Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)

Abstraction of Nondeterministic Situation Calculus Action Theories

Bita Banihashemi[1] , **Giuseppe De Giacomo**[2] , **Yves Lespérance**[3]

[1]Ronin Institute
[2]University of Oxford
[3]York University
bita@ronininstitute.org, giuseppe.degiacomo@cs.ox.ac.uk, lesperan@eecs.yorku.ca

Abstract

We develop a general framework for abstracting the behavior of an agent that operates in a nondeterministic domain, i.e., where the agent does not control the outcome of the nondeterministic actions, based on the nondeterministic situation calculus and the ConGolog programming language. We assume that we have both an abstract and a concrete nondeterministic basic action theory, and a refinement mapping which specifies how abstract actions, decomposed into agent actions and environment reactions, are implemented by concrete ConGolog programs. This new setting supports strategic reasoning and strategy synthesis, by allowing us to quantify separately on agent actions and environment reactions. We show 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, 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.

1 Introduction

When working in realistic dynamic domains, the use of *abstraction* has proven essential in many areas of artificial intelligence, for example, in improving the efficiency of planning (e.g., [Chen and Bercher, 2021]), explaining agents' behavior (e.g., [Seegebarth *et al.*, 2012]), and in reinforcement learning (e.g., [Sutton *et al.*, 1999]).

Recently, [Banihashemi *et al.*, 2017] (BDL17) proposed a formal account of *agent abstraction* based on the situation calculus [McCarthy and Hayes, 1969; Reiter, 2001] and the ConGolog agent programming language [De Giacomo *et al.*, 2000]. They assume that one has a high-level/abstract action theory, a low-level/concrete action theory, both representing the agent's behavior at different levels of detail, and a *refinement mapping* between the two. The 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 state formula. This work defines notions of abstractions between such action theories in terms of the existence of a suitable bisimulation relation [Milner, 1971] between their respective models. Abstractions have many useful properties that ensure that one 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. The framework can also be used to generate high-level explanations of low-level behavior.

This framework was formulated assuming a deterministic environment, as usual in the situation calculus. Hence, the only nondeterminism was coming from ConGolog programs and was *angelic* in nature, i.e., under the control of the agent [Levesque *et al.*, 1997; De Giacomo *et al.*, 2000].

However, many agents operate in *nondeterministic environments* where the agent does not fully control the outcome of her actions (e.g., flipping a coin where the outcome may be heads or tails). Recently, [De Giacomo and Lespérance, 2021] (DL21) proposed a simple and elegant situation calculus account of nondeterministic environments where they clearly distinguish between the nondeterminism associated with agent choices and that associated with environment choices, the first being angelic for the agent, and the second being devilish, i.e., not under the agent's control.[1] The presence of environment nondeterminism deeply influences reasoning about action: if the agent wants to complete a task, she cannot simply do standard logical reasoning (i.e., satisfiability/logical implication), but needs to do *realizability/synthesis*, i.e., devise a strategy that, in spite of the uncontrollable reactions of the environment, guarantees successful completion of the task [Church, 1963; Pnueli and Rosner, 1989; Abadi *et al.*, 1989]. We see this, for example in planning, where in the classical deterministic setting the problem can be solved by heuristic search, while in fully observable nondeterministic (FOND) domains forms of adversarial (AND/OR) search are required [Cimatti *et al.*, 1998; Haslum *et al.*, 2019].

In this paper, we develop an account of agent abstraction in presence of nondeterministic environments based on (DL21). The refinement mapping between high-level fluents and low-level state formulas remains as in (BDL17). Instead we consider each high-level action, now nondeterministic, as being

[1]Earlier accounts dealing with the topic in the situation calculus were more complex and did not clearly distinguish between these two forms of nondeterminism, e.g., [Pinto *et al.*, 2000] which deals with stochastic actions, and [Bacchus *et al.*, 1999] which deals with uncertainty, noisy acting, and noisy sensing.

composed of an *agent action* and an (implicit) *environment reaction*. As a consequence, we map the agent action (without the environment reaction) into a low-level *agent program* that appropriately reflects the nondeterminism of the environment, and the complete high-level action, including both the agent action and the environment reaction, into a low-level *system program* that relates the high-level environment reaction to the low-level ones.

We show that the (BDL17) notion of m-bisimulation extends naturally to this new setting. This allows for exploiting abstraction when reasoning about executions as in (BDL17), in spite of the nondeterministic environment. But, notably, this new setting now supports strategic reasoning and strategy synthesis, by allowing us to quantify separately on agent actions and environment reactions. As a result, we can exploit abstraction by finding abstract strategies at the high-level and then refine them into concrete low-level ones. In particular, we show that if the agent has a (strong FOND) plan/strategy to achieve a goal/complete a task at the high level (i.e., no matter what the environment does), and she can always execute the nondeterministic high-level actions to completion at the low level (even if not controlling their outcome), then there exists a refinement of it that is a (strong FOND) plan/strategy to achieve the refinement of the goal/task at the low level.

2 Preliminaries

Situation calculus. The *situation calculus* is a well known predicate logic language for representing and reasoning about dynamically changing worlds [McCarthy and Hayes, 1969; Reiter, 2001]. 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 (e.g., $Open(Door1, s)$). The abbreviation $do([a_1, \ldots, a_n], s)$ stands for $do(a_n, do(a_{n-1}, \ldots, do(a_1, s) \ldots))$; for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$. In this language, a dynamic domain can be represented by a *basic action theory (BAT)*, where successor state axioms represent the causal laws of the domain and provide a solution to the frame problem [Reiter, 2001]. A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; the precondition axioms characterize this predicate. Abbreviation $Executable(s)$ means that every action performed in reaching situation s was possible in the situation in which it occurred.

Nondeterministic situation calculus. A major limitation of the standard situation calculus and BATs is that atomic actions are deterministic. [De Giacomo and Lespérance, 2021] (DL21) propose a simple extension of the framework to handle nondeterministic actions while preserving the solution to the frame problem. For any primitive action by the agent in a nondeterministic domain, there can be a number of different outcomes. (DL21) takes the outcome as being determined by the agent's action and the environment's reaction to this action. This is modeled by having every action type/function $A(\vec{x}, e)$ take an additional environment reaction parameter e, ranging over a new sort *Reaction* of environment reactions. The agent cannot control the environment reaction, so it performs the reaction-suppressed version of the action $A(\vec{x})$ and the environment then selects a reaction e to produce the complete action $A(\vec{x}, e)$. We call the reaction-suppressed version of the action $A(\vec{x})$ an *agent action* and the full version of the action $A(\vec{x}, e)$ a *system action*.

Nondeterministic basic action theories (NDBATs). These can be seen as a special kind of BAT, where every action function takes an environment reaction parameter, and moreover, for each agent action $A(\vec{x})$, we

- have its agent action precondition denoted by: $Poss_{ag}(A(\vec{x}), s) \doteq \phi_A^{agPoss}(\vec{x}, s)$;
- have a reaction independence requirement, stating that the precondition for the agent action is independent of any environment reaction: $\forall e.Poss(A(\vec{x}, e), s) \supset Poss_{ag}(A(\vec{x}), s)$;
- have a *reaction existence* requirement, stating that if the precondition of the agent action holds then there exists a reaction to it which makes the complete system action executable: $Poss_{ag}(A(\vec{x}), s) \supset \exists e.Poss(A(\vec{x}, e), s)$.

The above requirements *must* be entailed by the action theory for it to be an NDBAT.

A NDBAT $\mathcal{D}$ is the union of the following disjoint sets: foundational, domain independent, axioms of the situation calculus (Σ) as in standard BATs, axioms describing the initial situation D_{S_0} as in standard BATs, unique name axioms for actions ($\mathcal{D}_{una}$) as in standard BATs, successor state axioms (SSAs) describing how fluents change after *system* actions are performed ($\mathcal{D}_{ssa}$), and *system* action precondition axioms, one for each action type, stating when the complete system action can occur ($\mathcal{D}_{poss}$); these are of the form: $Poss(A(\vec{x}, e), s) \equiv \phi_A^{Poss}(\vec{x}, e, s)$.

Example. Our running example is based on a triangle tireworld domain (see Fig. 1). The agent's goal is to drive from location 11 to location 13. When driving from one location to the next, the possibility of a tire going flat exists. If there is a spare tire in the location where the car is at (indicated by circles in Fig. 1), the agent can use it to fix a flat.

NDBAT $\mathcal{D}_l^{tt}$ models the domain at the low level. The system action $drive(o, d, r)$ can be performed by the agent to drive from origin location o to destination location d, and r indicates environment's reaction. This action is executable

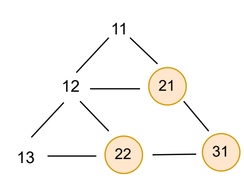

Figure 1: A 3-Level Triangle Tireworld

when the agent action $drive(o, d)$ is executable (i.e., when the agent is at o, a road connects o to d, and agent does not have a flat tire), and r can take on two values: $FlatTire$ if the tire goes flat and $NoFlatTire$ otherwise. The fluent $At_{LL}(l, s)$ indicates agent's location, $Road_{LL}(o, d, s)$ specifies the road connections in Fig. 1, $Flat_{LL}(s)$ describes whether there is a flat tire, and $Visited_{LL}(l, s)$ indicates if the location l has already been visited by the agent.

The system action $fixFlatTire(l, r)$ fixes a flat tire, and the environment reaction r is $Success_{LF}$ (assumed for simplicity). It is executable if the fluent $Spare_{LL}(l, s)$ holds, i.e., there is a spare tire. The action $wait_{LL}$ can be performed by the agent to remain idle at a location.

$\mathcal{D}_l^{tt}$ includes the following action precondition axioms (throughout the paper, we assume that free variables are universally quantified from the outside):

$$
\begin{aligned}
&Poss_{ag}(drive(o, d), s) \doteq \\
&\quad o \neq d \wedge At_{LL}(o, s) \wedge Road_{LL}(o, d, s) \wedge \neg Flat_{LL}(s) \\
&Poss_{ag}(fixFlatTire(l), s) \doteq \\
&\quad At_{LL}(l, s) \wedge Spare_{LL}(l, s) \wedge Flat_{LL}(s) \\
&Poss_{ag}(wait_{LL}, s) \doteq TRUE \\
&Poss(drive(o, d, r), s) \equiv \\
&\quad Poss_{ag}(drive(o, d), s) \wedge (r = FlatTire \vee r = NoFlatTire) \\
&Poss(fixFlatTire(l, r), s) \equiv \\
&\quad Poss_{ag}(fixFlatTire(l), s) \wedge r = Success_{LF} \\
&Poss(wait_{LL}(r), s) \equiv \\
&\quad Poss_{ag}(wait_{LL}, s) \wedge r = Success_{LW}
\end{aligned}
$$

$\mathcal{D}_l^{tt}$ also includes the following SSAs:

$$
\begin{aligned}
&At_{LL}(l, do(a, s)) \equiv \exists o, r.a = drive(o, l, r) \vee \\
&\quad At_{LL}(l, s) \ \wedge \forall d, r.a \neq drive(l, d, r) \\
&Flat_{LL}(do(a, s)) \equiv \\
&\quad \exists o, d.a = drive(o, d, FlatTire) \vee \\
&\quad Flat_{LL}(s) \wedge \forall r, l.a \neq fixFlatTire(l, r) \\
&Visited_{LL}(l, do(a, s)) \equiv \\
&\quad \exists o, r.a = drive(o, l, r) \ \vee Visited_{LL}(l, s)
\end{aligned}
$$

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

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

$$
\begin{aligned}
&Road_{LL}(o, d, S_0) \equiv (o, d) \in \{(11, 12), (11, 21), (12, 21), \\
&(12, 22), (12, 13), (13, 22), (22, 31), (21, 31), (12, 11), (21, 11), \\
&(21, 12), (22, 12), (13, 12), (22, 13), (31, 22), (31, 21)\}, \\
&Spare_{LL}(l, S_0) \equiv l \in \{21, 22, 31\}, Visited_{LL}(l, S_0) \equiv l = 11, \\
&At_{LL}(l, S_0) \equiv l = 11, Dest_{LL}(l, S_0) \equiv l = 13, \neg Flat_{LL}(S_0).
\end{aligned}
$$

FOND planning and synthesis in NDBATs. We start with some definitions. A *weak plan* is one that achieves the goal when the environment "cooperates" and selects environment reactions to nondeterministic actions that make this happen. Formally, we say that a sequence of agent actions $\vec{a}$ is a *weak plan* to achieve $Goal$ if $\exists s'.Do_{ag}(\vec{a}, S_0, s') \wedge Goal(s')$ holds, i.e., there exists an execution of $\vec{a}$ that takes us from the initial situation S_0 to situation s' where the goal holds. $Do_{ag}(\vec{a}, s, s')$ means that the system may reach situation s' when the agent executes the sequence of agent actions $\vec{a}$ depending on environment reactions:

$Do_{ag}(\epsilon, s, s') \doteq s' = s$ (where ϵ is the empty sequence of actions)

$$
\begin{aligned}
&Do_{ag}([A(\vec{x}), \sigma], s, s') \doteq \\
&\quad \exists e.Poss(A(\vec{x}, e), s) \wedge Do_{ag}(\sigma, do(A(\vec{x}, e), s), s')
\end{aligned}
$$

A *strong plan* for the agent guarantees the achievement of a goal no matter how the environment reacts; it is a strategy for the agent to follow to ensure that the goal is achieved. (DL21) define a strategy as a function from situations to (instantiated) agent actions. That is, $f(s) = A(\vec{t})$ denotes that the strategy f applied to situation s returns $A(\vec{t})$ as the next action to do. The special agent action $stop$ (with no effects and preconditions) may be returned when the strategy wishes to stop. Given a strategy, we can check whether it forces the goal to become true in spite of the environment reactions, i.e., is a strong plan to achieve the goal. Formally, we have $AgtCanForceBy(Goal, s, f)$, i.e., the agent can force $Goal$ to become true by following strategy f in s:

$AgtCanForceBy(Goal, s, f) \doteq \forall P.[\ldots \supset P(s)]$
where ... stands for

$$
\begin{aligned}
&[(f(s) = stop \wedge Goal(s)) \supset P(s)] \wedge \\
&[\exists A.\exists \vec{t}.(f(s) = A(\vec{t}) \neq stop \wedge Poss_{ag}(A(\vec{t}), s) \wedge \\
&\quad \forall e.(Poss(A(\vec{t}, e), s) \supset P(do(A(\vec{t}, e), s)))) \\
&\supset P(s)]
\end{aligned}
$$

We say that $AgtCanForce(Goal, s)$ holds iff there is a strategy f such that $AgtCanForceBy(Goal, s, f)$ holds.

Example cont. Strategies use the $stop$ action, hence we define a $stop_{LL}$ action at the low level; this action always terminates with the $Success_{LS}$ reaction. The strategy that guarantees reaching location 13 is defined as follows:

$\mathcal{D}_l^{tt} \models AgtCanForceBy(At_{LL}(13), S_0, f_l)$
where

$$
f_l(s) \doteq \begin{cases}
stop_{LL} & \text{if } At_{LL}(13, s) \\
fixFlatTire(l) & \text{if } At_{LL}(l, s) \wedge l \neq 13 \wedge Flat_{LL}(s) \\
drive(o, d) & \text{if } At_{LL}(o, s) \wedge o \neq 13 \wedge \neg Flat_{LL}(s) \\
 & \wedge\, Spare_{LL}(d, s) \wedge Road_{LL}(o, d, s) \\
 & \wedge \neg Visited_{LL}(d) \\
wait_{LL} & \text{otherwise}
\end{cases}
$$

That is, the agent should stop if she is already at location 13, otherwise she drives to a location she has not visited before and that has a spare tire; in case of a flat tire, action $fixFlatTire$ is executed; in all other cases the agent waits.

High-level programs and ConGolog. To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called *high-level programming languages* have been defined. Here we concentrate on (a variant of) ConGolog [De Giacomo *et al.*, 2000] 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. As usual, we denote by $\varphi[s]$ the formula obtained from φ by restoring the situation argument s into all fluents in φ. 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 (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. 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 represents 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: *(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 [De Giacomo *et al.*, 2010]; differently from [De Giacomo *et al.*, 2000], the test construct $\varphi?$ does not yield any transition, but is final when satisfied.

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*. We use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

For simplicity in this paper, we use a restricted class of ConGolog programs which are *situation-determined* (SD) [De Giacomo *et al.*, 2012], i.e., for every sequence of actions, the remaining program is uniquely determined by the resulting situation:

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

ConGolog program execution in ND domains. To specify agent behaviors in nondeterministic domains, (DL21) use ConGolog *agent programs*, which are composed as usual, but only involve agent (reaction-suppressed) atomic actions. The semantics for such agent programs is a variant of the one above where a transition for an atomic agent action may occur whenever there exists a reaction that can produce it.

Strategic reasoning in executing programs in ND domains. To represent the ability of the agent to execute an agent program in a ND domain (DL21) introduce $AgtCanForceBy(\delta, s, f)$ as an adversarial version of Do in presence of environment reactions. This predicate states that strategy f, a function from situations to agent actions (including the special action $stop$), executes SD ConGolog agent program δ in situation s considering its nondeterminism angelic, as in the standard Do, but also considering the nondeterminism of environment reactions devilish/adversarial:

$$AgtCanForceBy(\delta, s, f) \doteq \forall P.[\ldots \supset P(\delta, s)]$$
where $\ldots$ stands for
$$[(f(s) = stop \wedge Final(\delta, s)) \supset P(\delta, s)] \wedge$$
$$[\exists A.\exists\vec{t}.(f(s) = A(\vec{t}) \neq stop \wedge$$
$$\exists e.\exists\delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge$$
$$\forall e.(\exists\delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s))) \supset$$
$$\exists\delta'.Trans(\delta, s, \delta', do(A(\vec{t}, e), s)) \wedge P(\delta', do(A(\vec{t}, e), s))$$
$$\supset P(\delta, s)]$$

We say that predicate $AgtCanForce(\delta, s)$ holds iff there exists a strategy f s.t. $AgtCanForceBy(\delta, s, f)$ holds.

Example cont. Suppose we have a program $\delta_{go}^{LL}(l)$ which lets the agent drive to any location, or fix a flat any number of times until a destination l is reached: $\delta_{go}^{LL}(l) = (\pi o, d.drive(o, d) \mid \pi d.fixFlatTire(d))^*; At_{LL}(l)?$

Now suppose the agent has been assigned the task of going to location 13. We can show that she has a strategy for executing the task $\delta_{go}^{LL}(13)$:

$$\mathcal{D}_l^{tt} \models AgtCanForceBy(\delta_{go}^{LL}(13), S_0, f)$$

Here, the strategy f to execute the task is same as f_l above.

3 Abstraction in Nondeterministic Domains

In this section, we show how we can extend the agent abstraction framework of (BDL17) to handle nondeterministic domains. As in (BDL17), we assume that there is a high-level/abstract (HL) action theory $\mathcal{D}_h$ and a low-level/concrete (LL) action theory $\mathcal{D}_l$ representing the agent's possible behaviors at different levels of detail. In (BDL17), these are standard BATs; here, we assume that they are both NDBATs. $\mathcal{D}_h$ (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 sort *Object* are assumed be a countably infinite set $\mathcal{N}$ of standard names for which we have the unique name assumption and domain closure. We also assume that *Reaction* is a sub-sort of *Object*. Also, $\mathcal{D}_h$ and $\mathcal{D}_l$ are assumed to share no domain specific symbols except for the set of standard names for objects $\mathcal{N}$. For simplicity and w.l.o.g., it is assumed that there are no functions other than constants and no non-fluent predicates.

Refinement mapping. As in (BDL17), we assume that one relates the HL and LL theories by defining a *refinement mapping* that specifies how HL atomic actions are implemented at the low level and how HL fluents can be translated into LL state formulas. In deterministic domains, one can simply map HL atomic actions to a LL program that the agent uses to implement the action. In nondeterministic domains however, we additionally need to specify what HL reaction the environment performs in each LL refinement of the HL action.

Definition 1 (NDBAT Refinement Mapping)**.** *A* NDBAT refinement mapping *m is a triple $\langle m_a, m_s, m_f\rangle$ where m_a associates each HL primitive action type A in $\mathcal{A}_h$ to a SD ConGolog* agent program *δ_A^{ag} defined over the LL theory that implements the agent (i.e., reaction-suppressed) action, i.e., $m_a(A(\vec{x})) = \delta_A^{ag}(\vec{x})$, m_s associates each A to a SD ConGolog* system program *δ_A^{sys} defined over the LL theory that implements the system action, i.e., $m_s(A(\vec{x}, e)) = \delta_A^{sys}(\vec{x}, e)$, thus specifying when the HL reaction occurs (system programs are interpreted using the standard transition semantics), and (as in (BDL17)) m_f 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 in a situation, i.e., $m_f(\vec{x}) = \phi_F(\vec{x})$.*

We can extend a mapping to a sequence of agent 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 $m_a(\epsilon) \doteq nil$, and similarly for sequences of system actions. Note that $m_a(\vec{\alpha})$ is well defined even if the action parameters are not ground; but the action functions must be given. We also extend the notation so that $m_f(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula ϕ by $m_f(F(\vec{x}))$.

Agent actions and system actions must be mapped in a consistent way. To ensure this, we require the following:

Constraint 2. *(Proper Refinement Mapping)*
For every high-level system action sequence $\vec{\alpha}$ and every high-level action A, we have that:

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_s(\vec{\alpha}), S_0, s) \supset \\ \forall \vec{x}, s'.(Do_{ag}(m_a(A(\vec{x})), s, s') \equiv \exists e.Do(m_s(A(\vec{x}, e)), s, s')))$$

This ensures that (1) for every situation s' that can be reached from s by executing a refinement of the HL agent action $A(\vec{x})$, there is a HL reaction e that generates it, and (2) that for every situation s' that can be reached from s by executing a refinement of the HL system action $A(\vec{x}, e)$, there is a refinement of the HL agent action that generates it (in the above, s is any situation reached by executing a sequence of HL system actions). Note that (1) essentially states that the reaction existence requirement holds at the low level for refinements of HL actions and (2) amounts to having the reaction independence requirement hold. We say that an NDBAT refinement mapping m is *proper wrt low-level NDBAT $\mathcal{D}_l$* if this constraint holds.

Example cont. NDBAT $\mathcal{D}_h^{tt}$ abstracts over driving and fixing potential flat tires. Action $driveAndTryFix(o, d, r)$ can be performed to drive from origin o to destination d, and fix a flat if one occurs and a spare is available. This action is executable when $driveAndTryFix(o, d)$ is executable, i.e., when the agent is at o, a road connects o to d, and agent does not have a flat tire; the environment reaction r may take on the following values: $DrvNoFlat$, if the tire does not go flat, $DrvFlat$ if the tire goes flat and it is not possible to fix it, or $DrvFlatFix$ if the tire went flat and the flat was fixed. Action $wait_{HL}$ lets the agent remain idle at a location. $\mathcal{D}_h^{tt}$ includes the following system and agent action precondition axioms:

$$\begin{aligned}
&Poss_{ag}(driveAndTryFix(o, d), s) \doteq \\
&\quad o \neq d \wedge At_{HL}(o, s) \wedge Road_{HL}(o, d, s) \wedge \neg Flat_{HL}(s) \\
&Poss_{ag}(wait_{HL}, s) \doteq TRUE \\
&Poss(driveAndTryFix(o, d, r), s) \equiv \\
&\quad Poss_{ag}(driveAndTryFix(o, d), s) \wedge \\
&\quad (r = DrvNoFlat \vee \\
&\quad\ \neg Spare_{HL}(d, s) \wedge r = DrvFlat \vee \\
&\quad\ Spare_{HL}(d, s) \wedge r = DrvFlatFix) \\
&Poss(wait_{HL}(r), s) \equiv \\
&\quad Poss_{ag}(wait_{HL}, s) \wedge r = Success_{HW}
\end{aligned}$$

$\mathcal{D}_h^{tt}$ also includes the following SSAs:

$$\begin{aligned}
&At_{HL}(l, do(a, s)) \equiv \exists o, r.a = driveAndTryFix(o, l, r) \vee \\
&\quad At_{HL}(l, s) \wedge \forall d, r.a \neq driveAndTryFix(l, d, r) \\
&Flat_{HL}(do(a, s)) \equiv \\
&\quad \exists o, d.a = driveAndTryFix(o, d, DrvFlat) \vee Flat_{HL}(s) \\
&Visited_{HL}(l, do(a, s)) \equiv \\
&\quad \exists o, r.a = driveAndTryFix(o, l, r) \vee Visited_{HL}(l, s)
\end{aligned}$$

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

The initial state axioms of $\mathcal{D}_h^{tt}$ are same as $\mathcal{D}_l^{tt}$, with high-level fluents having a distinct name (with HL suffix); e.g., $Road_{HL}$ is axiomatized exactly as $Road_{LL}$.

We specify the relationship between the HL and LL NDBATs through the following refinement mapping m^{tt}:

$$\begin{aligned}
&m_a^{tt}(driveAndTryFix(o, d)) = \\
&\quad drive(o, d); \\
&\quad \textbf{if } \neg Flat_{LL} \textbf{ then } nil \textbf{ else} \\
&\qquad \textbf{if} \neg Spare_{LL}(d) \textbf{ then } nil \textbf{ else } fixFlatTire(d) \textbf{ endIf} \\
&\quad \textbf{endIf} \\
&m_a^{tt}(wait_{HL}) = wait_{LL} \\
&m_s^{tt}(driveAndTryFix(o, d, r_h)) = \\
&\quad \pi r_l.drive(o, d, r_l); \\
&\quad \textbf{if } \neg Flat_{LL} \textbf{ then } r_h = DrvNoFlat? \textbf{ else} \\
&\qquad \textbf{if} \neg Spare_{LL}(d) \textbf{ then } r_h = DrvFlat? \\
&\qquad \textbf{else } fixFlatTire(d, Success_{LF}); r_h = DrvFlatFix? \\
&\qquad \textbf{endIf} \\
&\quad \textbf{endIf} \\
&m_s^{tt}(wait_{HL}(r_h)) = \pi r_l.wait_{LL}(r_l); r_h = Success_{HW}? \\
&m_f^{tt}(Flat_{HL}) = Flat_{LL}
\end{aligned}$$

Thus, the HL agent action $driveAndTryFix(o, d)$ is implemented by an LL program where the agent first performs $drive(o, d)$, and depending on whether the tire has gone flat and a spare exists at location d, fixes the tire or does nothing. Fluents $At_{HL}(l)$, $Spare_{HL}(l)$, $Road_{HL}(o, d)$, $Dest_{HL}(l)$, and $Visited_{HL}(l)$ are mapped to their low-level counterparts similar to $Flat_{HL}$. We can show that:[2]

Proposition 3. *NDBAT refinement mapping m^{tt} is* proper *wrt $\mathcal{D}_l^{tt}$.*

m-Bisimulation. To relate the HL and LL models/theories, (BDL17) define a variant of bisimulation [Milner, 1971; Milner, 1989].[3] The base condition for the bisimulation is:

Definition 4 (m-isomorphic situations)**.** *Let M_h be model of the HL theory $\mathcal{D}_h$, and M_l a model of the LL theory $\mathcal{D}_l \cup \mathcal{C}$. We say that situation s_h in M_h is* m-isomorphic *to situation s_l in M_l, written $s_h \sim_m^{M_h, M_l} s_l$, if and only if*

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

for every high-level primitive fluent $F(\vec{x})$ in $\mathcal{F}_h$ and every variable assignment v.

If $s_h \sim_m^{M_h, M_l} s_l$, then s_h and s_l evaluate all HL fluents the same.

Definition 5 (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 between M_h and M_l *if $\langle s_h, s_l \rangle \in B$ implies that: (i) $s_h \sim_m^{M_h, M_l} s_l$; (ii) for every HL 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 $\langle s'_h, s'_l \rangle \in B$; and (iii) for every HL 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*

[2] For proofs, see [Banihashemi *et al.*, 2023]

[3] As usual, $M, v \models \phi$ means that model M and assignment v satisfy formula ϕ (where ϕ may contain free variables that are interpreted by v); also $v[x/e]$ represents the variable assignment that is just like v but assigns variable x to entity e.

*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 $\langle s'_h, s'_l \rangle \in B$. M_h is m-*bisimilar *to M_l, 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 $(S_0^{M_h}, S_0^{M_l}) \in B$.*

The definition of m-bisimulation can remain as in (BDL17) where conditions (ii) and (iii) are applied to high-level *system* primitive actions and their mapping m_s.

The definition of m-bisimulation ensures that performing a HL system action results in m-bisimilar situations. We can show that with the restriction to proper mappings, this automatically carries over to HL agent actions (so there is no need to change the definition of m-bisimulation to get this):

Theorem 6. *Suppose that $M_h \sim_m M_l$, where $M_h \models \mathcal{D}_h$, $M_l \models \mathcal{D}_l \cup \mathcal{C}$, $\mathcal{D}_h$ and $\mathcal{D}_l$ are both NDBATs, and m is proper wrt D_l. Then for any HL system action sequence $\vec{\alpha}$ and any HL primitive action A, we have that:*

1. *if there exist s_h and s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Do(\vec{\alpha}, S_0, s) \wedge Do_{ag}(A(\vec{x}), s, s')$, then there exist s_l and s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m_s(\vec{\alpha}), S_0, s) \wedge Do_{ag}(m_a(A(\vec{x})), s, s')$, $s_h \sim_m^{M_h, M_l} s_l$, and $s'_h \sim_m^{M_h, M_l} s'_l$;*
2. *if there exist s_l and s'_l such that $M_l, v[s/s_l, s'/s'_l] \models Do(m_s(\vec{\alpha}), S_0, s) \wedge Do_{ag}(m_a(A(\vec{x})), s, s')$, then there exist s_h and s'_h such that $M_h, v[s/s_h, s'/s'_h] \models Do(\vec{\alpha}, S_0, s) \wedge Do_{ag}(A(\vec{x}), s, s')$, $s_h \sim_m^{M_h, M_l} s_l$, and $s'_h \sim_m^{M_h, M_l} s'_l$.*

(BDL17) use m-bisimulation to define notions of sound/complete abstraction between a high-level action theory and a low-level one: $\mathcal{D}_h$ is a *sound abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_l of $\mathcal{D}_l \cup \mathcal{C}$, there exists a model M_h of $\mathcal{D}_h$ such that $M_h \sim_m M_l$. 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 considers the executability of a refinement of a high-level action is satisfiable, then the high level does also. A dual notion is also defined: $\mathcal{D}_h$ is a *complete abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_h of $\mathcal{D}_h$, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ such that $M_l \sim_m M_h$.

The notion of m-bisimulation provides the semantic underpinning of these notions of abstraction. (BDL17) also prove the following results that identify a set of properties that are necessary and sufficient to have a sound/complete abstraction (here we adjust the notation to use system actions) and can be used to verify that one has a sound/complete abstraction, e.g., by model checking or theorem proving techniques:

Theorem 7 (BDL17)**.** *$\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to mapping m if and only if for any sequence of high-level system actions $\vec{\alpha}$:*

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

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

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

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

Theorem 8 (BDL17)**.** *If $\mathcal{D}^h$ is a sound abstraction of $\mathcal{D}^l$ relative to mapping m, then $\mathcal{D}^h$ is also a complete abstraction of $\mathcal{D}^l$ wrt mapping m if and only if for every model M_h of $\mathcal{D}^h_{S_0} \cup \mathcal{D}^h_{ca} \cup \mathcal{D}^h_{coa}$, there exists a model M_l of $\mathcal{D}^l_{S_0} \cup \mathcal{D}^l_{ca} \cup \mathcal{D}^l_{coa}$ such that $S_0^{M_h} \sim_m^{M_h, M_l} S_0^{M_l}$.*

Using the above results, we can show that:

Proposition 9. *$\mathcal{D}^{tt}_h$ is a sound and complete abstraction of $\mathcal{D}^{tt}_l$ wrt m^{tt}.*

Note that in this paper, we are focusing on fully observable domains, so we will just present results about m-bisimilar NDBAT models; $M_h \sim_m M_l$ essentially means that M_h is a sound and complete abstraction of M_l relative to m.

4 Results about Action Executions

We now show some interesting results about the use of NDBAT abstractions to reason about action executions. Our results will be mostly about m-bisimilar NDBAT models, i.e., where $M_h \sim_m M_l$; unless stated otherwise, we assume we have NDBATs $\mathcal{D}_h$ and $\mathcal{D}_l$, that $M_h \models \mathcal{D}_h \cup \mathcal{C}$ and $M_l \models \mathcal{D}_l \cup \mathcal{C}$, and that m is proper wrt $\mathcal{D}_l$.

Firstly, we have that m-isomorphic situations satisfy the same high-level situation-suppressed formulas:

Lemma 10 (BDL17)**.** *If $s_h \sim_m^{M_h, M_l} s_l$, then for any high-level situation-suppressed formula ϕ, we have that:*

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

Secondly, we can show that in m-bisimilar models, the same sequences of high-level system actions are executable, and that in the resulting situations, the same high-level situation-suppressed formulas hold:

Theorem 11. *If $M_h \sim_m M_l$, then for any sequence of high-level system actions $\vec{\alpha}$ and any high-level situation-suppressed formula ϕ, we have that*

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

Here and below, sequences of high-level system actions $\vec{\alpha}$ may contain free variables in the action parameters, but the action functions must be given, and similarly for sequences of high-level agent actions (this generalizes (BDL17) which only considers ground action sequences).

We can extend this result to sequences of *agent actions* by exploiting the fact that m is a proper mapping:

Theorem 12. *If $M_h \sim_m M_l$, then for any sequence of high-level agent actions $\vec{\alpha}$ and any high-level situation-suppressed formula ϕ, we have that*

$$M_l, v \models \exists s'. Do_{ag}(m_a(\vec{\alpha}), S_0, s') \wedge m_f(\phi)[s'] \quad \text{if and only if} \quad M_h, v \models \exists s'. Do_{ag}(\vec{\alpha}, S_0, s') \wedge \phi[s'].$$

This means that if the agent has a weak plan to achieve a goal at the high level, then there exists a refinement of it that is a weak plan to achieve the mapped goal at the low level.

Example cont. At the high level, a weak plan to achieve the goal of getting to location 13 is to first drive to 12 and then to 13:

$$\mathcal{D}_h^{tt} \models \exists s'.Do_{ag}(\vec{\alpha}, S_0, s') \wedge At_{HL}(13, s') \text{ where}$$
$$\vec{\alpha} = driveAndTryFix(11, 12); driveAndTryFix(12, 13)$$

This plan works provided no flat occurs when driving to 12, as there is no spare there. By Th. 12, there exists a weak plan at the low level that refines this high-level plan:

$$\mathcal{D}_l^{tt} \models \exists s'.Do_{ag}(m_a(\vec{\alpha}), S_0, s') \wedge At_{LL}(13, s') \wedge Do_{ag}([drive(11, 12); drive(12, 13)], S_0, s')$$

5 Results about Strategic Reasoning

Let us now discuss how NDBAT abstractions can be used in FOND domains to synthesize strategies to fulfill reachability/achievement goals as well as temporally extended goals/tasks. First of all, we need to consider how much strategic reasoning the agent needs to do to execute a high-level atomic action at the low level. A given HL agent action $A(\vec{x})$ is mapped to a LL agent program $m_a(A(\vec{x}))$ that implements it. As we have seen, in m-bisimilar models, Constraint 2 ensures that if $A(\vec{x})$ is executable at the HL, then there exists a terminating execution of $m_a(A(\vec{x}))$ at the LL (this holds for HL system actions as well). But this does not mean that all executions of $m_a(A(\vec{x}))$ terminate, as some may block or diverge, due to either agent or environment choices. In general, the agent must do strategic reasoning to ensure that the execution of $m_a(A(\vec{x}))$ terminates (and the environment may need to cooperate as well). But we can impose further constraints on the mapping of HL actions to avoid this or ensure that a strategy exists. Note that ensuring that the execution of $m_a(A(\vec{x}))$ terminates does not mean that the agent controls the action's outcome (e.g., not having a flat); the outcome is still determined by the environment reactions.

Constraints on HL action implementation. First, we may want to require that the mapping of HL actions is such that the implementation program always terminates and no LL strategic reasoning is required to ensure termination. To do this, we first define:

$$InevTerminates(\delta, s) \doteq \forall P.[\ldots \supset P(\delta, s)]$$
where $\ldots$ stands for
$$[(Final(\delta, s)) \supset P(\delta, s)] \wedge$$
$$[\exists s'.\exists \delta'.Trans(\delta, s, \delta', s') \wedge$$
$$\forall s'.\forall \delta'.(Trans(\delta, s, \delta', s') \supset P(\delta', s'))$$
$$\supset P(\delta, s)]$$

$InevTerminates(\delta, s)$ means that program δ executed starting in situation *s inevitably terminates* , i.e., all its executions eventually terminate.[4]

Then we can ensure that for any HL agent action that is possibly executable at the low level, all executions terminate (i.e., they never block or diverge) by requiring the following:

[4] This is analogous to the CTL [Clarke and Emerson, 1981] formula $AF\phi$, i.e., on all paths eventually ϕ.

Constraint 13. *(HL actions Inevitably Terminate) For every high-level system action sequence $\vec{\alpha}$ and every high-level action A, we have that:*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_s(\vec{\alpha}), S_0, s) \supset \forall \vec{x}.(\exists s'.Do_{ag}(m_a(A(\vec{x})), s, s') \supset InevTerminates(m_a(A(\vec{x})), s)))$$

Proposition 14. *NDBAT $\mathcal{D}_l^{tt}$ and mapping m^{tt} satisfy Constraint 13.*

If we impose Constraint 13, then the agent that executes the program that implements the HL action can be a dumb executor. But this may seem too restrictive. An alternative is to impose the weaker requirement that for any HL agent action that is possibly executable at the LL, the agent has a strategy to execute it to termination no matter how the environment reacts (even if not controlling its outcome). Formally:

Constraint 15. *(Agt Can Always Execute HL actions) For every high-level system action sequence $\vec{\alpha}$ and every high-level action A, we have that:*

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s.(Do(m_s(\vec{\alpha}), S_0, s) \supset \forall \vec{x}. (\exists s'.Do_{ag}(m_a(A(\vec{x})), s, s') \supset AgtCanForce(m_a(A(\vec{x})), s)))$$

Note that if we only impose Constraint 15, then there is no guarantee that when a HL system action $A(\vec{x}, e)$ is possibly executable, the environment can actually ensure that reaction e occurs when the agent executes $m_a(A(\vec{x}))$. We can define an additional constraint that ensures this; see [Banihashemi *et al.*, 2023] for details.

Planning for achievement goals. Returning to planning, we can now show that if Constraint 15 holds (i.e., the agent always knows how to execute high-level primitive actions at the low level, even if not controlling their outcome) and the agent has a strong plan to achieve a goal at the high level, then there exists a refinement of the high-level plan that is a strong plan to achieve the refinement of the goal at the low level:

Theorem 16. *If $M_h \sim_m M_l$ and Constraint 15 holds, then for any high-level system action sequence $\vec{\alpha}$ and any high-level situation-suppressed formula ϕ, we have that:*

if $M_h, v \models Executable(do(\vec{\alpha}, S_0)) \wedge AgtCanForce(\phi, do(\vec{\alpha}, S_0))$
then $M_l, v \models \exists s.Do(m_s(\vec{\alpha}), S_0, s) \wedge \forall s.Do(m_s(\vec{\alpha}), S_0, s) \supset AgtCanForce(m_f(\phi), s)$

Proof sketch. The proof is by induction on the length of the HL strategy f_h at $do(\vec{\alpha}, S_0)$, where the length of a strategy is the length of its longest branch, i.e., $length(f, s) \doteq 0$ if $f(s) = stop$ and $length(f, s) \doteq max_{Do_{ag}(f(s),s,s')}(length(f, s'))$ otherwise.

Assume the antecedent. It follows that there is a strategy f_h such that $AgtCanForceBy(\phi, s_h, f_h)$, where $s_h = do(\vec{\alpha}, S_0)$. By proof of Theorem 11 we have that at the LL $\exists s_l.Do(m_s(\vec{\alpha}), S_0, s_l)$ and for all such s_l we have that $s_h \sim_m^{M_h, M_l} s_l$. Take an arbitrary such s_l. We show that there exists a LLl strategy f_l such that $AgtCanForceBy(m_f(\phi), s_l, f_l)$.

Base case, when $length(f_h, s_h) = 0$, and $f_h(s_h) =$

$stop \wedge \phi[s_h]$. By Lemma 10, we have that $m_f(\phi)[s_l]$ and $AgtCanForceBy(m_f(\phi), s_l, f_l)$ for any strategy f_l such that $f_l(s_l) = stop$.

Inductive case: Assume the result holds for any HL strategy f_h of length at most k. We show that it must also hold for a HL strategy f_h of length $k+1$.
At the HL, by the definition of $AgtCanForceBy$, we have that there exists an high-level action $A(\vec{t})$ such that $f_h(s_h) = A(\vec{t}) \wedge \exists s'_h.Do_{ag}(A(\vec{t}), s_h, s'_h) \wedge$
$\forall s'_h.Do_{ag}(A(\vec{t}), s_h, s'_h) \supset AgtCanForceBy(\phi, s'_h, f_h)$.
By Constraint 15, there exists a LL strategy g_l such that $AgtCanForceBy(m_a(A(\vec{t})), s_l, g_l)$. For any s'_l such that $Do_{ag}(g_l, s_l, s'_l)$, it follows by Constraint 2 that there exists s'_h such that $s'_h \sim_m^{M_h, M_l} s'_l$. The length of f_h at s'_h is at most k. Thus by the induction hypothesis, there exists a LL strategy f'_l such that $AgtCanForceBy(m_f(\phi), s'_l, f'_l)$. It follows that $AgtCanForceBy(m_f(\phi), s_l, f_l)$ for $f_l = f'_l \circ g_l$. □

Example cont. As strategies use the $stop$ action, we first define a $stop_{HL}$ action at the high level which is mapped to the $stop_{LL}$ action at the concrete level; $stop_{HL}$ always terminates with the $Success_{HS}$ reaction. By Proposition 14, Constraint 13 (and thus also the weaker Constraint 15) is satisfied for our example domain.

We can show that the agent has a strong plan to achieve the goal of being at location 13 at the high level (dtf abbreviates $driveAndTryFix$):

$$\mathcal{D}_h^{tt} \models AgtCanForceBy(At_{HL}(13), S_0, f_h) \text{ where}$$

$$f_h(s) \doteq \begin{cases} stop_{HL} & \text{if } At_{HL}(13, s) \\ dtf(o,d) & \text{if } At_{HL}(o, s) \wedge o \neq 13 \wedge Road_{HL}(o, d, s) \\ & \wedge \neg Visited_{HL}(d, s) \wedge Spare_{HL}(d, s) \\ wait_{HL} & \text{otherwise} \end{cases}$$

The agent's high-level strategy f_h is to stop if she is already at location 13, otherwise to drive to a location which has not been visited previously and has a spare tire and fix a flat if one occurs; in all other cases she waits.

By Th. 16 there exists a strategy at the low level that is refinement of the high-level strategy; this strategy is the same as strategy f_l in Section 2. It is easy to show that f_l is a refinement of the high-level strategy f_h.

Planning for temporally extended goals/tasks. We can also show a similar result to the previous theorem for high-level programs/tasks/temporally extended goals: if Constraint 15 holds and the agent has a strategy to successfully execute an agent program (without concurrency) at the high level, then the agent also has a strategy to successfully execute some refinement of it at the low level:

Theorem 17. *If $M_h \sim_m M_l$ and Constraint 15 holds, then for any high-level system action sequence $\vec{\alpha}$ and any SD ConGolog high-level agent program δ without the concurrent composition construct, we have that:*

$$\begin{aligned} &\text{if } M_h, v \models Executable(do(\vec{\alpha}, S_0)) \wedge \\ &\qquad AgtCanForce(\delta, do(\vec{\alpha}, S_0)) \\ &\text{then } M_l, v \models \exists s.Do(m_s(\vec{\alpha}), S_0, s) \wedge \\ &\qquad \forall s.Do(m_s(\vec{\alpha}), S_0, s) \supset AgtCanForce(m_a(\delta), s) \end{aligned}$$

Here, we apply the mapping to a high-level agent program δ without concurrency to produce a low level agent program $m_a(\delta)$; this can be defined in the obvious way, using m_a to map atomic actions and m_f for tests as usual, and by mapping the components and composing the result for other constructs, e.g., $m_a(\delta_1|\delta_2) = m_a(\delta_1)|m_a(\delta_2)$ (we discuss concurrency in [Banihashemi *et al.*, 2023]).

Proof sketch. The proof is by induction on the length of the high-level strategy f_h for executing δ at $do(\vec{\alpha}, S_0)$ and is similar to that of the above theorem. Note that since δ is SD, does not involve concurrency, and the high-level atomic actions are all mapped to SD programs, $m_a(\delta)$ must also be SD (we execute a high-level action using the appropriate strategy essentially as a procedure call and the high-level program continues after the call has completed). □

Example cont. Consider the high-level program $\delta_{go}^{HL}(l) = (\pi o, d.driveAndTryFix(o, d))^*; At_{HL}(l)?$ which goes to location l by repeatedly picking adjacent locations and doing $driveAndTryFix(o, d)$ until the agent is at l. We can show that the agent has a strategy to successfully execute this program to get to location 13:

$$\mathcal{D}_h^{tt} \models AgtCanForceBy(\delta_{go}^{HL}(13), S_0, f_h)$$

Here, the high-level strategy f_h to execute the program is same as the one we saw above. It can also be shown that the low-level strategy f_l seen earlier is a refinement of f_h and can be used to to execute $m_a(\delta_{go}^{HL}(13))$. To further illustrate the use of high-level programs, notice that we could give the agent the program $\delta_{go}^{HL}(31); \delta_{go}^{HL}(13)$ to execute. This is a more complex task as she must first go to location 31 and then to 13; but it is easier to to find a strategy to do it.

In [Banihashemi *et al.*, 2023], we show how we can handle more complex domains where reactions represent (sets of) exogenous actions.

6 Monitoring and Explanation

Being able to monitor what a low-level agent is doing and describe it in abstract terms (e.g., to a client or manager) is particularly important in the context of *Explainable AI*. (BDL17) show how a sound abstraction wrt a mapping m can be used to do this when certain conditions hold. They define an inverse mapping m^{-1} that takes a sequence of low-level actions $\vec{a}$ and returns the sequence of high-level actions $\vec{\alpha}$ that it refines in a given model. In [Banihashemi *et al.*, 2023], we show that we can easily adapt this approach for NDBAT abstractions.

Example cont. Consider a variant of our running example where, if the agent has a flat and there is no spare tire in a location (i.e., at locations 11, 12 and 13), she can either buy one, or she can use a roadside assistance service to get one provided the location is within the covered area, We can have HL actions for these with the mapping:

$$\begin{aligned} &m_s^{tt+}(buyAndFix(l, r_h)) = \pi r.order(l, r); \\ &\quad \pi r.pay(l, r); \pi r.fixFlatTire(l, r); r_h = Succ_{HBuy}? \\ &m_s^{tt+}(serviceAndFix(l, r_h)) = \pi.r.callService_{LL}(l, r); \\ &\quad \pi r.fixFlatTire(l, r); r_h = Succ_{HServ}? \end{aligned}$$

Suppose that at the LL, the agent knows that both locations 12 and 13 fall under the roadside service coverage, while at the HL, the agent only knows this for 13 (we have imperfect information). We can define NDBATs for this and show $\mathcal{D}_h^{tt+}$ is a sound abstraction of $\mathcal{D}_l^{tt+}$[Banihashemi *et al.*, 2023].

Now suppose that at the LL, the sequence of system actions $\vec{a} = drive(11, 21, NoFlatTire); drive(21, 12, FlatTire)$ has occurred. In location 12, no spare tire is available, and if the agent wants to move on with her journey, she needs to either call for service or order a new tire. The inverse mapping tells us that HL system action sequence $\vec{\alpha} = driveAndTryFix(11, 21, DrvNoFlat); driveAndTryFix(21, 12, DrvFlat)$ has occurred. Since $\mathcal{D}_h^{tt+} \models At_{HL}(12, do(\vec{\alpha}, S_0))$, we can conclude at the HL that the agent is now at location 12. Since $\mathcal{D}_h^{tt+} \cup \{Poss(serviceAndFix(12, Succ_{HServ}), do(\vec{\alpha}, S_0))\}$ is satisfiable, we can also conclude that $serviceAndFix(12, Succ_{HServ})$ might occur next, and similarly for $buyAndFix(12, Succ_{HBuy})$; in fact we know at the HL that the latter is executable: $\mathcal{D}_h^{tt+} \models Poss(buyAndFix(12, Succ_{HBuy}), do(\vec{\alpha}, S_0))$.

7 Discussion

There has been much previous work on fully/partially observable nondeterministic planning, e.g., [Ghallab *et al.*, 2004; Cimatti *et al.*, 2003; Muise *et al.*, 2014], strategy logics such as ATL [Alur *et al.*, 2002], and strategic reasoning [Xiong and Liu, 2016b; Xiong and Liu, 2016a]. But such work assumes that the domain is represented at a single level of abstraction.

In planning, approaches such as hierarchical task networks (HTNs) [Erol *et al.*, 1994] support abstract tasks that facilitate search. [Kuter *et al.*, 2009] propose an algorithm which combines HTN-based search-control strategies with Binary Decision Diagram-based state representations for planning in nondeterministic domains. [Chen and Bercher, 2021] extend HTN planning with nondeterministic primitive tasks. [Bonet *et al.*, 2017] investigate nondeterministic abstractions for generalized planning, where one looks for a (typically iterative) plan that solves a whole class of related planning problems. [Cui *et al.*, 2021] applies the abstraction framework of (BDL17) to generalized planning. They use Golog programs to represent nondeterministic actions, and our results should make it possible to simplify their framework.

[Banihashemi *et al.*, 2018] generalized the approach of (BDL17) to deal with agents that may acquire new knowledge during execution, and formalized (piecewise) hierarchical refinements of an agent's ability (i.e., strategy) to achieve a goal. However, this work does not consider nondeterministic actions and is based on a more complex framework involving online executions.

Abstraction is important for efficient reasoning and explainability. This paper presents foundational results, i.e., a generic framework for abstraction in nondeterministic domains, which can be used for many reasoning tasks, such as planning, execution monitoring, and explanation. One potential practical application area is smart manufacturing. For instance, [De Giacomo *et al.*, 2022] present a "manufacturing as a service" framework where facilities (made up of *concrete* resources) bid to produce products, given an *abstract* product recipe, which is based on an *abstract* information model in the cloud. Our work shows how one can develop an abstract domain model where planning for arbitrary high-level goals is easier and where we have guarantees that a high-level plan can be refined into a low-level one. It also allows one to monitor the low-level system by generating high-level descriptions of low-level system executions from which one can reason at the high level about what might happen next. Note that often manufacturing processes depend on the data and objects (parts) they produce and consume; to formalize this aspect, the situation calculus which provides a first-order representation of the state of the processes can indeed be used. Another potential practical application area is business process management, to support execution monitoring, explanation, and failure handling, see e.g., [Marrella *et al.*, 2017].

As discussed in Section 3, (BDL17) identifies a set of properties that are necessary and sufficient to have a sound/complete abstraction wrt a mapping, which can be used to verify that one has such an abstraction. This means that in the finite domain/propositional case, verifying that one has a sound abstraction is decidable and one can use theorem proving techniques to do it. Moreover, when one has complete information and a single model, one can use model-checking techniques. Bounded basic action theories [De Giacomo *et al.*, 2016] constitute an important class of infinite domain theories where verification is decidable. In the general infinite-domain case, there are sound but incomplete reasoning methods that can be used [De Giacomo *et al.*, 2010].

Automatically synthesizing sound/complete abstractions is another interesting problem. [Luo *et al.*, 2020] show that one can use the well-explored notion of forgetting (of low-level fluent and action symbols) to automatically obtain a sound and complete high-level abstraction of a low-level BAT *for a given mapping* under certain conditions. They also show that such an abstraction is computable in the propositional case. However in general, there may be many different abstractions of a low-level theory, each of which may be useful for a different purpose. So defining an abstract language and mapping for a domain is not a trivial problem. Some human intervention is likely to be required, e.g., the modeler might specify the goals of the abstraction, or which details can be considered unimportant.

In this paper, we have not considered fairness constraints and strong cyclic plans [D'Ippolito *et al.*, 2018; Aminof *et al.*, 2020]; this is a topic for future work. Another interesting direction for future research is to consider a notion of multi-tier planning [Ciolek *et al.*, 2020], where planning may be done using a simple model without unlikely contingencies, but where one can fall back to a more complex model when such contingencies do occur or more robustness is required. Related to this is the notion of "best effort strategies" (in presence of multiple [contradictory] assumptions about the environment) as proposed by [Aminof *et al.*, 2021]. These are agent plans which, for each of the environment specifications individually, realize the agent's goal against a maximal set of environments satisfying that specification. We are also interested in extending the current framework to deal with partial observability and sensing.

Acknowledgments

Work supported by the ERC Advanced Grant WhiteMech (No. 834228), by the EU ICT-48 2020 project TAILOR (No. 952215), as well as by the National Science and Engineering Research Council of Canada.

References

[Abadi *et al.*, 1989] Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and Unrealizable Specifications of Reactive Systems. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 1–17. Springer, 1989.

[Alur *et al.*, 2002] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[Aminof *et al.*, 2020] Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Stochastic fairness and language-theoretic fairness in planning in nondeterministic domains. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 20–28. AAAI Press, 2020.

[Aminof *et al.*, 2021] Benjamin Aminof, Giuseppe De Giacomo, Alessio Lomuscio, Aniello Murano, and Sasha Rubin. Synthesizing best-effort strategies under multiple environment specifications. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 42–51, 2021.

[Bacchus *et al.*, 1999] Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors and effectors in the situation calculus. *Artif. Intell.*, 111(1-2):171–208, 1999.

[Banihashemi *et al.*, 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1048–1055. AAAI Press, 2017.

[Banihashemi *et al.*, 2018] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction of agents executing online and their abilities in the situation calculus. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1699–1706. ijcai.org, 2018.

[Banihashemi *et al.*, 2023] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction of nondeterministic situation calculus action theories – Extended version. *CoRR*, abs/2305.14222, 2023.

[Bonet *et al.*, 2017] Blai Bonet, Giuseppe De Giacomo, Hector Geffner, and Sasha Rubin. Generalized planning: Nondeterministic abstractions and trajectory constraints. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 873–879. ijcai.org, 2017.

[Chen and Bercher, 2021] Dillon Chen and Pascal 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, Guangzhou, China (virtual), August 2-13, 2021*, pages 74–84. AAAI Press, 2021.

[Church, 1963] Alonzo Church. Logic, arithmetics, and automata. In *Proc. International Congress of Mathematicians, 1962*. institut Mittag-Leffler, 1963.

[Cimatti *et al.*, 1998] Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning in non-deterministic domains via model checking. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998*, pages 36–43. AAAI, 1998.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[Ciolek *et al.*, 2020] Daniel Alfredo Ciolek, Nicolás D'Ippolito, Alberto Pozanco, and Sebastian Sardiña. Multi-tier automated planning for adaptive behavior. In *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*, pages 66–74. AAAI Press, 2020.

[Clarke and Emerson, 1981] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[Cui *et al.*, 2021] Zhenhe Cui, Yongmei Liu, and Kailun Luo. A uniform abstraction framework for generalized planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 1837–1844. ijcai.org, 2021.

[De Giacomo and Lespérance, 2021] Giuseppe De Giacomo and Yves Lespérance. The nondeterministic situation calculus. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*, pages 216–226, 2021.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *Principles of Knowledge Representation and*

Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010. AAAI Press, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise. On supervising agents in situation-determined ConGolog. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1031–1038. IFAAMAS, 2012.

[De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artif. Intell.*, 237:172–203, 2016.

[De Giacomo *et al.*, 2022] Giuseppe De Giacomo, Paolo Felli, Brian Logan, Fabio Patrizi, and Sebastian Sardiña. Situation calculus for controller synthesis in manufacturing systems with first-order state representation. *Artif. Intell.*, 302:103598, 2022.

[D'Ippolito *et al.*, 2018] Nicolás D'Ippolito, Natalia Rodríguez, and Sebastian Sardiña. Fully observable non-deterministic planning as assumption-based reactive synthesis. *J. Artif. Intell. Res.*, 61:593–621, 2018.

[Erol *et al.*, 1994] Kutluhan Erol, James A. Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2*, pages 1123–1128. AAAI Press / The MIT Press, 1994.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019.

[Kuter *et al.*, 2009] Ugur Kuter, Dana S. Nau, Marco Pistore, and Paolo Traverso. Task decomposition on abstract states, for planning under nondeterminism. *Artif. Intell.*, 173(5-6):669–695, 2009.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[Luo *et al.*, 2020] Kailun Luo, Yongmei Liu, Yves Lespérance, and Ziliang Lin. Agent abstraction via forgetting in the situation calculus. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 809–816. IOS Press, 2020.

[Marrella *et al.*, 2017] Andrea Marrella, Massimo Mecella, and Sebastian Sardiña. Intelligent process adaptation in the smartpm system. *ACM Trans. Intell. Syst. Technol.*, 8(2):25:1–25:43, 2017.

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

[Milner, 1971] Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971*, pages 481–489. William Kaufmann, 1971.

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

[Muise *et al.*, 2014] Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 2322–2329. AAAI Press, 2014.

[Pinto *et al.*, 2000] Javier Pinto, Amílcar Sernadas, Cristina Sernadas, and Paulo Mateus. Non-determinism and uncertainty in the situation calculus. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.*, 8(2):127–150, 2000.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, 1989.

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

[Seegebarth *et al.*, 2012] Bastian Seegebarth, Felix Müller, Bernd Schattenberg, and Susanne 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, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI, 2012.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Xiong and Liu, 2016a] Liping Xiong and Yongmei Liu. Strategy representation and reasoning for incomplete information concurrent games in the situation calculus. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1322–1329. IJCAI/AAAI Press, 2016.

[Xiong and Liu, 2016b] Liping Xiong and Yongmei Liu. Strategy representation and reasoning in the situation calculus. In *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)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 982–990. IOS Press, 2016.