Strategy Synthesis for First-Order Agent Programs over Finite Traces

Till Hofmann[1] and Jens Claßen[2]

[1] RWTH Aachen University, Germany
till.hofmann@cs.rwth-aachen.de
[2] Roskilde University, Denmark
classen@ruc.dk

In this work,[1] we consider the task of synthesizing an execution strategy for an agent from a high-level description of the initial state of the world, the actions available to the agent, a control program, and a temporal goal. In particular, we look at the case of infinite-state systems with unbounded object domains, based on a specification formalism with first-order expressiveness, as well as exogenous events, triggered by the non-deterministic environment. More specifically, we use the agent programming language Golog [11]. Golog in turn is based on the situation calculus [13, 15], a first-order logic formalism for reasoning about change. Here, a situation calculus action theory (perhaps incompletely) describes the initial state of the world, together with the preconditions and effects of primitive actions at the agent's disposal, while Golog programs then combine primitive actions into more complex behaviours using sequence, iteration, non-deterministic branching, and concurrency [5]. Since both imperative and non-deterministic program constructs are included, this allows for combining programming and planning in a flexible manner.

As an example, adapted from [3], consider a robot that has to clean dirty dishes. It can move between a number of different rooms and the kitchen, load (an arbitrary number of) dirty dishes onto itself located in specific rooms, and unload dishes it carries into the dishwasher in the kitchen. Here we use a modal variant of the situation calculus called $\mathcal{ES}$ [10], where these actions would be represented through functions $goto(x)$, $load(x, y)$, and $unload(x)$, respectively, whereas properties that change due to actions are encoded by *fluent* predicates such as $At(x)$ or $OnRobot(x)$. An action theory $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{pre}} \cup \mathcal{D}_{\text{post}}$ then consists of three parts:

1. The *initial theory* $\mathcal{D}_0$ is a finite set of axioms that encodes what is true initially. For example, the robot might be in the kitchen, and not carry any dirty dishes yet:

$$At(kitchen) \wedge \neg\exists x\, OnRobot(x)$$

2. The *precondition axiom* $\mathcal{D}_{\text{pre}}$ states when each action a can be executed in terms of the special fluent $\text{Poss}(a)$. For instance, the robot can unload a dirty dish x into the dishwasher iff it is currently carrying x and it is located in the kitchen (the modal operator $\Box$ expresses that the subformula is true now and after any sequence of actions; free variable x is understood as implicitly $\forall$-quantified from the outside):

$$\Box\,\text{Poss}(unload(x)) \equiv OnRobot(x) \wedge At(kitchen)$$

3. The *successor state axioms* (SSAs) $\mathcal{D}_{\text{post}}$ express the changes to fluents' values due to actions. For example, the robot will hold dish x after action a just in case a was the action of loading x from room y, or the robot was already holding x previously and a was not the action of unloading x (the modal operator $[a]$ reads as "after action a"):

$$\Box[a]\, OnRobot(x) \;\equiv\; \exists y.\, a = load(x, y) \vee OnRobot(x) \wedge a \neq unload(x)$$

[1] This paper gives an overview. A full version with all formal details is available at [8].

Given such a theory, a Golog program for the robot could be the following:

loop:
while $\exists x.\, \mathit{OnRobot}(x)$ **do** $\pi x : \{d_1, \ldots, d_m\}.\, \mathit{unload}(x)$;
$\pi y : \{r_1, \ldots, r_n\}.\, \mathit{goto}(y)$;
while $\exists x.\, \mathit{DirtyDish}(x, y)$ **do** $\pi x : \{d_1, \ldots, d_m\}.\, \mathit{load}(x, y)$;
$\mathit{goto}(\mathit{kitchen})$

That is to say, the agent is instructed to iterate (**loop**) a subprogram where in each cycle, it first performs a loop to unload any dishes it is holding into the dishwasher. Afterwards, it chooses a room to move to, where it loads up all dirty dishes (if any) in another loop. Finally, it moves back to the kitchen. Here, π operators denote a finitary non-deterministic choice of argument; e.g., $\pi x : \{d_1, \ldots, d_m\}$ means "choose some x from among dishes $d_1, \ldots, d_m$", where each d_i is an $\mathcal{ES}$ constant.

The program thus defines a general structure for the robot's course of action, but leaves certain choices open, such as what room to go next to. Typically, it is assumed that the agent is in complete control, i.e., that all such non-determinism is "angelic". More recently, different forms of "demonic" non-determinism have been studied [4, 2], where actions have outcomes that are determined by the environment. For example, we might have another program running in parallel that occasionally triggers an action to place some new dirty dish x into some room y (to simulate a dynamic environment with people that use the dishes):

loop: $\pi x : \{d_1, \ldots, d_m\}, y : \{r_1, \ldots, r_n\}.\, \mathit{newDish}(x, y)$

Here we are particularly interested in scenarios where agent and environment do not act in turns, as is often assumed, but where they more realistically may act in arbitrary order, similar to *supervisory control* [14]. In this setting, program realization becomes a synthesis task. The goal is to determine a policy that executes the program, while also satisfying a temporal goal, independent of and reacting to all possible environment behaviors. Temporal formulas are expressed in terms of $\mathrm{LTL_f}$, a restriction of Linear Temporal Logic (LTL) to finite traces [6]. For example, we may want to require that eventually ($\mathcal{F}$) there will always ($\mathcal{G}$) be no more dirty dish:

$$\mathcal{F}\,\mathcal{G}\,\neg\exists x, y.\, \mathit{DirtyDish}(x, y) \tag{1}$$

The Golog language provides a large degree of expressiveness, in particular in terms of first-order quantification, allowing to represent infinite-state systems over unbounded domains. The synthesis problem is thus highly undecidable in general. In this work, we present a decidable approach that works on a non-trivial fragment. Specifically, exploiting results on decidable verification of Golog [17], we require that

- all non-modal subformulas fall into the (decidable) two-variable fragment of first-order logic with counting quantifiers [7],
- the non-deterministic choice of action arguments π only ranges over finite sets, and
- dependencies among fluent predicates in successor state axioms satisfy a certain acyclicity criterion.

We can then construct an abstract, finite "game arena" (a special form of transition system) that captures all possible program executions while also tracking the satisfaction of the temporal specification. Using an encoding of $\mathrm{LTL_f}$ formulas that interprets temporal formulas as propositional atoms [12], the construction works on-the-fly and avoids building irrelevant parts.

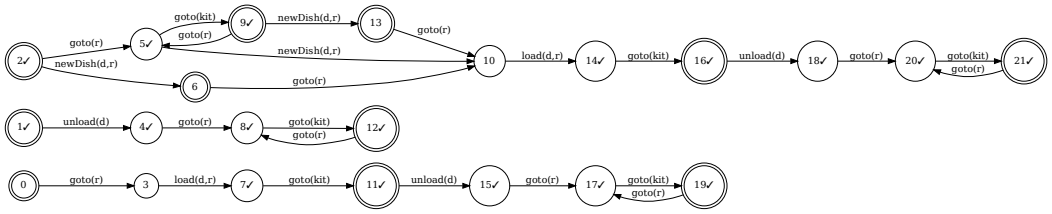

Figure 1: Example game arena with single room `r` and single dish `d`. Left are initial states with one new dish and none on robot (0), no new dish and one on robot (1), and no new dish and none on robot (2). Double circles indicate *final* states where program execution may terminate. Check marks indicate *accepting* states where the temporal goal is completed.

A game-theoretic approach can then be applied to synthesize a policy. Figure 1 shows an example game arena for the dishwasher robot if there is only one room r and one dish d, but where the initial state is underspecified so that the dish may initially be in the room, on the robot, or neither.

We implemented the method for the Golog interpreter `vergo` [1], which uses embedded theorem provers [16, 9] for first-order reasoning tasks and a first-order variant of binary decision diagrams for concisely representing formulas. We did an experimental evaluation on the dishwasher robot domain as well as a domain with a warehouse robot that moves boxes which may fall non-deterministically and break their contents. In the experiment, we varied the overall numbers of dishes, rooms, and boxes, and measured the method's runtime as well as the size of the resulting game arenas and extracted strategies. The set time-out of 1500 seconds was reached quickly for instances with 3 or more rooms, 3 or more dishes, and 3 or more boxes, yielding game arenas up to around 3000 states and transitions. Note that although decidable, the problem is very hard: In the worst case, the number of states in the abstract game arena is double exponential in the size of the input, and computing them involves consistency checks over sets of formulas that take up to double exponential time.

While the experiments thus demonstrate that the method works in principle, they also point out possible avenues of future work in terms of possible improvements. In particular, Golog programs often contain symmetries in the sense that there is a number of objects each of which need to be handled independently in the same way (e.g., the dishes in our example). For solving the task, the order of handling objects is hence irrelevant, yet the current method materializes all possible permutations, resulting in a severe blow-up of the size of the abstract transition system. It may therefore be interesting to study how the approach can be adapted to detect and deal with symmetries of this kind. Another limitation is the restriction that the π operator ranges over finite sets only. It can be shown that dropping this constraint altogether quickly results in undecidability, but the condition seems very harsh in light of the fact that non-finitary quantification is allowed in other places such as the action theory or the temporal goal. We are therefore interested in identifying perhaps "softer" restrictions that still guarantee decidability, yet allow for picking action arguments from potentially infinite sets.

References

[1] Jens Claßen. Symbolic verification of Golog programs with first-order BDDs. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *Proceedings of the Sixteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2018)*, pages 524–529. AAAI Press, 2018.

[2] Jens Claßen and James P. Delgrande. An Account of Intensional and Extensional Actions, and its Application to Belief, Nondeterministic Actions and Fallible Sensors. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, volume 18, pages 194–204, September 2021.

[3] Jens Claßen, Martin Liebenberg, Gerhard Lakemeyer, and Benjamin Zarrieß. Exploring the boundaries of decidable verification of non-terminating Golog programs. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1012–1019. AAAI Press, 2014.

[4] Giuseppe De Giacomo and Yves Lespérance. The nondeterministic situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, volume 18, pages 216–226. AAAI Press, September 2021.

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

[6] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1558–1564. AAAI Press, 2015.

[7] Erich Grädel, M. Otto, and E. Rosen. Two-variable logic with counting is decidable. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 306–317, June 1997.

[8] Till Hofmann and Jens Claßen. LTLf synthesis on first-order action theories, 2024. arXiv:2410.00726.

[9] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Proceedings of the Twentyfifth International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[10] Gerhard Lakemeyer and Hector J. Levesque. A semantic characterization of a useful fragment of the situation calculus with knowledge. *Artificial Intelligence*, 175(1):142–164, 2010.

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

[12] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. SAT-based explicit LTLf satisfiability checking. *Artificial Intelligence*, 289:103369, December 2020.

[13] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[14] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

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

[16] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In Pascal Fontaine, editor, *Proceedings of the Twenty-Seventh International Conference on Automated Deduction (CADE 2019)*, volume 11716 of *Lecture Notes in Computer Science*, pages 495–507. Springer, 2019.

[17] Benjamin Zarrieß and Jens Claßen. Decidable verification of Golog programs over non-local effect actions. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1109–1115. AAAI Press, 2016.