Strategic Reasoning over Golog Programs in the Nondeterministic Situation Calculus

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

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

Overview

Context and Motivation:

- Automata-based planning and synthesis handle declarative goals, not procedures.
- Golog programs offer rich, high-level task specifications.
- Existing approaches do not fully handle adversarial nondeterminism.

Objective:

- Provide a game-theoretic framework for reasoning over Golog programs.
- Enable strategy synthesis for executing the program under adversarial environments.

Framework Principles

Main Components:

- Program Graph: captures the control flow of a Golog program independently of the domain.
- Domain Specification: nondeterministic first-order basic action theory (NDBAT).
- Game Arena: cross product of the program graph and domain representing agent–environment interactions.

Synthesis Problem:

- Synthesize a winning strategy for the agent that guarantees program completion against all environment moves.

Tools: NDBAT

Situation Calculus:

- FO formalism for specifying dynamically-evolving worlds and reasoning about actions.
- A situation is a sequence of actions, starting from an initial situation S_0; $do(a, s)$ is the situation that results from doing action a in situation s.
- A fluent is a predicate that depends on the situation.
- A Basic Action Theory is a set of axioms including precondition and successor state axioms

Nondeterministic Situation Calculus:

- Extends Situation Calculus to handle unpredictable outcomes for actions.
- The environment behavior is modeled by an additional reaction parameter e.
- The agent action $a(\vec{t})$ is the decision under the agent's control.
- The system action $a(\vec{t}, e)$ depends also on environment choices.

Tools: Golog

Golog is a high-level language for writing programs that are executed over (ND-)BATs.

Syntax: defined by

$$\delta := a(\vec{t}) \mid \varphi? \mid \delta_1;\delta_2 \mid \delta_1|\delta_2 \mid \pi x.\delta(x) \mid \delta^*$$

where $a(\vec{t})$ is an action term, and φ is a situation-suppressed formula.

Semantics: specified in terms of single steps, using $Trans(\delta, \vec{x}, s, \delta', \vec{x}', s')$ and $Final(\delta, \vec{x}, s)$.

Syntactic closure (Γ_{δ_0}): defined inductively as follows

$$\begin{array}{l} \delta_0, nil \in \Gamma_{\delta_0} \\ \text{if } \delta_1;\delta_2 \in \Gamma_{\delta_0} \text{ and } \delta_1' \in \Gamma_{\delta_1}, \text{ then } \delta_1';\delta_2 \in \Gamma_{\delta_0} \text{ and } \Gamma_{\delta_2} \subseteq \Gamma_{\delta_0} \\ \text{if } \delta_1 \mid \delta_2 \in \Gamma_{\delta_0}, \text{ then } \Gamma_{\delta_1}, \Gamma_{\delta_2} \subseteq \Gamma_{\delta_0} \\ \text{if } \delta^* \in \Gamma_{\delta_0}, \text{ then } \delta;\delta^* \in \Gamma_{\delta_0} \end{array}$$

Γ_{δ_0} is linear in the size of δ_0.

Program Graph Construction

Key Idea:

- Each node represents a subprogram from Γ_{δ_0} (the remaining part of the execution).
- Edges correspond to possible execution steps, annotated with guards (preconditions and test formulas).
- A label is associated with each node to indicate whether the subprogram is final.

Properties:

- Provides a fully syntactic, domain-independent, and compact representation of programs.
- Execution paths in the graph correspond to transitions in standard Golog semantics.
- If the program is situation determined, the program graph is deterministic

Example: Coffee-Delivery Robot

Description:

- A robot must deliver coffee to rooms behind doors.
- The doors are closed.
- Pressing a button opens a random door (environment nondeterminism).
- If the room has already been served, it is ignored, and a new room should be selected by pressing the button again.
- If the room has not been served, the robot delivers the coffee.
- Goal: all rooms eventually receive coffee.

Program:

$$\begin{aligned}\delta_0 = (&\mathit{Pickup}; (\mathit{PressButton}; \exists r.\phi(r)?)^*;\\ &\pi r.\phi(r)?; \mathit{DeliverTo}(r))^*; \forall r.\mathit{Delivered}(r)?\end{aligned}$$

Example: Coffee-Delivery Robot

$$\begin{tikzcd}
\boxed{\delta_0 = (\mathit{Pickup}; (\mathit{PressButton}; \exists r.\phi(r)?)^*; \pi r.\phi(r)?; \mathit{DeliverTo}(r))^*; \forall r.\mathit{Delivered}(r)?} \arrow[d, "\text{Pickup}"'] \\
\boxed{\delta_1 = \mathit{nil};\ (\mathit{PressButton}; \exists r.\phi(r)?)^*; \pi r.\phi(r)?; \mathit{DeliverTo}(r); \delta_0} \arrow[loop above, "\text{PressButton}"] \arrow[d, bend left, "{\{r\},\ \text{DeliverTo}}"] \\
\boxed{\delta_2 = \mathit{nil}; \delta_0} \arrow[u, bend left, "\text{Pickup}"]
\end{tikzcd}$$

Game-Theoretic Synthesis

Agent vs. Environment:

- Agent executes program actions.
- Environment introduces nondeterministic outcomes (reactions).

Game Arena:

- States: pairs of (program node, situation).
- Transitions: alternation of agent and environment moves.
- Objective: reach a terminal state where the program is completed.

What this gives you:

- A winning strategy for successful program execution.
- Sound synthesis procedure with formal guarantees.
- Enables strategic reasoning in a first-order setting.

Ongoing Work and Future Directions

Constraining the Environment:

- Real-world environments are adversarial, but structured, not arbitrary.
- Use environment programs δ_e with single action $DoReaction(e)$.
- Adapt the single-step semantics to allow interleaved execution of δ_a and δ_e.
- Alternatively, use LTL constraints to guide the behavior of the environment.

Propositional Setting:

- Make the FO synthesis effective by moving to a finite-state, propositional framework, where we have finitely many objects, actions and fluents.
- Use a symbolic procedure to compute the winning strategy.
- Develop a running implementation and benchmark its performance.
- Provide an empirical comparison with declarative-based approaches.