Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Exploring the Boundaries of Decidable Verification of Non-Terminating Golog Programs

Jens Claßen, Martin Liebenberg and Gerhard Lakemeyer
Knowledge-Based Systems Group
RWTH Aachen University, Germany
{classen,liebenberg,gerhard}@kbsg.rwth-aachen.de

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

Abstract

The action programming language GOLOG has been found useful for the control of autonomous agents such as mobile robots. In scenarios like these, tasks are often open-ended so that the respective control programs are non-terminating. Before deploying such programs on a robot, it is often desirable to verify that they meet certain requirements. For this purpose, Claßen and Lakemeyer recently introduced algorithms for the verification of temporal properties of GOLOG programs. However, given the expressiveness of GOLOG, their verification procedures are not guaranteed to terminate. In this paper, we show how decidability can be obtained by suitably restricting the underlying base logic, the effect axioms for primitive actions, and the use of actions within GOLOG programs. Moreover, we show that dropping any of these restrictions immediately leads to undecidability of the verification problem.

Introduction

The GOLOG family of action programming languages (De Giacomo, Lespérance, and Levesque 2000; Levesque et al. 1997) and its underlying logic, the Situation Calculus (McCarthy and Hayes 1969; Reiter 2001), have been found useful for the control of autonomous agents such as mobile robots (Burgard et al. 1999; Ferrein and Lakemeyer 2008). Usually, the task of such an agent is open-ended, that is, there is no predefined goal or terminal state that the agent tries to reach, but (at least ideally) the robot works indefinitely, and its corresponding control program is thus *non-terminating*.

As a simple example, consider a mobile robot whose task it is to remove dirty dishes from certain locations in an office on request. A program for this robot might look like this:

loop : **while** $(\exists x.OnRobot(x))$ **do**
$\quad\quad\pi x.unload(x)$ **endWhile**;
$\quad\pi y.goToRoom(y)$;
$\quad$**while** $(\exists x.DirtyDish(x,y))$ **do**
$\quad\quad\pi x.load(x,y)$ **endWhile**;
$\quad goToKitchen$

We assume that the robot is initially in the kitchen, its home base. During each iteration of its infinite control loop, the robot first unloads all dishes it carries, then selects a room in the office building, goes to this room, loads all dirty dishes in this room, and returns to the kitchen. Here, $DirtyDish(x,y)$ should be read as "dirty dish x is in room y" and $load(x,y)$ as "load dish x in room y." During the execution of the program, people can send requests indicating that there is a dirty dish in a certain room (not shown here). Action pre- and postconditions are represented by an action theory incorporating Reiter's (1991) solution to the frame problem.

Before actually deploying such a program on the robot and executing it in the physical world, it is often desirable to verify that it meets certain requirements, for example that "every request will eventually be served by the robot" or whether "it is possible that no request is ever served."

Nowadays the automated, formal verification of non-terminating processes is typically associated with *model checking*. The classical approach for propositional temporal logics such as CTL and LTL (Clarke and Emerson 1981; Clarke, Emerson, and Sistla 1986) can be boosted to handle very large (Burch et al. 1992; McMillan 1993) and even infinite-state systems (Burkart et al. 2001). However, the underlying formalisms are usually chosen very carefully to ensure decidability or tractability of the method. Consequently, these formalisms are of very restricted expressiveness, in particular regarding first-order quantification, which is either supported only in a very limited fashion, or not at all.

In order to verify GOLOG programs without sacrificing expressiveness, Claßen and Lakemeyer (2008) (henceforth CL) developed a technique which is inspired by classical CTL model checking yet deals directly with the Situation Calculus and GOLOG. They begin by introducing the logic $\mathcal{ESG}$, an extension of the modal Situation Calculus variant $\mathcal{ES}$ (Lakemeyer and Levesque 2010) to specify temporal properties of GOLOG programs. They then provide algorithms for the verification of a first-order subset of the logic that resembles CTL. The method relies on regression-based reasoning and a newly introduced graph representation of GOLOG programs, which then allows for a systematic exploration of a program's configuration space within a fixpoint approximation. The downside of retaining full expressiveness is that termination is not guaranteed.

In this paper, we explore how much expressiveness needs to be sacrificed in order to guarantee termination and hence decidability of the verification process. In particular, we

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

show that decidability can indeed be achieved as follows:

1. We restrict the base logic to a two-variable fragment of $\mathcal{ES}$ and use a corresponding variant of Reiter's regression operator (Gu and Soutchanski 2010). Thus, the basic task of projection (needed among other things for checking whether the fixpoint computation loop has converged) becomes decidable, while the remaining expressiveness is sufficient to subsume (most) description logics.
2. Since (as we will see) this is not sufficient to guarantee the termination of the overall method, we furthermore restrict successor state axioms in the agent's action theory to be of special forms, namely either *context-free* (Lin and Reiter 1997) or *local-effect* (Liu and Levesque 2005).
3. Finally, we require that a finitary variant of the pick operator π (non-deterministic choice of argument) is used in GOLOG programs such that all actions are ground.

The above identifies a non-trivial fragment of CL's original formalism that goes beyond simple finite-state transition systems as used in classical model checking. In particular, both the agent's action theory as well as the GOLOG program may still contain first-order quantification over infinite domains. We complement this result by showing that all of the above restrictions are indeed necessary in the sense that no strict subset of them suffices to guarantee decidability.

The rest of this paper is organized as follows. First, we recapitulate the logic $\mathcal{ESG}$ and the corresponding verification method. Afterwards, we present our decidability results for the above mentioned classes of action theories. Next, we show undecidability when dropping any one of our three restrictions. We then review related work before we conclude.

The Logic $\mathcal{ESG}$

This section briefly recapitulates the logic $\mathcal{ESG}$, which extends $\mathcal{ES}$ by constructs for expressing temporal properties of GOLOG program executions. For a more detailed presentation please refer to CL and (Claßen 2013).

Syntax

The language is a first-order modal dialect with equality and sorts of type *object* and *action*. For each sort, it includes countably infinitely many standard names which are syntactically treated as constants, but which are assumed to be isomorphic to the (fixed) domain of discourse. Also included are both fluent and rigid predicate as well as rigid function symbols. Fluents vary as the result of actions, but rigids do not. We assume that the fluents include the unary predicate $Poss$ whose argument is of type action and which will be used to specify when an action is executable.

In addition to the usual connectives $\wedge, \neg, \forall$ (treating $\vee, \supset, \subset, \equiv$, and $\exists$ as abbreviations), formulas may contain modal subformulas of the form $\Box\alpha$ ("α holds after any number of actions"), $[t]\alpha$ ("α holds after doing action t"), and $[\![\delta]\!]\varphi$ ("temporal property φ holds for all runs of program δ").

The latter is the interesting, novel part about $\mathcal{ESG}$. In a subformula of the form $[\![\delta]\!]\varphi$ (where we will use $\langle\!\langle\delta\rangle\!\rangle\varphi$ to abbreviate $\neg[\![\delta]\!]\neg\varphi$), the δ argument has to be a GOLOG program, and φ a temporal property to be satisfied by its runs. Programs are constructed as follows:

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

That is we allow primitive actions t (where t can be any action term), tests $\alpha?$ (where α is a static formula as defined below), sequence, nondeterministic branching, nondeterministic choice of argument, concurrency, and nondeterministic iteration. Moreover, conditionals, while loops and infinite loops can be defined in terms of the above:

$$\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \stackrel{def}{=} [\phi?;\delta_1] \mid [\neg\phi?;\delta_2] \quad (2)$$

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

$$\textbf{loop}: \; \delta \stackrel{def}{=} \textbf{while } \top \textbf{ do } \delta \textbf{ endWhile} \quad (4)$$

For brevity we sometimes use δ^ω instead of **loop** : δ.

Temporal properties φ are constructed using the connectives $\wedge$, $\neg$ and $\forall$, together with subformulas containing temporal operators $\boldsymbol{X}\phi$ ("ϕ holds in the next situation") and $\phi \; \boldsymbol{U} \; \psi$ ("ϕ holds until ψ holds"). We also define $\boldsymbol{F}\phi = (\top \; \boldsymbol{U} \; \phi)$ ("eventually ϕ") and $\boldsymbol{G}\phi = \neg\boldsymbol{F}\neg\phi$ ("always ϕ").

We call a formula *fluent* when it contains no $[\cdot]$, no $\Box$, and no $[\![\cdot]\!]$ operators, nor the special fluent $Poss$. A formula is called *static* if it contains no $[\cdot]$, no $\Box$ and no $[\![\cdot]\!]$ operators. It is *bounded* when it contains no $\Box$ and no $[\![\cdot]\!]$ operators.

Semantics

Terms and formulas are interpreted wrt a semantic model called a *world*: Let $\mathcal{P}_O$ and $\mathcal{P}_A$ be the set of primitive terms of sort object, and action, respectively. A *primitive term* is of the form $f(n_1,\ldots,n_k)$, where all the n_i are standard names. Similarly, $\mathcal{P}_F$ is the set of all *primitive formulas* of the form $F(n_1,\ldots,n_k)$ for predicates F. Moreover, let $\mathcal{N}_O$ and $\mathcal{N}_A$ be the sets of all standard names of sort object and action, respectively, $\mathcal{N} = \mathcal{N}_O \cup \mathcal{N}_A$, and $\mathcal{Z} = \mathcal{N}_A{}^*$ the set of all finite sequences of action names, including the empty sequence $\langle\rangle$. A world w then maps primitive terms to co-referring standard names of the corresponding sort, and primitive formulas to truth values:

- $w: \mathcal{P}_O \times \mathcal{Z} \to \mathcal{N}_O$ and
- $w: \mathcal{P}_A \times \mathcal{Z} \to \mathcal{N}_A$ and
- $w: \mathcal{P}_F \times \mathcal{Z} \to \{0,1\}$

Worlds are also required to respect *rigidity*, i.e. if R is a rigid function or predicate, then for all $z, z' \in \mathcal{Z}$, $w[R(\vec{n}), z] = w[R(\vec{n}), z']$, as well as *unique action names*, i.e. if $g(\vec{n})$ and $g'(\vec{n}')$ are distinct primitive action terms, then for all $z \in \mathcal{Z}$, $w[g(\vec{n}), z] \neq w[g'(\vec{n}'), z]$. $\mathcal{W}$ denotes the set of all worlds.

Given a ground term t, a world w, and an action sequence $z \in \mathcal{Z}$, $|t|_w^z$ (read: "the co-referring standard name for t given w and z") is given by recursive evaluation: If $t \in \mathcal{N}$, then $|t|_w^z = t$; if $t = f(t_1,\ldots,t_k)$, then $|t|_w^z = w[f(n_1,\ldots,n_k), z]$, where $n_i = |t_i|_w^z$.

To interpret programs, we need the notion of *program configurations*. A configuration $\langle z, \delta\rangle$ consists of an action sequence z and a program δ, where intuitively z is the history of actions that have already been performed, while δ is the program that remains to be executed. A *transition relation* $\stackrel{w}{\longrightarrow}$ among configurations wrt a world w is then given

by a set of inductive rules, e.g. in case of primitive actions and nondeterministic branching (see CL for complete list):

1. $\langle z, t\rangle \xrightarrow{w} \langle z \cdot p, \top?\rangle$, if $p = |t|^z_w$;
   …
4. $\langle z, \delta_1|\delta_2\rangle \xrightarrow{w} \langle z \cdot p, \delta'\rangle$,
   if $\langle z, \delta_1\rangle \xrightarrow{w} \langle z \cdot p, \delta'\rangle$ or $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot p, \delta'\rangle$;
   …

A *trace* is given by an infinite sequence $\tau \in \mathcal{N}_A{}^\omega$. Let $\tau^{\leq i}$ stand for the finite sequence that consists of the first i elements of τ and $\tau^{>i}$ for the remaining ones. Given a world w and a finite sequence of $action$ standard names z, $\|\delta\|^z_w$ denotes the traces induced by the program δ:

$$\{\tau \mid \langle z, \delta\rangle \xrightarrow{w} \langle z \cdot \tau^{\leq 1}, \delta_1\rangle \xrightarrow{w} \langle z \cdot \tau^{\leq 2}, \delta_2\rangle \xrightarrow{w} \cdots\}$$

Note that since transitions depend on worlds (of which there are infinitely many) the state space is generally infinite.

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. $w, z \models F(t_1, \ldots, t_k)$ iff $w[F(|t_1|^z_w, \ldots, |t_k|^z_w), z] = 1$;
2. $w, z \models (t_1 = t_2)$ iff $|t_1|^z_w$ and $|t_2|^z_w$ 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 of same sort as x;
6. $w, z \models \Box\alpha$ iff $w, z \cdot z' \models \alpha$ for all $z' \in \mathcal{Z}$;
7. $w, z \models [t]\alpha$ iff $w, z \cdot |t|^z_w \models \alpha$;
8. $w, z \models [\![\delta]\!]\varphi$ iff for all $\tau \in \|\delta\|^z_w,\ w, z, \tau \models \varphi$.

Temporal properties φ are interpreted wrt some $w \in \mathcal{W}$, some $z \in \mathcal{Z}$, and a trace τ:

1. $w, z, \tau \models \alpha$ iff $w, z \models \alpha$, if α is a non-temporal formula;
2. $w, z, \tau \models \phi \wedge \psi$ iff $w, z, \tau \models \phi$ and $w, z, \tau \models \psi$;
3. $w, z, \tau \models \neg\phi$ iff $w, z, \tau \not\models \phi$;
4. $w, z, \tau \models \forall x.\phi$ iff $w, z, \tau \models \phi^x_n$ for all n of the right sort;
5. $w, z, \tau \models \boldsymbol{X}\phi$ iff $\tau = p \cdot \tau'$ and $w, z \cdot p, \tau' \models \phi$;
6. $w, z, \tau \models \phi\ \boldsymbol{U}\ \psi$ iff for some $i \geq 0$, $w, z \cdot \tau^{\leq i}, \tau^{>i} \models \psi$ and for all $j < i$, $w, z \cdot \tau^{\leq j}, \tau^{>j} \models \phi$.

Basic Action Theories and Regression

We use a *basic action theory* (BAT) to define the pre- and postconditions of primitive actions that can occur in a program. Formally, a BAT is given by $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, where

1. Σ_0, *the initial database*, is a finite set of fluent sentences describing the initial state of the world.
2. Σ_{pre} is a *precondition axiom* of the form $\Box Poss(a) \equiv \pi$, where π is a fluent formula whose only free variable is a.
3. Σ_{post} is a finite set of *successor state axioms* (SSAs) $\Box[a]F(\vec{x}) \equiv \gamma^+_F \vee F(\vec{x}) \wedge \neg\gamma^-_F$, one for each fluent F relevant to the application domain, where γ^+_F and γ^-_F are fluent formulas with free variables among $\vec{x}$ and a. SSAs incorporate Reiter's (1991) solution to the frame problem.

The verification method relies on the $\mathcal{ES}$ equivalent of Reiter's *regression* operator $\mathcal{R}[\alpha]$. The rough idea is that, whenever we encounter a subformula of the form $[t]F(\vec{x})$ within α, where t is an action term, we may substitute it by the right-hand side of the SSA for fluent F. This is sound in the sense that the axiom defines the two expressions to be equivalent. The result of the substitution will be true in exactly the same worlds satisfying Σ as the original one, but contains one less modal operator $[t]$. Similarly, $Poss(t)$ is replaced by the right-hand side of the precondition axiom. By iteratively applying such substitutions, we eventually get a fluent formula that describes exactly the conditions on the initial situation under which the original, non-static formula holds:

Theorem 1. *Let Σ be a BAT and α a bounded sentence. Then $\mathcal{R}[\alpha]$, the regression of α, is a fluent sentence and $\Sigma \models \alpha$ iff $\Sigma_0 \models \mathcal{R}[\alpha]$.*

Proof. This theorem follows directly from an analogous result for $\mathcal{ES}$ (Lakemeyer and Levesque 2010) due to the fact that, by definition, neither a BAT Σ nor the bounded sentence α contains any of the extensions of $\mathcal{ESG}$, hence all involved formulas are in $\mathcal{ES}$. □

Verification in $\mathcal{ESG}$

CL consider a CTL-like fragment of $\mathcal{ESG}$ whose formulas may contain subformulas of the form $\langle\langle\delta\rangle\rangle\boldsymbol{X}\psi$, $\langle\langle\delta\rangle\rangle\boldsymbol{G}\phi$, and $\langle\langle\delta\rangle\rangle\phi\ \boldsymbol{U}\ \psi$, where δ is a non-terminating program of the form $\delta_1{}^\omega\| \cdots \|\delta_k{}^\omega$. The idea behind their verification algorithm is to recursively substitute these subformulas by fluent formulas that are equivalent wrt a given BAT Σ. For each of these three types of subformulas, they provide a verification procedure. Due to lack of space, we only consider the one for the "always" operator, depicted below.[1]

Procedure 1 CHECKEG$[\delta, \phi]$

1: $L' := $ LABEL$[\mathcal{G}_\delta, \bot]$; $L := $ LABEL$[\mathcal{G}_\delta, \phi]$;
2: **while** $L \not\equiv L'$ **do**
3: $L' := L$; $L := L'$ AND PRE$[\mathcal{G}_\delta, L']$;
4: **end while**
5: **return** INITLABEL$[\mathcal{G}_\delta, L]$

The procedure operates on a so-called *characteristic graph* $\mathcal{G}_\delta = \langle V, E, v_0\rangle$ for the given program δ to encode the space of reachable program configurations. The nodes V in such a graph are of the form $\langle\delta', \phi\rangle$, denoting the remaining program of a current run and the condition under which execution may terminate there. v_0 is the initial node. Edges in E are labeled with tuples $\pi\vec{x} : t/\psi$, where $\vec{x}$ is a list of variables (if it is empty, we omit the leading π), t is an action term and ψ is a formula (which we omit when it is $\top$). Intuitively, this means when one wants to take action t, one has to choose instantiations for the $\vec{x}$ and ψ must hold. Again, the interested reader is referred to CL and (Claßen 2013) for the formal definition. An example is presented in Figure 1,

[1]Note however that all results of this paper equally hold for the other temporal modalities.

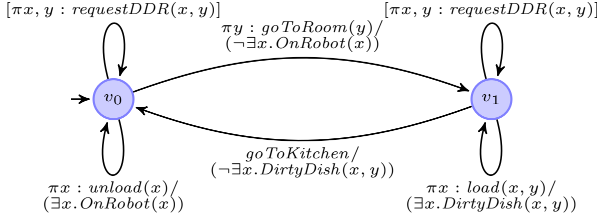

Figure 1: Characteristic graph for the robot example

which shows the graph corresponding to $\delta_{robot} \| \delta_{exo}$, where δ_{robot} denotes the control program presented in the introduction and $\delta_{exo} = (\pi x, y : requestDDR(x, y))^\omega$ encodes exogenous actions (here: requests for the removal of dirty dish x from room y). The nodes are $v_0 = \langle \delta_{robot} \| \delta_{exo}, \bot \rangle$ and $v_1 = \langle (\delta_1; \delta_{robot}) \| \delta_{exo}, \bot \rangle$, where δ_1 is the program

$$(\pi x. (\exists x DirtyDish(x, y))?;\ load(x, y))^*;$$
$$(\neg\exists x DirtyDish(x, y))?;\ goToKitchen.$$

The verification algorithms work on *labels* of the characteristic graph, where a label is given by $\langle v, \psi \rangle$ with $v \in V$ and ψ being a fluent formula. Intuitively, it represents all program configurations corresponding to v as well as all worlds w and action name sequences z satisfying ψ. A *labelling* is then given by a set of labels, one for each node of the graph. The following operations on labels are used:

$$\textsc{Label}[\langle V, E, v_0 \rangle, \alpha] \overset{def}{=} \{\langle v, \alpha \rangle \mid v \in V\}$$

$$L_1 \textsc{ And } L_2 \overset{def}{=} \{\langle v, \psi_1 \wedge \psi_2 \rangle \mid \langle v, \psi_1 \rangle \in L_1, \langle v, \psi_2 \rangle \in L_2\}$$

$$L_1 \equiv L_2 \overset{def}{\text{iff}} \text{ for } \langle v, \psi_1 \rangle \in L_1 \text{ and } \langle v, \psi_2 \rangle \in L_2,\ \models \psi_1 \equiv \psi_2$$

$$\textsc{InitLabel}[\langle V, E, v_0 \rangle, L] \overset{def}{=} \psi \text{ such that } \langle v_0, \psi \rangle \in L$$

$$\textsc{Pre}[\langle V, E, v_0 \rangle, L] \overset{def}{=} \{\langle v, \textsc{Pre}[v, L] \rangle \mid v \in V\}$$

where $\textsc{Pre}[v, L]$ stands for

$$\bigvee \{\mathcal{R}[\exists \vec{x}.\phi \wedge [t]\psi] \mid v \xrightarrow{\pi\vec{x}:t/\phi} v' \in E,\ \langle v', \psi \rangle \in L\}.$$

Procedure 1 thus proceeds as follows. First, the labelling L is initialized to every node being labelled with ϕ. As long as L is not equivalent to the old label L' (initialized to $\bot$), L is conjoined with its pre-image, which roughly gives us a description of the corresponding predecessor configurations (note the use of regression to eliminate the action term t). Once the label set is converged, the formula at the initial node is returned as result. The algorithm is sound as follows:

Theorem 2 (CL). *Let Σ be a BAT, δ a program and ϕ a fluent formula. If the procedure terminates,* $\textsc{CheckEG}[\delta, \phi]$ *is a fluent formula and $\Sigma \models \langle\langle \delta \rangle\rangle \boldsymbol{G}\phi$ iff $\Sigma_0 \models$* $\textsc{CheckEG}[\delta, \phi]$.

Decidability

The algorithm presented above cannot be guaranteed to terminate for two reasons. First, equivalence checks over first-order formulas as used in the while loop condition are in general undecidable. Furthermore, even if all equivalence checks terminate, the label set may never converge.

A Decidable Base Logic

As for the first source of non-termination, we can exploit results by Gu and Soutchanski (2010) who present a two-variable fragment of the Situation Calculus for which the projection problem, solved by means of regression, is decidable. Here we capture this fragment as a subset of $\mathcal{ESG}$ and $\mathcal{ES}$ called $\mathcal{ES}^2$, which restricts formulas to only contain

- variables x, y or rigid constant symbols as object terms;
- action function symbols with at most two arguments;
- fluents with at most two arguments.

Decidability is then essentially obtained from the fact that the result of applying the regression operator corresponds to a formula in FO^2, the two-variable fragment of first-order logic, which is known to be decidable (Mortimer 1975; Grädel, Kolaitis, and Vardi 1997).

We define an $\mathcal{ES}^2$ formula to be *regressable* if it is bounded and all its actions terms are ground. Then we have:

Theorem 3. *$\mathcal{ES}^2$ is closed under regression: If α is a regressable $\mathcal{ES}^2$ formula, then $\mathcal{R}[\alpha]$ is equivalent to a fluent $\mathcal{ES}^2$ formula.*

Proof. (Sketch) This can be achieved by modifying the regression operator $\mathcal{R}$ such that by means of appropriate substitutions, no new variable is introduced in the process of regression. The construction is similar to Definition 2 in (Gu and Soutchanski 2010). □

Theorem 4. *Let Σ_0 be a finite set of fluent $\mathcal{ES}^2$ formulas and ψ a single fluent $\mathcal{ES}^2$ formula. Then $\Sigma_0 \models \psi$ is decidable.*

Proof. (Sketch) Using a similar mapping as Lakemeyer and Levesque (2010) who embed $\mathcal{ES}$ in the original Situation Calculus, it is possible to show that truth of fluent $\mathcal{ES}^2$ formulas can be reduced to truth of FO^2 formulas. □

In the following, we assume that all fluent subformulas occurring in the BAT Σ, the program δ and the property φ are in $\mathcal{ES}^2$. By Theorem 3, labels thus will only contain fluent $\mathcal{ES}^2$ formulas. Theorem 4 then guarantees that both label equivalence checks $\models \psi_1 \equiv \psi_2$ as well as the final test for $\Sigma_0 \models \textsc{CheckEG}[\delta, \phi]$ are decidable.

Note that even though the restriction to only two variable symbols prohibits to express certain properties such as transitivity, it is nonetheless not as limited as it may initially seem due to the fact that variables may be reused. It is thus for instance possible to express the existence of a path of length k over some binary relation R for any given $k \geq 0$. For example, for the case $k = 3$ we have:

$$\exists x \exists y\, (R(x, y) \wedge \exists x\, (R(y, x) \wedge \exists y\, R(x, y)))$$

Convergence with Context-Free BATs

As we will see in the following section, using a decidable base logic is unfortunately not sufficient to also eliminate the second source of non-termination of the algorithm as the verification problem still remains undecidable. Convergence can however be guaranteed if we restrict ourselves to certain subclasses of BATs. The first possibility is to use the

ones with *context-free* SSAs (Lin and Reiter 1997), which formally means that effect conditions γ_F^+ and γ_F^- contain no fluents (but maybe rigids). A BAT is context-free if each SSA is context-free.

An example for an SSA that is context-free is the following, where $Fragile(x)$ is assumed to be rigid:

$$\begin{aligned}\Box[a]Broken(x) \equiv a = drop(x) \wedge Fragile(x)\\ \vee Broken(x) \wedge a \neq repair(x)\end{aligned}$$

On the other hand, if $HavePaint(x)$ is a fluent, then the following is not context-free:

$$\begin{aligned}\Box[a]Colour(x,y) \equiv a = paint(x,y) \wedge HavePaint(y)\\ \vee Colour(x,y) \wedge \neg\exists y.a = paint(x,y)\end{aligned}$$

Among other things, context-free SSAs also include the *strictly context-free* ones as special case, which correspond precisely to STRIPS planning operators as shown by Lin and Reiter (1997).

In order to ensure our prerequisite that formulas to be regressed only contain ground terms, we prohibit the usage of the non-deterministic pick operator π. Note that we can still allow a finitary variant defined in terms of nondeterministic branching over finitely many constants:

$$\pi x : \{c_1, \ldots, c_k\}.\delta \stackrel{def}{=} \delta^x_{c_1} | \cdots | \delta^x_{c_k}.$$

We then have the following theorem:

Theorem 5. *If Σ is a context-free BAT and δ a program without pick operators, Procedure 1 will converge.*

Proof. (Sketch) The central property for this proof is:

$$\mathcal{R}\big[[t_i]\mathcal{R}\big[[t_n]\ldots[t_i]\ldots[t_1]\varphi\big]\big] \equiv \mathcal{R}\big[[t_n]\ldots[t_i]\ldots[t_1]\varphi\big].$$

That is, regressing some fluent formula φ through the same ground action multiple times produces an equivalent result as only regressing once through that action. The reason is that in each step we conjoin and disjoin the corresponding ${\gamma^+_F}^a_{t_i}$ and ${\gamma^-_F}^a_{t_i}$, which will then remain unchanged by further regression since they are situation-independent formulas, and conjoining or disjoining them again hence has no effect. Because the program (and thus the characteristic graph) has only finitely many actions all of which are ground, there are only finitely many such sequences of actions to consider. Similarly, there are only finitely many fluent subformulas φ to which this applies, namely those given by the input property as well as the finitely many test conditions appearing in the program. We then exploit the fact that the body of the while loop in the verification procedure is monotone, i.e. it always produces a label formula subsumed by the previous one. Hence, eventually the label set converges. □

Note that convergence here and below is shown independent from the assumption of using $\mathcal{ES}^2$ as base logic. In fact, we could also "plug in" any other decidable fragment of first-order logic, such as C^2, the two-variable fragment with counting quantifiers (Pacholski, Szwast, and Tendera 2000).

Convergence with Local-Effect BATs

An alternative to ensure termination is to restrict ourselves to BATs whose SSAs are *local-effect* (Liu and Levesque 2005), where we formally require that both γ_F^+ and γ_F^- are disjunctions of formulas 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 $\vec{y}$. ϕ is called a *context formula* and contains no quantifiers. A BAT is local-effect if each SSA is local-effect.

It can be argued that the restriction to local-effect BATs is not that harsh. There are of course examples which cannot be expressed through a local-effect SSA, such as exploding a bomb:

$$\begin{aligned}\Box[a]Dead(x) \equiv \exists y\,(a = explode(y) \wedge Close(x,y))\\ \vee Dead(x)\end{aligned}$$

However, in many practical scenarios, actions only affect fluent values within a limited scope. The following is a local-effect SSA because both $lock(x,y)$ and $unlock(x,y)$ contain the fluent's argument x as parameter:

$$\begin{aligned}\Box[a]Locked(x) \equiv \exists y\,(a = lock(x,y) \wedge HaveKey(y))\\ \vee Locked(x) \wedge \neg\exists y\,(a = unlock(x,y) \wedge HaveKey(y))\end{aligned}$$

Indeed, all SSAs presented in this paper except the bomb example above are local-effect. Moreover, since local-effect SSAs are a generalization of strictly context-free ones, they also subsume everything that can be expressed by means of STRIPS.

Theorem 6. *If Σ is a local-effect BAT and δ a program without pick operators, Procedure 1 will converge.*

Proof. (Sketch) This proof relies on the fact that by using unique names of actions, the instantiation of a local-effect SSA on a ground action can be significantly simplified (Liu and Levesque 2005). Let $t = A(\vec{c})$ be a ground action. Then any ${\gamma^+_F}^a_t$ or ${\gamma^-_F}^a_t$ is equivalent to a formula of the form

$$\vec{x} = \vec{d}_1 \wedge \psi_1 \vee \cdots \vee \vec{x} = \vec{d}_n \wedge \psi_n$$

where $\vec{d}_i$ is a vector of constants contained in $\vec{c}$, and ψ_i is a quantifier-free sentence. We can then identify a finite number of equivalence classes for label formulas, as there are only finitely many actions in the graph (all of which are ground) and only finitely many fluents in the BAT, and hence only finitely many such instantiations for a finite number of relevant fluent formulas given by the test conditions in the program and the input property. Using the monotonicity argument again, convergence is guaranteed. □

An Example

Revisiting our earlier example, we show a verification run for a local-effect BAT. Observe that the program is already in the two-variable fragment, but we have to replace pick operators by their finitary counterparts, using constants d_i for dishes and r_i for rooms. This yields the program δ'_{robot}:

loop : **while** $(\exists x.OnRobot(x))$ **do**
$\pi x : \{d_1, d_2\}.unload(x)$ **endWhile**;
$\pi y : \{r_1, r_2\}.goToRoom(y)$;
while $(\exists x.DirtyDish(x,y))$ **do**
$\pi x : \{d_1, d_2\}.load(x,y)$ **endWhile**;
$goToKitchen$

Assume that all actions are always possible. The SSAs are (omitting the robot's location for simplicity):

$$\Box[a] \mathit{DirtyDish}(x,y) \equiv a = \mathit{requestDDR}(x,y) \vee \mathit{DirtyDish}(x,y) \wedge \neg[a = \mathit{load}(x,y)]$$

$$\Box[a] \mathit{OnRobot}(x) \equiv \exists y.\, a = \mathit{load}(x,y) \vee \mathit{OnRobot}(x) \wedge \neg[a = \mathit{unload}(x)].$$

Note that these axioms are in fact both context-free and local-effect.

Suppose we want to verify whether a run of $\delta'_{robot} \| \delta_{exo}$ is possible where some dish in some room remains dirty forever, i.e. $\neg\exists x,y.\langle\!\langle \delta'_{robot} \rangle\!\rangle \boldsymbol{G} \mathit{DirtyDish}(x,y)$. Thus, the call to Procedure 1 is CHECKEG$[\delta'_{robot} \| \delta_{exo}, \mathit{DirtyDish}(x,y)]$. It starts with the following label set:

$$L_0 = \{\langle v_0, \mathit{DirtyDish}(x,y)\rangle, \langle v_1, \mathit{DirtyDish}(x,y)\rangle\}.$$

For determining the pre-image for a node, each of its outgoing edges has to be considered. Note that for every edge of the original graph shown in Figure 1, there are now several edges corresponding to the instantiations of the variables by the d_i and r_j. One of the disjuncts of PRE$[v_0, L_0]$ thus is

$$\mathcal{R}[[\mathit{requestDDR}(d_1, r_1)] \mathit{DirtyDish}(x,y)]$$

which (using unique names of actions) reduces to

$$x = d_1 \wedge y = r_1 \vee \mathit{DirtyDish}(x,y).$$

Using similar reductions for the other edges we obtain PRE$[v_0, L_0]$ and PRE$[v_1, L_0]$ both being equivalent to

$$x = d_1 \wedge y = r_1 \vee x = d_2 \wedge y = r_1 \vee x = d_1 \wedge y = r_2 \vee x = d_2 \wedge y = r_2 \vee \mathit{DirtyDish}(x,y).$$

Then $L_1 = L_0$ AND PRE$[\mathcal{G}_\delta, L_0]$, which reduces to

$$\{\langle v_0, \mathit{DirtyDish}(x,y)\rangle, \langle v_1, \mathit{DirtyDish}(x,y)\rangle\}$$

hence $L_0 \equiv L_1$, i.e. the algorithm terminates and returns $\neg\exists x,y.\mathit{DirtyDish}(x,y)$. Thus, there is no run with some dish forever remaining dirty in some room iff there is no dirty dish initially. Intuitively, this is correct because $\boldsymbol{G}\phi$ means that ϕ persists to hold during the *entire* run, including the initial situation. Therefore, only if a dish is dirty initially it may happen that it never gets cleaned, namely when the robot never visits the corresponding room.

Undecidability

In this section we argue that all three restrictions, namely (1) fluent formulas being from a decidable base logic like $\mathcal{ES}^2$, (2) SSAs being either context-free or local-effect, and (3) disallowing pick operators, are necessary in the sense that no strict subset suffices to guarantee decidability.

Clearly, dropping restriction (1) immediately leads to undecidability as this would allow us to formulate arbitrary first-order sentences as tests and preconditions. Let us therefore consider the case of a restricted BAT Σ but a program δ with unlimited usage of the pick operator.

Theorem 7. *The verification problem for* GOLOG *programs over non-ground actions based on a context-free or local-effect BAT is undecidable.*

Proof. (Sketch) The proof is by reduction from the Halting problem of a Turing machine (TM). Given a TM $\mathcal{T}$ we devise a BAT $\Sigma_\mathcal{T}$ and a program $\delta_\mathcal{T}$ such that $\mathcal{T}$ halts just in case $\Sigma_\mathcal{T} \cup \{\langle\!\langle \delta_\mathcal{T} \rangle\!\rangle \boldsymbol{F} \mathit{State}_{q_F}\}$ is satisfiable. Assuming a right-infinite tape, we use the fluent $\mathit{Pos}(x)$ to denote the current head position, a fluent State_q for each state q, a fluent $\mathit{Symbol}_b(x)$ for each symbol b to express that cell x currently holds b, a fluent $\mathit{Visited}(x)$ to memorize already visited cells, and a fluent $\mathit{Right}(x)$ to memorize the rightmost cell that has been visited. Furthermore, the rigid predicate $\mathit{Adj}(x,y)$ is used to represent the adjacency relation of tape cells. For each transition rule (q, b, q', b', m) we then have an action $do_{(q,b,q',b',m)}(x,y)$, where x stands for the head position before applying the transition rule, and y for the resulting one. The definition of the initial theory, precondition axioms and (both context-free and local-effect) SSAs is straightforward. The program $\delta_\mathcal{T}$ then is an infinite loop that non-deterministically branches over finitely many subprograms, each corresponding to one transition rule. For example, a rule with movement to the right is encoded as

$$\pi x\, \pi y.\, \mathit{Pos}(x) \wedge \mathit{Adj}(x,y) \wedge (\mathit{Right}(x) \supset \neg \mathit{Visited}(y)) \wedge \mathit{State}_q \wedge \mathit{Symbol}_b(x)?;\, do_{(q,b,q',b',r)}(x,y).$$

The pick operators allow us to quantify objects from an unbounded domain, thus representing the unbounded tape of a TM. Since the Halting Problem for TMs is undecidable, so is the verification problem for GOLOG programs over non-ground actions and context-free or local-effect BATs. $\square$

If we disallow the pick operator again, but instead allow for non-local and non-context-free effects, we also get:

Theorem 8. *The verification for* GOLOG *programs over ground actions based on unrestricted BATs is undecidable.*

Proof. (Sketch) Again, this is shown by simulating a TM, using a similar construction as above. However, we now have a 0-ary action $do_{(q,b,q',b',m)}$ for each transition rule, and formalize the unrestricted quantification of new tape cells within SSAs. For example, the one for $\mathit{Pos}(x)$ is

$$\Box[a]\mathit{Pos}(x) \equiv \bigvee_{(q,b,q',b',m)} a = do_{(q,b,q',b',m)} \wedge \mathit{Next}_m(x),$$

where $\mathit{Next}_m(x)$ is an abbreviation that depends on the direction of movement, e.g. $\mathit{Next}_r(x)$ (for moving right) is

$$\exists y.\mathit{Pos}(y) \wedge \mathit{Adj}(y,x) \wedge (\mathit{Right}(y) \supset \neg \mathit{Visited}(x)).$$

Note that the above is neither context-free nor local-effect. Undecidability of the verification problem for GOLOG programs over ground actions and arbitrary BATs again follows from undecidability of the Halting Problem for TMs. $\square$

Related Work

Verification of non-terminating GOLOG programs was first discussed by De Giacomo, Ternovska and Reiter (1997), but only in the form of manual, meta-theoretic proofs, where properties were expressed using μ-calculus formulas instead of temporal modalities. De Giacomo, Lespérance and

Pearce (2010) applied the idea of verifying GOLOG programs through iterative fixpoint approximations using characteristic graphs within multi-agent scenarios, where properties are expressed in Alternating-Time Temporal Logic. Neither approach was concerned with decidability.

Baader, Liu and ul Mehdi (2010), on the other hand, address decidability by resorting to a dynamic extension (Baader et al. 2005) of the decidable description logic (DL) $\mathcal{ALC}$ (Baader et al. 2003) to represent pre- and postconditions of actions and consider properties expressed by a variant of LTL over $\mathcal{ALC}$ assertions (Baader, Ghilardi, and Lutz 2008). Instead of a fully-fledged GOLOG, they approximate programs by finite Büchi automata and show that under these restrictions, verification reduces to a decidable reasoning task within the underlying DL. Compared with our approach, the Büchi automaton representation loses one particularly important feature of GOLOG, namely the possibility to include test conditions in the form of formulas.

Baader, Liu and ul Mehdi's results were extended by Baader and Zarrieß (2013), who show decidability of verifying $\mathcal{ALC}$-LTL properties over GOLOG programs including test conditions. They make a similar restriction as we by prohibiting pick operators. As we have seen, there is good reason for this as arbitrary usage of picks quickly leads to undecidability. However, representing action effects in $\mathcal{ALC}$ only admits basic STRIPS-style addition and deletion of literals, which is less expressive than allowing context-free or local-effect SSAs as we do. Zarrieß and Claßen (2014) finally extend the above to a formalism that includes local-effect SSAs and is thus comparable in expressiveness to ours. In fact, their result can be viewed as complementing ours: While we apply restrictions to ensure termination of an algorithm for an originally undecidable problem, they construct a finite abstraction of the infinite-state transition system such that classical model checking can be used.

Another line of research on decidable verification is followed by De Giacomo, Lespérance and Patrizi (2012). They show decidability for first-order μ-calculus properties for a class of BATs that only admits 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, whereas our approach relies on purely syntactical restrictions. In related work, Hariri et al. (2013a; 2013b) consider the verification of μ-calculus properties in the context of relational databases and light-weight DLs, respectively, where new information may be added at any time. Among other things, they show that decidability obtains provided the added information is bounded.

Conclusion

Perhaps the main insight of the paper is that the decidability of the verification problem considered in this paper depends crucially on limiting both the basic action theory and the GOLOG program. Decidability obtains essentially because, under these restrictions, there exist finite abstractions of the infinite state space in terms of label formulas together with the respective program states. Moreover, giving up any one of these restrictions immediately leads to undecidability.

It is worth noting that the fragment we obtain under these restrictions is still non-trivial and of practical interest. Our base formalism is sufficiently expressive to subsume most description logics, in particular if we resort to C^2 instead of FO^2 as explained in an earlier remark. Moreover, context-free and local-effect SSAs both extend what can be expressed by basic STRIPS in a non-trivial manner, and thus cover a large fragment of plan operators in the planning language PDDL (Ghallab et al. 1998).

In the future, we plan to further investigate the computational complexity of decidable verification and provide an implementation. In this regard it may also be interesting to directly construct the finite state abstractions as in (Zarrieß and Claßen 2014) and then use a model checker such as NuSMV (Cimatti et al. 2002). It would also be interesting to consider other restrictions that would allow an arbitrary use of pick operators while maintaining decidability.

Acknowledgments

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

References

Baader, F., and Zarrieß, B. 2013. Verification of Golog programs over description logic actions. In *Proc. FroCoS'13*, 181–196. Springer-Verlag.

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

Baader, F.; Lutz, C.; Miličić, M.; Sattler, U.; and Wolter, F. 2005. Integrating description logics and action formalisms: First results. In *Proc. AAAI 2005*, 572–577. AAAI Press.

Baader, F.; Ghilardi, S.; and Lutz, C. 2008. LTL over description logic axioms. In *Proc. KR 2008*, 684–694. AAAI Press.

Baader, F.; Liu, H.; and ul Mehdi, A. 2010. Verifying properties of infinite sequences of description logic actions. In *Proc. ECAI 2010*, 53–58. IOS Press.

Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J. 1992. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2):142–170.

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

Burkart, O.; Caucal, D.; Moller, F.; and Steffen, B. 2001. Verification on infinite structures. In *Handbook of Process Algebra*. Elsevier. 545–623.

Cimatti, A.; Giunchiglia, E.; Pistore, M.; Roveri, M.; Sebastiani, R.; and Tacchella, A. 2002. Integrating BDD-based and SAT-based symbolic model checking. In *Proc. of the 4th Int. Workshop on Frontiers of Combining Systems*, FroCoS '02, 49–56. Springer-Verlag.

Clarke, E. M., and Emerson, E. A. 1981. Design and synthesis of synchronization skeletons using branching time logics. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*. Springer-Verlag. 52–71.

Clarke, E. M.; Emerson, E. A.; and Sistla, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8(2):244–263.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In *Proc. KR 2008*, 589–599. AAAI Press.

Claßen, J. 2013. *Planning and Verification in the Agent Language Golog*. Ph.D. Dissertation, Department of Computer Science, RWTH Aachen University.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2012. Bounded situation calculus action theories and decidable verification. In *Proc. KR 2012*. AAAI Press.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation calculus based programs for representing and reasoning about game structures. In *Proc. KR 2010*, 445–455. AAAI Press.

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

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems*.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. *PDDL—The Planning Domain Definition Language*.

Grädel, E.; Kolaitis, P. G.; and Vardi, M. Y. 1997. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic* 3(1):53–69.

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

Hariri, B. B.; Calvanese, D.; Giacomo, G. D.; Deutsch, A.; and Montali, M. 2013a. Verification of relational data-centric dynamic systems with external services. In *Proc. PODS 2013*, 163–174. ACM Press.

Hariri, B. B.; Calvanese, D.; Montali, M.; Giacomo, G. D.; Masellis, R. D.; and Felli, P. 2013b. Description logic knowledge and action bases. *J. Artif. Intell. Res. (JAIR)* 46:651–686.

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

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1–3):59–83.

Lin, F., and Reiter, R. 1997. How to progress a database. *Artificial Intelligence* 92(1–2):131–167.

Liu, Y., and Levesque, H. J. 2005. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proc. IJCAI 2005*, 522–527. Professional Book Center.

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

McMillan, K. L. 1993. *Symbolic Model Checking*. Kluwer Academic Publishers.

Mortimer, M. 1975. On languages with two variables. *Mathematical Logic Quaterly* 21:135–140.

Pacholski, L.; Szwast, W.; and Tendera, L. 2000. Complexity results for first-order two-variable logic with counting. *SIAM Journal on Computing* 29(4):1083–1117.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy* 359–380.

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

Zarrieß, B., and Claßen, J. 2014. On the decidability of verifying LTL properties of golog programs. In *Proceedings of the AAAI Spring Symposium 2014 on Knowledge Representation and Reasoning in Robotics (KRR'14)*. Palo Alto, California, USA: AAAI Press.