Strategic Reasoning over Golog Programs in the Nondeterministic Situation Calculus – Extended Abstract

Giuseppe De Giacomo[1,2], Yves Lespérance[3] and Matteo Mancanelli[2,*]

[1]University of Oxford, Oxford, UK
[3]York University, Toronto, ON, Canada
[2]University of Rome La Sapienza, Rome, Italy

Abstract

Automata-based synthesis has seen increasing interest in the last decade, mostly focused on declarative specifications. Here, we consider procedural programs that specify high-level agent behaviors over nondeterministic domains. Specifically, we tackle the problem of synthesizing strategies that guarantee the successful execution of a Golog program δ within a nondeterministic basic action theory (NDBAT). Our approach constructs symbolic program graphs that capture the control flow of δ independently of the domain, enabling reactive synthesis via their cross product with the domain model. We formally relate graph-based transitions to standard Golog semantics and show how our framework modularly separates agent and environment behaviors. This flexibility supports strategic reasoning and synthesis in complex nondeterministic settings, with programs acting as independent controllers for both agent and environment.

Keywords

Program Execution, Golog, Situation Calculus, Nondeterministic Domains, Strategic Reasoning, Reactive Synthesis

1. Overview

Proposed Framework. The problem we address in this paper is a variant of the *high-level program execution task*, that is, given a program δ and an action theory $\mathcal{D}$, find a strategy for executing δ that guarantees successful termination from the initial situation S_0 in the domain specified by $\mathcal{D}$. If the domain is deterministic and the program is situation determined (i.e., the remaining subprogram is uniquely fixed by the resulting situation), then such a strategy can simply specify a sequence of actions $\vec{a}$ such that $\mathcal{D} \models Do(\delta, S_0, do(\vec{a}, S_0))$. If the program is not situation determined, the strategy must also specify the remaining program after each step. In this paper, we consider the case where the domain is nondeterministic and specified by a NDBAT. For a situation determined program δ, we aim to synthesize a strategy f (a mapping from situations to actions) such that $\mathcal{D} \models AgtCanForce(\delta, S_0, f)$ holds. When δ is not situation determined, we also generate a strategy g mapping situations to subprograms, such that $\mathcal{D} \models AgtCanForce(\delta, S_0, f, g)$. The same applies at the model level, as in FOND planning, when full information about the initial state is available, i.e., for a model M such that $M \models \mathcal{D}$.

Most prior work on Golog program execution [1, 2, 3] focuses on deterministic environments, compiling Golog into the domain so classical planners can be used. We instead target nondeterministic domains using a reactive synthesis approach, building on recent advances. A recent example is [4], which uses the C^2 decidable fragment of FOL but suffers from scalability issues. Our framework is simpler and more intuitive, while offering formal guarantees that relate program executions with classical Golog semantics and synthesis techniques. Our framework is also very flexible, and it can easily be extended to accommodate different challenges. To show this, in the last section we allow one to specify separate programs as constraints on agent's behavior and environment's behavior only, while [4] takes the Golog program as a constraint on the behavior of the whole system, i.e., both the agent and environment.

7th International Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis (OVERLAY), October 26, 2025, Bologna, Italy

*Corresponding author.
giuseppe.degiacomo@cs.ox.ac.uk (G. De Giacomo); lesperan@eecs.yorku.ca (Y. Lespérance); mancanelli@diag.uniroma1.it (M. Mancanelli)

Preliminaries. Our approach builds on the *nondeterministic situation calculus* [5], where each agent action $A(\vec{x})$ is paired with an environment reaction e outside the agent's control, determining the outcome (e.g., a flipped coin may land heads or tails). A nondeterministic basic action theory (NDBAT) can be seen as a special kind of action theory, where we have *system* actions $A(\vec{x}, e)$, successor state axioms $\mathcal{D}_{ssa}$, describing how predicates and functions change after system actions are performed, and action precondition axioms $\mathcal{D}_{poss}$, stating when each system action can occur. Throughout the paper, we assume standard names for actions and objects of the domain, for which we impose the unique name assumption and domain closure. Golog programs are executed over a (possibly nondeterministic) BAT. We consider a variant of Golog where the test construct yields no transitions, and is final when the condition is satisfied [6, 7]. The key feature of our variant is that programs instruct agent actions, rather than system actions. Programs are of the form $\delta ::= a(\vec{t}) \mid \varphi? \mid \delta_1;\delta_2 \mid \delta_1|\delta_2 \mid \pi x.\delta \mid \delta^*$, where $a(\vec{t})$ is an action term, $\delta_1;\delta_2$ is the sequential execution of δ_1 and δ_2, $\delta_1|\delta_2$ is the nondeterministic choice of δ_1 and δ_2, $\pi x.\delta$ executes program δ for *some* nondeterministic choice of the object variable x, and δ^* performs δ zero or more times. φ is a situation-suppressed formula, and $\varphi[s]$ the formula obtained by restoring s into all fluents in φ. We use nil as an abbreviation for $\texttt{True}?$.

To construct the program graph of a Golog program δ_0, we define its subprograms, those encountered during partial execution, via the notion of *syntactic closure* [8]. They separate the program terms themselves from the assignments to pick variables. Let's assume all pick variables are renamed apart and ordered. We define a n-tuple of object terms $\vec{x} = \langle x_1, ..., x_n\rangle$, called *environment term*, where x_i is the current value of i-th pick variable. In the following, $\mathcal{O}^n$ denotes the Cartesian product of n objects, and x_z is the value assigned to variable z.. At the beginning of an execution, the environment term is arbitrarily instantiated, and at each step of the computation it maintains only n values. Tracking pick variables separately in $\vec{x}$ avoids enumerating all (possibly infinite) bindings from $\pi x.\delta_0$, enabling a finite set of subprograms. The syntactic closure Γ_{δ_0} is defined inductively: *(i)* $\delta_0, nil \in \Gamma_{\delta_0}$, *(ii)* if $\delta_1;\delta_2 \in \Gamma_{\delta_0}$ and $\delta_1' \in \Gamma_{\delta_1}$, then $\delta_1';\delta_2 \in \Gamma_{\delta_0}$ and $\Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$, *(iii)* if $\delta_1 \mid \delta_2 \in \Gamma_{\delta_0}$, then $\Gamma_{\delta_1}, \Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$, *(iv)* if $\pi z.\delta \in \Gamma_{\delta_0}$, then $\Gamma_\delta \subseteq \Gamma_{\delta_0}$, *(v)* if $\delta^* \in \Gamma_{\delta_0}$, then $\delta;\delta^* \in \Gamma_{\delta_0}$.

A complete configuration is a triple $(\delta, \vec{x}, s)$, where $\delta \in \Gamma_{\delta_0}$, $\vec{x}$ is the environment term, and s is the current situation. The semantics of Golog is defined in terms of single-steps using two predicates: $Final$, indicating a program can terminate in a situation, and $Trans$, specifying one-step transitions to a new situation and remaining program. Here, we define *Trans* and *Final* considering both the specified configuration version and the environment reactions as follows:

$$
\begin{aligned}
&\mathit{Trans}(a, \vec{x}, s, \delta', \vec{x}', s') \equiv \exists e.Poss(a[\vec{x}](e), s) \land \\
&\quad \delta' = nil \land \vec{x}' = \vec{x} \land s' = do(a[\vec{x}](e), s)\\
&\mathit{Trans}(\varphi?, \vec{x}, s, \delta', \vec{x}', s') \equiv \texttt{False}\\
&\mathit{Trans}(\delta_1;\delta_2, \vec{x}, s, \delta', \vec{x}', s') \equiv \mathit{Trans}(\delta_1, \vec{x}, s, \delta_1', \vec{x}', s') \land \\
&\quad \delta' = \delta_1';\delta_2 \lor \mathit{Final}(\delta_1, \vec{x}, s) \land \mathit{Trans}(\delta_2, \vec{x}, s, \delta', \vec{x}', s')\\
&\mathit{Trans}(\delta_1|\delta_2, \vec{x}, s, \delta', \vec{x}', s') \equiv \mathit{Trans}(\delta_1, \vec{x}, s, \delta', \vec{x}', s') \lor \\
&\quad \mathit{Trans}(\delta_2, \vec{x}, s, \delta', \vec{x}', s')\\
&\mathit{Trans}(\pi z.\delta, \vec{x}, s, \delta', \vec{x}', s') \equiv \exists d.\mathit{Trans}(\delta, \vec{x}^z_d, s, \delta', \vec{x}', s')\\
&\mathit{Trans}(\delta^*, \vec{x}, s, \delta', \vec{x}', s') \equiv \mathit{Trans}(\delta, \vec{x}, s, \delta'', \vec{x}', s') \land \delta' = \delta'';\delta^*
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{Final}(a, \vec{x}, s) \equiv \texttt{False}\\
&\mathit{Final}(\varphi?, \vec{x}, s) \equiv \varphi[\vec{x}][s]\\
&\mathit{Final}(\delta_1;\delta_2, \vec{x}, s) \equiv \\
&\quad \mathit{Final}(\delta_1, \vec{x}, s) \land \mathit{Final}(\delta_2, \vec{x}, s)\\
&\mathit{Final}(\delta_1|\delta_2, \vec{x}, s) \equiv \\
&\quad \mathit{Final}(\delta_1, \vec{x}, s) \lor \mathit{Final}(\delta_2, \vec{x}, s)\\
&\mathit{Final}(\pi z.\delta, \vec{x}, s) \equiv \\
&\quad \exists d.\mathit{Final}(\delta, \vec{x}^z_d, s)\\
&\mathit{Final}(\delta^*, \vec{x}, s) \equiv \texttt{True}
\end{aligned}
$$

where $a[\vec{x}]$ and $\varphi[\vec{x}]$ denote the action term and formula with free pick variables replaced by corresponding terms in $\vec{x}$, and $\vec{x}^z_d$ denotes $\vec{x}$ updated by binding variable z to d. We use *Trans** to denote the transitive closure of *Trans*, i.e., *Trans*$^*(\delta, \vec{x}, s, \delta', \vec{x}', s')$ means that there exists a sequence of one-step transitions from $(\delta, \vec{x}, s)$ to $(\delta', \vec{x}', s')$. We define a program *situation determined* (SD) [9] if, at any times, the remaining program is uniquely determined by the resulting situation: $SituationDetermined(\delta, \vec{x}, s) \doteq \forall s', \delta', \delta'', \vec{x}', \vec{x}''.Trans^*(\delta, \vec{x}, s, \delta', \vec{x}', s') \land Trans^*(\delta, \vec{x}, s, \delta'', \vec{x}'', s') \supset \delta' = \delta'' \land \vec{x}' = \vec{x}''$. We can prove lemmas to provide guarantees that all programs that δ_0 can evolve into according to $Trans$ and $Final$, must be in its syntactic closure Γ_{δ_0}.

Lemma 1. *Let δ_0 be a Golog program and Γ_{δ_0} its syntactic closure. Let $\mathcal{D}$ an NDBAT over which δ_0 is executed and M a model of $\mathcal{D}$. If $\delta \in \Gamma_{\delta_0}$ and $M \models Trans(\delta, \vec{x}, s, \delta', \vec{x}', s')$, then $\delta' \in \Gamma_{\delta_0}$, and if $M \models Trans^*(\delta_0, \vec{x}_0, s_0, \delta, \vec{x}, s)$, then $\delta \in \Gamma_{\delta_0}$.*

2. First-Order Program Graphs

Constructing the Program Graph. The program graph of a Golog program δ_0 characterizes all its possible executions, independently of the domain (which is integrated later via a cross product). The program graph is thus a symbolic structure that captures the control flow semantics of Golog at the syntactic level, decoupled from domain dynamics. Each node corresponds to a program in the syntactic closure Γ_{δ_0}, i.e. a possible remaining subprogram, and each edge represents a guarded one-step transition between such nodes. We define transitions and termination conditions based on $Trans$ and $Final$. Since pick variables are unbound at this stage, the graph tracks which variables need to be bound, deferring the actual binding until domain details are available. Specifically, we define:

$$
\begin{array}{ll}
T(a,a) = \{(Poss(a), \emptyset, nil)\} & \\
T(a,b) = \{\} & F(a) = False \\
T(\varphi?, a) = \{\} & F(\varphi?) = \varphi \\
T(\delta_1;\delta_2, a) = \{(\neg F(\delta_1) \wedge \varphi, P, \delta_1';\delta_2) \mid (\varphi, P, \delta_1') \in T(\delta_1, a)\} \cup & F(\delta_1;\delta_2) = F(\delta_1) \wedge F(\delta_2) \\
\quad \{(F(\delta_1) \wedge \varphi, P, \delta_2') \mid (\varphi, P, \delta_2') \in T(\delta_2, a)\} & F(\delta_1|\delta_2) = F(\delta_1) \vee F(\delta_2) \\
T(\delta_1|\delta_2, a) = T(\delta_1, a) \cup T(\delta_2, a) & F(\pi z.\delta) = \exists z.F(\delta) \\
T(\pi z.\delta, a) = \{(\varphi, P \cup z, \delta') \mid (\varphi, P, \delta') \in T(\delta, a)\} & F(\delta^*) = True \\
T(\delta^*, a) = \{(\neg F(\delta) \wedge \varphi, P, \delta';\delta^*) \mid (\varphi, P, \delta') \in T(\delta, a)\} &
\end{array}
$$

Given a program δ and an action a, $T(\delta, a)$ returns guarded transitions (φ, P, δ'), where φ is a Boolean guard over $Poss(a)$ and tests in δ, P is the set of pick variables to be instantiated, and δ' is the remaining program. $F(\delta)$ gives the condition under which δ may terminate.

Now, we can provide a definition for the program graph.

Definition 2. *Let Φ be a boolean formula over tests and $Poss$, V the set of pick variables and $\mathcal{A}$ the set of agent actions. If δ_0 is a Golog program, the program graph $\mathcal{G}$ is a tuple $\langle \Phi \times 2^V \times \mathcal{A}, Q, q_0, \tau, \mathcal{L} \rangle$, where*

- $\Phi \times 2^V \times \mathcal{A}$ *is the alphabet*
- $Q = \Gamma_{\delta_0}$ *is the set of nodes*
- $q_0 = \delta_0$ *is the initial node*
- $\tau(q, \varphi, P, a, q')$ *iff* $(\varphi, P, q') \in T(q, a)$
- $\mathcal{L}(q) = F(q)$ *indicates that the state q is assigned a label according to F*

Results about Program Graphs We now summarize key properties of the program graph used to symbolically encode the structure of Golog programs. First, we can talk about the dimensions:

Theorem 3. *Let δ_0 be a Golog program and $\mathcal{G}$ the corresponding program graph. The number of nodes and edges in $\mathcal{G}$ is linear in the size of δ_0.*

This theorem ensures the graph remains compact wrt the original dimension of the program.

We also relate program graphs with Golog operational semantics. In particular, symbolic transitions in the graph correspond to executable transitions in the model, and vice versa, provided guard conditions are satisfied, and the final condition $F(\delta)$ characterizes the same terminating configurations as $Final$.

Theorem 4. *Let δ_0 be a Golog program and $\mathcal{G}$ the corresponding program graph. Let $\mathcal{D}$ an NDBAT over which δ_0 is executed and M a model of $\mathcal{D}$. Then, for all subprograms $\delta, \delta' \in \Gamma_{\delta_0}$, environment terms $\vec{x}, \vec{x}'$, agent action a, and situation s:*

1. *If $\tau(\delta, \varphi, P, a, \delta') \in \mathcal{G}, \forall z \notin P.x_z' = x_z$ and $M \models \varphi[\vec{x}'][s]$, then there exists a reaction e such that $M \models Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s))$;*
2. *For every reaction e, if $M \models Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s))$, then there exists φ such that $\tau(\delta, \varphi, P, a, \delta') \in \mathcal{G}, \forall z \notin P.x_z' = x_z$, and $M \models \varphi[\vec{x}'][s]$.*

Theorem 5. *For all subprograms $\delta \in \Gamma_{\delta_0}$, environment term $\vec{x}$ and situation s, $M \models Final(\delta, \vec{x}, s)$ iff $M \models F(\delta)[\vec{x}][s]$.*

Having the previous theorems in place, we can prove that a full execution trace to a final configuration exists in the model if and only if there is a corresponding path in the program graph.

Theorem 6. *Let $(\delta_0, \vec{x}_0, s_0)$ be an initial configuration. Then, $M \models Trans^*(\delta_0, \vec{x}_0, s_0, \delta, \vec{x}, s) \wedge Final(\delta, \vec{x}, s)$ iff there exists a sequence of transitions $\tau(q_0, \phi_1, P_1, a_1, q_1)$, $\tau(q_1, \phi_2, P_2, a_2, q_2), \ldots$, $\tau(q_{n-1}, \phi_n, P_n, a_n, q_n)$ in $\mathcal{G}$, environment terms $x_0, \ldots, x_n$ and reactions $e_1, \ldots, e_n$ such that $q_0 = \delta_0$, $q_n = \delta$, $\vec{x}_n = \vec{x}$ and $s = s_n$, and for each $i = 1, \ldots, n$,* (i) $\forall z \notin P_i, x'_{i,z} = x_{i,z}$, (ii) $s_i = do(a_i[x_i](e_i), s_{i-1})$, (iii) $M \models \phi_i[x_i][s_{i-1}]$, *and* (iv) $M \models F(q_n)[x_n][s_n]$.

Finally, if we have a SD program, then the symbolic transition relation becomes functional.

Theorem 7. *Let δ_0 be a SD program in S_0 wrt $\mathcal{D}$. For any transitions $\tau(q, \varphi_1, P, a, q'_1)$ and $\tau(q, \varphi_2, P, a, q'_2)$, there exists $\vec{x}$, $\vec{x}'$, s s.t. $\forall z \notin P.x'_z = x_z$ and $M \models \varphi_1[\vec{x}'][s] \wedge \varphi_2[\vec{x}'][s]$, then $q'_1 = q'_2$.*

Thus, when δ_0 is situation determined in S_0 with respect to $\mathcal{D}$, the characteristic graph becomes deterministic, and we can replace the relation $\tau(q, \varphi, P, a, q')$ by the function $\tau(q, \varphi, P, a) = q'$.

3. Synthesis and Reasoning over Program Graphs.

TS-based Synthesis for FO Domains Consider a nondeterministic domain $\mathcal{D}_M = \langle 2^{\mathcal{F}}, \mathcal{A}, s_0, \rho, \alpha \rangle$, where $\mathcal{F}$ is the set of fluents, $\mathcal{A}$ the agent actions, $2^{\mathcal{F}}$ the state space, s_0 the initial state, $\alpha(s) \subseteq \mathcal{A}$ the action preconditions, and $\rho(s, a, s')$ the transition relation for $a \in \alpha(s)$. For any NDBAT $\mathcal{D}$ and model $M \models \mathcal{D}$, there exists a corresponding domain $\mathcal{D}_M$. We construct a game arena by taking the cross product of the program graph $\mathcal{G}$ and domain $\mathcal{D}_M$:

Definition 8. *Let δ_0 be a program and $\mathcal{D}_M$ a ND domain. The cross product is a tuple $\langle \mathcal{A} \times Q \times \mathcal{O}^n \times 2^{\mathcal{F}}, Q \times \mathcal{O}^n \times 2^{\mathcal{F}}, (\delta_0, \vec{x}_0, s_0), Tr, Fin \rangle$, where*

- $\mathcal{A} \times Q \times \mathcal{O}^n \times 2^{\mathcal{F}}$ *is the alphabet*
- $Q \times \mathcal{O}^n \times 2^{\mathcal{F}}$ *is a set of states*
- $(\delta_0, \vec{x}_0, s_0)$ *is the initial state*
- $Tr((\delta, \vec{x}, s), (a, \delta', \vec{x}^P, s')) = (\delta', \vec{x}', s')$ *is the transition function where* (i) $\exists \varphi.\tau(\delta, \varphi, P, a, \delta')$, (ii) $\forall z \notin P.x'_z = x_z$ *and* $\forall z \in P.x'_z = x^P_z$, (iii) $s \models \varphi[\vec{x}']$, (iv) $\rho(s, a[\vec{x}'], s')$
- $Fin = \{(\delta, \vec{x}, s) \mid s \models F(\delta)[\vec{x}]\}$ *is the set of final states*

This can be viewed as a game arena where the agent controls the action $\mathcal{A}$, the remaining program Q, and the binding of the pick variables $\mathcal{O}^n$, and where the environment controls the next state $2^{\mathcal{F}}$. A game strategy is a function $\kappa : G \to \mathcal{A} \times Q \times \mathcal{O}^n$ mapping states of the game $g \in G$ to agent actions, remaining programs, and bindings for the pick variables. The set of plays induced by a game strategy κ in a game arena A, $Play(\kappa, A)$, is the set of all plays $g_0, g_1, \ldots \in G^\omega$ such that g_0 is the initial state of A and there exists an environment state s' such that $Tr(g_i, (\kappa(g_i), s')) = g_{i+1}$. A game strategy κ is winning in A if, for every play $g_0, g_1, \ldots$ in $Play(\kappa, A)$, there exists some i such that $Fin(g_i)$.

Agent Control and Strategic Reasoning. To represent the ability of the agent to execute an agent program in a ND domain, [5] introduce $AgtCanForceIf(\delta, f, s)$ as an adversarial version of Do in the presence of environment reactions. It states that strategy f, a function from situations to agent actions (including the special action $stop$), executes SD Golog 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. Here is the version of $AgtCanForceIf$ that considers the presence of environment term:

$$AgtCanForceIf(\delta, \vec{x}, f, s) \doteq \forall P.[\ldots \supset P(\delta, \vec{x}, s)]$$
where … stands for
$$[(\exists \vec{x}.f(s) = stop \wedge Final(\delta, \vec{x}, s)) \supset P(\delta, \vec{x}, s)] \wedge$$
$$[\exists a.(f(s) = a \neq stop \wedge \exists e.\exists \delta'.\exists \vec{x}, \vec{x}'.Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s)) \wedge$$
$$\forall e.(\exists \delta'.\exists \vec{x}, \vec{x}'.Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s))) \supset$$
$$\exists \delta'.\exists \vec{x}, \vec{x}'.Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s)) \wedge P(\delta', \vec{x}', do(a[\vec{x}'](e), s)) \supset P(\delta, \vec{x}, s)]$$

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

Theorem 9. *For any subprogram $\delta \in \Gamma_{\delta_0}$, situation s and strategy f, we have that:*

$M \models AgtCanForceIf(\delta, \vec{x}, f, s)$
iff (δ, s) is in the least set P_μ such that:
if $f(s) = stop$ and $M \models Final(\delta, \vec{x}, s)$, then $(\delta, \vec{x}, s) \in P_\mu$
if $f(s) = a \neq stop$ and there exists $e, \delta', \vec{x}$ and $\vec{x}'$ s.t. $M \models Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s))$ and for all e the existence of $\delta', \vec{x}$ and $\vec{x}'$ s.t. $M \models Trans(\delta, \vec{x}, s, \delta', \vec{x}', do(a[\vec{x}'](e), s))$ implies $(\delta', \vec{x}', do(a[\vec{x}'](e), s))$ is in P_μ, then $(\delta, \vec{x}, s) \in P_\mu$
iff $(\delta, \vec{x}, s)$ is in the least set P_μ such that
if $f(s) = stop$ and $M \models F(\delta)[\vec{x}][s]$, then $(\delta, \vec{x}, s) \in P_\mu$
if $f(s) = a \neq stop$ and there exists $\varphi, \delta', \vec{x}$ and $\vec{x}'$ s.t. $\tau(\delta, \varphi, P, a, \delta')$, for all z not in P, $x'_z = x_z$, and $M \models \varphi[\vec{x}'][s]$, and for all e the existence of $\delta', \vec{x}$ and $\vec{x}'$ s.t. $\tau(\delta, \varphi, P, a, \delta')$ and $M \models \varphi[\vec{x}'][s]$ implies that $(\delta', \vec{x}', do(a[\vec{x}'](e), s))$ is in P_μ, then $(\delta, \vec{x}, s) \in P_\mu$

Theorem 10. *Let δ_0 be a Golog program, $\mathcal{D}$ an NDBAT where the program is executed, M a model of $\mathcal{D}$, $\mathcal{D}_M$ the nondeterministic domain corresponding to M, and A the game arena generated by program δ_0 and domain $\mathcal{D}_M$. Then $M \models AgtCanForce(\delta_0, S_0)$ iff there exists a winning strategy in A.*

4. Using Programs to Constrain the Environment

Our framework naturally extends to scenarios where the environment's behavior is constrained by its own program. We assume that the environment program contains only one action, namely $DoReaction$, taking the reaction e as a parameter, and it can contain a special symbol ac for referring the current action executed by the agent. Here, we need to change the semantics of the programs and define $Trans$ and $Final$ for both the agent program, denoted δ_a, and the environment program, denoted δ_e. Below is a sketch of the transition relations:

$Trans_a(a, \vec{x}, s, \delta'_a, \vec{x}', a) \equiv$	$Trans_e(DoReaction(e), \vec{x}, a, s, \delta'_e, \vec{x}', e) \equiv$
$\exists e.Poss_{ag}(a[\vec{x}], s) \wedge \delta' = nil \wedge \vec{x}' = \vec{x}$	$Poss(a(e), s) \wedge \delta'_e = nil \wedge \vec{x}' = \vec{x}$
$Trans_a(\varphi?, \vec{x}, s, \delta'_a, \vec{x}', a) \equiv \texttt{False}$	$Trans_e(\varphi?, \vec{x}, a, s, \delta'_e, \vec{x}', e) \equiv \texttt{False}$
...	...

$Trans_a$ is the same as in the original definition, but it takes the action performed a as last parameter instead of the next situation s'; $Trans_e$ also is similar, but it takes as input both a and e (because the reaction depends on the action chosen by the agent), and again it drops s'. $Final_a$ and $Final_e$ are equal to the original definition, except that $Final_e$ has a among its parameter, and in the final condition for tests we substitute the symbol ac with the actual action a, i.e. $Final_e(\varphi?, \vec{x}, a, s) \equiv \varphi[\vec{x}, ac/a][s]$.

Finally, system transitions result from the interleaved execution of the agent program. More formally, we define the transition relation: $Trans(\delta_a, \vec{x}, \delta_e, \vec{y}, s, \delta'_a, \vec{x}', \delta'_e, \vec{y}', s') \equiv Trans_a(\delta_a, \vec{x}, s, \delta'_a, \vec{x}', a) \wedge Trans_e(\delta_e, \vec{y}, a[\vec{x}'], s, \delta'_e, \vec{y}', e) \wedge s' = do(a[\vec{x}'](e), s)$. A final configuration is reached when both programs have terminated: $Final(\delta_a, \vec{x}, \delta_e, \vec{y}, s) \equiv Final_a(\delta_a, \vec{x}, s) \wedge Final_e(\delta_e, \vec{y}, a, s)$. Note that we could easily construct a program graph for the environment programs, and if we compute the cross product between the program graph of δ_a and the program graph of δ_e, we still obtain a program graph. This means that everything we have shown carries over even in this setting.

This leads to a joint fixpoint definition of agent-environment interaction. We extend the predicate $AgtCanForceIf$ to capture whether an agent strategy f can enforce successful execution of δ_a in the presence of an adversarial but constrained environment running δ_e.

$AgtCanForceIf(\delta_a, \vec{x}, \delta_e, \vec{y}, f, s) \doteq \forall P.[\ldots \supset P(\delta_a, \vec{x}, \delta_e, \vec{y}, s)]$
where ... stands for
$[(f(s) = stop \wedge Final(\delta_a, \vec{x}, \delta_e, \vec{y}, s)) \supset P(\delta_a, \vec{x}, \delta_e, \vec{y}, s)] \wedge$
$[\exists a.(f(s) = a \neq stop \wedge \exists e.\exists \vec{x}', \vec{y}'.\exists \delta'_a, \delta'_e.Trans(\delta_a, \vec{x}, \delta_e, \vec{y}, s, \delta'_a, \vec{x}', \delta'_e, \vec{y}', do(a[\vec{x}'](e), s)) \wedge$
$\forall e.\forall \vec{y}'.(\exists \delta'_a, \delta'_e.\exists \vec{x}.Trans(\delta_a, \vec{x}, \delta_e, \vec{y}, s, \delta'_a, \vec{x}', \delta'_e, \vec{y}', do(a[\vec{x}'](e), s)) \supset$
$P(\delta'_a, \vec{x}', \delta'_e, \vec{y}', do(a[\vec{x}'](e), s)) \supset P(\delta_a, \vec{x}, \delta_e, \vec{y}, s)]$

Acknowledgments

This work has been partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project FAIR (No. PE0000013), the Italian National Ph.D. on Artificial Intelligence at Sapienza University of Rome, the National Science and Engineering Research Council of Canada, and York University.

References

[1] J. A. Baier, C. Fritz, S. A. McIlraith, Exploiting procedural domain control knowledge in state-of-the-art planners., in: ICAPS, 2007, pp. 26–33.

[2] J. A. Baier, C. Fritz, M. Bienvenu, S. A. McIlraith, Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners., in: AAAI, 2008, pp. 1509–1512.

[3] C. Fritz, J. A. Baier, S. A. McIlraith, Congolog, sin trans: Compiling congolog into basic action theories for planning and beyond, in: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning, 2008, pp. 600–610.

[4] T. Hofmann, J. Claßen, Ltlf synthesis on first-order agent programs in nondeterministic environments, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 14976–14986.

[5] G. De Giacomo, Y. Lespérance, The nondeterministic situation calculus, in: M. Bienvenu, G. Lakemeyer, E. Erdem (Eds.), Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021, 2021, pp. 216–226. URL: https://doi.org/10.24963/kr.2021/21. doi:10.24963/KR.2021/21.

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

[7] G. De Giacomo, Y. Lespérance, A. R. Pearce, Situation calculus based programs for representing and reasoning about game structures, in: KR, 2010, pp. 445–455.

[8] G. De Giacomo, Y. Lespérance, F. Patrizi, S. Sardina, Verifying congolog programs on bounded situation calculus theories, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016.

[9] G. De Giacomo, Y. Lespérance, C. J. Muise, On supervising agents in situation-determined ConGolog, in: AAMAS, IFAAMAS, 2012, pp. 1031–1038.