Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Verification of Knowledge-Based Programs over Description Logic Actions

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

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

Abstract

A knowledge-based program defines the behavior of an agent by combining primitive actions, programming constructs and test conditions that make explicit reference to the agent's knowledge. In this paper we consider a setting where an agent is equipped with a Description Logic (DL) knowledge base providing general domain knowledge and an incomplete description of the initial situation. We introduce a corresponding new DL-based action language that allows for representing both physical and sensing actions, and that we then use to build knowledge-based programs with test conditions expressed in the epistemic DL. After proving undecidability for the general case, we then discuss a restricted fragment where verification becomes decidable. The provided proof is constructive and comes with an upper bound on the procedure's complexity.

1 Introduction

Since the GOLOG [Levesque *et al.*, 1997; De Giacomo *et al.*, 2000] family of action programming languages has become a popular means for control of high-level agents, the verification of temporal properties of GOLOG programs has recently received increasing attention [Claßen and Lakemeyer, 2008; De Giacomo *et al.*, 2010]. Both the GOLOG language itself and the underlying Situation Calculus [McCarthy and Hayes, 1969; Reiter, 2001a] are of high (first-order) expressivity, which renders the general problem undecidable. Identifying non-trivial fragments where decidability is given is therefore a worthwhile endeavour [De Giacomo *et al.*, 2012; Zarrieß and Claßen, 2014].

Here we consider the class of so-called *knowledge-based programs*, which are suited for more realistic scenarios where the agent possesses only incomplete information about its surroundings and has to use sensing in order to acquire additional knowledge at run-time. As opposed to classical GOLOG, knowledge-based programs contain explicit references to the agent's knowledge, thus enabling it to choose its course of action based on what it knows and does not know. Formalizations of knowledge-based programs were proposed by Reiter [2001b] and later by Claßen and Lakemeyer [2006] based on Scherl and Levesque's [2003] account of an epistemic Situation Calculus and Lakemeyer and Levesque's [2004; 2010] modal variant $\mathcal{ES}$, respectively. Common to these approaches is that conditions in the program are evaluated by reducing reasoning about both knowledge and action to standard first-order theorem proving.

In this paper, we propose a new epistemic action formalism based on the basic Description Logic (DL) $\mathcal{ALC}$ by combining and extending earlier proposals for DL action formalisms [Baader *et al.*, 2005] and epistemic DLs [Donini *et al.*, 1998]. From the latter we use a concept constructor for knowledge to formulate test conditions within programs and desired properties thereof, while we extend the former by not only including physical, but also sensing actions. As will become apparent, representing and verifying knowledge-based programs with this language yields multiple advantages. First, we obtain decidability of verification for a formalism whose expressiveness goes far beyond propositional logic. Moreover, it enables us to resort to powerful DL reasoning systems. Finally, the new formalism also inherits many useful properties of the epistemic Situation Calculus and $\mathcal{ES}$ such as Reiter's [1991] solution to the frame problem, a variant of Levesque's [1990] notion of only-knowing, and a reasoning mechanism resembling Levesque and Lakemeyer's [2001] Representation Theorem where reasoning about knowledge is reduced to standard DL reasoning.

As a motivating example, consider a mobile robot in a factory whose task it is to detect faults in gears and do the necessary repairs before turning them on. The agent is equipped with an (objective) DL knowledge base consisting of a TBox and an ABox, as usual. The TBox defines basic terminology such as the role name *has-f* that relates a system to its faults. The ABox gives an incomplete description of the initial situation and provides some properties of possible faults. Assume that the agent has two pure sensing actions at its disposal, namely $\texttt{sense-f}(\mathit{gear}, x)$ to sense whether the individual *gear* has fault x and $\texttt{sense-on}(\mathit{gear})$ to check if *gear* is on or not. Furthermore, the physical action $\texttt{repair}(\mathit{gear}, x)$ is available to remove a fault. An example for a knowledge-based program for this agent is given in Figure 1. As long as the agent does not know that *gear* has no known fault, a known fault x is chosen non-deterministically for which it is unknown whether *gear* has it or not. The agent then senses

while $\neg\mathbf{K}(\forall \textit{has-f}.\neg\mathbf{K}\textit{Fault})(\textit{gear})$
  pick(x) : $\mathbf{K}\textit{Fault}(x) \wedge \neg\mathbf{K}\textit{has-f}(\textit{gear}, x) \wedge \neg\mathbf{K}\neg\textit{has-f}(\textit{gear}, x)?$.
    sense-f$(\textit{gear}, x)$;
    if $\mathbf{K}\textit{has-f}(\textit{gear}, x)$ **then** repair$(\textit{gear}, x)$ **else continue**;
end
turn-on$(\textit{gear})$; sense-on$(\textit{gear})$;
if $\mathbf{K}\neg\textit{On}(\textit{gear})$ **then** raise-alarm **else continue**;

Figure 1: Example program

whether *gear* has this fault and repairs it if necessary. After completing the loop the agent turns on the gear system and checks if this was successful. If not, then there must be an unknown fault and an alarm is raised. An example for a property of this program to be verified is if a gear initially has an unknown critical fault, then the agent will eventually come to know it.

The remainder of this paper starts with recalling basic notion of DLs for representing initial knowledge, effect conditions of primitive actions, sensing properties, tests in programs and temporal properties of programs. In Section 3 we present our new action formalism that allows us to model both sensing and acting and consider the projection problem as a basic reasoning task. Section 4 then is about the verification of programs. We define syntax and semantics of the programming language and show how to specify temporal properties of programs. Afterwards, we discuss a restricted fragment where verification becomes decidable and that in some respect even goes beyond earlier work on non-epistemic programs [Zarrieß and Claßen, 2014], namely by re-introducing a limited variant of the operator for the nondeterministic choice of arguments ("pick operator"). We provide a constructive proof (along with an upper bound on the procedure's complexity) in which our variant of the Representation Theorem is used to build a finite abstraction of a program's transition system by means of DL reasoning, after which standard propositional model checking can be applied.

Due to space constraints all detailed proofs must be omitted. They can be found in the accompanying technical report [Zarrieß and Claßen, 2015].

2 The Epistemic Description Logic $\mathcal{ALCOK}$

Here we introduce the syntax and semantics of the epistemic DL $\mathcal{ALCOK}$ following [Donini *et al.*, 1998]. It extends the basic DL $\mathcal{ALC}$ by singleton concepts called nominals ($\mathcal{O}$), and by an epistemic role and concept constructor ($\mathcal{K}$). Although we sometimes only use the objective sublogics $\mathcal{ALC}$ and $\mathcal{ALCO}$, we present all basic notions here for full $\mathcal{ALCOK}$.

Let N_R, N_C, N_I be countably infinite sets of *role names*, *concept names* and *individual names*, respectively. A *generalized $\mathcal{ALCOK}$-role* P is built from role names using the *epistemic role constructor* and *role negation* (see Table 1). A *simple $\mathcal{ALCOK}$-role* (*role* for short) is a generalized role without role negation. *$\mathcal{ALCOK}$-concept descriptions* (*concepts* for short) are built from (simple) roles, concept names and the concept constructors shown in the lower part of Table 1 where $a \in N_I$. The following concept constructors are defined as abbreviations: $C \sqcup D := \neg(\neg C \sqcap \neg D)$, $\mathbf{Kw}C := \mathbf{K}C \sqcup \mathbf{K}\neg C$ and $\forall R.C := \neg\exists R.\neg C$. Intuitively, an object belongs to the concept $\mathbf{Kw}C$ if it is *known whether* it belongs to C or to $\neg C$. In the following we often use the symbols r for a role name, R for a role, P for a generalized role, A for a concept name, C, D for possibly complex concepts, and a, b for individual names (*individuals*).

Name	*Syntax*	*Semantics under* $(\mathcal{I}, \mathcal{W})$
role name	r	$r^{\mathcal{I}}$
role negation	$\neg P$	$(\Delta \times \Delta) \setminus P^{\mathcal{I},\mathcal{W}}$
epistemic role	$\mathbf{K}P$	$\bigcap_{\mathcal{J}\in\mathcal{W}} P^{\mathcal{J},\mathcal{W}}$
concept name	A	$A^{\mathcal{I}}$
top	$\top$	Δ
negation	$\neg C$	$\Delta \setminus C^{\mathcal{I},\mathcal{W}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I},\mathcal{W}} \cap D^{\mathcal{I},\mathcal{W}}$
exist. rest.	$\exists R.C$	$\{d \mid \exists e : (d,e) \in R^{\mathcal{I},\mathcal{W}}, e \in C^{\mathcal{I},\mathcal{W}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
epist. concept	$\mathbf{K}C$	$\bigcap_{\mathcal{J}\in\mathcal{W}}(C^{\mathcal{J},\mathcal{W}})$

Table 1: Syntax and semantics of roles and concepts

The different kinds of $\mathcal{ALCOK}$-axioms are shown in Table 2. A TBox $\mathcal{T}$ is a finite set of *concept inclusions* (*CIs* for short) and an ABox $\mathcal{A}$ is a finite set of *concept and role assertions* (also called *ABox assertions*), where assertions of the form $A(a), \neg A(a), r(a,b), \neg r(a,b)$ are called *literals*. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox and an ABox.

	Name	*Axiom* ϱ	$(\mathcal{I}, \mathcal{W}) \mid\!\models \varrho$, iff
TBox $\mathcal{T}$	concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I},\mathcal{W}} \subseteq D^{\mathcal{I},\mathcal{W}}$
ABox $\mathcal{A}$	concept assertion	$C(a)$	$a \in C^{\mathcal{I},\mathcal{W}}$
	role assertion	$P(a,b)$	$(a,b) \in P^{\mathcal{I},\mathcal{W}}$

Table 2: Syntax and semantics of axioms

$\mathcal{ALCOK}$ uses a *possible world semantics* defined in terms of interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty domain $\Delta^{\mathcal{I}}$ and a mapping $\cdot^{\mathcal{I}}$ with $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for all $A \in N_C$, $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for all $r \in N_R$ and $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for all $a \in N_I$. Here we adopt the so called *standard name assumption* (SNA): all interpretations are defined over a fixed countably infinite domain of standard names, denoted by Δ, with a fixed interpretation of individuals: We define $\Delta := N_I$ and we assume that for any interpretation $\mathcal{I}$, $\Delta^{\mathcal{I}} = \Delta$ and $a^{\mathcal{I}} = a$ for all $a \in N_I$. An *epistemic interpretation* is a pair $(\mathcal{I}, \mathcal{W})$ consisting of an interpretation $\mathcal{I}$ and a set of interpretations $\mathcal{W}$. The interpretation function $\cdot^{\mathcal{I},\mathcal{W}}$ maps concepts and roles to subsets of Δ and $\Delta \times \Delta$, respectively, as given in Table 1. *Satisfaction of an axiom* ϱ in $(\mathcal{I}, \mathcal{W})$, denoted by $(\mathcal{I}, \mathcal{W}) \mid\!\models \varrho$, is defined as given in Table 2, and translates to ABoxes, TBoxes and KBs in the obvious way.

An *epistemic model* of a KB $\mathcal{K}$ is a non-empty set of interpretations $\mathcal{M}$ such that for all $\mathcal{I} \in \mathcal{M}$ it holds that

$(\mathcal{I},\mathcal{M}) \mid\!\models \mathcal{K}$ and for all sets of interpretations $\mathcal{M}'$ with $\mathcal{M} \subsetneq \mathcal{M}'$ there exists $\mathcal{J} \in \mathcal{M}'$ such that $(\mathcal{J},\mathcal{M}') \not\mid\!\models \mathcal{K}$.

An axiom ϱ is *epistemically entailed* by $\mathcal{K}$, written as $\mathcal{K} \mid\!\models \varrho$, iff for all epistemic models $\mathcal{M}$ of $\mathcal{K}$ and for all $\mathcal{I} \in \mathcal{M}$ it holds that $(\mathcal{I},\mathcal{M}) \mid\!\models \varrho$. $\mathcal{K}$ is called *consistent* if $\mathcal{K}$ has an epistemic model.

A concept, axiom, TBox, ABox or KB without any occurrences of $\mathbf{K}$ is called *objective*, and *subjective* if all occurring concept and role names occur within the scope of a $\mathbf{K}$.

In case of an objective concept C we sometimes write $C^{\mathcal{I}}$ instead of $C^{\mathcal{I},\mathcal{W}}$ and $\mathcal{I} \models X$ instead of $(\mathcal{I},\mathcal{W}) \mid\!\models X$ for an objective axiom, KB, TBox or ABox X. Likewise, for an objective KB $\mathcal{K}$ and objective axiom ϱ we write $\mathcal{K} \models \varrho$ instead of $\mathcal{K} \mid\!\models \varrho$. We also sometimes omit $\mathcal{I}$ if we deal with subjective concepts, roles and axioms.

For an objective KB $\mathcal{K}$ there exists a unique epistemic model. This unique epistemic model of $\mathcal{K}$ is given by $\mathcal{M}(\mathcal{K}) = \{\mathcal{I} \mid \mathcal{I} \models \mathcal{K}\}$.

3 Actions with Sensing

We introduce a simple notion of *primitive actions* describing the basic abilities of an agent to change the world and to gain new information from the environment. Primitive actions are defined as syntactical objects composed of (parametrized) ABox assertions representing effects, effect conditions and properties of the environment that can be sensed, and are equipped with a purely model-theoretic semantics.

To define the syntax of actions, let N_V be a countably infinite set of variables. An *atom* is an ABox assertion where in place of individuals also variables are allowed, i.e. atoms are of the form $C(z)$ or $P(z,z')$ with $z,z' \in N_V \cup N_I$. *Primitive atoms* are of the form $A(z)$, $\neg A(z)$, $r(z,z')$ or $\neg r(z,z')$. A *formula* is a boolean combination of atoms. The set of all variables occurring in a formula ψ is denoted by $\mathsf{Var}(\psi)$. A formula without variables is called *ground formula*. Subjectivity and objectivity of formulas is defined in the obvious way. We will use the symbols φ for atoms, γ for primitive atoms and ψ for formulas.

Definition 1. An *effect* is either of the form ψ/γ (*conditional effect*) or of the form γ (*unconditional effect*), where ψ is an objective formula called *effect condition* and γ a primitive atom. A *primitive action* α is a pair $\alpha : (\mathsf{eff},\mathsf{sense})$ where eff is a finite set of effects and sense a finite set of objective formulas. We often write $\alpha(x_1,\ldots,x_n)$ where $x_1,\ldots,x_n$ are the variables occurring in α. An action without variables is called *ground action*.

$$\texttt{turn-on}(x) : (\mathsf{eff} = \{(\neg\exists \textit{has-f.CritFault}(x))/\textit{On}(x)\})$$
$$\texttt{sense-on}(x) : (\mathsf{sense} = \{\textit{On}(x)\}).$$

Figure 2: Example actions

Figure 2 shows an example of a pure physical action and a pure sensing action where the sets sense and eff, respectively, are omitted. $\texttt{turn-on}(x)$ has a single conditional effect that causes x to be *On* after the action is executed only if x previously has no critical fault. Note that here effects and their conditions are restricted to be objective since eff is supposed to only encode physical effects. $\texttt{sense-on}(x)$ is a sensing action that represents the agent's ability to perceive whether $\textit{On}(x)$ is true in the real world. Again, formulas in sense are objective since sensors only provide information about the outside world.

Semantically, a primitive action induces a binary relation on epistemic interpretations $(\mathcal{I},\mathcal{W})$ which allows us to explicitly distinguish changes affecting the real world represented by $\mathcal{I}$ and changes to the knowledge state $\mathcal{W}$.

To execute an action we first need to instantiate it. A *variable mapping* ν is a total function of the form $\nu : N_V \to \Delta$. We write ψ^ν to denote the ground formula that is obtained from the formula ψ by replacing each variable $x \in \mathsf{Var}(\psi)$ by $\nu(x)$. Similarly, given a primitive action $\alpha : (\mathsf{eff},\mathsf{sense})$, $\alpha^\nu : (\mathsf{eff}^\nu,\mathsf{sense}^\nu)$ denotes the corresponding ground action.

Next we define how a single interpretation $\mathcal{I}$ is affected by a set of literals L. The *update of $\mathcal{I}$ with* L is an interpretation $\mathcal{I}^{\mathsf{L}}$ that is defined as follows:

$$A^{\mathcal{I}^{\mathsf{L}}} := (A^{\mathcal{I}} \setminus \{a \mid \neg A(a) \in \mathsf{L}\}) \cup \{a \mid A(a) \in \mathsf{L}\};$$
$$r^{\mathcal{I}^{\mathsf{L}}} := (r^{\mathcal{I}} \setminus \{(a,b) \mid \neg r(a,b) \in \mathsf{L}\}) \cup \{(a,b) \mid r(a,b) \in \mathsf{L}\}$$

for all $A \in N_C$ and all $r \in N_R$. For a given set of ground effects E and an interpretation $\mathcal{I}$ we define an *effect function* $\mathcal{E}$ that maps E and $\mathcal{I}$ to a set of literals given by:

$$\mathcal{E}(\mathsf{E},\mathcal{I}) := \{\gamma \mid \psi/\gamma \in \mathsf{E}, \mathcal{I} \models \psi\} \cup \{\gamma \in \mathsf{E}\}.$$

To define how sensing affects the knowledge of the agent we introduce the notion of *sensing compatibility of interpretations*. Two interpretations $\mathcal{I}$ and $\mathcal{J}$ are sensing compatible w.r.t. a primitive ground action $\beta : (\mathsf{eff},\mathsf{sense})$, written as $\mathcal{I} \sim_\beta \mathcal{J}$, if for all $\psi \in \mathsf{sense}$ it holds that $\mathcal{I} \models \psi$ iff $\mathcal{J} \models \psi$.

Now we are ready to define the execution semantics of a primitive ground action.

Definition 2. Let $\beta : (\mathsf{eff},\mathsf{sense})$ be a primitive ground action, $(\mathcal{I},\mathcal{W})$ an epistemic interpretation with $\mathcal{I} \in \mathcal{W}$ and $(\mathcal{I}',\mathcal{W}')$ an epistemic interpretation. We write $(\mathcal{I},\mathcal{W}) \Longrightarrow_\beta (\mathcal{I}',\mathcal{W}')$, if the following conditions are satisfied:

- $\mathcal{I}' = \mathcal{I}^{\mathsf{L}}$ with $\mathsf{L} = \mathcal{E}(\mathsf{eff},\mathcal{I})$ and
- $\mathcal{W}' = \{\mathcal{J}^{\mathsf{L}} \mid \mathcal{J} \in \mathcal{W}, \mathcal{J} \sim_\beta \mathcal{I}, \mathsf{L} = \mathcal{E}(\mathsf{eff},\mathcal{J})\}$.

Let $\sigma = \beta_0,\ldots,\beta_{n-1}$ be a sequence of primitive ground actions. We write $(\mathcal{I}_0,\mathcal{W}_0) \Longrightarrow_\sigma (\mathcal{I}_n,\mathcal{W}_n)$ as an abbreviation for $(\mathcal{I}_0,\mathcal{W}_0) \Longrightarrow_{\beta_0} (\mathcal{I}_1,\mathcal{W}_1) \Longrightarrow_{\beta_1} \cdots \Longrightarrow_{\beta_{n-1}} (\mathcal{I}_n,\mathcal{W}_n)$.

The real world $\mathcal{I}$ of an epistemic interpretation $(\mathcal{I},\mathcal{W})$ is updated according to the physical effects eff. Intuitively, from $\mathcal{I}$ the agent receives information about the truth of each formula in sense. Interpretations in $\mathcal{W}$ contradicting this information are discarded, while those that agree with $\mathcal{I}$ are updated as well, yielding the new knowledge state. Thus, in our semantics the agent is fully aware of all effects of an action.

As a basic reasoning task we consider *projection*.

Definition 3 (projection). Let $\mathcal{K} = (\mathcal{T},\mathcal{A})$ be an objective KB, σ a sequence of primitive ground actions and ψ an $\mathcal{ALCOK}$-ground formula or $\mathcal{ALC}$-CI. We say that ψ is *valid after executing σ in $\mathcal{K}$* iff for all $\mathcal{I} \in \mathcal{M}(\mathcal{K})$ it holds that $(\mathcal{I}',\mathcal{W}') \mid\!\models \psi$ where $(\mathcal{I},\mathcal{M}(\mathcal{K})) \Longrightarrow_\sigma (\mathcal{I}',\mathcal{W}')$.

Note that the TBox $\mathcal{T}$ is only required to hold and to be known in the initial state, and that later states resulting from the execution of actions may violate it. While the persistence of $\mathcal{T}$ is thus not enforced in our formalization, checking this property is simply a special case of the projection problem.

$$\mathcal{T} = \{\mathit{Fault} \sqsubseteq \mathit{CritFault} \sqcup \mathit{UncritFault},$$
$$\exists \mathit{has\text{-}f}.\top \sqsubseteq \mathit{System}, \mathit{System} \sqsubseteq \forall \mathit{has\text{-}f}.\mathit{Fault}\}$$
$$\mathcal{A} = \{\mathit{System}(\mathit{gear}), \neg \mathit{On}(\mathit{gear}), \mathit{Fault}(\mathit{blocked})\}$$

Figure 3: Example initial knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$

Example 4. Figure 3 shows an initial KB $\mathcal{K}$ for our example domain. The first CI in $\mathcal{T}$ states that faults are critical faults or uncritical ones, the last two CIs define the domain *System* and range *Fault* for the role *has-f*. $\mathcal{A}$ describes a simple initial situation. Assume $\mathcal{K}$ is all the agent knows initially about the world. Thus, it is known that *gear* is *not on*, but the effect condition $\neg\exists \mathit{has\text{-}f}.\mathit{CritFault}(\mathit{gear})$ of $\texttt{turn-on}(\mathit{gear})$ is unknown (there is a least one possible world in $\mathcal{M}(\mathcal{K})$ satisfying it and one that does not). Consequently, after executing $\texttt{turn-on}(\mathit{gear})$ in $\mathcal{K}$, we get that $\neg\mathbf{Kw}\mathit{On}(\mathit{gear})$ is valid. If the agent now in turn executes $\texttt{sense-on}(\mathit{gear})$, it will also come to know whether *gear* has a critical fault or not, i.e. both $\mathbf{Kw}\exists \mathit{has\text{-}f}.\mathit{CritFault}(\mathit{gear})$ and $\mathbf{Kw}\mathit{On}(\mathit{gear})$ are valid.

The correctness of our action semantics can be shown by an embedding into the epistemic Situation Calculus based on the modal first-order logic $\mathcal{ES}$ [Lakemeyer and Levesque, 2004; 2010]. The embedding is done by translating the initial KB and the primitive actions into a Reiter-style basic action theory (BAT) formulated in $\mathcal{ES}$. The definition of our action semantics and the projection problem cause the initial KB and the (possibly conditional) effects of primitive actions to be *everything the agent knows* about the world and its dynamics. We have shown in [Zarrieß and Claßen, 2015] that the projection problem in our formalism thus exactly corresponds to an entailment problem in $\mathcal{ES}$ formulated by means of the only-knowing modality and the translated BAT.

4 Verification of Knowledge-Based Programs

We are now ready to assemble complex programs for describing the behavior of a knowledge-based agent. To specify desired properties of such programs we use a temporal extension of $\mathcal{ALCOK}$. Our main objective is to identify fragments of the programming language such that the verification problem, i.e. the problem of deciding whether all runs of a program satisfy the specified property, is decidable.

4.1 $\mathcal{ALCOK}$-Golog Programs

In this section we define the syntax and semantics of a Golog-like action programming language [Levesque *et al.*, 1997] that uses the action formalism introduced in the previous section. It allows to define complex behaviour through program expressions which are constructed from primitive actions, programming constructs, and tests formulated in $\mathcal{ALCOK}$. A program expression δ is built according to the following grammar:

$$\delta ::= [] \mid \alpha \mid \psi? \mid \delta;\delta \mid \delta^* \mid \delta|\delta \mid \mathsf{pick}(\vec{x}) : \psi?.\delta$$

A program can thus be the *empty program* [], a primitive action α, a *test* $\psi?$, where ψ is an $\mathcal{ALCOK}$-formula, or constructed from subprograms by means of *sequence* $\delta;\delta$, *nondeterministic iteration* δ^* (meaning execute δ zero or more times), *non-deterministic choice* $\delta|\delta$, and the *guarded pick constructor* $\mathsf{pick}(\vec{x}) : \psi?.\delta$, where ψ is a $\mathcal{ALCOK}$-formula and $\vec{x}$ are free variables in ψ and δ. Intuitively, the agent executes a guarded pick by non-deterministically choosing a binding $\vec{a}$ from Δ for the $\vec{x}$ such that ψ with $\vec{x}$ replaced by $\vec{a}$ is satisfied, after which it executes δ using the same bindings. An expression $\mathsf{pick}(\vec{x}) : \psi?$ is called *guarded pick*. Conditionals and while-loops can be defined in terms of the above, namely by **if** ψ **then** δ **else** δ' **end** $:= (\psi?;\delta)|(\neg\psi?;\delta')$ and **while** ψ **do** δ **end** $:= (\psi?;\delta)^*;\neg\psi?$.

Definition 5 ($\mathcal{ALCOK}$-Golog Program). Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a consistent $\mathcal{ALC}$-KB, Σ a finite set of primitive actions, and δ a *program expression* such that all primitive actions in δ are from Σ. An *$\mathcal{ALCOK}$-Golog program* is of the form $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$, where we require that Σ contains two predefined purely physical actions $\epsilon : (\mathsf{eff} = \{\mathit{Term}(p)\})$ and $\mathfrak{f} : (\mathsf{eff} = \{\mathit{Fail}(p)\})$ for indicating termination and failure of a program, respectively. Both special actions do not occur in δ and also the names *Term* and *Fail* are not used in $\mathcal{T}$ or in any other action or test. Furthermore, we require that $\{\neg\mathit{Term}(p), \neg\mathit{Fail}(p)\} \subseteq \mathcal{A}$ and δ *is closed*, i.e. all variables in δ are bound by a guarded pick. $\mathcal{P}$ is called *knowledge-based* if all tests in δ are subjective.

For the execution of a program we split up the program expression into its atomic programs and then execute these atomic programs step by step. An *atomic program*, denoted by $\mathfrak{a}$, is either a primitive action, a test or a guarded pick. We introduce two functions $\mathsf{head}(\cdot)$ and $\mathsf{tail}(\cdot,\cdot)$. Intuitively, $\mathsf{head}(\delta)$ contains those atomic programs that can be executed first when executing the program expression δ. It is defined by induction on the structure of δ. For instance, we have $\mathsf{head}([]) := \{\epsilon\}$, $\mathsf{head}(\delta^*) := \{\epsilon\} \cup \mathsf{head}(\delta)$ and

$$\mathsf{head}(\delta_1;\delta_2) := \{\mathfrak{a} \in \mathsf{head}(\delta_1) \mid \mathfrak{a} \neq \epsilon\} \cup \{\mathfrak{a} \in \mathsf{head}(\delta_2) \mid \epsilon \in \mathsf{head}(\delta_1)\}.$$

For $\mathfrak{a} \in \mathsf{head}(\delta)$, $\mathsf{tail}(\mathfrak{a}, \delta)$ yields the remainder of the program, i.e., the part that still needs to be executed after $\mathfrak{a}$ has been executed. For instance we have

$$\mathsf{tail}(\mathfrak{a}, \delta^*) := \{\delta';(\delta)^* \mid \delta' \in \mathsf{tail}(\mathfrak{a}, \delta), \mathfrak{a} \neq \epsilon\} \cup \{[] \mid \mathfrak{a} = \epsilon\}$$
$$\mathsf{tail}(\mathfrak{a}, \delta_1;\delta_2) := \{\delta';\delta_2 \mid \delta' \in \mathsf{tail}(\mathfrak{a}, \delta_1)\} \cup \{\delta' \mid \epsilon \in \mathsf{head}(\delta_1), \delta' \in \mathsf{tail}(\mathfrak{a}, \delta_2)\}.$$

The complete definitions of head and tail can be found in [Zarrieß and Claßen, 2015]. Note that if a program is "eaten up" completely, the termination action ϵ can be found in the head. We call a program expression that can be reached by a sequence of such head and tail applications a reachable

subprogram. The set of all reachable subprograms for an expression δ is denoted by $\mathsf{sub}(\delta)$. For a program expression ρ and a variable mapping ν, ρ^ν denotes the program expression obtained by replacing each occurrence of a free variable x in ρ by $\nu(x)$. We define $\mathsf{subg}(\rho) := \{\zeta^\nu \mid \zeta \in \mathsf{sub}(\rho), \nu \text{ is a variable mapping}\}$.

Given these auxiliary notions, we can define the semantics of a program $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$ as a transition system $T_\mathcal{P} = (Q, \to, I)$ with a set of *states* Q, a *transition relation* $\to$ and a set of *initial states* $I \subseteq Q$. A *state* is a pair $\langle(\mathcal{I}, \mathcal{W}), \rho\rangle \in Q$ consisting of an epistemic interpretation $(\mathcal{I}, \mathcal{W})$ with $\mathcal{I} \in \mathcal{W}$ and $\rho \in \mathsf{subg}(\delta)$.

There is a transition $\langle(\mathcal{I}, \mathcal{W}), \rho\rangle \to \langle(\mathcal{I}', \mathcal{W}'), \rho'\rangle$, iff

- there exists a primitive action $\alpha \in \mathsf{head}(\rho)$ such that $(\mathcal{I}, \mathcal{W}) \Longrightarrow_\alpha (\mathcal{I}', \mathcal{W}')$ and $\rho' \in \mathsf{tail}(\alpha, \rho)$ or
- there exists a test $\psi? \in \mathsf{head}(\rho)$ such that $(\mathcal{I}, \mathcal{W}) \mid\!\models \psi$, $(\mathcal{I}', \mathcal{W}') = (\mathcal{I}, \mathcal{W})$ and $\rho' \in \mathsf{tail}(\psi?, \rho)$ or
- there exists $\mathsf{pick}(\vec{x}) : \psi? \in \mathsf{head}(\rho)$ and a variable map ν such that $(\mathcal{I}, \mathcal{W}) \mid\!\models \psi^\nu$, $(\mathcal{I}', \mathcal{W}') = (\mathcal{I}, \mathcal{W})$ and there is $\zeta \in \mathsf{tail}(\mathsf{pick}(\vec{x}) : \psi?, \rho)$ with $\rho' = \zeta^\nu$,
- or, if no atomic program in the head of ρ is applicable, then the failure action $\mathfrak{f}$ is executed in $(\mathcal{I}, \mathcal{W})$ by leaving ρ unchanged.

Due to the distinguished actions ϵ and $\mathfrak{f}$, a successor state is always guaranteed to exist. The initial states are of the form $\langle(\mathcal{I}, \mathcal{M}(\mathcal{K})), \delta\rangle$ with $\mathcal{I} \in \mathcal{M}(\mathcal{K})$. A *run* π of a program $\mathcal{P}$ is an infinite path in $T_\mathcal{P} = (Q, \to, I)$ starting in an initial state. The infinite sequence of epistemic interpretations occurring in the states along a run π is denoted by $\mathfrak{I}(\pi)$.

4.2 Specifying Temporal Properties of Programs

To specify temporal properties of a given program we use the logic $\mathcal{ALCOK}$-LTL that generalizes $\mathcal{ALC}$-LTL [Baader *et al.*, 2008]. $\mathcal{ALCOK}$-LTL formulas are built according to the following grammar:

$$\Phi ::= \varrho \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \mathsf{X}\Phi \mid \Phi_1 \,\mathsf{U}\, \Phi_2$$

where ϱ stands for an $\mathcal{ALCOK}$-ABox assertion or $\mathcal{ALC}$-CI. We use the following usual abbreviations: $\Diamond\Phi := \top(a)\,\mathsf{U}\,\Phi$ (*eventually*), $\Box\Phi := \neg\Diamond\neg\Phi$ (*globally*) and $\Phi_1 \to \Phi_2 := \neg\Phi_1 \vee \Phi_2$.

The semantics of $\mathcal{ALCOK}$-LTL is based on the notion of an *$\mathcal{ALCOK}$-LTL structure*, which is an infinite sequence of epistemic interpretations $\mathfrak{I} = (\mathcal{I}_i, \mathcal{W}_i)_{i=0,1,2,\ldots}$. Let Φ be an $\mathcal{ALCOK}$-LTL formula, $\mathfrak{I}$ an $\mathcal{ALCOK}$-LTL structure, and $i \in \{0, 1, 2, \ldots\}$ a time point. Validity of Φ in $\mathfrak{I}$ at time i, denoted by $\mathfrak{I}, i \models \Phi$, is defined as follows:

$$\begin{aligned}
&\mathfrak{I}, i \models \varrho && \text{iff } (\mathcal{I}_i, \mathcal{W}_i) \models \varrho,\\
&\mathfrak{I}, i \models \neg\Phi && \text{iff } \mathfrak{I}, i \not\models \Phi,\\
&\mathfrak{I}, i \models \Phi_1 \wedge \Phi_2 && \text{iff } \mathfrak{I}, i \models \Phi_1 \text{ and } \mathfrak{I}, i \models \Phi_2,\\
&\mathfrak{I}, i \models \mathsf{X}\Phi && \text{iff } \mathfrak{I}, i+1 \models \Phi,\\
&\mathfrak{I}, i \models \Phi_1 \,\mathsf{U}\, \Phi_2 && \text{iff } \exists k \geq i : \mathfrak{I}, k \models \Phi_2 \text{ and}\\
& && \quad \forall j, i \leq j < k : \mathfrak{I}, j \models \Phi_1.
\end{aligned}$$

Now, we are ready to define the *verification problem*.

Definition 6. Let $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$ be an $\mathcal{ALCOK}$-Golog program and Φ an $\mathcal{ALCOK}$-LTL formula. The formula Φ is *valid* in $\mathcal{P}$ iff for all runs π of $\mathcal{P}$ it holds that $\mathfrak{I}(\pi), 0 \models \Phi$.

Example 7. In a program that uses the program expression given in Figure 1 and the initial KB in Figure 3 the following property is valid:

$$\exists \textit{has-f}.(\textit{CritFault} \sqcap \neg\mathbf{K}\textit{Fault})(\textit{gear}) \to \Diamond\mathbf{K}\exists \textit{has-f}.(\textit{CritFault} \sqcap \neg\mathbf{K}\textit{Fault})(\textit{gear})$$

saying that if *gear* has an unknown critical fault initially, then the agent will eventually recognize it. We can also verify executability and whether the TBox $\mathcal{T}$ is preserved along each run by checking validity of $\Box\big(\bigwedge_{\varrho\in\mathcal{T}} \varrho \wedge \neg\textit{Fail}(p)\big)$.

4.3 Verifying Restricted Programs

Unfortunately, it turns out that in the general case the verification problem is undecidable. The main source of undecidability is the high degree of non-determinism introduced by the guarded pick operator that allows to quantify arguments ranging over the whole countably infinite domain Δ. We get undecidability even in an already quite restricted subset of our language by a straightforward reduction of the halting problem of two-counter machines [Minsky, 1967].

Theorem 8. *The verification problem is undecidable even if no CIs are used, the initial TBox is empty and the primitive actions have only unconditional effects, at most one argument, and provide no sensing result.*

In order to retain decidability of the verification problem we restrict the *syntax of the guards* allowed in the pick-operators.

Definition 9. Let $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$ be an $\mathcal{ALCOK}$-Golog program. $\mathcal{P}$ is called *restricted* if all guarded picks occurring in δ are of the form

$$\mathsf{pick}(x_1, \ldots, x_n) : (\psi_1 \vee \cdots \vee \psi_m) \wedge \psi'?$$

with $m \geq 1$ and for each ψ_i with $1 \leq i \leq m$ it holds that $\{x_1, \ldots, x_n\} \subseteq \mathsf{Var}(\psi_i)$ and ψ_i is a conjunction of concept and role atoms of the form $\mathbf{K}C(z)$ and $\mathbf{K}r(z, z')$, respectively, where C is an $\mathcal{ALC}$-concept with $\mathcal{K} \not\models \top \sqsubseteq C$. There are no restrictions on the formula ψ'.

Note that the example program in Figure 1 is restricted. The restricted atoms in the guard of the pick-operator can also be viewed as objective instance queries posed to the current KB of the agent. The agent then chooses a binding for the variables among the retrieved answers. As a consequence of the restriction we obtain the following property for the transition system. From now on let $\mathsf{Ind} \subseteq \Delta$ be a fixed finite set of individuals, including all individuals occurring in $\mathcal{P}$.

Lemma 10. *Let $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$ be a restricted program, $\langle(\mathcal{I}, \mathcal{W}), \rho\rangle$ a state in $T_\mathcal{P}$ reachable from an initial state, C an $\mathcal{ALC}$-concept with $\mathcal{K} \not\models \top \sqsubseteq C$ and r a role name. It holds that $(\mathbf{K}r)^\mathcal{W} \subseteq \mathsf{Ind} \times \mathsf{Ind}$ and $(\mathbf{K}C)^\mathcal{W} \subseteq \mathsf{Ind}$.*

Consequently, the concepts and roles we use as qualifiers for objects chosen in the guards of a pick-operator have finite extensions under knowledge states reachable from an initial state. Thus, the range of the pick operator is limited to objects mentioned in $\mathcal{P}$. Intuitively, the agent won't be able to

directly cope with unknown individuals. As seen in our running example, the agent is able to recognize whether or not there is *some unknown* fault, but there is no possibility to directly access it.

However, even in this restricted setting the transition system has still infinitely many states. Since the epistemic model $\mathcal{M}(\mathcal{K})$ is infinite, there also infinitely many initial states, namely one initial state for each $\mathcal{I} \in \mathcal{M}(\mathcal{K})$.

To show decidability of the verification problem for restricted programs, we basically proceed as in [Baader and Zarrieß, 2013; Zarrieß and Claßen, 2014]: We show that a finite bisimilar abstraction of the transition system of the program can be constructed such that the verification problem boils down to a propositional model checking problem of the finite abstraction against the propositional abstraction of the $\mathcal{ALCOK}$-LTL formula. In the following we briefly outline the main steps of the construction of the abstraction.

Recall that a state in the transition system is of the form $\langle(\mathcal{I}, \mathcal{W}), \rho\rangle$ where $\rho \in \mathsf{subg}(\delta)$. As a consequence of Lemma 10 we only need to consider possible instantiations with objects from the finite set Ind. Therefore, we are able to show that there are only polynomially many (in the size of δ) reachable program expressions in the transition system. Our focus now is on how to partition the epistemic interpretations that appear in the transition system into finitely many equivalence classes such that the satisfaction of temporal properties is preserved. First, we use Lemma 10 to identify a finite set of relevant axioms. By considering all axioms in the initial KB, in the temporal property, all groundings of atoms appearing as tests in the program and primitive actions, we obtain a finite set of relevant $\mathcal{ALCOK}$-axioms that we call *context* of $\mathcal{P}$, denoted by $\mathcal{C}_{\mathbf{K}}(\mathcal{P})$. This set is the basis for constructing *types*, i.e. equivalence classes, of single interpretations. To do this we need to deal with the references to knowledge in the axioms of the context. We make use of the following simple structure of the reachable knowledge states.

Lemma 11. *Let $\mathbf{K}D$ be an $\mathcal{ALCOK}$-concept mentioning only individuals from Ind and $\langle(\mathcal{I}, \mathcal{W}), \rho\rangle$ a reachable state in the transition system of a restricted program. If there exists $d \in \Delta \setminus \mathsf{Ind}$ such that $d \in (\mathbf{K}D)^{\mathcal{W}}$, then $\Delta \setminus \mathsf{Ind} \subseteq (\mathbf{K}D)^{\mathcal{W}}$.*

Consequently, $(\mathbf{K}D)^{\mathcal{W}}$ is a finite subset of Ind, or a finite subset of Ind plus all unnamed elements from $\Delta \setminus \mathsf{Ind}$. This result together with the property of epistemic roles shown in Lemma 10 gives rise to a rewriting of epistemic sub-concepts by non-epistemic ones. We define the notion of an *instance function* κ that maps concepts of the form $\mathbf{K}D$ to a subset of $\{\{a\} \mid a \in \mathsf{Ind}\} \cup \{\neg N\}$ where $N := \bigsqcup_{a \in \mathsf{Ind}} \{a\}$, and a given role name r and an individual a to a subset of $\{\{a\} \mid a \in \mathsf{Ind}\}$. Intuitively, an instance function represents an abstraction of a possible knowledge state. Given an $\mathcal{ALCOK}$-concept C and an instance function κ we have defined a rewriting that yields an objective concept C^{κ}. The rewriting technique is very similar to the one used in the Representation Theorem [Levesque and Lakemeyer, 2001], where the idea is to replace each knowledge subformula by a description of its instances.

Clearly, the context $\mathcal{C}_{\mathbf{K}}(\mathcal{P})$ contains only finitely many epistemic roles and epistemic subconcepts. Therefore there are only finitely many relevant instance functions we need to consider. Using the rewriting operator we can close up the context under all possible rewritings of epistemic subconcepts and roles, which yields the so-called *knowledge closure* of the context. Let $\widehat{\mathcal{C}}(\mathcal{P})$ be the set of all non-epistemic axioms contained in the knowledge closure and 2^{Lit} denote the set of all sets of literals occurring in the context, used for representing accumulated physical effects. The *dynamic type* of an interpretation $\mathcal{I}$ is then given by:

$$\mathsf{d\text{-}type}(\mathcal{I}) := \{(\varrho, \mathsf{E}) \in \widehat{\mathcal{C}}(\mathcal{P}) \times 2^{Lit} \mid \mathcal{I}^{\mathsf{E}} \models \varrho\}.$$

Obviously, there are only finitely many different dynamic types. We showed that the problem whether a given subset of $\widehat{\mathcal{C}}(\mathcal{P}) \times 2^{Lit}$ represents the dynamic type of an interpretation $\mathcal{I} \in \mathcal{M}(\mathcal{K})$ is reducible to a consistency check in $\mathcal{ALCO}$.

The knowledge closure and the dynamic types are the main technical notions used to construct the abstract transition system. The abstraction of a state is given by the dynamic type of the real world, the set of dynamic types for the possible worlds in the knowledge state, and the program expression. Given this finite abstract transition system which preserves satisfaction of temporal properties, we can treat axioms in the $\mathcal{ALCOK}$-LTL property as propositional atoms and apply LTL model checking to solve the verification problem.

Theorem 12. *Let $\mathcal{P}$ be a restricted $\mathcal{ALCOK}$-Golog program and Φ an $\mathcal{ALCOK}$-LTL formula. The problem whether Φ is valid in $\mathcal{P}$ or not is decidable in* 2EXPSPACE.

Since the size of $\widehat{\mathcal{C}}(\mathcal{P})$ and 2^{Lit} is exponential in the size of the input, there are at most double-exponentially many dynamic types and at most triple-exponentially many states in the abstract transition system, which can be constructed on-the-fly. We thus obtain a 2EXPSPACE upper bound.

The presented decidable fragment is also maximal in the sense that relaxing any of the restrictions leads to undecidability. For instance it can be shown that this happens if the restricted conjunct of the guard is allowed to contain nominals or nested $\mathbf{K}$ operators within concept atoms, or role atoms of the form $\mathbf{K}\neg r(z, z')$. The problem is that such extensions may open the domain of the pick-operator for anonymous objects, which immediately causes undecidability.

Finally note that although the projection problem (see Definition 3) can be viewed as a special case of the verification problem, it is worth special consideration as it is also the basic reasoning task for *executing* a knowledge-based program. We have the following result:

Theorem 13. *The projection problem is* EXPTIME*-complete, and* PSPACE*-complete if the initial TBox is empty.*

The proofs of the results presented in this section can be found in the technical report [Zarrieß and Claßen, 2015].

5 Conclusion

So far, little work has been done on decidable verification of knowledge-based Golog programs. De Giacomo, Lespérance and Patrizi [2013] present a class of epistemic Situation Calculus action theories for which they show decidability of μ-calculus properties, however they do not consider Golog and

rely on a purely semantical definition of this class. On the propositional level, Lang and Zanuttini [2012] have investigated the complexity of verifying post-conditions of a class of modal logic knowledge-based programs that could be viewed as a (deterministic) subset of Golog.

Finally, alternative approaches for reasoning about actions and programs using DLs were proposed. The formalization presented in [Calvanese *et al.*, 2011] for example adopts Levesque's functional view on knowledge bases, where all interactions with the agent's KB to happen through the two operations ASK (test evaluation) and TELL (update after action execution). While this allows for tractable solutions to the executability and projection problems for certain light-weight DLs, this non-declarative representation makes no distinction between world-changing and sensing actions as we do. Also, verification of temporal properties is not considered.

In this paper, we introduced an action language for both physical and sensing actions based on the epistemic DL $\mathcal{ALCOK}$. We showed that under suitable restrictions, verifying LTL properties over possibly epistemic $\mathcal{ALCOK}$-axioms of knowledge-based Golog programs based on our action language is decidable. The main idea to obtain decidability is to syntactically limit the domain of the guarded pick operator to contain named objects only.

Furthermore, we investigated the complexity of projection as the basic reasoning task for executing knowledge-based programs. As future work, among other things, we want to investigate whether the obtained upper complexity bound of the verification problem can be improved, or it is actually tight.

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

References

[Baader and Zarrieß, 2013] F. Baader and B. Zarrieß. Verification of Golog programs over description logic actions. In *FroCoS'13*, volume 8152 of *LNAI*. Springer-Verlag, 2013.

[Baader *et al.*, 2005] F. Baader, C. Lutz, M. Miličić, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In *AAAI 2005*, AAAI Press, 2005.

[Baader *et al.*, 2008] F. Baader, S. Ghilardi, and C. Lutz. LTL over description logic axioms. In *KR 2008*, AAAI Press, 2008.

[Calvanese *et al.*, 2011] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Actions and programs over description logic knowledge bases: A functional approach. In *Knowing, Reasoning, and Acting: Essays in Honour of Hector J. Levesque*. College Publications, 2011.

[Claßen and Lakemeyer, 2006] J. Claßen and G. Lakemeyer. Foundations for knowledge-based programs using ES. In *KR 2006*, AAAI Press, 2006.

[Claßen and Lakemeyer, 2008] J. Claßen and G. Lakemeyer. A logic for non-terminating Golog programs. In *KR 2008*, AAAI Press, 2008.

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

[De Giacomo *et al.*, 2010] G. De Giacomo, Y. Lespérance, and A. R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *KR 2010*, AAAI Press, 2010.

[De Giacomo *et al.*, 2012] G. De Giacomo, Y. Lespérance, and F. Patrizi. Bounded situation calculus action theories and decidable verification. In *KR 2012*, AAAI Press, 2012.

[De Giacomo *et al.*, 2013] G. De Giacomo, Y. Lespérance, and F. Patrizi. Bounded epistemic situation calculus theories. In *IJCAI 2013*, IJCAI/AAAI, 2013.

[Donini *et al.*, 1998] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.

[Lakemeyer and Levesque, 2004] G. Lakemeyer and H. J. Levesque. Situations, si! situation terms, no! In *KR 2004*, AAAI Press, 2004.

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

[Lang and Zanuttini, 2012] J. Lang and B. Zanuttini. Knowledge-based programs as plans - the complexity of plan verification. In *ECAI 2012*, 2012.

[Levesque and Lakemeyer, 2001] H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.

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

[Levesque, 1990] H. J. Levesque. All I know: A study in autoepistemic logic. *Artif. Intell.*, 42(2-3):263–309, 1990.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. American Elsevier, New York, 1969.

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

[Reiter, 1991] R. Reiter. 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*, pages 359–380, 1991.

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

[Reiter, 2001b] R. Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Trans. Comput. Log.*, 2(4):433–457, 2001.

[Scherl and Levesque, 2003] R. B. Scherl and H. J. Levesque. Knowledge, action, and the frame problem. *Artif. Intell.*, 144(1-2):1–39, 2003.

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

[Zarrieß and Claßen, 2015] B. Zarrieß and J. Claßen. Verification of knowledge-based programs over description logic actions. LTCS-Report 15-10, TU Dresden, 2015. See http://lat.inf.tu-dresden.de/research/reports.html.