = Rt_B \supset \neg(BadWeather(s) \vee Express(sID, s))) \\ &\qquad \equiv \exists s' Do(m(takeRoute(sID, r, o, d)), s, s').\end{aligned}$$

It is easy to show that the latter holds as the left hand side of the $\equiv$ is equivalent to $m(takeRoute(sID, r, o, d))$ being executable in s. First, we can see that the left hand side of the $\equiv$ is equivalent to the preconditions of first *takeRoad* action in $m(takeRoute(sID, r, o, d))$, noting that in the case where $r = Rt_B$, $takeRoute(sID, r, o, d)$ is mapped into *takeRoad* to destination $L3$, and that the only road that is closed is Rd_e; the latter can be proved to always remain true by induction on situations. Moreover, the preconditions of the second *takeRoad* action in $m(takeRoute(sID, r, o, d))$ must hold given this and that the first *takeRoad* has occurred.

(c) For the high-level action *deliver* we must show that:

$$\begin{aligned}&\mathcal{D}^l \cup C \vDash Do(\text{ANYSEQHL}, S_0, s) \supset \\ &\quad \forall sID, s'.(Do(m(deliver(sID)), s, s') \supset \\ &\qquad \bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi^{ssa}_{F_i, deliver}(\vec{y}, sID))[s] \equiv m(F_i(\vec{y}))[s'])).\end{aligned}$$

For the high-level fluent *Delivered*, we must show that

$$\mathcal{D}^l \cup \mathcal{C} \models Do(\text{ANYSEQHL}, S_0, s) \supset$$
$$\forall sID, s'.(Do(m(deliver(sID)), s, s') \supset$$
$$\forall sID'((sID' = sID \vee Unloaded(sID', s') \wedge Signed(sID', s')) \equiv$$
$$Unloaded(sID', s') \wedge Signed(sID', s')).$$

This is easily shown given that $m^{eg}(deliver(sID)) = unload(sID); getSignature(sID)$, using the successor state axioms for $Unloaded$ and $Signed$. For the other high-level fluents, the result follows easily as $m^{eg}(deliver(sID))$ does not affect their refinements.
For the action $takeRoute$ we must show that:

$$\mathcal{D}^l \cup \mathcal{C} \models Do(\text{ANYSEQHL}, S_0, s) \supset$$
$$\forall sID, r, o, d, s'.(Do(m(takeRoute(sID, r, o, d)), s, s') \supset$$
$$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi^{ssa}_{F_i, takeRoute}(\vec{y}, sID, r, o, d))[s] \equiv m(F_i(\vec{y}))[s'])).$$

For the high-level fluent At_{HL}, we must show that

$$\mathcal{D}^l \cup \mathcal{C} \models Do(\text{ANYSEQHL}, S_0, s) \supset$$
$$\forall sID, r, o, d, s'.(Do(m(takeRoute(sID, r, o, d)), s, s') \supset$$
$$\forall sID', l.(At_{LL}(sID', l, s')) \equiv$$
$$(sID' = sID \wedge l = d) \vee$$
$$At_{LL}(sID', l, s) \wedge \neg(sID' = sID \wedge o = l)).$$

This is easily shown given how $takeRoute$ is refined by m^{eg}, using the successor state axioms for At_{LL}. For the other high-level fluents, the result follows easily as $m^{eg}(takeRoute(sID, r, o, d))$ does not affect their refinements. □

A.3. Complete abstraction

Theorem 18. *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 ϕ, if $\mathcal{D}_l \cup \mathcal{C} \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$, then $\mathcal{D}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$.*

Proof. Assume that $\mathcal{D}_h$ is a complete abstraction of $\mathcal{D}_l$ wrt m and that $\mathcal{D}_l \cup \mathcal{C} \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$. Take an arbitrary model M_h of $\mathcal{D}_h$. Since $\mathcal{D}_h$ is a complete abstraction of $\mathcal{D}_l$ wrt m, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ such that $M_h \sim_m M_l$. It must be the case that $\mathcal{M}_l \models \exists s.Do(m(\vec{\alpha}), S_0, s) \wedge m(\phi)[s]$. Therefore by Theorem 6, we must also have that $\mathcal{M}_h \models Executable(do(\vec{\alpha}, S_0)) \wedge \phi[do(\vec{\alpha}, S_0)]$. Since M_h was an arbitrarily chosen model of $\mathcal{D}_h$, the thesis follows. □

Theorem 20. *If $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ wrt 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 $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$.*

Proof. Assume that $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ wrt mapping m.
($\Rightarrow$) Suppose that $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt mapping m. Take an arbitrary model of M_h of $\mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ca} \cup \mathcal{D}^h_{coa}$. By the relative satisfiability theorem for basic action theories of [73,75], M_h can be extended to satisfy all of $\mathcal{D}^h$. Since $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt m, by definition, there exists a model M_l of $\mathcal{D}_l$ such that $M_l \sim_m M_h$. It follows by the definition of m-bisimulation that $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$.
($\Leftarrow$) Suppose that 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 $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$. Take an arbitrary model M_h of $\mathcal{D}^h$. Since M_h is also a model of $\mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ca} \cup \mathcal{D}^h_{coa}$, then there exists a model M_l of $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa}$ such that $S_0^{M_h} \simeq_m^{M_h, M_l} S_0^{M_l}$. Clearly, M_l can be extended to satisfy all of $\mathcal{D}^l$ by the relative satisfiability theorem for basic action theories of [73,75]. Moreover, M_l can be extended to satisfy $\mathcal{C}$ (by the results in [27]). Since $\mathcal{D}^h$ is also a sound abstraction of $\mathcal{D}^l$ wrt m, by Theorem 14 it follows that:

$$M_l \models Do(\text{ANYSEQHL}, S_0, s) \supset$$
$$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi^{Poss}_{A_i}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$$
and
$$M_l \models Do(\text{ANYSEQHL}, S_0, s) \supset$$
$$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$$
$$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s'])),$$

where $\phi^{Poss}_{A_i}(\vec{x})$ and $\phi^{ssa}_{F_i, A_i}(\vec{y}, \vec{x})$ are as in Theorem 14. Thus by Lemma 27, it follows that $M_h \sim_m M_l$. Thus $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt m, by the definition of complete abstraction. □

Theorem 21. *$\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ relative to mapping m if and only if for every model M_h of $\mathcal{D}^h$, there exists a model M_l of $\mathcal{D}^l \cup \mathcal{C}$ such that*

(a) $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$,
(b) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$,
(c) $M_l \models \forall s.Do(ANYSEQHL, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

where $\phi_{A_i}^{Poss}(\vec{x})$ and $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ are as in Theorem 14.

Proof. ($\Rightarrow$) Suppose that $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt mapping m. Take an arbitrary model of M_h of $\mathcal{D}^h$. Since $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt m, by definition, there exists a model M_l of $\mathcal{D}_l$ such that $M_l \sim_m M_h$. It follows by the definition of m-bisimulation that $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$. Furthermore, by Lemma 26, it follows that:

$M_l \models \forall s.Do(\text{ANYSEQHL}, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}.(m(\phi_{A_i}^{Poss}(\vec{x}))[s] \equiv \exists s' Do(m(A_i(\vec{x})), s, s'))$
and
$M_l \models \forall s.Do(\text{ANYSEQHL}, S_0, s) \supset$
$\bigwedge_{A_i \in \mathcal{A}^h} \forall \vec{x}, s'.(Do(m(A_i(\vec{x})), s, s') \supset$
$\bigwedge_{F_i \in \mathcal{F}^h} \forall \vec{y}(m(\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x}))[s] \equiv m(F_i(\vec{y}))[s']))$,

where $\phi_{A_i}^{Poss}(\vec{x})$ and $\phi_{F_i,A_i}^{ssa}(\vec{y}, \vec{x})$ are as in Theorem 14.
($\Leftarrow$) The thesis follows immediately from Lemma 27 and the definition of complete abstraction. □

Corollary 22. *If $\mathcal{D}_{S_0}^h$ is a complete theory (i.e., for any situation suppressed formula ϕ, either $\mathcal{D}_{S_0}^h \models \phi[S_0]$ or $\mathcal{D}_{S_0}^h \models \neg\phi[S_0]$) and $\mathcal{D}^l$ is satisfiable, then if $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ wrt m, then $\mathcal{D}^h$ is also a complete but abstraction of $\mathcal{D}^l$ wrt m.*

Proof. $\mathcal{D}^l$ is satisfiable, so it has a model M_l, and since $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ wrt m, $\mathcal{D}^h$ has a model M_h such that $M_h \sim_m M_l$. By the definition of m-bisimilar model, this implies that $S_0^{M_h} \simeq_m^{M_h,M_l} S_0^{M_l}$. Take an arbitrary model M_h' of $\mathcal{D}_h$. Since $\mathcal{D}_{S_0}^h$ is a complete theory, we have that $M_h' \models \phi[S_0]$ iff $M_h \models \phi[S_0]$ for all situation suppressed formulas ϕ. It follows that $S_0^{M_h'} \simeq_m^{M_h,M_l} S_0^{M_l}$. Then by Theorem 20, we have that $\mathcal{D}^h$ is a complete abstraction of $\mathcal{D}^l$ wrt m. □

A.4. Monitoring and explanation

Theorem 23. *For any refinement mapping m from $\mathcal{D}_h$ to $\mathcal{D}_l$, we have that:*

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

Proof. (1) We have that $\mathcal{D}_l \cup C \models Do(\text{ANYSEQHL}, S_0, S_0)$ since ANYSEQHL is a nondeterministic iteration that can execute 0 times. So even if there is no s'' such that $S_0 < s'' \leq s \wedge Do(\text{ANYSEQHL}, S_0, s'')$, the result holds.
(2) Take an arbitrary model M_l of $\mathcal{D}_l \cup C$ and assume that $M_l, v \models lp_m(s, s_1) \wedge lp_m(s, s_2)$. We have that $\mathcal{D}_l \cup C \models lp_m(s, s') \supset s' \leq s$. Moreover, we have a total ordering on situations s' such that $s' \leq s$. If $M_l, v \models s_1 < s_2$, then s_1 can't be the largest prefix of s that can be produced by executing a sequence of high-level actions, and we can't have $M_l, v \models lp_m(s, s_1)$. Similarly if $M_l, v \models s_2 < s_1$, we can't have $M_l, v \models lp_m(s, s_2)$. It follows that $M_l, v \models s_1 = s_2$. □

Theorem 24. *Suppose that we have a refinement mapping m from $\mathcal{D}_h$ to $\mathcal{D}_l$ and that Constraint 1 holds. Let M_l be a model of $\mathcal{D}_l \cup C$. 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 $M_l \models Do(m(\vec{\alpha}), S_s, S_e)$.*

Proof. Since, $M_l \models Do(\text{ANYSEQHL}, S_s, S_e)$, there exists a $n \in \mathbb{N}$ such that $M_l \models Do(\text{ANY1HL}^n, S_0, S)$. Since we have standard names for objects, it follows that there exists a ground high-level action sequence $\vec{\alpha}$ such that $M_l \models Do(m(\vec{\alpha}), S_s, S_e)$. Now let's show by induction on the length of $\vec{\alpha}$ that there is no ground high-level action sequence $\vec{\alpha}' \neq \vec{\alpha}$ such that $M_l \models Do(m(\vec{\alpha}'), S_s, S_e)$. Base case $\vec{\alpha} = \epsilon$: Then $M_l \models Do(m(\vec{\alpha}), S_s, S_e)$ implies $M_l \models S_s = S_e$ and there is no $\vec{\alpha}' \neq \epsilon$ such that $M_l \models Do(m(\vec{\alpha}'), S_s, S_e)$, since by Constraint 1(c) $\mathcal{D}_l \cup C \models Do(m(\beta), s, s') \supset s < s'$ for any ground high-level action term β. Induction step: Assume that the claim holds for any $\vec{\alpha}$ of length k. Let's show that it must hold for any $\vec{\alpha}$ of length $k+1$. Let $\vec{\alpha} = \beta\vec{\gamma}$. There exists S_i such that $Do(m(\beta), S_s, S_i) \wedge S_i \leq S_e$. By Constraint 1(a), there is no $\beta' \neq \beta$ and S_i' such that $Do(m(\beta'), S_s, S_i') \wedge S_i' \leq S_e$. By Constraint 1(b), there is no $S_i' \neq S_i$ such that $Do(m(\beta), S_0, S_i') \wedge S_i' \leq S$. Then by the induction hypothesis, there is no ground high-level action sequence $\vec{\gamma}' \neq \vec{\gamma}$ such that $M_l \models Do(m(\vec{\gamma}'), S_i, S_e)$. □

Theorem 25. *If m is a refinement mapping from $\mathcal{D}_h$ to $\mathcal{D}_l$ and Constraint 2 holds, then we have that:*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s, s'.Executable(s) \wedge lp_m(s, s') \supset \exists \delta.Trans^*(ANY1HL, s', \delta, s).$$

Proof. Take an arbitrary model M_l of $\mathcal{D}_l \cup \mathcal{C}$ and assume that $M_l, v \models Executable(s) \wedge lp_m(s, s')$. Since $M_l, v \models lp_m(s, s')$, we have that $M_l, v \models Do(\text{ANYSEQHL}, S_0, s')$ and thus that $M_l, v \models Trans^*(\text{ANYSEQHL}, S_0, \text{ANYSEQHL}, s')$. Since $M_l, v \models Executable(s)$, by Constraint 2 we have that $M_l, v \models \exists \delta.Trans^*(\text{ANYSEQHL}, S_0, \delta, s)$. Thus, it follows that $M_l, v \models \exists \delta.Trans^*(\text{ANY1HL}, s', \delta, s)$. □

Appendix B. Additional details on ConGolog semantics

The definitions of $Trans$ and $Final$ for the ConGolog constructs used in this paper are as in [32]. Note that since $Trans$ and $Final$ take programs (that include tests of formulas) as arguments, this requires encoding formulas and programs as terms; see [27] for details.

The predicate $Trans$ is characterized by the following set of axioms:

$$Trans(\alpha, s, \delta', s') \equiv s' = do(\alpha, s) \wedge Poss(\alpha, s) \wedge \delta' = True?$$
$$Trans(\varphi?, s, \delta', s') \equiv False$$
$$Trans(\delta_1; \delta_2, s, \delta', s') \equiv$$
$$\quad Trans(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1'; \delta_2 \vee$$
$$\quad Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s')$$
$$Trans(\delta_1 | \delta_2, s, \delta', s') \equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s')$$
$$Trans(\pi x.\delta, s, \delta', s') \equiv \exists x.Trans(\delta, s, \delta', s')$$
$$Trans(\delta^*, s, \delta', s') \equiv Trans(\delta, s, \delta'', s') \wedge \delta' = \delta''; \delta^*$$
$$Trans(\delta_1 \| \delta_2, s, \delta', s') \equiv$$
$$\quad Trans(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1' \| \delta_2 \vee$$
$$\quad Trans(\delta_2, s, \delta_2', s') \wedge \delta' = \delta_1 \| \delta_2'$$

The predicate $Final$ is characterized by the following set of axioms:

$$Final(\alpha, s) \equiv False$$
$$Final(\varphi?, s) \equiv \varphi[s]$$
$$Final(\delta_1; \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$
$$Final(\delta_1 | \delta_2, s) \equiv Final(\delta_1, s) \vee Final(\delta_2, s)$$
$$Final(\pi x.\delta, s) \equiv \exists x.Final(\delta, s)$$
$$Final(\delta^*, s) \equiv True$$
$$Final(\delta_1 \| \delta_2, s) \equiv Final(\delta_1, s) \wedge Final(\delta_2, s)$$

These are in fact the usual ones [27], except that, following [22], the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Thus, it is a *synchronous* version of the original test construct (it does not allow interleaving).

Also, note that the construct **if** ϕ **then** δ_1 **else** δ_2 **endIf** is defined as $[\phi?; \delta_1] \mid [\neg\phi?; \delta_2]$ and **while** ϕ **do** δ **endWhile** is defined as $(\phi : \delta)^*; \neg\phi?$.

Data availability

No data was used for the research described in the article.

References

[1] J.R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press, 2010.
[2] J.S. Aguas, S.J. Celorrio, A. Jonsson, Hierarchical finite state controllers for generalized planning, in: S. Kambhampati (Ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July, 2016, IJCAI/AAAI Press, 2016, pp. 3235–3241, http://www.ijcai.org/Abstract/16/458.
[3] R. Alford, U. Kuter, D.S. Nau, E. Reisner, R.P. Goldman, Maintaining focus: overcoming attention deficit disorder in contingent planning, in: H.C. Lane, H.W. Guesgen (Eds.), Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference, May 19-21, 2009, Sanibel Island, Florida, USA, AAAI Press, 2009, http://aaai.org/ocs/index.php/FLAIRS/2009/paper/view/128.
[4] C. Baier, J. Katoen, Principles of Model Checking, MIT Press, 2008.
[5] J.A. Baier, S.A. McIlraith, On planning with programs that sense, in: P. Doherty, J. Mylopoulos, C.A. Welty (Eds.), Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2–5, 2006, AAAI Press, 2006, pp. 492–502, http://www.aaai.org/Library/KR/2006/kr06-051.php.
[6] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction in situation calculus action theories, in: S.P. Singh, S. Markovitch (Eds.), Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA, AAAI Press, 2017, pp. 1048–1055, http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14807.
[7] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction of agents executing online and their abilities in the situation calculus, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 1699–1706.

[8] B. Banihashemi, G. De Giacomo, Y. Lespérance, Hierarchical agent supervision, in: E. André, S. Koenig, M. Dastani, G. Sukthankar (Eds.), Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018, International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018, pp. 1432–1440, http://dl.acm.org/citation.cfm?id=3237914.
[9] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction of nondeterministic situation calculus action theories, in: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China, ijcai.org, 2023, pp. 3112–3122.
[10] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction of nondeterministic situation calculus action theories - extended version, CoRR, arXiv:2305.14222, https://doi.org/10.48550/arXiv.2305.14222, 2023.
[11] B. Banihashemi, M. Patel, Y. Lespérance, Using large language models for abstraction of planning domains, in: M. Thimm, J. Fichte, J.C. Jung (Eds.), Proceedings of the First International Workshop on Trends in Knowledge Representation and Reasoning (TKR'25), Zenodo (2025), https://doi.org/10.5281/zenodo.15799038.
[12] F. Belardinelli, A. Lomuscio, J. Michaliszyn, Agent-based refinement for predicate abstraction of multi-agent systems, in: G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, the Hague, the Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), IOS Press, 2016, pp. 286–294.
[13] V. Belle, Abstracting probabilistic models: relations, constraints and beyond, Knowl.-Based Syst. 199 (2020) 105976, https://doi.org/10.1016/j.knosys.2020.105976.
[14] V. Belle, G. Lakemeyer, Reasoning about probabilities in unbounded first-order dynamical domains, in: C. Sierra (Ed.), Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, ijcai.org, 2017, pp. 828–836.
[15] J. van Benthem, Modal Correspondence Theory, Ph.D. thesis, University of Amsterdam, 1977.
[16] P. Blackburn, J. van Benthem, Modal logic: a semantic perspective, in: Handbook of Modal Logic, in: Studies in Logic and Practical Reasoning, vol. 3, North-Holland, 2007, pp. 1–84.
[17] B. Bonet, H. Geffner, Features, projections, and representation change for generalized planning, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 4667–4673.
[18] A. Cimatti, M. Pistore, M. Roveri, P. Traverso, Weak, strong, and strong cyclic planning via symbolic model checking, Artif. Intell. 147 (2003) 35–84, https://doi.org/10.1016/S0004-3702(02)00374-0.
[19] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: E.A. Emerson, A.P. Sistla (Eds.), Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings, Springer, 2000, pp. 154–169.
[20] E.M. Clarke, O. Grumberg, D.E. Long, Model checking and abstraction, ACM Trans. Program. Lang. Syst. 16 (1994) 1512–1542, https://doi.org/10.1145/186025.186051.
[21] J. Claßen, P. Eyerich, G. Lakemeyer, B. Nebel, Towards an integration of golog and planning, in: IJCAI, 2007, pp. 1846–1851.
[22] J. Claßen, G. Lakemeyer, A logic for non-terminating golog programs, in: G. Brewka, J. Lang (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008, AAAI Press, 2008, pp. 589–599, http://www.aaai.org/Library/KR/2008/kr08-058.php.
[23] Z. Cui, W. Kuang, Y. Liu, Automatic verification of sound abstractions for generalized planning, CoRR, arXiv:2205.11898, https://doi.org/10.48550/arXiv.2205.11898, 2022.
[24] Z. Cui, Y. Liu, K. Luo, A uniform abstraction framework for generalized planning, in: Z. Zhou (Ed.), Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021, ijcai.org, 2021, pp. 1837–1844.
[25] G. De Giacomo, P. Felli, B. Logan, F. Patrizi, S. Sardiña, Situation calculus for controller synthesis in manufacturing systems with first-order state representation, Artif. Intell. 302 (2022) 103598, https://doi.org/10.1016/j.artint.2021.103598.
[26] G. De Giacomo, Y. Lespérance, The nondeterministic situation calculus, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online Event, November 3-12, 2021, pp. 216–226.
[27] G. De Giacomo, Y. Lespérance, H.J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Artif. Intell. 121 (2000) 109–169, https://doi.org/10.1016/S0004-3702(00)00031-X.
[28] G. De Giacomo, Y. Lespérance, M. Mancanelli, Situation calculus temporally lifted abstractions for generalized planning, in: T. Walsh, J. Shah, Z. Kolter (Eds.), AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, AAAI Press, 2025, pp. 14848–14857.
[29] G. De Giacomo, Y. Lespérance, C.J. Muise, On supervising agents in situation-determined ConGolog, in: W. van der Hoek, L. Padgham, V. Conitzer, M. Winikoff (Eds.), International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes), IFAAMAS, 2012, pp. 1031–1038, http://dl.acm.org/citation.cfm?id=2343844.
[30] G. De Giacomo, Y. Lespérance, F. Patrizi, Bounded situation calculus action theories, Artif. Intell. 237 (2016) 172–203, https://doi.org/10.1016/j.artint.2016.04.006.
[31] G. De Giacomo, Y. Lespérance, F. Patrizi, S. Sardiña, Verifying ConGolog programs on bounded situation calculus theories, in: D. Schuurmans, M.P. Wellman (Eds.), Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA, AAAI Press, 2016, pp. 950–956, http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12470.
[32] G. De Giacomo, Y. Lespérance, A.R. Pearce, Situation calculus based programs for representing and reasoning about game structures, in: F. Lin, U. Sattler, M. Truszczynski (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010, AAAI Press, 2010, http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1388.
[33] G. De Giacomo, H.J. Levesque, An incremental interpreter for high-level programs with sensing, in: H.J. Levesque, F. Pirri (Eds.), Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter, Springer, Berlin, 1999, pp. 86–102.
[34] H. Dong, Z. Shi, H. Zeng, Y. Liu, An automatic sound and complete abstraction method for generalized planning with baggable types, in: T. Walsh, J. Shah, Z. Kolter (Eds.), AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, AAAI Press, 2025, pp. 14875–14884.
[35] R.J. Doyle, Constructing and refining causal explanations from an inconsistent domain theory, in: T. Kehler (Ed.), Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science, Morgan Kaufmann, 1986, pp. 538–544, http://www.aaai.org/Library/AAAI/1986/aaai86-091.php.
[36] K. Erol, J.A. Hendler, D.S. Nau, Complexity results for HTN planning, Ann. Math. Artif. Intell. 18 (1996) 69–93, https://doi.org/10.1007/BF02136175.
[37] L. Fang, X. Wang, Z. Chen, K. Luo, Z. Cui, Q. Guan, A syntactic approach to computing complete and sound abstraction in the situation calculus, in: T. Walsh, J. Shah, Z. Kolter (Eds.), AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, AAAI Press, 2025, pp. 14911–14921.
[38] A. Gabaldon, Programming hierarchical task networks in the situation calculus, in: AIPS 02 Workshop on On-line Planning and Scheduling, 2002.
[39] H. Geffner, B. Bonet, A concise introduction to models and methods for automated planning, in: Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2013.
[40] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning - Theory and Practice, Elsevier, 2004.
[41] M. Ghallab, D.S. Nau, P. Traverso, Automated Planning and Acting, Cambridge University Press, 2016, http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB.
[42] C. Ghidini, F. Giunchiglia, Local models semantics, or contextual reasoning=locality+compatibility, Artif. Intell. 127 (2001) 221–259, https://doi.org/10.1016/S0004-3702(01)00064-9.

[43] C. Ghidini, F. Giunchiglia, A semantics for abstraction, in: R.L. de Mántaras, L. Saitta (Eds.), Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004, IOS Press, 2004, pp. 343–347.
[44] F. Giunchiglia, T. Walsh, A theory of abstraction, Artif. Intell. 5 (1992) 323–389.
[45] L. Guan, K. Valmeekam, S. Sreedharan, S. Kambhampati, Leveraging pre-trained large language models to construct and utilize world models for model-based task planning, CoRR, arXiv:2305.14909, 2023.
[46] J.R. Hobbs, Granularity, in: A.K. Joshi (Ed.), Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, CA, USA, August 1985, Morgan Kaufmann, 1985, pp. 432–435, http://ijcai.org/Proceedings/85-1/Papers/084.pdf.
[47] W. Hodges, A Shorter Model Theory, Cambridge University Press, 1997.
[48] T. Hofmann, V. Belle, Abstracting noisy robot programs, in: N. Agmon, B. An, A. Ricci, W. Yeoh (Eds.), Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023, 2023, pp. 534–542, https://dl.acm.org/doi/10.5555/3545946.3598681.
[49] Y. Hu, G. De Giacomo, Generalized planning: synthesizing plans that work for multiple environments, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, IJCAI/AAAI, 2011, pp. 918–923.
[50] N. Immerman, M.Y. Vardi, Model checking and transitive-closure logic, in: O. Grumberg (Ed.), Computer Aided Verification, 9th International Conference, CAV'97, Haifa, Israel, June 22-25, 1997, Proceedings, Springer, 1997, pp. 291–302.
[51] R.E. Korf, Planning as search: a quantitative approach, Artif. Intell. 33 (1987) 65–88.
[52] D. Kuske, N. Schweikardt, First-order logic with counting, in: 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, IEEE Computer Society, 2017, pp. 1–12.
[53] M. Lenzerini, Data integration: a theoretical perspective, in: L. Popa, S. Abiteboul, P.G. Kolaitis (Eds.), Proceedings of the Twenty-First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA, 2002, pp. 233–246.
[54] Y. Lespérance, G. De Giacomo, M. Rostamigiv, S.M. Khan, Abstraction of situation calculus concurrent game structures, in: M.J. Wooldridge, J.G. Dy, S. Natarajan (Eds.), Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada, AAAI Press, 2024, pp. 10624–10634.
[55] Y. Lespérance, H.J. Levesque, F. Lin, R.B. Scherl, Ability and knowing how in the situation calculus, Stud. Log. 66 (2000) 165–186, https://doi.org/10.1023/A:1026761331498.
[56] H.J. Levesque, Planning with loops, in: L.P. Kaelbling, A. Saffiotti (Eds.), IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005, Professional Book Center, 2005, pp. 509–515, http://ijcai.org/Proceedings/05/Papers/0305.pdf.
[57] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R.B. Scherl, GOLOG: a logic programming language for dynamic domains, J. Log. Program. 31 (1997) 59–83, https://doi.org/10.1016/S0743-1066(96)00121-5.
[58] A. Lomuscio, H. Qu, F. Raimondi, MCMAS: an open-source model checker for the verification of multi-agent systems, Int. J. Softw. Tools Technol. Transf. 19 (2017) 9–30, https://doi.org/10.1007/s10009-015-0378-x.
[59] K. Luo, Automated verification of propositional agent abstraction for classical planning via CTLK model checking, in: B. Williams, Y. Chen, J. Neville (Eds.), Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023, AAAI Press, 2023, pp. 6475–6482, https://ojs.aaai.org/index.php/AAAI/article/view/25796.
[60] K. Luo, Y. Liu, Automatic verification of FSA strategies via counterexample-guided local search for invariants, in: S. Kraus (Ed.), Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019, 2019, pp. 1814–1821.
[61] K. Luo, Y. Liu, Y. Lespérance, Z. Lin, Agent abstraction via forgetting in the situation calculus, in: G. De Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), IOS Press, 2020, pp. 809–816.
[62] K. Luo, Y. Liu, Y. Lespérance, Z. Lin, Agent abstraction via forgetting in the situation calculus, in: G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), IOS Press, 2020, pp. 809–816.
[63] J. McCarthy, P.J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, Mach. Intell. 4 (1969) 463–502.
[64] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL – the planning Domain Definition Language, Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.
[65] S.A. McIlraith, R. Fadel, Planning with complex actions, in: Proceedings of the 9th International Workshop on Non-Monotonic Reasoning, 2002, pp. 356–364.
[66] R. Milner, An algebraic definition of simulation between programs, in: D.C. Cooper (Ed.), Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971, William Kaufmann, 1971, pp. 481–489, http://ijcai.org/Proceedings/71/Papers/044.pdf.
[67] R. Milner, Communication and Concurrency, PHI Series in Computer Science, Prentice Hall, 1989.
[68] P. Mo, N. Li, Y. Liu, Automatic verification of Golog programs via predicate abstraction, in: G.A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, the Hague, the Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), IOS Press, 2016, pp. 760–768.
[69] R.C. Moore, A formal theory of knowledge and action, in: Formal Theories of the Commonsense World, Ablex Publishing Corporation, Norwood, New Jersey, 1985, pp. 319–358.
[70] P.P. Nayak, A.Y. Levy, A semantic theory of abstractions, in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes, Morgan Kaufmann, 1995, pp. 196–203, http://ijcai.org/Proceedings/95-1/Papers/026.pdf.
[71] E.P.D. Pednault, ADL: exploring the middle ground between STRIPS and the situation calculus, in: Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Morgan Kaufmann, 1989, pp. 324–332.
[72] W. Penczek, A. Lomuscio, Verifying epistemic properties of multi-agent systems via bounded model checking, in: The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings, ACM, 2003, pp. 209–216.
[73] F. Pirri, R. Reiter, Some contributions to the metatheory of the situation calculus, J. ACM 46 (1999) 325–361, https://doi.org/10.1145/316542.316545.
[74] D.A. Plaisted, Theorem proving with abstraction, Artif. Intell. 16 (1981) 47–108, https://doi.org/10.1016/0004-3702(81)90015-1.
[75] R. Reiter, Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems, The MIT Press, 2001.
[76] E.D. Sacerdoti, Planning in a hierarchy of abstraction spaces, Artif. Intell. 5 (1974) 115–135, https://doi.org/10.1016/0004-3702(74)90026-5.
[77] L. Saitta, J.D. Zucker, Abstraction in Artificial Intelligence and Complex Systems, Springer-Verlag, New York, 2013.
[78] S. Sardiña, G. De Giacomo, Y. Lespérance, H.J. Levesque, On the semantics of deliberation in IndiGolog - from theory to implementation, Ann. Math. Artif. Intell. 41 (2004) 259–299, https://doi.org/10.1023/B:AMAI.0000031197.13122.aa.
[79] O. Schulte, J.P. Delgrande, Representing von Neumann-Morgenstern games in the situation calculus, Ann. Math. Artif. Intell. 42 (2004) 73–101, https://doi.org/10.1023/B:AMAI.0000034523.18162.6b.
[80] S. Shapiro, Y. Lespérance, H.J. Levesque, The cognitive agents specification language and verification environment for multiagent systems, in: The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings, ACM, 2002, pp. 19–26.

[81] S. Shoham, O. Grumberg, Monotonic abstraction-refinement for CTL, in: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings, Springer, 2004, pp. 546–560.
[82] P. Smirnov, F. Joublin, A. Ceravola, M. Gienger, Generating consistent PDDL domains with large language models, CoRR, arXiv:2404.07751, 2024.
[83] S. Srivastava, N. Immerman, S. Zilberstein, A new representation and associated algorithms for generalized planning, Artif. Intell. 175 (2011) 615–647, https://doi.org/10.1016/j.artint.2010.10.006.
[84] Stanford Encyclopedia of Philosophy. Model theory, https://plato.stanford.edu/entries/model-theory/.
[85] J.D. Tenenberg, Preserving consistency across abstraction mappings, in: J.P. McDermott (Ed.), Proceedings of the 10th International Joint Conference on Artificial Intelligence, Milan, Italy, August 23-28, 1987, Morgan Kaufmann, 1987, pp. 1011–1014, http://ijcai.org/Proceedings/87-2/Papers/090.pdf.