An Incremental Method for Synthesizing Action Theory Abstractions

Bita Banihashemi[1], Yves Lespérance[2] and Matteo Mancanelli[3]

[1]IGDORE, Gothenburg, Sweden
[2]York University, Toronto, ON, Canada
[3]Sapienza University of Rome, Rome, Italy

Abstract

Abstraction is widely used in reasoning about action, for instance to facilitate planning or to provide explanations of agent behavior. In this paper, we present an incremental method to synthesize an abstract basic action theory (BAT) in the situation calculus that suppresses uninteresting details from a concrete BAT based on specifications provided by a knowledge engineer. The engineer can introduce abstract actions that are implemented by/mapped into Golog programs over the low-level theory and drop some low-level actions from the abstract BAT. She can also perform state abstraction by introducing high-level fluents that are mapped to low-level state formulas and drop some low-level fluents from the abstract BAT. The method allows the engineer to synthesize the abstract BAT step-by-step, providing the mapping along the way. The abstraction steps are validated to ensure that the resulting high-level BAT is guaranteed to be a sound and/or complete abstraction of the original low-level BAT. Abstraction steps that would fail to yield a sound/complete abstraction are rejected with an explanation for the failure, which guides the engineer towards a solution. Regression and SMT reasoning are used to automatically generate the axiomatization of the new abstract actions and fluents.

Keywords

Situation Calculus, Action Theories, Abstraction

1. Introduction

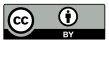
The ability to generate abstractions that ignore irrelevant details is a crucial human cognitive ability that supports reasoning and communication. When one wants to solve a problem, one looks for a suitable abstraction of the domain, i.e., a model that captures the aspects of the domain that are relevant for solving it. In dynamic domains, this applies to planning. Abstraction is also important in explanation. To explain to a user why a particular action or plan must be performed, one needs to obtain a model of the domain that the user can understand. The topic of abstraction has inspired significant research in AI, where for instance, abstraction has been exploited to improve the efficiency of planning (e.g., [1]), provide explanations of agents' behavior (e.g., [2]), and in reinforcement learning (e.g., [3]).

[4, 5] developed a general framework for *agent abstraction* in *dynamic domains* which is based on the situation calculus [6, 7] and the ConGolog agent programming language [8]. This framework formalizes notions of sound/complete abstractions between a high-level (HL) action theory and a low-level (LL) action theory representing the agent's possible behaviors at different levels of detail. These notions are based on the existence of a bisimulation relation between their respective models relative to a *refinement mapping* that maps high-level fluents to low-level state formulas and high-level actions to ConGolog programs over the low-level theory that implement them. Sound/complete abstractions have many useful properties that allow one to reason at the high level and refine the results at the low level, and they can also be used for monitoring and explanation.

While [5] specifies under what conditions a HL action theory is a sound/complete abstraction of a LL

Joint proceedings of the Workshop on Theory and Methods for Abstraction (THEMA 2026) and the Workshop on Modularity and Splitting Techniques for Knowledge Representation and Reasoning (MoST 2026) at the 9th Federated Logic Conference (FLoC 2026), July 13–29, Lisbon, Portugal

bita.banihashemi@igdore.org (B. Banihashemi); lesperan@eecs.yorku.ca (Y. Lespérance); mancanelli@diag.uniroma1.it (M. Mancanelli)
0000-0002-7147-484X (B. Banihashemi); 0000-0003-1625-0226 (Y. Lespérance); 0009-0004-9547-7115 (M. Mancanelli)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-15

CEUR-WS.org/Vol-4331/paper1.pdf

one under a given refinement mapping, it does not say how one obtains such an abstract action theory or refinement mapping. [9] shows that action theory abstraction can be viewed as a form of forgetting, which provides a way to synthesize an abstract action theory given a LL theory and a mapping, but the theories produced are second order in general. [10] develops an automated method for synthesizing a propositional high-level action theory that is a sound and complete abstraction wrt a restricted type of refinement mapping under certain conditions. Much work in generalized planning proceeds by obtaining an abstract version of the planning domain and then generating a (typically iterative) strategy that solves the abstracted version of the planning problem in this abstracted domain. The solution plan can then be used to solve any instance of the generalized planning problem in the concrete domain [11]. [12] proposes a general abstraction framework for solving generalized planning problems. Based on this framework and using the SMT solver Z3, [13] develops a verification system for soundness of bounded qualitative numeric planning (BQNP) abstractions of generalized planning problems. [14] develops an automatic method for generating sound and complete BQNP abstractions for generalized planning from classical planning problem instances with baggable types. Note that in generalized planning one focuses on a specific goal; instead in action theory abstraction, one wants to capture all of the ways that the agent may change the world by performing a range of abstract actions.

In this paper, we address the problem of engineering action theory abstractions. We propose an incremental method for synthesizing a BAT in the situation calculus that suppresses uninteresting details from a concrete BAT based on specifications provided by a knowledge engineer. The engineer can introduce abstract actions that are implemented by Golog programs over the low-level theory and drop some low-level actions from the abstract BAT. She can also perform state abstraction by introducing high-level fluents that are mapped to low-level state formulas and drop some low-level fluents from the abstract BAT. The proposed action and fluent mappings are tentative: our method checks whether they can support a sound and/or complete abstraction and, when they can, generates the corresponding high-level precondition and successor state axioms. If a step fails, the method rejects it with diagnostic information that guides the engineer in revising the abstraction.

2. Background

2.1. Situation Calculus and High-Level Programs

Situation Calculus The *situation calculus* [6, 7] is a well-known predicate logic language for representing and reasoning about dynamically changing worlds [6, 7]. 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, and the function $do(a, s)$ denotes the successor situation after executing action a in situation s. We write $s \leq s'$ if s' can be obtained from s by performing a finite sequence of executable actions $\vec{a}$, i.e. $s' = do(\vec{a}, s)$. 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. A predicate $Poss(a, s)$ is used to state that a is executable in 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 BATs as proposed in [7]. We will use $\mathcal{D}_{ca}$ to denote unique name axioms for actions and domain closure on action types, and $\mathcal{D}_{coa}$ to denote unique name axioms and domain closure for object constants. In BATs, we can rely on the mechanism of regression [7, 15] to reduce reasoning about a given future situation to reasoning about S_0.

ConGolog. ConGolog is a high-level language for writing programs that are executed over a BAT. Here we concentrate on (a variant of) ConGolog that includes the following constructs:

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

where, 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. As usual, we denote by $\varphi[s]$ the formula obtained by restoring the situation argument s into all fluents in φ; this allows formulas to be evaluated

at a specific 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 binding for object variable x. 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. The construct **if** ϕ **then** δ_1 **else** δ_2 **endIf** is defined as $[\phi?; \delta_1] \mid [\neg\phi?; \delta_2]$. We also use nil, an abbreviation for $True?$, to represent the *empty program*.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates [8]: *(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 *(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 [8], except that the test construct $\varphi?$ does not yield any transition, but is final when satisfied [16, 17].

Predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has as a legal terminating situation s'. Formally, $Do(\delta, s, s') \doteq \exists\delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$, where *Trans** denotes the reflexive transitive closure of *Trans*. A ConGolog program δ is *situation-determined* (SD) in a situation s [18] if for every sequence of actions, 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*$^*(\delta, s, \delta'', s') \supset \delta' = \delta''$. In the following, we use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

Regression To compute state descriptions for executability conditions and effects of Golog programs, [19, 9] define existentially extended regression. Given a situation-suppressed formula ϕ, $\mathcal{R}[\phi(s), \delta]$ produces a state formula that is the weakest precondition for the existence of an execution of δ starting from s that achieves ϕ. It is inductively defined as follows:

$$\mathcal{R}[\phi(s), \alpha] \doteq \mathcal{R}_D[Poss(\alpha, s) \wedge \phi(do(\alpha, s))] \qquad \mathcal{R}[\phi(s), \psi?] \doteq \psi[s] \wedge \phi(s)$$
$$\mathcal{R}[\phi(s), \delta_1; \delta_2] \doteq \mathcal{R}[\mathcal{R}[\phi(s), \delta_2], \delta_1] \qquad \mathcal{R}[\phi(s), \delta_1|\delta_2] \doteq \mathcal{R}[\phi(s), \delta_1] \vee \mathcal{R}[\phi(s), \delta_2]$$
$$\mathcal{R}[\phi(s), (\pi x)\delta(x)] \doteq (\exists x)\mathcal{R}[\phi(s), \delta(x)] \qquad \mathcal{R}[\phi(s), \delta^*] \doteq [\mathbf{lfp}_{Z,s}\phi(s) \vee \mathcal{R}[Z(s), \delta]](s)$$

where $\mathcal{R}_D$ is the classical regression operator [7, 15]. Note that the case handling nondeterministic iteration $\mathcal{R}[\phi(s), \delta^*]$ is defined using least fixed-point (**lfp**) logic, see [19, 9] for details. We only include it for completeness, as we do not handle abstract actions that use nondeterministic iteration in this paper (this is left for future work). [9] proves that, given a BAT $\mathcal{D}$, a Golog program δ and a situation-suppressed formula ϕ, we have $\mathcal{D} \models \mathcal{R}[\phi(s), \delta] \equiv \exists s'.Do(\delta, s, s') \wedge \phi[s']$.

2.2. Abstracting Agent Behavior

In the agent abstraction framework of [5], there is a high-level (abstract) (HL) BAT $\mathcal{D}_h$ and a low-level (concrete) (LL) basic action theory $\mathcal{D}_l$ representing the agent's possible behaviors at different levels of detail. $\mathcal{D}_h$ (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. $\mathcal{F}_l$). The terms of object sort are assumed to be a countably infinite set $\mathcal{N}$ of standard names for which we have the unique name assumption and domain closure. To relate the HL and LL models/theories, [5] rely on the notions of refinement mapping, and on a variant of bisimulation [20, 21]. Here, we briefly present these notions and define sound/complete abstractions.

Refinement Mapping. A *refinement mapping* m is a function that associates each HL primitive action type A in $\mathcal{A}_h$ to a SD ConGolog program δ_A defined over the LL theory that implements the action, i.e., $m(A(\vec{x})) = \delta_A(\vec{x})$; moreover, m maps each situation-suppressed HL fluent $F(\vec{x})$ in $\mathcal{F}_h$ to a situation-suppressed formula $\phi_F(\vec{x})$ defined over the LL theory that characterizes the concrete conditions under which $F(\vec{x})$ holds. A mapping can be extended to a sequence of actions in the obvious way, i.e., $m_a(\alpha_1; \ldots; \alpha_n) \doteq m_a(\alpha_1); \ldots; m_a(\alpha_n)$ for $n \geq 1$ and $m_a(\epsilon) \doteq nil$. The notation can also be extended so that $m_f(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in ϕ by $m_f(F(\vec{x}))$.

m-Bisimulation. Given M_h a model of $\mathcal{D}_h$, and M_l a model of $\mathcal{D}_l \cup \mathcal{C}$, 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* if $\langle s_h, s_l \rangle \in B$ implies that: (i) $s_h \sim_m^{M_h, M_l} s_l$, i.e., s_h and s_l evaluate each HL primitive fluent the same; (ii) for every HL primitive action type A in $\mathcal{A}_h$, if there exists s_h' s.t. $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$, then there exists s_l' s.t. $M_l \models Do(m(A(\vec{x})), s_l, s_l')$ and $\langle s_h', s_l' \rangle \in B$; and (iii) for every HL primitive action type A in $\mathcal{A}_h$, if there exists s_l' s.t. $M_l \models Do(m(A(\vec{x})), s_l, s_l')$, then there exists s_h' such that $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$ and $\langle s_h', s_l' \rangle \in B$. M_h is *m-bisimilar* to M_l, written $M_h \sim_m M_l$, iff there exists an m-bisimulation relation B between M_h and M_l s.t. $\langle S_0^{M_h}, S_0^{M_l} \rangle \in B$.

Sound/complete abstractions. In [5], $\mathcal{D}_h$ is a *sound abstraction of $\mathcal{D}_l$ relative to m* iff, for all models M_l of $\mathcal{D}_l \cup \mathcal{C}$, there exists a model M_h of $\mathcal{D}_h$ s.t. $M_h \sim_m M_l$. With a sound abstraction, whenever the HL theory *entails* that a sequence of actions is executable and achieves a certain condition, the LL must also entail that there exists an executable refinement of the sequence s.t. the "translated" condition holds afterwards. Moreover, whenever the LL considers the executability of a refinement of a HL action to be satisfiable, then the HL does also. A characterization that provides the basis for automatically verifying that one has a sound abstraction is also given. A dual notion is also defined: $\mathcal{D}_h$ is a *complete abstraction of $\mathcal{D}_l$ relative to m* iff, for all models M_h of $\mathcal{D}_h$, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ s.t. $M_l \sim_m M_h$.

How does one verify that one has a sound abstraction? First, we can define some LL programs that characterize the refinements of HL action/action sequences:

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

Now, we can identify necessary and sufficient conditions for having a sound abstraction:

Theorem 1 (Theorem 10 in [5]). *$\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to mapping m if and only if*

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

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

Intuitively, Theorem 1 reduces soundness of an abstraction to three checks over the low-level theory. Condition (a) requires the mapped HL initial theory to be entailed by the LL initial theory. Condition (b) says that the HL precondition axiom for each abstract action must exactly capture the executability of its LL refinement. Condition (c) says that the HL SSA for each fluent/action pair must exactly capture the effect of executing the corresponding LL refinement.

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

Corollary 2 (Corollary 16 in [5]). *If $\mathcal{D}^h_{S_0}$ is a complete theory (i.e., for any situation suppressed formula ϕ, either $\mathcal{D}^h_{S_0} \models \phi[S_0]$ or $\mathcal{D}^h_{S_0} \models \neg\phi[S_0]$) 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.*

2.3. Abstraction as Forgetting

Abstraction in action theories can be viewed as a form of forgetting. Intuitively, a theory $forget(T, \mu)$ is the result of forgetting μ from the theory T if for every model of $forget(T, \mu)$ there is a model of T that agrees with it about everything except possibly about the interpretation of μ. Forgetting a symbol results in a weaker theory which entails the same set of sentences that are $irrelevant$ to the symbol.

Prior work [9] has shown that, under suitable conditions, sound and complete abstractions can be characterized in terms of sound and complete forgetting of actions and fluents from the low-level theory. However, these approaches may yield second-order theories, making them difficult to use in practice. In contrast, our work focuses on an incremental and constructive procedure, where the knowledge engineer specifies the refinement mapping and the system synthesizes a corresponding high-level BAT while ensuring soundness and/or completeness.

3. A Methodology for Abstraction

How does a knowledge engineer go about obtaining a useful abstraction of a dynamic domain? What formalisms, symbolic reasoning methods, and tools can we develop to support this? Let's explore these questions. We assume that the engineer wants to synthesize an abstract specification of a dynamic domain bottom up, starting from a concrete/LL specification of the domain. We will assume that the LL specification takes the form of a BAT $\mathcal{D}^l$ that is expressed in terms of a finite set of LL fluents $\mathcal{F}^l$ and a finite set of LL atomic actions $\mathcal{A}^l$ (the dynamic domain could also be represented by a PDDL specification, perhaps in the ADL fragment, which can easily be translated to a BAT; but we don't assume that the object domain is finite). We assume that the engineer builds the abstraction in a series of steps, where she introduces some new HL actions and associated HL fluents, and drops some LL actions and fluents. After one step of abstraction, the process can be repeated until all the LL details that should be abstracted have been eliminated. We also assume that the HL specification of the dynamic domain will be represented as a BAT $\mathcal{D}^h$ expressed in terms of a finite set of HL fluents $\mathcal{F}^h$ and a finite set of HL atomic actions $\mathcal{A}^h$. Moreover, we assume that the relationship between the HL and LL BATs is specified by a refinement mapping m, which is also developed incrementally by the engineer. Finally, we assume that $\mathcal{D}^h$ should be a sound and possibly also complete abstraction of $\mathcal{D}^l$ wrt the mapping m.

Since the abstraction proceeds step by step, we in fact have a sequence of more and more abstract BATs $\mathcal{D}^0, \mathcal{D}^1, \ldots, \mathcal{D}^k$, starting from the original LL BAT $\mathcal{D}^l = \mathcal{D}^0$, until we obtain the final desired abstract BAT $\mathcal{D}^h = \mathcal{D}^k$. As we will show, sound (complete) abstraction wrt a mapping is transitive so if $\mathcal{D}^{i+1}$ is a sound (complete) abstraction of $\mathcal{D}^i$ wrt mapping m_i for all $i : 0 \leq i < k$, then $\mathcal{D}^k = \mathcal{D}^h$ will be a sound (complete) abstraction of $\mathcal{D}^0 = \mathcal{D}^l$ wrt the composed mapping $m = m_0 \circ \ldots \circ m_{k-1}$.

For each step in the abstraction process, we assume that the engineer starts with an informal abstraction objective. From this, she produces a set of new HL actions $\mathcal{A}^+$, with each $a_i(\vec{x}) \in \mathcal{A}^+$ mapped into a Golog program that implements it in the LL theory $m(a_i(\vec{x})) = \delta_i(\vec{x})$. Similarly, she also produces a set of new HL fluents $\mathcal{F}^+$, with each $F_i(\vec{x}) \in \mathcal{F}^+$ mapped into a state formula that defines it in the LL theory $m(F_i(\vec{x})) = \phi_i(\vec{x})$. Each new HL action is usually associated with a HL goal and/or effects that it achieves, which can be expressed in terms of the HL fluents. But these effects need not be specified yet apart from the specification of the new HL fluents and their mapping. The engineer also identifies a set $\mathcal{A}^-$ of LL actions that she wants to drop from the HL theory, so that $\mathcal{A}^h = \mathcal{A}^+ \cup \mathcal{A}^l \setminus \mathcal{A}^-$. And similarly, she specifies a set of $\mathcal{F}^-$ of LL fluents that she wants to drop from the HL theory, so that $\mathcal{F}^h = \mathcal{F}^+ \cup \mathcal{F}^l \setminus \mathcal{F}^-$. We assume that the LL actions and fluents that are kept are mapped to themselves. This specification and mapping are tentative. As we will see, our method checks some conditions for the existence of a sound (complete) abstraction wrt such a mapping. In particular, the HL fluents must be mapped in a way that can capture the preconditions and effects of the implementation of the HL actions. If the check fails, some explanation for the failure is given and the engineer has to reformulate the specification until the check succeeds, in which case our method generates the HL BAT that is a sound (complete) abstraction of the LL BAT wrt the mapping.

To obtain a sound abstraction, it must be possible to accurately represent the preconditions and effects of the LL implementation of the HL actions in terms of the HL fluents (conditions (b) and (c) of Theorem 1). A frequent problem is that a HL action can produce many different LL executions and the resulting LL situations do not map to the same HL state. When this happens, the engineer must either replace the problematic HL action by different HL actions that lead to LL situations that map to the same state or introduce additional HL fluents that allow such states to be distinguished.

Example Our running example concerns a travel booking domain. We start by introducing the precondition axioms and the effect axioms of the low-level action theory.

$\mathcal{D}^{ta01}$ includes the following precondition axioms:

$$Poss(selectHotelRoom(c,r,h),s) \equiv AvailableHotelRoom(r,h,s) \wedge Hotel(h,s) \wedge HotelRoom(r,s) \wedge Customer(c,s)$$
$$Poss(payHotelRoom(c,r,h),s) \equiv SelectedHotelRoom(c,r,h,s) \wedge PayMethodAvail(c,h,s) \wedge Hotel(h,s) \wedge HotelRoom(r,s) \wedge Customer(c,s)$$
$$Poss(applyHotelReward(c,r,h),s) \equiv SelectedHotelRoom(c,r,h) \wedge HasRewardPlan(c,h,s) \wedge Hotel(h,s) \wedge HotelRoom(r,s) \wedge Customer(c,s)$$
$$Poss(bookAirBnbRoom(c,r,h),s) \equiv AvailableAirBnbRoom(r,h,s) \wedge PayMethodAvail(c,h,s) \wedge Customer(c,s) \wedge AirBnb(h,s) \wedge AirBnbRoom(r,s)$$

and the following effect axioms:

$$SelectedHotelRoom(c,r,h,do(selectHotelRoom(c,r,h),s))$$
$$\neg AvailableHotelRoom(r,h,do(payHotelRoom(c,r,h),s))$$
$$PaidHotelRoom(c,r,h,do(payHotelRoom(c,r,h),s))$$
$$AppliedHotelReward(c,r,h,do(applyHotelReward(c,r,h),s))$$
$$AppliedHotelReward(c,r,h,s) \wedge rewardPointsBalance(c,h,s) = p \supset rewardPointsBalance(c,h,do(payHotelRoom(c,r,h),s)) = p + 100$$
$$\neg AvailableAirBnbRoom(r,h,do(bookAirBnbRoom(c,r,h),s))$$
$$BookedAirBnbRoom(c,r,h,do(bookAirBnbRoom(c,r,h),s))$$

Consequently, the successor state axioms will be:

$$SelectedHotelRoom(c,r,h,do(a,s)) \equiv a = selectHotelRoom(c,r,h) \vee SelectedHotelRoom(c,r,h,s)$$
$$AvailableHotelRoom(r,h,do(a,s)) \equiv AvailableHotelRoom(r,h,s) \wedge \forall c.a \neq payHotelRoom(c,r,h)$$
$$PaidHotelRoom(c,r,h,do(a,s)) \equiv a = payHotelRoom(c,r,h) \vee PaidHotelRoom(c,r,h,s)$$
$$AppliedHotelReward(c,r,h,do(a,s)) \equiv a = applyHotelReward(c,r,h) \vee AppliedHotelReward(c,r,h,s)$$
$$rewardPointsBalance(c,h,do(a,s)) = p \equiv a = payHotelRoom(c,r,h) \wedge AppliedHotelReward(c,r,h,s) \wedge rewardPointsBalance(c,h,s) = p - 100 \vee rewardPointsBalance(c,h,s) = p$$
$$AvailableAirBnbRoom(r,h,do(a,s)) \equiv AvailableAirBnbRoom(r,h,s) \wedge \forall c.a \neq bookAirBnbRoom(c,r,h)$$
$$BookedAirBnbRoom(c,r,h,do(a,s)) \equiv a = bookAirBnbRoom(c,r,h) \vee BookedAirBnbRoom(c,r,h,s)$$

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

Finally, $\mathcal{D}^{ta01}$ includes the following initial state axioms:

$$Hotel(hotel1,S_0) \quad Customer(mary,S_0)$$
$$HotelRoom(r1,S_0) \quad HotelRoom(r2,S_0)$$
$$AirBnb(ab_1,S_0) \quad AirBnbRoom(r3,S_0)$$
$$AvailableHotelRoom(r1,hotel1,S_0) \quad AvailableHotelRoom(r2,hotel1,S_0)$$
$$AvailableAirBnbRoom(r3,ab_1,S_0) \quad HasRewardPlan(mary,hotel1,S_0)$$
$$PayMethodAvail(mary,hotel1,S_0) \quad PayMethodAvail(mary,ab_1,S_0)$$
$$\neg\exists r,h.SelectedHotelRoom(mary,r,h,S_0) \quad \neg\exists r,h.PaidHotelRoom(mary,r,h,S_0)$$
$$\neg\exists r,h.BookedAirBnbRoom(mary,r,h,S_0) \quad rewardPointsBalance(mary,hotel1,S_0) = 100$$

Suppose that as a first step, we want to abstract over the details of booking a hotel room. For this, we want to introduce a HL action $bookHotelRoom(c,r,h)$ that is mapped to a LL deterministic program as per the following mapping m^{ta01}:

$$m^{ta01}(bookHotelRoom(c,r,h)) = selectHotelRoom(c,r,h); \textbf{ if } HasRewardPlan(c,h) \textbf{ then } applyHotelReward(c,r,h) \textbf{ endif}; payHotelRoom(c,r,h)$$

We want to keep the LL action $bookAirBnbRoom$, so

$$m^{ta01}(bookAirBnbRoom(c,r,h)) = bookAirBnbRoom(c,r,h)$$

and drop all the other LL actions, so

$$\mathcal{A}^h = \{bookHotelRoom(c, r, h), bookAirBnbRoom(c, r, h)\}$$

To capture the goal/effects of the new HL action, we want to introduce a HL fluent $BookedHotelRoom(c, r, h)$ that is mapped to the LL state formula $SelectedHotelRoom(c, r, h) \wedge PaidHotelRoom(c, r, h)$, as we don't need to track the intermediate states we go through when a hotel room is booked. That is, we want a mapping as follows:

$$m^{ta02}(BookedHotelRoom(c, r, h)) = SelectedHotelRoom(c, r, h) \wedge PaidHotelRoom(c, r, h)$$

We assume that the $SelectedHotelRoom$ and $PaidHotelRoom$ fluents should be dropped and that all other fluents should be kept (and mapped to themselves).

Later in the paper, we will see how we can use our method to check that a sound abstraction wrt this mapping exists, and synthesize the desired abstract BAT. We will also examine further ways of abstracting the specification, and negative examples where such abstractions are not sound.

4. Transitivity of Sound/Complete Abstraction

In our methodology, we synthesize a HL theory by performing a sequence of abstraction steps. We can show that this preserves the soundness/completeness of abstractions.

A refinement mapping m is defined on primitive fluents and atomic actions. But it can be easily extended to complex situation-suppressed formulas and Golog programs homomorphically in the standard way, i.e., it commutes with Boolean connectives, quantifiers, and program constructors.[1]

Given two mappings m_1 and m_2, their composition $m = m_1 \circ m_2$ is defined on primitive symbols by $(m_1 \circ m_2)(\sigma) = m_2(m_1(\sigma))$, with σ being either primitive action or fluent. Again, it can be extended homomorphically to formulas and Golog programs.

Theorem 3. *Suppose that $M_h \sim_m M_l$ and $s_h \sim_m^{M_h, M_l} s_l$. Then for any HL SD Golog program δ, we have:*

1. *if $M_h, v[s/s_h, s'/s'_h] \models Do(\delta, s, s')$, then there exists s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m(\delta), s, s')$ and $s'_h \sim_m^{M_h, M_l} s'_l$, and*
2. *if $M_l, v[s/s_l, s'/s'_l] \models Do(m(\delta), s, s')$, then there exists s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Do(\delta, s, s')$ and $s'_h \sim_m^{M_h, M_l} s'_l$.*

Theorem 4. *Suppose that we have $M_1 \models \mathcal{D}_1$, $M_2 \models \mathcal{D}_2$, and $M_3 \models \mathcal{D}_3$. If $M_1 \sim_{m_1} M_2$ and $M_2 \sim_{m_2} M_3$, then $M_1 \sim_{m_1 \circ m_2} M_3$*

Corollary 5. *If $\mathcal{D}_1$ is a sound (resp. complete) abstraction of $\mathcal{D}_2$ wrt mapping m_1 and $\mathcal{D}_2$ is a sound (resp. complete) abstraction of $\mathcal{D}_3$ wrt mapping m_2, then $\mathcal{D}_1$ is a sound (resp. complete) abstraction of $\mathcal{D}_3$ wrt mapping $m_1 \circ m_2$.*

5. Abstraction of Deterministic Programs

In this section, we define a general procedure for introducing abstract actions that are mapped to *deterministic* Golog programs without iteration[2] into a LL $\mathcal{D}_l$ theory and obtaining a HL theory $\mathcal{D}_h$ that includes them. LL actions may be kept or dropped from the HL theory. Here we assume that the set of fluents remains unchanged; the abstraction of fluents will be considered in the next section.

[1] If we want to extend a mapping m to ConGolog programs, we must restrict the use of the concurrent composition construct to ensure that refinements of HL atomic actions are executed as an atomic unit at the LL. For example, if $m(A) = (a \| b)$ and $m(B) = (c \| d)$, then we should have $Do(m_p(A \| B), s, s') \equiv Do(m_p((A; B) | (B; A)), s, s')$. If we want to allow $m_p(\delta_1 \| \delta_2)$, then we should have $m_p(\alpha) = atomic(m(\alpha))$. For simplicity in this paper, we assume that we only apply a mapping to Golog programs.

[2] Handling unbounded iteration requires fixpoint solving; we leave this for future work.

Suppose that we want to introduce a new HL atomic action $a(\vec{x})$ which is mapped to/implemented by a deterministic Golog program without iteration $\delta_a(\vec{x})$. The preconditions of $a(\vec{x})$ in $\mathcal{D}_h$ can be specified by the following axiom $PreAx_a$:

$$Poss(a(\vec{x}), s) \equiv \mathcal{R}(True(s), \delta_a(\vec{x}))$$

Now let's see how we can obtain the set of effect axioms for $a(\vec{x})$ given $\mathcal{D}_l$. Observe that a fluent $F(\vec{y})$ is a positive effect of $a(\vec{x})$ in s iff $\neg F(\vec{y}, s) \wedge \mathcal{R}(F(\vec{y}, s), \delta_a(\vec{x}))$ holds in s. So we can obtain a positive effect axiom for $a(\vec{x})$ and fluent F as

$$\phi^+_{a,F}(\vec{z}, s) \supset F(\vec{y}, do(a(\vec{x}), s))$$

where $\mathcal{D}_l \models \phi^+_{a,F}(\vec{z}, s) \equiv \neg F(\vec{y}, s) \wedge \mathcal{R}(F(\vec{y}, s), \delta_a(\vec{x}))$. If $\mathcal{D}_l \models \phi^+_{a,F}(\vec{z}, s) \equiv False$, we can leave this out as $a(\vec{x})$ cannot make F become true. Similarly, we can obtain a negative effect axiom for $a(\vec{x})$ and fluent F as

$$\phi^-_{a,F}(\vec{z}, s) \supset \neg F(\vec{y}, do(a(\vec{x}), s))$$

where $\mathcal{D}_l \models \phi^-_{a,F}(\vec{z}, s) \equiv F(\vec{y}, s) \wedge \mathcal{R}(\neg F(\vec{y}, s), \delta_a(\vec{x}))$. Note that any fluent that does not appear in the effect axioms of some LL action in $\delta_a(\vec{x})$ is not affected by it, so there won't be any effect axioms for those. We denote by $\mathit{Eff}(a(\vec{x}), \delta_a(\vec{x}), \mathcal{D}_l)$ the set of effect axioms for the new HL action $a(\vec{x})$ mapped to/implemented by a deterministic Golog program without iteration $\delta_a(\vec{x})$ defined over LL theory $\mathcal{D}_l$ obtained by this procedure.

Consider a mapping m to a LL theory $\mathcal{D}_l$ with a set of LL actions $\mathcal{A}^l$, where we introduce a set $\mathcal{A}^+$ of new HL actions mapped to deterministic Golog programs, i.e., $m(A(\vec{x})) = \delta_a(\vec{x})$, and drop a set of LL actions $\mathcal{A}^-$. Thus the set of HL actions is $\mathcal{A}^h = \mathcal{A}^+ \cup (\mathcal{A}^l \setminus \mathcal{A}^-)$. LL actions that are kept (formally denoted by $\mathcal{A}^h_l = \mathcal{A}^l \setminus \mathcal{A}^-$) are mapped to themselves, i.e., $m(A(\vec{x})) = A(\vec{x})$. We assume that the set of fluents remains unchanged i.e., $\mathcal{F}^h = \mathcal{F}^l$ and $m(F(\vec{x})) = F(\vec{x})$ for all fluents.

We define the abstraction method $DPabs(\mathcal{A}^+, \mathcal{A}^-, \mathcal{A}^l, D_l, m)$ of LL theory $\mathcal{D}_l$ wrt such a mapping m obtained by the method described above, that is:

- $\mathcal{D}^h_{S_0} = \mathcal{D}^l_{S_0}$,
- $\mathcal{D}^h_{ap} = \{PreAx_a | a \in \mathcal{A}^+ \cup \mathcal{A}^h_l\}$,
- $\mathcal{D}^h_{\mathit{eff}} = \bigcup_{a \in \mathcal{A}^+_h} \mathit{Eff}(a(\vec{x}), \delta_a(\vec{x}), \mathcal{D}_l) \cup \{\mathit{EffAx}_a | a \in \mathcal{A}^h_l\}$,
- $\mathcal{D}^h_{ssa}$ is the set of SSAs obtained by the usual method given the set of effect axioms in $\mathcal{D}^h_{\mathit{eff}}$,
- $\mathcal{D}_{una}$ is the set of unique names for actions axioms for the set of actions $\mathcal{A}^h$.

We can show that:

Theorem 6. *$DPabs(\mathcal{A}^+, \mathcal{A}^-, \mathcal{A}^l, D_l, m)$ is a sound and complete abstraction of $\mathcal{D}_l$ wrt mapping m.*

Example (cont). As mentioned earlier, we want to introduce a HL action $bookHotelRoom(c, r, h)$ that is mapped to a LL deterministic program as per the following mapping m^{ta01}:

$m^{ta01}(bookHotelRoom(c, r, h)) = selectHotelRoom(c, r, h);$
if $HasRewardPlan(c, h)$ **then** $applyHotelReward(c, r, h)$ **endif**; $payHotelRoom(c, r, h)$

We want to keep the LL action $bookAirBnbRoom$, so

$$m^{ta01}(bookAirBnbRoom(c, r, h)) = bookAirBnbRoom(c, r, h)$$

and drop all the other LL actions, so

$\mathcal{A}^+ = \{bookHotelRoom(c, r, h)\}$
$\mathcal{A}^- = \{selectHotelRoom(c, r, h), applyHotelReward(c, r, h), payHotelRoom(c, r, h)\}$
$\mathcal{A}^h = \{bookHotelRoom(c, r, h), bookAirBnbRoom(c, r, h)\}$

For now, we can take the set of HL fluents to be identical to the LL one and for every LL fluent $F(\vec{x}, s)$, we have $m^{ta01}(F(\vec{x})) = F(\vec{x})$.

Using the above procedure, we can obtain a HL BAT $\mathcal{D}^{ta02} = DPabs(\mathcal{A}^+, \mathcal{A}^-, \mathcal{A}^l, \mathcal{D}^{ta01}, m^{ta01})$ that is a sound and complete abstraction of $\mathcal{D}^{ta01}$. In particular, $\mathcal{D}^{ta02}$ includes the following precondition axioms:

$$\begin{aligned}
&Poss(bookHotelRoom(c, r, h), s) \equiv AvailableHotelRoom(r, h, s) \wedge \\
&\quad PayMethodAvail(c, h, s) \wedge Customer(c, s) \wedge Hotel(h, s) \wedge HotelRoom(r, s) \\
&Poss(bookAirBnbRoom(c, r, h), s) \equiv AvailableAirBnblRoom(r, h, s) \wedge \\
&\quad PayMethodAvail(c, h, s) \wedge Customer(c, s) \wedge AirBnb(h, s) \wedge AirBnbRoom(r, s)
\end{aligned}$$

Note that the RHS of the precondition axiom for $bookHotelRoom$ above is $\mathcal{R}(True(s), m^{ta01}(bookHotelRoom(c, r, h)))$.

$\mathcal{D}^{ta02}$ includes also the following effect axioms:

$$\begin{aligned}
&SelectedHotelRoom(c, r, h, do(bookHotelRoom(c, r, h), s)) \\
&\neg AvailableHotelRoom(r, h, do(bookHotelRoom(c, r, h), s)) \\
&PaidHotelRoom(c, r, h, do(bookHotelRoom(c, r, h), s)) \\
&HasRewardPlan(c, h, s) \supset AppliedHotelReward(c, r, h, do(bookHotelRoom(c, r, h), s)) \\
&HasRewardPlan(c, h, s) \wedge rewardPointsBalance(c, h, s) = p \supset \\
&\quad rewardPointsBalance(c, h, do(bookHotelRoom(c, r, h), s)) = p + 100 \\
&\neg AvailableAirBnbRoom(r, h, do(bookAirBnbRoom(c, r, h), s)) \\
&BookedAirBnbRoom(c, r, h, do(bookAirBnbRoom(c, r, h), s))
\end{aligned}$$

Note that the antecedent in the effect axiom for $AppliedHotelReward$ is

$$\mathcal{R}(AppliedHotelReward(c, h, s), m^{ta01}(bookHotelRoom(c, r, h)))$$

The SSAs can be derived by the effect axioms as follows:

$$\begin{aligned}
&SelectedHotelRoom(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \vee SelectedHotelRoom(c, r, h, s) \\
&AvailableHotelRoom(r, h, do(a, s)) \equiv AvailableHotelRoom(r, h, s) \wedge \forall c.a \neq bookHotelRoom(c, r, h) \\
&PaidHotelRoom(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \vee PaidHotelRoom(c, r, h, s) \\
&AppliedHotelReward(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \wedge HasRewardPlan(c, h, s) \vee \\
&\quad AppliedHotelReward(c, r, h, s) \\
&rewardPointsBalance(c, h, do(a, s)) = p \equiv a = bookHotelRoom(c, r, h) \wedge HasRewardPlan(c, h, s) \wedge \\
&\quad rewardPointsBalance(c, h, s) = p - 100 \vee rewardPointsBalance(c, h, s) = p \\
&AvailableAirBnbRoom(r, h, do(a, s)) \equiv AvailableAirBnbRoom(r, h, s) \wedge \\
&\quad \forall c.a \neq bookAirBnbRoom(c, r, h) \\
&BookedAirBnbRoom(c, r, h, do(a, s)) \equiv a = bookAirBnbRoom(c, r, h) \vee \\
&\quad BookedAirBnbRoom(c, r, h, s)
\end{aligned}$$

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

Finally, note that $\mathcal{D}^{ta02}$ includes the same initial state axioms as $\mathcal{D}^{ta01}$.

6. State/Predicate Abstraction

In this section, we discuss how we can obtain an abstraction by abstracting states in a BAT. Essentially, we want to introduce some new HL fluents that are mapped to state formulas over the LL fluents and drop some LL fluents. First, we look at how we can add the new HL fluents. Assume that we have a LL BAT $\mathcal{D}_l$ with the set of LL fluents $\mathcal{F}^l$. Consider a mapping m where we introduce a set $\mathcal{F}^+$ of new HL fluents mapped to state formulas over the LL theory $\mathcal{D}_l$, $m(F(\vec{x})) = \phi(\vec{x})$ where $\phi(\vec{x})$ is a situation-suppressed formula over the LL fluents, for all $F \in \mathcal{F}^+$. After this, the set of HL fluents will be $\mathcal{F}^h = \mathcal{F}^l \cup \mathcal{F}^+$. We assume that the set of actions $\mathcal{A}^h = \mathcal{A}^l$ remains unchanged.

We can obtain effect axioms for the new HL fluents as follows. We obtain a positive effect axiom for action $A(\vec{x})$ and new fluent $F(\vec{y})$ as

$$\phi^+_{A,F}(\vec{z}, s) \supset F(\vec{y}, do(A(\vec{x}), s))$$

where $\mathcal{D}_l \models \phi^+_{A,F}(\vec{z}, s) \equiv \neg m(F(\vec{y}))[s] \wedge \mathcal{R}(m(F(\vec{y}))[s], A(\vec{x}))$. If $\mathcal{D}_l \models \phi^+_{A,F}(\vec{z}, s) \equiv False$, we can leave this out as $A(\vec{x})$ cannot make F become true. Similarly, we can obtain a negative effect axiom for action $A(\vec{x})$ and new fluent $F(\vec{y})$ as

$$\phi^-_{A,F}(\vec{z}, s) \supset \neg F(\vec{y}, do(A(\vec{x}), s))$$

where $\mathcal{D}_l \models \phi^-_{A,F}(\vec{z}, s) \equiv m(F(\vec{y}))[s] \wedge \mathcal{R}(\neg m(F(\vec{y}))[s], A(\vec{x}))$. Let's denote the set of effect axioms for the action $A(\vec{x})$ on the new HL fluent $F(\vec{y})$ defined over LL theory $\mathcal{D}_l$ obtained by this procedure as $\mathit{EffOn}(A(\vec{x}), F(\vec{y}), \mathcal{D}_l)$. Using this, we obtain a HL theory $\mathcal{D}^h = AddHLfluents(\mathcal{F}^+, \mathcal{F}^l, D_l, m)$ with fluents $\mathcal{F}^h = \mathcal{F}^l \cup \mathcal{F}^+$ from LL theory $\mathcal{D}^l$ using mapping m as follows:

- $\mathcal{D}^h_{S_0} = \mathcal{D}^l_{S_0} \cup \{F(\vec{x}, S_0) \equiv m(F(\vec{x}))[S_0] | F \in \mathcal{F}^+\}$,
- $\mathcal{D}^h_{ap} = \mathcal{D}^l_{ap}$,
- $\mathcal{D}^h_{\mathit{eff}} = \bigcup_{a \in \mathcal{A}, F \in \mathcal{F}^+} \mathit{EffOn}(a(\vec{x}), F(\vec{y}), \mathcal{D}_l) \cup \mathcal{D}^l_{\mathit{eff}}$,
- $\mathcal{D}^h_{ssa}$ is the set of SSAs obtained by the usual method given the set of effect axioms in $\mathcal{D}^h_{\mathit{eff}}$
- $\mathcal{D}^h_{una} = \mathcal{D}^l_{una}$.

We can show that:

Theorem 7. *$\mathcal{D}^h = AddHLfluents(\mathcal{F}^+, \mathcal{F}^l, D_l, m)$ is a sound and complete abstraction of $\mathcal{D}_l$ wrt m.*

Now, let's examine how we can drop some LL fluents to get a theory that abstracts unnecessary details from the state. Assume that we have a LL BAT $\mathcal{D}_l$ with the set of LL fluents $\mathcal{F}^l$. We want to drop a set of LL fluents $\mathcal{F}^- \subset \mathcal{F}^l$. So after this, the set of HL fluents will be $\mathcal{F}^h = \mathcal{F}^l \setminus \mathcal{F}^-$. We assume that the set $\mathcal{A}^h = \mathcal{A}^l$ of actions remains unchanged. Let's say that two LL situations s and s' are in the same HL state if they evaluate all HL fluents the same:

$$SameHLState(s, s') \doteq \bigwedge_{F \in \mathcal{F}^h} \forall \vec{x}.(m(F(\vec{x}))[s] \equiv m(F(\vec{x}))[s'])$$

Following [9], we say that a mapping m is Markovian if for any pair of LL situations s and s' s.t. they are the results of executing ANYSEQHL from S_0, if $SameHLState(s, s')$, then for any HL action in $\mathcal{A}^h$, the executability condition and action effects are indistinguishable in these two situations. We can show that:

Theorem 8. *Under our assumptions. i.e. $\mathcal{A}^h = \mathcal{A}^l$ and $\mathcal{F}^h = \mathcal{F}^l \setminus \mathcal{F}^-$, m is Markovian wrt $\mathcal{D}^l$ iff*

$$\begin{aligned}\mathcal{D}^l \models \forall s, s'.S_0 \leq s \wedge S_0 \leq s' \wedge SameHLstate(s, s') \supset \\ \forall a.(Poss(a, s) \equiv Poss(a, s')) \wedge \textstyle\bigwedge_{F \in \mathcal{F}^h} \forall \vec{x}.(F(\vec{x}, do(a, s)) \equiv F(\vec{x}, do(a, s')))\end{aligned}$$

Let $\Phi_m = \{F(\vec{x})[s] \equiv m(F(\vec{x}))[s] \mid F \in \mathcal{F}\}$ be a set of axioms specifying the mapping m. The following result identifies sufficient conditions to have a sound and possibly complete abstraction for our case of state abstraction:

Theorem 9. *Suppose that we have a LL BAT $\mathcal{D}^l$ with set of state constraints Φ_{sc}, i.e., for all $\phi \in \Phi_{sc}$, $D^l \models \forall s.S_0 \leq s \supset \phi[s]$, and a Markovian mapping m where $\mathcal{A}^h = \mathcal{A}^l$ and $\mathcal{F}^h = \mathcal{F}^l \setminus \mathcal{F}^-$. If $\mathcal{D}^h$ is a HL BAT with $\mathcal{A}^h = \mathcal{A}^l$ and $\mathcal{F}^h = \mathcal{F}^l \setminus \mathcal{F}^-$ and the following conditions are satisfied:*

1. *$\mathcal{D}^l_{S_0} \cup \Phi_m[S_0] \models m(\mathcal{D}^h_{S_0})$ and $\mathcal{D}^h_{S_0}$ contains no occurrences of fluents in $\mathcal{F}^-$;*
2. *for all $A \in \mathcal{A}^h$, if $Poss(A(\vec{x}), s) \equiv \phi(\vec{x})[s] \in \mathcal{D}^l_{ap}$, then*
   $Poss(A(\vec{x}), s) \equiv \phi'(\vec{x})[s] \in \mathcal{D}^h_{ap}$ and
   $\Phi_{sc} \cup \Phi_m \models \phi(\vec{x}) \equiv \phi'(\vec{x})$ and
   $\phi'(\vec{x})$ contains no occurrences of fluents in $\mathcal{F}^-$;
3. *for all $A \in \mathcal{A}^h$ and all $F \in \mathcal{F}^h$, if $\phi(\vec{z})[s] \supset (\neg)F(\vec{x}, do(A(\vec{y}), s)) \in \mathcal{D}^l_{\mathit{eff}}$, then*
   $\phi'(\vec{z})[s] \supset (\neg)F(\vec{x}, do(A(\vec{y}), s)) \in \mathcal{D}^h_{\mathit{eff}}$ and
   $\Phi_{sc} \cup \Phi_m \models \phi(\vec{z}) \equiv \phi'(\vec{z})$, and
   $\phi'(\vec{z})$ contains no occurrences of fluents in $\mathcal{F}^-$;

4. $\mathcal{D}^h_{ssa}$ *is the set of SSAs obtained by the usual method given the set of effect axioms in* $\mathcal{D}^h_{eff}$,
5. $\mathcal{D}^h_{una} = D^l_{una}$.

Then $\mathcal{D}^h$ *is a sound abstraction of* $\mathcal{D}^l$ *wrt* m.
Moreover, if it is also the case that $m(\mathcal{D}^h_{S_0}) \models \mathcal{D}^l_{S_0}$, *then* $\mathcal{D}^h$ *is also a complete abstraction of* $\mathcal{D}^l$ *wrt mapping* m.

Example (cont.) Let's now perform state abstraction on the abstract BAT obtained in the previous section $\mathcal{D}^{ta02}$. We want to introduce a HL fluent $BookedHotelRoom(c, r, h)$ that is mapped to the LL state formula $SelectedHotelRoom(c, r, h) \wedge PaidHotelRoom(c, r, h)$, as we don't need to track the intermediate states we go through when a hotel room is booked. That is, we want a mapping as follow:

$$m^{ta02}(BookedHotelRoom(c, r, h)) = SelectedHotelRoom(c, r, h) \wedge PaidHotelRoom(c, r, h)$$

We assume that the $SelectedHotelRoom$ and $PaidHotelRoom$ fluents should be dropped and that all other fluents should be kept (and mapped to themselves). Let's see how we can obtain an abstract BAT, $\mathcal{D}^{ta03}$, that is a sound abstraction of $\mathcal{D}^{ta02}$ wrt m^{ta02}.

We can take $\mathcal{D}^{ta03}$ to have the same set of precondition axioms as $\mathcal{D}^{ta02}$ as these do not mention the fluents that we are dropping. If they did, but every occurrence of the dropped LL fluents appeared in subformulas that are equivalent to $\phi(\vec{x})$, the formula into which a HL fluent is mapped, i.e., $m(F(\vec{x})) = \phi(\vec{x})$, then we can just replace these subformulas by the instantiated HL fluent $F(\vec{x})$.

We can take $\mathcal{D}^{ta03}$ to include all the effect axioms in $\mathcal{D}^{ta02}$ except the ones whose RHS involves the fluents we are dropping. Moreover we can try to generate effect axioms for the new fluents[3], e.g., $BookedHotelRoom$, by using regression in the same way that we obtained them for the new HL action in Section 3. If the resulting effect axiom contains occurrences of the dropped LL fluents in its antecedent, we can try to eliminate them using the technique mentioned above for preconditions. For our example, this is easy because only the action $bookHotelRoom(c, r, h)$ affects $BookedHotelRoom(c, r, h)$, making it become true; the effect does not depend on any condition. This yields the following set of effect axioms:

$$\begin{array}{l}
BookedHotelRoom(c, r, h, do(bookHotelRoom(c, r, h), s)) \\
\neg AvailableHotelRoom(r, h, do(bookHotelRoom(c, r, h), s)) \\
HasRewardPlan(c, h, s) \supset AppliedHotelReward(c, r, h, do(bookHotelRoom(c, r, h), s)) \\
HasRewardPlan(c, h, s) \wedge rewardPointsBalance(c, h, s) = p \supset \\
\quad rewardPointsBalance(c, h, do(bookHotelRoom(c, r, h), s)) = p + 100 \\
\neg AvailableAirBnbRoom(r, h, do(bookAirBnbRoom(c, r, h), s)) \\
BookedAirBnbRoom(c, r, h, do(bookAirBnbRoom(c, r, h), s))
\end{array}$$

Given this, we get that $\mathcal{D}^{ta03}$ includes the following successor state axioms:

$$\begin{array}{l}
BookedHotelRoom(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \vee BookedHotelRoom(c, r, h, s) \\
AvailableHotelRoom(r, h, do(a, s)) \equiv AvailableHotelRoom(r, h, s) \wedge \forall c.a \neq bookHotelRoom(c, r, h) \\
AppliedHotelReward(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \wedge HasRewardPlan(c, h, s) \vee \\
\quad AppliedHotelReward(c, r, h, s) \\
rewardPointsBalance(c, h, do(a, s)) = p \equiv a = bookHotelRoom(c, r, h) \wedge HasRewardPlan(c, h, s) \wedge \\
\quad rewardPointsBalance(c, h, s) = p - 100 \vee rewardPointsBalance(c, h, s) = p \\
AvailableAirBnbRoom(r, h, do(a, s)) \equiv AvailableAirBnbRoom(r, h, s) \wedge \\
\quad \forall c.a \neq bookAirBnbRoom(c, r, h) \\
BookedAirBnbRoom(c, r, h, do(a, s)) \equiv a = bookAirBnbRoom(c, r, h) \vee \\
\quad BookedAirBnbRoom(c, r, h, s)
\end{array}$$

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

We can take $\mathcal{D}^{ta03}$ to have the same initial state axioms as $\mathcal{D}^{ta02}$ except that we drop the ones involving $SelectedHotelRoom$ and $PaidHotelRoom$.

[3] In general, we must do this for every new fluent and every action. In practice, the number of fluent/action pairs that need to be considered can often be reduced by a simple dependency analysis: for a new fluent F, it is only necessary to consider actions that may affect some low-level fluent occurring in $m(F)$.

Example (cont.) Now, assume that we want to perform a further abstraction in which reward bookkeeping details are suppressed. In particular we want to abstract away $AppliedHotelReward$ and $rewardPointsBalance$, and introduce a new fluent $HasRewardBenefit(c, h)$ such that

$$m(HasRewardBenefit(c, h)) \equiv HasRewardPlan(c, h) \wedge rewardPointsBalance(c, h) \geq 100$$

All the other fluents remain untouched, i.e. the mapping is an identity function. The new abstract BAT $\mathcal{D}^{ta04}$ must be a sound abstraction of $\mathcal{D}^{ta03}$. Once again, we can take $\mathcal{D}^{ta04}$ to have the same set of precondition axioms as $\mathcal{D}^{ta02}$ as these do not mention the fluents that we are dropping.

The new set of effect axioms will not include the axioms referring to the removed fluents, but it will introduce a new axiom related to the new HL fluent $HasRewardBenefit$:

$$\begin{array}{l} BookedHotelRoom(c, r, h, do(bookHotelRoom(c, r, h), s)) \\ \neg AvailableHotelRoom(r, h, do(bookHotelRoom(c, r, h), s)) \\ HasRewardPlan(c, h, s) \supset HasRewardBenefit(c, h, do(bookHotelRoom(c, r, h), s)) \\ \neg AvailableAirBnbRoom(r, h, do(bookAirBnbRoom(c, r, h), s)) \\ BookedAirBnbRoom(c, r, h, do(bookAirBnbRoom(c, r, h), s)) \end{array}$$

Note that $rewardPointsBalance$ does not occur in the axiom for $HasRewardBenefit$. This is because executing $bookHotelRoom$ for a customer with a reward plan applies the reward and increases the balance by 100 points, and there is no action that decreases it. Hence, the condition $rewardPointsBalance \geq 100$ is guaranteed to hold after the action.

The derived SSAs for $\mathcal{D}^{ta04}$ will be:

$$\begin{array}{l} BookedHotelRoom(c, r, h, do(a, s)) \equiv a = bookHotelRoom(c, r, h) \vee BookedHotelRoom(c, r, h, s) \\ AvailableHotelRoom(r, h, do(a, s)) \equiv AvailableHotelRoom(r, h, s) \wedge \forall c.a \neq bookHotelRoom(c, r, h) \\ HasRewardBenefit(c, h, do(a, s)) \equiv \exists r.a = bookHotelRoom(c, r, h) \wedge HasRewardPlan(c, h, s) \vee \\ \quad HasRewardBenefit(c, h, s) \\ AvailableAirBnbRoom(r, h, do(a, s)) \equiv AvailableAirBnbRoom(r, h, s) \wedge \\ \quad \forall c.a \neq bookAirBnbRoom(c, r, h) \\ BookedAirBnbRoom(c, r, h, do(a, s)) \equiv a = bookAirBnbRoom(c, r, h) \vee \\ \quad BookedAirBnbRoom(c, r, h, s) \end{array}$$

Now suppose that the action theory also includes an action $redeemRewardPoints$ that decreases $rewardPointsBalance$ by 100. Then the reward balance no longer evolves monotonically, and the proposed abstraction is no longer sound. Indeed, two LL situations may agree on all HL fluents while differing on the hidden value of $rewardPointsBalance$. After executing $redeemRewardPoints$, one resulting LL situation may satisfy $HasRewardBenefit$, while the other may not. Thus, the effect of the abstract action depends on hidden LL information, and the proposed abstraction fails.

7. Future Work

In this paper, we proposed an incremental approach for mechanically synthesizing abstract BATs from concrete ones given a refinement mapping. Several promising directions remain to be explored, both in terms of practical development and theoretical extensions. We briefly mention some of them below.

7.1. SMT-based Tool for Verifying and Synthesizing Abstract Theories

A natural next step is the development of a tool that automates the verification and synthesis procedures described in this paper. Such a tool would take as input a LL BAT together with a partially specified abstraction, and would assist the engineer by checking the required conditions and synthesizing missing components of the abstract theory. Leveraging SMT solvers, the tool could encode the regression-based conditions for soundness and completeness as FO constraints over the initial situation, enabling automated checking of equivalences and implications between formulas. The solver could also provide diagnostic feedback when conditions fail, helping the engineer refine the abstraction iteratively.

7.2. Handling Iteration

As mentioned, our approach focuses on programs that do not involve iteration. Extending it to handle iterative constructs raises non-trivial challenges. In particular, regression through loops may fail to terminate or may require reasoning about loop invariants, which are not directly available in our current synthesis framework. This breaks the reduction to formulas over the initial situation that underlies our method. Identifying suitable restrictions or invariant-based techniques that enable effective synthesis in the presence of iteration would significantly broaden the range of programs that can be handled.

7.3. Abstraction of Nondeterministic Programs

The approach presented in this paper focuses on deterministic implementations of HL actions. Extending it to nondeterministic Golog programs introduces additional challenges, as different executions of a nondeterministic program may lead to distinct LL states. For a sound abstraction, these outcomes must be indistinguishable at the abstract level, i.e., they must satisfy the same HL fluents under the refinement mapping. Ensuring this property generally requires additional constraints on both the refinement mapping and the structure of the programs. In particular, one must either restrict nondeterministic choices so that all possible executions are abstractly equivalent, or refine the abstraction by introducing additional HL fluents that capture the relevant distinctions between outcomes.

7.4. Synthesizing Nondeterministic Abstraction

In addition to handling nondeterminism at the LL, it is natural to consider abstractions in which HL actions themselves are nondeterministic. In this case, a single abstract action captures a set of possible concrete executions. A key issue that arises in this setting is the choice of abstraction granularity, i.e., how much information about the LL evolution should be exposed at the HL. In the nondeterministic situation calculus [22], this corresponds to deciding what is modeled as an environment reaction. As illustrated in generalized planning abstractions [23], environment reactions can encode different observational features of the underlying domain. This suggests that abstraction synthesis is not only about constructing mappings, but also about identifying an appropriate information interface between levels. In particular, one must select a set of abstract outcomes that is sufficient to distinguish behaviors relevant to the goal, while ignoring irrelevant details. Developing principled criteria or automated methods for choosing this level of granularity (possibly guided by the structure of the goals, trace constraints, or the class of instances considered in generalized planning) remains an open challenge.

Being able to generate nondeterministic HL abstractions would also be interesting for facilitating and speeding up planning and reactive synthesis, in particular when using procedural approaches based on Golog in nondeterministic domains [24, 25], where simplifying the underlying action theory by abstracting away irrelevant details can directly improve the efficiency of strategy synthesis.

7.5. LLM-based Abstraction Synthesis

Another interesting direction is the use of large language models (LLMs) to assist in the synthesis of abstractions. Recent work [26] has shown that LLMs can generate abstract planning domains from concrete specifications and natural language descriptions, producing candidate abstractions that align with a given purpose. In our setting, LLMs could be used to propose candidate refinement mappings, HL fluents, or action implementations directly from a description of the domain. These candidates could then be validated using the approach developed in this paper, combining the flexibility of data-driven models with the guarantees provided by logical verification. This suggests a hybrid pipeline where LLMs perform heuristic abstraction generation, while logical reasoning ensures soundness and completeness. Exploring this integration, and evaluating its effectiveness in practice, is a promising step toward a principled neuro-symbolic framework for abstraction synthesis.

Acknowledgments

This work has been supported by the PNRR MUR project FAIR (No. PE0000013), the Italian National Ph.D. on Artificial Intelligence at Sapienza University of Rome, the National Science and Engineering Research Council of Canada, and York University.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT and Claude in order to: Grammar and spelling check. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] D. Chen, P. Bercher, Fully observable nondeterministic HTN planning - formalisation and complexity results, in: Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, AAAI Press, 2021, pp. 74–84.

[2] B. Seegebarth, F. Müller, B. Schattenberg, S. Biundo, Making hybrid plans more clear to human users - A formal approach for generating sound explanations, in: Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, AAAI, 2012, pp. 225–233.

[3] R. S. Sutton, D. Precup, S. Singh, Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artificial Intelligence 112 (1999) 181–211. doi:10.1016/S0004-3702(99)00052-1.

[4] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstraction in situation calculus action theories, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI Press, 2017, pp. 1048–1055.

[5] B. Banihashemi, G. De Giacomo, Y. Lespérance, Abstracting situation calculus action theories, Artificial Intelligence 348 (2025) 104407. URL: https://doi.org/10.1016/j.artint.2025.104407. doi:10.1016/J.ARTINT.2025.104407.

[6] J. McCarthy, P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, Machine Intelligence 4 (1969) 463–502.

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

[8] G. De Giacomo, Y. Lespérance, H. J. Levesque, ConGolog, a concurrent programming language based on the situation calculus, Artificial Intelligence 121 (2000) 109–169. doi:10.1016/S0004-3702(00)00031-X.

[9] K. Luo, Y. Liu, Y. Lespérance, Z. Lin, Agent abstraction via forgetting in the situation calculus, in: ECAI 2020 - 24th European Conference on Artificial Intelligence, volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 809–816.

[10] 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: AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, AAAI Press, 2025, pp. 14911–14921. doi:10.1609/AAAI.V39I14.33635.

[11] 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. URL: https://doi.org/10.24963/ijcai.2018/649. doi:10.24963/ijcai.2018/649.

[12] 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. URL: https://doi.org/10.24963/ijcai.2021/253. doi:10.24963/ijcai.2021/253.

[13] Z. Cui, W. Kuang, Y. Liu, Automatic verification for soundness of bounded QNP abstractions for generalized planning, in: IJCAI, ijcai.org, 2023, pp. 3149–3157.

[14] H. Dong, Z. Shi, H. Zeng, Y. Liu, An automatic sound and complete abstraction method for generalized planning with baggable types, in: AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, AAAI Press, 2025, pp. 14875–14884. doi:10.1609/AAAI.V39I14.33631.

[15] F. Pirri, R. Reiter, Some contributions to the metatheory of the situation calculus, Journal of the ACM (JACM) 46 (1999) 325–361.

[16] J. Claßen, G. Lakemeyer, A logic for non-terminating Golog programs, in: KR, 2008, pp. 589–599.

[17] G. De Giacomo, Y. Lespérance, A. R. Pearce, Situation calculus-based programs for representing and reasoning about game structures, Proc. of KR (2010) 445–455.

[18] G. De Giacomo, Y. Lespérance, C. J. Muise, On supervising agents in situation-determined ConGolog, in: International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, IFAAMAS, 2012, pp. 1031–1038.

[19] N. Li, Y. Liu, Automatic verification of partial correctness of Golog programs., in: IJCAI, 2015, pp. 3113–3119.

[20] R. Milner, An algebraic definition of simulation between programs, in: Proceedings of the 2nd International Joint Conference on Artificial Intelligence, William Kaufmann, 1971, pp. 481–489.

[21] R. Milner, Communication and concurrency, PHI Series in computer science, Prentice Hall, 1989.

[22] 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, 2021, pp. 216–226. URL: https://doi.org/10.24963/kr.2021/21. doi:10.24963/kr.2021/21.

[23] G. De Giacomo, Y. Lespérance, M. Mancanelli, Situation calculus temporally lifted abstractions for generalized planning, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 14848–14857.

[24] G. De Giacomo, Y. Lespérance, M. Mancanelli, Strategic reasoning over Golog programs in the nondeterministic situation calculus, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2026.

[25] G. De Giacomo, Y. Lespérance, M. Mancanelli, G. Parretti, Reactive synthesis for Golog specifications in the propositional situation calculus, in: Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning, 2026. (to appear).

[26] B. Banihashemi, M. Patel, Y. Lespérance, Using large language models for abstraction of planning domains-extended version, arXiv preprint arXiv:2510.20258 (2025).