Verifying CTL* Properties of GOLOG Programs over Local-Effect Actions

Benjamin Zarrieß¹ and Jens Claßen²

Abstract. GOLOG is a high-level action programming language for controlling autonomous agents such as mobile robots. It is defined on top of a logic-based action theory expressed in the Situation Calculus. Before a program is deployed onto an actual robot and executed in the physical world, it is desirable, if not crucial, to verify that it meets certain requirements (typically expressed through temporal formulas) and thus indeed exhibits the desired behaviour. However, due to the high (first-order) expressiveness of the language, the corresponding verification problem is in general undecidable. In this paper, we extend earlier results to identify a large, non-trivial fragment of the formalism where verification is decidable. In particular, we consider properties expressed in a first-order variant of the branching-time temporal logic CTL*. Decidability is obtained by (1) resorting to the decidable first-order fragment $C^2$ as underlying base logic, (2) using a fragment of GOLOG with ground actions only, and (3) requiring the action theory to only admit local effects.

1 Introduction

GOLOG [11, 16], a family of high-level action programming languages, has proven to be a useful means for the control of autonomous agents such as mobile robots [5, 10]. It is defined on top of first-order logic (with some second-order features) for representing and reasoning in dynamic application domains. Before a GOLOG program is deployed onto a robot and executed in the physical world, it is often desirable, if not crucial, to verify that certain criteria are met, typical examples being safety, liveness and fairness conditions.

Verification of GOLOG programs was first considered by De Giacomo, Ternovska and Reiter [12] who presented a corresponding semantics of non-terminating processes defined by means of (second-order) Situation Calculus axioms, expressed temporal properties through (second-order) fixpoint formulas, and provided meta-theoretic proofs to show the satisfaction of such properties. Since it is more preferable to do verification in an automated fashion, Claßen and Lakemeyer [6] later proposed an approach based on a new logic called $\mathcal{ES}$, an extension of the modal Situation Calculus variant $\mathcal{ES}$ [15] by modalities for expressing temporal properties of GOLOG programs. Using regression-based reasoning and a newly introduced graph representation for programs, they provided algorithms for the verification of a fragment of the formalism resembling a first-order, but non-nested variant of the branching-time temporal logic CTL. Their procedures, which perform a fixpoint computation to do a systematic exploration of the state space, could be proven to be sound, but no general termination guarantee could be given due to the verification problem being highly undecidable.

It would of course be desirable if termination were guaranteed, which could be achieved by imposing appropriate restrictions on the input formalism such that the verification problem becomes decidable, while preferably retaining as much first-order expressiveness as possible. One corresponding approach is presented by Baader, Liu and ul Mehdi [2] who, instead of using the fully-fledged Situation Calculus and GOLOG, resort to an action language [3] based on the decidable Description Logic (DL) $\mathcal{ALC}$ [1] and approximate programs through finite Büchi automata. They could show that verification of LTL properties over $\mathcal{ALC}$ axioms thus reduces to a decidable reasoning task within the underlying DL. Baader and Zarrieß [4] later lifted these results to support a more expressive fragment of GOLOG program expressions that goes beyond what can be represented by Büchi automata, in particular regarding test conditions $\phi$ that are needed for expressing imperative programming constructs such as while loops and conditionals, but restricts all actions to be ground.

Another approach is taken in [7] where it is shown that Claßen and Lakemeyer’s original verification algorithms can be guaranteed to terminate by restricting oneself to a decidable, two-variable fragment of FOL as base logic for the Situation Calculus [13], allowing only ground action terms in GOLOG programs, and requiring actions to only have local effects [17]. Compared to the above mentioned results by Baader and Zarrieß, local-effect action theories represent an increase in expressiveness as $\mathcal{ALC}$-based action definitions only allow for basic STRIPS-style addition and deletion of literals, but the class of non-nested CTL-like properties supported by Claßen and Lakemeyer’s method is less expressive than LTL.

In this paper, we turn towards unifying these earlier approaches within a single formal framework, while even increasing expressiveness. In particular, we (1) use $C^2$ as base logic, the two variable fragment of first-order logic with counting quantifiers, which subsumes both $\mathcal{ALC}$ and the two-variable fragment of FOL; we (2) formulate action effects through $\mathcal{ES}$-style local-effect action theories; and (3) we show that verification is decidable for GOLOG programs with only ground actions even in the case of properties expressed in the branching-time temporal logics CTL*, which is strictly more expressive than both CTL and LTL. We obtain decidability by constructing a finite abstraction of the infinite transition system induced by a program, and showing that the abstraction preserves satisfiability of CTL* properties over $C^2$ axioms. We also obtain a 2-NEXPTIME upper bound for the computational complexity of the problem.

The remainder of this paper is organized as follows. Section 2 presents our action formalism, namely the modal Situation Calculus variant $\mathcal{ES}$ using $C^2$ (the two-variable fragment of first-order logic

1 Theoretical Computer Science, TU Dresden, Germany, email: zarrriess@tcs.inf.tu-dresden.de
2 Knowledge-Based Systems Group, RWTH Aachen University, Germany, email: classen@kbsg.rwth-aachen.de
with counting quantifiers) as decidable base logic, as well as Golog programs and their semantics. In Section 3 we then define the problem of verifying $\text{CTL}^*$ properties over such programs, and show (in a constructive manner) that the problem is indeed decidable. We then discuss related work and conclude. Because of space constraints, detailed proofs of our results have to be omitted. They can be found in [22].

2 Preliminaries

2.1 The Modal Situation Calculus $\mathcal{ES}$ based on $\mathbb{C}^2$

In this subsection we recall the main definitions of the modal Situation Calculus variant $\mathcal{ES}$ [15]. Instead of using full first-order logic, we restrict ourselves to the two variable fragment with equality and counting of FOL named $\mathbb{C}^2$.

Syntax: We start by fixing a set of terms. In our language we consider terms of two sorts, namely object and action. Terms can be built using the following symbols: an infinite supply of variables $x, y, \ldots$ of sort object, a single variable $a$ of sort action, an infinite set $N_f$ of rigid object constant symbols (i.e. 0-ary function symbols), and a finite set $N_A$ of rigid action function symbols. A term is called ground term if it contains no variables. Note that in contrast to [13, 7] we also allow action functions with arity $\geq 2$. We use the following symbols for vectors of object terms and object constants: $\vec{x}$ denotes a vector of at most two variables, $\vec{y}$ a vector of arbitrarily many variables and $\vec{z}$ a vector of arbitrarily many constants.

To build formulas, we consider a set of fluent predicate symbols $N_F$ and a set of rigid predicate symbols $N_R$, each of which takes at most two arguments of sort object. Intuitively, fluents are properties that may change due to actions, while rigidics do not. In addition, we consider terms of two sorts, namely object and action, respectively.

Terms and formulas are interpreted w.r.t. a semantic model called a world. Let $\overline{N}_O$ and $\overline{N}_A$ be the set of ground terms (also called "standard names") of sort object, and action, respectively, and $\mathcal{P}_F$ the set of all ground atoms of the form $F(t_1, \ldots, t_k)$ for predicates $F \in N_F \cup N_R$. Moreover, let $\mathcal{Z} = \overline{N}_O^*$ be the set of all finite sequences of ground action terms (including the empty sequence $\mathcal{Z}^0$). A world $w$ then maps ground atoms and action sequences to truth values

$$w : \mathcal{P}_F \times \mathcal{Z} \to \{0, 1\}$$

respecting the rigidity constraint: if $R$ is a rigid predicate, then for all $z, z' \in \mathcal{Z}, w[R(z), z] = w[R(z'), z'].$ Let $\mathcal{W}$ denote the set of all worlds. Given a world $w \in \mathcal{W}$ and a sentence $\alpha$, we define $w \models \alpha$ as $w, \mathcal{Z} \models \alpha$, where for any $z \in \mathcal{Z}$:

1. $w, z \models F(t_1, \ldots, t_k)$ iff $w[F(t_1, \ldots, t_k), z] = 1$;
2. $w, z \models (t_1 = t_2)$ iff $t_1$ and $t_2$ are identical;
3. $w, z \models \alpha \land \beta$ iff $w, z \models \alpha$ and $w, z \models \beta$;
4. $w, z \models \neg \alpha$ iff $w, z \not\models \alpha$;
5. $w, z \models \forall x. \alpha$ iff $w, z \models \alpha^t$ for all $t \in \overline{N}_A^*$.

Basic Action Theories: To encode a dynamic application domain, we use a basic action theory (BAT), which is a set of formulas $D = D_0 \cup D_{pre} \cup D_{post}$, where

1. $D_0$, the initial theory, is a finite set of fluent sentences describing the initial state of the world.
2. $D_{pre}$ is a set of precondition axioms such that for any action function $A \in N_A$ there is an axiom of the form $\Box Poss(A(\vec{y})) \equiv \varphi$, with $\varphi$ being a fluent formula with free variables $\vec{y}$.
3. $D_{post}$ is a finite set of successor state axioms (SSAs), one for each Fluent $F \in N_F$, incorporating Reiter’s [21] solution to the frame problem, and encoding the effects that actions have on the different fluents. The SSA for a fluent $F$ has the form $\Box[F(\vec{d}) \equiv F(\vec{x}) \land \neg F(\vec{y})$, where $F(\vec{d})$ and $F(\vec{y})$ are fluent formulas with free variables $\vec{x}$ and $\vec{a}$.

In this paper we restrict ourselves to local-effect BATs, i.e. in all SSAs the positive and negative effect conditions $F(\vec{d})$ and $F(\vec{y})$, respectively, are disjunctions of formulas $\exists x[a = A(\vec{y}) \land \phi]$, where $A$ is an action function, $\vec{y}$ contains $\vec{x}$, $\vec{z}$ are the remaining variables of $\vec{y}$, and $\phi$ is a fluent formula with free variables $\vec{y}$ and at most two non-free variables that are all of sort object. Intuitively, in a local-effect action theory executing an action $A(\vec{z})$ with arguments $\vec{z}$ means that a fluent $F(\vec{d})$ can change its truth value for $\vec{d}$ only if $\vec{d}$ is contained in $\vec{z}$.

2.2 Golog Programs

Given a BAT axiomatizing preconditions and effects of atomic actions, we now define syntax and semantics of complex actions. The program expressions we consider here are the ones admitted by the following grammar:

$$\delta ::= () | t | \alpha? | \delta_1 ; \delta_2 | \delta_1 [\delta_2 | \delta_1] [\delta_2 | \delta_1] \delta^*$$

That is we consider the fragment of COGLOG [11] that includes the empty program $()$, primitive actions $t$ (where $t$ is a ground action term), tests $\alpha$? (where $\alpha$ is a fluent sentence), sequence, nondeterministic branching, concurrency, and nondeterministic iteration. Note that thus also if statements and while loops are included:

$$\text{if } \alpha \text{ then } \delta_1 \text{ else } \delta_2 \text{ endIf } \overset{\text{def}}{=} [\alpha?; \delta_1] [\neg\alpha?; \delta_2]$$

while $\alpha$ do $\delta$ endWhile $\overset{\text{def}}{=} [\alpha; \delta]^*; \neg\alpha?$

A Golog program $\mathcal{P} = (\mathcal{D}, \delta)$ is then given by a BAT $\mathcal{D}$ and a program expression $\delta$.

Program expressions are interpreted as follows. A configuration $\langle z, \delta \rangle$ consists of an action sequence $z \in \mathcal{Z}$ and a program expression $\delta$, where intuitively $z$ is the history of actions that have already been performed, while $\delta$ is the program that remains to be executed.

Definition 1 (Program Transition Semantics). Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program. The transition relation $\rightarrow_{\mathcal{P}}$ among configurations, given a world $w$ with $w, () \models D_0$, is defined by induction on the size of $\delta$ as the least set satisfying
Let $\text{sub}_T$ reachable subprograms (Transition System) runs by having “sinks” that continue looping.

We are now ready to define the verification problem:

**Definition 4 (Verification Problem).** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program, $\mathcal{T}_P = (Q, \rightarrow, I)$ the corresponding transition system and $\Phi$ an $\mathcal{ES}C^2$-CTL* state formula. The formula $\Phi$ is valid in $\mathcal{P}$ if $\mathcal{T}_P \models \Phi$. The formula $\Phi$ is satisfiable in $\mathcal{P}$ if there exists a $s_0 \in I$ such that $\mathcal{T}_P, s_0 \models \Phi$.

Note that the logic $\mathcal{ES}C^2$-CTL* is expressive enough to encode several variants of the verification problem. For example, one can express global domain constraints as a conjunction of fluent sentences. The problem of whether these constraints persist during the execution of a program $\mathcal{P}$ corresponds to validity of the formula $\mathcal{AG}_{\mathcal{P}}$ in the program $\mathcal{P}$. Furthermore, the fluents $\text{Term}$ and $\text{Fail}$ can be used to express facts about the termination or failing of a program.

### 3.2 Deciding the Verification Problem

In general, the transition system induced by a Golog program is infinite. In order to obtain a decision procedure, we show that it is possible to construct a finite abstraction that preserves satisfiability of $\mathcal{ES}C^2$-CTL* formulas. We do this by defining an equivalence relation over the set of semantic models (worlds) and identifying the corresponding equivalence classes (called types). Since it can be shown that there are only finitely many such classes for a given program and since their computation reduces to a consistency check in the underlying decidable base logic $C^2$, the abstract transition system can be effectively computed, and standard model checking techniques can be used to verify the given $\mathcal{ES}C^2$-CTL* property.
3.2.1 Regression with Ground Actions

The first ingredient for our result is to use a technique presented in [18, 17] for simplifying local-effect SSAs in presence of ground actions only. The idea is that it is sufficient to consider the ground instantiations of SSAs obtained by substituting the action variable $\alpha$ by each ground action term and simplifying using unique names for constants and actions:

**Lemma 2.** Let $D$ be a BAT. $\Box[\alpha]F(\vec{x}) \equiv \gamma_+^D \lor F(\vec{x}) \land \neg \gamma_-^D \in D_{\text{pos}}$ the local-effect SSA for the fluent $F(\vec{x})$ and $t = \text{Poss}(\vec{c})$ a ground action term. For the ground instantiated SSA for $F(\vec{x})$ w.r.t. $t$, given as $\Box[t]F(\vec{x}) \equiv \gamma_{+\vec{c}}^D \lor F(\vec{x}) \land \neg \gamma_{-\vec{c}}^D$, it holds that both $\gamma_{+\vec{c}}^D$ and $\gamma_{-\vec{c}}^D$ are equivalent to disjunctions of the form

$$\vec{x} = c_1 \land \phi_1 \lor \cdots \lor \vec{x} = c_n \land \phi_n,$$

where the vectors of constants $c_i$ are contained in $\vec{c}$ and the formulas $\phi_i$ are fluent sentences with $i = 1, \ldots, n$.

From now on we hence assume that in the ground instantiated SSAs, the $\gamma_{+\vec{c}}^D$ and $\gamma_{-\vec{c}}^D$ are of the form (6). We use the notation $(\vec{c}, \phi) \in \gamma_{+\vec{c}}^D$ and $(\vec{c}, \phi) \in \gamma_{-\vec{c}}^D$ to express that there is a disjunct of the form $\vec{x} = \vec{c} \land \phi \lor \vec{c}$ or $\vec{c} \land \phi$, respectively.

Next, we define an effect function mapping a world $w$, a finite action sequence $\vec{z}$ and a ground action term $t$ to a set of fluent lifter facts if $\text{Poss}(t)$ is satisfied in the situation represented by $w, \vec{z}$.

**Definition 5 (Effect Function).** Let $D$ be a BAT over the signature $\Sigma$ and $\text{Lit}$ the set of all positive and negative ground fluent atoms, given as follows:

$$\text{Lit} := \{ F(\vec{c}), -F(\vec{c}) \mid \exists \vec{c} \in N_{\Sigma} : (\vec{c}, \phi) \in \gamma_{+\vec{c}}^D \text{ or } (\vec{c}, \phi) \in \gamma_{-\vec{c}}^D \}$$

The effect function $E : W \times Z \times N_{\Sigma} \rightarrow 2^{\text{Lit}}$ for $P$ is a partial function where

$$E(w, z, t) := \{ F(\vec{c}) \in \text{Lit} \mid \exists (\vec{c}, \phi) \in \gamma_{+\vec{c}}^D \land w, z \models \phi \} \cup \{ -F(\vec{c}) \in \text{Lit} \mid \exists (\vec{c}, \phi) \in \gamma_{-\vec{c}}^D \land w, z \models F(\vec{c}) \land \phi \}$$

if $w, z \models \text{Poss}(t)$, and otherwise undefined.

Next, we show that a simplified version of Reiter’s regression operator can be defined w.r.t. an effect function, given a consistent set of fluent lifter facts and a fluent sentence. A subset $E \subseteq \text{Lit}$ is called non-contradictory if there is no fluent atom $F(\vec{c})$ such that $\{ F(\vec{c}), -F(\vec{c}) \} \subseteq E$.

**Definition 6 (Regression Operator).** Let $F(\vec{v})$ be a formula where $F$ is a fluent and $\vec{v}$ a vector of variables or constants and let $E \subseteq \text{Lit}$ be non-contradictory. We define the regression of $F(\vec{v})$ through $E$, written as $[F(\vec{v})]_{\text{Reg}(E)}$, as follows:

$$[F(\vec{v})]_{\text{Reg}(E)} := \left( F(\vec{v}) \land \bigwedge_{F(\vec{c}) \in E} \lnot \left( \vec{v} = \vec{c} \right) \right) \lor \bigvee_{F(\vec{c}) \in E} \left( \vec{v} = \vec{c} \right)$$

If $\alpha$ is a fluent sentence, $\alpha_{\text{Reg}(E)}$ denotes the result of replacing any occurrence of a fluent $F(\vec{v})$ by $[F(\vec{v})]_{\text{Reg}(E)}$.

Clearly, the regression result $\alpha_{\text{Reg}(E)}$ is again a fluent sentence. Intuitively, if we want to know whether a formula $\alpha$ holds after executing an action with effects $E$, it is sufficient to test whether the regressed formula $\alpha_{\text{Reg}(E)}$ is satisfied in the current situation.

**Lemma 3.** Let $\mathcal{D}$ be a BAT, $w \in W$ with $w \models \mathcal{D}$, $\alpha$ a fluent sentence and $t = \text{Poss}(\vec{c})$ a ground action term. For all $z \in Z$, it holds that for $E = E(w, z, t)$, $w, z \models \alpha_{\text{Reg}(E)}$ if $w, z \cdot t \models \alpha$.

We note that an iterated application of the regression operator can be reduced to an application of the operator for a single set of fluent lifter facts. For a set $E \subseteq \text{Lit}$ we define $\neg E := \{ \lnot F \mid F \in E \}$ (modulo double negation). For two non-contradictory subsets $E, E'$ of $\text{Lit}$ and a sentence $\alpha$ it holds that

$$[\alpha_{\text{Reg}(E')} \cup E'](E) \equiv \alpha_{\text{Reg}(E' \cup \neg E')}. \quad (7)$$

3.2.2 A Finite Abstraction

Next, we identify a finite set of relevant fluent sentences (called context) by considering all tests occurring in $\delta$, all formulas defined as preconditions for ground actions occurring in $\delta$, and all formulas occurring in the ground instantiated SSAs of $D$.

**Definition 7 (Context for a Golog Program).** Let $\mathcal{P} = (D, \delta)$ be a program. A context $C$ for $\mathcal{P}$ is a finite set of fluent sentences satisfying the following condition: Let $\alpha$ be a fluent sentence. If

- $\alpha$ is a test in $\delta$,
- or $\alpha = \varphi_+^D$ and there is a ground action $A(\vec{c})$ in $\delta$ with $\text{Poss}(\vec{c}) \equiv \varphi \in D_{\text{pos}}$,
- or $\alpha = \varphi$ and there exists a ground action $t$ in $\delta$ and a vector of constants $\vec{c}$ satisfying that $\{ \vec{c}, \varphi \} \in \gamma_{+\vec{c}}^D$, $\{ \vec{c}, \varphi \} \in \gamma_{-\vec{c}}^D$, or $\{ \vec{c}, \varphi \} \in \gamma_{\vec{c}}^D$ for some $\phi$,

then $\{ \alpha, \lnot \alpha \} \subseteq C$. We require that $C$ is closed under negation.

The following lemma shows some properties of $C$. Intuitively, if we consider a situation consisting of a world and a finite sequence of ground actions, then the effects of applying a ground action in that situation depend only on the formulas in $C$ that are satisfied or not satisfied in that situation. Furthermore, whether a program configuration is final in a world or has a successor configuration in this world only depends on the context as well.

**Lemma 4.** Let $C$ be a context for a program $\mathcal{P} = (D, \delta)$. Let $w_0, w_1 \in W$ satisfying $D$ and $z_0, z_1 \in Z$ such that $w_0, z_0 \models \alpha$ iff $w_1, z_1 \models \alpha$ for all $\alpha \in C$.

1. Let $t$ be a ground action that occurs in $\delta$. It holds that $E(w_0, z_0, t) = E(w_1, z_1, t)$.
2. Let $\rho$ be a program that contains only ground actions from $\delta$ and all tests in $\rho$ are contained in $C$.

(a) $\langle z_0, \rho \rangle \models w_0 \iff \langle z_1, \rho \rangle \models w_1$.
(b) $\langle z_0, \rho \rangle \rightarrow_{\rho} \langle z_1, \rho' \rangle \iff \langle z_1, \rho \rangle \rightarrow_{\rho} \langle z_1, \rho' \rangle$.

For the identification of equivalence classes, note that it is not sufficient to consider only contexts within single states. We rather have to make sure that equivalence is preserved when doing a transition. For this purpose, we define types of worlds such that two worlds of the same type will satisfy the same temporal properties. In the following we use the notation $2^{\text{Lit}}$ to denote the set of all non-contradictory subsets of $\text{Lit}$. First, we need the notion of a set of type elements for a program $\mathcal{P}$ and a context $C$ for $\mathcal{P}$:

$$TE(\mathcal{P}, C) := \{ (\alpha, E) \mid \alpha \in C, E \in 2^{\text{Lit}} \}.$$
\textbf{Definition 8 (Type of a World).} Let $\mathcal{P}$ be a program, $\mathcal{C}$ a context for $\mathcal{P}$ and $w$ a world with $w \models \mathcal{P}$. The type of $w$ w.r.t. $\mathcal{P}$ and $\mathcal{C}$ is:

$$\text{type}(w) := \{ (\alpha, E) \in TE(\mathcal{P}, \mathcal{C}) \mid w, \epsilon \models \alpha^{R(E)} \}.$$

\textbf{Example 1.} Consider a single fluent $\text{OnTable}(x)$, an action $\text{remove}(z)$ and an object constant $b$. The initial theory is given by $D_0 = \{ \text{OnTable}(b) \}$, $D_{\text{out}}$ contains a single SSA $\Box[a] \text{OnTable}(x) \equiv \text{OnTable}(x) \land \neg a = \text{remove}(x)$.

As a context for the BAT $\mathcal{D}$ and program $\text{remove}(b)$ we choose the following set consisting of two formulas (abbreviated by $\alpha_0$ and $\alpha_3$, respectively) and their negations

$$\mathcal{C} = \{ \alpha_0 : (\neg)\text{OnTable}(b), \alpha_3 : (\neg)\exists x. \text{OnTable}(x) \}.$$

We consider two worlds $w_0, w_1$ such that

1. $w_0, \epsilon \models \text{OnTable}(b)$ and
2. $w_0, \epsilon \not\models \text{OnTable}(b)$ for all $b' \in \mathcal{N}_O$ with $b \neq b'$ and
3. in $w_1$ it holds that
4. $w_1, \epsilon \models \text{OnTable}(b)$ and
5. $w_1, \epsilon \not\models \text{OnTable}(b')$ for some $b' \in \mathcal{N}_O$ with $b \neq b'$,

We have to consider three non-contradictory sets of literals $L_0 = \emptyset$, $L^+ = \{ \text{OnTable}(b) \}$ and $L^- = \{ (\neg)\text{OnTable}(b) \}$. The different types of $w_0$ and $w_1$ are given by:

- $\text{type}(w_0) := \{ (\alpha_0, L_0), (\alpha_3, L_0), (\alpha_0, L^+), (\alpha_3, L^+), (\neg \alpha_0, L^-), (\neg \alpha_3, L^-) \}$;
- $\text{type}(w_1) := \{ (\alpha_0, L_0), (\alpha_3, L_0), (\alpha_0, L^+), (\alpha_3, L^+), (\neg \alpha_0, L^-), (\neg \alpha_3, L^-) \}$.

In this simple example $b$ is the only object known to be on the table initially and it is the only object that can be affected by an action. But nevertheless, since we have only complete information about the initial world, we also have to consider possibly unknown objects. For example, we don’t know whether there is exactly one object on the table or not. As we see here, the type of $w_1$ is different from the type of $w_0$, because the formula $\exists x. \text{OnTable}(x)$ in context $\mathcal{C}$ remains true in $w_1$ after removing the object $b$.

\textbf{Lemma 5.} Consider two worlds $w, w'$ and their types w.r.t. $\mathcal{P}$ and $\mathcal{C}$. It holds that:

1. Let $z \in \mathcal{N}_A^0$ be a sequence of ground actions that occur in $\delta$ and $t$ a ground action occurring in $\delta$. If $\text{type}(w) = \text{type}(w')$, then $\mathcal{E}(w, z, t) = \mathcal{E}(w', z, t)$.
2. Let $\alpha \in \mathcal{C}$ and $z \in \mathcal{N}_A^0$ a sequence of ground actions that occur in $\delta$. If $\text{type}(w) = \text{type}(w')$, then $w, z, \epsilon \models \alpha$ iff $w', z, \epsilon \models \alpha$.

As a consequence of this lemma we can determine $\mathcal{E}(w, z, t)$ based on $\text{type}(w)$. Consider a sequence $z = t_0 t_1 \cdots t_n$ of ground actions in $\delta$. Using the equivalence (7) we can accumulate the set of effects of each prefix of $z$ into a single set of literals. The accumulated set of effects for $z$ in world $w$ is denoted by $E(w, z)$. It clearly holds that $w, z, \epsilon \models \alpha$ iff $w, \epsilon \models \alpha^{R(E(w, z))}$ for any fluent sentence $\alpha \in \mathcal{C}$.

As the equivalence relation on states of the transition system can thus be defined as follows:
the quotient transition system from the initial state \((\tau, \emptyset, \delta)\) can be easily obtained from the type representation \(\tau\). For more details we refer to the technical report [22].

Having this finite abstraction of the transition system, the verification problem boils down to a propositional CTL\(^*\) model checking problem, as described above. The complexity of this decision procedure is mainly determined by the complexity of the test whether a set of type elements \(\tau\) represents the type of a world. To do this we have to test consistency of an exponentially large \(C^2\) knowledge base. Knowledge base consistency in \(C^2\) can be decided in \textsc{Nexptime} [20]. Therefore, checking whether \(\tau\) is a type can be done in \textsc{2-Nexptime}. It turns out that \textsc{2-Nexptime} is also an upper bound for the overall complexity.

**Theorem 1.** Satisfiability of an \(\mathcal{ES}\)-\(C^2\)-CTL\(^*\) formula \(\Phi\) in a Golog program \(P = (D, \delta)\) is decidable in \textsc{2-Nexptime}.

4 Related Work

Apart from what was discussed in the introduction, other researchers have looked at decidable verification in the context of the Situation Calculus. One line of research is followed by De Giacomo, Lespérance and Patrizi [8] who show decidability for first-order \(\mu\)-calculus properties for a class of BATs that only admit finitely many instances of fluents to hold. In contrast, our approach also allows fluents with infinite extensions. Moreover, their notion of boundedness is a semantical condition that is in general undecidable to check, whereas our approach relies on purely syntactical restrictions. In related work, Hariri et al. [14] consider the verification of \(\mu\)-calculus properties in the context of light-weight DLs, where new information may be added at any time. Among other things, they show that decidability obtains if the added information is bounded. But in contrast to the action theories we consider, the frame problem is not solved in their underlying action formalism, i.e. the actions are memoryless.

5 Conclusion

We presented results on the verification of temporal properties for Golog programs. We have extended previous decidability results as presented in [7] and [4] to a fragment that uses the expressive, yet decidable fragment \(C^2\) of FOL as base logic, that allows for local-effect actions with arbitrarily many arguments, and that admits properties expressed in the branching-time temporal logics CTL\(^*\) over \(C^2\) axioms. Decidability is obtained by constructing a finite abstraction of the infinite transition system induced by the Golog program and its action theory. Note that the restrictions we impose (decidable base logic, ground actions only, and local-effects) are all necessary in the sense that we can show [22] that dropping any one of them would instantly render the verification problem undecidable again.

There are many directions for future work. We plan to further investigate the exact computational complexity and evaluate the approach practically by means of an implementation. It would also be interesting to extend our results in terms of expressiveness, e.g. by considering SSAs that go beyond local-effect theories, or by re-introducing non-ground actions in a limited, decidable fashion.

Acknowledgements

This work was supported by the German National Science Foundation (DFG) research unit FOR 1513 on Hybrid Reasoning for Intelligent Systems (http://www.hybrid-reasoning.org).

**REFERENCES**


