The Nondeterministic Situation Calculus

Giuseppe De Giacomo[1] and **Yves Lespérance**[2]
[1]DIAG, Università degli Studi di Roma “La Sapienza”, Rome, Italy
[2]EECS, York University, Toronto, Canada
degiacomo@diag.uniroma1.it, lesperan@eecs.yorku.ca

Abstract

The standard situation calculus assumes that atomic actions are deterministic. But many domains involve nondeterministic actions, with problems such as fully observable nondeterministic (FOND) planning and high-level program execution requiring solutions. Various approaches have been proposed to accommodate nondeterminism on top of the standard situation calculus language, for instance by introducing nondeterministic programs as in Golog and ConGolog. But a key problem in these approaches is that they don't clearly distinguish between choices that can be made by the agent and choices that are made by the environment, i.e., angelic vs. devilish nondeterminism. In this paper, we propose a simple extension to the standard situation calculus that accommodates nondeterministic actions and preserves Reiter's solution to the frame problem and answering projection queries through regression. We also provide a formalization of FOND planning and show how ConGolog high-level program execution in nondeterministic domains can be defined.

1 Introduction

The situation calculus is a popular predicate logic formalism for reasoning about action and change (McCarthy and Hayes 1969; Reiter 2001). In this formalism, a dynamic domain can be represented by a basic action theory, where successor state axioms represent the causal laws of the domain and provide a solution to the frame problem (Pirri and Reiter 1999; Reiter 2001). The formalism has been extended in many ways and used to formalize many problems and support reasoning in applications.

In the standard situation calculus and basic action theories, *atomic actions are deterministic*, i.e., the situation/state that results from performing an action in a situation/state is unique and determined by the action and situation/state prior to it. This is the case in two ways. First, the situation that results from performing action a in situation s is represented by term $do(a, s)$, which denotes a unique situation in each model. Secondly, in each model of a basic action theory, the extension of each fluent in $do(a, s)$ is uniquely determined by fluent extensions in s and the action a, as specified by the successor state axioms. So the resulting state is determined by the action and state prior to it.

However, sometimes we need to model actions that are *nondeterministic*, e.g., the action of flipping a coin where the result may be heads or tails, or the unreliable action of a robot unstacking a block where the result may be either the robot holding the block (success) or the block falling on the table (failure mode 1) or the block remaining where it was (failure mode 2). Such nondeterministic actions are really the results of two phenomena: the agent instructing the action and the environment reacting to it, as in a game. But once the agent's action has been instructed and the environment's reaction has been chosen, we get a single resulting situation/outcome. We may say that the “system action”, formed by the agent's action and the environment's reaction, remains indeed deterministic, as in the standard situation calculus.

The need to handle nondeterministic actions has long been recognized in the area, and somehow handled in an ad-hoc manner, but never studied in detail. One option is to model the different outcomes as distinct (system) actions, e.g., $flipHead$ and $flipTail$ for coin tossing, but this does not distinguish between the instruction of the action, which is decided by the agent, and its outcome, which is chosen by the environment. So we have to add an additional mechanism to say that the agent cannot really distinguish before execution between $flipHead$ and $flipTail$ and just sees them as $flip$ (Bacchus *et al.* 1999). Having to introduce indistinguishability to handle such a basic feature is cumbersome (indeed the notion was introduced for handling stochastic uncertainty, see below), and in any case requires a fair amount of bookkeeping overhead. Another option is to introduce exogenous events (De Giacomo *et al.* 2000) to model the environment's reaction explicitly. Then nondeterministic actions are represented by an ordinary action corresponding to the agent instructing the action itself and exogenous events corresponding to the possible environment responses. E.g., we could have action $flip$, which *per se* only makes the preconditions of the possible exogenous events $head$ and $tail$ true, and then an exogenous event chosen among $head$ and $tail$ which brings about the actual (nondeterministic) effect of the $flip$. While this is a possible solution, it is again cumbersome and requires significant bookkeeping for making everything work as desired.

In this paper, we propose a simple extension to the standard situation calculus to handle nondeterministic actions that preserves the solution to the frame problem and answering projection queries through regression. A key aspect of our formalization is that we maintain deterministic *system actions*, as in the standard situation calculus, but see them as formed by two separate contributions, the *agent's action* and the *environment's reaction*. Specifically, we simply model the environment reaction as an extra parameter of the action. System actions have this extra parameter, while the agent's actions are simply environment reaction-suppressed system actions. We show that this introduces only minimal overhead, while giving us the power to quantify independently over agent actions and environment reactions. This allows us to distinguish between the *angelic nondeterminism* of the agent's choosing actions, through existential quantification, and the *devilish nondeterminism* of the environment's reaction (which is not controlled by the agent), through universal quantification, see (Broy and Wirsing 1981). We also show how we can formalize fully observable nondeterministic (FOND) planning in our framework. Furthermore, we show how ConGolog (De Giacomo *et al.* 2000) high-level program execution in nondeterministic domains can be defined. This is of particular interest since we can indeed write angelically nondeterministic programs for the agent, as advocated in (Levesque *et al.* 1997), in a devilish nondeterministic environment.

As already mentioned, nondeterministic actions are related to stochastic actions and noisy sensing, which have indeed been studied in detail in the situation calculus. In (Reiter 2001), (see also (Pinto *et al.* 2000)), a situation calculus formalization of stochastic actions and decision-theoretic planning is proposed, where an abbreviation $choice(agent\_action, system\_action)$ is introduced to specify nature's choices, e.g., one might have $choice(\mathit{flip}(c), a) \doteq a{=}\mathit{flipHead}(c,s) \lor a{=}\mathit{flipTail}(c,s)$, and then probabilities are specified for each possible nature choice for an agent action in a situation. The account then goes on to define probabilities for executions of programs involving such stochastic actions as well as for a condition holding after the execution of such a program (sensing actions are also handled as a kind of stochastic actions, as are exogenous actions). The programs themselves however, are deterministic, so all choices are nature's choices. It is also shown how one can define the expected value of policies in MDP-like domains.

Nondeterministic actions are also considered in Bacchus *et al*'s situation calculus-based account of uncertainty and noisy acting and sensing (Bacchus *et al.* 1995; Bacchus *et al.* 1999). They model nondeterministic actions by introducing an extra parameter (like us) into the action function whose value is chosen by nature, e.g., $\mathit{flip}(c, outcome)$. These possible choices can be specified to follow a given probability distribution. The agent is not aware of the action's outcome until she performs suitable sensing actions. Degree of belief in propositions is defined in a "possible world" model where the weight of each possible situation depends on that of the initial situation it evolved from and the probability of each action transition leading to it. Degree of belief in a proposition following a sequence of agent actions is handled by quantifying existentially over nature's choices (and relying on Golog's Do). But they do not consider programs involving nondeterminism over the agent's choice of action, nor do they address stochastic planning. Later work in the same line of research in (Belle and Levesque 2020) essentially retains this approach.

Finally, as mentioned, the dynamics of a system with nondeterministic actions can be seen as a game between the agent and the environment, with each making their moves. This game-theoretic view can indeed be formalized in situation calculus variants for capturing games (De Giacomo *et al.* 2010; De Giacomo *et al.* 2016c). However, it seems like overkill to handle nondeterministic actions by moving to such a sophisticated setting. The solution we propose here is much simpler, and more elegant as it remains very close to the standard situation calculus and basic action theories.

2 The Situation Calculus

The *situation calculus* is a well known sorted predicate logic language for representing and reasoning about dynamically changing worlds (McCarthy and Hayes 1969; Reiter 2001). It includes three sorts: $Action$, $Situation$, and $Object$. 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 where no actions have yet been performed. 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., $Holding(x, s)$). $s \sqsubset s'$ means that s is a predecessor situation to s', and $s \sqsubseteq s'$ stands for $s \sqsubset s' \lor s = s'$. The abbreviation $do([a_1, \ldots, a_n], s)$ stands for $do(a_n, do(a_{n-1}, \ldots do(a_1, s) \ldots))$.

Within the language, one can formulate action theories that describe how the world changes as a result of the available actions. A *basic action theory (BAT)* $\mathcal{D}$ (Pirri and Reiter 1999; Reiter 2001) is the union of the following disjoint sets of axioms: the foundational, domain independent, axioms of the situation calculus (Σ), which include a second order axiom characterizing the situation domain; precondition axioms stating when actions can be legally performed ($\mathcal{D}_{poss}$); successor state axioms describing how fluents change between situations ($\mathcal{D}_{ssa}$); unique name axioms for actions ($\mathcal{D}_{una}$); and axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; precondition axioms in $\mathcal{D}_{poss}$ characterize this predicate. $Executable(s)$ means that every action performed in reaching situation s is executable in the situation in which it occurs. In turn, successor state axioms encode the causal laws of the world being modeled; they take the place of the so-called effect axioms and provide a solution to the frame problem.

A key feature of BATs is the existence of a sound and complete *regression mechanism* for answering queries about situations resulting from performing a sequence of actions (Pirri and Reiter 1999; Reiter 2001). In a nutshell, the regression operator $\mathcal{R}^*$ reduces a formula ϕ about a particular future situation to an equivalent formula $\mathcal{R}^*[\phi]$ about the initial situation S_0, essentially by substituting fluent relations with the right-hand side formula of their successor state axioms. A formula ϕ is *regressable* if and only if *(i)* all situation terms in it are of the form $do([a_1, \dots, a_n], S_0)$, *(ii)* in every atom of the form $Poss(a, \sigma)$, the action function is specified, i.e., a is of the form $A(t_1, \dots, t_n)$, *(iii)* it does not quantify over situations, and *(iv)* it does not contain $\sqsubset$ or equality over situation terms. Thus in essence, a formula is regressable if it does not contain situation variables. Another key result about BATs is the relative satisfiability theorem (Pirri and Reiter 1999; Reiter 2001): $\mathcal{D}$ is *satisfiable* if and only if $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$ is satisfiable (the latter being a purely first-order theory). This implies that we can check if a regressable formula ϕ is entailed by $\mathcal{D}$, by checking if its regression $\mathcal{R}^*[\phi]$ is entailed by $\mathcal{D}_{S_0} \cup \mathcal{D}_{una}$ only.

3 The Nondeterministic Situation Calculus

For any primitive action by the agent in a nondeterministic domain, there can be a number of different outcomes. We take the outcome as being determined by the agent's action and the environment's reaction to this action. We represent this 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*. We use the notation $A(\vec{x})[e]$ to integrate the reaction e to the agent action $A(\vec{x})$. Note that $A(\vec{x})[e] \doteq A(\vec{x}, e)$. As usual, we denote the situation that results from the system action $A(\vec{x}, e)$ occurring in situation s by the term $do(A(\vec{x}, e), s)$ (i.e., the agent's action $A(\vec{x})$ followed by environment reaction e).

Nondeterministic Basic Action Theories (NDBATs) We represent nondeterministic domains using action theories called Nondeterministic Basic Action Theories (NDBATs), which can be seen as a special kind of BAT, where:

- every *action function* takes an *environment reaction parameter*;
- for each agent action we have an *agent action precondition* formula, stating when the action can be performed by the agent, specifically for each agent action $A(\vec{x})$ we denote by $Poss_{ag}(A(\vec{x}), s)$ its agent action precondition:[1]

$$Poss_{ag}(A(\vec{x}), s) \doteq \phi_A^{agPoss}(\vec{x}, s);$$

- for each agent action we have a *reaction independence requirement*, stating that the precondition for the agent action is independent of any environment reaction:

$$\forall e. Poss(A(\vec{x}, e), s) \supset Poss_{ag}(A(\vec{x}), s)$$

these requirements *must be entailed* by the action theory for it to be an NDBAT;

- for each agent action we also 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, i.e., the environment cannot prevent the agent from performing an action when its agent action precondition holds:

$$Poss_{ag}(A(\vec{x}), s) \supset \exists e. Poss(A(\vec{x}, e), s);$$

again these requirements *must be entailed* by the action theory for it to be an NDBAT.

A *nondeterministic basic action theory (NDBAT)* $\mathcal{D}$ is the union of the following disjoint sets:

- *Foundational, domain independent, axioms of the situation calculus* (Σ), as in standard BATs (Pirri and Reiter 1999; Reiter 2001);
- *Unique name axioms for actions* ($\mathcal{D}_{una}$), as in standard BATs.
- *Axioms describing the initial situation* ($\mathcal{D}_{S_0}$), as in standard BATs;
- *Successor state axioms* describing how fluents change after system actions are performed ($\mathcal{D}_{ssa}$), as in standard BATs;
- *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:

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

In practice, most often these axioms have the form

$$Poss(A(\vec{x}, e), s) \equiv Poss_{ag}(A(\vec{x}), s) \land \psi_A^{poss}(\vec{x}, e, s)$$

where $Poss_{ag}(A(\vec{x}), s)$ guarantees the executability of the agent action $A(\vec{x})$ and $\psi_A^{poss}(\vec{x}, e, s)$ characterizes the possible reactions to such an agent action; when this is the case, the reaction independence requirement $\forall e. Poss(A(\vec{x}, e), s) \supset Poss_{ag}(A(\vec{x}), s)$ is trivially satisfied, while the reaction existence requirement $Poss_{ag}(A(\vec{x}), s) \supset \exists e. Poss(A(\vec{x}, e), s)$ reduces to the simpler requirement that $Poss_{ag}(A(\vec{x}), s) \supset \exists e. \psi_A^{poss}(\vec{x}, e, s)$, i.e., if the agent action is possible then some environment reaction must always exist. must be

Example 1. *Suppose that we have an agent that can move back and forth along a track. Initially, the agent is at position* 0. *The agent can perform the action fwd to go forward, but this action is nondeterministic and as a result, she may move forward by one or two positions. The agent can also perform the deterministic action* $bk1$*, to go backwards by exactly one postion. We can specify this domain as an NDBAT*

[1] In this paper, when we write $\varphi(\vec{x}, s)$, as e.g., here $\phi_A^{agPoss}(\vec{x}, s)$, we intend that the only free variables of $\varphi(\vec{x}, s)$ are among $\vec{x}$ and s, and moreover that $\varphi(\vec{x}, s)$ is uniform in s, i.e., s is the only situation term, see (Reiter 2001).

$\mathcal{D}_{trk}$ as follows:

$$\begin{array}{l}
pos(S_0) = 0\\
Poss_{ag}(\mathit{fwd}, s) \doteq True\\
Poss_{ag}(bk1, s) \doteq pos(s) > 0\\
Poss(\mathit{fwd}(e), s) \equiv\\
\quad Poss_{ag}(\mathit{fwd}, s) \land (e = f(1) \lor e = f(2))\\
Poss(bk1(e), s) \equiv Poss_{ag}(bk1, s) \land e = f(1)\\
pos(do(a, s)) = k \equiv\\
\quad a = \mathit{fwd}(e) \land \exists n.e = f(n) \land k = pos(s) + n \lor\\
\quad a = bk1(e) \land k = pos(s) - 1 \lor\\
\quad pos(s) = k \land (\neg\exists e.a = \mathit{fwd}(e)) \land (\neg\exists e.a = bk1(e))
\end{array}$$

Note that we use a function f to map natural numbers to reactions. We also include unique name axioms for actions, axioms stating that f is a bijection, the axioms of first-order arithmetic for natural numbers, and the foundational axioms in $\mathcal{D}_{trk}$. Note also that the reaction independence and reaction existence requirements are trivially entailed.

Note that successor state axioms can be obtained as usual using Reiter's method (Reiter 2001) if we assume that we have an effect axiom of the form

$$\phi(\vec{x}, \vec{y}, e, s) \supset [\neg]F(\vec{x}, \vec{y}, do(A(\vec{x}, e), s))$$

for each fluent $F(\vec{x}, \vec{y}, s)$ that is affected by a system action $A(\vec{x}, e)$.

Example 2. *For the domain in Example 1, we can specify the effect axioms as follows:*

$$\begin{array}{l}
pos(do(\mathit{fwd}(f(1)), s) = pos(s) + 1\\
pos(do(\mathit{fwd}(f(2)), s) = pos(s) + 2\\
pos(do(bk1(e), s) = pos(s) - 1
\end{array}$$

We can then obtain the successor state axiom in Example 1 by making the explanation closure assumption and applying Reiter's method.

Observe that an NDBAT is formally a BAT,[2] which must entail the additional reaction independence and reaction existence requirements. Hence, when writing a NDBAT, the designer must check that these requirements are indeed entailed. However all results in the literature about BATs apply to NDBATs as well.

The key feature of NDBATs is that we have separated the agent action $A(\vec{x})$ and the environment reaction e, in the system action $A(\vec{x}, e)$. This allows us to independently quantify over the two components. In particular, we can talk about the various possible executions (system actions) that may result from an agent performing an agent action and what may hold afterwards, considering the reaction of the environment.

Consider the familiar notation $Do(A(\vec{x}, e), s, s') \doteq Poss(A(\vec{x}, e), s) \land s' = do(A(\vec{x}, e), s)$, which denotes a relation between the current situation s and the next situation s' resulting from performing the complete system action $A(\vec{x}, e)$. Note that such a relation is *functional* since the next situation is unique if it exists, i.e., when $A(\vec{x}, e)$ is executable ($Poss(A(\vec{x}, e), s)$ holds).

For agent actions, we can define an analogous relation $Do_{ag}(A(\vec{x}), s, s')$, meaning that the system may reach situation s' when the agent executes the agent action $A(\vec{x})$ in situation s depending on the environment reaction. We define $Do_{ag}(A(\vec{x}), s, s')$ as follows (again as an abbreviation):

$$\begin{array}{l}
Do_{ag}(A(\vec{x}), s, s') \doteq\\
\quad \exists e.Poss(A(\vec{x}, e), s) \land s'{=}do(A(\vec{x}, e), s)
\end{array}$$

There is at least one situation s' reachable after the agent executes the agent action $A(\vec{x})$ in situation s if $A(\vec{x})$ is executable in s (given the requirement that $Poss_{ag}(A(\vec{x}), s) \supset \exists e.Poss(A(\vec{x}, e), s)$), and there may be several such situations depending on the environment reactions.

Example 3. *Continuing with our running example, the domain in Example 1, it is easy to show that:*

$$\begin{array}{l}
\mathcal{D}_{trk} \models pos(do(\mathit{fwd}(f(1)), S_0)) = 1 \land\\
\quad pos(do(\mathit{fwd}(f(2)), S_0)) = 2 \land\\
\quad pos(do(bk1(e), do(\mathit{fwd}(f(2)), S_0))) = 1 \land\\
\quad (Do_{ag}(\mathit{fwd}, S_0, s) \equiv\\
\qquad s = do(\mathit{fwd}(f(1)), S_0) \land pos(s) = 1 \lor\\
\qquad s = do(\mathit{fwd}(f(2)), S_0) \land pos(s) = 2) \land\\
\quad (Do_{ag}(bk1, S_0, s) \equiv s = do(bk1(f(1)), S_0))
\end{array}$$

By using Do_{ag} we can talk about outcomes that are *certainly true* after executing the agent action $A(\vec{x})$, i.e., true for all environment reactions:

$$\mathit{CertainlyAfter}(A(\vec{x}), \phi, s) \doteq \forall s'.Do_{ag}(A(\vec{x}), s, s') \supset \phi[s']$$

and *possibly true*, i.e., true after some environment reaction:

$$\mathit{PossiblyAfter}(A(\vec{x}), \phi, s) \doteq \exists s'.Do_{ag}(A(\vec{x}), s, s') \land \phi[s']$$

Indeed, $\mathit{CertainlyAfter}(A(\vec{x}), \phi, s)$ means that for all executions of agent action $A(\vec{x})$ in situation s (if any), ϕ holds afterwards, while $\mathit{PossiblyAfter}(A(\vec{x}), \phi, s)$ means that there exists an execution of agent action $A(\vec{x})$ in situation s after which ϕ holds. In the above, ϕ is a situation-suppressed formula (i.e., a formula with all situation arguments in fluents supressed); $\phi[s]$ denotes the formula obtained by restoring the situation argument s into all fluents in ϕ, see (Reiter 2001).

Example 4. *Continuing with our running example, we have:*

$$\begin{array}{l}
\mathcal{D}_{trk} \models \mathit{CertainlyAfter}(\mathit{fwd}, (pos = 1 \lor pos = 2), S_0) \land\\
\quad (k = 1 \lor k = 2 \supset \mathit{PossiblyAfter}(\mathit{fwd}, pos = k, S_0)) \land\\
\quad \mathit{CertainlyAfter}(\mathit{fwd}, (pos = 3 \lor pos = 4),\\
\qquad\qquad do(\mathit{fwd}(f(2)), S_0))
\end{array}$$

Example 5. *To illustrate how NDBATs go beyond propositional logic, consider a variant $\mathcal{D}_{trk^+}$ of the the domain in Example 1 where the fwd action may move the agent by arbitrarily many steps. We can axiomatize this by simply replacing the action precondition axiom for fwd in $\mathcal{D}_{trk^+}$ by the following:*

$$Poss(\mathit{fwd}(e), s) \equiv Poss_{ag}(\mathit{fwd}, s) \land \exists n.n{>}0 \land e{=}f(n)$$

In the above, we take n as ranging over the natural numbers.

For this domain, we can show that:

$$\mathcal{D}_{trk^+} \models \forall k.k > 0 \supset \mathit{PossiblyAfter}(\mathit{fwd}, pos = k, S_0)$$

[2] Assuming to use (a part of) the object sort as the reaction sort.

4 Executability, Projection, and Regression

We now turn to sequences of actions. First, we consider sequences of system actions. In this case, all standard definitions apply. In particular, we say that a (finite) sequence of system actions $[a_1, \dots, a_n]$ is *executable* in the initial situation by writing $Executable(do(a_n, , \dots, do(a_1, S_0)\dots))$ which is defined as usual (Reiter 2001):

$$Executable(s) \doteq \forall a, s'.do(a, s') \sqsubseteq s \supset Poss(a, s')$$

We also have a recursive characterization:

$$\begin{aligned}&Executable(s) \equiv s = S_0 \lor\\ &\quad \exists a, s'.s = do(a, s') \land Poss(a, s') \land Executable(s')\end{aligned}$$

By using $Executable$, we can check whether a situation-suppressed formula ϕ would hold after an executable sequence of system actions $[a_1, \dots, a_n]$ by checking whether $Executable(do(a_n, \dots, do(a_1, S_0)\dots)) \land \phi[do(a_n, \dots, do(a_1, S_0)\dots)]$. This fundamental form of hypothetical reasoning is called *projection* (Reiter 2001).[3]

When we separate the agent action from the environment reaction, the notions of executability and projection become more involved. In particular, we need to capture whether a sequence of agent actions is executable independently of the environment reactions, i.e., *certainly executable*, or whether it is executable for suitably chosen environment reactions, i.e., *possibly executable*. Formally we can define:

$$\begin{aligned}&CertainlyExecutable(\epsilon, s) \doteq True\\ &CertainlyExecutable([A(\vec{x}), \sigma], s) \doteq\\ &\quad Poss_{ag}(A(\vec{x}), s) \land\\ &\quad \forall s'.Do_{ag}(A(\vec{x}), s, s') \supset CertainlyExecutable(\sigma, s')\end{aligned}$$

that is, in every situation that may result from the agent performing the first action, the remaining action sequence must be certainly executable. Also, we can define:

$$\begin{aligned}&PossiblyExecutable(\epsilon, s) \doteq True\\ &PossiblyExecutable([A(\vec{x}), \sigma], s) \doteq\\ &\quad \exists s'.Do_{ag}(A(\vec{x}), s, s') \land PossiblyExecutable(\sigma, s')\end{aligned}$$

that is, in some situation that may follow the agent performing the first action, the remaining action sequence is possibly executable.

Next we turn to projection for sequences of agent actions. We start by extending the notion of $Do_{ag}(A(\vec{x}), s, s')$ from single agent actions $A(\vec{x})$ to (finite) sequences of agent actions $\sigma = [A_1(\vec{x_1}), \dots, A_n(\vec{x_n})]$. The notion $Do_{ag}(\sigma, s, s')$ means that the system, starting from situation s, may reach situation s' when the agent executes the agent action sequence σ, with each agent action getting an environment reaction (which are not under the agent's control). This is defined as follows (again as an abbreviation):

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

With $Do_{ag}(\sigma, s, s')$, we can easily look at the various notions of projection that come out of the separation of the agent actions and the environment reactions. Specifically, given a (situation-suppressed) formula ϕ, we can check whether it holds after the execution of a sequence of agent actions independently of the reactions of the environment, i.e., ϕ is *certainly true after* the sequence of agent actions σ, or that it holds with some reaction of the environment, i.e., ϕ is *possibly true after* the sequence of agent actions σ. Formally:

$$\begin{aligned}&CertainlyAfter(\sigma, \phi, s) \doteq \forall s'.Do_{ag}(\sigma, s, s') \supset \phi[s']\\ &PossiblyAfter(\sigma, \phi, s) \doteq \exists s'.Do_{ag}(\sigma, s, s') \land \phi[s']\end{aligned}$$

$CertainlyAfter(\sigma, \phi, s)$ does not guarantee per se that the sequence σ is indeed executable for all environment reactions (or even for same of them). If we want to ensure that σ forces ϕ to hold afterwards, we need to ensure both that ϕ is certainly true after σ and that σ is executable independently of the reactions of the environment:

$$\begin{aligned}&ForcesAfter(\sigma, \phi, s) \doteq\\ &\quad CertainlyAfter(\sigma, \phi, s) \land CertainlyExecutable(\sigma, s)\end{aligned}$$

Note that $CertainlyAfter$ is analogous to a *partial correctness* requirement in Formal Methods while $ForcesAfter$ is analogous to a *total correctness* requirement (Hoare 1969).

Example 6. *Continuing with our running example, we have:*

$$\begin{aligned}&\mathcal{D}_{trk} \models PossiblyAfter([fwd, bk1], (pos = 0), S_0) \land\\ &\quad PossiblyAfter([fwd, bk1], (pos = 1), S_0) \land\\ &\quad ForcesAfter([fwd, bk1], (pos = 0 \lor pos = 1), S_0) \land\\ &\quad CertainlyAfter([fwd, bk1, bk1], (pos = 0), S_0) \land\\ &\quad \neg CertainlyExecutable([fwd, bk1, bk1], S_0)\end{aligned}$$

One of the main results for the situation calculus is that you can perform both executability and projection checks through *regression* when the situation is sufficiently ground (it may contain object, or for us reaction variables, but not situation or action variables) (Reiter 2001). We can show that this also applies to NDBATs:

Theorem 1. *Let $\mathcal{D}$ be an NDBAT, σ a ground agent action sequence, s a situation term containing no situation or action variables, and ϕ a situation-suppressed formula containing no action variables. Then $CertainlyExecutable(\sigma, s)$ is equivalent to a regressable formula, and so are $PossiblyExecutable(\sigma, s)$, $CertainlyAfter(\sigma, \phi, s)$, $PossiblyAfter(\sigma, \phi, s)$, and $ForcesAfter(\sigma, \phi, s)$.*

Proof: See Appendix A.

5 FOND Planning and Synthesis

Let us now consider fully observable nondeterministic (FOND) domains as used in AI planning (Cimatti *et al.* 1998; Geffner and Bonet 2013; Geffner and Geffner 2018). We use a first-order PDDL-like notation for FOND domains (Haslum *et al.* 2019).

A FOND domain (we include the initial state in the domain) is a tuple $\texttt{Dom} = (\texttt{Flu}, \texttt{Const}, \texttt{Act}, \texttt{Init})$ where:

[3] Projection can also be checked for sequences of actions that are not executable, thus separating projection from executability (Reiter 2001). We instead consider projection only for executable sequences of actions.

- $\mathtt{Flu}$ is a finite set of (situation-suppressed) fluents, each with a specified arity;
- $\mathtt{Const}$ is a finite set of object constants;
- $\mathtt{Init}$ is finite set of fluent atoms obtained from the fluents $\mathtt{Flu}$ and constants $\mathtt{Const}$, which are true initially, with each fluent being initially false for all other objects (i.e., we have complete information about the initial situation);
- $\mathtt{Act}$ is a finite set of actions $A(\vec{x})$ with parameters $\vec{x}$ with:
  - Precondition specification $\mathtt{Pre}(A(\vec{x}))$, which is a first-order formula over $\mathtt{Flu}$ stating the condition under which action $A(\vec{x})$ can be performed;
  - Effects specification $\mathtt{Eff}(A(\vec{x}))$, which is an expression of the form $\mathtt{oneof}(\mathtt{eff}_1, \ldots, \mathtt{eff}_n)$ with $n \geq 1$, where each alternative effect specification $\mathtt{eff}_i$ is a conjunction of clauses of the form $\mathtt{forall}\,\vec{y}.\,\mathtt{when}(\phi_j(\vec{x},\vec{y}), \psi_j(\vec{x},\vec{y}))$, with $\phi_j(\vec{x},\vec{y})$ a formula over $\mathtt{Flu}$ and $\mathtt{Const}$ with free variables in $\vec{x},\vec{y}$ and with all such $\phi_j(\vec{x},\vec{y})$ in one alternative effect specification $\mathtt{eff}_i$ mutually exclusive, and $\psi_j(\vec{x},\vec{y})$ a conjunction of (non-contradictory) positive and negative literals, having fluents in $\mathtt{Flu}$ as predicates, and either constants or variables in $\vec{x},\vec{y}$ as terms, where each such clause means that for all $\vec{y}$, if $\phi_j(\vec{x},\vec{y})$ holds when the action is performed, then afterwards the literals in $\psi_j(\vec{x},\vec{y})$ come to hold, and $\mathtt{oneof}(\mathtt{eff}_1, \ldots, \mathtt{eff}_n)$ means that one of the alternative effects $\mathtt{eff}_i$ where $1 \leq i \leq n$ is applied.

A goal is a closed first-order formula $\mathtt{Goal}$ over $\mathtt{Flu}$.

Following (Claßen *et al.* 2007), we translate such a FOND domain into a corresponding NDBAT $\mathcal{D}$ as follows:[4]

- For each fluent $F(\vec{x}) \in \mathtt{Flu}$, we include a situation calculus fluent $F(\vec{x})[s]$, i.e., $F(\vec{x}, s)$.
- For each action $A(\vec{x}) \in \mathtt{Act}$, we include an agent action $A(\vec{x})$ and a system action $A(\vec{x}, e)$, where e is the reaction parameter.
- We define the initial situation description as $F(\vec{x}, S_0) \equiv \bigvee_{F(\vec{t}) \in \mathtt{Init}} \vec{x} = \vec{t}$, for every fluent $F(\vec{x}) \in \mathtt{Flu}$.
- We define $Poss_{ag}(A(\vec{x}), s) \doteq \mathtt{Pre}(A(\vec{x}))[s]$.
- We define $Poss(A(\vec{x}, e), s) \equiv Poss_{ag}(A(\vec{x}), s) \land e{=}1 \lor \cdots \lor e{=}n$ where $\mathtt{Eff}(A(\vec{x})) = \mathtt{oneof}(\mathtt{eff}_1, \ldots, \mathtt{eff}_n)$. Note that the *reaction independence requirement* and *reaction existence requirement* are both trivially satisfied (in the FOND specification every effect/reaction can occur when the action preconditions hold).
- We define effect axioms
$$\phi_i(\vec{x},\vec{y})[s] \supset [\neg]F(\vec{x},\vec{y}, do(A(\vec{x}, i), s)$$
for each clause $\mathtt{forall}\,\vec{y}.\,\mathtt{when}(\phi_j(\vec{x},\vec{y}), \psi_i(\vec{x},\vec{y}))$ in $\mathtt{eff}_i$ such that $[\neg]F$ is in one of the conjuncts in ψ_i. From these effect axioms we obtain the successor axioms by handling the frame problem through explanation closure in the usual way (Reiter 2001).

We want to check whether there exists a (strong) plan, i.e., a conditional plan, to achieve the goal $\mathtt{Goal}$. The agent has a strong plan if she can choose an action such that for all environment reactions, she takes a step towards $\mathtt{Goal}$, and she can continue choosing actions in this way until she reaches $\mathtt{Goal}$. Formally, we can capture this by defining inductively a predicate $AgtCanForce(\mathtt{Goal}, s)$ as follows:

$$AgtCanForce(\mathtt{Goal}, s) \doteq \forall P.[\ \ldots\ \supset P(s)]$$

where $\ldots$ stands for

$$\begin{array}{l}[\mathtt{Goal}[s] \supset P(s)] \land \\ [\exists A.\exists\vec{x}.(Poss_{ag}(A(\vec{x}), s) \land \\ \qquad \forall e.(Poss(A(\vec{x}, e), s) \supset P(do(A(\vec{x}, e), s)))) \\ \supset P(s)]\end{array}$$

That is, $AgtCanForce(\mathtt{Goal}, s)$ is defined as the least set of situations such that (1) if $\mathtt{Goal}$ holds in situation s, then s is in the set, and (2) if there exists an action $A(\vec{x})$ that the agent can perform in situation s such that for every environment reaction e, the resulting situation $do(A(\vec{x}, e), s)$ is in the set, then s is in the set ($\exists A$ is a shorthand for a disjunction over all action types/functions).

Observe that we are handling simultaneously *two kinds of nondeterminism* here: an *angelic* one coming from the (existential) choices of the agent, and a *devilish* one coming from the possible (adversarial) reactions of the environment, which are indeed quantified universally. The ability to quantify separately over agent actions and environment reactions is crucial for defining such a notion.

We take a *strategy* to be 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. Note that situations represent histories in the situation calculus, since they record all system actions, i.e., agent actions and environment reaction; thus, a strategy may prescribe a different action in different situations where the state is the same. For convenience, we assume to have a special $\mathtt{stop}$ agent action with no effects and no preconditions, that we use to denote when the strategy stops.

Given a strategy, we can check whether it forces $\mathtt{Goal}$ to become true in spite of the environment reactions, i.e., is a strong plan to achieve the goal. When $\mathtt{Goal}$ is achieved, we require it to return the special action $\mathtt{stop}$. Formally, we can capture this by defining inductively a predicate $AgtCanForceBy(\mathtt{Goal}, s, f)$ as follows:

$$AgtCanForceBy(\mathtt{Goal}, s, f) \doteq \forall P.[\ \ldots\ \supset P(s)]$$

where $\ldots$ stands for

$$\begin{array}{l}[(f(s) = \mathtt{stop} \land \mathtt{Goal}[s]) \supset P(s)] \land \\ [\exists A.\exists\vec{t}.(f(s) = A(\vec{t}) \neq \mathtt{stop} \land Poss_{ag}(A(\vec{t}), s) \land \\ \qquad \forall e.(Poss(A(\vec{t}, e), s) \supset P(do(A(\vec{t}, e), s)))) \\ \supset P(s)]\end{array}$$

[4] In fact $\mathcal{D}$ can be generalized in several way maintaining all the properties we talk about later. First we could have, infinite many constants, and an initial situation description expressed as a set of first-order formulas, as long as they are admit only one model modulo isomorphism (we have complete information on the initial situation). Second, we could allow environment reactions of the form $e_i = E_i(\vec{t})$, $(i = 1, \ldots, n)$, which include object parameters $\vec{t}$ to be chosen by the environment. In this way even if the constants are finite in the initial situation description corresponding to $\mathtt{Init}$, arbitrarily many new objects can be introduced by the environment in its response along the execution. Studying conditions for decidability of such theories, such as forms of state-boundedness (De Giacomo *et al.* 2016a), is an interesting line for further work.

Notice that since the we are defining the predicate by induction (i.e., by least fixpoint), a strategy that defers forever without ever reaching the Goal would not satisfy such a definition. In other words, we are guaranteed that sooner or later by executing the actions prescribed by the strategy f, the agent will achieve Goal no matter what reactions the environment chooses.

The planning problem consists of finding a strategy f such that $AgtCanForceBy(\texttt{Goal}, S_0, f)$ holds. In particular, if $\mathcal{D}$ corresponds to a FOND Dom then any technique to synthesize (strong) plans in FOND, actually generates as the plan one such f.

Theorem 2. *Let* $\texttt{Dom} = (\texttt{Flu}, \texttt{Const}, \texttt{Act}, \texttt{Init})$ *be a FOND and* Goal *a goal over it. Let* $\mathcal{D}$ *be the NDBAT corresponding to* Dom*. Then every strong plan* f *that achieves* Goal *in* Dom *is a strategy such that* $\mathcal{D} \models AgtCanForceBy(\texttt{Goal}, S_0, f)$ *holds, and conversely any strategy* f *such that* $\mathcal{D} \models AgtCanForceBy(\texttt{Goal}, S_0, f)$ *holds is a strong plan for achieving* Goal *in* Dom*.*

Proof. (Sketch) We follow along the lines of (Claßen *et al.* 2007) to show the correctness of the translation of Dom into the NDBAT $\mathcal{D}$. Moreover that given the description in Dom, and in particular the fact that we have only finitely many constants, a finite complete description of the initial states in terms of facts, and that effects cannot introduce new terms, we may consider the object domain of $\mathcal{D}$ finite. Then we exploit the second-order definition of $AgtCanForceBy(\texttt{Goal}, S_0, f)$ to get the result. □

Related to this result, it is interesting to observe that the predicate $AgtCanForce(\texttt{Goal}, s)$ could alternatively be expressed in the Mu-Calculus as a least fixpoint (for details see (De Giacomo *et al.* 2020)):

$$\begin{array}{l} AgtCanForce(\texttt{Goal}, s) \doteq \\ \quad [\mu X, s.\texttt{Goal}(s) \lor \\ \qquad \exists A.\exists \vec{x}.Poss_{ag}(A(\vec{x}), s) \land \\ \qquad \forall e.(Poss(A(\vec{x}, e), s) \supset X(do(A(\vec{x}, e), s)))](s) \end{array}$$

This means that in the propositional case, or when we have finitely many objects, as in the case $\mathcal{D}$ is obtained from Dom, we can generate a finite transition system that is bisimilar to the situation tree of the NDBAT $\mathcal{D}$, and on such a transition system the above formula can be evaluated directly as a Mu-Calculus formula (De Giacomo *et al.* 2016a), thus giving us a technique to check for the existence of a strategy to achieve Goal via model checking, and then using the witnesses of the existentials in the fixpoint computation to build the strategy itself. In fact, the same idea applies also if $\mathcal{D}$ has an infinite object domain, but is *state-bounded* (De Giacomo *et al.* 2016a).

We close this section by observing that here we have not considered any fairness constraints in the domain (D'Ippolito *et al.* 2018; Aminof *et al.* 2020). Also, we have not handed incomplete information about the initial situation (Sardiña *et al.* 2006), which relates to generalized planning, i.e., have a plan that works in different domains (in our case models) (Srivastava *et al.* 2011; Bonet and Geffner 2018). Both of these notions are indeed of interest and deserve further study.

6 ConGolog Program Execution in Nondeterministic Domains

Next we turn to *high-level programs*, which comprise actions and tests that belong to the domain of concern (rather than being based on classical variables and assignments), and are meant to be executed against a theory of action. In the situation calculus, several such languages have been developed, including Golog (Levesque *et al.* 1997), which provides the usual structured programming constructs as well as constructs for nondeterministic choices, ConGolog (De Giacomo *et al.* 2000), which extends Golog to accommodate concurrency, and IndiGolog (Sardina *et al.* 2004), which also supports interleaving planning and execution.

Here, we consider programs in a variant of ConGolog without recursive procedures (De Giacomo *et al.* 2000) and where the test construct yields no transition and is final when satisfied (Claßen and Lakemeyer 2008; De Giacomo *et al.* 2010).[5] The key feature of our variant is that *programs instruct agent actions and not system actions.*

We consider the usual ConGolog constructs:

$A(\vec{x})$	agent action
$\phi?$	test for a condition
$\delta_1; \delta_2$	sequence
$\delta_1 \mid \delta_2$	nondeterministic branch
$\pi x.\delta$	nondeterministic choice of argument
δ^*	nondeterministic iteration
$\delta_1 \| \delta_2$	interleaved concurrency

where $A(\vec{x})$ is an *agent action* and ϕ is situation-suppressed (uniform) situation calculus formula. We require that the variable x in programs of the form $\pi x.\delta$ range over objects, and occurs in some action term in δ, i.e., $\pi x.\delta$ acts as a construct for the nondeterministic choice of action parameters. Conditional and while-loop constructs are definable as usual: **if** ϕ **then** δ_1 **else** $\delta_2 = \phi?; \delta_1 | \neg\phi?; \delta_2$ and **while** ϕ **do** $\delta = (\phi?; \delta)^*; \neg\phi?$. We use ϵ to denote the empty program; in fact ϵ is simply an abbreviation for $True?$.

Programs are meant to be executed over an NDBAT. This means that the actions, fluents, and constants mentioned in a program δ must be those in the NDBAT.

The semantics of ConGolog is specified as usual in terms of single-steps, using the following two predicates (De Giacomo *et al.* 2000): $Final(\delta, s)$, specifying that the program δ may terminate in situation s, and $Trans(\delta, s, \delta', s')$, specifying that one step of program δ in situation s may lead to situation s' with δ' remaining to be executed. We can call *configuration* a pair (δ, s) with program δ and situation s. $Trans$ denotes one-step transitions between configurations and $Final$ denotes that a configuration may terminate.

For *agent actions*, we have $Final(A(\vec{x}, s) \equiv False$ as usual, while

$$\begin{array}{l} Trans(A(\vec{x}), s, \delta', s') \equiv \\ \qquad \exists e.Poss(A(\vec{x}, e), s) \land \delta' = \epsilon \land s' = do(A(\vec{x}, e), s). \end{array}$$

This reflects the fact that agent actions are nondeterministic and their outcome depends on the environment reaction.

[5] This results in a *synchronous test* construct that does not allow interleaving (every transition involves the execution of an action).

The definitions of *Trans* and *Final* for the other ConGolog constructs are as in (De Giacomo *et al.* 2010):

$$
\begin{array}{l}
\mathit{Final}(\phi?, s) \equiv \phi[s] \\
\mathit{Final}(\delta_1; \delta_2, s) \equiv \mathit{Final}(\delta_1, s) \wedge \mathit{Final}(\delta_2, s) \\
\mathit{Final}(\delta_1 | \delta_2, s) \equiv \mathit{Final}(\delta_1, s) \vee \mathit{Final}(\delta_2, s) \\
\mathit{Final}(\pi x.\delta, s) \equiv \exists x.\mathit{Final}(\delta, s) \\
\mathit{Final}(\delta^*, s) \equiv \mathit{True} \\
\mathit{Final}(\delta_1 \| \delta_2, s) \equiv \mathit{Final}(\delta_1, s) \wedge \mathit{Final}(\delta_2, s)
\end{array}
$$

$$
\begin{array}{l}
\mathit{Trans}(\phi?, s, \delta', s') \equiv \mathit{False} \\
\mathit{Trans}(\delta_1; \delta_2, s, \delta', s') \equiv \\
\qquad \mathit{Trans}(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1'; \delta_2 \vee \\
\qquad \mathit{Final}(\delta_1, s) \wedge \mathit{Trans}(\delta_2, s, \delta', s') \\
\mathit{Trans}(\delta_1 \mid \delta_2, s, \delta', s') \equiv \\
\qquad \mathit{Trans}(\delta_1, s, \delta', s') \vee \mathit{Trans}(\delta_2, s, \delta', s') \\
\mathit{Trans}(\pi x.\delta, s, \delta', s') \equiv \exists x.\mathit{Trans}(\delta, s, \delta', s') \\
\mathit{Trans}(\delta^*, s, \delta', s') \equiv \mathit{Trans}(\delta, s, \delta'', s') \wedge \delta' = \delta''; \delta^* \\
\mathit{Trans}(\delta_1 \| \delta_2, s, \delta', s') \equiv \\
\qquad \mathit{Trans}(\delta_1, s, \delta_1', s') \wedge \delta' = \delta_1' \| \delta_2 \vee \\
\qquad \mathit{Trans}(\delta_2, s, \delta_2', s') \wedge \delta' = \delta_1 \| \delta_2'
\end{array}
$$

Using *Trans* and *Final*, we can define $Do(\delta, s, s')$, which says that the *complete execution* of the program δ from s may result in the new situation s' (Levesque *et al.* 1997; De Giacomo *et al.* 2000). Formally $Do(\delta, s, s') \doteq \exists \delta'.\ \mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Final}(\delta', s')$, where Trans^* denotes the reflexive transitive closure (defined by induction) of the one-step transition *Trans*, i.e., $\mathit{Trans}^*(\delta, s, \delta', s')$ means that there exists a sequence of one-step transitions taking the configuration (δ, s) into the configuration (δ', s').

Note that in $Do(\delta, s, s')$, we can think of s' as describing a sequence of actions from s that correspond to a nondeterministically chosen execution of the program δ in s. When one uses a standard BAT where actions are deterministic, s' can also be considered as implicitly representing a strategy (c.f., previous section) for the agent to execute the program. However when $Do(\delta, s, s')$ is used with an NDBAT, we lose the interpretation of s' as an agent strategy, since we need to consider the nondeterministic environment reactions, which are implicitly assumed to be chosen by the environment *cooperatively* with the agent so as to arrive to a successful terminating execution of the program.

It is of great interest to define a version of *Do* that separates the nondeterminism in the program due to the agent choices and can be considered angelic, from the nondeterminism due to environment choices, which under skeptical reasoning should be considered adversarial. For this reason, we define an adversarial version of *Do* that maintains the angelic nondeterminism for the agent, but assumes a devilish (adversarial) nondeterminism for the environment. Such a definition requires the explicit use of agent strategies already introduced in the previous section. Specifically we introduce a predicate $\mathit{AgtCanForceBy}(\delta, s, f)$ that states that the strategy f, a function from situation to agent action (including the special action `stop`), executes the nondeterministic program δ considering its nondeterminism angelic, as in the standard *Do*, but also considering the nondeterminism of environment reactions devilish (i.e., adversarial). Formally, we define $\mathit{AgtCanForceBy}(\delta, s, f)$ inductively:

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

Crucially in the above, the choice of δ' is considered angelic, i.e., in favor of the agent, and quantified *existentially*, while instead the choice of the environment reaction e is considered devilish, i.e., adversarial, and quantified *universally*.

To fully grasp this definition, let's suppose for a moment that the program δ is *situation determined* (De Giacomo *et al.* 2012), that is:

$$
\begin{array}{l}
\mathit{SituationDetermined}(\delta, s) \doteq \forall s', \delta', \delta''. \\
\qquad \mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Trans}^*(\delta, s, \delta'', s') \supset \delta' = \delta''
\end{array}
$$

Then, given a configuration (δ, s) and a new situation $do(A(\vec{t}, e), s)$, there will be only one choice for δ' such that $\mathit{Trans}(\delta, s, \delta', do(A(\vec{t}, e), s))$ in the definition above.

If instead the program is not situation determined, there could be many possible δ'. Notice that the agent does not choose any of them explicitly, but the existential quantification in $\mathit{AgtCanForceBy}(\delta, s, f)$ will be resolved in favor of actually making the formula true, in this way guaranteeing that the strategy f leads to termination. In the original paper on Golog, where *Do* was first introduced (Levesque *et al.* 1997), one of the key feature of the language is angelic nondeterminism. Quoting from that paper: "*The GOLOG programmer can define complex action schemas-advice to a robot about how to achieve certain effects–without specifying in detail how to perform these actions. It becomes the theorem prover's responsibility (i.e., the interpreter's responsibility, ndr) to figure out one or more detailed executable sequences of primitive actions which will achieve the desired effects.*" Here we are maintaining this angelic nondeterminism, though now the strategy cannot be simply a sequence of actions, since we are executing our program in a nondeterministic (i.e., adversarial) environment.

We observe that if we want the agent to control also the remaining program in the transitions, we need a second strategy function g, such that that given the current program δ and the next situation $do(A(\vec{t}, e), s)$, as determined by the current situation s, the agent action $A(\vec{t})$, and the environment reaction e, returns the remaining program after the transition δ'. We can define a new version of *AgtCanForceBy* that also takes g as an argument:

$$
\mathit{AgtCanForceBy}(\delta, s, f, g) \doteq \forall P.[\ \ldots\ \supset P(\delta, s)]
$$

where . . . stands for

$$\begin{array}{l} [(f(s) = \texttt{stop} \land \mathit{Final}(\delta, s)) \supset P(\delta, s)] \land \\ [\exists A.\exists \vec{t}.(f(s) = A(\vec{t}) \neq \texttt{stop} \land \\ \exists e.\exists \delta'.\mathit{Trans}(\delta, s, \delta', do(A(\vec{t}, e), s)) \land \\ \forall e.(\exists \delta'.\mathit{Trans}(\delta, s, \delta', do(A(\vec{t}, e), s)) \supset \\ \qquad \exists \delta'.g(\delta, do(A(\vec{t}, e), s)) = \delta' \land \\ \qquad \mathit{Trans}(\delta, s, \delta', do(A(\vec{t}, e), s)) \land P(\delta', do(A(\vec{t}, e), s)))) \\ \supset P(\delta, s)] \end{array}$$

Finally, note that in a setting of complete information (i.e., we have complete information about the initial state), we can check for the existence of such strategies by:

$$\mathit{AgtCanForce}(\delta, s) \doteq \forall P.[\ \dots\ \supset P(\delta, s)]$$

where . . . stands for

$$\begin{array}{l} [\mathit{Final}(\delta, s)) \supset P(\delta, s)] \land \\ [\exists A.\exists \vec{x}.(\exists e.\exists \delta' \mathit{Trans}(\delta, s, \delta', do(A(\vec{x}, e), s))) \land \\ \forall e.(\exists \delta'.\mathit{Trans}(\delta, s, \delta', do(A(\vec{x}, e), s)) \supset \\ \ \exists \delta'.\mathit{Trans}(\delta, s, \delta', do(A(\vec{x}, e), s)) \land P(\delta', do(A(\vec{x}, e), s))) \\ \supset P(\delta, s)] \end{array}$$

The strategy itself here is implicit in the existential choices.

Example 7. *Continuing with our running example, we have:*

$$\begin{array}{l} \mathcal{D}_{trk} \models \mathit{AgtCanForceBy}(\delta_1, S_0, f) \\ \textit{where } \delta_1 \doteq \textbf{while}\ \mathit{pos} < \mathit{Dest}\ \textbf{do}\ \mathit{fwd}; \\ \qquad\qquad \textbf{if}\ \mathit{pos} > \mathit{Dest}\ \textbf{then}\ \mathit{bk}1\ \textbf{else}\ \epsilon \\ \textit{and } f(s) = \begin{cases} \mathit{fwd} & \textit{if } \mathit{pos}(s) < \mathit{Dest}, \\ \mathit{bk}1 & \textit{if } \mathit{pos}(s) > \mathit{Dest}, \\ \mathit{stop} & \textit{if } \mathit{pos}(s) = \mathit{Dest} \end{cases} \end{array}$$

Here δ_1 is a program that fully specifies the strategy, i.e., do fwd until at or beyond the destination and then do $bk1$ once if beyond. We can also show that

$$\begin{array}{l} \mathcal{D}_{trk} \models \mathit{AgtCanForceBy}(\delta_2, S_0, f) \\ \textit{where } \delta_2 \doteq (\mathit{fwd}|\mathit{bk}1)^*; \mathit{pos}(s) = \mathit{Dest}? \end{array}$$

Here δ_2 is a very general program that repeatedly does fwd or $bk1$ and may stop when the agent is at the destination. Note that δ_2 is situation-determined. We also have that

$$\begin{array}{l} \mathcal{D}_{trk} \models \mathit{Dest} > 0 \supset \mathit{AgtCanForceBy}(\delta_3, S_0, f) \\ \textit{where } \delta_3 \doteq \mathit{fwd}^*; \mathit{fwd}; \\ \qquad\qquad \textbf{if}\ \mathit{pos} > \mathit{Dest}\ \textbf{then}\ \mathit{bk}1\ \textbf{else}\ \epsilon \end{array}$$

δ_3 is not situation-determined and the executor has to choose when to do the last fwd.

Let us focus for a moment on the case where the environment is deterministic in its responses, i.e.:

$$\begin{array}{l} \mathit{Poss}_{ag}(A(\vec{x}), s) \supset \\ \quad \mathit{Poss}(A(\vec{x}, e), s) \land \mathit{Poss}(A(\vec{x}, e'), s) \supset e = e' \end{array}$$

Note that we already have $\mathit{Poss}_{ag}(A(\vec{x}), s) \supset \exists e.\mathit{Poss}(A(\vec{x}, e), s)$ by the reaction existence requirements. Let $\mathcal{C}$ be our ConGolog axioms and definitions plus the representation of programs as terms as in (De Giacomo *et al.* 2000). Then we have

Theorem 3. *Let $\mathcal{D}$ be an NDBAT be such that the environment is deterministic in its responses. Then*

$$\mathcal{D} \cup \mathcal{C} \models \mathit{AgtCanForce}(\delta, s) \equiv \exists s'.\mathit{Do}(\delta, s, s')$$

Proof. (Sketch) If the the environment is deterministic in its responses, then we can substitute the universal quantification over e in $\mathit{AgtCanForce}(\delta, s)$ by an existential, and then check that the resulting formula is equivalent to $\exists \delta'.\ \mathit{Trans}^*(\delta, s, \delta', s') \land \mathit{Final}(\delta', s')$. □

This theorem shows that our $\mathit{AgtCanForce}(\delta, s)$, and hence $\mathit{AgtCanForce}(\delta, s, f)$, are extensions of $\exists s'.\mathit{Do}(\delta, s, s')$ and $\mathit{Do}(\delta, s, s')$ respectively.

In a setting with incomplete information the existential choices, and hence strategy implicit in the resulting situation s' in Do, as well as the strategy f in $\mathit{AgtCanForce}(\delta, s, f)$, would depend on the model. In both cases, in different models the strategy would be different in general. Hence the agent would need to know in which model she is, in order to adopt the correct strategy. If we want instead a strategy the works for all models, then in a deterministic environment we would need to find a ground system action sequence $\vec{a}$ such that $\mathcal{D} \cup \mathcal{C} \models \mathit{Do}(\delta, S_0, do(\vec{a}, S_0))$, and in our general setting find a ground strategy f such that $\mathcal{D} \cup \mathcal{C} \models \mathit{AgtCanForceBy}(\delta, S_0, f)$ (or f and g such that $\mathcal{D} \cup \mathcal{C} \models \mathit{AgtCanForceBy}(\delta, S_0, f)$). In both cases, this is difficult since it requires second-order quantification over functions to quantify over strategies.

7 Conclusion

In this paper, we have presented a nondeterministic variant of the situation calculus. We started by separating the agent's actuation of the action from the environment's determination of the action's outcome. We modeled this by adding to every action type/function an additional parameter that captures the environment's reaction to the action. In this way, we retain a notion of "system action" formed by the agent's action together with the environment's reaction, as in the standard situation calculus. In addition, we gain the ability to quantify separately over the agent's action and the environment's reaction. Thus notably, we can capture the *angelic nondeterminism* of the agent's choosing actions through existential quantification, and the *devilish nondeterminism* of the environment's response (which is not controlled by the agent), though universal quantification. We have also shown that we can capture planning in nondeterministic domains, as well as the original idea of having Golog/ConGolog programs specify program sketches, with angelic nondeterminism to be resolved by the agent as needed. In particular, we make this powerful idea work seamlessly in our nondeterministic situation calculus.

We note that predicates that define planning in nondeterministic domains, as well as nondeterministic program execution, must be expressed in second order logic, thus making automated reasoning challenging (see (Reiter 2001) for some approaches without termination guarantees). But, it should be noted that all these formulas are in the end fixpoint-formulas. So if we consider the propositional case, or if we consider domains with a finitely many objects (and environment reactions based on these objects) as in (Ternovskaia 1999), then all these properties can be reduced to Mu-Calculus formulas over situations (De Giacomo *et al.* 2016a) or over configurations formed by pairs of programs and situations (De Giacomo *et al.* 2016b)). This means that checking these properties is decidable, has it is decidable to synthesize strategies/plans to fulfill them. Moreover, reasoning remains decidable when the NDBAT admits infinitely many objects, but is state-bounded (De Giacomo *et al.* 2016a; Calvanese *et al.* 2018; De Giacomo *et al.* 2021).

Acknowledgements

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.

A Proof of Theorem 1

Theorem 1. Let $\mathcal{D}$ be an NDBAT, σ a ground agent action sequence, s a situation term containing no situation or action variables, and ϕ a situation-suppressed formula containing no action variables. Then $\mathit{CertainlyExecutable}(\sigma, s)$ is equivalent to a regressable formula, and so are $\mathit{PossiblyExecutable}(\sigma, s)$, $\mathit{CertainlyAfter}(\sigma, \phi, s)$, $\mathit{PossiblyAfter}(\sigma, \phi, s)$, and $\mathit{ForcesAfter}(\sigma, \phi, s)$.

Proof: Assume that σ is ground agent action sequence and s a situation term containing no situation or action variables.

First, we show that $\mathit{CertainlyExecutable}(\sigma, s)$ is equivalent to a regressable formula by induction on the length of σ. Base case: $\mathcal{D} \models \mathit{CertainlyExecutable}(\epsilon, s) \equiv \mathit{True}$, which is regressable. For the general case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{CertainlyExecutable}([A(\vec{x}), \sigma], s)\\
&\quad \equiv \mathit{Poss}_{ag}(A(\vec{x}), s) \wedge\\
&\quad\quad \forall s'.\mathit{Do}_{ag}(A(\vec{x}), s, s') \supset \mathit{CertainlyExecutable}(\sigma, s')\\
&\quad \equiv \mathit{Poss}_{ag}(A(\vec{x}), s) \wedge\\
&\quad\quad \forall e.(\mathit{Poss}(A(\vec{x}, e), s) \supset\\
&\quad\quad\quad\quad \mathit{CertainlyExecutable}(\sigma, do(A(\vec{x}, e), s)))\\
&\quad \equiv \exists e.\phi_A^{Poss}(\vec{x}, e, s) \wedge \forall e.(\phi_A^{Poss}(\vec{x}, e, s) \supset\\
&\quad\quad\quad\quad\quad \mathit{CertainlyExecutable}(\sigma, do(A(\vec{x}, e), s)))
\end{aligned}$$

where $\phi_A^{Poss}(\vec{x}, e, s)$ is the right-hand-side of the action precondition axiom for A and is regressable when s contains no situation or action variables. The result follows by the induction hypothesis.

Secondly, we show that $\mathit{PossiblyExecutable}(\sigma, s)$ is equivalent to a regressable formula by induction on the length of σ. Base case: $\mathcal{D} \models \mathit{PossiblyExecutable}(\epsilon, s) \equiv \mathit{True}$, which is regressable. For the general case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{PossiblyExecutable}([A(\vec{x}), \sigma], s)\\
&\quad \equiv \exists s'.\mathit{Do}_{ag}(A(\vec{x}), s, s') \wedge \mathit{PossiblyExecutable}(\sigma, s')\\
&\quad \equiv \exists e.\mathit{Poss}(A(\vec{x}, e), s) \wedge\\
&\quad\quad\quad \mathit{PossiblyExecutable}(\sigma, do(A(\vec{x}, e), s))\\
&\quad \equiv \exists e.\phi_A^{Poss}(\vec{x}, e, s) \wedge\\
&\quad\quad\quad \mathit{PossiblyExecutable}(\sigma, do(A(\vec{x}, e), s))
\end{aligned}$$

where $\phi_A^{Poss}(\vec{x}, e, s)$ is the right-hand-side of the action precondition axiom for A and is regressable when s contains no situation or action variables. The result follows by the induction hypothesis.

Thirdly, we show that $\mathit{CertainlyAfter}(\sigma, \phi, s)$ is equivalent to a regressable formula by induction on the length of σ. For the base case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{CertainlyAfter}(\epsilon, \phi, s)\\
&\quad \equiv \forall s'.\mathit{Do}_{ag}(\epsilon, s, s') \supset \phi[s']\\
&\quad \equiv \forall s'.s' = s \supset \phi[s']\\
&\quad \equiv \phi[s]
\end{aligned}$$

The latter is regressable given our assumptions about s. For the general case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{CertainlyAfter}([A(\vec{x}), \sigma], \phi, s)\\
&\quad \equiv \forall s'.\mathit{Do}_{ag}([A(\vec{x}), \sigma], s, s') \supset \phi[s']\\
&\quad \equiv \forall s''.\mathit{Do}_{ag}(A(\vec{x}), s, s'')\\
&\quad\quad \supset \forall s' \mathit{Do}_{ag}(\sigma, s'', s') \supset \phi[s']\\
&\quad \equiv \forall s'.\mathit{Do}_{ag}(A(\vec{x}), s, s') \supset \mathit{CertainlyAfter}(\sigma, \phi, s')\\
&\quad \equiv \forall e.\mathit{Poss}(A(\vec{x}), e), s) \supset\\
&\quad\quad\quad \mathit{CertainlyAfter}(\sigma, \phi, do(A(\vec{x}, e), s))\\
&\quad \equiv \forall e.\phi_A^{Poss}(\vec{x}, e, s) \supset\\
&\quad\quad\quad \mathit{CertainlyAfter}(\sigma, \phi, do(A(\vec{x}, e), s))
\end{aligned}$$

where $\phi_A^{Poss}(\vec{x}, e, s)$ is the right-hand-side of the action precondition axiom for A and is regressable when s contains no situation or action variables. The result follows by the induction hypothesis.

Fourthly, we show that $\mathit{PossiblyAfter}(\sigma, \phi, s)$ is equivalent to a regressable formula by induction on the length of σ. For the base case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{PossiblyAfter}(\epsilon, \phi, s)\\
&\quad \equiv \exists s'.\mathit{Do}_{ag}(\epsilon, s, s') \wedge \phi[s']\\
&\quad \equiv \exists s'.s' = s \wedge \phi[s']\\
&\quad \equiv \phi[s]
\end{aligned}$$

The latter is regressable given our assumptions about s. For the general case, we have:

$$\begin{aligned}
&\mathcal{D} \models \mathit{PossiblyAfter}([A(\vec{x}), \sigma], \phi, s)\\
&\quad \equiv \exists s'.\mathit{Do}_{ag}([A(\vec{x}), \sigma], s, s') \wedge \phi[s']\\
&\quad \equiv \exists s''.\mathit{Do}_{ag}(A(\vec{x}), s, s'')\\
&\quad\quad \wedge \exists s'.\mathit{Do}_{ag}(\sigma, s'', s') \wedge \phi[s']\\
&\quad \equiv \exists s'.\mathit{Do}_{ag}(A(\vec{x}), s, s') \wedge \mathit{PossiblyAfter}(\sigma, \phi, s')\\
&\quad \equiv \exists e.\mathit{Poss}(A(\vec{x}), e), s) \wedge\\
&\quad\quad\quad \mathit{PossiblyAfter}(\sigma, \phi, do(A(\vec{x}, e), s))\\
&\quad \equiv \exists e.\phi_A^{Poss}(\vec{x}, e, s) \wedge\\
&\quad\quad\quad \mathit{PossiblyAfter}(\sigma, \phi, do(A(\vec{x}, e), s))
\end{aligned}$$

where $\phi_A^{Poss}(\vec{x}, e, s)$ is the right-hand-side of the action precondition axiom for A and is regressable when s contains no situation or action variables. The result follows by the induction hypothesis.

Finally, $\mathit{ForcesAfter}(\sigma, \phi, s)$ is defined as $\mathit{CertainlyAfter}(\sigma, \phi, s) \wedge \mathit{CertainlyExecutable}(\sigma, s)$. It follows immediately by our first and third results above that is $\mathit{ForcesAfter}(\sigma, \phi, s)$ equivalent to a regressable formula under the assumptions. □

References

Benjamin Aminof, Giuseppe De Giacomo, and Sasha Rubin. Stochastic fairness and language-theoretic fairness in planning in nondeterministic domains. In *ICAPS*, pages 20–28. AAAI Press, 2020.

Fahiem Bacchus, Joseph Y. Halpern, and Hector J. Levesque. Reasoning about noisy sensors in the situation calculus. In *IJCAI*, pages 1933–1940, 1995.

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.

Vaishak Belle and Hector J. Levesque. Regression and progression in stochastic domains. *Artif. Intell.*, 281:103247, 2020.

Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *IJCAI*, pages 4667–4673. ijcai.org, 2018.

Manfred Broy and Martin Wirsing. On the algebraic specification of nondeterministic programming languages. In *CAAP*, volume 112 of *LNCS*, pages 162–179. Springer, 1981.

Diego Calvanese, Giuseppe De Giacomo, Marco Montali, and Fabio Patrizi. First-order μ-calculus over generic transition systems and applications to the situation calculus. *Inf. Comput.*, 259(3):328–347, 2018.

Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning in non-deterministic domains via model checking. In *AIPS*, pages 36–43. AAAI, 1998.

Jens Claßen and Gerhard Lakemeyer. A logic for non-terminating Golog programs. In *KR*, pages 589–599, 2008.

Jens Claßen, Yuxiao Hu, and Gerhard Lakemeyer. A situation-calculus semantics for an expressive fragment of PDDL. In *AAAI*, pages 956–961. AAAI Press, 2007.

Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1–2):109–169, 2000.

Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *KR*, 2010.

Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise. On supervising agents in situation-determined ConGolog. In *AAMAS*, pages 1031–1038. IFAAMAS, 2012.

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

Giuseppe De Giacomo, Yves Lespérance, Fabio Patrizi, and Sebastian Sardiña. Verifying ConGolog programs on bounded situation calculus theories. In *AAAI*, pages 950–956. AAAI Press, 2016.

Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus game structures and GDL. In *ECAI*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 408–416. IOS Press, 2016.

Giuseppe De Giacomo, Eugenia Ternovska, and Ray Reiter. Non-terminating processes in the situation calculus. *Ann. Math. Artif. Intell.*, 88(5-6):623–640, 2020.

Giuseppe De Giacomo, Paolo Felli, Brian Logan, Fabio Patrizi, and Sebastian Sardina. Situation calculus for controller synthesis in manufacturing systems with first-order state representation. *Artif. Intell.*, 2021. To appear.

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.

H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool, 2013.

Tomas Geffner and Hector Geffner. Compact policies for fully observable non-deterministic planning as SAT. In *ICAPS*, 2018.

Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool, 2019.

C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

Hector J. Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

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

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.

Fiora Pirri and Raymond Reiter. Some contributions to the metatheory of the situation calculus. *J. ACM*, 46(3):325–361, 1999.

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

Sebastian Sardina, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the semantics of deliberation in IndiGolog – From theory to implementation. *Ann. Math. Artif. Intell.*, 41(2–4):259–299, August 2004.

Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the limits of planning over belief states under strict uncertainty. In *KR*, pages 463–471. AAAI Press, 2006.

Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artif. Intell.*, 175(2):615–647, 2011.

Eugenia Ternovskaia. Automata theory for reasoning about actions. In *IJCAI*, pages 153–159. Morgan Kaufmann, 1999.