$\mathrm{LTL_f}$ Synthesis on First-Order Agent Programs in Nondeterministic Environments

Till Hofmann[1], **Jens Claßen**[2]

[1]Department of Computer Science, RWTH Aachen University
[2]Institute for People and Technology, Roskilde University
till.hofmann@cs.rwth-aachen.de, classen@ruc.dk

Abstract

We investigate the synthesis of policies for high-level agent programs expressed in Golog, a language based on situation calculus that incorporates nondeterministic programming constructs. Unlike traditional approaches for program realization that assume full agent control or rely on incremental search, we address scenarios where environmental nondeterminism significantly influences program outcomes. Our synthesis problem involves deriving a policy that successfully realizes a given Golog program while ensuring the satisfaction of a temporal specification, expressed in Linear Temporal Logic on finite traces (LTLf), across all possible environmental behaviors. By leveraging an expressive class of first-order action theories, we construct a finite game arena that encapsulates program executions and tracks the satisfaction of the temporal goal. A game-theoretic approach is employed to derive such a policy. Experimental results demonstrate this approach's feasibility in domains with unbounded objects and non-local effects. This work bridges agent programming and temporal logic synthesis, providing a framework for robust agent behavior in nondeterministic environments.

1 Introduction

Agents operating in dynamic environments often need to react to changes beyond their control. For example, a service robot may be tasked with serving coffee to customers, who may place an order at any time. Also, some actions may have unexpected outcomes, e.g., while attempting to fulfill the task, the robot might accidentally drop the coffee. Such scenarios can be modeled as *fully observable nondeterministic* (FOND) planning tasks (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016) where actions have multiple possible outcomes, or as reactive synthesis problem (e.g., based on *LTL on finite traces* ($\mathrm{LTL_f}$) (De Giacomo and Vardi 2015)), where certain propositions are controlled by the agent while others are governed by the environment. However, these approaches have some limitations. They assume a fixed, finite set of propositions, effectively imposing a closed-world assumption and requiring a completely known initial state. Furthermore, they rely on alternating actions between the agent and the environment, which fails to capture scenarios where the environment may perform an arbitrary number of actions before the agent can respond. Additionally, existing solutions often generate arbitrary plans or policies without incorporating user-specified partial strategies unless explicitly encoded in the specification.

On the other hand, GOLOG (Levesque et al. 1997), a well-established agent programming language, offers significant flexibility. Based on the situation calculus (McCarthy and Hayes 1969; Reiter 2001a), GOLOG supports first-order reasoning over arbitrarily large or even infinite domains and accommodates incomplete information about the initial state. It also allows for the specification of partial strategies through nondeterministic programs. Given such a program, *program realization* is the task of resolving program nondeterminism to produce a successful program execution, e.g., by means of search, or in an online incremental fashion (De Giacomo et al. 2009). However, it is typically assumed that the agent is in complete control, even if it only has incomplete knowledge (Reiter 2001b; Claßen and Neuss 2016) or its actions are stochastic (Boutilier et al. 2000). Recently, the situation calculus has been extended with nondeterministic actions (De Giacomo and Lespérance 2021; Claßen and Delgrande 2021) similar to FOND planning, where the environment chooses an outcome. However, this still assumes that agent and environment act in turns.

To address these limitations, we propose an extension to GOLOG that partitions actions into agent actions and environment actions. In this framework, the agent selects among currently applicable agent actions, guided by the program and the basic action theory, but cannot constrain the environment. The environment may select any applicable environment action or any action chosen by the agent. This allows for arbitrary sequences of environment actions, similar to the *supervisory control* paradigm (Ramadge and Wonham 1989). We also propose to describe the agent's goal as a temporal formula, which allows for formulating trajectory constraints such as safety and reachability on the program.

In this setting, program realization becomes a synthesis task. Given a GOLOG program and a temporal goal, the task is to synthesize a policy that executes the program while satisfying the temporal goal, independent of and reacting to all possible environment behaviors. In this paper, we focus on the decidable fragment of GOLOG with acyclic basic action theories restricted to C^2 (Zarrieß and Claßen 2016) and temporal goals given as $\mathrm{LTL_f}$ formulas (De Giacomo and Vardi 2013). We provide a decidable approach for this problem by constructing a finite game arena that captures all possible program executions while tracking the satisfaction of the temporal specification, and then applying a game-theoretic approach to synthesize a policy. Exploiting an encoding of $\mathrm{LTL_f}$ formulas that interprets temporal formulas as propositional atoms (Li et al. 2020), the construction works on-the-fly and avoids building irrelevant parts.

The remainder of this paper is structured as follows. After discussing related work in Section 2, we summarize GOLOG and introduce $\mathrm{LTL_f}$ in the context of GOLOG programs in Section 3. We describe the synthesis approach in Section 4 and evaluate it experimentally in Section 5, before concluding in Section 6.

2 Related Work

Verification of GOLOG programs has been explored in various contexts. Initially, verification efforts relied on manual proofs (De Giacomo, Ternovska, and Reiter 1997; Liu 2002; Shapiro, Lespérance, and Levesque 2002). Claßen and Lakemeyer (2008) describe a (possibly not terminating) system that is capable of automatically verifying properties of non-terminating GOLOG programs. Later research identified decidable fragments of GOLOG grounded in C^2, the decidable two-variable fragment of first-order logic with counting (Grädel, Otto, and Rosen 1997). Verification of GOLOG programs with *context-free* or *local-effect* basic action theories (BATs) in C^2 and with pick operators restricted to finite domains is decidable for properties in CTL (Claßen et al. 2014), LTL (Zarrieß and Claßen 2014a), and CTL^* (Zarrieß and Claßen 2014b). Beyond local-effect BATs, verification remains decidable if the BAT is *acyclic*, i.e., there is no cyclic dependency between fluents in the effect descriptors, or *flat*, i.e., effect descriptors are quantifier-free (Zarrieß and Claßen 2016). *Bounded theories*, where the number of objects described by any situation is bounded, also results in decidable verification (De Giacomo, Lespérance, and Patrizi 2016). All these approaches rely on a finite abstraction of the infinite program configuration space, which yields decidability, and hence could be used as basis for our approach.

Related to verification is *synthesis* of temporal properties, which can be described as two-player games between the system and the environment (Abadi, Lamport, and Wolper 1989; Pnueli and Rosner 1989). Given a specification, e.g., in Linear Temporal Logic (LTL), and a partition of the symbols into controllable and uncontrollable ones, the players alternate selecting a subset of their symbols. LTL has also been used to describe temporally extended goals for planning (Bacchus and Kabanza 1998; De Giacomo and Vardi 2000; Geffner and Bonet 2013), possibly resulting in infinite plans (Patrizi et al. 2011). LTL can also be used to specify *conformant planning* problems with temporally extended goals (Calvanese, De Giacomo, and Vardi 2002) and synthesis is related to FOND planning (Camacho et al. 2017, 2018; De Giacomo and Rubin 2018) as a nondeterministic effect can be seen as an environment action. Moreover, there has been a particular interest in $\mathrm{LTL_f}$ (De Giacomo and Vardi 2013), where the synthesis problem can be solved by transforming the $\mathrm{LTL_f}$ specification into a finite automaton (De Giacomo and Vardi 2015). Like LTL, $\mathrm{LTL_f}$ synthesis is 2EXPTIME-complete, although $\mathrm{LTL_f}$ synthesis tools usually perform better. Recently, several methods have been proposed to improve the performance of $\mathrm{LTL_f}$ synthesis, e.g., based on BDDs (Zhu et al. 2017) and on-the-fly forward search (Xiao et al. 2021; De Giacomo et al. 2022; Favorito 2023).

3 Preliminaries

We describe the logic $\mathcal{ES}$ and an $\mathcal{ES}$-based variant of GOLOG and then introduce $\mathrm{LTL_f}$ in the context of GOLOG programs.

The Logic $\mathcal{ES}$

The logic $\mathcal{ES}$ (Lakemeyer and Levesque 2010) is a first-order modal variant of the situation calculus. Following (Zarrieß and Claßen 2016), we consider $\mathcal{ES}$ formulas restricted to C^2.

Syntax *Terms* are of sort *object* or *action*. We use $x, y, \ldots$ (possibly with decorations) to denote object variables, and a for a variable of sort action. N_O is a countably infinite set of *object constant symbols*, and N_A a countably infinite set of action function symbols whose arguments are all of sort object. Let $\mathcal{N}_O$ denote the set of all ground terms (called *standard names*) of sort object, and $\mathcal{N}_A$ those of sort action. Formulas are constructed over equality atoms and *fluent* predicates with at most two arguments of sort object, using the usual Boolean connectives, quantifiers, counting quantifiers, as well as modalities $\Box\phi$ ("ϕ holds after any sequence of actions"), and $[t]\phi$ ("ϕ holds after executing action t"). We call a formula *fluent* if it does not mention $\Box$ or $[\cdot]$. A *sentence* is a formula without free variables. A *C^2-fluent formula* is a fluent formula without actions and with at most two variables.

Semantics A *trace* is a finite sequence of action standard names. When a trace represents a history of already executed actions, it is called a *situation*. For a trace $z = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathcal{Z}$, we write $|z|$ for the length n of z, $z \cdot \alpha$ for the concatenation $\langle \alpha_1, \ldots, \alpha_n, \alpha \rangle$ of z with an action α, $z[i]$ for the ith action α_i, $z[..i]$ for the prefix $\langle \alpha_1, \ldots, \alpha_i \rangle$, and $z[i..]$ for the suffix $\langle \alpha_i, \ldots, \alpha_n \rangle$. Let $\mathcal{Z} = \mathcal{N}_A^*$ be the set of all traces, and $\mathcal{P}_F$ the set of all *primitive formulas* $F(n_1, ..., n_k)$, where F is a k-ary fluent with $0 \leq k \leq 2$ and the n_i are object standard names. A *world* w maps primitive formulas and situations to truth values, i.e., $w : \mathcal{P}_F \times \mathcal{Z} \to \{0, 1\}$. The set of all worlds is denoted by $\mathcal{W}$.

Definition 1 (Truth of Formulas). *Let $w \in \mathcal{W}$ be a world and α an action standard name. We define for every $z \in \mathcal{Z}$:*

1. $w, z \models F(n_1, \ldots, n_k)$ *iff* $w[F(n_1, \ldots, n_k), z] = 1$*;*
2. $w, z \models (n_1 = n_2)$ *iff* n_1 *and* n_2 *are identical;*
3. $w, z \models \phi_1 \wedge \phi_2$ *iff* $w, z \models \phi_1$ *and* $w, z \models \phi_2$*;*
4. $w, z \models \neg\phi$ *iff* $w, z \not\models \phi$*;*
5. $w, z \models \forall x.\phi$ *iff* $w, z \models \phi^x_n$ *for every* $n \in \mathcal{N}_x$*;*
6. $w, z \models \exists^{\leq m} x.\phi$ *iff* $|\{n \in \mathcal{N}_x \mid w, z \models \phi^x_n\}| \leq m$*;*
7. $w, z \models \exists^{\geq m} x.\phi$ *iff* $|\{n \in \mathcal{N}_x \mid w, z \models \phi^x_n\}| \geq m$*;*
8. $w, z \models \Box\phi$ *iff* $w, z \cdot z' \models \phi$ *for every* $z \in \mathcal{Z}$*;*
9. $w, z \models [\alpha]\phi$ *iff* $w, z \cdot \alpha \models \phi$*.*

Here, $\mathcal{N}_x$ refers to the set of all standard names of the same sort as x, and ϕ^x_n the result of simultaneously replacing all free occurrences of x in ϕ by n. We understand $\vee$, $\exists$, $\supset$, $\equiv$, $\top$ and $\bot$ as the usual abbreviations. For a set of sentences Σ and a sentence α, we write $\Sigma \models \alpha$ (read: Σ entails α) to mean that for every w, if $w, \langle\rangle \models \alpha'$ for every $\alpha' \in \Sigma$, then $w, \langle\rangle \models \alpha$. Finally, we write $\models \alpha$ (read: α is valid) to mean $\{\} \models \alpha$. Note that rule 2 above includes a unique names assumption for actions and objects into the semantics.

Basic Action Theories

To encode a dynamic domain, we employ a *basic action theory* (BAT) (Reiter 2001a) with additional restrictions (Zarrieß and Claßen 2016) for ensuring decidability:

Definition 2 (Basic Action Theory)**.** *A basic action theory (BAT) $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{post}$ is a set of axioms, where $\mathcal{D}_0$ is a finite set of C^2-fluent sentences describing the initial state of the world, and $\mathcal{D}_{post}$ is a finite set of successor state axioms (SSAs), one for each fluent, of the form*[1] $\Box[a]F(\vec{x}) \equiv \gamma_F^+ \vee F(\vec{x}) \wedge \neg\gamma_F^-$, *where the* positive effect condition γ_F^+ *and the* negative effect condition γ_F^- *are disjunctions of formulas of the form $\exists\vec{y}.\,(a = A(\vec{v}) \wedge \varepsilon \wedge \kappa)$ such that*

- *the free variables of the formula $\exists\vec{y}.\,(a = A(\vec{v}) \wedge \varepsilon \wedge \kappa)$ are among $\vec{x}$ and a,*
- *$A(\vec{v})$ is an action term and $\vec{v}$ contains $\vec{y}$,*
- *the* effect descriptor *ε is a fluent formula with no terms of sort action and the number of variables in ε that do not occur in $\vec{v}$ or occur bound in ε is less than or equal to two,*
- *the* context condition *κ is a fluent formula with free variables among $\vec{v}$, no terms of sort action, and at most two bound variables.*

Intuitively, the effect descriptor is the part of the effect condition that expresses *which objects* are affected, while the context condition encodes *whether* the effect takes place.

Acyclic BATs For a BAT $\mathcal{D}$, we can construct the *fluent dependency graph* $\Delta_{\mathcal{D}}$, which captures the dependencies between fluents in the effect descriptors. In $\Delta_{\mathcal{D}}$, each node is a fluent of $\mathcal{D}$ and there is a directed edge (F, F') from fluent F to fluent F' if there exists a disjunct $\exists\vec{y}.(a = A(\vec{v}) \wedge \varepsilon \wedge \kappa)$ in γ_F^+ or γ_F^- such that F' occurs in ε. A BAT is *acyclic* if $\Delta_{\mathcal{D}}$ is acyclic. Furthermore, the *fluent depth* of an acyclic BAT, denoted by $\mathrm{fd}(\mathcal{D})$, is the length of the longest path in $\Delta_{\mathcal{D}}$ and the *fluent depth of F w.r.t. $\mathcal{D}$*, denoted by $\mathrm{fd}_{\mathcal{D}}(F)$, is the length of the longest path in $\Delta_{\mathcal{D}}$ starting in F.

GOLOG Programs

We consider a set of program expressions that includes ground actions (α), tests for C^2-fluent sentences ($\phi?$), sequence of subprograms ($\delta_1; \delta_2$), nondeterministic choice ($\delta_1|\delta_2$), interleaved concurrent execution ($\delta_1\|\delta_2$), and nondeterministic iteration (δ^*). We write $\mathrm{nil} \doteq \top?$ for the empty program that always succeeds.

A GOLOG *program* $\mathcal{G} = (\mathcal{D}, \delta)$ consists of a C^2-BAT $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\mathrm{post}}$ and a program expression δ, where all fluents occurring in $\mathcal{D}$ and δ have a SSA in $\mathcal{D}_{\mathrm{post}}$. For a program $\mathcal{G} = (\mathcal{D}, \delta)$, we write $\mathcal{A}_{\mathcal{G}}$ for all action terms occurring in δ and we may omit the subscript if $\mathcal{G}$ is clear from context.

The semantics of GOLOG programs is based on transitions between configurations, where a configuration $\langle z, \rho\rangle$ consists of a sequence of already performed actions $z \in \mathcal{Z}$ and the remaining program $\rho \in \mathrm{sub}(\delta)$. Given a world $w \in \mathcal{W}$, the transition relation $\xrightarrow{w}$ among configurations is defined inductively. The set of final configurations $\mathrm{Fin}(w)$ defines the configurations where the program may terminate.

Definition 3 (Program Transition Semantics)**.** *For any world w, the set of final configurations $\mathrm{Fin}(w)$ is the smallest set such that*

$$\begin{aligned}
\langle z, \phi?\rangle &\in \mathrm{Fin}(w) \textit{ if } w, z \models \phi\\
\langle z, \delta_1; \delta_2\rangle &\in \mathrm{Fin}(w) \textit{ if } \langle z, \delta_1\rangle \in \mathrm{Fin}(w) \textit{ and } \langle z, \delta_2\rangle \in \mathrm{Fin}(w)\\
\langle z, \delta_1|\delta_2\rangle &\in \mathrm{Fin}(w) \textit{ if } \langle z, \delta_1\rangle \in \mathrm{Fin}(w) \textit{ or } \langle z, \delta_2\rangle \in \mathrm{Fin}(w)\\
\langle z, \delta_1\|\delta_2\rangle &\in \mathrm{Fin}(w) \textit{ if } \langle z, \delta_1\rangle \in \mathrm{Fin}(w) \textit{ and } \langle z, \delta_2\rangle \in \mathrm{Fin}(w)\\
\langle z, \delta^*\rangle &\in \mathrm{Fin}(w)
\end{aligned}$$

For any world w, the transition relation $\xrightarrow{w}$ among configurations is the least set satisfying

$$\begin{aligned}
\langle z, \alpha\rangle &\xrightarrow{w} \langle z \cdot \alpha, \mathrm{nil}\rangle \textit{ if } \alpha \textit{ is a ground action}\\
\langle z, \delta_1; \delta_2\rangle &\xrightarrow{w} \langle z', \rho; \delta_2\rangle \textit{ if } \langle z, \delta_1\rangle \xrightarrow{w} \langle z', \rho\rangle\\
\langle z, \delta_1; \delta_2\rangle &\xrightarrow{w} \langle z', \rho\rangle \textit{ if } \langle z, \delta_1\rangle \in \mathrm{Fin}(w) \textit{ and } \langle z, \delta_2\rangle \xrightarrow{w} \langle z', \rho\rangle\\
\langle z, \delta_1|\delta_2\rangle &\xrightarrow{w} \langle z', \rho\rangle \textit{ if } \langle z, \delta_1\rangle \xrightarrow{w} \langle z', \rho\rangle \textit{ or } \langle z, \delta_2\rangle \xrightarrow{w} \langle z', \rho\rangle\\
\langle z, \delta_1\|\delta_2\rangle &\xrightarrow{w} \langle z', \rho\|\delta_2\rangle \textit{ if } \langle z, \delta_1\rangle \xrightarrow{w} \langle z', \rho\rangle\\
\langle z, \delta_1\|\delta_2\rangle &\xrightarrow{w} \langle z', \delta_1\|\rho\rangle \textit{ if } \langle z, \delta_2\rangle \xrightarrow{w} \langle z', \rho\rangle\\
\langle z, \delta^*\rangle &\xrightarrow{w} \langle z', \rho; \delta^*\rangle \textit{ if } \langle z, \delta\rangle \xrightarrow{w} \langle z', \rho\rangle
\end{aligned}$$

We write $\|\delta\|^z_w$ for the set of traces starting in configuration $\langle z, \delta\rangle$ and ending in a final configuration.

Situation-Determined Programs Following (De Giacomo, Lespérance, and Muise 2012), we say that a program $\mathcal{G} = (\mathcal{D}, \delta)$ is *situation-determined*, iff for all $w \in \mathcal{W}$ with $w \models \mathcal{D}$, all $z, z' \in \mathcal{Z}$, and all program expressions δ', δ'': $\langle z, \delta\rangle \xrightarrow{w}{}^* \langle z', \delta'\rangle$ and $\langle z, \delta\rangle \xrightarrow{w}{}^* \langle z', \delta''\rangle$ implies $\delta' = \delta''$. We assume that all programs are situation-determined.

LTL$_\mathrm{f}$

For temporal properties, we define temporal formulas with the same syntax as LTL$_\mathrm{f}$ formulas, but replacing propositions with C^2-fluent sentences ϕ, i.e., $\Phi ::= \phi \mid \Phi \wedge \Phi \mid \mathcal{X}\,\Phi \mid \Phi\,\mathcal{U}\,\Phi$. For a temporal formula Φ, we denote the set of subformulas of Φ with $\mathrm{cl}(\Phi)$. For a set of formulas Ψ, we write $\bigwedge\Psi$ for $\bigwedge_{\Phi\in\Psi}\Phi$. As usual, we define $\mathcal{F}\,\Phi \doteq \top\,\mathcal{U}\,\Phi$ and $\mathcal{G}\,\Phi \doteq \neg\,\mathcal{F}\,\neg\Phi$, as well as $\Phi_1 \vee \Phi_2 \doteq \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $\mathcal{N}\,\Phi \doteq \neg\,\mathcal{X}\,\neg\Phi$, and $\Phi_1\,\mathcal{R}\,\Phi_2 \doteq \neg(\neg\Phi_1\,\mathcal{U}\,\neg\Phi_2)$. We define the truth of a temporal formula Φ, given a world w and traces z, z':

- $w, z, z' \models \phi$ iff $w, z \models \phi$,
- $w, z, z' \models \Phi_1 \wedge \Phi_2$ iff $w, z, z' \models \Phi_1$ and $w, z, z' \models \Phi_2$,
- $w, z, z' \models \mathcal{X}\,\Phi$ iff $z' = \alpha \cdot z'' \neq \langle\rangle$ and $w, z \cdot \alpha, z'' \models \Phi$,

[1] The operator $\Box$ has lowest precedence while $[\cdot]$ has highest precedence and free variables are implicitly assumed to be universally quantified from the outside.

- $w, z, z' \models \Phi_1 \,\mathcal{U}\, \Phi_2$ iff there exists $k \leq |z'|$ such that $w, z \cdot z'[..k], z'[k+1..] \models \Phi_2$ and for all $0 \leq i < k$, $w, z \cdot z'[..i], z'[i+1..] \models \Phi_1$.

TNF and XNF As we intend to track the satisfiability of the temporal formula Φ over the traces of the program, we adapt Tail Normal Form (TNF) and neXt Normal Form (XNF) from (Li et al. 2020). TNF explicitly marks the end of satisfying traces, while XNF allows us to split the temporal formula into a local part, which can be evaluated at the current state, and a future part, which is evaluated against the remaining trace. First, we say a formula is in *Negated Normal Form* (NNF) if all negations are in front of only atoms. Each LTL_f formula can be transformed into NNF by using the dual operators to push negation inwards. Based on NNF, we define TNF, which marks the last state of satisfying traces:

Definition 4. *Let Φ be an LTL_f formula in NNF. Its TNF $\operatorname{tnf}(\Phi)$ is defined as $\operatorname{t}(\Phi) \wedge \mathcal{F}\, \mathit{Tail}$, where Tail is a new atom to identify the last state of satisfying traces and $\operatorname{t}(\Phi)$ is an LTL_f formula defined recursively as follows:*

1. $\operatorname{t}(\Phi) = \Phi$ *if Φ is $\top, \bot$, or a C^2-fluent sentence;*
2. $\operatorname{t}(\mathcal{X}(\Psi)) = \neg \mathit{Tail} \wedge \mathcal{X}(\operatorname{t}(\Psi))$*;*
3. $\operatorname{t}(\mathcal{N}(\Psi)) = \mathit{Tail} \vee \mathcal{X}(\operatorname{t}(\Psi))$*;*
4. $\operatorname{t}(\Phi_1 \wedge \Phi_2) = \operatorname{t}(\Phi_1) \wedge \operatorname{t}(\Phi_2)$*;*
5. $\operatorname{t}(\Phi_1 \vee \Phi_2) = \operatorname{t}(\Phi_1) \vee \operatorname{t}(\Phi_2)$*;*
6. $\operatorname{t}(\Phi_1 \,\mathcal{U}\, \Phi_2) = (\neg \mathit{Tail} \wedge \operatorname{t}(\Phi_1)) \,\mathcal{U}\, \operatorname{t}(\Phi_2)$*;*
7. $\operatorname{t}(\Phi_1 \,\mathcal{R}\, \Phi_2) = (\mathit{Tail} \vee \operatorname{t}(\Phi_1)) \,\mathcal{R}\, \operatorname{t}(\Phi_2)$.

When interpreting a TNF formula over a trace, Tail needs to be treated separately, as it is not a fluent sentence. We define: $w, z, z' \models \mathit{Tail}$ iff $z' = \langle\rangle$. It can be shown that Φ and $\operatorname{tnf}(\Phi)$ are equivalent:[2]

Theorem 1. *Let Φ be a temporal formula, w a world, and z and z' traces. Then $w, z, z' \models \Phi$ iff $w, z, z' \models \operatorname{tnf}(\Phi)$.*

In the following, each LTL_f formula is assumed to be in TNF and we may omit the common part $\mathcal{F}\, \mathit{Tail}$.

We continue by interpreting temporal formulas as propositional formulas by treating sub-formulas with a temporal operator as outermost connective as if they were propositional atoms. For a temporal formula Φ, we define the set of *propositional atoms* $\operatorname{PA}(\Phi)$ of Φ inductively: (1) $\operatorname{PA}(\Phi) = \{\Phi\}$ if Φ is an atom, $\mathcal{X}$, $\mathcal{U}$, or $\mathcal{R}$ formula; (2) $\operatorname{PA}(\Phi) = \operatorname{PA}(\Psi)$ if $\Phi = \neg\Psi$; and (3) $\operatorname{PA}(\Phi) = \operatorname{PA}(\Phi_1) \cup \operatorname{PA}(\Phi_2)$ if $\Phi = \Phi_1 \wedge \Phi_2$ or $\Phi = \Phi_1 \vee \Phi_2$. For a temporal formula Φ, let Φ^p be Φ understood as a propositional formula over $\operatorname{PA}(\Phi)$. A propositional assignment P of Φ^p is a partial function $P : \operatorname{PA}(\Phi) \to \{0, 1\}$ that assigns truth values to the propositional atoms $\operatorname{PA}(\Phi)$. We write $P \models \Phi^p$ if P satisfies Φ^p. A propositional assignment P can also be understood as a set of literals $\{p \in \operatorname{PA}(\Phi) \mid P(p) = 1\} \cup \{\neg p \in \operatorname{PA}(\Phi) \mid P(p) = 0\}$ and we use P to denote both interchangeably.

If Φ is satisfiable, then there exists a corresponding propositional assignment:

Lemma 2. *Let w be a world, Φ an LTL_f formula, and z and z' traces. Then $w, z, z' \models \Phi$ implies there exists a propositional assignment P with $P \models \Phi^p$ and $w, z, z' \models \bigwedge P$.*

The converse is not necessarily true: Let $\Phi = \mathcal{X}(a) \wedge \mathcal{X}(\neg a)$. Clearly, Φ is not satisfiable, but $\{\mathcal{X}(a), \mathcal{X}(\neg a)\}$ is a satisfying propositional assignment of Φ^p.

We now define XNF, where each $\mathcal{U}$ and $\mathcal{R}$ operator is pushed inwards such that the only outermost temporal connective is $\mathcal{X}$:

Definition 5. *Let Φ be a temporal formula. Its neXt Normal Form (XNF) $\operatorname{xnf}(\Phi)$ is defined recursively as follows:*

1. $\operatorname{xnf}(\Phi) = \Phi$ *if Φ is $\top, \bot$, a C^2-fluent sentence, or $\mathcal{X}\,\Psi$;*
2. $\operatorname{xnf}(\Phi_1 \wedge \Phi_2) = \operatorname{xnf}(\Phi_1) \wedge \operatorname{xnf}(\Phi_2)$*;*
3. $\operatorname{xnf}(\Phi_1 \vee \Phi_2) = \operatorname{xnf}(\Phi_1) \vee \operatorname{xnf}(\Phi_2)$*;*
4. $\operatorname{xnf}(\Phi_1 \,\mathcal{U}\, \Phi_2) = \operatorname{xnf}(\Phi_2) \vee (\operatorname{xnf}(\Phi_1) \wedge \mathcal{X}(\Phi_1 \,\mathcal{U}\, \Phi_2))$*;*
5. $\operatorname{xnf}(\Phi_1 \,\mathcal{R}\, \Phi_2) = \operatorname{xnf}(\Phi_2) \wedge (\operatorname{xnf}(\Phi_1) \vee \mathcal{X}(\Phi_1 \,\mathcal{R}\, \Phi_2))$.

It can be shown that Φ and $\operatorname{xnf}(\Phi)$ are equivalent:

Theorem 3. *Let Φ be a temporal formula, w a world, and z and z' finite traces. Then $w, z, z' \models \Phi$ iff $w, z, z' \models \operatorname{xnf}(\Phi)$.*

For a propositional assignment P of Φ^p in XNF, we define $L(P) = \{l \mid l \in P \text{ is a literal other than } (\neg)\, \mathit{Tail}\,\}$, $X(P) = \{\theta \mid \mathcal{X}\,\theta \in P\}$, and $T(P) = \top$ if $\mathit{Tail} \in P$ and $T(P) = \bot$ otherwise.

XNF allows us to track the partial satisfaction of a temporal formula over a trace. After each action, we will determine each satisfying assignment P such that $L(P)$ is satisfied by the current state and we will track $X(P)$ in the remaining trace. We will use this in the following to construct a game arena that tracks the satisfaction of a temporal formula Φ.

4 Approach

Our goal is to determine an execution of a given GOLOG program that satisfies the given temporal formula, for all possible environment behaviors. The controller must determine which actions to execute; more specifically, which branch to follow in all nondeterministic choices of the program, while not restricting the environment in its actions. Formally, our goal is to find a successful policy, defined as follows:

Definition 6 (Policy). *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a GOLOG program and $\mathcal{A} = \mathcal{A}_C \dot\cup \mathcal{A}_E$ a partition of the actions $\mathcal{A}$ of $\mathcal{G}$ into controllable and environment actions. A* policy *is a partial mapping $\pi : \mathcal{W} \times \mathcal{Z} \times \operatorname{sub}(\delta) \to 2^{\mathcal{A}}$ such that:*

1. if $w \models \mathcal{D}$, then π is defined on $(w, \langle\rangle, \delta)$;
2. if $\alpha \in \pi(w, z, \rho)$, then $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot \alpha, \rho'\rangle$ for some $\rho' \in \operatorname{sub}(\delta)$;
3. if $\alpha \in \pi(w, z, \rho)$ and $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot \alpha, \rho'\rangle$, then π is defined on $(w, z \cdot \alpha, \rho')$;
4. if $\alpha \in \mathcal{A}_E$ and $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot \alpha, \rho'\rangle$ for some $\rho' \in \operatorname{sub}(\delta)$, then $\alpha \in \pi(w, z, \rho)$;
5. if $\pi(w, z, \rho) = \emptyset$, then $\langle z, \rho\rangle \in \operatorname{Fin}(w)$.

Intuitively, a policy chooses a subset $\pi(w, z, \rho)$ from all possible actions in the current configuration $\langle z, \rho\rangle$ and world w. From this subset, the environment then chooses one action to be executed. The agent's choices are restricted: Every possible environment action must be selected, hence the agent can never limit the environment's choices.

A policy π induces a set of traces $\|\pi\|_w$ in world w, where $z = \langle \alpha_1, \ldots, \alpha_n\rangle \in \|\pi\|_w$ if there are $\rho_1, \ldots, \rho_n$

[2] Proofs can be found in the appendix.

such that (1) $\langle\langle\rangle,\delta\rangle \xrightarrow{w} \langle z[..1],\rho_1\rangle \xrightarrow{w} \cdots \xrightarrow{w} \langle z,\rho_n\rangle$; (2) $\alpha_{i+1} \in \pi(w,z[..i],\rho_i)$; and (3) $\pi(w,z,\rho_n) \subseteq \mathcal{A}_E$ and $\langle z,\rho_n\rangle \in \mathrm{Fin}(w)$. Hence, the environment may choose to terminate the execution if $\langle z,\rho\rangle$ is a final configuration and the agent chose no further actions to execute. Note that by definition, a policy is a restriction of the program execution, i.e., $\|\pi\|_w \subseteq \|\delta\|_w$. We call a policy *terminating* if for every infinite sequence of π-compatible configurations $\langle\langle\rangle,\delta\rangle, \langle z_1,\rho_1\rangle, \langle z_2,\rho_2\rangle, \ldots$ and for every i, there is a $j \geq i$ such that $\pi(w,z_j,\rho_j) \subseteq \mathcal{A}_E$ and $\langle z_j,\rho_j\rangle \in \mathrm{Fin}(w)$. Intuitively, a terminating policy ensures that at any point of the execution trace, there is some future final configuration where the policy does not choose any agent actions and hence the environment may terminate. A policy may still result in an infinite trace if the environment continues to select actions indefinitely. However, we exclude those from consideration as we assume that the environment eventually stops. We can now formalize our goal:

Definition 7 (Synthesis Problem)**.** *Given a* GOLOG *program* $\mathcal{G} = (\mathcal{D},\delta)$ *and a temporal formula* Φ*, find a* terminating policy π *for* $\mathcal{G}$ *that* satisfies Φ*, i.e., for every world* w *with* $w \models \mathcal{D}$ *and every* $z \in \|\pi\|_w$*, it holds that* $w,\langle\rangle,z \models \Phi$*.*

We note that it is in general undecidable to determine whether a satisfying policy exists. In (Zarrieß and Claßen 2014a, 2016) it was shown that the related verification problem (a special case of the synthesis problem) becomes decidable if (1) C^2 is used as base logic, (2) successor state axioms are acyclic, and (3) "pick operators" are disallowed, i.e., all actions in the program are ground. Furthermore, dropping any of these three restrictions while maintaining the other two immediately leads to undecidability: for (1) this is due to the undecidability of FOL, and for (2) and (3) due to the possibility of reducing the halting problem for Turing machines to the verification problem.

In the following, applying the same three restrictions, we describe a sound and complete method for determining a terminating policy π that satisfies Φ. We will do so by constructing a finite game arena $\mathbb{A}^{\Phi}_{\mathcal{G}}$ that captures the possible program executions while tracking the satisfaction of Φ. Once we have constructed $\mathbb{A}^{\Phi}_{\mathcal{G}}$, we can use a game-theoretic approach to determine a terminating policy that satisfies Φ. However, as both the number of worlds satisfying $\mathcal{D}$ and the number of reachable program configurations is generally infinite, we first need to construct a finite abstraction based on *characteristic graphs* and *types*.

Characteristic Graphs

Characteristic graphs (Claßen and Lakemeyer 2008) provide a finite encoding of the reachable program configurations. In such a graph, the nodes correspond to programs ρ, intuitively representing what remains to be executed, while an edge $\rho \xrightarrow{\alpha:\psi} \rho'$ encodes that a transition is possible from ρ to ρ' through action α, if formula ψ holds. In addition, each program ρ has an associated *termination condition* $\varphi(\rho)$, in the form of a fluent formula.

Definition 8 (Characteristic Graph)**.** *Given a program expression* δ*, the* termination condition $\varphi(\delta)$ *of* δ *is a fluent formula inductively defined as follows:*

$$\begin{aligned} \varphi(\alpha) &= \bot \text{ if } \alpha \text{ is a ground action} & \varphi(\phi?) &= \phi \\ \varphi(\delta_1;\delta_2) &= \varphi(\delta_1) \wedge \varphi(\delta_2) & \varphi(\delta_1|\delta_2) &= \varphi(\delta_1) \vee \varphi(\delta_2) \\ \varphi(\delta_1\|\delta_2) &= \varphi(\delta_1) \wedge \varphi(\delta_2) & \varphi(\delta^*) &= \top \end{aligned}$$

For any program expression δ*, the set of* outgoing edges $\delta \xrightarrow{\alpha:\psi} \rho$ *with action* α *and guard condition* ψ *to resulting program* ρ *is defined inductively as follows:*

- $\alpha \xrightarrow{\alpha:\top} \mathrm{nil}$*, if* α *is a primitive action;*
- $(\delta_1;\delta_2) \xrightarrow{\alpha:\psi} (\rho;\delta_2)$*, if* $\delta_1 \xrightarrow{\alpha:\psi} \rho$*;*
- $(\delta_1;\delta_2) \xrightarrow{\alpha:\varphi(\delta_1)\wedge\psi} \rho$*, if* $\delta_2 \xrightarrow{\alpha:\psi} \rho$*;*
- $(\delta_1|\delta_2) \xrightarrow{\alpha:\psi} \rho$*, if* $\delta_1 \xrightarrow{\alpha:\psi} \rho$ *or* $\delta_2 \xrightarrow{\alpha:\psi} \rho$*;*
- $(\delta_1\|\delta_2) \xrightarrow{\alpha:\psi} (\rho\|\delta_2)$*, if* $\delta_1 \xrightarrow{\alpha:\psi} \rho$*;*
- $(\delta_1\|\delta_2) \xrightarrow{\alpha:\psi} (\delta_1\|\rho)$*, if* $\delta_2 \xrightarrow{\alpha:\psi} \rho$*;*
- $\delta^* \xrightarrow{\alpha:\psi} (\rho;\delta^*)$*, if* $\delta \xrightarrow{\alpha:\psi} \rho$*.*

For any program expression δ*, the corresponding* characteristic graph *is given by* $\mathcal{C}_\delta = \langle v_0, V, E\rangle$*, where* $v_0 = \delta$ *(initial node), and the nodes* V *and edges* E *are the smallest sets such that (1)* $\delta \in V$*; and (2) if* $\delta' \in V$ *and* $\delta' \xrightarrow{\alpha:\psi} \delta''$*, then* $\delta'' \in V$ *and* $\delta' \xrightarrow{\alpha:\psi} \delta'' \in E$*.*

We denote the set V with $\mathrm{sub}(\delta)$, the *subprograms reachable from* δ. We note (Claßen and Lakemeyer 2008):

Lemma 4. *For any program* δ*,* $\mathcal{C}_\delta$ *is finite, and for any world* w*, situation* z*, and* $\delta' \in \mathrm{sub}(\delta)$*, it holds that (1)* $\langle z,\delta'\rangle \in \mathrm{Fin}(w)$ *iff* $w,z \models \varphi(\delta')$*; and (2)* $\langle z,\delta'\rangle \xrightarrow{w} \langle z\cdot\alpha,\delta''\rangle$ *iff* $\delta' \xrightarrow{\alpha:\psi} \delta''$ *and* $w,z \models \psi$*.*

Characteristic graphs therefore exactly capture the program transition semantics. We can hence use them as finite abstractions of the reachable program configurations. Also, using characteristic graphs, there is a (simple to test) sufficient condition for programs being situation-determined:

Lemma 5. *If every ground action* α *occurs at most once among the outgoing edges of every node in* $\mathcal{C}_\delta$*, then* δ *is situation-determined.*

Types

With characteristic graphs, we already have a finite representation of the possible program configurations. However, there are additional sources of infiniteness. For one, during the execution of a program, we may accumulate infinitely many effects. Second, there are infinitely many possible worlds that satisfy the BAT $\mathcal{D}$. However, for acyclic BATs, it has been shown that the set of possible effects is finite, and that the set of worlds that satisfy $\mathcal{D}$ can be represented by a finite set of equivalence classes, so-called *types of worlds* (Zarrieß and Claßen 2016). We will now describe how to construct types for a given BAT $\mathcal{D}$.

As our programs may only mention finitely many ground actions, we can rewrite the SSAs of an acyclic BAT by grounding the effects. This is done by replacing each SSA for a fluent $F(\vec{x})$ by a set of instantiated formulas, one for each $\alpha \in \mathcal{A}$, of the form $\square[\alpha]F(\vec{x}) \equiv (\gamma^+_F)^a_\alpha \vee F(\vec{x}) \wedge$

$\neg(\gamma_F^-)^a_\alpha$. As each $\gamma_F^\pm$ is a disjunction of formulas of the form $\exists\vec{y}.(a = A(\vec{v}) \wedge \epsilon \wedge \kappa)$, the resulting positive effect condition $(\gamma_F^+)^a_\alpha$ is equivalent to a disjunction of the form $\epsilon_1 \wedge \kappa_1 \vee \ldots \vee \epsilon_n \wedge \kappa_n$, which allows us to write $(\gamma_F^+)^a_\alpha$ as a set of pairs $(\gamma_F^+)^a_\alpha = \bigvee_i \{(\epsilon_i, \kappa_i)\}_i$. We write $(\epsilon, \kappa) \in (\gamma_F^+)^a_\alpha$ if (ϵ, κ) occurs in the disjunction (analogously for $(\gamma_F^-)^a_\alpha$). For a fluent F, the set of positive effect descriptors is then defined as $\mathsf{eff}^+_{\mathcal{A}}(F) := \{\varepsilon \mid (\varepsilon, \kappa) \in (\gamma_F^+)^a_\alpha$ for some $\alpha \in \mathcal{A}\}$, and similarly for negative effect descriptors $\mathsf{eff}^-_{\mathcal{A}}(F)$. Hence, we can write a set of effects E as a set of pairs $E = \{\langle F_i^\pm, \varepsilon_i\rangle\}_i$, where $\varepsilon_i \in \mathsf{eff}^+_{\mathcal{A}}(F)$ or $\varepsilon_i \in \mathsf{eff}^-_{\mathcal{A}}(F)$. We define a variant of *regression* on such a set of effects:

Definition 9 (Regression)**.** *Let E be a set of effects and φ a C^2 fluent formula. The* regression of φ through E*, denoted by $\mathcal{R}[E, \varphi]$ is a C^2 fluent formula obtained from φ by replacing each occurrence of a fluent $F(\vec{v})$ in φ by the formula $F(\vec{v}) \wedge \bigwedge_{\langle F^-, \varepsilon\rangle \in E} \neg \varepsilon^{\vec{x}}_{\vec{v}} \vee \bigvee_{\langle F^+, \varepsilon\rangle \in E} \varepsilon^{\vec{x}}_{\vec{v}}$.*

Furthermore, in an acyclic BAT, the effect descriptor ε of a fluent F with $\mathrm{fd}(F) = i$ may only mention fluents with depth strictly smaller than i. Thus, when regressing the effect descriptor ε of a fluent F with $\mathrm{fd}(F) = i$, only effects on fluents with depth strictly smaller than i are relevant. Hence, for a GOLOG program $\mathcal{G} = (\mathcal{D}, \delta)$ with an acyclic BAT $\mathcal{D}$, there are only finitely many possible effects that can be generated by action sequences from $\mathcal{A}$. We denote the *set of all relevant effects* on all fluents with depth $\leq j$ with $j = 0, \ldots, \mathsf{fd}(\mathcal{D})$ by $\mathfrak{E}^{\mathcal{D},\mathcal{A}}_j$, and define it as follows:

$$
\begin{aligned}
\mathfrak{E}^{\mathcal{D},\mathcal{A}}_0 &\doteq \{\langle F^\pm, \varepsilon\rangle \mid \mathsf{fd}_{\mathcal{D}}(F) = 0, \varepsilon \in \mathsf{eff}^-_{\mathcal{A}}(F) \cup \mathsf{eff}^+_{\mathcal{A}}(F)\}\\
\mathfrak{E}^{\mathcal{D},\mathcal{A}}_i &\doteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}_{i-1}\\
&\qquad \cup \{\langle F^-, \mathcal{R}[\mathsf{E}, \varepsilon]\rangle \mid \mathsf{fd}_{\mathcal{D}}(F) = i, \varepsilon \in \mathsf{eff}^-_{\mathcal{A}}(F), \mathsf{E} \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}_{i-1}\}\\
&\qquad \cup \{\langle F^+, \Xi\rangle \mid \mathsf{fd}_{\mathcal{D}}(F) = i, \phi \in \mathsf{eff}^+_{\mathcal{A}}(F), \mathsf{E} \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}_{i-1},\\
&\qquad\qquad\qquad X \subseteq \mathsf{eff}^-_{\mathcal{A}}(F) \times 2^{\mathfrak{E}^{\mathcal{D},\mathcal{A}}_{i-1}}\}\\
&\qquad \text{with } \Xi \doteq \Big(\mathcal{R}[\mathsf{E}, \phi] \wedge \bigwedge_{(\varepsilon, \mathsf{E}') \in X} \neg\mathcal{R}[\mathsf{E}', \varepsilon]\Big)\\
\mathfrak{E}^{\mathcal{D},\mathcal{A}} &\doteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}_n \text{ with } \mathsf{fd}(\mathcal{D}) = n
\end{aligned}
$$

Additionally, we define the *context of a program* $\mathcal{C}(\mathcal{G})$ as the set of relevant C^2-fluent sentences that occur in the initial theory, in context conditions of the instantiated SSAs, in guards and termination conditions of the characteristic graph, and in the temporal formula, and we ensure that the context is closed under negation. We can now define *types*:

Definition 10 (Type of a world)**.** *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a* GOLOG *program with an acyclic BAT $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{post}$ w.r.t. a finite set of ground actions $\mathcal{A}$. Furthermore, let $\mathcal{C}(\mathcal{G})$ be the context of $\mathcal{G}$ and $\mathfrak{E}^{\mathcal{D},\mathcal{A}}$ the set of all relevant effects. The* set of all type elements *is given by* $\mathrm{TE}(\mathcal{G}) \doteq \{(\psi, E) \mid \psi \in \mathcal{C}(\mathcal{G}), E \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}\}$*. A* type w.r.t. $\mathcal{G}$ *is a set $\tau \subseteq \mathrm{TE}(\mathcal{G})$ that satisfies:*

1. *For all $\psi \in \mathcal{C}(\mathcal{G})$ and all $E \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}$ it holds that either $(\psi, E) \in \tau$ or $(\neg\psi, E) \in \tau$;*
2. *There exists a world $w \in \mathcal{W}$ such that $w \models \mathcal{D}_0 \cup \{\mathcal{R}[E, \psi] \mid (\psi, E) \in \tau\}$.*

The set of all types w.r.t. $\mathcal{G}$ *is denoted by* $\mathrm{Types}(\mathcal{G})$*. The* type of a world $w \in \mathcal{W}$ w.r.t. $\mathcal{G}$ *is given by* $\mathrm{type}(w) \doteq \{(\psi, E) \in \mathrm{TE}(\mathcal{G}) \mid w \models \mathcal{R}[E, \psi]\}$.

Definition 11. *Let $\tau \in \mathrm{Types}(\mathcal{G})$, $E \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}$, and $\alpha \in \mathcal{A}$. The* effects of executing α in (τ, E) *are given by*

$$
\begin{aligned}
\mathcal{E}_{\mathcal{D}}(\tau, E, \alpha) \doteq {} & \{\langle F^+, \varepsilon\rangle \mid \exists(\varepsilon, \kappa) \in (\gamma_F^+)^a_\alpha \textit{ s.t. } (\kappa, E) \in \tau\} \cup\\
& \{\langle F^-, \varepsilon\rangle \mid \exists(\varepsilon, \kappa) \in (\gamma_F^-)^a_\alpha \textit{ s.t. } (\kappa, E) \in \tau\}
\end{aligned}
$$

Definition 12. *Let φ be a C^2 fluent formula and E_0 and E_1 two sets of effects. The accumulation $E_0 \triangleright E_1$ of E_0 and E_1 is defined as follows:*

$$
\begin{aligned}
&E_0 \triangleright E_1 \doteq \{\langle F^\pm, \mathcal{R}[E_0, \varphi]\rangle \mid \langle F^\pm, \varphi\rangle \in E_1\}\\
&\cup \{\langle F^+, (\varphi \wedge \bigwedge_{\langle F^-, \varphi\rangle \in E_1} \neg\mathcal{R}[E_0, \varphi'])\rangle \mid \langle F^+, \varphi\rangle \in E_0\} \cup \{\langle F^-, \varphi\rangle \in E_0\}
\end{aligned}
$$

Let w be a world with $w \models \mathcal{D}$, $\mathrm{type}(w) = \tau$, and $z = \langle\alpha_1, \ldots, \alpha_n\rangle$ a trace. We define $E_0 \doteq \emptyset$ and $E_i \doteq E_{i-1} \triangleright \mathcal{E}_{\mathcal{D}}(\tau, E, \alpha)$ for $1 \leq i \leq n$. We also write E_z for the effect E_n that is generated by executing $z = \langle\alpha_1, \ldots, \alpha_n\rangle$ in w. The following theorem shows the correctness of the construction (Zarrieß and Claßen 2016):

Theorem 6. *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a* GOLOG *program, w a world with $w \models \mathcal{D}$, and $z \in \mathcal{A}^*$ a trace. Then $w, z \models \phi$ iff $(\phi, E_z) \in \mathrm{type}(w)$.*

Hence, types provide a finite representation of the worlds satisfying $\mathcal{D}$ and all effects that can be generated by δ.

Game Arena

With types, characteristic graphs, and XNF formulas, we can define a game arena $\mathbb{A}^\Phi_{\mathcal{G}}$ that captures the possible executions of a program $\mathcal{G}$ while tracking the satisfaction of Φ:

Definition 13. *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a* GOLOG *program and Φ a temporal formula. The game arena $\mathbb{A}^\Phi_{\mathcal{G}} = (\mathcal{S}, \mathcal{S}_0, \to, \mathcal{S}_F, \mathcal{S}_A)$ for $\mathcal{G}$ and Φ is defined as follows:*

- *Each state $s \in \mathcal{S}$ is of the form $s = (\tau, E, A, \rho)$ where*
  1. *$\tau \in \mathrm{Types}(\mathcal{G})$;*
  2. *$\rho \in \mathrm{sub}(\delta)$ is a node of the characteristic graph;*
  3. *$E \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}$;*
  4. *$A = \{(\chi_i, \theta_i)\}_i$, where $\chi_i \subseteq \mathrm{cl}(\Phi)$, $\theta_i \in \{\top, \bot\}$.*
- *A state $s = (\tau, E, A, \rho)$ is an initial state $s \in \mathcal{S}_0$ if*
  1. *$\tau = \mathrm{type}(w)$ for some w with $w \models \mathcal{D}$;*
  2. *$\rho = \delta$ is the initial program expression;*
  3. *$E = \emptyset$;*
  4. *$(\chi, \theta) \in A$ iff there is a propositional assignment P of $\mathrm{xnf}(\Phi)^p$ such that $\{(\psi, E) \mid \psi \in L(P)\} \subseteq \tau$, $\chi = X(P)$, and $\theta = T(P)$.*
- *There is a transition $s_1 \xrightarrow{\alpha} s_2$ from $s_1 = (\tau, E_1, A_1, \rho_1)$ to $s_2 = (\tau, E_2, A_2, \rho_2)$ if*
  1. *there is an edge $\rho_1 \xrightarrow{\alpha:\psi} \rho_2$ in $\mathcal{C}_\delta$ with $(\psi, E_1) \in \tau$;*
  2. *$E_2 = E_1 \triangleright \mathcal{E}_{\mathcal{D}}(\tau, E_1, \alpha)$;*
  3. *$(\chi_2, \theta_2) \in A_2$ if there is a propositional assignment P of $\mathrm{xnf}(\bigwedge \chi_1^p)$ for some $(\chi_1, \theta_1) \in A_1$ such that (1) $\theta_1 = \bot$, (2) $\{(\psi, E_2) \mid \psi \in L(P)\} \subseteq \tau$, (3) $\chi_2 = X(P)$, and (4) $\theta_2 = T(P)$.*

A state $s = (\tau, E, A, \rho)$ is final *if $(\varphi(\rho), E) \in \tau$ and* accepting *if $(\emptyset, \top) \in A$. We denote the set of all final states with $\mathcal{S}_F$ and the set of all accepting states with $\mathcal{S}_A$. We also write $\mathrm{type}(s) = \tau$ for the type of the world in s.*

Each state consists of (1) a type τ, representing an equivalence class of worlds; (2) a node ρ from the characteristic graph, capturing the remaining program and its termination condition; (3) a set of accumulated effects E; and (4) a set of temporal formulas A, which must be satisfied in the remaining execution to fulfill the specification Φ. The initial states are those states with the initial program expression and no accumulated effects. Furthermore, regarding the temporal formula Φ and A of an initial state, we first compute all the propositional assignments of $\mathrm{xnf}(\Phi)^p$. For each assignment P, we check whether the local part $L(P)$ is satisfied by the state. If so, the pair $(\chi, \theta) = (X(P), T(P))$ is added to A, which intuitively states that χ must be satisfied in the future and the program should terminate if θ is true. For transitions, we first check whether there is an edge in the characteristic graph that allows the execution of the next action. If so, we accumulate the effects and check whether there is a propositional assignment of $\mathrm{xnf}(\bigwedge \chi_1^p)$ for some $(\chi_1, \theta_1) \in A_1$ that allows the satisfaction of the temporal formulas in A_2. Similar to the initial states, we do so by checking whether the local part $L(P)$ is satisfied by the current state and tracking $X(P)$ and $T(P)$ in the future.

By definition, a state is *final* if the program may terminate and it is *accepting* if Φ is satisfied. Also note that $\mathbb{A}_{\mathcal{G}}^{\Phi}$ is finite as both types and reachable sub-programs are finite. It is also deterministic, as $\mathcal{G}$ is situation-determined and for action successors, the satisfying assignments of $\mathrm{xnf}(\Phi)^p$ are collected in a single successor state.

We can show that $\mathbb{A}_{\mathcal{G}}^{\Phi}$ indeed corresponds to the executions of $\mathcal{G}$ while tracking the satisfaction of Φ:

Theorem 7. *Every execution of $\mathcal{G} = (\mathcal{D}, \delta)$ satisfies Φ iff every reachable final state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$ is accepting.*

This provides us a decidable method for verifying an $\mathrm{LTL_f}$ property Φ against a GOLOG program $\mathcal{G}$. However, the goal is to determine a policy that executes $\mathcal{G}$ while satisfying Φ.

Synthesis

Above, we have described a finite game arena $\mathbb{A}_{\mathcal{G}}^{\Phi}$ that captures the executions of a program $\mathcal{G}$ while tracking the satisfaction of a given $\mathrm{LTL_f}$ formula Φ. In the following, we use a game-theoretic approach on $\mathbb{A}_{\mathcal{G}}^{\Phi}$ to determine a policy that successfully executes $\mathcal{G}$ while satisfying Φ. We do so by defining a game between two players, the system and the environment, that play on $\mathbb{A}_{\mathcal{G}}^{\Phi}$. We start by defining a *strategy*, which intuitively translates the conditions on a policy to the game arena $\mathbb{A}_{\mathcal{G}}^{\Phi}$:

Definition 14 (Strategy)**.** *Let $\mathbb{A}_{\mathcal{G}}^{\Phi}$ be the game arena for some* GOLOG *program $\mathcal{G}$ and temporal formula Φ. Let $s \in \mathcal{S}$ be a state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$. A set of actions $U \subseteq \mathcal{A}$ is* valid *in s under the following conditions:*

1. *if $\alpha \in U$, then there is an edge $s \xrightarrow{\alpha} s'$ for some $s' \in \mathcal{S}$*
2. *if $s \xrightarrow{\alpha} s'$ for some $\alpha \in \mathcal{A}_E$ and $s' \in \mathcal{S}$, then $\alpha \in U$*
3. *if $U = \emptyset$, then s is a final state*

Algorithm 1: Computing a strategy from $\mathbb{A}_{\mathcal{G}}^{\Phi}$

```
for all $H \in 2^{\mathcal{S}_F \cap \mathcal{S}_A}$ do
    $G \leftarrow H$; $R \leftarrow \{s \in G \mid \mathrm{Succ}_E(s) = \emptyset\}$; $\sigma \leftarrow \emptyset$
    $Q \leftarrow \{s \in \mathcal{S} \mid \mathrm{Succ}(s) \cap G \neq \emptyset\}$
    while $Q \neq \emptyset$ do
        $s \leftarrow \textsc{pop}(Q)$
        if $s \in \mathcal{S}_F \setminus \mathcal{S}_A \wedge \mathrm{Succ}_C(s) = \emptyset$ then continue
        if $s \in R$ then continue
        if $\mathrm{Succ}_E(s) \neq \emptyset \wedge \forall s' \in \mathrm{Succ}_E(s) : s' \in G \vee$
          $\mathrm{Succ}_E(s) = \emptyset \wedge \exists s' \in \mathrm{Succ}_C(s) : s' \in G$ then
            $G \leftarrow G \cup \{s\}$; $R \leftarrow R \cup \{s\}$
            if $s \in \mathcal{S}_F \cap \mathcal{S}_A$ then
                $\sigma(s) \leftarrow \{\alpha \mid \exists s' \in \mathrm{Succ}_E(s).\ s \xrightarrow{\alpha} s'\}$
            else $\sigma(s) \leftarrow \{\alpha \mid \exists s' \in G.\ s \xrightarrow{\alpha} s'\}$
            $Q \leftarrow Q \cup \{s' \mid s \in \mathrm{Succ}(s')\}$
    if $H \cup \mathcal{S}_0 \subseteq R$ then return $\sigma$
```

A strategy *in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ is a partial function $\sigma : \mathcal{S} \to 2^{\mathcal{A}}$ such that:*

1. *σ is defined on every initial state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$*
2. *if σ is defined on $s \in \mathcal{S}$, then $\sigma(s)$ is valid in s*
3. *if σ is defined on $s \in \mathcal{S}$, $\alpha \in \sigma(s)$, and $s \xrightarrow{\alpha} s'$ for some $s' \in \mathcal{S}$, then σ is defined on s'*

We also write $s \xrightarrow{\sigma} s'$ if there is $\alpha \in \sigma(s)$ such that $s \xrightarrow{\alpha} s'$. A strategy σ induces a set of plays $\mathrm{plays}(\sigma)$, which are those paths in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ consistent with σ. Formally, $p = \langle s_0, \ldots, s_n \rangle \in \mathrm{plays}(\sigma)$ if

1. *s_0 is an initial state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$*
2. *for each i, $s_i \xrightarrow{\sigma} s_{i+1}$*
3. *$\sigma(s_n) \subseteq \mathcal{A}_E$ and s_n is a final state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$*

A play is winning *if it ends in an accepting state. A strategy σ is* winning *if every play $p \in \mathrm{plays}(\sigma)$ is winning. We call a strategy σ* terminating *if for every infinite sequence of states $s_0, s_1, \ldots$ with $s_k \xrightarrow{\sigma} s_{k+1}$ for every k, it holds that for every i, there is a $j \geq i$ such that $\sigma(s_j) \subseteq \mathcal{A}_E$ and s_j is final.*

Proposition 8. *There is a terminating and winning strategy σ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ if and only if there exists a terminating policy π for $\mathcal{G}$ that satisfies Φ.*

Hence, we need to determine a terminating and winning strategy in $\mathbb{A}_{\mathcal{G}}^{\Phi}$. In principle, this can be done with backward search starting in a set of *good* states and then checking whether the agent can force every play to end in a good state. However, not every final and accepting state is necessarily good, as the environment may force a play from this state that ends in a non-accepting state. On the other hand, every winning play must end in an accepting state, so if a strategy exists, there must be an enforceable set of final and accepting states. Hence, we can guess which final and accepting states are enforceable and then check if there is indeed a strategy that can force every play to end in those states.

This approach is formalized in Algorithm 1. It starts with a hypothesis $H \subseteq \mathcal{S}_F \cap \mathcal{S}_A$ of good states G and tracks the states R that can reach G. It then iteratively checks the predecessors of all states in G whether the agent can force the play to end in G. This is the case if all environment successors $\mathrm{Succ}_E(s)$ are in G or if there is a control successor

$\mathrm{Succ}_C(s)$ in G. If a state is found that can be forced to end in G, it is added to G and R and σ are updated accordingly. Finally, if all states of H and all initial states $\mathcal{S}_0$ can in fact reach G, then σ is a winning and terminating strategy:

Theorem 9. *Algorithm 1 terminates and returns a winning and terminating strategy if one exists.*

5 Experiments

We implemented the method (Claßen and Hofmann 2025) in the Prolog-based Golog interpreter `vergo` (Claßen 2018), that, different from other implementations, uses full FOL as base logic, where an embedded theorem prover (Schulz, Cruanes, and Vukmirovic 2019) is used for reasoning tasks such as deciding entailment and consistency. The system contains optimizations for handling FO expressions, in particular an FO variant of binary decision diagrams. We evaluated the method on two domains, a dishwasher robot and a warehouse robot, that we will be described in detail below. All experiments were conducted on an Intel® Core™ i5-7300U @2.60GHz with 8GB of RAM, running Debian 12 with WSL2 under Windows 10, using SWI-Prolog 9.0.4 and version 3.2 of the E theorem prover.

Incremental Construction

In our implementation, the construction of the abstract game arena follows closely Definition 13. However, the construction is done in an incremental fashion, where only the relevant and reachable parts are actually materialized. This is achieved by keeping the types as general as possible, and only including additional formulas once they are needed. More specifically, the method works by iterating the following steps, until no more changes occur:

Initialize: Create initial states $(\tau, \emptyset, A, \delta)$, where types τ are constructed only from formulas in $\mathcal{D}_0$ and literals $L(P)$ of propositional assignments over A.

Split: If there is a state (τ, E, A, ρ) that does not entail a truth value for some required condition ψ (the transition condition for an action α, the termination condition $\varphi(\rho)$, the condition κ of an effect, or a literal $l \in L(P)$ of a propositional assignment over A), then create two copies of all states and transitions, where one includes ψ and the other includes $\neg\psi$ into τ, discarding states with inconsistent τ.

Expand: If a state $s = (\tau, E, A, \rho)$ admits an action α, create the successor state s' and the transition $s \xrightarrow{\alpha} s'$.

We represent τ directly by the regressed versions of formulas to avoid having to regress them repeatedly. The construction also stops in states where $A = \emptyset$, since the corresponding traces can never satisfy the input property.

Dishwasher Robot

The first domain is inspired by the dishwasher robot example used in (Claßen et al. 2014), but adds additional fluents. A robot can move between a number of rooms and the kitchen, load (an arbitrary number of) dirty dishes onto itself, and unload dishes it carries into the dishwasher. The environment has actions that represent used dishes being placed in arbitrary rooms. Every dish can only be used once in this fashion. The basic action theory, program, and temporal specification are specified below.

Algorithm 2: The program for the dishwasher robot

```
loop
    while $\exists x.\, OnRobot(x)$ do $\pi x : \{d_1, d_2\}.\, \underline{unload(x)}$
    $\pi y : \{r_1, r_2\}.\, \underline{goto(y)}$;
    while $\exists x.\, DirtyDish(x, y)$ do $\pi x : \{d_1, d_2\}.\, \underline{load(x, y)}$
    $\underline{goto(kitchen)}$
||
loop $\pi x : \{d_1, d_2\}, y : \{r_1, r_2\}.\, \underline{addDish(x, y)}$
```

R	D	Nodes (TS)	Edges (TS)	Nodes (St)	Edges (St)	Time [s]
1	1	22	25	16	19	2.6
1	2	150	203	128	176	155.4
1	3	–	–	–	–	–
2	1	109	168	87	110	69.0
2	2	–	–	–	–	–
3	1	483	857	413	543	1885.7
3	2	–	–	–	–	–

Table 1: Evaluation Results for the Dish Robot Domain

Initial situation:

$$Dish(x) \equiv (x = d_1 \lor x = d_2),\ Room(x) \equiv (x = r_1 \lor x = r_2)$$
$$\forall x.\, At(x) \equiv x = kitchen$$
$$\forall x.\, New(x) \equiv Dish(x) \land \forall y.\, \neg DirtyDish(x, y) \land \neg OnRobot(x)$$
$$OnRobot(x) \supset Dish(x) \land \neg\exists y DirtyDish(x, y)$$
$$DirtyDish(x, y) \supset Dish(x) \land Room(y) \land \neg OnRobot(x)$$

Precondition axioms:

$$\Box\,\mathrm{Poss}(load(x, y)) \equiv DirtyDish(x, y) \land At(y)$$
$$\Box\,\mathrm{Poss}(unload(x)) \equiv OnRobot(x) \land At(kitchen)$$
$$\Box\,\mathrm{Poss}(addDish(x, y)) \equiv New(x) \land Room(y)$$
$$\Box\,\mathrm{Poss}(goto(x)) \equiv Room(x) \lor x = kitchen$$

Successor state axioms:

$$\Box[a] DirtyDish(x, y) \equiv a = addDish(x, y)$$
$$\lor\ DirtyDish(x, y) \land a \neq load(x, y)$$
$$\Box[a] OnRobot(x) \equiv \exists y.\, a = load(x, y)$$
$$\lor\ OnRobot(x) \land a \neq unload(x)$$
$$\Box[a] New(x) \equiv New(x) \land \neg\exists y.\, a = addDish(x, y)$$
$$\Box[a] At(x) \equiv a = goto(x) \lor At(x) \land \neg\exists y. a = goto(y)$$

Program: The program is shown in Algorithm 2. It is to be understood as being *precondition extended*, i.e., an underlined action $\underline{A(\vec{o})}$ stands for $\pi?; A(\vec{o})$, where π is the right-hand side of the precondition axiom for A, instantiated by $\vec{o}$. For better readability, δ^* is written as **loop** δ.

Specification: $\mathcal{F}\,\mathcal{G}\,\neg\exists x, y.\, DirtyDish(x, y)$

Results: Table 1 summarizes the experimental results on the dishwasher domain. Here, *R* and *D* denote the number of rooms and dishes, respectively, *Nodes (TS)* and *Edges (TS)* refer to the number of nodes and edges of the resulting transition system, while *Nodes (St)* and *Edges (St)* denote the corresponding metrics of the discovered strategy. *Time* indicates the duration in seconds for completing the algorithm, with a timeout set at 120 minutes. As expected, the size of transition system, and the time needed to construct it, grows with additional rooms or dishes. Interestingly, the number of dishes has a bigger impact than the number of rooms. Intuitively, this is because the program contains more choices for dishes than for rooms, which are furthermore nested inside inner loops. Accordingly, adding one more dish results in a more significant blow-up than adding a room.

Warehouse Robot

The second domain is a warehouse robot, adapted from an example in (Claßen and Zarrieß 2017). Here, the robot can move boxes from one shelf of a warehouse to another. The boxes may contain an unknown number of objects, and it is unknown whether and which objects are fragile. Accidentally (i.e., due to the environment's choice), the robot may drop a box, breaking all fragile objects in it, unless the box contains bubble wrap. The robot has the option to put bubble wrap into a box.

Initial situation:

$$\begin{array}{l}
\forall x.\, \mathit{Shelf}(x) \equiv (x = s_1 \lor x = s_2) \\
\forall x.\, \mathit{Box}(x) \equiv (x = b_1 \lor x = b_2) \\
\forall x.\, \exists y \mathit{In}(x, y) \supset \neg \mathit{Shelf}(x) \land \neg \mathit{Box}(x) \\
\exists x.\, \mathit{Wrap}(x) \\
\forall x.\, \neg \mathit{Broken}(x) \land \neg \mathit{Holding}(x) \\
\mathit{RAt}(s_1) \land \forall x.\, \mathit{Box}(x) \supset \mathit{At}(x, s_1) \\
\forall x, y.\, (\mathit{In}(x, y) \land \mathit{Box}(y)) \supset \mathit{At}(x, s_1) \\
\forall y.\, y \neq s_1 \supset \neg \mathit{RAt}(y) \land \forall x.\, \neg \mathit{At}(x, y) \\
\forall x, y.\, \mathit{In}(x, y) \supset \neg \mathit{Wrap}(x)
\end{array}$$

Precondition axioms:

$$\begin{array}{l}
\Box\, \mathrm{Poss}(\mathit{take}(x, y)) \equiv \mathit{At}(x, y) \land \mathit{RAt}(y) \\
\Box\, \mathrm{Poss}(\mathit{move}(x, y)) \equiv \mathit{RAt}(x) \land \mathit{Shelf}(y) \land (x \neq y) \\
\Box\, \mathrm{Poss}(\mathit{put}(x, y)) \equiv \mathit{Holding}(x) \land \mathit{RAt}(y) \\
\Box\, \mathrm{Poss}(\mathit{addWrap}(x)) \equiv \exists y.\, \mathit{RAt}(y) \land \mathit{At}(x, y) \\
\Box\, \mathrm{Poss}(\mathit{drop}(x)) \equiv \mathit{Holding}(x)
\end{array}$$

Successor state axioms:

$$\begin{array}{l}
\Box[a]\mathit{RAt}(y) \equiv \exists x.\, a = \mathit{move}(x, y) \\
\qquad \lor\, \mathit{RAt}(y) \land \neg \exists z. a = \mathit{move}(y, z) \\
\Box[a]\mathit{At}(x, y) \equiv \exists z[a = \mathit{move}(z, y) \land \\
\quad \exists v(\mathit{Holding}(v) \land (v = x \lor \mathit{In}(x, v)))] \\
\quad \lor\, \mathit{At}(x, y) \land \neg \exists z[a = \mathit{move}(y, z) \land \\
\quad \exists v(\mathit{Holding}(v) \land (v = x \lor \mathit{In}(x, v)))] \\
\Box[a]\mathit{Holding}(x) \equiv \exists y.\, a = \mathit{take}(x, y) \\
\qquad \lor\, \mathit{Holding}(x) \land \neg \exists y.\, a = \mathit{put}(x, y) \\
\Box[a]\mathit{Broken}(x) \equiv \exists y.\, a = \mathit{drop}(y) \land \mathit{In}(x, y) \land \mathit{Fragile}(x) \\
\quad \land\, \neg \exists z.\, \mathit{In}(z, y) \land \mathit{Wrap}(z) \lor \mathit{Broken}(x) \\
\Box[a]\mathit{In}(x, y) \equiv a = \mathit{addWrap}(y) \land \mathit{Wrap}(x) \lor \mathit{In}(x, y)
\end{array}$$

Program: The program for the warehouse robot is shown in Algorithm 3. The notation $\delta^?$ stands for an optional execution of δ, and is formally defined as $\delta^? \doteq (\delta \mid \mathrm{nil})$. Note that the choice for putting bubble wrap is up to the robot, but that of the box getting dropped is due to the environment.

Algorithm 3: The program for the warehouse robot

```
loop
    $\pi l_0, l_1 : \{s_1, s_2\}.\ [\underline{move(l_0, l_1)}^?;$
        $\pi b : \{b_1, b_2\}.\ (\underline{wrap(b)}^?; \underline{take(b, s_1)}; \underline{drop(b)}^?;$
            $\pi l_2 : \{s_1, s_2\}.\ \underline{move(l_1, l_2)}; \underline{put(b, l_2)})]$
```

B	Nodes (TS)	Edges (TS)	Nodes (St)	Edges (St)	Time [s]
1	162	162	46	58	11.8
2	7584	7830	2038	2647	13328.9
3	–	–	–	–	–

Table 2: Evaluation Results for the Warehouse Domain

Specification: $\mathcal{F}\, \forall o.\, \mathit{In}(o, b_1) \supset \neg \mathit{Broken}(o) \land \mathit{At}(o, s_2)$

Results: Table 2 presents the results of the experiments on the warehouse robot domain, where B denotes the number of boxes, and the other columns are as before (with a timeout of 240 minutes). As can be seen, the method struggles more with this domain than the previous one, which is due to several reasons. For one, the successor state axioms for the warehouse robot actually exploit the expressivity of the class of acyclic theories more than do the ones for the dishwasher robot. Note that the dishwasher BAT actually falls into the class of *local-effect* theories (Liu and Lakemeyer 2009), a subset of acyclic theories where regression works much simpler (i.e., does not introduce additional quantifiers), and consequently results in less complex formulas. Moreover, the warehouse robot suffers from the same problem that causes the *Gripper* domain to be a challenge in classical planning: There is a number of objects, each of which has to be handled in the same way. For solving the task, the order in which objects are handled is hence irrelevant, yet the system considers all possible permutations, resulting in a blow-up. The problem is amplified by the fact that handling a single box in this domain is a slightly complex task in itself, containing a sequence of actions with several choice points. An interesting avenue for future work would be to improve our method to be able to detect and deal with symmetries of this kind.

6 Conclusion

In this paper, we have presented an approach to the realization of GOLOG programs with uncontrollable actions. We have formulated the realization problem as a synthesis problem, where parts of the program are under the environment's control and the agent needs to determine a policy that realizes the program while satisfying the temporal specification. The presented approach synthesizes policies for $\mathrm{LTL_f}$ specifications on GOLOG programs with first-order action theories that allow for an unbounded number of objects and non-local effects, an expressive and decidable fragment of the situation calculus. We have demonstrated the feasibility of the approach in two example domains. The synthesis method can also be understood as a (restricted) first-order variant of $\mathrm{LTL_f}$ synthesis, where the user may provide a declarative specification of the agent's capabilities along with a partial strategy. Future work could further investigate this relation.

Acknowledgements

The research has been supported by the Alexander von Humboldt Foundation with funds from the Federal Ministry for Education and Research, Germany, by the European Research Council (ERC), Grant agreement No. 885107, and by the Excellence Strategy of the Federal Government and the Länder, Germany.

References

Abadi, M.; Lamport, L.; and Wolper, P. 1989. Realizable and Unrealizable Specifications of Reactive Systems. In *Automata, Languages and Programming*, 1–17. Berlin, Heidelberg: Springer.

Bacchus, F.; and Kabanza, F. 1998. Planning for Temporally Extended Goals. *Annals of Mathematics and Artificial Intelligence*, 22(1-2): 5–27.

Boutilier, C.; Reiter, R.; Soutchanski, M.; and Thrun, S. 2000. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI)*, 355–362. AAAI Press.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about Actions and Planning in LTL Action Theories. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 593–602. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Camacho, A.; Baier, J. A.; Muise, C.; and McIlraith, S. A. 2018. Finite LTL Synthesis as Planning. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS)*.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. A. 2017. Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

Claßen, J. 2018. Symbolic Verification of Golog Programs with First-Order BDDs. In Thielscher, M.; Toni, F.; and Wolter, F., eds., *Proceedings of the Sixteenth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2018)*, 524–529. AAAI Press.

Claßen, J.; and Delgrande, J. P. 2021. 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, 194–204.

Claßen, J.; and Hofmann, T. 2025. vergo 0.1.1. https://doi.org/10.5281/zenodo.14690219.

Claßen, J.; and Lakemeyer, G. 2008. A Logic for Non-Terminating Golog Programs. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 589–599. AAAI Press.

Claßen, J.; Liebenberg, M.; Lakemeyer, G.; and Zarrieß, B. 2014. Exploring the Boundaries of Decidable Verification of Non-Terminating Golog Programs. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, 1012–1019. AAAI Press.

Claßen, J.; and Neuss, M. 2016. Knowledge-Based Programs with Defaults in a Modal Situation Calculus. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, 1309–1317. IOS Press.

Claßen, J.; and Zarrieß, B. 2017. Decidable Verification of Decision-Theoretic Golog. In *Frontiers of Combining Systems*, volume 10483, 227–243. Cham: Springer International Publishing.

De Giacomo, G.; Favorito, M.; Li, J.; Vardi, M.; Xiao, S.; and Zhu, S. 2022. LTLf Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*.

De Giacomo, G.; and Lespérance, Y. 2021. The Nondeterministic Situation Calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, volume 18, 216–226. AAAI Press.

De Giacomo, G.; Lespérance, Y.; Levesque, H. J.; and Sardina, S. 2009. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. In *Multi-Agent Programming*. Springer.

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 1031–1038. IFAAMAS.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2016. Bounded Situation Calculus Action Theories. *Artificial Intelligence*, 237: 172–203.

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 4729–4735. Stockholm, Sweden: AAAI Press.

De Giacomo, G.; Ternovska, E.; and Reiter, R. 1997. Non-Terminating Processes in the Situation Calculus. In *Proceedings of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*.

De Giacomo, G.; and Vardi, M. Y. 2000. Automata-Theoretic Approach to Planning for Temporally Extended Goals. In *Recent Advances in AI Planning*, 226–238. Berlin, Heidelberg: Springer.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, 854–860.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 1558–1564. AAAI Press.

Favorito, M. 2023. Efficient Algorithms for LTLf Synthesis. In *Multi-Agent Systems*, 540–546. Cham: Springer Nature Switzerland.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. 22. Cham: Springer.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Grädel, E.; Otto, M.; and Rosen, E. 1997. Two-Variable Logic with Counting Is Decidable. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS)*, 306–317.

Lakemeyer, G.; and Levesque, H. J. 2010. A semantic characterization of a useful fragment of the situation calculus with knowledge. *Artificial Intelligence*, 175(1): 142–164.

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31(1-3): 59–83.

Li, J.; Pu, G.; Zhang, Y.; Vardi, M. Y.; and Rozier, K. Y. 2020. SAT-based Explicit LTLf Satisfiability Checking. *Artificial Intelligence*, 289: 103369.

Liu, Y. 2002. A Hoare-Style Proof System for Robot Programs. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 74–79. USA: American Association for Artificial Intelligence.

Liu, Y.; and Lakemeyer, G. 2009. On First-Order Definability and Computability of Progression for Local-Effect Actions and Beyond. In Boutilier, C., ed., *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, 860–866. AAAI Press.

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

Patrizi, F.; Lipoveztky, N.; De Giacomo, G.; and Geffner, H. 2011. Computing Infinite Plans for LTL Goals Using a Classical Planner. In *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI)*.

Pnueli, A.; and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 179–190. New York, NY: ACM.

Ramadge, P.; and Wonham, W. 1989. The Control of Discrete Event Systems. *Proceedings of the IEEE*, 77(1): 81–98.

Reiter, R. 2001a. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Reiter, R. 2001b. On Knowledge-Based Programming with Sensing in the Situation Calculus. *ACM Transactions on Computational Logic*, 2(4): 433–457.

Schulz, S.; Cruanes, S.; and Vukmirovic, P. 2019. Faster, Higher, Stronger: E 2.3. In Fontaine, P., ed., *Proceedings of the Twenty-Seventh International Conference on Automated Deduction (CADE 2019)*, volume 11716 of *Lecture Notes in Computer Science*, 495–507. Springer.

Shapiro, S.; Lespérance, Y.; and Levesque, H. J. 2002. The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, 19–26. New York, NY, USA: Association for Computing Machinery.

Xiao, S.; Li, J.; Zhu, S.; Shi, Y.; Pu, G.; and Vardi, M. 2021. On-the-Fly Synthesis for LTL over Finite Traces. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7): 6530–6537.

Zarrieß, B.; and Claßen, J. 2014a. On the Decidability of Verifying LTL Properties of Golog Programs. In *Proceedings of the AAAI 2014 Spring Symposium: Knowledge Representation and Reasoning in Robotics (KRR)*. AAAI Press.

Zarrieß, B.; and Claßen, J. 2014b. Verifying CTL* Properties of Golog Programs over Local-Effect Actions. In *Proceedings of the Twenty-First European Conference on Artificial Intelligence (ECAI 2014)*, 939–944. IOS Press.

Zarrieß, B.; and Claßen, J. 2016. Decidable Verification of Golog Programs over Non-Local Effect Actions. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, 1109–1115. AAAI Press.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic LTLf Synthesis. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 1362–1369. Melbourne, Australia: AAAI Press.

A Proofs

Theorem 1. *Let Φ be a temporal formula, w a world, and z and z' traces. Then $w, z, z' \models \Phi$ iff $w, z, z' \models \operatorname{tnf}(\Phi)$.*

Proof. We show by structural induction on Φ that for arbitrary w, z, z', it holds that $w, z, z' \models \Psi$ iff $w, z, z' \models \operatorname{tnf}(\Psi)$.

- Let Φ be $\top, \bot$, or a C^2-fluent sentence. Then $\operatorname{tnf}(\Phi) = \Phi$ and the claim holds.
- The Boolean cases follow immediately by induction.
- Let Φ be $\mathcal{X}(\Psi)$. Then $w, z, z' \models \mathcal{X}(\Psi)$ iff $z' = \alpha \cdot z'' \neq \langle\rangle$ and $w, z \cdot \alpha, z'' \models \Psi$. By induction, $w, z \cdot \alpha, z' \models \Psi$ iff $w, z \cdot \alpha, z'' \models \operatorname{tnf}(\Psi)$. On the other hand, by definition, $w, z, z' \models \operatorname{tnf}(\mathcal{X}(\Psi))$ iff $w, z, z' \models \neg\, \mathit{Tail} \land \mathcal{X}(\mathrm{t}(\Psi)) \land \mathcal{F}\ \mathit{Tail}$ iff $z' \neq \langle\rangle$ and $w, z, z' \models \mathcal{X}(\operatorname{tnf}(\Psi))$. Hence, the claim holds.
- Let Φ be $\mathcal{N}(\Psi)$ and so $\operatorname{tnf}(\Psi) = (\mathit{Tail} \lor \mathcal{X}(\mathrm{t}(\Psi))) \land \mathcal{F}\ \mathit{Tail}$. If $z' = \langle\rangle$, then $w, z, z' \models \mathit{Tail}$ and so $w, z, z' \models \operatorname{tnf}(\Psi)$. Otherwise, $z' \neq \langle\rangle$ and so $w, z, z' \models \mathcal{N}\,\Psi$ iff $w, z \cdot \alpha, z'' \models \Psi$ for $z' = \alpha \cdot z''$. On the other hand, $w, z \cdot \alpha, z'' \models \operatorname{tnf}(\Psi)$ iff $w, z, z' \models \mathcal{X}(\mathrm{t}(\Psi)) \land \mathcal{F}\ \mathit{Tail}$. By induction, $w, z \cdot \alpha, z'' \models \operatorname{tnf}(\Psi)$ iff $w, z \cdot \alpha, z'' \models \Psi$. With $\operatorname{tnf}(\Psi) = \mathrm{t}(\Psi) \land \mathcal{F}\ \mathit{Tail}$, the claim holds.
- Let Φ be $\Psi_1\, \mathcal{U}\, \Psi_2$ and so $\operatorname{tnf}(\Phi) = (\neg\, \mathit{Tail} \land \mathrm{t}(\Psi_1))\, \mathcal{U}\, \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$.
  ⇒: Suppose $w, z, z' \models \Psi_1 \mathcal{U}\, \Psi_2$. Then there is some $k \leq |z'|$ such that $w, z \cdot z'[..k], z'[k{+}1..] \models \Psi_2$ and for all $0 \leq i < k$, $w, z \cdot z'[..i], z'[i+1..] \models \Psi_1$. By induction, it follows that $w, z \cdot z'[..k], z'[k+1] \models \operatorname{tnf}(\Phi_2)$, which holds iff $w, z[..k], z'[k+1..] \models \mathrm{t}(\Phi_2) \land \mathcal{F}\ \mathit{Tail}$. Furthermore, for every $i < k$, $w, z \cdot z'[..i], z'[i{+}1..] \models \Phi_1 \land \neg\, \mathit{Tail}$ and so by induction $w, z[..i], z'[i+1..] \models \operatorname{tnf}(\Phi_1) \land \neg\, \mathit{Tail}$, which implies $w, z[..i], z'[i+1..] \models \neg\, \mathit{Tail} \land \mathrm{t}(\Phi_1)$. Hence, $w, z, z' \models (\neg\, \mathit{Tail} \land \mathrm{t}(\Psi_1))\, \mathcal{U}\, \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$.
  ⇐: Suppose $w, z, z' \models (\neg\, \mathit{Tail} \land \mathrm{t}(\Psi_1)) \mathcal{U} \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$. Hence, there is a k such that $w, z \cdot z'[..k], z'[k+1..] \models \mathrm{t}(\Psi_2)$ and for all $0 \leq i < k$, $w, z \cdot z'[..i], z'[i+1..] \models \mathrm{t}(\Psi_1) \land \neg\, \mathit{Tail}$. By induction, $w, z \cdot z'[..k], z'[k+1..] \models \Psi_2$ and for all $0 \leq i < k$, $w, z \cdot z'[..i], z'[i+1..] \models \Psi_1$. Therefore, $w, z, z' \models \Psi_1\, \mathcal{U}\, \Psi_2$.
- Let $\Phi = \Psi_1\, \mathcal{R}\, \Psi_2$ and so $\operatorname{tnf}(\Phi) = (\mathit{Tail} \lor \mathrm{t}(\Psi_1))\, \mathcal{R}\, \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$.
  ⇒: Suppose $w, z, z' \models \Phi$. We have two cases: First, $w, z \cdot z'[..i], z'[i+1..] \models \Psi_2$ for all $i \leq |z'|$. By induction, for each i, $w, z \cdot z'[..i], z'[i+1..] \models \operatorname{tnf}(\Psi_2)$ and so $w, z \cdot z'[..i], z'[i+1..] \models \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$ and hence $w, z, z' \models (\mathit{Tail} \lor \mathrm{t}(\Psi_1))\, \mathcal{R}\, \mathrm{t}(\Psi_2)$. Second, there is an i such that $w, z \cdot z'[..i], z'[i+1..] \models \Psi_1$ and $w, z \cdot z'[..j], z'[j+1..] \models \Psi_2$ for all $j \leq i$. Again by induction, for this i, $w, z \cdot z'[..i], z'[i+1..] \models \operatorname{tnf}(\Psi_1)$ and $w, z \cdot z'[..j], z'[j+1..] \models \operatorname{tnf}(\Psi_2)$ for each $j \leq i$. Therefore, $w, z, z' \models (\mathit{Tail} \lor \mathrm{t}(\Psi_1))\, \mathcal{R}\, \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$.
  ⇐: Suppose $w, z, z' \models (\mathit{Tail} \lor \mathrm{t}(\Psi_1))\, \mathcal{R}\, \mathrm{t}(\Psi_2) \land \mathcal{F}\ \mathit{Tail}$. Then there is some $k \leq |z'|$ such that $w, z \cdot z'[..k], z'[k+1..] \models \mathit{Tail} \lor \mathrm{t}(\Psi_1)$ and for all $0 \leq i \leq k$, $w, z \cdot z'[..i], z'[i+1..] \models \mathrm{t}(\Psi_2)$. Hence, by induction, $w, z \cdot z'[..k], z'[k+1..] \models \Psi_1$ and for all $0 \leq i \leq k$, $w, z \cdot z'[..i], z'[i+1..] \models \Psi_2$. Thus, $w, z, z' \models \Psi_1\, \mathcal{R}\, \Psi_2$. □

Lemma 10. *Let w be a world, Φ an LTL$_f$ formula, and z and z' traces. Then $w, z, z' \models \Phi$ implies there exists a propositional assignment P with $P \models \Phi^p$ and $w, z, z' \models \bigwedge P$.*

Proof. [Adapted from (Li et al. 2020), Theorem 2]
By structural induction on Φ.

- If Φ is a literal, $\mathcal{X}$, $\mathcal{U}$, or $\mathcal{R}$ formula, then $P = \{\Phi\}$ is a satisfying propositional assignment and $w, z, z' \models \bigwedge P$.
- For $\Phi = \Psi_1 \land \Psi_2$, by induction, there P_1 and P_2 with $P_1 \models \Psi_1^p$ and $P_2 \models \Psi_2^p$. Let $P = P_1 \cup P_2$ be a consistent propositional assignment, in which no literal occurs both positively and negatively. Such a propositional assignment must exist because otherwise, $w, z, z' \not\models \Phi$. Then $P \models \Phi^p$ and $w, z, z' \models \bigwedge P$.
- For $\Phi = \Psi_1 \lor \Psi_2$, we have $w, z, z' \models \Psi_1$ or $w, z, z' \models \Psi_2$. Wlog, $w, z, z' \models \Psi_1$ and so by induction, there exists a propositional assignment P_1 with $P_1 \models \Psi_1^p$ and $w, z, z' \models \bigwedge P_1$. □

Theorem 3. *Let Φ be a temporal formula, w a world, and z and z' finite traces. Then $w, z, z' \models \Phi$ iff $w, z, z' \models \operatorname{xnf}(\Phi)$.*

Proof. By structural induction on Φ.

- If Φ is $\top, \bot$, a C^2-fluent sentence, or $\mathcal{X}\,\Psi$, then $\operatorname{xnf}(\Phi) = \Phi$ and the claim holds.
- The Boolean cases follow immediately by induction.
- Let $\Phi = \Psi_1\, \mathcal{U}\, \Psi_2$. By semantics of $\mathcal{U}$, $w, z, z' \models \Phi$ iff $w, z, z' \models \Psi_2$ or $w, z, z' \models \Psi_1 \land \mathcal{X}(\Psi_1\, \mathcal{U}\, \Psi_2)$. By induction, $w, z, z' \models \Psi_2$ iff $w, z, z' \models \operatorname{xnf}(\Psi_2)$ and $w, z, z' \models \Psi_1 \land \mathcal{X}(\Psi_1\, \mathcal{U}\, \Psi_2)$ iff $w, z, z' \models \operatorname{xnf}(\Psi_1) \land \mathcal{X}(\operatorname{xnf}(\Psi_1\, \mathcal{U}\, \Psi_2))$ and so the claim follows.
- Let $\Phi = \Psi_1\, \mathcal{R}\, \Psi_2$. By semantics of $\mathcal{R}$, $w, z, z' \models \Phi$ iff $w, z, z' \models \Psi_2$ and $w, z, z' \models \Psi_1 \lor \mathcal{X}(\Psi_1\, \mathcal{R}\, \Psi_2)$. By induction, $w, z, z' \models \Psi_2$ iff $w, z, z' \models \operatorname{xnf}(\Psi_2)$ and $w, z, z' \models \Psi_1 \lor \mathcal{X}(\Psi_1\, \mathcal{R}\, \Psi_2)$ iff $w, z, z' \models \operatorname{xnf}(\Psi_1) \lor \mathcal{X}(\operatorname{xnf}(\Psi_1\, \mathcal{R}\, \Psi_2))$ and so the claim follows. □

Lemma 11. *For any program δ, $\mathcal{C}_\delta$ is finite, and for any world w, situation z, and $\delta' \in \operatorname{sub}(\delta)$, it holds that (1) $\langle z, \delta' \rangle \in \operatorname{Fin}(w)$ iff $w, z \models \varphi(\delta')$; and (2) $\langle z, \delta' \rangle \xrightarrow{w} \langle z \cdot \alpha, \delta'' \rangle$ iff $\delta' \xrightarrow{\alpha:\psi} \delta''$ and $w, z \models \psi$.*

Lemma 12. *If every ground action α occurs at most once among the outgoing edges of every node in $\mathcal{C}_\delta$, then δ is situation-determined.*

Proof. [Proof Idea] By induction on the length of traces starting in $\langle z, \delta \rangle$, using Lemma 4. □

We can show that $\mathbb{A}_{\mathcal{G}}^{\Phi}$ indeed tracks the program executions of $\mathcal{G}$:

Lemma 13. *Let $z = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathcal{Z}$ be an arbitrary trace and $\mathcal{G} = (\mathcal{D}, \delta)$ a* GOLOG *program. Then $z \in \|\delta\|_w$ for some world w with $w \models \mathcal{D}$ iff there is a path $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ such that s_0 is an initial state with $\operatorname{type}(s_0) = \operatorname{type}(w)$ and s_n is a final state.*

Proof. We first show by induction on n that $\langle\langle\rangle, \delta\rangle \xrightarrow{w} \langle z[..1], \delta_1\rangle \xrightarrow{w} \cdots \xrightarrow{w} \langle z, \rho_n\rangle$ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ iff $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ such that $\mathrm{type}(s_0) = \mathrm{type}(w) = \tau$ and where for every context formula $\phi \in \mathcal{C}(\mathcal{G})$, we have $w, z[..i] \models \phi$ iff $(\phi, E_i) \in \mathrm{type}(s_i)$.
Let each s_i be of the form $s_i = (\tau, E_i, A_i, \rho_i)$.
Base case. $n = 0$: By definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, if $w \models \mathcal{D}$, then there is an initial state s_0 with $\mathrm{type}(s_0) = \mathrm{type}(w)$. Also, $E_0 = \emptyset$ and so $(\phi, E_0) \in \mathrm{type}(s_0)$ iff $w \models \phi$.
Induction step. By definition, there is a transition $s_i \xrightarrow{\alpha_i} s_{i+1}$ iff $\rho_i \xrightarrow{\alpha_i:\psi} \rho_{i+1}$ and $(\psi, E_i) \in \tau$. By induction, $(\psi, E_i) \in \tau$ iff $w, z[..i] \models \psi$ and with Lemma 4, it follows that $s_i \xrightarrow{\alpha_i} s_{i+1}$ iff $\langle z[..i], \rho_i\rangle \xrightarrow{w} \langle z[..i+1], \rho_{i+1}\rangle$. By definition $E_2 = E_1 \triangleright \mathcal{E}_{\mathcal{D}}(\tau, E_1, \alpha_i)$ and so, with Theorem 6, for every $\phi \in \mathcal{C}(\mathcal{G})$, we have $(\phi, E_{i+1}) \in \tau$ iff $w, z[..i+1] \models \phi$.

Now, by Lemma 4, $z \in \|\delta\|_w$ iff $w, z \models \varphi(\delta)$. From above, it follows that $w, z \models \varphi(\delta)$ iff $(\varphi(\delta), E_n) \in \tau$ iff s_n is final. □

Regarding the temporal formula Φ, the following two lemmas show that $\mathbb{A}_{\mathcal{G}}^{\Phi}$ indeed tracks the satisfaction of Φ:

Lemma 14. *Suppose $w, \langle\rangle, z \models \Phi$ with $z = \langle\alpha_1, \ldots, \alpha_n\rangle$. Then there is a path $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ starting in an initial state s_0 with $\mathrm{type}(s_0) = \mathrm{type}(w)$ such that $s_i = (\tau, E_i, A_i, \rho_i)$ and such that for every $i \leq n$, $w, z[..i], z[i+1..] \models \bigwedge_{\Psi \in \chi_i} \mathcal{X}\,\Psi$ for some $(\chi_i, \theta_i) \in A_i$.*

Proof. By induction on i.
Base case. Let $i = 0$. By Lemma 2, there is a propositional assignment P_0 of $\mathrm{xnf}(\Phi)^p$ with $w, \langle\rangle, z \models \bigwedge P_0$ and therefore also $w, \langle\rangle, z \models \bigwedge_{\Psi \in X(P_0)} \mathcal{X}\,\Psi$. By definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, $(X(P_0), \theta_0) \in A_0$ for some θ_0.
Induction step. By induction, $w, z[..i-1], z[i..] \models \bigwedge_{\Psi \in \chi_{i-1}} \mathcal{X}\,\Psi$ for some $(\chi_{i-1}, \theta_{i-1}) \in A_{i-1}$. Hence, $w, z[..i], z[i+1..] \models \bigwedge \chi_{i-1}$. By Lemma 2, there is a propositional assignment P_i of $\mathrm{xnf}(\bigwedge \chi_{i-1})^p$ with $w, z[..i], z[i+1..] \models \bigwedge P_i$ and hence also $w, z[..i], z[i+1..] \models \bigwedge_{\Psi \in X(P_i)} \mathcal{X}\,\Psi$. By definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, $(X(P_i), T(P_i)) \in A_i$. □

Lemma 15. *Let $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ be a path in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ starting in an initial state s_0 with $\mathrm{type}(s_0) = \mathrm{type}(w)$ and ending in an accepting state s_n. Suppose $s_i = (\tau, E_i, A_i, \rho_i)$ for each i. Then there is a sequence $\chi_0, \ldots, \chi_n$ such that for each i, $(\chi_i, \theta_i) \in A_i$ and $w, z[..i], z[i+1..] \models \bigwedge_{\Psi \in \chi_i} \mathcal{X}\,\Psi$.*

Proof. By induction on i from n to 0.
Base case. Let $i = n$. Then s_n is accepting and so there is $(\chi_n, \theta_n) \in A_n$ with $\theta_n = \top$ and $\chi_n = \emptyset$. Trivially, $w, z, \langle\rangle \models \bigwedge_{\Psi \in \chi_n} \mathcal{X}\,\Psi$.
Induction step. By induction, there is $(\chi_i, \theta_i) \in A_i$ such that $w, z[..i], z[i+1..] \models \bigwedge_{\Psi \in \chi_i} \mathcal{X}\,\Psi$. By definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, there is a propositional assignment P_i of $\mathrm{xnf}(\bigwedge \chi_{i-1})^p$ (as otherwise $A_i = \emptyset$) such that $X(P_i) = \chi_i$, $T(P_i) = \theta_i$, and $\{(\psi, E_i) \mid \psi \in L(P_i)\} \subseteq \tau$. Therefore, $w, z[..i], z[i+1..] \models L(P) \wedge T(P) \wedge \bigwedge_{\Psi \in \chi_i} \mathcal{X}\,\Psi$ and so $w, z[..i], z[i+1..] \models \mathrm{xnf}(\bigwedge \chi_{i-1})$. It directly follows that $w, z[..i-1], z[i...] \models \bigwedge_{\Psi \in \chi_{i-1}} \mathcal{X}\,\Psi$. Again by definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, $(\chi_{i-1}, \theta_{i-1}) \in A_{i-1}$. □

Combining the results, we obtain the following theorem:

Theorem 7. *Every execution of $\mathcal{G} = (\mathcal{D}, \delta)$ satisfies Φ iff every reachable final state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$ is accepting.*

Proof.
⇒: By contradiction. Suppose there is a reachable final state $s_n = (\tau, E, A, \rho)$ that is not accepting and let $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ be a path in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ starting in an initial state s_0 with $\mathrm{type}(s_0) = \mathrm{type}(w)$ and ending in s_n. By Lemma 13, $z = \langle\alpha_1, \ldots, \alpha_n\rangle \in \|\delta\|_w$. By assumption, $w, \langle\rangle, z \models \Phi$ and so, with Lemma 14, $w, z, \langle\rangle \models \bigwedge_{\Psi \in \chi} \mathcal{X}\,\Psi$ for some $(\chi, \theta) \in A$. Clearly, $w, z, \langle\rangle \not\models \mathcal{X}\,\Psi$ for arbitrary Ψ, and so $\chi = \emptyset$. Furthermore, $w, z, \langle\rangle \models \mathit{Tail}$ and so $\theta = \top$. But then, s_n is accepting, a contradiction.
⇐: By contradiction. Suppose there is a trace $z = \langle\alpha_1, \ldots, \alpha_n\rangle$ such that $z \in \|\delta\|_w$ but $w, \langle\rangle, z \not\models \Phi$. By Lemma 13, there is a path $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ starting in an initial state s_0 with $\mathrm{type}(s_0) = \mathrm{type}(w)$ and ending in a final state s_n. By assumption, s_n is accepting. By Lemma 15, $w, \langle\rangle, z \models \bigwedge_{\Psi \in \chi} \mathcal{X}\,\Psi$ for some $(\chi, \theta) \in A$ and $s_0 = (\tau, E, A, \rho)$. By definition, for each $\chi \in A$, there is a propositional assignment P of $\mathrm{xnf}(\bigwedge \chi^p)$ such that $w, \langle\rangle, z \models \bigwedge L(P)$ and $\chi = X(P)$. But then, $w, \langle\rangle, z \models L(P) \wedge \bigwedge_{\Psi \in X(P)} \mathcal{X}\,\Psi$ and so $w, \langle\rangle, z \models \Phi$, a contradiction. □

Proposition 16. *There is a terminating and winning strategy σ in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ if and only if there exists a terminating policy π for $\mathcal{G}$ that satisfies Φ.*

Proof.
⇒: Let σ be a terminating and winning strategy in $\mathbb{A}_{\mathcal{G}}^{\Phi}$. For a play $p = \langle s_0, \ldots, s_n\rangle \in \mathrm{plays}(\sigma)$, let $\mathrm{Acts}(p)$ denote the (unique) trace $\langle\alpha_1, \ldots, \alpha_n\rangle$ such that $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n$. We construct π as follows: For every play with $p = \langle s_0, \ldots, s_n\rangle \in \mathrm{plays}(\sigma)$ where $s_i = (\tau, E_i, A_i, \rho_i)$ (note that by definition, τ is the same for each s_i) and $\mathrm{Acts}(p) = z = \langle\alpha_1, \ldots, \alpha_n\rangle$ and for every world w with $\mathrm{type}(w) = \tau$, we define $\pi(w, z[..i], \rho_i) = \sigma(s_i)$.
We first show that π is a proper policy for $\mathcal{G}$ by showing that it satisfies the conditions of Definition 6: First, note that $\mathbb{A}_{\mathcal{G}}^{\Phi}$ contains an initial state with $s = (\tau, \emptyset, A, \delta)$ for every w with $w \models \mathcal{D}$ and so 1 is satisfied. Also, for every state s, $\sigma(s)$ is valid and hence 2 as well as 4 is satisfied. Furthermore, by definition of the strategy, if $\alpha \in \sigma(s)$ and $s \xrightarrow{\alpha} s'$, then σ is defined on s' and so π is defined on the corresponding $(w, z \cdot \alpha, \rho')$ and hence 3 is satisfied. Finally, again because each $\sigma(s)$ is valid, 5 is satisfied.

Furthermore, π is terminating and satisfies Φ: From σ being a terminating strategy, it directly follows that π is terminating. Now, let $z \in \|\pi\|_w$ for some world w. By definition of π, there is a play $p = \langle s_0, \ldots, s_n\rangle \in \mathrm{plays}(\sigma)$

with $\mathrm{Acts}(p) = z$ for some $s_0 = (\tau, \emptyset, A_0, \delta)$ and with $\mathrm{type}(w) = \tau$. By Lemma 14, there is some $(\chi, \theta) \in A_0$ such that $w, \langle\rangle, z \models \bigwedge_{\Psi \in \chi} \mathcal{X}\,\Psi$. By definition of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, there is a propositional assignment P such that $X(P) = \chi$, $T(P) = \theta$, and $\{(\psi, E) \mid \psi \in L(P)\} \subseteq \tau$. By Theorem 6, $w, z \models \bigwedge L(P)$ and so $w, \langle\rangle, z \models \Phi$.
$\Leftarrow$: Let π be a terminating policy for $\mathcal{G}$ that satisfies Φ. Note that we cannot directly construct a strategy σ from π as the policy is defined on traces and hence we may have $\pi(w, z_1, \rho) \neq \pi(w, z_2, \rho)$ even if z_1 and z_2 correspond to the same state in $\mathbb{A}_{\mathcal{G}}^{\Phi}$. Hence, we define σ on $\mathbb{A}_{\mathcal{G}}^{\Phi}$ as follows: First, for any w and $z \in \|\pi\|_w$ and every $i \leq |z|$, let $\rho_{z[..i]}$ be the remaining program after $z[..i]$, i.e., $\langle\langle\rangle, \delta\rangle \xrightarrow{w}^{*} \langle z[..i], \rho_{z[..i]}\rangle$. The program expression $\rho_{z[..i]}$ is well-defined because $\mathcal{G}$ is situation-determined. Now, suppose $s = (\tau, E, A, \rho)$ is a state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$, then let Z_s^{π} be the set of traces from an initial state to s that are compatible with π, i.e., $z = \langle \alpha_1, \ldots, \alpha_n \rangle \in Z_s$ if $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s$ is a path in $\mathbb{A}_{\mathcal{G}}^{\Phi}$ where s_0 is an initial state and $\alpha_{i+1} \in \pi(w, z[..i], \rho_{z[..i]})$ for some w with $\mathrm{type}(w) = \tau$. If there is $z \in \|\pi\|_w$ such that $z[..i] \in Z_s^{\pi}$ and for all $j > i$, $z[..j] \notin Z_s^{\pi}$ (i.e., π does not return to s after $z[..i]$), then we define $\sigma(s) = \pi(w, z[..i], \rho_{z[..i]})$. Otherwise, there must be a cycle in π that passes through a final and accepting configuration (as otherwise π would either be non-terminating or not satisfying Φ). Hence, let $z \in \|\pi\|_w$ be the corresponding trace such that for some i, $z[..i] \in Z_s^{\pi}$, $\langle z[..j], \rho_{z[..j]}\rangle \in \mathrm{Fin}(w)$ for some $j > i$, and $w, \langle\rangle, z \models \Phi$ and $z[..k] \notin Z_s^{\pi}$ for all $i < k < j$. We set $\sigma(s) = \pi(w, z[..i], \rho_{z[..i]})$ and so σ visits a final and accepting state before visiting s again.

We first show that σ is a proper strategy for $\mathbb{A}_{\mathcal{G}}^{\Phi}$: Clearly, as π is a proper policy and thus by Definition 6-1 defined on every initial configuration, σ is defined on every initial state of $\mathbb{A}_{\mathcal{G}}^{\Phi}$. Second, every $\sigma(s)$ is valid, because π satisfies 2, 4, and 5 of Definition 6. Finally, σ is defined on every σ-reachable state s, as σ follows π and by Definition 6-3, π is defined on every successor configuration.

It remains to be shown that σ is winning and terminating. As $\mathbb{A}_{\mathcal{G}}^{\Phi}$ is finite, every infinite path must visit a state twice. By construction, σ visits a final state before visiting a state s again. Furthermore, as π is terminating, there must be such a state with $\sigma(s) \subseteq \mathcal{A}_E$ and so σ is terminating. Finally, by construction, every play $p \in \mathrm{plays}(\sigma)$ corresponds to a trace $z \in \|\pi\|_w$ for some w with $w, \langle\rangle, z \models \Phi$. Let $s = (\tau, E, A, \rho)$ be the last state of p. By Lemma 14, there is $(\chi, \theta) \in A$ such that $w, z, \langle\rangle \models \bigwedge_{\Psi \in \chi} \mathcal{X}\,\Psi$. However, by the semantics of temporal formulas, this is only possible if $\chi = \emptyset$ and $\theta = \top$. Hence, s is accepting and so every $p \in \mathrm{plays}(\sigma)$ is winning. As every play is winning, σ is winning. □

Theorem 9. *Algorithm 1 terminates and returns a winning and terminating strategy if one exists.*

Proof. It is easy to see that Algorithm 1 terminates: Note that a state s is only added to Q if one of its successors is added to R or if it is in Q initially. As there are only finitely many states in $\mathcal{S}$, only finitely many states can be added to Q, and hence Q is eventually empty. Finally, again because $\mathcal{S}$ is finite, there can only be finitely many hypotheses H.
We continue by showing each returned strategy is winning and terminating: Assume Algorithm 1 returns a strategy σ that is not winning. Then there is a play $p = \langle s_0, s_1, \ldots, s_n \rangle \in \mathrm{plays}(\sigma)$ that is not winning, i.e., ending in a state s_n that is final but not accepting. Clearly, s_n is only added to R if every environment successor is in G, or if there is a control successor in G. As the play ends in s_n, $\sigma(s_n) \subseteq \mathcal{A}_E$ and so every environment successor of s_n is in G. However, as s_n is final but non-accepting, by line 6, s_n is not added to G and hence also not to R, contradicting the assumption.
Now, assume σ is non-terminating. Then there is an infinite sequence of σ-compatible states $s_0, s_1, \ldots$ such that for some i, every state s_j for $j \geq i$ is non-final or $\sigma(s_j) \cap \mathcal{A}_C \neq \emptyset$. As initially G only consists of final and accepting states, it is easy to see that for every j, $\alpha \in \sigma(s_j)$ and $s_j \xrightarrow{\alpha} s_{j+1}$ implies that s_{j+1} is closer to some final and accepting state than s_j. As there are only finitely many states in $\mathcal{S}$, for every j, there must be a $k \geq j$ such that s_k is final and accepting. Finally, by line 11, $\sigma(s_k) \subseteq \mathcal{A}_E$, contradicting the assumption.
Finally, we show that the algorithm is complete. Assume σ is a winning and terminating strategy but Algorithm 1 does not return a winning and terminating strategy. First, from above, it directly follows that it returns $\bot$ (as any strategy returned is in fact winning and terminating). Now, let H be the final and accepting states that are visited by σ. We define a distance $d(s)$ as the maximal number of steps to reach a final and accepting state from s in any play of σ, i.e., $d(s) = \max\{j \mid p_0, \ldots, p_i, s, s_1, \ldots s_j \in \mathrm{plays}(\sigma), s_j \in H, \forall i < j : s_i \notin H\}$. Clearly, $d(s)$ is defined and finite for all initial states s and all states in H, as otherwise σ would not be winning. We can now show by induction on $d(s)$ that every state s visited by σ is added to G. The base case is trivial. For the induction step, let s be a state with $d(s) = n$ and assume that every state s' with $d(s') < n$ is in G. As σ is winning, for every $s \in \mathrm{Succ}_E(s)$, there is an action $\alpha \in \sigma(s)$ such that $s \xrightarrow{\alpha} s'$. By definition, $d(s') < n$ and so $s' \in G$. If $\mathrm{Succ}_E(s) = \emptyset$, there must be an action $\alpha \in \mathcal{A}_C$ with $\alpha \in \sigma(s)$. Again, for every $s' \in \mathrm{Succ}_C(s)$, $d(s') < n$ and so $s' \in G$. By line 8, s is added to G. Hence, after the while loop terminates, $H \cup \mathcal{S} \subseteq R$ and so the algorithm returns some strategy, a contradiction. □