Decidable Verification of Golog Programs over Non-Local Effect Actions

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

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

Abstract

The Golog action programming language is a powerful means to express high-level behaviours in terms of programs over actions defined in a Situation Calculus theory. In particular for physical systems, verifying that the program satisfies certain desired temporal properties is often crucial, but undecidable in general, the latter being due to the language's high expressiveness in terms of first-order quantification, range of action effects, and program constructs. So far, approaches to achieve decidability involved restrictions where action effects either had to be *context-free* (i.e. not depend on the current state), *local* (i.e. only affect objects mentioned in the action's parameters), or at least *bounded* (i.e. only affect a finite number of objects). In this paper, we introduce two new, more general classes of action theories that allow for context-sensitive, non-local, unbounded effects, i.e. actions that may affect an unbounded number of possibly unnamed objects in a state-dependent fashion. We contribute to the further exploration of the boundary between decidability and undecidability for Golog, showing that for our new classes of action theories in the two-variable fragment of first-order logic, verification of CTL* properties of programs over ground actions is decidable.

Introduction

When it comes to the design and programming of an autonomous agent, the Golog (Levesque et al. 1997) family of action languages offers a powerful means to express high-level behaviours in terms of complex programs whose basic building blocks are the primitive actions described in a Situation Calculus (Reiter 2001) action theory. Golog's biggest advantage perhaps is the fact that a programmer can freely combine imperative control structures with non-deterministic constructs, leaving it to the system to resolve non-determinism in a suitable manner.

In particular when Golog is used to control physical robots, it is often crucial to verify a program against some specification of desired behaviour, for example in order to ensure liveness and safety properties, typically expressed by means of temporal formulas. Unfortunately, the general verification problem for Golog is undecidable due to the language's high expressivity in terms of first-order quantification, range of action effects, and program constructs. For this reason, there have recently been endeavours to identify restricted, but non-trivial fragments of Golog where verification (and hence other reasoning tasks such as projection) becomes decidable, while a great deal of expressiveness is retained.

So far, approaches to decidability (Claßen et al. 2014; Zarrieß and Claßen 2014; De Giacomo, Lespérance, and Patrizi 2012) required action theories to be restricted such that action effects are either *context-free* (not depend on the current state), *local* (only affect objects mentioned in the action's parameters), or at least *bounded* (only affect a finite number of objects). Examples that do *not* fall into either of these categories are the classical briefcase domain (Pednault 1988) and exploding a bomb (Lin and Reiter 1997): When a briefcase is moved, (unboundedly many, unmentioned) objects that are currently in it are being moved along, and if a bomb explodes, everything in its vicinity is destroyed.

In this paper, we extend the results from (Zarrieß and Claßen 2014) and present two new, more general classes of action theories over the decidable FOL fragment C^2 that also allow for context-sensitive, non-local, unbounded effects, i.e. actions that may affect an unbounded number of possibly unnamed objects in a state-dependent fashion. In our classes of action theories we do not impose any bound on the number of affected objects, but restrict the dependencies between fluents in the successor state axioms. This allows for a much wider range of application domains, including the above mentioned briefcase and bomb examples.

In a transportation domain such as the briefcase example, the action of moving a briefcase changes the location of objects represented by the fluent predicate At. To describe the actual set of objects affected one also has to refer to the fluent predicate In relating the briefcase to its content. Thus, the effect of the move action on At depends on In. The class of *acyclic theories* is obtained by disallowing cyclic dependencies between fluents, and another class we call *flat theories* is obtained by resorting to quantifier-free formulas for defining the set of affected objects. Both are syntactic restrictions and are decidable to check.

After proving that verification of CTL* properties is generally undecidable for Golog, even when restricted to ground actions and C^2, we then show that for our new classes of

action theories, decidability can be achieved. The proof introduces a new, compact form of regression of formulas and establishes an abstraction to propositional model checking. Due to space constraints all detailed proofs must be omitted. They can be found in the technical report (Zarrieß and Claßen 2015).

Preliminaries

The Logic $\mathcal{ES}$

We use a fragment of the first-order modal logic $\mathcal{ES}$ (Lakemeyer and Levesque 2010) for reasoning about actions. We consider Situation Calculus *Basic Action Theories* (BATs) (Reiter 2001) formulated in $\mathcal{ES}$ where the base logic is restricted to the *two-variable fragment of FOL with equality and counting* named C^2.

Syntax There are *terms* of sort *object* and *action*. Variables of sort object are denoted by symbols $x, y, \ldots$, and a denotes a variable of sort action. N_O is a countably infinite set of *object constant symbols* and N_A a countably infinite set of *action function symbols* with arguments of sort object. 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 $\mathcal{N}_A$.

Formulas are built using *fluent* predicate symbols (predicates that may vary as the result of actions) with at most two arguments of sort object, and equality, using the usual logical connectives, quantifiers, and counting quantifiers. In addition we have two modalities for referring to future situations, where $\Box\phi$ says that ϕ holds after any sequence of actions, and $[t]\phi$ means that ϕ holds after executing action t.

A formula is called *fluent formula* if it contains no $\Box$ and no $[\cdot]$. A *fluent sentence* is a fluent formula without free variables. A C^2*-fluent formula* is a fluent formula that contains no terms of sort action and at most two variables.

Semantics In the Situation Calculus a situation is characterized by a finite sequence of actions as the history of actions that have been executed so far. Let $\mathcal{Z} := \mathcal{N}_A^*$ be the set of all finite action sequences (including the empty sequence $\langle\rangle$) and $\mathcal{P}_F$ the set of all *primitive formulas* $F(n_1, ..., n_k)$, where F is a k-ary fluent with $0 \leq k \leq 2$ and the n_i are object standard names. A *world* w is a mapping from primitive formulas and situations to truth values:

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

The set of all worlds is denoted by $\mathcal{W}$.

Definition 1 (truth of formulas)**.** Given a world $w \in \mathcal{W}$ and a closed formula ψ, we define $w \models \psi$ as $w, \langle\rangle \models \psi$, 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. $w, z \models \psi_1 \wedge \psi_2$ iff $w, z \models \psi_1$ and $w, z \models \psi_2$;
4. $w, z \models \neg\psi$ iff $w, z \not\models \psi$;
5. $w, z \models \forall x.\phi$ iff $w, z \models \phi_n^x$ for all $n \in \mathcal{N}_x$;
6. $w, z \models \exists^{\leq m} x.\phi$ iff $|\{n \in \mathcal{N}_x \mid w, z \models \phi_n^x\}| \leq m$;
7. $w, z \models \exists^{\geq m} x.\phi$ iff $|\{n \in \mathcal{N}_x \mid w, z \models \phi_n^x\}| \geq m$;
8. $w, z \models \Box\psi$ iff $w, z \cdot z' \models \psi$ for all $z' \in \mathcal{Z}$;
9. $w, z \models [t]\psi$ iff $w, z \cdot t \models \psi$. ▲

Above, $\mathcal{N}_x$ refers to the set of all standard names of the same sort as x. We moreover use ϕ_n^x to denote the result of simultaneously replacing all free occurrences of x in ϕ by n. Note that by rule 2 above, the unique names assumption for actions and object constants is part of our semantics. In the following we use the notation $\vec{x}$ and $\vec{y}$ for sequences of object variables and $\vec{v}$ for a sequence of object terms. We understand $\vee$, $\exists$, $\supset$, $\equiv$ and $\top$ and $\bot$ as the usual abbreviations.

Definition 2. A C^2*-basic action theory* (C^2-BAT) $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{post}}$ is a set of axioms that describes the dynamics of a specific application domain, where

1. $\mathcal{D}_0$, *the initial theory*, is a finite set of C^2-fluent sentences describing the initial state of the world;
2. $\mathcal{D}_{\text{post}}$ is a finite set of *successor state axioms* (SSAs), one for each fluent relevant to the application domain, incorporating Reiter's (2001) 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

$$\forall a.\forall\vec{x}.\Box\big([a]F(\vec{x}) \equiv \gamma_F^+ \vee \big(F(\vec{x}) \wedge \neg\gamma_F^-\big)\big)$$

where the *positive effect condition* γ_F^+ and *negative effect condition* γ_F^- are fluent formulas. We additionally require that γ_F^+ and γ_F^- are (possibly empty) disjunctions of formulas of the form $\exists\vec{y}.\big(a = A(\vec{v}) \wedge \phi \wedge \phi'\big)$ such that

   (a) $\exists\vec{y}.\big(a = A(\vec{v}) \wedge \phi \wedge \phi'\big)$ contains the free variables $\vec{x}$ and a and no other free variables;
   (b) $A(\vec{v})$ is an action term and $\vec{v}$ contains $\vec{y}$;
   (c) ϕ is a fluent formula with no terms of sort action and the number of variable symbols in ϕ that do not occur in $\vec{v}$ or occur bound in ϕ is less or equal two;
   (d) ϕ' is a fluent formula with free variables among $\vec{v}$, no terms of sort action, and at most two bound variables.

   ϕ is called *effect descriptor* and ϕ' *context condition*. ▲

The restrictions 2a and 2b on SSAs are w.l.o.g. and describe the usual syntactic form of SSAs. Intuitively, the effect descriptor ϕ possibly defines a complex set of objects (or a set of pairs of objects in case F is a binary fluent) that are added to or deleted from the relational fuent F, respectively, if $A(\vec{v})$ is executed. Provided that free occurrences of variables in ϕ that occur as arguments of $A(\vec{v})$ are instantiated, the condition 2c ensures definability of the (instantiated) effect descriptor in our base logic C^2. In contrast to the effect descriptor the context condition ϕ' only tells us *whether* $A(\vec{v})$ has an effect on F but *not which* objects are actually affected. As for the effect descriptor, condition 2d ensures that after instantiation of the action, the context condition is a sentence in C^2. Therefore the variables $\vec{x}$ mentioned in 2a may have free occurrences in ϕ but not in ϕ'.

Example 3. We consider a domain with *servers* hosting *virtual machines* and *processes* that might be classified as *malware*. There is a fluent $Avail(x)$ denoting processes x that are currently available, and $Ovl(x)$ for a server x that is overloaded. $Hosts(x, y)$ furthermore says that a server x

hosts a virtual machine or a process y, and $Runs(x, y)$ is true for a virtual machine x running a process y.

The agent can migrate a virtual machine (v) hosted on server (s) to a server (s') if s' is not overloaded using the action $Migr(v, s, s')$. We also have exogenous actions, i.e. actions not under the control of the agent, of the form $Att(s)$, saying that a server is subject of an attack causing it to be overloaded, and $Repair(s)$, which returns the server s to its original state. Figure 2 exemplarily shows the effect conditions for the fluents $Avail(x)$, $Ovl(x)$ and $Hosts(x, y)$. The effect descriptors are underlined with a solid line and the context conditions with a dashed line. Consider the execution of $Migr(vm, s_1, s_2)$ in an initial situation incompletely described by the axioms in Figure 1. The action has an effect on the fluent $Avail(x)$ because the context condition is satisfied, i.e. the target server s_2 is not overloaded. The instantiated effect descriptor yields that for all objects d, $Avail(d)$ is *true after* doing the action if $Runs(vm, d)$ is *true before* doing the action. Thus, all processes running on vm become available. Furthermore, the fluent $Hosts(x, y)$ is also affected: all processes running on vm are now hosted by s_2 and no longer by s_1. A BAT based on these axioms for example entails

$$[Migr(vm, s_1, s_2)]\big(\forall x.Runs(vm, x) \supset Avail(x)\big). \quad \blacktriangle$$

$$Hosts(s_1, vm), Hosts(s_1, p), Runs(vm, p), \neg Avail(p)$$
$$Server(s_2), \neg Ovl(s_2), \forall y.\exists^{\leq 1} x.Hosts(x, y),$$
$$\forall x, y.Hosts(x, y) \supset Server(x) \wedge \big(Proc(y) \vee VM(y)\big)$$

Figure 1: Example initial theory

$$\begin{aligned}
\gamma^{+}_{Avail} &:= \exists v, s, s'.\big(a = Migr(v, s, s') \wedge \\
&\qquad \underline{Runs(v, x)} \wedge \underline{\neg Ovl(s')}\big) \vee \\
&\quad \exists s.\big(a = Repair(s) \wedge \underline{Hosts(s, x) \wedge Proc(x)}\big); \\
\gamma^{-}_{Avail} &:= \exists s.\big(a = Att(s) \wedge \underline{Hosts(s, x) \wedge Proc(x)} \wedge \\
&\qquad \underline{\exists y.Hosts(s, y) \wedge Malware(y)}\big); \\
\gamma^{+}_{Ovl} &:= \exists s.\big(a = Att(s) \wedge \underline{x = s} \wedge \\
&\qquad \underline{\exists y.Hosts(s, y) \wedge Malware(y)}\big); \\
\gamma^{-}_{Ovl} &:= \exists s.\big(a = Repair(s) \wedge \underline{x = s}\big); \\
\gamma^{+}_{Hosts} &:= \exists v, s, s'.\big(a = Migr(v, s, s') \wedge \underline{x = s'} \wedge \\
&\qquad \big(\underline{Runs(v, y)} \vee y = v\big) \wedge \underline{\neg Ovl(s')}\big); \\
\gamma^{-}_{Hosts} &:= \exists v, s, s'.\big(a = Migr(v, s, s') \wedge \underline{x = s} \wedge \\
&\qquad \big(\underline{Runs(v, y)} \vee y = v\big) \wedge \underline{\neg Ovl(s')}\big)
\end{aligned}$$

Figure 2: Example effect conditions

Golog programs and the verification problem

In a Golog program over ground actions we combine actions, whose effects are defined in a C^2-BAT, and tests, using a set of programming constructs to define a complex action.

Definition 4 (Golog program)**.** A *program expression* δ is built according to the following grammar

$$\delta ::= \langle\rangle \mid t \mid \psi? \mid \delta;\delta \mid \delta|\delta \mid \delta^* \mid \delta\|\delta.$$

A program expression can thus be the *empty program* $\langle\rangle$, a ground action term t, a *test* $\psi?$, where ψ is a C^2-fluent sentence, or constructed from subprograms be means of *sequence* $\delta;\delta$, *non-deterministic choice* $\delta|\delta$, *non-deterministic iteration* δ^*, and *interleaving* $\delta\|\delta$.

A *Golog program* $\mathcal{G} = (\mathcal{D}, \delta)$ consists of a C^2-BAT $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{post}}$ and a program expression δ where all fluents occurring in $\mathcal{D}$ and δ have an SSA in $\mathcal{D}_{\text{post}}$.

To handle termination and failure of a program we use two 0-ary fluents $Final$ and $Fail$ and two 0-ary action functions ϵ and $\mathfrak{f}$ and include the SSAs $\Box[a]Final \equiv a = \epsilon \vee Final$ and $\Box[a]Fail \equiv a = \mathfrak{f} \vee Fail$ in $\mathcal{D}_{\text{post}}$. Furthermore, we require that $\neg Final \in \mathcal{D}_0$ and $\neg Fail \in \mathcal{D}_0$, and that the fluents $Final$, $Fail$ and actions ϵ and $\mathfrak{f}$ do not occur in δ. $\blacktriangle$

Following (Claßen and Lakemeyer 2008) we define the transition semantics of programs meta-theoretically. A *configuration* $\langle z, \rho\rangle$ consists of an action sequence $z \in \mathcal{Z}$ and a program expression ρ, where intuitively z is the history of actions that have already been performed, while ρ is the program that remains to be executed. Execution of a program in a world $w \in \mathcal{W}$ yields a *transition relation* $\xrightarrow{w}$ *among configurations* that is defined inductively over program expressions. For example, for primitive actions and nondeterministic choice we have

1. $\langle z, t\rangle \xrightarrow{w} \langle z \cdot t, \langle\rangle\rangle$;

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 $\langle z, \delta_2\rangle \xrightarrow{w} \langle z \cdot t, \delta'\rangle$.

For the set of final configurations $\mathsf{Fin}(w)$ w.r.t. a world w, we similarly have for tests and nondeterministic choice

2. $\langle z, \psi?\rangle \in \mathsf{Fin}(w)$ if $w, z \models \psi$;

3. $\langle z, \delta_1|\delta_2\rangle \in \mathsf{Fin}(w)$
   if $\langle z, \delta_1\rangle \in \mathsf{Fin}(w)$ or $\langle z, \delta_2\rangle \in \mathsf{Fin}(w)$.

We omit the remaining rules due to space restrictions and refer the interested reader to (Claßen and Lakemeyer 2008). Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a Golog program and $w \in \mathcal{W}$ a world with $w \models \mathcal{D}$. Execution of δ in w yields the *transition system of* $\mathcal{G}$ *w.r.t.* w given by $\mathsf{T}^w_\delta = \big(\mathsf{Reach}(w, \delta), \xrightarrow{w}\big)$, where $\mathsf{Reach}(w, \delta)$ denotes the set of reachable configurations from $\langle\langle\rangle, \delta\rangle$ using $\xrightarrow{w}$. In addition, final and failing configurations are extended to infinite paths by executing ϵ and $\mathfrak{f}$, respectively, indefinitely.

For an infinite path π in T^w_δ starting in $\langle z_0, \rho_0\rangle$ we denote for any $j \geq 0$ the suffix $\langle z_j, \rho_j\rangle \xrightarrow{w} \langle z_{j+1}, \rho_{j+1}\rangle \xrightarrow{w} \cdots$ by $\pi[j..]$. The *set of all paths* starting in $\langle z, \rho\rangle$ is denoted by $\mathsf{Paths}(\langle z, \rho\rangle, \mathsf{T}^w_\delta)$.

Definition 5 (temporal properties of programs)**.** We define *temporal formulas*, whose syntax is the same as for propositional CTL*, but in place of propositions we allow for C^2-fluent sentences:

$$\Phi ::= \psi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathsf{E}\Psi \tag{1}$$
$$\Psi ::= \Phi \mid \neg\Psi \mid \Psi \wedge \Psi \mid \mathsf{X}\Psi \mid \Psi\,\mathsf{U}\,\Psi \tag{2}$$

Above, ψ can be any C^2-fluent sentence. We call formulas according to (1) *temporal state formulas*, and formulas according to (2) *temporal path formulas*. We use the usual abbreviations $\mathsf{A}\Psi$ (Ψ holds on *all* paths) for $\neg\mathsf{E}\neg\Psi$, $\mathsf{F}\Psi$ (*eventually* Ψ) for $\top\ \mathsf{U}\ \Psi$ and $\mathsf{G}\Psi$ (*globally* Ψ) for $\neg\mathsf{F}\neg\Psi$.

Let Φ be a temporal state formula, T_δ^w the transition system of a program $\mathcal{G} = (\mathcal{D}, \delta)$ w.r.t. a world w with $w \models \mathcal{D}$, and $\langle z, \rho\rangle \in \mathsf{Reach}(w, \delta)$. Truth of Φ in $\mathsf{T}_\delta^w, \langle z, \rho\rangle$, denoted by $\mathsf{T}_\delta^w, \langle z, \rho\rangle \models \Phi$, is defined as follows:

- $\mathsf{T}_\delta^w, \langle z, \rho\rangle \models \psi$ iff $w, z \models \psi$;
- $\mathsf{T}_\delta^w, \langle z, \rho\rangle \models \mathsf{E}\Psi$
  iff there is $\pi \in \mathsf{Paths}(\langle z, \rho\rangle, \mathsf{T}_\delta^w)$ such that $\mathsf{T}_\delta^w, \pi \models \Psi$.

Let Ψ be a temporal path formula, T_δ^w and $\langle z, \rho\rangle$ as above, and $\pi \in \mathsf{Paths}(\langle z, \rho\rangle, \mathsf{T}_\delta^w)$. Truth of Ψ in T_δ^w, π, denoted by $\mathsf{T}_\delta^w, \pi \models \Psi$, is defined as follows:

- $\mathsf{T}_\delta^w, \pi \models \Phi$ iff $\mathsf{T}_\delta^w, \langle z, \rho\rangle \models \Phi$;
- $\mathsf{T}_\delta^w, \pi \models \mathsf{X}\Psi$ iff $\mathsf{T}_\delta^w, \pi[1..] \models \Psi$;
- $\mathsf{T}_\delta^w, \pi \models \Psi_1\ \mathsf{U}\ \Psi_2$ iff $\exists k \geq 0 : \mathsf{T}_\delta^w, \pi[k..] \models \Psi_2$
  and $\forall j, 0 \leq j < k : \mathsf{T}_\delta^w, \pi[j..] \models \Psi_1$.

In both cases, Boolean connectors are defined as usual. ▲

Note that we *disallow* temporal modalities within the scope of object quantifiers which is a quite common restriction.

Definition 6 (verification problem)**.** A temporal state formula Φ is *valid* in a program $\mathcal{G} = (\mathcal{D}, \delta)$ iff for all worlds $w \in \mathcal{W}$ with $w \models \mathcal{D}$ it holds that $\mathsf{T}_\delta^w, \langle\langle\rangle, \delta\rangle \models \Phi$. ▲

Example 7. Consider the program expressions in Figure 3. In δ_{avail} the virtual machine vm is migrated from server s_1 to server s_2 if s_1 hosts vm and is overloaded and vice versa if s_2 is overloaded. δ_{exo} consists of the exogenous attack and repair actions. To describe the actions that occur in the domain, both parts δ_{avail} and δ_{exo} are concurrently executed in infinite loops. A temporal property one might want to verify for the Golog program consisting of the C^2-BAT described in Example 3 and the program expression δ_{domain} could be:

$$\mathsf{A}\big(\mathsf{GF}\big(Ovl(s_1) \wedge Ovl(s_2)\big)\big) \supset$$
$$\mathsf{E}\big(\mathsf{GF}\,\forall x.Runs(vm, x) \supset Avail(x)\big).$$

Validity of this property ensures that it is always possible that all processes running on vm are infinitely often available even if both servers are both infinitely often overloaded. ▲

$$\begin{aligned}
\delta_{avail} := \ & \exists x.(Hosts(x, vm) \wedge Ovl(x))?; \\
& \big(Hosts(s_1, vm)?; Migr(vm, s_1, s_2) \mid \\
& \ Hosts(s_2, vm)?; Migr(vm, s_2, s_1)\big) \\
\delta_{exo} := \ & \big(Att(s_1) \mid Att(s_2) \mid Repair(s_1) \mid Repair(s_2)\big) \\
\delta_{domain} := \ & \big[\big(\delta_{avail}\big)^*; \bot?\big] \parallel \big[\big(\delta_{exo}\big)^*; \bot?\big]
\end{aligned}$$

Figure 3: Example program

(Un-)decidability of Verification

As shown in (Gu and Soutchanski 2007), the projection problem that asks for a sequence of ground actions over some C^2-BAT whether a given C^2-fluent sentence holds after executing that sequence, is decidable. Unfortunately, verification for programs over ground actions is not:

Theorem 8. *The verification problem is undecidable.*

Proof sketch. We show undecidability by a reduction of the undecidable halting problem of two-counter machines (Minsky 1967). A two-counter machine M manipulates the non-negative integer values of two counters, in the following denoted by c_0 and c_1. A machine M is given by a finite list of instructions. There are instructions for incrementing and decrementing a counter by one, for conditional jumps to the next instruction where the condition is a zero test of a counter, and for halting the machine.

A *configuration* of M is of the form (i, v_0, v_1), where i is the index of the instruction to be executed next, and $v_0, v_1 \in \mathbb{N}$ are the values of the two counters. M induces a transition relation on configurations, denoted by $\vdash_{\mathsf{M}}$. We say that M halts iff there exists a computation such that $(0, 0, 0) \vdash_{\mathsf{M}}^* (j, v_0, v_1)$ where 0 is the index of the first instruction, $v_0, v_1 \in \mathbb{N}$ and the j-th instruction is a halting instruction. To encode the values of counters we axiomatize an infinite chain of objects starting in an object constant $\mathbf{0} \in \mathcal{N}_O$ using the binary predicate Adj. For the counters we use two unary fluents C_0 and C_1. We ensure that in each situation $C_\ell(n)$ is true for exactly one object n in this chain. Intuitively, the distance of n from $\mathbf{0}$ in the Adj-chain represents the value of the counter c_ℓ. The initial theory for a program simulating M consists of the following axioms where $Halt$ is a 0-ary fluent saying whether M is in a halting configuration and the other 0-ary fluents J_i serve as labels pointing to the instruction to be executed next:

$$\begin{aligned}
& \forall x.\big(x = \mathbf{0} \equiv C_0(x)\big), \forall x.\big(x = \mathbf{0} \equiv C_1(x)\big), \\
& \neg Halt, J_0, \neg J_1, \ldots, \neg J_m, \forall x.\exists^{=1} y.Adj(x, y), \\
& \forall x.\big(x \neq \mathbf{0} \supset \exists^{=1} y.Adj(y, x)\big), \forall x.\neg Adj(x, \mathbf{0}).
\end{aligned}$$

We use ground actions $Inc_0, Inc_1, Dec_0, Dec_1$ for incrementing and decrementing a counter. The effect conditions for the fluents $C_\ell(x)$ with $\ell = 0, 1$ are given as follows:

$$\begin{aligned}
\gamma^+_{C_\ell} := \ & a = Inc_\ell \wedge \exists y.\big(C_\ell(y) \wedge Adj(y, x)\big) \vee \\
& a = Dec_\ell \wedge \exists y.\big(C_\ell(y) \wedge Adj(x, y)\big) \\
\gamma^-_{C_\ell} := \ & a = Inc_\ell \wedge C_\ell(x) \vee a = Dec_\ell \wedge C_\ell(x).
\end{aligned}$$

Adj is rigid: its SSA is $\Box[a]Adj(x, y) \equiv Adj(x, y)$. The jumps to the next instruction can be implemented in the obvious way by actions setting the corresponding fluent J_j to true and all other labels to false. Similarly for $Halt$. It is now straightforward to assemble a program simulating M. It can then be shown that the temporal state formula $\mathsf{EF}Halt$ is valid in the constructed program iff M halts. □

Fluent dependencies and acyclic theories

To analyze the source of undecidability, we investigate dependencies between fluents occurring in the effect descriptors of the SSAs in the action theory.

Definition 9. Let $\mathcal{D}$ be a C^2-BAT. The *fluent dependency graph* for $\mathcal{D}$, denoted by $G_\mathcal{D}$, consists of a set of nodes, one for each fluent in $\mathcal{D}$. There is a directed edge (F, F') from fluent F to fluent F' iff there exists a disjunct $\exists \vec{y}.\big(a = A(\vec{v}) \wedge \phi \wedge \phi'\big)$ in γ_F^+ or γ_F^- such that F' occurs in the effect descriptor ϕ. We call $\mathcal{D}$ *acyclic* iff $G_\mathcal{D}$ is acyclic. The *fluent depth of an acyclic action theory* $\mathcal{D}$, denoted by $\mathsf{fd}(\mathcal{D})$, is given by the length of the longest path in $G_\mathcal{D}$. For a fluent F in an acyclic BAT $\mathcal{D}$ the *fluent depth of F w.r.t. $\mathcal{D}$*, denoted by $\mathsf{fd}_\mathcal{D}(F)$, is given by the length of the longest path in $G_\mathcal{D}$ starting in F. ▲

Example 10. First, consider the BAT in the undecidability proof. Obviously, the dependency graph is cyclic as there are edges (C_0, C_0) and (C_1, C_1).

On the other hand, the BAT from Example 3 has an acyclic dependency graph (with fluent depth 2) as shown in Figure 4. Fluents Ovl, $Server$ and VM were omitted as they are not incident to any edges. Ovl for instance only occurs in the context conditions of γ^+_{Avail}, γ^+_{Hosts} and γ^-_{Hosts}, and $Hosts$ in the context condition of γ^+_{Ovl}. For the dependency graph however, only effect descriptors are relevant. For instance, there is an edge from $Avail$ to $Runs$ because $Runs$ occurs in the effect descriptor in conjunction with the migration action in γ^+_{Avail}, i.e. the migration of a virtual machine may affect the availability of all processes running on this machine. In an analogous way $Avail$ and $Hosts$, $Proc$ are related due to the effect descriptor of the repair action in γ^+_{Avail}. The other edges can be explained similarly. ▲

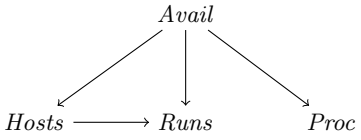

Figure 4: Example fluent dependencies

Note that if actions have only *local-effects* (Vassos, Lakemeyer, and Levesque 2008), then $\mathcal{D}$ is acyclic. In case of local-effect actions the effect descriptors do not contain any fluents. Consequently, the corresponding BAT has fluent depth 0. Another well-known special case are context-free actions (Lin and Reiter 1997) where the positive and negative effect conditions are restricted to contain only *rigid predicate symbols*. Clearly, BATs restricted in this way have at most fluent depth 1. The so called *solitary stratified theories* considered in (McIlraith 2000) are based on a similar acyclicity condition, but without distinguishing between effect descriptors and context conditions. The action theory in our example is therefore not a solitary stratified theory.

Decidability of verification with acyclic theories

In this section we restrict our attention to programs over ground actions with an acyclic C^2-BAT $\mathcal{D}$. Note that we only consider programs over ground actions and have dropped the *pick constructor* for non-deterministic choice of action arguments. The full pick construct introduces another source of infiniteness: an infinite branching degree in the transition system. The other non-deterministic constructs only lead to a finite branching degree (there are only finitely many ground actions), but of course the transition system still has infinitely many states due to non-local action effects, the infinite domain and the open-world assumption.

The finite set of ground actions (including ϵ and $\mathfrak{f}$) occurring in the program will be denoted by $\mathcal{A}$. We construct finite propositional abstractions of the transition systems T^w_δ with $w \models \mathcal{D}$. The essential part for this abstraction is a compact representation of the effects generated by executing a *sequence* of ground actions in a given world satisfying $\mathcal{D}$.

The case of local-effect actions was considered in (Zarrieß and Claßen 2014), where the idea was that the execution of a local-effect ground action $A(\vec{c})$ changes only the truth values of primitive formulas $F(\vec{n})$ for $\vec{n}$ that are arguments of the action. Thus, effects of an action could be captured by considering sets of literals built from fluent predicates and objects mentioned in the program. As ground actions from acyclic BATs may influence infinitely many fluent values, we have to extend this representation accordingly.

First we simplify SSAs w.r.t. the finitely many ground actions mentioned in $\mathcal{G}$. If $F(\vec{x})$ is a fluent and $t \in \mathcal{A}$, the *grounding* of the SSA of F w.r.t. t is of the form

$$\Box[t]F(\vec{x}) \equiv \big(\gamma_F^+\big)^a_t \vee F(\vec{x}) \wedge \neg\big(\gamma_F^-\big)^a_t.$$

The instantiated positive and negative effect conditions $\big(\gamma_F^+\big)^a_t$ and $\big(\gamma_F^-\big)^a_t$ then are each equivalent to a disjunction

$$\phi_1^{\mathsf{eff}} \wedge \phi_1^{\mathsf{con}} \vee \cdots \vee \phi_n^{\mathsf{eff}} \wedge \phi_n^{\mathsf{con}}$$

for some $n \geq 0$, where the ϕ_i^{eff} (effect descriptors) are C^2-fluent formulas with $\vec{x}$ as their only free variables, and the ϕ_i^{con} (context conditions) are C^2-fluent sentences. In the following we often view $\big(\gamma_F^+\big)^a_t$ and $\big(\gamma_F^-\big)^a_t$ as sets and for example write $(\phi_i^{\mathsf{eff}}, \phi_i^{\mathsf{con}}) \in \big(\gamma_F^+\big)^a_t$ to express that the corresponding disjunct is present.

With the above, we can now define a generalized *effect function* to represent the effects of a ground action:

Definition 11. Let $F(\vec{x})$ be a fluent and ϕ a C^2-fluent formula with free variables $\vec{x}$, where $\vec{x}$ is empty or $\vec{x} = x$ or $\vec{x} = (x, y)$. We call the expression $\langle F^+, \phi\rangle$ a *positive effect on* F, and the expression $\langle F^-, \phi\rangle$ a *negative effect on* F. We use the notation $\langle F^\pm, \phi\rangle$ for an effect if we do not explicitly distinguish between a positive or a negative effect on F. Let $\mathcal{D}$ be a C^2-BAT, w a world with $w \models \mathcal{D}$, $z \in \mathcal{Z}$ and $t \in \mathcal{A}$. The *effects of executing t in* (w, z) are defined as follows:

$$\mathcal{E}_\mathcal{D}(w, z, t) :=$$
$$\{\langle F^+, \phi^{\mathsf{eff}}\rangle \mid \exists(\phi^{\mathsf{eff}}, \phi^{\mathsf{con}}) \in \big(\gamma_F^+\big)^a_t \text{ s. t. } w, z \models \phi^{\mathsf{con}}\} \cup$$
$$\{\langle F^-, \phi^{\mathsf{eff}}\rangle \mid \exists(\phi^{\mathsf{eff}}, \phi^{\mathsf{con}}) \in \big(\gamma_F^-\big)^a_t \text{ s. t. } w, z \models \phi^{\mathsf{con}}\}.$$ ▲

Intuitively, if $\langle F^+, \phi\rangle \in \mathcal{E}_\mathcal{D}(w, z, t)$ and $\vec{c}$ is an instance of ϕ *before* executing t in w, z, then $F(\vec{c})$ will be true *after* the execution. Likewise, if $\langle F^-, \phi\rangle \in \mathcal{E}_\mathcal{D}(w, z, t)$ and $\vec{c}$ is an instance of ϕ *before* executing t in w, z, then $F(\vec{c})$ will be false *after* the execution. To accumulate the effects of consecutively executed actions we define a regression operator applied to a C^2-fluent formula given a set of effects. Wlog we assume that only the object variable symbols x and y are used in C^2-fluent formulas.

Definition 12. Let E be a set of effects and φ a C^2-fluent formula. The *regression of φ through* E, denoted by $\mathcal{R}[\mathsf{E}, \varphi]$, is a C^2-fluent formula obtained from φ by replacing each occurrence of a fluent $F(\vec{v})$ in φ by the formula

$$F(\vec{v}) \wedge \bigwedge_{\langle F^-, \phi\rangle \in \mathsf{E}} \neg\phi^{\vec{x}}_{\vec{v}} \vee \bigvee_{\langle F^+, \phi\rangle \in \mathsf{E}} \phi^{\vec{x}}_{\vec{v}}.$$

By appropriately renaming variables in the effect descriptors ϕ it can be ensured that $\mathcal{R}[\mathsf{E}, \varphi]$ is again a C^2-fluent sentence. ▲

Next, if we first execute the set of effects E_0, and afterwards E_1, the result is a combined set of effects $\mathsf{E}_0 \rhd \mathsf{E}_1$ given by:

$$\begin{aligned}
&\{\langle F^{\pm}, \mathcal{R}[\mathsf{E}_0, \varphi]\rangle \mid \langle F^{\pm}, \varphi\rangle \in \mathsf{E}_1\} \cup \\
&\{\langle F^{+}, \big(\varphi \wedge \bigwedge_{\langle F^-, \varphi'\rangle \in \mathsf{E}_1} \neg\mathcal{R}[\mathsf{E}_0, \varphi']\big)\rangle \mid \langle F^{+}, \varphi\rangle \in \mathsf{E}_0\} \cup \\
&\{\langle F^{-}, \varphi\rangle \in \mathsf{E}_0\}.
\end{aligned}$$

It can be shown that for any C^2-fluent sentence ϕ,

$$\mathcal{R}[\mathsf{E}_0, \mathcal{R}[\mathsf{E}_1, \phi]] \equiv \mathcal{R}[\mathsf{E}_0 \rhd \mathsf{E}_1, \phi].$$

We can therefore accumulate the effects of a sequence of actions into a single set as follows. Let w be a world with $w \models \mathcal{D}$, and $z = t_1 t_2 \cdots t_n \in \mathcal{A}^*$ a sequence of ground actions of length $n \in \mathbb{N}$. If for $i \leq n$, $z[i]$ denotes the subsequence of z consisting of the first i elements of z, we set

$$\begin{aligned}
\mathsf{E}_1 &:= \mathcal{E}_{\mathcal{D}}(w, \langle\rangle, t_1) \\
\mathsf{E}_i &:= \mathsf{E}_{i-1} \rhd \mathcal{E}_{\mathcal{D}}(w, z[i-1], t_i) \text{ for } i = 2, \ldots, n.
\end{aligned}$$

and say that E_n *is generated by executing $t_1 t_2 \cdots t_n$ in w*. Then, for the effects E_z generated by z in w and a C^2-fluent sentence ψ, it holds that

$$w, z \models \psi \text{ iff } w, \langle\rangle \models \mathcal{R}[\mathsf{E}_z, \psi].$$

For a given Golog program $\mathcal{G} = (\mathcal{D}, \delta)$ with an acyclic BAT $\mathcal{D}$ and finitely many ground actions $\mathcal{A}$ occurring in δ we show that there are only finitely many possible effects that can be generated by action sequences from $\mathcal{A}$. We observe that for an effect $\langle F^{\pm}, \varphi\rangle$ on fluent F with depth $\mathsf{fd}_{\mathcal{D}}(F) = i$ all fluents occurring in φ have a depth that is strictly smaller than i. Thus, for regressing the effect descriptor φ only effects on fluents with depth strictly smaller than i are relevant. Using this argument we can define the set of all relevant effects as follows: For a fluent F the set of all positive effect descriptors for F are given by

$$\mathsf{eff}^{+}_{\mathcal{A}}(F) := \{\phi^{\mathsf{eff}} \mid (\phi^{\mathsf{eff}}, \phi^{\mathsf{con}}) \in \big(\gamma^{+}_{F}\big)^{a}_{t} \text{ for some } t \in \mathcal{A}\},$$

and analogous for the negative effect descriptors $\mathsf{eff}^{-}_{\mathcal{A}}(F)$. For an acyclic BAT $\mathcal{D}$ and finite set of ground actions $\mathcal{A}$ the *set of all relevant effects* on all fluents with depth $\leq j$ with $j = 0, \ldots, \mathsf{fd}(\mathcal{D})$ is denoted by $\mathfrak{E}^{\mathcal{D},\mathcal{A}}_{j}$ and is given in Figure 5. We define $\mathfrak{E}^{\mathcal{D},\mathcal{A}} := \mathfrak{E}^{\mathcal{D},\mathcal{A}}_{n}$ with $\mathsf{fd}(\mathcal{D}) = n$. For a given fluent F with $\mathsf{fd}_{\mathcal{D}}(F) = 0$ it holds that either F is rigid, i.e. there are no effects on F, or there are only local effects on F. Consequently, all effects on F generated by a ground action sequence from $\mathcal{A}$ must be contained in $\mathfrak{E}^{\mathcal{D},\mathcal{A}}_{0}$.

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

Figure 5: Sets of all relevant effects with $1 \leq i \leq \mathsf{fd}(\mathcal{D})$

For fluents F with $\mathsf{fd}_{\mathcal{D}}(F) = i$ and $i > 0$ the fluents in the effect descriptors may also be subject to changes but have a depth strictly smaller than i. To obtain all relevant effects on F it is therefore sufficient to consider the effects in $\mathfrak{E}^{\mathcal{D},\mathcal{A}}_{i-1}$.

Lemma 13. *Let $\mathcal{D}$ and $\mathcal{A}$ be as above, $z \in \mathcal{A}^*$, $w \models \mathcal{D}$ and E_z the effects generated by executing z in w. For each $\langle F^{\pm}, \varphi\rangle \in \mathsf{E}_z$ there exists $\langle F^{\pm}, \varphi'\rangle \in \mathfrak{E}^{\mathcal{D},\mathcal{A}}$ with $\varphi \equiv \varphi'$.*

Using the finite representation of action effects we construct finite abstractions of the transition systems generated by executing the program in worlds satisfying an acyclic C^2-BAT. First, we identify a finite set of *relevant C^2-fluent sentences* called *context of a program*, denoted by $\mathcal{C}(\mathcal{G})$. It consists of all C^2-fluent sentences occurring in the initial theory, in context conditions in the instantiated SSAs, in tests in the program, and in the temporal property. Furthermore, the context is closed under negation.

Central for the abstraction is the notion of a *type of a world*, representing an equivalence class over $\mathcal{W}$. Intuitively, a type says which of the context axioms are satisfied initially and in all relevant future situations of that world.

Definition 14 (type of a world)**.** Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a Golog program with an acyclic BAT $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_{\text{post}}$ w.r.t. finite set of ground actions $\mathcal{A}$ (including ϵ and $\mathfrak{f}$). Furthermore, let $\mathcal{C}(\mathcal{G})$ be the context of $\mathcal{G}$ and $\mathfrak{E}^{\mathcal{D},\mathcal{A}}$ the set of all relevant effects. The *set of all type elements* is given by

$$\mathsf{TE}(\mathcal{G}) := \{(\psi, \mathsf{E}) \mid \psi \in \mathcal{C}(\mathcal{G}), \mathsf{E} \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}\}.$$

A *type w.r.t. $\mathcal{G}$* is a set $\tau \subseteq \mathsf{TE}(\mathcal{G})$ that satisfies:

1. For all $\psi \in \mathcal{C}(\mathcal{G})$ and all $\mathsf{E} \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}$ it holds that either $(\psi, \mathsf{E}) \in \tau$ or $(\neg\psi, \mathsf{E}) \in \tau$.
2. There exists a world $w \in \mathcal{W}$ such that

$$w \models \mathcal{D}_0 \cup \{\mathcal{R}[\mathsf{E}, \psi] \mid (\psi, \mathsf{E}) \in \tau\}.$$

The *set of all types w.r.t. $\mathcal{G}$* is denoted by $\mathsf{Types}(\mathcal{G})$. The *type of a world $w \in \mathcal{W}$ w.r.t. $\mathcal{G}$* is given by

$$\mathsf{type}(w) := \{(\psi, \mathsf{E}) \in \mathsf{TE}(\mathcal{G}) \mid w \models \mathcal{R}[\mathsf{E}, \psi]\}. \quad ▲$$

The abstraction of a world state consisting of a world $w \in \mathcal{W}$ with $w \models \mathcal{D}$ and an action sequence $z \in \mathcal{A}^*$ is then given by $\mathsf{type}(w)$ and the set of effects $\mathsf{E}_z \subseteq \mathfrak{E}^{\mathcal{D},\mathcal{A}}$ generated by executing z in w. Furthermore, there are only finitely many control states in the transition system of a program. To

capture the set of reachable subprograms we use a representation similar to the *characteristic program graphs* defined in (Claßen and Lakemeyer 2008). A lifting of the transition relation to the level of types then yields a finite abstraction of T_δ^w based on the abstraction $\mathsf{type}(w)$. Now the strong decoupling of the temporal part and the C^2 part in the temporal property comes in to play: To verify the abstraction of T_δ^w against a temporal state formula Φ over C^2-fluent sentences, we replace the axioms in Φ contained in the context $\mathcal{C}(\mathcal{G})$ by atomic proposition and then use propositional model checking. Since there are only finitely many world types that can be computed using a decidable consistency check in C^2, this approach yields a decision procedure for the verification problem.

Theorem 15. *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a program with an acyclic C^2-BAT and Φ a temporal state formula. It is decidable to verify whether Φ is valid in $\mathcal{G}$.*

The techniques introduced for acyclic theories can also be applied to programs with a C^2-BAT $\mathcal{D}$ where all the effect descriptors in the SSAs in $\mathcal{D}$ are quantifier-free but may contain cycles. (The domain in Example 3 satisfies also this restriction). We call this class *flat action theory*. It is straightforward to show that in this case only finitely many effects can be generated. We use the same arguments as for the acyclic case to show that a finite abstraction of the transition system can be constructed such that satisfaction of temporal properties is preserved.

Theorem 16. *Let $\mathcal{G} = (\mathcal{D}, \delta)$ be a program with a flat C^2-BAT and Φ a temporal state formula over axioms in $\mathcal{C}(\mathcal{G})$. It is decidable to verify whether Φ is valid in $\mathcal{G}$.*

Obviously, there is no trivial extension of the two (incomparable) decidable classes. As we have seen in the proof of Theorem 8, already a simple cycle within the scope of a quantifier causes undecidability again. See the technical report (Zarrieß and Claßen 2015) for detailed proofs.

Related Work

De Giacomo, Lespérance and Patrizi (2012) show decidability for first-order μ-calculus properties for a class of BATs where fluent extensions are bounded by some fixed threshold. Moreover, their notion of boundedness is a semantical condition that is in general undecidable to check, whereas our approach relies on purely syntactical restrictions. (Hariri et al. 2014) investigate acyclicity conditions that ensure *state-boundedness* in data-aware dynamic systems. State-boundedness then in turn allows for decidable verification by constructing finite abstraction of infinite transition systems. However, the setting is quite different: The transition systems in (Hariri et al. 2014) have a fixed database instance as initial state, actions do not respect the frame assumption but for example may cause an infinite branching degree.

Conclusion

In this paper we broadened the class of Golog programs and action theories for which decidability of verification can be achieved. The new class of acyclic theories subsumes many of the ones that were previously studied, including the context-free and local-effect ones and also the class considered in Theorem 16 subsumes local-effect theories. We observe that the decidability does not merely depend on whether actions may affect an unbounded number of objects, i.e. have non-local effects, but also on the dependencies between fluents in the action theory. Interestingly, it turns out that in domains as the one described in Example 3, in the *briefcase domain* (Pednault 1988), or in the *logistics domain* (Bacchus 2001), actions have non-local effects but dependencies are acyclic. Note that we refer to non-propositional models of the domains in the Situation Calculus, i.e. ones that admit a (possibly) infinite number of objects.

Acknowledgments This work was supported by the German Research Foundation (DFG) research unit FOR 1513 on Hybrid Reasoning for Intelligent Systems, project A1.

References

Bacchus, F. 2001. The AIPS '00 planning competition. *AI Magazine* 22(3):47–56.

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

Claßen, J.; Liebenberg, M.; Lakemeyer, G.; and Zarrieß, B. 2014. Exploring the boundaries of decidable verification of non-terminating Golog programs. In *Proc. of AAAI, 2014*.

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

Gu, Y., and Soutchanski, M. 2007. Decidable reasoning in a modified situation calculus. In *Proc. of IJCAI, 2007*.

Hariri, B. B.; Calvanese, D.; Montali, M.; and Deutsch, A. 2014. State-boundedness in data-aware dynamic systems. In *Proc. of KR, 2014*.

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.

McIlraith, S. A. 2000. Integrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *Artificial Intelligence* 116(1-2):87–121.

Minsky, M. L. 1967. *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Pednault, E. P. D. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4:356–372.

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

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

Zarrieß, B., and Claßen, J. 2014. Verifying CTL* properties of Golog programs over local-effect actions. In *Proc. of ECAI, 2014*.

Zarrieß, B., and Claßen, J. 2015. Decidable verification of golog programs over non-local effect actions. LTCS-Report 15–19, TU Dresden. See http://lat.inf.tu-dresden.de/research/reports.html.