TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

On the Decidability of Verifying LTL Properties of Golog Programs (Extended Version)

Benjamin Zarrieß Jens Claßen

LTCS-Report 13-10

In this extended version we extend the decidability result for the verification problem to the temporal logic CTL* over C^2-axioms.

Postal Address:
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nöthnitzer Str. 46
Dresden

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

March 3, 2014

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.

*Supported by DFG Research Unit FOR 1513, project A1

Contents

1 Introduction

Golog [GLL00, LRL+97], 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 [BCF+99, FL08]. It is defined on top of action theories expressed in the Situation Calculus [MH69, Rei01], a dialect 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 [GTR97] 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 manual, meta-theoretic proofs to show the satisfaction of such properties. Since it is more preferable to do verification in automated fashion, Claßen and Lakemeyer [CL08] later proposed an approach based on a new logic called $\mathcal{ESG}$, an extension of the modal Situation Calculus variant $\mathcal{ES}$ [LL10] 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 [BLM10] who, instead of using fully-fledged Situation Calculus and Golog, resort to an action language [BLM+05] based on the decidable Description Logic (DL) $\mathcal{ALC}$ [BCM+03] 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 [BZ13] 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 [CLL13] 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 the Situation Calculus [GS10] as base logic, allowing only ground action terms in Golog programs, and requiring actions to only have local effects [LL09]. Compared to the above mentioned results by Baader and Zarrieß, local-effect action theories represents 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 Situation Calculus; 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 decidable base logic, namely the modal Situation Calculus variant $\mathcal{ES}$ based on the two-variable fragment of first-order logic with counting quantifiers C^2, as well as GOLOG programs and their semantics. In Section 3 we then define the problem of verifying CTL* properties over such programs, and show (in a constructive manner) that the problem is indeed decidable. We then discuss related work and conclude.

2 Preliminaries

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

In this subsection we recall the main definitions of the modal Situation Calculus variant $\mathcal{ES}$ [LL10]. 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.

2.1.1 Syntax

We start by defining a set of *terms*.

Definition 1 (Terms)**.** In our language we consider terms of two sorts *object* and *action*. They can be built using the following symbols:

- variables $x, y, \cdots$ of sort *object*;
- a single variable a of sort *action*;
- a countably infinite set N_I of *rigid object constant symbols* (i.e. 0-ary function symbols);
- a countably infinite set N_A of *rigid action function symbols* with arguments of sort object;

A term is called *ground term* if it contains no variables. We denote the set of all 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 pre-conditions 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 $\Box$ referring to future situations.

Definition 2 (Formulas)**.** 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 α and β 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 $m \in \mathbb{N}$, $\Box\alpha$ (α always holds) and $[t]\alpha$ (α 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 *fluent formula* if it contains no $\Box$, no $[\cdot]$ and not the predicate *Poss*. A *fluent sentence* is a fluent formula without free variables. ▲

2.1.2 Semantics

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

Definition 3 (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

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

such that 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 predicate symbols do not take different values in different situations, as expected.

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 4 (Satisfaction of Formulas)**.** Given a world $w \in \mathcal{W}$ and a formula α, 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 α^x_n to denote the result of simultaneously replacing all free occurrences of x by n. Note that by rule 2 above, the unique names assumption (UNA) for actions and object constants is part of our semantics.

Basic Action Theories We now define a theory as a set of axioms of a pre-defined structure in order to model a dynamic application domain.

Definition 5 (Basic Action Theory)**.** 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 theory*, 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 relevant to the application domain, there is an axiom of the form $\Box Poss(A(\vec{x})) \equiv \varphi(\vec{x})$, with $\varphi(\vec{x})$ 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 relevant to the application domain, incorporating Reiter's [Rei01] solution to the frame problem, and encoding the effects the actions have on the different fluents. The SSA for a fluent predicate has the form $\Box[a]F(\vec{x}) \equiv \gamma_F^+ \vee F(\vec{x}) \wedge \neg\gamma_F^-$, where γ_F^+ and γ_F^- are fluent formulas with free variables $\vec{x}$ and a.

▲

2.2 Golog Programs

Given a BAT axiomatizing pre-conditions 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:

$$\delta ::= \langle\rangle \mid t \mid \alpha? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \pi x.\delta \mid \delta_1 \| \delta_2 \mid \delta^* \tag{1}$$

That is we allow the empty program $\langle\rangle$, primitive actions t (where t can be any action term), tests $\alpha?$ (where α is a fluent sentence), sequence, nondeterministic branching, nondeterministic choice of argument, concurrency, and nondeterministic iteration. Recall, that thus also **if** statements and **while** loops are included:

$$\textbf{if } \alpha \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \stackrel{def}{=} [\alpha?; \delta_1] \mid [\neg\alpha?; \delta_2] \tag{2}$$

$$\textbf{while } \alpha \textbf{ do } \delta \textbf{ endWhile} \stackrel{def}{=} [\alpha?; \delta]^*; \neg\phi? \tag{3}$$

A Golog program consists of a BAT and a program expression.

Definition 6 (Golog Program)**.** A *Golog program* $\mathcal{P} = (\mathcal{D}, \delta)$ consists of a BAT $\mathcal{D}$ and a program expression δ which only mentions actions and fluents defined in $\mathcal{D}$. The set of action terms occurring in δ is denoted by Act. ▲

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

Definition 7 (Program Transition Semantics)**.** The transition relation $\xrightarrow{w}$ among configurations, given a world w, 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 $\langle z, \delta_n^x\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$ 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 $\langle z, \delta_n^x\rangle \in \mathcal{F}^w$ 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. $\langle z, \delta^*\rangle \in \mathcal{F}^w$;
7. $\langle z, \langle\rangle\rangle \in \mathcal{F}^w$.

▲

Let $\xrightarrow{w}{}^*$ denote the reflexive and transitive closure of $\xrightarrow{w}$. Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a program. 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 follows that $w, z \models Poss(t)$.

To handle non-terminating, terminating and failing runs of a program uniformly, we introduce two fresh 0-ary fluents $Term$ and $Fail$ and two 0-ary action functions ϵ and $\mathfrak{f}$ and include axioms $\Box Poss(\epsilon) \equiv true$ and $\Box Poss(\mathfrak{f}) \equiv true$ in $\mathcal{D}_{\text{pre}}$ as well as axioms $\Box[a]Term \equiv a = \epsilon \vee Term$ and $\Box[a]Fail \equiv a = \mathfrak{f} \vee Fail$ in $\mathcal{D}_{\text{post}}$. Termination and failure are thus represented through non-terminating runs by having "sinks" that continue looping ϵ and $\mathfrak{f}$ indefinitely.

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

Definition 8 (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. There is no $\langle z'', \rho'' \rangle$ s.t. $\langle z, \rho \rangle \xrightarrow{w} \langle z'', \rho'' \rangle$ and $\langle z, \rho \rangle \notin \mathcal{F}^w$, and 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 $T_{\mathcal{P}} = (Q, \rightarrow, I)$ 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 $act(\mathfrak{r}) = t_0 t_1 t_2 \cdots$.

3 Verification

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

$$\Phi ::= \alpha \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathsf{E}\Psi \tag{4}$$

$$\Psi ::= \Phi \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \mathsf{X}\Psi \mid \Psi_1 \,\mathsf{U}\, \Psi_2 \tag{5}$$

Above, α can be any C^2-fluent sentence. We call formulas according to (4) *state formulas*, and formulas according to (5) *path formulas*. We use the usual abbreviations $\mathsf{A}\Psi$ for $\neg\mathsf{E}\neg\Psi$, $\mathsf{F}\Psi$ for $\top \,\mathsf{U}\, \Psi$ and $\mathsf{G}\Psi$ for $\neg\mathsf{F}\neg\Psi$. $\mathcal{ES}$-C^2-CTL* formulas are interpreted w.r.t. an $\mathcal{ES}$-transition system:

Definition 9 ($\mathcal{ES}$-C^2-CTL* Semantics)**.** Let Φ be a $\mathcal{ES}$-C^2-CTL* state formula and $T = (Q, \rightarrow, I)$ an $\mathcal{ES}$-transition system. Truth of Φ in T in a state $s = (w_s, z_s, \delta_s) \in Q$, denoted by $T, s \models \Phi$, is defined as follows:

- $T, s \models \alpha$ iff $w_s, z_s \models \alpha$;
- $T, s \models \neg\Phi$ iff $T, s \not\models \Phi$;
- $T, s \models \Phi_1 \wedge \Phi_2$ iff $T, s \models \Phi_1$ and $T, s \models \Phi_2$;
- $T, s \models \mathsf{E}\Psi$ iff there exists $\pi \in Paths(s)$ such that $T, \pi \models \Psi$.

Let Ψ be a $\mathcal{ES}$-C^2-CTL* path formula and T a transition system as above. Truth of Ψ in T in a path π starting in some state $s_0 \in Q$, denoted by $T, \pi \models \Psi$, is defined as follows:

- $T, \pi \models \Phi$ iff $T, s_0 \models \Phi$;
- $T, \pi \models \neg\Psi$ iff $T, \pi \not\models \Psi$;
- $T, \pi \models \Psi_1 \wedge \Psi_2$ iff $T, \pi \models \Psi_1$ and $T, \pi \models \Psi_2$;
- $T, \pi \models \mathsf{X}\Psi$ iff $T, \pi[1..] \models \Psi$;

- $T, \pi \models \Psi_1 \mathsf{U} \Psi_2$ iff
  $\exists k \geq 0 : T, \pi[k..] \models \Psi_2$ and $\forall j, 0 \leq j < k : T, \pi[j..] \models \Psi_1$.

We write $T \models \Phi$ if $T, s_0 \models \Phi$ for all $s_0 \in I$. ▲

We are now ready to define the *verification problem*:

Definition 10 (Verification Problem)**.** Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program, $T_\mathcal{P} = (Q, \rightarrow, I)$ the corresponding transition system and Φ an $\mathcal{ES}$-C^2-CTL* state formula. The formula Φ is *valid* in $\mathcal{P}$ if $T_\mathcal{P} \models \Phi$. The formula Φ is *satisfiable* in $\mathcal{P}$ if there exists a $s_0 \in I$ such that $T_\mathcal{P}, s_0 \models \Phi$. ▲

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 φ 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 $\mathsf{AG}\varphi$ in the program $\mathcal{P}$. Furthermore, the fluents $Term$ and $Fail$ can be used to express facts about the termination or failing of a program.

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

3.1 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 infinite in general. To achieve decidability we have to make certain restrictions 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, 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 δ is a program expression that can be built according to the following grammar

$$\delta ::= \langle\rangle \mid t \mid \alpha? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \delta_1 \| \delta_2 \mid \delta^* \tag{6}$$

where t is a ground action term and α 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 δ. The size of a program expression $|\delta|$ is defined as the number of ground actions, tests and program constructs occurring in δ.

Lemma 11. *Let δ be a program expression over ground actions. The cardinality of $\mathsf{sub}(\delta)$ is exponentially bounded in the size $|\delta|$ of δ.*

Before we prove this lemma we introduce some additional notation. As in [BZ13], we introduce the notion of a *guarded action*. A guarded action is a ground action preceded by a test. Since in our semantics a test does not cause a transition, we consider a test and an action as a unit.

Definition 12 (Guarded Action)**.** Let δ be a program expression. A guarded action in δ is of the form $\alpha?; t$ where $t \in Act$ or $t = \epsilon$ and α is a fluent sentence of the form $\alpha_1 \wedge \cdots \wedge \alpha_n$ where the α_i for $i = 1, ..., n$ are tests occurring in δ. ▲

We define two functions $\mathsf{head}(\cdot)$ and $\mathsf{tail}(\cdot,\cdot)$. Intuitively, $\mathsf{head}(\delta)$ contains those guarded actions that can be executed first when executing δ and $\mathsf{tail}(\alpha?;t,\delta)$ yields the program expressions that remain to be executed after executing the guarded action $\alpha?;t$ from the head of δ.

Definition 13. The function $\mathsf{head}(\cdot)$ maps a program expression to a set of guarded actions. It is defined by induction on the structure of program expressions:

1. $\mathsf{head}(\langle\rangle) := \{\epsilon\}$;
2. $\mathsf{head}(t) := \{t\}$ for all $t \in Act$;
3. $\mathsf{head}(\alpha?) := \{\alpha?;\epsilon\}$;
4. $\mathsf{head}(\delta^*) := \{\epsilon\} \cup \mathsf{head}(\delta)$;
5. $\mathsf{head}(\delta_1;\delta_2) := \{\mathfrak{t} \mid \mathfrak{t} = \alpha?;t \in \mathsf{head}(\delta_1) \wedge t \neq \epsilon\} \cup \{\alpha_1 \wedge \alpha_2?;t \mid \alpha_1?;\epsilon \in \mathsf{head}(\delta_1) \wedge; \alpha_2?;t \in \mathsf{head}(\delta_2)\}$
6. $\mathsf{head}(\delta_1|\delta_2) := \mathsf{head}(\delta_1) \cup \mathsf{head}(\delta_2)$;
7. $\mathsf{head}(\delta_1\|\delta_2) := \{\mathfrak{t} \mid \mathfrak{t} = \alpha?;t \in \mathsf{head}(\delta_i) \wedge i \in \{1,2\} \wedge t \neq \epsilon\} \cup \{\alpha_1 \wedge \alpha_2?;t \mid \alpha_1?;\epsilon \in \mathsf{head}(\delta_i) \wedge \alpha_2?;t \in \mathsf{head}(\delta_j) \wedge \{i,j\} = \{1,2\}\}$.

▲

Definition 14. The function $\mathsf{tail}(\cdot,\cdot)$ maps a guarded action and a program expression to a set of program expressions.

- If $\mathfrak{t} \notin \mathsf{head}(\delta)$, then $\mathsf{tail}(\mathfrak{t},\delta) = \emptyset$.
- If $\mathfrak{t} \in \mathsf{head}(\delta)$ and $\mathfrak{t} = \alpha?;\epsilon$, then $\mathsf{tail}(\mathfrak{t},\delta) = \{\langle\rangle\}$.
- If $\mathfrak{t} \in \mathsf{head}(\delta)$ and $\mathfrak{t} = \alpha_1 \wedge \ldots \wedge \alpha_2?;t$ for $t \in Act \setminus \{\epsilon\}$, then $\mathsf{tail}(\mathfrak{t},\delta)$ is defined by induction on the combined size of $\mathfrak{t}$ and δ:
  1. $\mathsf{tail}(\mathfrak{t},t') := \{\langle\rangle\}$ for $t \in Act$;[1]
  2. $\mathsf{tail}(\mathfrak{t},\delta^*) := \{\delta';(\delta)^* \mid \delta' \in \mathsf{tail}(\mathfrak{t},\delta)\}$;
  3. $\mathsf{tail}(\mathfrak{t},\delta_1;\delta_2) := \{\delta';\delta_2 \mid \delta' \in \mathsf{tail}(\mathfrak{t},\delta_1)\} \cup \{\delta'' \mid \alpha_1?;\epsilon \in \mathsf{head}(\delta_1) \wedge \alpha_2?;t \in \mathsf{head}(\delta_2) \wedge \delta'' \in \mathsf{tail}(\alpha_2?;t,\delta_2) \wedge \mathfrak{t}$ is of the form $(\alpha_1 \wedge \alpha_2)?;t\}$;
  4. $\mathsf{tail}(\mathfrak{t},\delta_1|\delta_2) := \mathsf{tail}(\mathfrak{t},\delta_1) \cup \mathsf{tail}(\mathfrak{t},\delta_2)$;
  5. $\mathsf{tail}(\mathfrak{t},\delta_1\|\delta_2) := \{\delta'\|\delta_2 \mid \delta' \in \mathsf{tail}(\mathfrak{t},\delta_1)\} \cup \{\delta_1\|\delta' \mid \delta' \in \mathsf{tail}(\mathfrak{t},\delta_2)\} \cup \{\delta'' \mid \alpha_1?;\epsilon \in \mathsf{head}(\delta_i) \wedge \alpha_2?;t \in \mathsf{head}(\delta_j) \wedge \delta'' \in \mathsf{tail}(\alpha_2?;\alpha,\delta_j) \wedge \{i,j\} = \{1,2\} \wedge \mathfrak{t}$ is of the form $(\alpha_1 \wedge \alpha_2)?;t\}$.

▲

In the next lemma we show that the transition semantics defined in Definition 7 using transition rules and the program semantics in [BZ13] that uses the functional-style definition with $\mathsf{head}(\cdot)$ and $\mathsf{tail}(\cdot,\cdot)$ coincide.

[0]Note that $\mathfrak{t} \in \mathsf{head}(t')$ implies $\mathfrak{t} = t'$.

Lemma 15. *Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a program, w a world with $w \models \mathcal{D}$ and $z \in \mathcal{Z}$.*

1. *$\langle z, \delta\rangle \in \mathcal{F}^w$ iff there exists $\alpha?;\epsilon \in \mathsf{head}(\delta)$ and $w, z \models \alpha$.*
2. *$\langle z, \delta\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$ iff there exists $\alpha?;t \in \mathsf{head}(\delta)$, $\delta' \in \mathsf{tail}(\alpha?;t, \delta)$ and $w, z \models \alpha \wedge Poss(t)$.*

Proof. We show the claims by induction on the structure of δ.

1. $\delta = \alpha?$: We have $\langle z, \alpha?\rangle \in \mathcal{F}^w$ iff $w, z \models \alpha$. By definition it holds that $\mathsf{head}(\alpha?) = \{\alpha?;\epsilon\}$.

   $\delta = \delta_1;\delta_2$: We have $\langle z, \delta_1;\delta_2\rangle \in \mathcal{F}^w$ implies $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \in \mathcal{F}^w$. By the induction hypothesis there are $\alpha_1;\epsilon \in \mathsf{head}(\delta_1)$, $\alpha_2;\epsilon \in \mathsf{head}(\delta_2)$ and $w, z \models \alpha_1$ and $w, z \models \alpha_2$. By definition of $\mathsf{head}(\delta_1;\delta_2)$ it holds that $(\alpha_1 \wedge \alpha_2)?;\epsilon \in \mathsf{head}(\delta_1;\delta_2)$ and it holds that $w, z \models \alpha_1 \wedge \alpha_2$.

   Since $\alpha?;\epsilon \in \mathsf{head}(\delta_1;\delta_2)$ there are $\alpha_1?;\epsilon \in \mathsf{head}(\delta_1)$and $\alpha_2?;\epsilon \in \mathsf{head}(\delta_2)$ s.t. $\alpha = \alpha_1 \wedge \alpha_2$. By the induction hypothesis and the assumption $w, z \models \alpha$ it follows that $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \in \mathcal{F}^w$. Therefore, $\langle z, \delta_1;\delta_2\rangle \in \mathcal{F}^w$.

   $\delta = \delta_1|\delta_2$: $\langle z, \delta_1|\delta_2\rangle \in \mathcal{F}^w$ implies $\langle z, \delta_1\rangle \in \mathcal{F}^w$ or $\langle z, \delta_2\rangle \in \mathcal{F}^w$. By induction it follows that there exists $\alpha_1?;\epsilon \in \mathsf{head}(\delta_1)$ with $w, z \models \alpha_1$ or there exists $\alpha_2?;\epsilon \in \mathsf{head}(\delta_2)$ with $w, z \models \alpha_2$. Assume w.l.o.g. the latter. Clearly, $\alpha_2?;\epsilon \in \mathsf{head}(\delta_1|\delta_2)$.

   $\alpha?;\epsilon \in \mathsf{head}(\delta_1|\delta_2)$ and $w, z \models \alpha$ implies $\alpha?;\epsilon \in \mathsf{head}(\delta_1)$ or $\alpha?;\epsilon \in \mathsf{head}(\delta_2)$. Assume w.l.o.g. the latter. By induction it follows $\langle z, \delta_2\rangle \in \mathcal{F}^w$ and therefore $\langle z, \delta_1|\delta_2\rangle \in \mathcal{F}^w$.

   $\delta = \delta_1\|\delta_2$: This case can be shown in a similar way as the case $\delta = \delta_1;\delta_2$.

   The cases for $\delta = \delta'^*$ and $\delta = \langle\rangle$ are trivial.

2. $\delta = t$: We have $\langle z, t\rangle \xrightarrow{w} \langle z \cdot t, \langle\rangle\rangle$. Clearly, it holds that $t \in \mathsf{head}(t)$ and $\langle\rangle \in \mathsf{tail}(t, t)$ and $w, z \models Poss(t)$.

   We have $\mathsf{head}(t') = \{t'\}$, $\mathsf{tail}(t', t') = \{\langle\rangle\}$ and $w, z \models Poss(t')$. It holds that $\langle z, t'\rangle \xrightarrow{w} \langle z \cdot t', \langle\rangle\rangle$.

   $\delta = \delta_1;\delta_2$: First, assume that $\langle z, \delta_1;\delta_2\rangle \xrightarrow{w} \langle z \cdot t, \gamma;\delta_2\rangle$ and $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot t, \gamma\rangle$. By the induction hypothesis it holds that there exists $\alpha?;t \in \mathsf{head}(\delta_1)$, $\gamma \in \mathsf{tail}(\alpha?;t, \delta_1)$ and $w, z \models \alpha \wedge Poss(t)$. By the definition of head and tail we get $\alpha?;t \in \mathsf{head}(\delta_1;\delta_2)$ and $\gamma;\delta_2 \in \mathsf{tail}(\alpha?;t, \delta_1;\delta_2)$. Now, assume $\langle z, \delta_1;\delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$, $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$. By the first claim of this lemma it holds that there exists $\alpha_1?;\epsilon \in \mathsf{head}(\delta_1)$ and $w, z \models \alpha_1$. By the induction hypothesis it follows that there exists $\alpha_2?;t \in \mathsf{head}(\delta_2)$, $\delta' \in \mathsf{tail}(\alpha_2?;t, \delta_2)$ and $w, z \models \alpha_2 \wedge Poss(t)$. By the definition of head and tail we get $(\alpha_1 \wedge \alpha_2)?;t \in \mathsf{head}(\delta_1;\delta_2)$ and $\delta' \in \mathsf{tail}(\alpha_1 \wedge \alpha_2?;t, \delta_1;\delta_2)$. Clearly, we have also $w, z \models \alpha_1 \wedge \alpha_2$.

   Assume $\alpha?;t \in \mathsf{head}(\delta_1;\delta_2)$, $\delta' \in \mathsf{tail}(\alpha?;t, \delta_1;\delta_2)$ and $w, z \models \alpha \wedge Poss(t)$. First, assume $t \neq \epsilon$ and $\alpha?;t \in \mathsf{head}(\delta_1)$. It follows that there exists $\gamma \in \mathsf{tail}(\alpha?;t, \delta_1)$. By induction it follows that $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot t, \gamma\rangle$. Therefore, it holds that $\delta' = \gamma;\delta_2 \in \mathsf{tail}(\alpha?;t, \delta_1;\delta_2)$ and $\langle z, \delta_1;\delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$.

   Next, assume $\alpha = \alpha_1 \wedge \alpha_2$ and $\alpha_1;\epsilon \in \mathsf{head}(\delta_1)$ and $\alpha_2;t \in \mathsf{head}(\delta_2)$. It holds that $\delta' \in \mathsf{tail}(\alpha_2;t, \delta_2)$. By induction it follows that $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$. By claim 1 it also holds that $\langle z, \delta_1\rangle \in \mathcal{F}^w$ and therefore $\langle z, \delta_1;\delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$.

   The remaining cases for $\delta = \delta_1|\delta_2$, $\delta = \delta_1\|\delta_2$ and $\delta = (\delta')^*$ can be shown in a similar way. □

It follows that we can obtain all reachable subprograms by a sequence of head and tail applications. The following lemma is a direct consequence of the lemma above and the definition of $\mathsf{sub}(\delta)$:

Lemma 16. *Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a program. It holds that $\rho \in \mathsf{sub}(\delta)$ iff there exists a sequence $\delta_0, ..., \delta_n$ such that*

- $\delta_0 = \delta$, $\delta_n = \rho$;
- *there exists a world $w, \langle\rangle \models \mathcal{D}_0$, $\alpha_i?; t_i \in \mathsf{head}(\delta_i)$, $\delta_{i+1} \in \mathsf{tail}(\alpha_i?; t_i, \delta_i)$ for $i = 0, ..., n$ such that $w, \langle\rangle \models \alpha_0 \wedge Poss(t_0)$ and $w, t_0 \cdots t_{j-1} \models \alpha_j \wedge Poss(t_j)$ for $j = 1, ..., n$.*

Having this characterization of the set of reachable subprograms, Lemma 11 is a direct consequence of Lemma 10 in [BZ13].

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 17 (Local-effect SSAs)**.** A successor state axiom is *local-effect* if both γ_F^+ and γ_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 $\vec{x}$, $\vec{z}$ is the remaining variables of $\vec{y}$, and $\phi(\vec{y})$ is a fluent formula with free variables $\vec{y}$ and at most two non-free variables that are all of sort object. The formulas $\phi(\vec{y})$ and ϕ 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 [VLL08, LL09]. Moreover, the DL-based action descriptions introduced in [BLM$^+$05] 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 [BZ13] to test whether a $\mathcal{ES}$-C^2-CTL* formula Φ is satisfiable in a program $\mathcal{P} = (\mathcal{D}, \delta)$ where $\mathcal{D}$ is a local-effect BAT and δ 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 base logic C^2.

Given this finite abstraction we can then apply standard model checking techniques to verify the $\mathcal{ES}$-C^2-CTL* formula.

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

3.1.1 Regression with Ground Actions

We use an idea from [LL05, LL09] to simplify the SSAs in presence of ground actions only.

Since we have only finitely many ground actions in our program it is enough to consider only the so called *ground instantiations* of the SSAs where the action variable a in the SSA is replaced with a ground action term. Because we have made the unique names assumption for constants and action functions, such a ground instantiated SSA can be further simplified as shown in the next lemma.

Lemma 18. *Let $\mathcal{D}$ be a BAT, $\Box[a]F(\vec{x}) \equiv \gamma_F^+ \vee F(\vec{x}) \wedge \neg\gamma_F^- \in \mathcal{D}_{post}$ the SSA for the fluent $F(\vec{x})$ and $t = A(\vec{c})$ a ground action term. 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 γ_{Ft}^{+a} and γ_{Ft}^{-a} are equivalent to a disjunctions of the form*

$$\vec{x} = \vec{c_1} \wedge \phi_1 \vee \cdots \vee \vec{x} = \vec{c_n} \wedge \phi_n, \tag{7}$$

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

Proof. This lemma is a direct consequence from the one shown in [LL05]. □

From now on we assume that in the ground instantiated SSAs the formulas γ_{Ft}^{+a} and γ_{Ft}^{-a} are of the form 7. 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 γ_{Ft}^{+a} and γ_{Ft}^{-a}, respectively.

As we will see in the following, the restriction to ground actions *and* local-effect BAT makes it 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 19 (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:

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

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

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

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

We illustrate this definition with an example.

Example 20. As an example we consider a simple BAT modeling a blocks word. We have two fluents $On(x, y)$ saying block x is on top of block y and $Clear(x)$ means that there is no block on top of block x. We consider a single action $move(x, y, z)$ moving a block x on block y to block z. There is the following pre-condition axiom in $\mathcal{D}_{\text{pre}}$:

$$\Box Poss(move(x, y, z)) \equiv On(x, y) \wedge Clear(x) \wedge Clear(z).$$

We have the following SSAs for $On(x, y)$ and $Clear(x)$:

$$\begin{aligned}\Box[a]On(x, y) &\equiv [\exists z.a = move(x, z, y)] \vee On(x, y) \wedge \neg[\exists z.a = move(x, y, z)], \\ \Box[a]Clear(x) &\equiv [\exists y, z.a = move(y, x, z)] \vee Clear(x) \wedge \neg[\exists y, z.a = move(y, z, x)].\end{aligned}$$

Consider the initial theory

$$\mathcal{D}_0 = \{On(c_1, c_2), \neg Clear(c_2), Clear(c_1), Clear(c_3)\}$$

and the ground action $move(c_1, c_2, c_3)$. The effect conditions of the ground instantiated SSAs with $move(c_1, c_2, c_3)$ look as follows:

$$\begin{aligned}\gamma_{On}^+(\vec{x}, a) = \exists z.a = move(x, z, y) &\rightsquigarrow \gamma_{On}^+(\vec{x}, move(c_1, c_2, c_3)) \equiv x = c_1 \wedge y = c_3, \\ \gamma_{On}^-(\vec{x}, a) = \exists z.a = move(x, y, z) &\rightsquigarrow \gamma_{On}^-(\vec{x}, move(c_1, c_2, c_3)) \equiv x = c_1 \wedge y = c_2, \\ \gamma_{Clear}^+(x, a) = \exists y, z.a = move(y, x, z) &\rightsquigarrow \gamma_{Clear}^+(x, move(c_1, c_2, c_3)) \equiv x = c_2, \\ \gamma_{Clear}^-(x, a) = \exists y, z.a = move(y, z, x) &\rightsquigarrow \gamma_{Clear}^-(x, move(c_1, c_2, c_3)) \equiv x = c_3.\end{aligned}$$

Now, let w be a world such that $w, \langle\rangle \models \mathcal{D}_0$. As expected, the effect function gives us the following set of literals:

$$\mathcal{E}(w, \langle\rangle, move(c_1, c_2, c_3)) = \{On(c_1, c_3), \neg On(c_1, c_2), Clear(c_2), \neg Clear(c_3)\}.$$

▲

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 21 (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 α 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 will hold after executing an action, it is sufficient to test whether the regressed formula is satisfied in the current situation.

Lemma 22. *Let $\mathcal{D}$ be a BAT, $w \in \mathcal{W}$ with $w \models \mathcal{D}$, α a fluent sentence and $t = A(\vec{c})$ a ground action term. For all $z \in \mathcal{Z}$ it holds that*

$$w, z \models \alpha^{R(E)} \text{ iff } w, z \cdot t \models \alpha$$

with $E = \mathcal{E}(w, z, t)$.

Proof. We show this claim by induction on the size of the fluent sentence α. As base case we assume that α is a primitive formula. Let $F(\vec{n}) = F(n_1, ..., n_k)$ with $k \in \{0, 1, 2\}$ be a primitive formula and $E = \mathcal{E}(w, z, t)$ is non-contradictory. We get

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

We show that $w, z \models F(\vec{n})^{R(E)}$ iff $w, z \cdot t \models F(\vec{n})$. Since we have made the UNA (see Def. 4 rule 2), it holds that $\vec{n} = \vec{c}$ iff $\vec{n}$ and $\vec{c}$ are identical vectors of standard names. In addition we have that E is non-contradictory. Therefore, the following equivalence holds:

$$\left(F(\vec{n}) \vee \bigvee_{F(\vec{c}) \in E} (\vec{n} = \vec{c})\right) \wedge \bigwedge_{\neg F(\vec{c}) \in E} (\vec{n} \neq \vec{c}) \equiv$$
$$\bigvee_{F(\vec{c}) \in E} (\vec{n} = \vec{c}) \vee F(\vec{n}) \wedge \bigwedge_{\neg F(\vec{c}) \in E} (\vec{n} \neq \vec{c})$$

Considering the SSA for F, we get the following equivalence by Lemma 18:

$$[t]F(\vec{n}) \equiv \gamma^{+\,a\,\vec{x}}_{F\,t\,\vec{n}} \vee F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}}.$$

It holds that $w,z \models \gamma^{+\,a\,\vec{x}}_{F\,t\,\vec{n}} \vee F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}}$ iff $w,z\cdot t \models F(\vec{n})$. It follows from the definition of $\mathcal{E}(w,z,t)$ that

$$w,z \models \gamma^{+\,a\,\vec{x}}_{F\,t\,\vec{n}} \text{ iff } w,z \models \bigvee_{F(\vec{c})\in\mathcal{E}(w,z,t)} (\vec{n}=\vec{c}).$$

We show:

$$w,z \models F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}} \text{ iff } w,z \models F(\vec{n}) \wedge \bigwedge_{\neg F(\vec{c})\in\mathcal{E}(w,z,t)} (\vec{n}\neq\vec{c}).$$

We have

$$\begin{aligned} F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}} &\equiv F(\vec{n}) \wedge \neg(\vec{n}=\vec{c_1} \wedge \phi_1 \vee \cdots \vee \vec{n}=\vec{c_m} \wedge \phi_m) \\ &\equiv F(\vec{n}) \wedge (\vec{n}\neq\vec{c_1} \vee \neg\phi_1) \wedge \cdots \wedge (\vec{n}\neq\vec{c_m} \vee \neg\phi_m). \end{aligned}$$

for some number m. First we show that assuming $w,z \models F(\vec{n}) \wedge \bigwedge_{\neg F(\vec{c})\in\mathcal{E}(w,z,t)} (\vec{n}\neq\vec{c})$ and $w,z \not\models F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}}$ leads to a contradiction. Since we have $w,z \models F(\vec{n})$ there exists an $i \in \{1,\cdots,m\}$ such hat $w,z \not\models (\vec{n}\neq\vec{c_i} \vee \neg\phi_i)$. It follows that $w,z \models (\vec{n}=\vec{c_i} \wedge \phi_i)$. This implies that $\vec{n}$ and $\vec{c_i}$ are identical and therefore it is also implied that $w,z \models F(\vec{c_i}) \wedge \phi_i$ holds. By definition of $\mathcal{E}(w,z,t)$ it follows that $\neg F(\vec{c_i}) \in \mathcal{E}(w,z,t)$ and therefore $\vec{n}\neq\vec{c_i}$ is a conjunct in $\bigwedge_{\neg F(\vec{c})\in\mathcal{E}(w,z,t)} (\vec{n}\neq\vec{c})$ which is a contradiction to our assumption.

To show the other direction assume to the contrary

$$w,z \models F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}} \text{ and } w,z \not\models F(\vec{n}) \wedge \bigwedge_{\neg F(\vec{c})\in\mathcal{E}(w,z,t)} (\vec{n}\neq\vec{c}).$$

Since $w,z \models F(\vec{n})$ there exists a vector $\vec{c}$ such that $\neg F(\vec{c}) \in \mathcal{E}(w,z,t)$ and $\vec{n}$ and $\vec{c}$ are identical. By definition of $\mathcal{E}(w,z,t)$ there exists an $i \in \{1,\cdots,m\}$ and a disjunct $\vec{n}=\vec{c_i} \wedge \phi_i$ in $\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}}$ such that $\vec{c}=\vec{c_i}$ and $w,z \models F(\vec{c_i}) \wedge \phi_i$. Therefore we have $w,z \not\models \vec{n}\neq\vec{c_i} \vee \neg\phi_i$ which is a contradiction to the assumption $w,z \models F(\vec{n}) \wedge \neg\gamma^{-\,a\,\vec{x}}_{F\,t\,\vec{n}}$. This finishes the proof of the base case.

Let α be of the form $\alpha_1 \wedge \alpha_2$ or $\neg\alpha_0$. Clearly, $\alpha_0, \alpha_1, \alpha_2$ are also fluent sentences and we can apply the induction hypothesis to them. If α is of the form $\forall x.\alpha'$, $\exists^{\leq m} x.\alpha'$ or $\exists^{\geq m} x.\alpha'$, then the induction hypothesis can be applied to all fluent sentences of the form α'^{x}_{n} for some $n \in \mathcal{N}_O$. Therefore the claim easily follows also for complex α. □

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).

Lemma 23. *Let α be a fluent sentence and E,E' non-contradictory subsets of Lit. It holds that $\left[\alpha^{R(E')}\right]^{R(E)} \equiv \alpha^{R(E\setminus\neg E'\cup E')}$.*

Proof. It is sufficient to show the claim for $\alpha = F(\vec{v})$ where F is a fluent and $\vec{v}$ a vector of terms of sort object. Let $L \subseteq Lit$ be non-contradictory. We define the abbreviations

$$L_+(\vec{v}) := \bigvee_{F(\vec{c})\in L} \vec{v}=\vec{c} \text{ and } L_-(\vec{v}) := \bigwedge_{\neg F(\vec{c})\in L} \vec{v}\neq\vec{c}.$$

Since L is non-contradictory and we have made the unique name assumption it holds that $L_+(\vec{v}) \wedge L_-(\vec{v}) \equiv L_+(\vec{v})$. We have

$$\alpha^{R(E')} = (F(\vec{v}) \vee E'_+(\vec{v})) \wedge E'_-(\vec{v})$$

and

$$
\begin{aligned}
&\left[\alpha^{R(E')}\right]^{R(E)} \\
&= \Big(\big([F(\vec{v}) \vee E_+(\vec{v})] \wedge E_-(\vec{v})\big) \vee E'_+(\vec{v})\Big) \wedge E'_-(\vec{v}) \\
&\equiv \Big([F(\vec{v}) \wedge E_-(\vec{v})] \vee E_+(\vec{v}) \vee E'_+(\vec{v})\Big) \wedge E'_-(\vec{v}) \\
&\equiv [F(\vec{v}) \wedge E_-(\vec{v}) \wedge E'_-(\vec{v})] \vee \\
&\quad [E_+(\vec{v}) \wedge E'_-(\vec{v})] \vee \\
&\quad [E'_+(\vec{v}) \wedge E'_-(\vec{v})] \\
&\equiv [F(\vec{v}) \wedge E_-(\vec{v}) \wedge E'_-(\vec{v})] \vee \\
&\quad [E_+(\vec{v}) \wedge E'_-(\vec{v})] \vee \\
&\quad E'_+(\vec{v})
\end{aligned}
$$

Let $D = E \setminus \neg E' \cup E'$. It is easy to see that it holds that

$$[E_+(\vec{v}) \wedge E'_-(\vec{v})] \vee E'_+(\vec{v}) \equiv D_+(\vec{v}).$$

and

$$[F(\vec{v}) \wedge E_-(\vec{v}) \wedge E'_-(\vec{v})] \vee D_+(\vec{v}) \equiv [F(\vec{v}) \wedge D_-(\vec{v})] \vee D_+(\vec{v}).$$

Since D is non-contradictory it holds that

$$[F(\vec{v}) \wedge D_-(\vec{v})] \vee D_+(\vec{v}) \equiv [F(\vec{v}) \vee D_+(\vec{v})] \wedge D_-(\vec{v})$$

and we finally get

$$\left[F(\vec{v})^{R(E')}\right]^{R(E)} \equiv [F(\vec{v})]^{R(E \setminus \neg E' \cup E')}.$$

□

3.1.2 Types of worlds

Next, we identify a finite set of relevant fluent sentences occurring in the program and the action theory.

Definition 24 (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 α be a fluent sentence. If

- α is a test in δ;
- or $\alpha = \varphi^x_{\vec{c}}$ and there is a ground action $A(\vec{c})$ in δ with $\Box Poss(A(\vec{x})) \equiv \varphi \in \mathcal{D}_{\text{pre}}$;
- or $\alpha = \phi$ and there exists a ground action t in δ and a vector of constants $\vec{c}$ such that $(\vec{c}, \phi) \in \gamma^{+a}_{F\,t}$ or $(\vec{c}, \phi) \in \gamma^{-a}_{F\,t}$;
- or $\alpha = F(\vec{c})$ and there exists a ground action t in δ such that $(\vec{c}, \phi) \in \gamma^{-a}_{F\,t}$ for some ϕ,

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

For the states in the transition system $T_{\mathcal{P}} = (Q, \to, I)$ and a context $\mathcal{C}$ for $\mathcal{P}$ we introduce a *context labeling* $L_{\mathcal{C}} : Q \to 2^{\mathcal{C}}$ that is defined by $L_{\mathcal{C}}(w, z, \rho) := \{\alpha \in \mathcal{C} \mid w, z \models \alpha\}$ for all $(w, z, \rho) \in Q$. We also apply $L_{\mathcal{C}}$ only to $w, z \in \mathcal{W} \times \mathcal{Z}$. The context label $L_{\mathcal{C}}(w, z)$ is defined accordingly.

Next, we show some properties of $\mathcal{C}$. Intuitively, the effects of applying a ground action in a situation w, z depend only on the context label. Furthermore, it depends only on the formulas in $\mathcal{C}$ whether a transition in the transition system $T_{\mathcal{P}}$ can be taken or not.

Lemma 25. *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 $L_{\mathcal{C}}(w_0, z_0) = L_{\mathcal{C}}(w_1, z_1)$.*

1. *Let t be a ground action that occurs in δ. It holds that $\mathcal{E}(w_0, z_0, t) = \mathcal{E}(w_1, z_1, t)$.*
2. *Let $\rho \in \mathsf{sub}(\delta)$. It holds that $(w_0, z_0, \rho) \xrightarrow{t} (w_0, z_0 \cdot t, \rho')$ iff $(w_1, z_1, \rho) \xrightarrow{t} (w_1, z_1 \cdot t, \rho')$*

Proof.

1. Let $t = A(\vec{c})$. The effect function is defined for $\mathcal{E}(w_0, z_0, t)$ iff $w_0, z_0 \models Poss(A(\vec{c}))$. According to the equivalence axiom in $\mathcal{D}_{\text{pre}}$ we have that $Poss(A(\vec{c}))$ is equivalent to a fluent sentence φ contained in $\mathcal{C}$. Therefore it is implied that $w_0, z_0 \models Poss(A(\vec{c}))$ iff $w_1, z_1 \models Poss(A(\vec{c}))$.

   It holds that $F(\vec{c'}) \in \mathcal{E}(w_0, z_0, t)$ iff $(\vec{c'}, \phi) \in \gamma^{+a}_{Ft}$ and $w_0, z_0 \models \phi$ iff $w_1, z_1 \models \phi$ (by assumption and $\phi \in \mathcal{C}$) iff $F(\vec{c'}) \in \mathcal{E}(w_1, z_1, t)$. It holds that $\neg F(\vec{c'}) \in \mathcal{E}(w_0, z_0, t)$ iff $(\vec{c'}, \phi) \in \gamma^{+a}_{Ft}$ and $w_0, z_0 \models F(\vec{c'}) \wedge \phi$ iff $w_1, z_1 \models F(\vec{c'}) \wedge \phi$ (by assumption and $\phi, F(\vec{c'}) \in \mathcal{C}$) iff $\neg F(\vec{c'}) \in \mathcal{E}(w_1, z_1, t)$.
2. For any $\alpha_1 \wedge \cdots \wedge \alpha_n?; t \in \mathsf{head}(\rho)$, there exists a formula $\varphi \equiv Poss(t)$ and $\varphi \in \mathcal{C}$. It is also implied that $\{\alpha_1, ..., \alpha_n\} \subseteq \mathcal{C}$ since $\rho \in \mathsf{sub}(\delta)$ and $\alpha_1, \ldots, \alpha_n$ are tests in δ. By assumption it follows that $w_0, z_0 \models \alpha_1 \wedge \cdots \wedge \alpha_n \wedge Poss(t)$ iff $w_1, z_1 \models \alpha_1 \wedge \cdots \wedge \alpha_n \wedge Poss(t)$ and therefore $(w_0, z_0, \rho) \xrightarrow{t} (w_0, z_0 \cdot t, \rho')$ iff $(w_1, z_1, \rho) \xrightarrow{t} (w_1, z_1 \cdot t, \rho')$.

□

Clearly, the context label $L_{\mathcal{C}}(w, z)$ is a maximal consistent subset of $\mathcal{C}$. We define an abstraction of the effect function that takes a maximal consistent subset of $\mathcal{C}$ and a ground action as input. Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a program and $\mathcal{C}$ a context for $\mathcal{P}$. Let $2^{\mathcal{C}}$ denote the set of all maximal consistent subsets of $\mathcal{C}$ and 2^{Lit} the set of all non-contradictory subsets of Lit. The *abstract effect function* $\widehat{\mathcal{E}} : 2^{\mathcal{C}} \times Act \to 2^{Lit}$ is defined as follows: Let $(M, t) \in 2^{\mathcal{C}} \times Act$. If $Poss(t) \in M$ then

$$\begin{aligned}\widehat{\mathcal{E}}(M, t) :=& \{F(\vec{c}) \in Lit \mid \exists (\vec{c}, \phi) \in \gamma^{+a}_{Ft} \wedge \phi \in M\} \cup \\ & \{\neg F(\vec{c}) \in Lit \mid \exists (\vec{c}, \phi) \in \gamma^{-a}_{Ft} \wedge \{F(\vec{c}), \phi\} \subseteq M\}.\end{aligned}$$

and otherwise, if $Poss(t) \notin M$, then $\widehat{\mathcal{E}}(M, t)$ is undefined. It follows that $\widehat{\mathcal{E}}(L_{\mathcal{C}}(w, z), t) = \mathcal{E}(w, z, t)$.

Note that even if $L_{\mathcal{C}}(w, \langle\rangle) = L_{\mathcal{C}}(w', \langle\rangle)$ holds for two worlds w, w' it is not implied that also $L_{\mathcal{C}}(w, t) = L_{\mathcal{C}}(w', t)$ holds for a ground action t. Therefore, the context labels do not provide sufficient information to partition the set of worlds into equivalence classes which we will call *types* in the following. Intuitively, if two worlds are of the same type, we want them to satisfy the same temporal properties if we execute the program in these worlds. In the following we use the notation 2^{Lit} to denote the set of all non-contradictory subsets of Lit. First, we define a set of *type elements* for a program $\mathcal{P}$ and a context $\mathcal{C}$ for $\mathcal{P}$:

$$TE(\mathcal{P}, \mathcal{C}) := \{(\alpha, E) \mid \alpha \in \mathcal{C}, E \in 2^{Lit}\}.$$

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

Definition 26 (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 an example.

Example 27. 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

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

and in w_1 it holds that

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

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 α_b and $\exists x.OnTable(x)$ by $\alpha_\exists$. The different types of w_0 and w_1 are given by:

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

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. ▲

In the next lemma we show some properties of types.

Lemma 28. *Consider two worlds w, w' and their types w.r.t. $\mathcal{P}$ and context $\mathcal{C}$ for $\mathcal{P}$.*

1. *It holds that $(\alpha, E) \in type(w)$ iff $(\neg\alpha, E) \notin type(w)$ for all $(\alpha, E) \in TE(\mathcal{P}, \mathcal{C})$.*
2. *Let $z \in \mathcal{N}_A^*$ be a sequence of ground actions that occur in δ. If $type(w) = type(w')$, then $L_{\mathcal{C}}(w, z) = L_{\mathcal{C}}(w', z)$.*

Proof.

1. Let $(\alpha, E) \in TE(\mathcal{P}, \mathcal{C})$. Since $\mathcal{C}$ is closed under negation we have also $(\neg\alpha, E) \in TE(\mathcal{P}, \mathcal{C})$. It holds that:
$$
\begin{aligned}
&(\neg\alpha, E) \notin \mathit{type}(w) \text{ iff}\\
&w, \langle\rangle \not\models (\neg\alpha)^{R(E)} \text{ iff}\\
&w, \langle\rangle \not\models \neg(\alpha^{R(E)}) \text{ iff}\\
&w, \langle\rangle \models \alpha^{R(E)} \text{ iff}\\
&(\alpha, E) \in \mathit{type}(w).
\end{aligned}
$$

2. We show the claim by induction on the length n of z. Let $n = 0$. It has to be shown that $L_{\mathcal{C}}(w, \langle\rangle) = L_{\mathcal{C}}(w', \langle\rangle)$. It holds that $\alpha \in L_{\mathcal{C}}(w, \langle\rangle)$ iff $w, \langle\rangle \models \alpha$ iff $(\alpha, \emptyset) \in \mathit{type}(w)$ iff $(\alpha, \langle\rangle) \in \mathit{type}(w')$ iff $\alpha \in L_{\mathcal{C}}(w', \langle\rangle)$.

   Now, let $z = t_0 t_1 \cdots t_n t_{n+1}$ and we assume by induction that $L_{\mathcal{C}}(w, z[0..i]) = L_{\mathcal{C}}(w', z[0..i])$ for all $i = 0, \ldots, n$.

   Let $E_0^w = \mathcal{E}(w, \langle\rangle, t_0)$ and $E_{j+1}^w = E_j^w \setminus \neg\mathcal{E}(w, z[0..j], t_{j+1}) \cup \mathcal{E}(w, z[0..j], t_{j+1})$. for $j = 0, ..., n$. By the induction hypothesis and Lemma 25.1 it holds that $E_k^w = E_k^{w'}$ for all $k = 0, ..., n+1$. We have $\alpha \in L_{\mathcal{C}}(w, z)$ iff $w, z \models \alpha$ iff $w, \langle\rangle \models [\alpha]^{R(E_{n+1}^w)}$ (by Lemma 22 and 23) iff $(\alpha, E_{n+1}^w) \in \mathit{type}(w)$ iff $(\alpha, E_{n+1}^w) \in \mathit{type}(w')$ iff $w', \langle\rangle \models [\alpha]^{R(E_{n+1}^w)}$ iff $w', z \models \alpha$ iff $\alpha \in L_{\mathcal{C}}(w', z)$.

□

As a consequence of this lemma we can determine $\mathcal{E}(w, z, t)$ based on $\mathit{type}(w)$. Consider a sequence $z = t_0 t_1 \cdots t_n$ of ground actions in δ. Since Lemma 23 holds we can accumulate the set of effects of each prefix of z into a single set of literals as follows:

$$
\begin{aligned}
E_0 &= \widehat{\mathcal{E}}(L_{\mathcal{C}}(w, \langle\rangle), t_0)\\
E_{i+1} &= E_i \setminus \neg\widehat{\mathcal{E}}(L_{\mathcal{C}}(w, z[0..i]), t_{i+1}) \cup \widehat{\mathcal{E}}(L_{\mathcal{C}}(w, z[0..i]), t_{i+1}).
\end{aligned}
\tag{8}
$$

for $i = 0, ..., n-1$. We define $E(w, z) := E_n$. We have that E_0 depends only on the type elements of the form $(\phi, \emptyset)$ in $\mathit{type}(w)$ and the set E_{i+1} depends only on the type elements of the form (ϕ, E_i) in $\mathit{type}(w)$. Also the context labels of all future situations in a world w are encoded in $\mathit{type}(w)$ and can be obtained as follows:

$$
\begin{aligned}
L_{\mathcal{C}}(w, \langle\rangle) &= \{\alpha \mid (\alpha, \emptyset) \in \mathit{type}(w)\}\\
L_{\mathcal{C}}(w, t_0) &= \{\alpha \mid (\alpha, E_0) \in \mathit{type}(w)\}\\
L_{\mathcal{C}}(w, t_0 t_1) &= \{\alpha \mid (\alpha, E_1) \in \mathit{type}(w)\}\\
&\vdots
\end{aligned}
$$

Now, we are ready to define an equivalence relation on the states of the transition system.

Definition 29. Consider $\mathcal{P}, \mathcal{C}$ and the transition system $T_{\mathcal{P}} = (Q, \to, I)$. Let $(w, z, \rho), (w', z', \rho')$ be states in Q. (w, z, ρ) and (w', z', ρ') are *equivalent*, written as $(w, z, \rho) \simeq (w', z', \rho')$ iff $\mathit{type}(w) = \mathit{type}(w')$ 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 are *bisimilar* w.r.t. the formulas in $\mathcal{C}$.

Lemma 30. *Let $\mathcal{C}$ be a context for the program $\mathcal{P}$ with the corresponding transition system $T_{\mathcal{P}} = (Q, \to, I)$. Let $s_0, s_1 \in Q$ with $s_0 \simeq s_1$.*

1. $L_{\mathcal{C}}(s_0) = L_{\mathcal{C}}(s_1)$

2. (a) *If there exists a state* s_0' *with* $s_0 \xrightarrow{t} s_0'$ *and* $z_{s_0'} = z_{s_0} \cdot t$ *for some* $t \in Act$*, then there exists a state* s_1' *with* $s_1 \xrightarrow{t} s_1'$*,* $z_{s_1'} = z_{s_1} \cdot t$ *and* $s_0' \simeq s_1'$*.*

   (b) *If there exists a state* s_1' *with* $s_1 \xrightarrow{t} s_1'$ *and* $z_{s_1'} = z_{s_1} \cdot t$ *for some* $t \in Act$*, then there exists a state* s_0' *with* $s_0 \xrightarrow{t} s_0'$*,* $z_{s_0'} = z_{s_0} \cdot t$ *and* $s_0' \simeq s_1'$*.*

Proof. Let $s_0 = (w_0, z_0, \rho)$ and $s_1 = (w_1, z_1, \rho)$.

1. It holds that $\alpha \in L_{\mathcal{C}}(s_0)$ iff $w_0, z_0 \models \alpha$ iff $w_0, \langle\rangle \models \alpha^{R(E(w_0,z_0))}$ (see (8)) iff $(\alpha, E(w_0, z_0)) \in type(w_0)$ iff $(\alpha, E(w_1, z_1)) \in type(w_1)$ (since $type(w_0) = type(w_1)$ and $E(w_0, z_0) = E(w_1, z_1)$) iff $\alpha \in L_{\mathcal{C}}(s_1)$.

2. Assume w.l.o.g. $(w_0, z_0, \rho) \xrightarrow{t} (w_0, z_0 \cdot t, \rho')$. By Claim 1 of this lemma it follows that $L_{\mathcal{C}}(w_0, z_0) = L_{\mathcal{C}}(w_1, z_1)$. If follows by Lemma 25.2 that $(w_1, z_1, \rho) \xrightarrow{t} (w_1, z_1 \cdot t, \rho')$. Since $L_{\mathcal{C}}(w_0, z_0) = L_{\mathcal{C}}(w_1, z_1)$ it follows that $\widehat{\mathcal{E}}(L_{\mathcal{C}}(w_0, z_0), t) = \widehat{\mathcal{E}}(L_{\mathcal{C}}(w_1, z_1), t)$ and therefore also $E(w_0, z_0 \cdot t) = E(w_1, z_1 \cdot t)$. This implies $(w_0, z_0 \cdot t, \rho') \simeq (w_1, z_1 \cdot t, \rho')$.

□

Basically, this property of $\simeq$ allows us to replace the transition system $T_{\mathcal{P}}$ by the finite *quotient transition system* of $T_{\mathcal{P}}$ w.r.t. $\simeq$ that is defined as follows.

Definition 31 (Quotient transition system)**.** Let $\mathcal{C}$ be a context for a program $\mathcal{P}$ with $T_{\mathcal{P}} = (Q, \to, I)$. The *quotient transition system* $T_{\mathcal{P}/\simeq} = (Q_{/\simeq}, \twoheadrightarrow, I_{/\simeq})$ is defined with

$$\begin{aligned} Q_{/\simeq} &:= \{[s]_\simeq \mid s \in Q\}, \\ \twoheadrightarrow &:= \{[s]_\simeq \xrightarrow{t} [s']_\simeq \mid s \xrightarrow{t} s'\} \text{ and} \\ I_{/\simeq} &:= \{[s]_\simeq \mid s \in I\}. \end{aligned}$$

▲

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 δ. Therefore, the quotient transition system is finite. To show how the quotient transition system can be used to decide whether $T_{\mathcal{P}} \models \Phi$ for a $\mathcal{ES}$-C^2-CTL* formula Φ we introduce the notion of *propositional abstraction.* Consider $T_{\mathcal{P}} = (Q, \to, I)$ of a program $\mathcal{P}$, a $\mathcal{ES}$-C^2-CTL* state formula Φ and a context $\mathcal{C}$ for $\mathcal{P}$ that also contains all C^2-fluent sentences occurring in Φ. For any $\alpha \in \mathcal{C}$ we introduce an atomic proposition $\widehat{\alpha}$. We define the relevant set of atomic proposition $\widehat{\mathcal{C}} := \{\widehat{\alpha} \mid \alpha \in \mathcal{C}\}$. Let $\widehat{\Phi}$ denote the propositional CTL* formula that is obtained from Φ by replacing any fluent sentence in Φ by the corresponding atomic proposition. Further, we introduce a labeling $\widehat{L}_{\mathcal{C}}$ that labels the states in $T_{\mathcal{P}/\simeq}$ with subsets of $\widehat{\mathcal{C}}$ as follows: $\widehat{L}_{\mathcal{C}}([s]_\simeq) := \{\widehat{\alpha} \mid \alpha \in L_{\mathcal{C}}(s)\}$. $\widehat{L}_{\mathcal{C}}([s]_\simeq)$ is uniquely defined since all states in the equivalence class $[s]_\simeq$ have the same context label. We define a binary relation $\widehat{\sim} := \{(s, q) \in Q \times Q_{/\simeq} \mid s \in q\}$, i.e. we have $s \,\widehat{\sim}\, q$ if s is contained in the equivalence class represented by q. For two paths $\pi_s = s_0 s_1 s_2 \cdots$ and $\pi_q = q_0 q_1 q_2 \cdots$ in $T_{\mathcal{P}}$ and $T_{\mathcal{P}/\simeq}$, respectively, we write $\pi_s \,\widehat{\sim}\, \pi_q$ iff $s_i \,\widehat{\sim}\, q_i$ for all $i = 0, 1, 2, \cdots$. We assume the usual definition of satisfaction of propositional state and path CTL* formulas in states and paths in a propositional transition system.

Lemma 32. *Let* $T_{\mathcal{P}} = (Q, \to, I)$ *be the transition system of a program* $\mathcal{P}$*,* $\mathcal{C}$ *a context for* $\mathcal{P}$*,* $T_{\mathcal{P}/\simeq}$ *the quotient transition system equipped with the labeling function* $\widehat{L}_{\mathcal{C}}$ *as defined above,* Φ *a* $\mathcal{ES}$-C^2-CTL* *state formula over* $\mathcal{C}$ *and* Ψ *a* $\mathcal{ES}$-C^2-CTL* *path formula over* $\mathcal{C}$*.*

1. *If* $s \mathrel{\widehat{\sim}} q$, *then* $T_{\mathcal{P}}, s \models \Phi$ *iff* $T_{\mathcal{P}/\simeq}, q \models \widehat{\Phi}$.

2. *If* $\pi_s \mathrel{\widehat{\sim}} \pi_q$, *then* $T_{\mathcal{P}}, \pi_s \models \Psi$ *iff* $T_{\mathcal{P}/\simeq}, \pi_q \models \widehat{\Psi}$.

Proof. We prove both claims by induction over the structure of Φ and Ψ. Let $s \mathrel{\widehat{\sim}} q$ and $\Phi = \alpha$. By Lemma 30 equivalent states have the same context label. Therefore it holds that $\alpha \in L_{\mathcal{C}}(s)$ iff $\widehat{\alpha} \in \widehat{L}_{\mathcal{C}}(q)$. It follows that $T_{\mathcal{P}}, s \models \alpha$ iff $T_{\mathcal{P}/\simeq}, q \models \widehat{\alpha}$.

Assume that Claim 1 holds for Φ_1, Φ_2 and Claim 2 holds for Ψ.

$\Phi = \neg\Phi_1$: It holds that $T_{\mathcal{P}}, s \models \neg\Phi_1$ iff $T_{\mathcal{P}}, s \not\models \Phi_1$ iff $T_{\mathcal{P}/\simeq}, q \not\models \widehat{\Phi_1}$ (by induction hypothesis) iff $T_{\mathcal{P}/\simeq}, q \models \widehat{\neg\Phi_1}$.

$\Phi = \Phi_1 \wedge \Phi_2$: It holds that $T_{\mathcal{P}}, s \models \Phi_1 \wedge \Phi_2$ iff $T_{\mathcal{P}}, s \models \Phi_1$ and $T_{\mathcal{P}}, s \models \Phi_2$ iff $T_{\mathcal{P}/\simeq}, q \models \widehat{\Phi_1}$ and $T_{\mathcal{P}/\simeq}, q \models \widehat{\Phi_2}$ (by induction hypothesis) iff $T_{\mathcal{P}/\simeq}, q \models \widehat{\Phi_1 \wedge \Phi_2}$.

$\Phi = \mathsf{E}\Psi$: First, we show that $T_{\mathcal{P}}, s_0 \models \mathsf{E}\Psi$ implies $T_{\mathcal{P}/\simeq}, q_0 \models \widehat{\mathsf{E}\Psi}$ with $s_0 \mathrel{\widehat{\sim}} q_0$. There exists a path $\pi_{s_0} = s_0 s_1 s_2 \cdots$ starting in s_0 such that $T_{\mathcal{P}}, \pi_{s_0} \models \Psi$. Since $s_0 \mathrel{\widehat{\sim}} q_0$ we have $q_0 = [s_0]_\simeq$, i.e. q_0 represents the equivalence class that contains s_0. By construction of $T_{\mathcal{P}/\simeq}$ there exists a path $\pi_{q_0} = [s_0]_\simeq [s_1]_\simeq [s_2]_\simeq \cdots$ in $T_{\mathcal{P}/\simeq}$ starting in q_0. By definition it holds that $s_i \mathrel{\widehat{\sim}} [s_i]_\simeq$ for all $i = 0, 1, 2, \ldots$ and therefore $\pi_{s_0} \mathrel{\widehat{\sim}} \pi_{q_0}$. By applying the induction hypothesis for Ψ we get $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi}$ and finally $T_{\mathcal{P}/\simeq}, q_0 \models \widehat{\mathsf{E}\Psi}$.

Now, assume $T_{\mathcal{P}/\simeq}, q_0 \models \widehat{\mathsf{E}\Psi}$. There exists a path $\pi_{q_0} = q_0 q_1 q_2 \cdots$ in $T_{\mathcal{P}/\simeq}$ starting in q_0 such that $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi}$. By construction of $T_{\mathcal{P}/\simeq}$ there exists a sequence of states $s'_0, s'_1, s'_2 \in Q$ with $q_i = [s'_i]_\simeq$ for all $i = 0, 1, 2, \ldots$ such that $s'_0 s'_1 s'_2 \cdots$ is path in $T_{\mathcal{P}}$. Since $s_0 \simeq s'_0$ and Lemma 30 holds, there exists a path π_{s_0} starting in s_0 of the form $\pi_{s_0} = s_0 s_1 s_2 \cdots$ with $s_i \simeq s'_i$ for all $i = 0, 1, 2, \cdots$. It follows that $s_i \mathrel{\widehat{\sim}} q_i$ and $\pi_{s_0} \mathrel{\widehat{\sim}} \pi_{q_0}$. By applying the induction hypothesis for Ψ it follows that $T_{\mathcal{P}}, \pi_{s_0} \models \Psi$ and hence $T_{\mathcal{P}}, s_0 \models \mathsf{E}\Psi$.

Assume Claim 1 holds for Φ and Claim 2 for Ψ_1 and Ψ_2. Let $\pi_{s_0} \mathrel{\widehat{\sim}} \pi_{q_0}$.

$\Psi = \Phi$: Assume π_{s_0} starts in s_0 and π_{q_0} starts in q_0. It holds that $T_{\mathcal{P}}, \pi_{s_0} \models \Phi$ iff $T_{\mathcal{P}}, s_0 \models \Phi$ iff $T_{\mathcal{P}/\simeq}, q_0 \models \widehat{\Phi}$ (with $s_0 \mathrel{\widehat{\sim}} q_0$ and induction hypothesis) iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Phi}$.

$\Psi = \neg\Psi_1$: It holds that $T_{\mathcal{P}}, \pi_{s_0} \models \neg\Psi_1$ iff $T_{\mathcal{P}}, \pi_{s_0} \not\models \Psi_1$ iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \not\models \widehat{\Psi_1}$ (by induction) iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\neg\Psi_1}$.

$\Psi = \Psi_1 \wedge \Psi_2$: It holds that $T_{\mathcal{P}}, \pi_{s_0} \models \Psi_1 \wedge \Psi_2$ iff $T_{\mathcal{P}}, \pi_{s_0} \models \Psi_1$ and $T_{\mathcal{P}}, \pi_{s_0} \models \Psi_2$ iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi_1}$ and $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi_2}$ (by induction) iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi_1 \wedge \Psi_2}$.

$\Psi = \mathsf{X}\Psi_1$: It holds that $T_{\mathcal{P}}, \pi_{s_0} \models \mathsf{X}\Psi_1$ iff $T_{\mathcal{P}}, \pi_{s_0}[1..] \models \Psi_1$ iff $T_{\mathcal{P}/\simeq}, \pi_{q_0}[1..] \models \widehat{\Psi_1}$ (by applying the induction hypothesis to Ψ_1 and $\pi_{s_0}[1..] \mathrel{\widehat{\sim}} \pi_{q_0}[1..]$) iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\mathsf{X}\Psi_1}$.

$\Psi = \Psi_1 \,\mathsf{U}\, \Psi_2$: It holds that $T_{\mathcal{P}}, \pi_{s_0} \models \Psi_1 \,\mathsf{U}\, \Psi_2$ iff there exists a k such that $T_{\mathcal{P}}, \pi_{s_0}[k..] \models \Psi_2$ and $T_{\mathcal{P}}, \pi_{s_0}[j..] \models \Psi_1$ for all $j = 0, \ldots, k-1$. iff $T_{\mathcal{P}/\simeq}, \pi_{q_0}[k..] \models \widehat{\Psi_2}$ and $T_{\mathcal{P}/\simeq}, \pi_{q_0}[j..] \models \widehat{\Psi_1}$ for all $j = 0, \ldots, k-1$ (by induction) iff $T_{\mathcal{P}/\simeq}, \pi_{q_0} \models \widehat{\Psi_1 \,\mathsf{U}\, \Psi_2}$.

□

Together with the labeling $\widehat{L}_{\mathcal{C}}$ the quotient transition system $T_{\mathcal{P}/\simeq}$ can be viewed as a finite propositional transition system. To verify whether $T_{\mathcal{P}/\simeq} \models \widehat{\Phi}$ we can apply standard model checking techniques for CTL*. It remains to be shown how the quotient transition system can be constructed.

3.1.3 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 τ 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 $T_{\mathcal{P}/\simeq}$.

In detail, the construction works as follows:

First we introduce some auxiliary notions:

To construct the state space of the finite quotient transition system we consider the set of all types, denoted by $Types(\mathcal{P}, \mathcal{C})$. For a set τ with $\tau \subseteq TE(\mathcal{P}, \mathcal{C})$ it holds that $\tau \in Types(\mathcal{P}, \mathcal{C})$ iff

1. For all $(\alpha, E) \in TE(\mathcal{P}, \mathcal{C})$ it holds that $(\alpha, E) \in \tau$ iff $(\neg\alpha, E) \notin \tau$ (modulo double negation).
2. There exists $w \in \mathcal{W}$ and $w, \langle\rangle \models \mathcal{D}_0 \cup \{\alpha^{R(E)} \mid (\alpha, E) \in \tau\}$.

Obviously, for a world $w, \langle\rangle \models \mathcal{D}_0$ it holds that $type(w) \in Types(\mathcal{P}, \mathcal{C})$ and for any $\tau \in Types(\mathcal{P}, \mathcal{C})$ there exists world $w, \langle\rangle \models \mathcal{D}_0$ with $type(w) = \tau$.

Definition 33 (Abstract transition system)**.** Let $\mathcal{C}$ be a context for the Golog program $\mathcal{P} = (\mathcal{D}, \delta)$ and 2^{Lit} the set of all non-contradictory subsets of Lit. The *abstract transition system* $\tilde{T}_{\mathcal{P}} = (\tilde{Q}, \hookrightarrow, \tilde{I}, L_{\mathcal{C}})$ with $\hookrightarrow \subseteq \tilde{Q} \times Act \times \tilde{Q}$, $\tilde{I} \subset \tilde{Q}$ and labeling function $L_{\mathcal{C}} : \tilde{Q} \to 2^{\mathcal{C}}$ is defined as follows:

- $\tilde{Q} := Types(\mathcal{P}, \mathcal{C}) \times 2^{Lit} \times \mathsf{sub}(\delta)$;
- $L_{\mathcal{C}}(\tau, E, \rho) := \{\alpha \mid (\alpha, E) \in \tau\}$;
- $\tilde{I} := \{(\tau, E, \rho) \mid E = \emptyset, \rho = \delta\}$;
- It holds that $((\tau, E, \rho), t, (\tau, E', \rho')) \in \hookrightarrow$ iff one of the following conditions is satisfied:
  1. There exists $\alpha_1 \wedge \cdots \wedge \alpha_n?; t \in \mathsf{head}(\rho)$ s.t. $\alpha_i \in L_{\mathcal{C}}(\tau, E, \rho)$ for all $i = 1, ..., n$ and we have either $t = \epsilon$ or $\varphi^{\vec{x}}_{\vec{c}} \in L_{\mathcal{C}}(\tau, E, \rho)$ where $\Box Poss(A(\vec{x})) \equiv \varphi \in \mathcal{D}_{\text{pre}}$ and $t = A(\vec{c})$. And it holds that
  $$E' := E \setminus \neg\widehat{\mathcal{E}}(L_{\mathcal{C}}(\tau, E, \rho), t) \cup \widehat{\mathcal{E}}(L_{\mathcal{C}}(\tau, E, \rho), t)$$
  and we have $\rho' \in \mathsf{tail}(\alpha_1 \wedge \cdots \wedge \alpha_n?; t, \rho')$.
  2. There exists *no* $\alpha_1 \wedge \cdots \wedge \alpha_n?; t \in \mathsf{head}(\rho)$ s.t. $\alpha_i \in L_{\mathcal{C}}(\tau, E, \rho)$ for all $i = 1, ..., n$ and either $t = \epsilon$ or $\varphi^{\vec{x}}_{\vec{c}} \in L_{\mathcal{C}}(\tau, E, \rho)$ where $\Box Poss(A(\vec{x})) \equiv \varphi \in \mathcal{D}_{\text{pre}}$ and $t = A(\vec{c})$ and it holds that $t = \mathsf{f}$, $\rho' = \rho$ and
  $$E' := E \setminus \{\neg Fail\} \cup \{Fail\}.$$

▲

Next, we show that the construction of the abstraction transition system indeed yields the quotient transition system.

Lemma 34. *Let $\mathcal{P} = (\mathcal{D}, \delta)$ be a Golog program, $\mathcal{C}$ a context and $T_{\mathcal{P}/\simeq}$ the quotient transition system and $\tilde{T}_{\mathcal{P}} = (\tilde{Q}, \hookrightarrow, \tilde{I})$ the abstract transition system. For a transition in $T_{\mathcal{P}/\simeq}$ it holds that $[(w,z,\rho)]_\simeq \overset{t}{\twoheadrightarrow} [(w,z',\rho')]_\simeq$ iff $(type(w), E(w,z), \rho) \overset{t}{\hookrightarrow} (type(w), E(w,z'), \rho')$.*

Proof. $\Rightarrow$: Assume $t \neq \mathfrak{f}$. From $[(w,z,\rho)]_\simeq \overset{t}{\twoheadrightarrow} [(w,z',\rho')]_\simeq$ it follows that there exists $\alpha_1 \wedge \cdots \wedge \alpha_n?; t \in \mathsf{head}(\rho)$ and $\rho' \in \mathsf{tail}(\alpha_1 \wedge \cdots \wedge \alpha_n?; t, \rho)$ such that $w, z \models \alpha_1 \wedge \cdots \wedge \alpha_n$ and $w, z \models Poss(t)$ and we have $E(w,z') = E(w,z) \setminus \neg\widehat{\mathcal{E}}(L(w,z,\rho),t) \cup \widehat{\mathcal{E}}(L(w,z,\rho),t)$. Therefore, it holds that $(\alpha_i, E(w,z)) \in type(w)$ and $(\varphi, E(w,z)) \in type(w)$ for $Poss(t) \equiv \varphi \in \mathcal{D}_{\text{pre}}$. It is implied that $(type(w), E(w,z), \rho) \overset{t}{\hookrightarrow} (type(w), E(w,z'), \rho')$. Now, assume $t = \mathfrak{f}$. There is no guarded action in the head of ρ such that the guard and the precondition is satisfied. Therefore, the second case in the definition of $\hookrightarrow$ applies here and we get $(type(w), E(w,z), \rho) \overset{t}{\hookrightarrow} (type(w), E(w,z'), \rho')$ with $z' = z \cdot \mathfrak{f}$ and $\rho' = \rho$.

$\Leftarrow$: Consider a transition in $\tilde{T}_{\mathcal{P}}$ of the form $(\tau, E, \rho) \overset{t}{\hookrightarrow} (\tau, E', \rho')$. There exists $(w,z) \in \mathcal{W} \times \mathcal{Z}$ with $\tau = type(w)$ and $E = E(w,z)$. Assume that the first case in the definition of $\hookrightarrow$ applies. There exists $\alpha?; t \in \mathsf{head}(\rho)$ and $(\alpha_i, E) \in type(w)$ for the conjuncts α_i in α. It is implied that $w, z \models \alpha$ and $w, z \models Poss(t)$. Therefore, we have $(w,z,\rho) \overset{t}{\rightarrow} (w, z \cdot t, \rho')$ by Lemma 15 and $[(w,z,\rho)]_\simeq \overset{t}{\twoheadrightarrow} [(w, z \cdot t, \rho')]_\simeq$. Clearly, the claim also holds if $t = \mathfrak{f}$.

□

The decision procedure for checking satisfiability of an $\mathcal{ES}$-C^2-CTL* formula Φ in a Golog program $\mathcal{P} = (\mathcal{D}, \delta)$ works as follows. First, we choose a context $\mathcal{C}$ that covers all fluent sentences that occur in Φ and construct the finite abstract transition system based on this context. Then we apply standard model checking techniques to check satisfiability of the propositional abstraction $\widehat{\Phi}$ in the finite abstract transition system.

To analyze the complexity of the decision procedure consider a Golog program $\mathcal{P} = (\mathcal{D}, \delta)$ and a property Φ. Obviously, it is possible to choose a context $\mathcal{C}$ covering also all fluent sentences occurring in Φ with size linear in the size of $\mathcal{P}$ and Φ. To build the abstract transition system we guess a subset $\tau \subseteq TE(\mathcal{P}, \mathcal{C})$ such that either $(\alpha, E) \in \tau$ or $(\neg\alpha, E) \in \tau$ for all $\alpha \in \mathcal{C}$ and all $E \in 2^{Lit}$. The set τ is exponentially large in the size of $\mathcal{P}$ and $\mathcal{C}$. To test whether τ represents the type of a world we have to test consistency of an exponentially large C^2 knowledge base. Consistency in C^2 can be decided in NEXPTIME [PST00]. Therefore checking whether τ is a type can be done in 2-NEXPTIME. Given a type τ, there are $|2^{Lit} \times \mathsf{sub}(\delta)|$ many reachable states in the abstract transition system. The size of $|2^{Lit} \times \mathsf{sub}(\delta)|$ is at most exponential in the size of $\mathcal{P}$. This reachable fragment can be constructed by reading off the value of the effect function from the SSAs in the BAT $\mathcal{D}$ and by using the head and tail functions in exponential time in the size of the input. The satisfiability test of the propositional abstraction $\widehat{\Phi}$ in the reachable fragment from the initial state $(\tau, \emptyset, \delta)$ can be done in polynomial space in the size of $\widehat{\Phi}$ and the size of the reachable fragment. It is implied that this test can be done in exponential space and therefore in double-exponential time in the size of the input. Therefore, the complexity upper bound of this decision procedure is in 2-NEXPTIME.

Theorem 35. *Satisfiability of a $\mathcal{ES}$-C^2-CTL* formula in a Golog program $\mathcal{P} = (\mathcal{D}, \delta)$ consisting of a local-effect BAT $\mathcal{D}$ and a program over ground actions δ is decidable in* 2-NEXPTIME.

3.2 Undecidable Extensions

In this section we show 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. Successor state axioms are all local-effect.
3. Disallow pick operators in GOLOG programs.

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

Cleary, dropping restriction 1 immediately leads to undecidability as this would allow us to formulate arbitrary first-order sentences as tests and preconditions:

Theorem 36. *The verification problem is undecidable if we drop restriction 1.*

Proof. To see why, consider a basic action theory $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{pre}} \cup \mathcal{D}_{\text{post}}$ where $\mathcal{D}_0 = \mathcal{D}_{\text{post}} = \emptyset$ and $\mathcal{D}_{\text{post}} = \{\Box Poss(A) \equiv \top\}$. That is to say we have an empty initial theory, no successor state axioms and some action A that is always possible. Furthermore, suppose δ is the program A^ω, which repeats A indefinitely. Let α be a sentence that mentions at most rigid predicates and functions, which implies that the truth value of α does not change by the application of actions. Then α is valid iff $\mathsf{EF}\alpha$ is valid in T_δ. Since any sentence of first-order logic corresponds to such an α, the proposition follows. □

In the following two subsections, we discuss the remaining two restrictions.

3.2.1 Undecidability due to Non-Local Effects

In this subsection, we show that the verification problem becomes undecidable again once we allow arbitrary SSAs, but still restrict ourselves to ground actions only.

To prove undecidability, we show that the resulting fragment is sufficiently expressive to simulate a Turing machine. That is, given a Turing machine $\mathcal{T}$, we construct a BAT $\mathcal{D}_\mathcal{T}$ and a program $\delta_\mathcal{T}$ such that the runs they admit correspond precisely to the computations of $\mathcal{T}$.

Here we consider a deterministic Turing machine with a right-infinite tape. We assume w.l.o.g. that it never attempts to move its head left when its head is on the leftmost tape cell. Let $\mathcal{T} = (Q, \Sigma, \Gamma, \theta, q_0, \Box, q_F)$ be a Turing Machine, where

- Q is a finite set of states;
- Σ is the finite input alphabet;
- Γ is the finite tape alphabet, $\Gamma = \Sigma \cup \{\Box\}$;
- $\theta : Q \times \Gamma \to Q \times \Gamma \times \{\ell, n, r\}$ is the transition function;
- $q_0 \in Q$ is the initial state;
- $\Box \in \Gamma$ is the blank symbol;
- $q_F \in Q$ is the (accepting) final state.

Note that in the following we will assume that there is no input word, i.e. the Turing machine starts on an empty tape. This is without loss of generality since a run of some Turing machine M on an input word u can always be simulated by devising a Turing machine M_u that first writes u on the tape, goes back to the initial position and from then on operates like M. In other words, the Halting Problem is equally undecidable if the machine starts on the empty tape as it is if an input word is given.

A *configuration* of the Turing machine $\mathcal{T}$ is a word uqv with $u, v \in \Gamma^*$ and $q \in Q$. The meaning of the word uqv is that the right-infinite tape contains the word uv with only blanks behind it and $\mathcal{T}$ is in state q, and the head is on the leftmost symbol of v.

Fluents and Rigids

The BAT $\mathcal{D}_\mathcal{T}$ uses the following fluents:

- the unary fluent predicate $Pos(x)$, intuitively denoting that the head is currently at tape cell x;
- for each $q \in Q$, a 0-ary fluent $State_q$, intuitively denoting that the machine is currently in state q;
- for each $b \in \Gamma$, a unary fluent $Symbol_b(x)$, intuitively denoting that tape cell x currently holds symbol b;
- the unary fluent predicate $Visited(x)$, intuitively meaning that the tape cell x was visited at least once during the run of the Turing machine;
- the unary fluent predicate $Right(x)$, denoting the rightmost cell that has been visited.

Moreover, we have

- the binary rigid predicate $NextTo(x, y)$, intuitively meaning that tape cell y is directly right of cell x.

Actions

The BAT $\mathcal{D}_\mathcal{T}$ will use the following actions: one 0-ary action $apply_{(q,b,q',b',m)}$ for each transition $(q, b) \in Q \times \Gamma$ and $(q', b', m) \in Q \times \Gamma \times \{\ell, r, n\}$ such that $\theta(q, b) = (q', b', m)$.

Constants

Finally, we have a distinguished constant pos_0 representing the leftmost tape cell.

Initial Theory

The initial theory contains the following sentences. First, the head is initially on the leftmost tape cell pos_0 (and is only on pos_0):

$$\forall x.(Pos(x) \equiv x = pos_0). \tag{9}$$

Furthermore, the machine is in the initial state:

$$State_{q_0} \wedge \bigwedge_{q \in Q \setminus \{q_0\}} \neg State_q. \tag{10}$$

Next, all tape cells are blank:

$$\forall x.(Symbol_{\square}(x) \wedge \bigwedge_{b \in \Sigma} \neg Symbol_b(x)). \tag{11}$$

Moreover, only the cell under the head's starting position has been visited, being the rightmost such cell:

$$\forall x.(\, Visited(x) \equiv x = pos_0); \tag{12}$$

$$\forall x.(Right(x) \equiv x = pos_0). \tag{13}$$

Finally, each tape cell has a cell right next to it:

$$\forall x.\exists y.NextTo(x, y). \tag{14}$$

We don't require that every cell has exactly one cell next to it. We also allow several cells next to one cell. The idea is to represent a configuration of the Turing machine $\mathcal{T}$ in a world w satisfying the reduction BAT $\mathcal{D}_{\mathcal{T}}$ as an acyclic graph of visited cells with a special structure. The nodes of this graph are the standard names n such that $Visited(n)$ is true in the world w. The edge relation is given by the *NextTo* predicate. The standard name pos_0 is the root of this graph, i.e. pos_0 has no ingoing edges (there is no n such that $NextTo(n, pos_0)$ holds) and all nodes in the graph are connected to pos_0. For one node n in the graph there can be several paths from pos_0 to this node n. But all these paths have the same length. This means we can distinguish the nodes by their distance from pos_0. All nodes with the same distance belong to the same level and represent a single tape cell in the configuration. There are only finitely many levels since a configuration of $\mathcal{T}$ is a given as a finite word.

We have to ensure that the changes corresponding to a transition action affect all nodes on the current level, i.e. the level labeled with the fluent *Pos*, simultaneously. Intuitively, a move of the head to the left or right means that we go one level to the left or to the right in the graph, respectively. The rightmost level (i.e. the nodes with the largest distance from pos_0) in the graph are labeled with the fluent *Right*. To enforce the desired structure we require that a cell labeled with *Right* has only cells next to it that are not visited. This constraint is modeled as a precondition of each transition action.

Preconditions

A transition can be taken just in case the Turing machine is in the correct state and reads the correct symbol. In addition we have to require that a cell x with $Right(x)$ has only cells next to it that are not visited. For every transition $\theta(q, b) = (q', b', m)$, we thus have

$$\begin{aligned} \square Poss(apply_{(q,b,q',b',m)}) \equiv\ & State_q \wedge \exists x.Pos(x) \wedge \forall x.(Pos(x) \supset Symbol_b(x)) \\ & \wedge \forall x, y.\big(Right(x) \wedge NextTo(x, y) \supset \neg Visited(y)\big). \end{aligned}$$

Successor State Axioms

For the fluents, we have the following successor state axioms. First, the $State_q$ fluents change according to the transition action that is applied:

$$\square[a]State_q \equiv \bigvee_{\theta(q',b')=(q,b,m)} a = apply_{(q',b',q,b,m)}$$

Similarly, the symbol under the head changes accordingly. That is a cell contains a symbol b just in case it was just overwritten with b or it already contained b and was not overwritten by anything else.

$$\begin{aligned}\Box[a]\mathit{Symbol}_b(x) \equiv \quad & (\mathit{Pos}(x) \wedge \bigvee_{\theta(q',b')=(q,b,m)} a = \mathit{apply}_{(q',b',q,b,m)}) \\ & \vee \mathit{Symbol}_b(x) \wedge \\ & \neg(\mathit{Pos}(x) \wedge \bigvee_{\substack{\theta(q',b')=(q'',b'',m') \\ b''\neq b}} a = \mathit{apply}_{(q',b',q'',b'',m')})\end{aligned} \tag{15}$$

The head position again changes according to the movement m of the applied transition action:

$$\Box[a]\mathit{Pos}(x) \equiv \bigvee_{\theta(q,b)=(q',b',m)} a = \mathit{apply}_{(q,b,q',b',m)} \wedge \mathit{NextPos}_m(x)$$

Above, $\mathit{NextPos}_m$ is an abbreviation as follows:

$$\mathit{NextPos}_\ell(x) \stackrel{def}{=} \exists y.\big(\mathit{Pos}(y) \wedge \mathit{NextTo}(x,y)\big) \wedge \mathit{Visited}(x); \tag{16}$$

$$\mathit{NextPos}_r(x) \stackrel{def}{=} \exists y.\big(\mathit{Pos}(y) \wedge \mathit{NextTo}(y,x)\big); \tag{17}$$

$$\mathit{NextPos}_n(x) \stackrel{def}{=} \mathit{Pos}(x). \tag{18}$$

Thus, a cell x belongs to the resulting head position, i.e. the next level in the configuration tree, after moving left, if this cell x was visited before and is on the left-hand side of a cell y in the current level according to the $\mathit{NextTo}(x,y)$ relation. Such an x always exists since we have assumed that in the leftmost position pos_0 the Turing machine don't move to the left. The result of moving right is determined in a similar fashion, but here x can be also a non-visited cell. If the head is not moved, the new position is identical to the previous one.

Note that due to the existentially quantified subformulas $\exists y. \ldots$ above, the successor state axioms are not local-effect. The new position x of the head depends on individuals y that are not given as arguments of the action term.

$$\begin{aligned}\Box[a]\,\mathit{Visited}(x) \equiv \quad & \bigvee_{\theta(q,b)=(q',b',m)} a = \mathit{apply}_{(q,b,q',b',m)} \wedge \mathit{NextPos}_m(x) \\ & \vee \mathit{Visited}(x)\end{aligned} \tag{19}$$

A cell has been visited iff we just moved there or it has been visited before. The rightmost visited cell is a non-visited cell reached by moving right or the previous rightmost visited cell if the head was not in the previous rightmost position or we didn't move to the right.

$$\begin{aligned}\Box[a]\mathit{Right}(x) \equiv \quad & \bigvee_{\theta(q,b)=(q',b',r)} a = \mathit{apply}_{(q,b,q',b',r)} \wedge \mathit{NextPos}_r(x) \wedge \neg\mathit{Visited}(x) \\ & \vee \mathit{Right}(x) \wedge \neg\big(\mathit{Pos}(x) \wedge \bigvee_{\theta(q,b)=(q',b',r)} a = \mathit{apply}_{(q,b,q',b',r)}\big)\end{aligned} \tag{20}$$

GOLOG Program

Let $\theta = \{((q_1^1, b_1^1), (q_1^2, b_1^2, m_1)), \ldots, ((q_n^1, b_n^1), (q_n^2, b_n^2, m_n))\}$. We define our GOLOG program to perform a non-terminating loop, where in each cycle a transition action is executed. Inside the

loop we have a non-deterministic choice over all transition actions:

$$\delta_{\mathcal{T}} \stackrel{def}{=} (apply_{(q_1^1,b_1^1,q_1^2,b_1^2,m_1)} \mid \cdots \mid apply_{(q_n^1,b_n^1,q_n^2,b_n^2,m_n)})^* \tag{21}$$

We note that all transitions (if any) of $\delta_{\mathcal{T}}$ always lead to a configuration which has $\delta_{\mathcal{T}}$ as program again:

Lemma 37. *If* $\langle\langle\rangle, \delta_{\mathcal{T}}\rangle \xrightarrow{w}{}^* \langle z, \delta'\rangle$, *then* $\delta' = \delta_{\mathcal{T}}$.

Proof. This property follows directly from the transition semantics. □

In this paper, we use a notation for Turing machine configurations that includes all blank symbols for previously visited cells. For example, if we start with an empty tape, then write the string "100", move left and erase the last "0", we write:

$$q_0\Box \vdash 1q_1\Box \vdash 10q_2\Box \vdash 100q_3\Box \vdash 10q_40\Box \vdash 10q_5\Box\Box$$

For the correctness proof, we need the following notion of a situation encoding a Turing configuration.

Definition 38. Let uqv be a Turing machine configuration with $q \in Q$, $u = b_1 \ldots b_k$ $(k \geq 0)$ and $v = b_{k+1} \cdots b_{k+l}$ $(l \geq 0)$. Then we define

$$\begin{aligned} Encode(uqv) \stackrel{def}{=}\ & State_q \wedge \bigwedge_{q' \in Q \setminus \{q\}} \neg State_{q'} \\ & \wedge \exists x_1 \ldots \exists x_{k+l}.\ \bigwedge_{i \neq j} (x_i \neq x_j) \wedge \bigwedge_{i=1}^{k+l-1} NextTo(x_i, x_{i+1}) \\ & \wedge \forall y(Pos(y) \equiv y = x_{k+1}) \\ & \wedge \forall y(Right(y) \equiv y = x_{k+l}) \\ & \wedge \bigwedge_{i=1}^{k+l} Symbol_{b_i}(x_i) \wedge \bigwedge_{b \neq b_i} \neg Symbol_b(x_i) \\ & \wedge \bigwedge_{i=1}^{k+l} Visited(x_i) \wedge\ \forall y.\ \bigwedge_{i=1}^{k+l} (y \neq x_i) \supset \neg Visited(y) \end{aligned} \tag{22}$$

▲

In the following, we rely on the definition from [LL04] that, given a world w and a BAT $\mathcal{D}$, determines a world $w_{\mathcal{D}}$ that is like w in the initial situation, but where *Poss* and the other fluents are interpreted in successive situations such that the precondition and successor state axioms from $\mathcal{D}$ are satisfied. $w_{\mathcal{D}}$ is unique and guaranteed to exist, and if $w \models \mathcal{D}_0$, then $w_{\mathcal{D}} \models \mathcal{D}$.

The central ingredient for the undecidability proof is the fact that a Turing machine configuration is reachable just in case there is a world that encodes that configuration in a situation reachable by the corresponding program transitions:

Lemma 39. $q_0\Box \vdash^* uqv$ *iff there is some world* w *with* $w \models \mathcal{D}_{\mathcal{T}}$ *such that* $\langle\langle\rangle, \delta_{\mathcal{T}}\rangle \xrightarrow{w}{}^* \langle z, \delta_{\mathcal{T}}\rangle$ *and* $w, z \models Encode(uqv)$.

Proof. We prove this property by induction on the length of the computation. The base case is trivial: The only Turing configuration reachable in zero steps is $q_0\Box$, and clearly $\mathcal{D}_0 \models$

Encode$(q_0\Box)$. Since $\mathcal{D}_0$ is obviously satisfiable, there is a world w with $w \models \mathcal{D}_0$, hence $w_{\mathcal{D}_\mathcal{T}} \models \mathcal{D}_\mathcal{T}$, and $w_{\mathcal{D}_\mathcal{T}} \models$ *Encode*$(q_0\Box)$.

"$\Rightarrow$": We construct a canonical model as follows. Assume that the set of object standard names is given by $\mathcal{N}_O = \{pos_0, n_1, n_2, \dots\}$. Let w be a world so that for all z,

$$\begin{aligned}
w[NextTo(pos_0, n_1), z] &= 1 \\
w[NextTo(n_i, n_j), z] &= 1 \text{ iff } j = i+1, \text{ for all } i = 1, 2, 3, \dots \\
w[Pos(n), \langle\rangle] &= 1 \text{ iff } n = pos_0 \\
w[Visited(n), \langle\rangle] &= 1 \text{ iff } n = pos_0 \\
w[Right(n), \langle\rangle] &= 1 \text{ iff } n = pos_0 \\
w[State_{q_0}, \langle\rangle] &= 1 \\
w[State_q, \langle\rangle] &= 0 \text{ iff } q \in Q \setminus \{q_0\} \\
w[Symbol_\Box(n), \langle\rangle] &= 1 \text{ for all } n \in \mathcal{N}_O \\
w[Symbol_b(n), \langle\rangle] &= 0 \text{ for all } n \in \mathcal{N}_O, \text{ for all } b \in \Sigma.
\end{aligned}$$

Now let $w_\mathcal{T} = w_{\mathcal{D}_\mathcal{T}}$. Clearly, $w_\mathcal{T} \models \mathcal{D}_\mathcal{T}$. It is also easy to check that the model correctly captures the computations of the Turing machine, and that hence $\langle\langle\rangle, \delta_\mathcal{T}\rangle \xrightarrow{w}{}^* \langle z, \delta_\mathcal{T}\rangle$ and $w, z \models$ *Encode*(uqv) if z consists of the transition actions $apply_{(q,b,q',b',m)}$ for the transitions used in the computation $q_0\Box \vdash^* uqv$.

"$\Leftarrow$": In case of the if-direction, let w be some world with $w \models \mathcal{D}_\mathcal{T}$ such that

$$\langle\langle\rangle, \delta_\mathcal{T}\rangle \xrightarrow{w}{}^* \langle z, \delta_\mathcal{T}\rangle \xrightarrow{w} \langle z \cdot t, \delta_\mathcal{T}\rangle$$

and $w, z \cdot t \models$ *Encode*(uqv), where t is equal to some $apply_{(q,b,q',b',m)}$. It is straightforward to show that $w, z \models$ *Encode*$(u'q'v')$ for some predecessor configuration $u'q'v'$. By induction, $q_0\Box \vdash^* u'q'v'$, and it can then be shown that $u'q'v' \vdash uqv$. □

As a consequence, we get:

Corollary 40. *If $q_0\Box \vdash^* uq_Fv$ then $\mathsf{EFState}_{q_F}$ is satisfiable in $\mathcal{P} = (\mathcal{D}_\mathcal{T}, \delta_\mathcal{T})$.*

The other direction also follows from the construction of $\mathcal{P} = (\mathcal{D}_\mathcal{T}, \delta_\mathcal{T})$. Due to the undecidability of the Halting Problem for Turing machines, we obtain:

Theorem 41. *Satisfiability of a $\mathcal{ES}$-C^2-CTL* formula in a* Golog *program over ground actions based on an unrestricted BAT is undecidable.*

3.2.2 Undecidability due to Pick Operators

In order to show that the verification problem is equally undecidable in case we introduce pick operators (but require SSAs to be local-effect again), we proceed as follows. We represent a given Turing machine $\mathcal{T}$ in a very similar manner as in the previous section. As before, we use the fluents $Pos(x)$, $State_q$, $Symbol_b(x)$, $Visited(x)$ and $Right(x)$ as well as the rigid predicate $NextTo(x, y)$ and the constant pos_0 to encode configurations.

Actions

The BAT $\mathcal{D}_\mathcal{T}$ uses one binary action $apply_{(q,b,q',b',m)}(x, y)$ for each transition $\theta(q, b) = (q', b', m)$. Different from the previous subsection, actions thus now have two parameters. Intuitively, the x parameter denotes the current tape cell, while y is the one to which the head moves after writing a new symbol.

Initial Theory

The initial theory is exactly the same as before, i.e. consists of formulas (9) to (14).

Preconditions

For every transition action $apply_{(q,b,q',b',m)}(x,y)$ we have:

$$\Box Poss(apply_{(q,b,q',b',m)}(x,y)) \equiv \forall x,y.\big(Right(x) \wedge NextTo(x,y) \supset \neg Visited(y)\big)$$

Successor State Axioms

Successor state axioms are very similar to the ones in the previous subsection. The only differences are that we now have to quantify action parameters that are not arguments of the corresponding fluent, and that we do not require the *NextPos* subformulas since the next position of the head is determined by the second action parameter:

$$\Box[a]Pos(x) \equiv \bigvee_{\theta(q,b)=(q',b',m)} \exists y.a = apply_{(q,b,q',b',m)}(y,x)$$

$$\Box[a]State_q \equiv \bigvee_{\theta(q',b')=(q,b,m)} \exists x,y.a = apply_{(q',b',q,b,m)}(x,y)$$

$$\begin{aligned}\Box[a]Symbol_b(x) \equiv\ & (Pos(x) \wedge \bigvee_{\theta(q',b')=(q,b,m)} \exists y.a = apply_{(q',b',q,b,m)}(x,y)\\ & \vee Symbol_b(x) \wedge\\ & \neg(Pos(x) \wedge \bigvee_{\substack{\theta(q',b')=(q'',b'',m')\\ b''\neq b}} \exists y.a = apply_{(q',b',q'',b'',m')}(x,y))\end{aligned}$$

$$\Box[a]Visited(x) \equiv \bigvee_{\theta(q,b)=(q',b',m)} \exists y.a = apply_{(q,b,q',b',m)}(y,x) \vee Visited(x)$$

$$\begin{aligned}\Box[a]Right(x) \equiv & \bigvee_{\theta(q,b)=(q',b',r)} \exists y.a = apply_{(q,b,q',b',r)}(y,x) \wedge \neg Visited(x)\\ & \vee Right(x) \wedge \neg\big(Pos(x) \wedge \bigvee_{\theta(q,b)=(q',b',r)} \exists y.a = apply_{(q,b,q',b',r)}(x,y)\big)\end{aligned}$$

Note that the above axioms are indeed local-effect according to Definition 17.

Golog Program

Let $\theta = \{((q_1^1,b_1^1),(q_1^2,b_1^2,m_1)),\ldots,((q_n^1,b_n^1),(q_n^2,b_n^2,m_n))\}$. The Golog program we use in this construction does a non-terminating loop, where in each a cycle a subbranch is chosen that executes one of the rules of the Turing machine. Action parameters are chosen using pick operators, and test operators ensure that the machine is in the correct state and reads the right symbol to apply that rule and that the x parameter is indeed the current head position and y

the resulting one:

$$\delta_{\mathcal{T}} \stackrel{def}{=} \left(\delta_{(q_1^1,b_1^1,q_1^2,b_1^2,m_1)} \mid \cdots \mid \delta_{(q_n^1,b_n^1,q_n^2,b_n^2,m_n)}\right)^{\omega} \tag{23}$$

$$\delta_{(q,b,q',b',\ell)} \stackrel{def}{=} \tag{24}$$
$$\pi x\, \pi y.$$
$$\quad Pos(x) \wedge NextTo(y,x) \wedge Visited(y) \wedge State_q \wedge Symbol_b(x)?;$$
$$\quad apply_{(q,b,q',b',\ell)}(x,y)$$

$$\delta_{(q,b,q',b',r)} \stackrel{def}{=} \tag{25}$$
$$\pi x\, \pi y.$$
$$\quad Pos(x) \wedge NextTo(x,y) \wedge \big(Right(x) \vee Visited(y)\big) \wedge State_q \wedge Symbol_b(x)?;$$
$$\quad apply_{(q,b,q',b',r)}(x,y)$$

$$\delta_{(q,b,q',b',n)} \stackrel{def}{=} \tag{26}$$
$$\pi x.$$
$$\quad Pos(x) \wedge State_q \wedge Symbol_b(x)?;$$
$$\quad apply_{(q,b,q',b',n)}(x,x)$$

Since we can show Lemmas 37 and 39 as well as Corollary 40 accordingly for the construction in this subsection, we get:

Theorem 42. *Satisfiability of an $\mathcal{ES}$-C^2-CTL* formula in a* Golog *program over non-ground actions based on a local-effect BAT is undecidable.*

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 [DLP12] who show decidability for first-order μ-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. [HCM$^+$13] consider the verification of μ-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 [CLL13] and [BZ13] 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 [ZC13] 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.

References

[BCF+99] Burgard, Wolfram ; Cremers, Armin B. ; Fox, Dieter ; Hähnel, Dirk ; Lakemeyer, Gerhard ; Schulz, Dirk ; Steiner, Walter ; Thrun, Sebastian: Experiences with an Interactive Museum Tour-Guide Robot. In: *Artificial Intelligence* 114 (1999), Nr. 1–2, S. 3–55

[BCM+03] Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003

[BLM+05] Baader, Franz ; Lutz, Carsten ; Miličić, Maja ; Sattler, Ulrike ; Wolter, Frank: Integrating Description Logics and Action Formalisms: First Results. In: Veloso, Manuela M. (Hrsg.) ; Kambhampati, Subbarao (Hrsg.): *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, AAAI Press, 2005, S. 572–577

[BLM10] Baader, Franz ; Liu, Hongkai ; Mehdi, Anees ul: Verifying Properties of Infinite Sequences of Description Logic Actions. In: Coelho, Helder (Hrsg.) ; Studer, Rudi (Hrsg.) ; Wooldridge, Michael (Hrsg.): *Proceedings of the Nineteenth European Conference on Artificial Intelligence (ECAI 2010)* Bd. 215, IOS Press, 2010 (Frontiers in Artifical Intelligence and Applications), S. 53–58

[BZ13] Baader, Franz ; Zarriess, Benjamin: Verification of Golog Programs over Description Logic Actions. In: Fontaine, Pascal (Hrsg.) ; Ringeissen, Christophe (Hrsg.) ; Schmidt, Renate A. (Hrsg.): *Proceedings of the Ninth International Symposium on Frontiers of Combining Systems (FroCoS'13)* Bd. 8152, Springer-Verlag, 2013 (Lecture Notes in Artificial Intelligence)

[CL08] Classen, Jens ; Lakemeyer, Gerhard: A Logic for Non-Terminating Golog Programs. In: Brewka, Gerhard (Hrsg.) ; Lang, Jérôme (Hrsg.): *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008)*, AAAI Press, 2008, S. 589–599

[CLL13] Classen, Jens ; Liebenberg, Martin ; Lakemeyer, Gerhard: On Decidable Verification of Non-terminating Golog Programs. In: Ji, Jianmin (Hrsg.) ; Strass, Hannes (Hrsg.) ; Wang, Xun (Hrsg.): *Proceedings of the 10th International Workshop on Nonmonotonic Reasoning, Action and Change (NRAC 2013)*, 2013, S. 13–20

[DLP12] De Giacomo, Giuseppe ; Lespérance, Yves ; Patrizi, Fabio: Bounded Situation Calculus Action Theories and Decidable Verification. In: *Proc. KR 2012*, 2012

[FL08] Ferrein, Alexander ; Lakemeyer, Gerhard: Logic-based Robot Control in Highly Dynamic Domains. In: *Robotics and Autonomous Systems* (2008)

[GLL00] Giacomo, Giuseppe D. ; Lespérance, Yves ; Levesque, Hector J.: ConGolog, a concurrent programming language based on the situation calculus. In: *Artificial Intelligence* 121 (2000), Nr. 1–2, S. 109–169

[GS10] Gu, Yilan ; Soutchanski, Mikhail: A description logic based situation calculus. In: *Annals of Mathematics and Artificial Intelligence* 58 (2010), Nr. 1–2, S. 3–83

[GTR97] Giacomo, Giuseppe D. ; Ternovska, Evgenia ; Reiter, Raymond: Non-terminating Processes in the Situation Calculus. In: *Working Notes of "Robots, Softbots, Immobots: Theories of Action, Planning and Control", AAAI'97 Workshop*, 1997

[HCM+13] Hariri, Babak B. ; Calvanese, Diego ; Montali, Marco ; Giacomo, Giuseppe D. ; Masellis, Riccardo D. ; Felli, Paolo: Description Logic Knowledge and Action Bases. In: *Journal of Artificial Intelligence Research* 46 (2013), S. 651–686. `http://dx.doi.org/10.1613/jair.3826`. – DOI 10.1613/jair.3826

[LL04] Lakemeyer, Gerhard ; Levesque, Hector J.: Situations, Si! Situation Terms, No! In: Dubois, Didier (Hrsg.) ; Welty, Christopher A. (Hrsg.) ; Williams, Mary-Anne (Hrsg.): *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR 2004)*, AAAI Press, 2004, S. 516–526

[LL05] Liu, Yongmei ; Levesque, Hector J.: Tractable Reasoning with Incomplete First-Order Knowledge in Dynamic Systems with Context-Dependent Actions. In: Kaelbling, Leslie P. (Hrsg.) ; Saffiotti, Alessandro (Hrsg.): *IJCAI*, Professional Book Center, 2005. – ISBN 0938075934, S. 522–527

[LL09] Liu, Yongmei ; Lakemeyer, Gerhard: On First-Order Definability and Computability of Progression for Local-Effect Actions and Beyond. In: Boutilier, Craig (Hrsg.): *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009)*, AAAI Press, 2009, S. 860–866

[LL10] Lakemeyer, Gerhard ; Levesque, Hector J.: A semantic characterization of a useful fragment of the situation calculus with knowledge. In: *Artificial Intelligence* 175 (2010), Nr. 1, S. 142–164. – in press

[LRL+97] Levesque, Hector J. ; Reiter, Raymond ; Lespérance, Yves ; Lin, Fangzhen ; Scherl, Richard B.: GOLOG: A Logic Programming Language for Dynamic Domains. In: *Journal of Logic Programming* 31 (1997), Nr. 1–3, S. 59–83

[MH69] McCarthy, John ; Hayes, Patrick: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B. (Hrsg.) ; Michie, D. (Hrsg.): *Machine Intelligence 4*. New York : American Elsevier, 1969, S. 463–502

[PST00] Pacholski, Leszek ; Szwast, Wieslaw ; Tendera, Lidia: *Complexity Results for First-Order Two-Variable Logic with Counting*. 2000

[Rei01] Reiter, Raymond: *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001

[VLL08] Vassos, Stavros ; Lakemeyer, Gerhard ; Levesque, Hector J.: First-Order Strong Progression for Local-Effect Basic Action Theories. In: Brewka, Gerhard (Hrsg.) ; Lang, Jérôme (Hrsg.): *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008)*, AAAI Press, 2008, S. 662–672

[ZC13] Zarriess, Benjamin ; Classen, Jens: On the Decidability of Verifying LTL Properties of Golog Programs / Chair of Automata Theory, TU Dresden. Dresden, Germany, 2013 (13-10). – LTCS-Report. – See http://lat.inf.tu-dresden.de/research/reports.html.