Strategic Reasoning over Golog Programs in the Nondeterministic Situation Calculus

Giuseppe De Giacomo[a,b], Yves Lespérance[c], and Matteo Mancanelli[b]

[a]University of Oxford, (Oxford, UK), [b]Sapienza University (Rome, Italy), [c]York University (Toronto, ON, Canada)

Overview

Context and Motivation:

- Automata-based synthesis focus on declarative goals.
- Golog programs offer procedural high-level task specifications.
- Existing approaches do not handle adversarial nondeterminism.

Objective:

- Provide a game-theoretic framework for Golog programs.
- Enable program execution under adversarial environments.

Framework Principles

Main Components:

- Program Graph: captures the control flow of a Golog program.
- Domain Specification: expressed as a NDBAT.
- Game Arena: cross product of the program graph and domain.

Synthesis Problem:

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

Tools: Nondeterministic Situation Calculus and Golog

Situation Calculus:

- FO formalism for specifying dynamically-evolving worlds.
- A situation is a sequence of actions, starting from an initial S_0
- A fluent is a predicate that depends on the situation.
- A Basic Action Theory (BAT) is a set of axioms including action preconditions and successor state axioms

Nondeterministic Situation Calculus:

- Extends SitCalc to handle unpredictable outcomes for actions.
- The environment behavior is modeled by a 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.

Golog: a high-level programming language for writing programs that are executed over (nondeterministic) action theories.

Syntax:

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

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

Syntactic closure (Γ_{δ_0}):

$\delta_0, nil \in \Gamma_{\delta_0}$
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 $\Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$
if $\delta_1 \mid \delta_2 \in \Gamma_{\delta_0}$, then $\Gamma_{\delta_1}, \Gamma_{\delta_2} \subseteq \Gamma_{\delta_0}$
if $\delta^* \in \Gamma_{\delta_0}$, then $\delta;\delta^* \in \Gamma_{\delta_0}$

Program Graph

Key Idea:

- Each node represents a subprogram from the syntactic closure Γ_{δ_0}.
- Edges correspond to possible execution steps, annotated with guards (preconditions and test formulas).
- Each node has a boolean label signaling if 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.
- In many real-world problems, the program graph is deterministic.

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.

Extensions and Applications

Constraining the Environment:

- Real-world environments are adversarial, but structured.
- Use environment programs δ_e with single action $DoReaction(e)$.
- Adapt Golog semantics to allow interleaved execution of δ_a and δ_e.
- Alternatively, specify admissible environment behaviors using LTL constraints.

Propositional Setting:

- Move to a finite-state, propositional framework, with finitely many objects, actions and fluents.
- Use a symbolic procedure to compute the winning strategy.
- Develop a running implementation.
- Provide an empirical comparison with LTL_f/LDL_f.

Example: Coffee-Delivery Robot

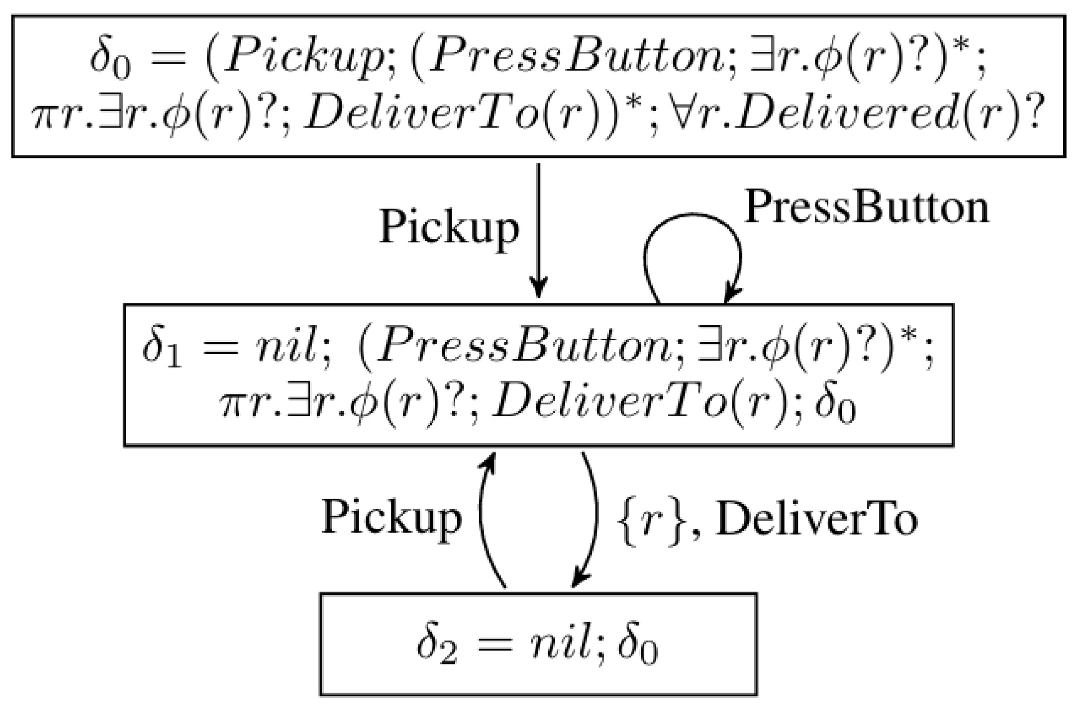
Description:

- A robot must deliver coffee to rooms behind doors.
- The doors are closed.
- Pressing a button opens a random door.
- If the room has already been served, it is ignored; 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.

Agent Program:

$$\delta_0 = (Pickup; (PressButton; \exists r.\phi(r)?)^*; \pi r.\phi(r)?; DeliverTo(r))^*; \forall r.Delivered(r)?$$

Program Graph:



Environment Program:

$$\delta_e = (((\pi x.ac \neq PressButton \land x = Success?; DoReaction(x))^*); (\pi y.ac = PressButton \land CanOpen(y)?; DoReaction(y)))^*$$

Learn More

Scan Me