# On the Decidability of Verifying LTL Properties of GOLOG Programs*

**Benjamin Zarrieß**
Theoretical Computer Science
TU Dresden, Germany
*zarriess@tcs.inf.tu-dresden.de*

**Jens Claßen**
Knowledge-Based Systems Group
RWTH Aachen University, Germany
*classen@kbsg.rwth-aachen.de*

## Abstract

The high-level action programming language GOLOG is a useful means for modeling the behavior of autonomous agents such as mobile robots. It relies on a representation given in terms of a logic-based action theory in the Situation Calculus (SC). To guarantee that the possibly non-terminating execution of a GOLOG program leads to the desired behavior of the agent, it is desirable to (automatically) verify that it satisfies certain requirements given in terms of temporal formulas. However, due to the high (first-order) expressiveness of the GOLOG language, the verification problem is in general undecidable. In this paper we show that for a fragment of the GOLOG language defined on top of the decidable logic $C^2$, the verification problem for linear time temporal properties becomes decidable, which extends earlier results to a more expressive fragment of the input formalism. Moreover, we justify the involved restrictions on program constructs and action theory by showing that relaxing any of these restrictions instantly renders the verification problem undecidable again.

## Introduction

The GOLOG (De Giacomo, Lespérance, and Levesque 2000; Levesque et al. 1997) family of high-level action programming languages and its underlying logic, the Situation Calculus (McCarthy and Hayes 1969; Reiter 2001), have proven to be useful means for the control of autonomous agents such as mobile robots (Burgard et al. 1999).

Before actually deploying such a program on the robot and executing it in the real world, it is often desirable if not crucial to verify that it meets certain requirements such as safety, liveness and fairness properties. Moreover, the verification is preferably done using an *automated* method, since manual, meta-theoretic proofs such as done in (De Giacomo, Ternovska, and Reiter 1997) tend to be tedious and prone to errors. For this purpose, Claßen and Lakemeyer (2008) proposed a new logic $\mathcal{ESG}$, an extension of the modal Situation Calculus variant $\mathcal{ES}$ (Lakemeyer and Levesque 2010) by constructs that allow to express temporal properties of GOLOG programs. They moreover provided algorithms for the verification of a subset of the logic that resembles the branching-time temporal logic CTL. Their methods rely on regression-based reasoning and a graph representation of GOLOG programs to do a systematic exploration of a program's configuration space within a fixpoint approximation loop. While the procedures are proven to be sound, no general guarantee can be given for termination. This is not at all surprising in light of the fact that in the presence of unrestricted first-order expressiveness, the verification problem is highly undecidable.

For verifying properties of GOLOG programs in practice, guaranteed termination would of course be very desirable. The obvious course of action for achieving this is to restrict the input formalism in an appropriate manner such that the verification problem becomes decidable. Ideally, we could do the verification within the very same formalism and reasoning methods that are used for the actual control of the agent, while retaining as much first-order expressiveness as possible.

Multiple approaches for addressing this problem have been proposed. Instead of using the full first-order expressiveness of the Situation Calculus or $\mathcal{ES}$, Baader, Liu and ul Mehdi (2010) resort to an action language (Baader et al. 2005) based on the decidable Description Logic (DL) $\mathcal{ALC}$ (Baader et al. 2003) to represent pre- and postconditions of actions, where properties are expressed by a variant of LTL over $\mathcal{ALC}$ axioms (Baader, Ghilardi, and Lutz 2008). Second, they approximate programs by finite Büchi automata accepting infinite sequences of DL actions. They could show that under these restrictions, verification reduces to a decidable reasoning task within the underlying DL.

While this was a step in the right direction, the restrictions that were employed were comparably harsh. In particular, representing action effects within $\mathcal{ALC}$ only allows for basic STRIPS-style addition and deletion of literals. Moreover, approximating programs through Büchi automata loses two important features of GOLOG, namely the possibility to include pick operators for non-deterministically chosing arguments of subprograms, as well as test conditions that allow to constrain program execution by requiring a given formula to hold, which is in particular useful for expressing imperative programming constructs such as while loops and if-then-else conditionals. The latter is addressed by Baader and Zarrieß (2013) who show that Baader, Liu and ul Mehdi's re-

*Supported by DFG Research Unit FOR 1513, project A1

sults can indeed be lifted to a more expressive fragment of GOLOG that includes test conditions. They obtain decidability by proving that the potentially infinite transition system induced by the GOLOG program can always be represented by a finite one that admits the same execution traces.

Another approach is taken by Claßen, Liebenberg and Lakemeyer (2013). They consider the two possible sources of non-termination of Claßen and Lakemeyer's original verification method: On the one hand, in each iteration one needs to check the validity of regressed formulas. Since this problem is already undecidable in the general first-order case, the idea is to restrict oneself to a two-variable fragment of the Situation Calculus where this form of reasoning becomes decidable (Gu and Soutchanski 2010). On the other hand, it may happen that the fixpoint computation loop never converges. Claßen, Liebenberg and Lakemeyer however show that for certain classes of successor state axioms such as the ones with only *local effects* (Liu and Lakemeyer 2009), termination can indeed be guaranteed for GOLOG programs that may contain test conditions, but not the above mentioned pick operators.

In this paper, we aim at laying the foundations for consolidating these earlier approaches within a single formal framework, while even increasing expressiveness. In particular, (1) as base logic we use $C^2$, the two variable fragment of first-order logic with counting quantifiers, which subsumes both $\mathcal{ALC}$ as well as the two-variable fragment without counting quantifiers. Furthermore, we (2) formulate action effects through $\mathcal{ES}$-style successor state axioms, which goes beyond the basic STRIPS-style addition and deletion of literals. Although we again have to restrict these axioms to be local-effect, we employ the more liberal definition of (Vassos, Lakemeyer, and Levesque 2008) that allows for quantifiers within context formulas. We then (3) show that the execution traces of a GOLOG program without pick operators can be represented by a finite transition system, and that verifying LTL properties of such programs is hence decidable. Finally, we prove that all of the above restrictions are indeed necessary as dropping any one of them would instantly lead to undecidability.

The remainder of this paper is organized as follows. The following section introduces the basic notions of action theories in the $\mathcal{ES}$ variant of the Situation Calculus defined on top of $C^2$ as base logic, and define syntax and semantics of (possibly) non-terminating GOLOG programs. In the subsequent section, we define LTL over $C^2$ sentences to represent desired or unwanted properties of runs of GOLOG programs and show that the verification problem becomes decidable for a fragment of GOLOG. In the next section we show that several extensions of this fragment lead to undecidability.

Because of space constraints, detailed proofs of our results have to be omitted. They can be found in (Zarrieß and Claßen 2013).

## Preliminaries

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

In this subsection we recall the main definitions of the modal Situation Calculus variant $\mathcal{ES}$ (Lakemeyer and Levesque 2010). But instead of using full first-order logic, we restrict ourselves to the *two variable fragment with equality and counting* of FOL named $C^2$.

We start by fixing a set of *terms*. In our language we consider terms of two sorts *object* and *action*. They can be built using the following symbols: two variables $x, y$ of sort *object*, a single variable $a$ of sort *action*, an infinite set $N_I$ of *rigid object constant symbols* (i.e. 0-ary function symbols) and a finite set $N_A$ of *rigid action function symbols* with at most two arguments, all of sort object. A term is called *ground term* if it contains no variables. We denote the set of ground terms (also called "standard names") of sort object by $\mathcal{N}_O$, and those of sort action by $\mathcal{N}_A$.

To build formulas we consider *fluent* and *rigid* predicate symbols with at most two arguments of sort object and one unary predicate $Poss$ with one argument of sort action later used to define preconditions of actions. Fluents vary as the result of actions, but rigids do not. Formulas are then built using the usual logical connectives and in addition we have two modal operators $[\cdot]$ and $\Box$ for referring to future situations.

Let $N_F$ be a set of fluent predicate symbols and $N_R$ a set of rigid predicate symbols. The set of *formulas* is defined as the least set satisfying the following conditions: If $t_1, ..., t_k$ are terms and $P \in N_F \cup N_R$ a $k$-ary predicate symbol with $0 \leq k \leq 2$, then $P(t_1, ..., t_k)$ is a formula. If $t_1$ and $t_2$ are terms, then $t_1 = t_2$ is a formula. If $\alpha$ and $\beta$ are formulas, $x$ a variable and $t$ a term of sort action, then $\alpha \wedge \beta$, $\neg\alpha$, $\forall x.\alpha$, $\exists^{\leq m} x.\alpha$ and $\exists^{\geq m} x.\alpha$ with $m \in \mathbb{N}$, $\Box\alpha$ ($\alpha$ always holds) and $[t]\alpha$ ($\alpha$ holds after executing $t$) are formulas. We understand $\vee$, $\exists$, $\supset$ and $\equiv$ as the usual abbreviations and use $true$ for a tautology. A formula is called *bounded* if it contains no $\Box$; it is called *static* if its bounded and contains no $[\cdot]$ and it is called *fluent* if it is static and does not contain the predicate $Poss$. A formula is called *sentence* if it contains no free variables.

The semantics of formulas is defined in terms of *worlds*.

**Definition 1 (Worlds).** Let $\mathcal{P}_F$ be the set of all primitive formulas $F(n_1, ..., n_k)$, where $F$ is $k$-ary predicate symbol with $0 \leq k \leq 2$ and the $n_i$ are standard names. Let $\mathcal{Z} := \mathcal{N}_A^*$. A *world* $w$ is a mapping

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

satisfying the rigidity constraint: If $R$ is a rigid predicate symbol, then for all $z, z' \in \mathcal{Z}$ it holds that $w[R(n_1, \ldots, n_k), z] = w[R(n_1, \ldots, n_k), z']$. The set of all worlds is denoted by $\mathcal{W}$.

A world thus maps primitive formulas to truth values. The rigidity constraint ensures that rigid symbols do not take different values in different situations, as expected. The unique names assumption for actions and object constants is also part of our semantic definition.

We use the symbol $\langle\rangle$ to denote the empty sequence of action standard names. We are now equipped to define the truth of formulas:

**Definition 2 (Satisfaction of Formulas).** Given a world $w \in \mathcal{W}$ and a sentence $\alpha$, we define $w \models \alpha$ as $w, \langle\rangle \models \alpha$, where for any $z \in \mathcal{Z}$:

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. $w, z \models \alpha \wedge \beta$ 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 $w, z \models \alpha^x_n$ for all $n \in \mathcal{N}_x$;
6. $w, z \models \exists^{\leq m} x.\alpha$ iff $|\{n \in \mathcal{N}_x \mid w, z \models \alpha^x_n\}| \leq m$;
7. $w, z \models \exists^{\geq m} x.\alpha$ iff $|\{n \in \mathcal{N}_x \mid w, z \models \alpha^x_n\}| \geq m$;
8. $w, z \models \Box\alpha$ iff $w, z \cdot z' \models \alpha$ for all $z' \in \mathcal{Z}$;
9. $w, z \models [t]\alpha$ iff $w, z \cdot t \models \alpha$;

Above, $\mathcal{N}_x$ refers to the set of all standard names of the same sort as $x$. We moreover use $\alpha^x_n$ to denote the result of simultaneously replacing all free occurrences of $x$ by $n$.

**Basic Action Theories** Given a signature as described above, we now define a theory as a set of axioms of a predefined structure in order to model a dynamic application domain.

**Definition 3.** A *basic action theory* (BAT) $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{pre}} \cup \mathcal{D}_{\text{post}}$ describes the dynamics of a specific application domain, where

1. $\mathcal{D}_0$, *the initial database*, is a finite set of fluent sentences describing the initial state of the world.
2. $\mathcal{D}_{\text{pre}}$ is a set of *precondition axioms* such that for any action function $A \in N_A$ there is an axiom of the form $\Box Poss(A(\vec{x})) \equiv \varphi$, with $\varphi$ being a fluent formula with free variables $\vec{x}$. Note that $\vec{x}$ can be either empty or it consists of one or two variables.
3. $\mathcal{D}_{\text{post}}$ is a finite set of *successor state axioms* (SSAs), one for each fluent $F \in N_F$, incorporating Reiter's (2001) solution to the frame problem, and encoding the effects the actions have on the different fluents. The SSA for a fluent $F$ has the form $\Box[a]F(\vec{x}) \equiv \gamma^+_F \vee F(\vec{x}) \wedge \neg\gamma^-_F$, where $\gamma^+_F$ and $\gamma^-_F$ are fluent formulas with free variables $\vec{x}$ and $a$.

## 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:

$$\delta ::= \langle\rangle \mid t \mid \alpha? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \pi x.\delta \mid \delta_1 \| \delta_2 \mid \delta^* \qquad (1)$$

That is we allow the empty program $\langle\rangle$, primitive actions $t$ (where $t$ can be any action term), tests $\alpha?$ (where $\alpha$ is a fluent sentence), sequence, nondeterministic branching, nondeterministic choice of argument, concurrency, and nondeterministic iteration.

**Definition 4 (Golog program).** A *Golog program* $\mathcal{P} = (\mathcal{D}, \delta)$ consists of a BAT $\mathcal{D}$ over the signature $\Sigma = (N_F, N_R, N_I, N_A)$ and a program expression $\delta$ where the action terms and tests in $\delta$ are built from symbols from $\Sigma$. The set of action terms occurring in $\delta$ is denoted by $Act$.

Program expressions are interpreted as follows. A *configuration* $\langle z, \delta\rangle$ consists of an action sequence $z$ 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 5 (Program Transition Semantics).** The transition relation $\xrightarrow{w}$ among configurations, given a world $w$ with $w \models \mathcal{D}$, is the least set satisfying

1. $\langle z, t\rangle \xrightarrow{w} \langle z \cdot t, \langle\rangle\rangle$, if $w, z \models Poss(t)$;
2. $\langle z, \delta_1; \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \gamma; \delta_2\rangle$, if $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot t, \gamma\rangle$;
3. $\langle z, \delta_1; \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$,
   if $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$;
4. $\langle z, \delta_1 | \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$,
   if $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$ or $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$;
5. $\langle z, \pi x.\delta\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$,
   if $\langle z, \delta^x_n\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$ for some $n \in \mathcal{N}_x$;
6. $\langle z, \delta_1 \| \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta' \| \delta_2\rangle$, if $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$;
7. $\langle z, \delta_1 \| \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta_1 \| \delta'\rangle$, if $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$.
8. $\langle z, \delta^*\rangle \xrightarrow{w} \langle z \cdot t, \gamma; \delta^*\rangle$, if $\langle z, \delta\rangle \xrightarrow{w} \langle z \cdot t, \gamma\rangle$;

The set of final configurations $\mathcal{F}^w$ of a world $w$ is the smallest set such that

1. $\langle z, \alpha?\rangle \in \mathcal{F}^w$ if $w, z \models \alpha$;
2. $\langle z, \delta_1; \delta_2\rangle \in \mathcal{F}^w$ if $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \in \mathcal{F}^w$;
3. $\langle z, \delta_1 | \delta_2\rangle \in \mathcal{F}^w$ if $\langle z, \delta_1\rangle \in \mathcal{F}^w$ or $\langle z, \delta_2\rangle \in \mathcal{F}^w$;
4. $\langle z, \pi x.\delta\rangle \in \mathcal{F}^w$ if $\langle z, \delta^x_n\rangle \in \mathcal{F}^w$ for some $n \in \mathcal{N}_x$;
5. $\langle z, \delta_1 \| \delta_2\rangle \in \mathcal{F}^w$ if $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \in \mathcal{F}^w$;
6. $\langle z, \delta^*\rangle \in \mathcal{F}^w$; $\langle z, \langle\rangle\rangle \in \mathcal{F}^w$.

Let $\xrightarrow{w}{}^*$ denote the reflexive and transitive closure of $\xrightarrow{w}$. The *set of reachable subprograms*, denoted by $\mathsf{sub}(\delta)$, is defined as follows:

$$\mathsf{sub}(\delta) := \{\delta' \mid \exists w \models \mathcal{D}, z \in \mathcal{Z} \text{ s.t. } \langle\langle\rangle, \delta\rangle \xrightarrow{w}{}^* \langle z, \delta'\rangle\}$$

Note, that by induction on the size of $|\delta|$ it can be shown that for a transition $\langle z, \delta\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$ it holds that $w, z \models Poss(t)$.

To handle non-terminating, terminating and failing runs of a program uniformly we proceed as follows:

First, we introduce two fresh 0-ary fluents $Term$ and $Fail$ and two 0-ary action functions $\epsilon$ and $\mathfrak{f}$. We assume that the axioms $\Box Poss(\epsilon) \equiv true$ and $\Box Poss(\mathfrak{f}) \equiv true$ belong to $\mathcal{D}_{\text{pre}}$. Then we include the following additional SSAs in $\mathcal{D}_{\text{post}}$ for $Term$ and $Fail$: $\Box[a]Term \equiv a = \epsilon \vee Term$, $\Box[a]Fail \equiv a = \mathfrak{f} \vee Fail$.

Now, we define an infinite transition system for a given program $\mathcal{P} = (\mathcal{D}, \delta)$.

**Definition 6 (Transition System).** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program. The *transition system* $T_{\mathcal{P}} = (Q, \rightarrow, I)$ induced by $\mathcal{P}$ consists of the set of *states*

$$Q := \{(w, z, \delta') \mid w \models \mathcal{D},\ z \in \mathcal{Z},\ \delta' \in \mathsf{sub}(\delta)\},$$

a transition relation $\rightarrow \subseteq Q \times \mathcal{N}_A \times Q$ and a set of *initial states* $I \subseteq Q$, which are defined as follows:

- $I := \{(w, \langle\rangle, \delta) \mid w, \langle\rangle \models \mathcal{D}_0\}$
- It holds that $(w, z, \rho) \xrightarrow{t} (w, z \cdot t, \rho')$ if one of the following conditions is satisfied:
  1. $\langle z, \rho\rangle \xrightarrow{w} \langle z \cdot t, \rho'\rangle$.
  2. $\langle z, \rho\rangle \in \mathcal{F}^w$, $t = \epsilon$ and $\rho' = \langle\rangle$.
  3. In case there is no $\langle z'', \rho''\rangle$ s.t. $\langle z, \rho\rangle \xrightarrow{w} \langle z'', \rho''\rangle$ and $\langle z, \rho\rangle \notin \mathcal{F}^w$, we have $t = \mathfrak{f}$ and $\rho' = \rho$.

A *run* of a program $\mathcal{P}$ is now defined as an infinite path in the corresponding transition system $T_\mathcal{P}$ starting in an initial state. A run in $T_\mathcal{P} = (Q, \to, I)$ has the following form:

$$\mathfrak{r} = (w, z_0, \rho_0) \xrightarrow{t_0} (w, z_1, \rho_1) \xrightarrow{t_1} (w, z_2, \rho_2) \xrightarrow{t_2} \cdots$$

with $(w, z_0, \rho_0) \in I$, $z_0 = \langle\rangle$ and $z_i = z_{i-1} \cdot t_{i-1}$ for $i = 1, 2, \cdots$. The *action trace* of a run $\mathfrak{r}$, denoted by $act(\mathfrak{r})$, is an infinite word over $\mathcal{N}_A$ given by $act(\mathfrak{r}) = t_0 t_1 t_2 \cdots$.

## Verification

First, we define the temporal logic used to specify properties of a given program. We define the logic $\mathcal{ES}$-$C^2$-LTL. The syntax is the same as for propositional LTL, but in place of propositions we allow for $C^2$ fluent sentences. The syntax is given by the following grammar:

$$\Phi ::= \alpha \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \boldsymbol{X}\Phi \mid \Phi_1 \boldsymbol{U} \Phi_2 \qquad (2)$$

where $\alpha$ can be any fluent sentence.

Similar to the definition of worlds we introduce the notion of a $\mathcal{ES}$-$C^2$-LTL structure to define the semantics of $\mathcal{ES}$-$C^2$-LTL.

**Definition 7 ($\mathcal{ES}$-$C^2$-LTL semantics).** A $\mathcal{ES}$-$C^2$-LTL *structure* $\mathfrak{w} = (w, \sigma)$ consists of a world $w$ and an infinite action sequence $\sigma \in \mathcal{N}_A^\omega$. For a given $i \in \mathbb{N}$, $\sigma[0...i]$ denotes the prefix of $\sigma$ up to position $i$.

Let $\Phi$ be a $\mathcal{ES}$-$C^2$-LTL formula, $\mathfrak{w} = (w, \sigma)$ a $\mathcal{ES}$-$C^2$-LTL structure and $i \in \mathbb{N}$. Validity of $\Phi$ in $\mathfrak{w}$ at time point $i$, denoted by $\mathfrak{w}, i \models \Phi$, is defined as follows:

- $\mathfrak{w}, i \models \alpha$ iff $w, \sigma[0..i] \models \alpha$
- $\mathfrak{w}, i \models \neg\Phi$ iff $\mathfrak{w}, i \not\models \Phi$
- $\mathfrak{w}, i \models \Phi_1 \wedge \Phi_2$ iff $\mathfrak{w}, i \models \Phi_1$ and $\mathfrak{w}, i \models \Phi_2$
- $\mathfrak{w}, i \models \boldsymbol{X}\Phi$ iff $\mathfrak{w}, i+1 \models \Phi$
- $\mathfrak{w}, i \models \Phi_1 \boldsymbol{U} \Phi_2$ iff $\exists k \geq i : \mathfrak{w}, k \models \Phi_2$ and $\forall j, i \leq j < k : \mathfrak{w}, j \models \Phi_1$

As usual, we use $\boldsymbol{F}\Phi$ (eventually) and $\boldsymbol{G}\Phi$ (globally) as abbreviations for $true \boldsymbol{U} \Phi$ and $\neg\boldsymbol{F}\neg\Phi$, respectively.

We can thus express properties of runs of a program. Consider a program $\mathcal{P}$ and the corresponding transition system $T_\mathcal{P}$. Let $r$ be a run of $\mathcal{P}$ starting in the initial state $(w, \langle\rangle, \delta)$. The $\mathcal{ES}$-$C^2$-LTL structure corresponding to $r$ is given by $\mathfrak{w}(r) = (w, act(r))$.

**Definition 8 (Verification Problem).** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program, $T_\mathcal{P}$ the corresponding transition system and $\Phi$ a $\mathcal{ES}$-$C^2$-LTL formula. The formula $\Phi$ is *valid* in $\mathcal{P}$, written as $T_\mathcal{P} \models \Phi$, if for all runs $r$ of $\mathcal{P}$ it holds that $\mathfrak{w}(r), 0 \models \Phi$. The formula $\Phi$ is *satisfiable* in $\mathcal{P}$ if there exists a run $r$ of $\mathcal{P}$ such that $\mathfrak{w}(r), 0 \models \Phi$.

Note, that the logic $\mathcal{ES}$-$C^2$-LTL is expressive enough to encode several variants of the verification problem. For example, consider domain constraints expressed as a conjunction $\varphi$ of $C^2$ fluent sentences. The problem of whether these constraints persist during the execution of a program $\mathcal{P}$ corresponds to validity of the formula $\boldsymbol{G}\varphi$ in the program $\mathcal{P}$. Furthermore, the fluents $Term$ and $Fail$ can be used to encode properties of, for example, terminating and non-failing runs of a program.

In the following we focus only on decidability of the satisfiability problem. This is sufficient since it clearly holds that $\Phi$ is valid in $\mathcal{P}$ iff $\neg\Phi$ is not satisfiable in $\mathcal{P}$.

### Programs over Ground and Local-effect Actions

The main problem we have to deal with when testing satisfiability of a property in a Golog program is that the corresponding transition system is in general infinite. To achieve decidability we have to make certain restriction on the action theory and on the programs. In the following we show that for a so called local-effect basic action theory and a program where we disallow the pick operator $\pi x.\delta$, a finite abstraction of the infinite transition system can be constructed that preserves the relevant information to verify the property.

In particular, we consider Golog programs $\mathcal{P} = (\mathcal{D}, \delta)$, where $\delta$ is a program expression that can be built according to the following grammar

$$\delta ::= \langle\rangle \mid t \mid \alpha? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \delta_1 \| \delta_2 \mid \delta^* \qquad (3)$$

where $t$ is a ground action term and $\alpha$ a fluent sentence.

Since we have to consider in this restricted setting only finitely many ground actions, the set of reachable subprograms $\mathsf{sub}(\delta)$ is finite and bounded by the size of $\delta$. The size of a program expression $|\delta|$ is defined as the number of ground actions, tests and program constructs occurring in $\delta$.

**Lemma 9.** *Let $\delta$ be a program expression over ground actions. The cardinality of $\mathsf{sub}(\delta)$ is exponentially bounded in the size $|\delta|$ of $\delta$.*

*Proof.* The proof of this lemma, as well as all other proofs can be found in the technical report (Zarrieß and Claßen 2013). □

In addition to the program expressions, we also restrict the structure of the action theory to be *local-effect*. Intuitively, this means a fluent can change its value as result of an action application only for arguments that occur as parameters of this action or are explicitly mentioned as constants in the SSA.

**Definition 10 (Local-effect SSAs).** A successor state axiom is *local-effect* if both $\gamma_F^+$ and $\gamma_F^-$ are disjunctions of formulas that are either of the form

- $\exists \vec{z}[a = A(\vec{y}) \wedge \phi(\vec{y})]$, where $A$ is an action function, $\vec{y}$ contains $\vec{x}$, and $\vec{z}$ is the remaining variables of $\vec{y}$, and $\phi(\vec{y})$ is a fluent formula with free variables $\vec{y}$; or of the form
- $[a = A \wedge \vec{x} = \vec{c} \wedge \phi]$, where $A$ is 0-ary action function, $\vec{c}$ a vector of constants from $N_I$ and $\phi$ a fluent sentence.

The formulas $\phi(\vec{y})$ and $\phi$ are called *context formulas*. A BAT $\mathcal{D}$ is local-effect if each SSA in $\mathcal{D}_{\text{post}}$ is local-effect.

Note that this definition subsumes the definitions of local-effect BATs given in (Vassos, Lakemeyer, and Levesque 2008; Liu and Lakemeyer 2009). Moreover, the DL-based action descriptions introduced in (Baader et al. 2005) can be translated into a local-effect BAT according to the above definition. Also note that even in this restricted setting the transition system of the Golog program is infinite: We still have to consider infinitely many possible worlds over an infinite domain of standard names, since the tests in the program, the preconditions and the context formulas in the SSAs possibly contain quantifiers that quantify over the whole domain.

We basically follow the approach from (Baader and Zarrieß 2013) to test whether a $\mathcal{ES}$-$C^2$-LTL formula $\Phi$ is satisfiable in a program $\mathcal{P} = (\mathcal{D}, \delta)$ where $\mathcal{D}$ is a local-effect BAT and $\delta$ a program expression over ground actions. It consists of the following steps: First we construct a finite abstraction of the infinite transition system that retains enough information to test satisfiability. To do this we introduce the notion of a *type of a world* such that if in any situation in two worlds the same relevant formulas are satisfied, then these two worlds are of the same type. Having these types, we define an equivalence relation on the states of the transition system. Then it is possible to construct the finite quotient transition system w.r.t. this equivalence relation. Essentially, this works because the computation of the (finitely many) world types reduces to a bounded number of consistency checks in the underlying decidable logic $C^2$.

Given this finite abstraction we can then apply standard techniques to verify the $\mathcal{ES}$-$C^2$-LTL formula, as an LTL formula can be translated into an automaton accepting exactly those structures satisfying the formula. The satisfiability test can then be reduced to a reachability test in the finite product of the automaton with the abstract transition system.

First, we introduce some auxiliary notions needed to define the types of worlds.

**Regression with Ground Actions** Since we have only finitely many ground actions in our program it is enough to consider only a simplified form of the SSAs, called *ground instantiations* where the action variable $a$ in the SSA is replaced with a ground action term. Consider the SSA of a fluent $F(\vec{x})$ of the form $\Box[a]F(\vec{x}) \equiv \gamma_F^+ \vee F(\vec{x}) \wedge \neg\gamma_F^-$ and a ground action term $t = A(\vec{c})$. For the ground instantiated SSA for $F(\vec{x})$ with $t$, given as $\Box[t]F(\vec{x}) \equiv \gamma_{Ft}^{+a} \vee F(\vec{x}) \wedge \neg\gamma_{Ft}^{-a}$, it holds that both $\gamma_{Ft}^{+a}$ and $\gamma_{Ft}^{-a}$ are equivalent to a disjunctions of the form

$$\vec{x} = \vec{c_1} \wedge \phi_1 \vee \cdots \vee \vec{x} = \vec{c_n} \wedge \phi_n, \tag{4}$$

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

From now on we assume that in the ground instantiated SSAs the formulas $\gamma_{Ft}^{+a}$ and $\gamma_{Ft}^{-a}$ are of the form (4). We use the notation $(\vec{c}, \phi) \in \gamma_{Ft}^{+a}$ and $(\vec{c}, \phi) \in \gamma_{Ft}^{-a}$ if there exists a disjunct of the form $\vec{x} = \vec{c} \wedge \phi$ in $\gamma_{Ft}^{+a}$ and $\gamma_{Ft}^{-a}$, respectively.

As we will see in the following, the restriction to ground actions *and* local-effect BAT makes it is possible to represent the effects of executing a ground action as a finite set of fluent literals that can be read off from the ground instantiated SSAs. We define an *effect function* mapping a world $w$, a finite action sequence $z$ and a ground action $t$ to a set of fluent literals if the precondition $Poss(t)$ is satisfied in the situation represented by $w, z$.

**Definition 11 (Effect Function).** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program and $Lit$ be the set of all positive and negative ground fluent atoms, given as follows:

$$Lit := \{F(\vec{c}), \neg F(\vec{c}) \mid \exists t \in Act, \phi : (\vec{c}, \phi) \in \gamma_{Ft}^{+a} \text{ or } (\vec{c}, \phi) \in \gamma_{Ft}^{-a}\}$$

The *effect function* $\mathcal{E} : \mathcal{W} \times \mathcal{Z} \times Act \to 2^{Lit}$ for $\mathcal{P}$ is a partial function and if $w, z \models Poss(t)$ then

$$\mathcal{E}(w, z, t) := \{F(\vec{c}) \mid \exists(\vec{c}, \phi) \in \gamma_{Ft}^{+a} \wedge w, z \models \phi\} \cup \{\neg F(\vec{c}) \mid \exists(\vec{c}, \phi) \in \gamma_{Ft}^{-a} \wedge w, z \models F(\vec{c}) \wedge \phi\}$$

otherwise, if $w, z \not\models Poss(t)$, then $\mathcal{E}(w, z, t)$ is undefined.

Next, we show that Reiter's version of the regression operator can be reformulated using the effect function. We define our version of the regression operator for a consistent set of fluent literals and a fluent sentence. A subset $E \subseteq Lit$ is called *non-contradictory* if there is no fluent atom $F(\vec{c})$ such that $\{F(\vec{c}), \neg F(\vec{c})\} \subseteq E$.

**Definition 12 (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 $E \subseteq Lit$ be non-contradictory. We define the regression of $F(\vec{v})$ through $E$, written as $[F(\vec{v})]^{R(E)}$, as follows:

$$[F(\vec{v})]^{R(E)} := \left(F(\vec{v}) \vee \bigvee_{F(\vec{c}) \in E} (\vec{v} = \vec{c})\right) \wedge \bigwedge_{\neg F(\vec{c}) \in E} (\vec{v} \neq \vec{c})$$

Let $\alpha$ be a fluent sentence. The fluent sentence $\alpha^{R(E)}$ is obtained by replacing any occurrence of a fluent $F(\vec{v})$ by $[F(\vec{v})]^{R(E)}$.

Clearly, it holds that the regression result $\alpha^{R(E)}$ is again a $C^2$ fluent sentence. Intuitively, if we want to know whether a formula $\alpha$ holds after executing an action with effects given by $E$, it is sufficient to test whether the regressed formula $\alpha^{R(E)}$ is satisfied in the current situation.

**Lemma 13.** *Let $\mathcal{D}$ be a BAT, $w \in \mathcal{W}$ with $w \models \mathcal{D}$, $\alpha$ a fluent sentence and $t = A(\vec{c})$ a ground action term. For all $z \in \mathcal{Z}$ it holds that $w, z \models \alpha^{R(E)}$ iff $w, z \cdot t \models \alpha$ with $E = \mathcal{E}(w, z, t)$.*

This, means $\cdot^{R(E)}$ can act as a one-step regression operator. But also an iterated application of the regression operator can be reduced to an application of the operator for a single set of fluent literals. For a set $E \subseteq Lit$ we define $\neg E := \{\neg l \mid l \in E\}$ (modulo double negation). It holds that

$$\left[\alpha^{R(E')}\right]^{R(E)} \equiv \alpha^{R(E \setminus \neg E' \cup E')}. \tag{5}$$

for two non-contradictory subsets $E$ and $E'$ of $Lit$.

**Types of worlds** Next, we identify the finite set of relevant fluent sentences occurring in the program and the action theory that is called *context*.

**Definition 14 (context).** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a program. A *context* $\mathcal{C}$ 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 $\alpha = \varphi(\vec{c})$ and there is a ground action $A(\vec{c})$ in $\delta$ with $\Box Poss(A(\vec{x})) \equiv \varphi(\vec{x}) \in \mathcal{D}_{\text{pre}}$;
- or $\alpha = \phi$ and there exists a ground action $t$ in $\delta$, a ground instantiated SSA for a fluent $F$ with $t$ and there is a disjunct of the form $\vec{x} = \vec{c} \wedge \phi$ in $\gamma^{+a}_{F\,t}$ or $\gamma^{-a}_{F\,t}$ of the SSA,
- or $\alpha = F(\vec{c})$ and there exists a ground action $t$ in $\delta$, a ground instantiated SSA for $F$ with $t$ and there is a disjunct of the form $\vec{x} = \vec{c} \wedge \phi$ in $\gamma^{-a}_{F\,t}$ of the SSA,

then $\alpha \in \mathcal{C}$. Further we close up $\mathcal{C}$ under negation.

Next, we show some properties of $\mathcal{C}$. 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 this situation depend only on the formulas in $\mathcal{C}$ that are satisfied or not satisfied in this situation. Furthermore, we show that whether a program configuration is final in a world or has a successor configuration in this world only depends on the context.

**Lemma 15.** *Let $\mathcal{C}$ be a context for a program $\mathcal{P} = (\mathcal{D}, \delta)$. Let $w_0, w_1 \in \mathcal{W}$ satisfying $\mathcal{D}$ and $z_0, z_1 \in \mathcal{Z}$ such that $w_0, z_0 \models \alpha$ iff $w_1, z_1 \models \alpha$ for all $\alpha \in \mathcal{C}$.*

1. *Let $t$ be a ground action that occurs in $\delta$. It holds that $\mathcal{E}(w_0, z_0, t) = \mathcal{E}(w_1, z_1, t)$.*
2. *Let $\rho$ be a program that contains only ground actions from $\delta$ and all tests in $\rho$ are contained in $\mathcal{C}$.*
   (a) *It holds that $\langle z_0, \rho\rangle \in \mathcal{F}^{w_0}$ iff $\langle z_1, \rho\rangle \in \mathcal{F}^{w_1}$.*
   (b) *It holds that*
   $$\langle z_0, \rho\rangle \xrightarrow{w_0} \langle z_0 \cdot t, \rho'\rangle \text{ iff } \langle z_1, \rho\rangle \xrightarrow{w_1} \langle z_1 \cdot t, \rho'\rangle.$$

Based on the context we partition the worlds satisfying the BAT into finitely many equivalence classes. To do this we introduce the notion of a *type* of a world. Intuitively, if two worlds are of the same type, the same temporal properties are satisfied in both worlds if we execute the program. First, we define a set of *type elements* for a program $\mathcal{P}$ and a context $\mathcal{C}$ for $\mathcal{P}$:

$$TE(\mathcal{P}, \mathcal{C}) := \{(\alpha, E) \mid \alpha \in \mathcal{C}, \\ E \subseteq Lit \text{ is non-contradictory}\}.$$

The *type* of a world is now defined as a set of type elements.

**Definition 16 (Type of a World).** Let $\mathcal{P}$ be a program, $\mathcal{C}$ a context for $\mathcal{P}$ and $w$ a world with $w \models \mathcal{D}$. The *type* of $w$ w.r.t. $\mathcal{P}$ and $\mathcal{C}$ is given as follows:

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

To illustrate this definition we give a very simple "minimal" example.

**Example 17.** Consider a single fluent $OnTable(x)$, an action $remove(x)$ and an object constant $b$. The initial theory is given by $\mathcal{D}_0 = \{OnTable(b)\}$, $\mathcal{D}_{\text{post}}$ contains a single SSA

$$\Box[a] OnTable(x) \equiv OnTable(x) \wedge \neg a = remove(x).$$

and in $\mathcal{D}_{\text{pre}}$ we have the axiom

$$\Box Poss(remove(x)) \equiv OnTable(x).$$

As a context for the BAT $\mathcal{D}$ and program $remove(b)$ we choose

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

We consider two worlds $w_0, w_1$ such that

$$w_0, \langle\rangle \models OnTable(b) \text{ and}$$
$$w_0, \langle\rangle \not\models OnTable(b') \text{ for all } b' \in \mathcal{N}_O \text{ with } b \neq b'$$

and in $w_1$ it holds that

$$w_1, \langle\rangle \models OnTable(b) \text{ and}$$
$$w_1, \langle\rangle \models OnTable(b') \text{ for some } b' \in \mathcal{N}_O \text{ with } b \neq b'.$$

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

$$type(w_0) := \{(\alpha_b, L_0), (\alpha_\exists, L_0), (\alpha_b, L^+), (\alpha_\exists, L^+), \\ (\neg\alpha_b, L^-), \underline{(\neg\alpha_\exists, L^-)}\};$$
$$type(w_1) := \{(\alpha_b, L_0), (\alpha_\exists, L_0), (\alpha_b, L^+), (\alpha_\exists, L^+), \\ (\neg\alpha_b, L^-), \underline{(\alpha_\exists, L^-)}\}.$$

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 incomplete 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 $\exists x.OnTable(x)$ in context $\mathcal{C}$ remains true in $w_1$ after removing the object $b$.

The next lemma states some properties of types that are direct consequences of the definition given above and Lemma 15.

**Lemma 18.** *Consider two worlds $w, w'$ and their types w.r.t. $\mathcal{P}$ and $\mathcal{C}$. It holds that:*

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

As a consequence of this lemma we can determine $\mathcal{E}(w, z, t)$ based on $type(w)$. Consider a sequence $z = t_0 t_1 \cdots t_n$ of ground actions in $\delta$. Using the equivalence (5) we can accumulate the set of effects of each prefix of $z$ into a single set of literals. The accumulated set of effect for $z$ in world $w$ is denoted by $E(w, z)$. It clearly holds that

$$w, z \models \alpha \text{ iff } w, \langle\rangle \models \alpha^{R(E(w,z))} \tag{6}$$

for any fluent sentence $\alpha \in \mathcal{C}$. Now, we are ready to define an equivalence relation on the states of the transition system.

**Definition 19.** Consider $\mathcal{P}$, $\mathcal{C}$ and the transition system $T_{\mathcal{P}} = (Q, \rightarrow, I)$. Let $(w, z, \rho), (w', z', \rho')$ be states in $Q$. $(w, z, \rho)$ and $(w', z', \rho')$ are equivalent, written as $(w, z, \rho) \simeq (w', z', \rho')$ iff $type(w) = type(w')$ and $E(w, z) = E(w', z')$ and $\rho = \rho'$.

Next, we show the desired property that two equivalent states simulate each other, i.e. they cannot be distinguished by a temporal property.

**Lemma 20.** *Let $\mathcal{C}$ be a context for the program $\mathcal{P}$ with the corresponding transition system $T_{\mathcal{P}} = (Q, \rightarrow, I)$. Let $s_0, s_1 \in Q$ with $s_0 \simeq s_1$.*

1. *If there exists a state $s_0'$ with $s_0 \xrightarrow{t} s_0'$, then there exists a state $s_1'$ with $s_1 \xrightarrow{t} s_1'$ and $s_0' \simeq s_1'$.*
2. *If there exists a state $s_1'$ with $s_1 \xrightarrow{t} s_1'$, then there exists a state $s_0'$ with $s_0 \xrightarrow{t} s_0'$ and $s_0' \simeq s_1'$.*

Basically, this lemma shows that the relation $\simeq$ on the state space $Q$ gives us a bisimulation w.r.t. the formulas in $\mathcal{C}$. Therefore, the following lemma is a direct consequence of this property.

**Lemma 21.** *Let $\mathcal{C}$ be a context for a program $\mathcal{P}$ with the transition system $T_{\mathcal{P}} = (Q, \rightarrow, I)$ and $\Phi$ a $\mathcal{ES}$-$C^2$-LTL formula that contains only fluent sentences from $\mathcal{C}$. Let $s, s' \in I$ and $s \simeq s'$. There exists a run $r$ starting in $s$ with $\mathfrak{w}(r), 0 \models \Phi$ iff there exists a run $r'$ starting in $s'$ with $\mathfrak{w}(r'), 0 \models \Phi$.*

This lemma shows that it is enough to consider the quotient transition system of $T_{\mathcal{P}}$ w.r.t. $\simeq$.

**Definition 22 (Quotient Transition System).** Let $\mathcal{C}$ be a context for a program $\mathcal{P}$. The *quotient transition system* $T_{\mathcal{P}/\simeq} = (\widehat{Q}, \twoheadrightarrow, \widehat{I})$ is given by $\widehat{Q} := \{[s]_\simeq \mid s \in Q\}$, $\twoheadrightarrow := \{[s]_\simeq \xrightarrow{t} [s']_\simeq \mid s \xrightarrow{t} s'\}$ and $I := \{[s]_\simeq \mid s \in I\}$.

The equivalence class $[w, z, \rho]_\simeq$, i.e. a state in the quotient transition system, can be characterized by the type $type(w)$, the subset $E(w, z)$ of $Lit$ and $\rho \in \mathsf{sub}(\delta)$. There are only finitely many world-types, subsets of $Lit$ and reachable subprograms of $\delta$. Hence, the quotient transition system is finite.

**Constructing the Quotient Transition System** Next, we describe how the quotient transition system can be constructed. Consider a program $\mathcal{P} = (\mathcal{D}, \delta)$ and a context $\mathcal{C}$. First, we guess a set of type elements $\tau \subseteq TE(\mathcal{P}, \mathcal{C})$ such that for all $\alpha \in \mathcal{C}$ and for all non-contradictory $E \subseteq Lit$, it holds that either $(\alpha, E) \in \tau$ or $(\neg\alpha, E) \in \tau$. Using the regression operator we test whether $\tau$ is indeed a type of a world that satisfies the BAT. This is done by checking consistency of the $C^2$ KB, given by $\mathcal{D}_0 \cup \{\alpha^{R(E)} \mid (\alpha, E) \in \tau\}$. If this KB is consistent, then there exists a world $w, \langle\rangle \models \mathcal{D}_0$ with $type(w) = \tau$. We get that $(\tau, \emptyset, \delta)$ represents the initial state $[(w, \langle\rangle, \delta)]_\simeq$ in the quotient transition system $T_{\mathcal{P}/\simeq}$. Moreover, we introduce a function $L$ that labels the states with the set of formulas from the context $\mathcal{C}$ that are satisfied in this state. This function is defined by $L(\tau, E, \rho) := \{\alpha \mid (\alpha, E) \in \tau\}$. To perform a transition to a successor state we determine which ground action $t$ can be executed next and what is the remaining program expression. The labeling of the state gives us sufficient information to check the conditions in the definition of the transition semantics. As shown in Lemma 18 item 1, it is also possible to compute the value of the effect function for a given type $\tau$, the accumulated effects $E$ of the action sequence executed so far and a ground action $t$. To compute the representation of the successor state of $(\tau, E, \rho)$ after executing an action $t$, we update $E$ accordingly and replace $\rho$ by the remaining subprogram after executing $t$.

Details of this construction and the proof that this indeed yields the quotient transition system can be found in the technical report (Zarrieß and Claßen 2013).

Having this finite abstraction of the transition system the verification problem basically boils down to a propositional LTL model checking problem. We simply introduce an atomic proposition for each formula in the context and then replace the formulas in the labeling of the quotient transition system and the formulas in the temporal property by its corresponding atomic proposition.

The complexity of this decision procedure is 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 NEXPTIME (Pacholski, Szwast, and Tendera 2000). Therefore, checking whether $\tau$ is a type can be done in 2-NEXPTIME. It turns out that 2-NEXPTIME is also an upper bound for the overall complexity.

**Theorem 23.** *Satisfiability of an $\mathcal{ES}$-$C^2$-LTL formula in a Golog program over ground actions w.r.t. a local-effect BAT is decidable in* 2-NEXPTIME.

## Undecidability

In this section we argue that the assumptions we made in order to establish the decidability results presented in the previous section are not arbitrary, but actually necessary. More precisely, we employed the following restrictions:

1. Fluent formulas have to be expressed in the base logic $C^2$.
2. Disallow pick operators in GOLOG programs.
3. Successor state axioms are all local-effect.

These restrictions are necessary in the sense that once we drop any one of them, the verification problem becomes undecidable again.

Clearly, dropping restriction (1) immediately leads to undecidability as this would allow us to formulate arbitrary first-order sentences as tests and preconditions. Let us therefore consider a program $\mathcal{P} = (\mathcal{D}, \delta)$ where $\mathcal{D}$ is local-effect, but we allow the pick operator for non-deterministic choice of arguments in the program expression $\delta$.

**Theorem 24.** *The verification problem for* GOLOG *programs over non-ground actions based on a local-effect BAT is undecidable.*

The proof is by reduction from the Halting problem of a Turing machine (TM). The operator $\pi x.\delta$ that allows us to pick objects from an unbounded domain of standard names inside a (possibly infinite) loop makes it possible to represent an unbounded tape of a TM. To create this tape two

predicates are essential. The binary predicate $NextTo(x, y)$ represents the adjacency relation of tape cells, whereas the fluent $Visited(x)$ memorizes tape cells that were visited along a run of the TM. Initially, no tape cell is visited. During the execution of a program that simulates the TM, the pick operators enable us to always ensure the existence of fresh, unused cells should the head be moved onto a previously unvisited position. Encoding the transition relation of the TM as actions is straightforward.

If we disallow the pick operator again, but instead allow for non-local effects, a similar construction is possible where picking "fresh" cells is now formalized inside the successor state axioms through appropriate (unrestricted) quantification. Again note that the details of the constructions used in this section can be found in the accompanying technical report (Zarrieß and Claßen 2013).

**Theorem 25.** *The verification for* GOLOG *programs over ground actions based on unrestricted BATs is undecidable.*

## Conclusion

In this paper we presented results on the verification of temporal properties for GOLOG programs. We have extended the decidability results obtained in (Claßen, Liebenberg, and Lakemeyer 2013) and (Baader and Zarrieß 2013) to a larger fragment of local-effect action theories, to the base logic $C^2$ and to LTL properties over $C^2$-axioms. Furthermore, we showed that requiring local effects and disallowing pick operators are necessary as dropping one of these restrictions leads to undecidability of the verification problem.

There are many possible directions for future work. Our approach could be extended in terms of expressiveness along many dimensions. For example, we will look at branching time temporal properties like CTL$^*$ instead of LTL. Moreover, one could explore classes of successor state axioms that go beyond local-effect theories. Finally, it would be interesting to see whether a limited form of the pick operator could be re-introduced without losing decidability.

## References


Baader, F., and Zarrieß, B. 2013. Verification of Golog programs over description logic actions. In: Proc. of FroCoS'13

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Baader, F.; Lutz, C.; Miličić, M.; Sattler, U.; and Wolter, F. 2005. Integrating description logics and action formalisms: First results. In: *Proc. of AAAI 2005*

Baader, F.; Ghilardi, S.; and Lutz, C. LTL over description logic axioms. In: *Proc. of KR 2008*

Baader, F.; Liu, H.; and ul Mehdi, A. Verifying properties of infinite sequences of description logic actions. In: *Proc. of the ECAI 2010*

Burgard, W.; Cremers, A. B.; Fox, D.; Hähnel, D.; Lakemeyer, G.; Schulz, D.; Steiner, W.; and Thrun, S. 1999. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence* 114(1–2):3–55.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In: *Proc. of KR 2008*

Claßen, J.; Liebenberg, M.; and Lakemeyer, G. On decidable verification of non-terminating Golog programs. In: *Proc. of NRAC 2013*

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2).

De Giacomo, G.; Ternovska, E.; and Reiter, R. 1997. Non-terminating processes in the situation calculus. In *Working Notes of "Robots, Softbots, Immobots: Theories of Action, Planning and Control", AAAI'97 Workshop*.

Gu, Y., and Soutchanski, M. 2010. A description logic based situation calculus. *Annals of Mathematics and Artificial Intelligence* 58(1–2):3–83.

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

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.

Liu, Y., and Lakemeyer, G. 2009. On first-order definability and computability of progression for local-effect actions and beyond. In: *Proc. of IJCAI 2009*

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. New York: American Elsevier. 463–502.

Pacholski, L.; Szwast, W.; and Tendera, L. 2000. Complexity results for first-order two-variable logic with counting.

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

Vassos, S.; Lakemeyer, G.; and Levesque, H. J. First-order strong progression for local-effect basic action theories. In: *Proc. of KR 2008*

Zarrieß, B., and Claßen, J. On the decidability of verifying LTL properties of Golog programs. LTCS-Report 13-10, Chair of Automata Theory, TU Dresden, Dresden, Germany. See http://lat.inf.tu-dresden.de/research/reports.html.