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 E5, 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) 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 E5 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 sense-f (gear, x) to sense whether the individual gear has fault x and sense-on (gear) to check if gear is on or not. Furthermore, the physical action repair (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...
Figure 1: Example program

\[\begin{align*}
\text{while } & \neg K\langle \text{Vhas-f,} \neg \text{KFault}\rangle(\text{gear}) \\
\text{pick}(x): & K\text{Fault}(x) \land \neg K\text{has-f}(\text{gear}, x) \land \neg K\langle \text{has-f,} \neg \text{gear}\rangle(\text{gear}, x)?, \\
\text{sense-f}(\text{gear}, x); \\
& \text{if } K\langle \text{has-f,} \text{gear}\rangle(x) \text{ then repair}(\text{gear}, x) \text{ else continue}; \\
\text{end} \\
\text{turn-on}(\text{gear}); & \text{sense-on}(\text{gear}); \\
& \text{if } K\langle \text{On,} \text{gear}\rangle \text{ then raise-alarm else continue};
\end{align*}\]

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 notions 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 non-deterministic 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 Logic $\text{ALCOK}$

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

Let $\mathcal{N}_R$, $\mathcal{N}_C$, $\mathcal{N}_I$ be countably infinite sets of role names, concept names and individual names, respectively. A generalized $\text{ALCOK}$-role $\mathcal{P}$ is built from role names using the epistemic role constructor and role negation (see Table 1). A simple $\text{ALCOK}$-role (role for short) is a generalized role without role negation. $\text{ALCOK}$-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 $a \in \mathcal{N}_I$. The following concept constructors are defined as abbreviations: $C \sqcup D := \neg (\neg C \cap \neg D)$, $\text{KW}C := \text{KC} \sqcup \neg \text{K}C$ and $\forall R.C := \neg \exists R.\neg C$. Intuitively, an object belongs to the concept $\text{KW}C$ 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, $\text{a}$ for a concept name, $C, D$ for possibly complex concepts, and $a, b$ for individual names (individuals).

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Semantics under $(I, W)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>role name</td>
<td>$r$</td>
<td>$r^I \subseteq \Delta \times \Delta \setminus P^I,W$</td>
</tr>
<tr>
<td>role negation</td>
<td>$\neg^P$</td>
<td>$\Delta \setminus C^I,W$</td>
</tr>
<tr>
<td>epistemic role</td>
<td>$\text{K}^P$</td>
<td>$\bigcap_{I \in W} \text{P}^I,W$</td>
</tr>
<tr>
<td>concept name</td>
<td>$\mathcal{A}$</td>
<td>$\mathcal{A}^I \subseteq \Delta$</td>
</tr>
<tr>
<td>top</td>
<td>$\top$</td>
<td>$\Delta$</td>
</tr>
<tr>
<td>negation</td>
<td>$\neg^C$</td>
<td>$\Delta \setminus C^I,W$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$C \sqcup D$</td>
<td>$C^I,W \cap D^I,W$</td>
</tr>
<tr>
<td>exist. rest.</td>
<td>$\exists R.C$</td>
<td>${d \mid \exists e : (d, e) \in R^I,W, e \in C^I,W}$</td>
</tr>
<tr>
<td>nominal</td>
<td>${a}$</td>
<td>${a^I}$</td>
</tr>
<tr>
<td>epistemic concept</td>
<td>$\text{KC}^C$</td>
<td>$\bigcap_{I \in W} (C^I,W)$</td>
</tr>
</tbody>
</table>

Table 1: Syntax and semantics of roles and concepts

The different kinds of $\text{ALCOK}$-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 $\text{ALCOK}$ assertions), where assertions of the form $\mathcal{A}(a), \neg \mathcal{A}(a)$, $\mathcal{R}(a,b)$, $\neg \mathcal{R}(a,b)$ are called literals. A known-
(I, M) \models K and for all sets of interpretations \( M' \) with \( M \subseteq M' \) there exists \( J \in M' \) such that \((J, M') \models K \).

An axiom \( \varphi \) is epistemically entailed by \( K \), written as \( K \models \varphi \), iff for all epistemic models \( M \) of \( K \) and for all \( I \in M \) it holds that \((I, M) \models \varphi \). \( K \) is called consistent if \( K \) has an epistemic model.

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

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

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

### 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 \( \textit{atom} \) 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 \( \textit{atoms} \) are of the form \( A(z), \neg A(z), r(z, z') \) or \( \neg r(z, z') \). A \textit{formula} is a boolean combination of atoms. The set of all variables occurring in a formula \( \psi \) is denoted by \( \text{Var}(\psi) \). A formula without variables is called \textit{ground formula}. Subjectivity and objectivity of formulas is defined in the obvious way. We will use the symbols \( \varphi \) for atoms, \( \gamma \) for primitive atoms and \( \psi \) for formulas.

**Definition 1.** An effect is either of the form \( \psi/\gamma \) (conditional effect) or of the form \( \gamma \) (unconditional effect), where \( \psi \) is an objective formula called effective condition and \( \gamma \) a primitive atom. A \textit{primitive action} \( \alpha \) is a pair \( \alpha : (\text{eff, sense}) \) where \( \text{eff} \) is a finite set of effects and \( \text{sense} \) 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 \( \alpha \). An action without variables is called \textit{ground action}.

\[
\text{turn-on}(x) : (\text{eff} = \{\neg \exists \text{has-f.CritFault}(x) / \text{On}(x)\}) \\
\text{sense-on}(x) : (\text{sense} = \{\text{On}(x)\}).
\]

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. \text{turn-on}(x) \ has a single conditional effect that causes \( x \) to be \text{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. \text{sense-on}(x) \ is a sensing action that represents the agent’s ability to perceive whether \text{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 \((I, W)\) which allows us to explicitly distinguish changes affecting the real world represented by \( I \) and changes to the knowledge state \( W \).

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

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

\[
A^{\nu} := (A^Z \setminus \{a \mid \neg A(a) \in L\}) \cup \{a \mid A(a) \in L\}; \\
r^{\nu} := (r^Z \setminus \{(a, b) \mid \neg r(a, b) \in L\}) \cup \{(a, b) \mid r(a, b) \in L\}
\]

for all \( A \in N_A \) and all \( r \in N_R \). For a given set of ground effects \( E \) and an interpretation \( I \) we define an effect function \( E \) that maps \( E \) and \( I \) to a set of literals given by:

\[
E(I) := \{\gamma \mid \psi/\gamma \in E, I \models \psi \} \cup \{\gamma \in E\}.
\]

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

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

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

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

Let \( \sigma = \beta_0, \ldots, \beta_{n-1} \) be a sequence of primitive ground actions. We write \((I_0, W_0) =\rightarrow_\sigma (I_n, W_n)\) as an abbreviation for \((I_0, W_0) =\rightarrow_{\beta_0} (I_1, W_1) =\rightarrow_{\beta_1} \cdots =\rightarrow_{\beta_{n-1}} (I_n, W_n)\). The real world \( I \) of an epistemic interpretation \((I, W)\) is updated according to the physical effects eff intuitively, from \( I \) the agent receives information about the truth of each formula in sense. Interpretations in \( W \) contradicting this information are discarded, while those that agree with \( I \) 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 \textit{projection}.

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

$$\mathcal{T} = \{ \text{Fault } \sqsubseteq \text{CritFault} \sqcup \text{UncritFault}, \exists \text{has-f.f.} \sqsubseteq \text{System}, \text{System } \sqsubseteq \forall \text{has-f.Fault} \}$$

$$\mathcal{A} = \{ \text{System}(\text{gear}), \neg \text{On}(\text{gear}), \text{Fault}(\text{blocked}) \}$$

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 $\text{System}$ and range $\text{Fault}$ for the role $\text{has-f}$. $\mathcal{A}$ describes a simple initial situation. Assume $\mathcal{K}$ is all the agent knows initially about the world. Thus, it is known that $\text{gear}$ is not on, but the effect condition $\neg \exists \text{has-f.CritFault}(\text{gear})$ of $\text{turn-on(\text{gear})}$ is unknown (there is at least one possible world in $\mathcal{M}(\mathcal{K})$ satisfying it and one that does not). Consequently, after executing $\text{turn-on(\text{gear})}$ in $\mathcal{K}$, we get that $\neg \text{KwOn}(\text{gear})$ is valid. If the agent now in turn executes $\text{sense-on(\text{gear})}$, it will also come to know whether $\text{gear}$ has a critical fault or not, i.e. both $\text{Kw}\exists \text{has-f.CritFault}(\text{gear})$ and $\text{KwOn}(\text{gear})$ 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 $\text{everything the agent knows}$ 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 $\text{ALCOK}$. 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 $\text{ALCOK-Golog}$ 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 $\text{ALCOK}$.

A program expression $\delta$ is built according to the following grammar:

$$\delta ::= [\, | \alpha \, | \psi ? | \delta; \delta | \delta^* | \delta|\delta | \text{pick}(\vec{x}) : \psi?, \delta$$

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

**Definition 5 ($\text{ALCOK-Golog}$ Program).** Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a consistent $\text{ALC}$-KB, $\Sigma$ a finite set of primitive actions, and $\delta$ a program expression such that all primitive actions in $\delta$ are from $\Sigma$. An $\text{ALCOK-Golog}$ program is of the form $\mathcal{P} = (\mathcal{K}, \Sigma, \delta)$, where we require that $\Sigma$ contains two predefined purely physical actions $\epsilon : (\text{eff} = \{\text{Term}(p)\})$ and $\mathcal{f} : (\text{eff} = \{\text{Fail}(p)\})$ for indicating termination and failure of a program, respectively. Both special actions do not occur in $\delta$ and also the names $\text{Term}$ and $\text{Fail}$ are not used in $\mathcal{T}$ or in any other action or test. Furthermore, we require that $\{\neg \text{Term}(p), \neg \text{Fail}(p)\} \subseteq \mathcal{A}$ and $\delta$ is closed, i.e. all variables in $\delta$ are bound by a guarded pick. $\mathcal{P}$ is called knowledge-based if all tests in $\delta$ 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 $\alpha$, is either a primitive action, a test or a guarded pick. We introduce two functions $\text{head}(\cdot)$ and $\text{tail}(\cdot)$. Intuitively, $\text{head}(\delta)$ contains those atomic programs that can be executed first when executing the program expression $\delta$. It is defined by induction on the structure of $\delta$. For instance, we have $\text{head}([\,]) := \{\epsilon\}$, $\text{head}(\delta^*) := \{\epsilon\} \cup \text{head}(\delta)$ and $\text{head}(\alpha; \delta) := \{a \in \text{head}(\alpha) \mid a \neq \epsilon\} \cup \{a \in \text{head}(\delta) \mid \epsilon \in \text{head}(\alpha)\}$.

For a $a \in \text{head}(\delta)$, $\text{tail}(\alpha; \delta)$ yields the remainder of the program, i.e., the part that still needs to be executed after $a$ has been executed. For instance we have $\text{tail}(a; \delta^*) := \{\delta'; (\delta^*) \mid \delta' \in \text{tail}(\alpha, \delta), a \neq \epsilon\} \cup \{[\,] \mid a = \epsilon\}$

$$\text{tail}(\alpha; \delta_1; \delta_2) := \{\delta' \mid \delta_2' \in \text{tail}(\alpha, \delta_1)\} \cup \{\delta' \mid \epsilon \in \text{head}(\delta_1), \delta' \in \text{tail}(\alpha, \delta_2)\}.$$
subprogram. The set of all reachable subprograms for an expression $\delta$ is denoted by $\text{sub}(\delta)$. For a program expression $\rho$ and a variable mapping $\nu$, $\rho'$ denotes the program expression obtained by replacing each occurrence of a free variable $x$ in $\rho$ by $\nu(x)$. We define $\text{sub}(\rho) := \{ \zeta'' \mid \zeta \in \text{sub}(\rho), \nu$ is a variable mapping $\}$.

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

There is a transition $\langle (I, W), \rho \rangle \rightarrow \langle (I', W'), \rho' \rangle$, iff

- there exists a primitive action $\alpha \in \text{head}(\rho)$ such that $(I, W) \models \alpha \iff (I', W') \models \rho$ and $\rho' \subseteq \text{tail}(\alpha, \rho)$ or
- there exists a test $\psi \in \text{head}(\rho)$ such that $(I, W) \models \psi \iff (I', W') \models \rho$ and $\rho' \subseteq \text{tail}(\psi, \rho)$ or
- there exists pick$(\vec{x}) : \psi? \in \text{head}(\rho)$ and a variable map $\nu$ such that $(I, W) \models \psi^\nu$, $(I', W') = (I, W)$ and there is $\zeta \in \text{tail}(\text{pick}(\vec{x}) : \psi?$, $\rho)$ with $\rho' \models \zeta''$, or
- if, at atomic program, in the head of $\rho$ is applicable, then the failure action $f$ is executed in $(I, W)$ by leaving $\rho$ unchanged.

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

### 4.2 Specifying Temporal Properties of Programs

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

\[ \Phi ::= G \mid \neg \Phi \mid \Phi_1 \land \Phi_2 \mid \Phi_1 \lor \Phi_2 \mid X \Phi \mid F_1 \lor F_2 \]

where $G$ stands for an $\text{ALCOK-\text{ABox}}$ assertion or $\text{ALC-\text{CI}}$. We use the following usual abbreviations: $\Box \Phi := \neg G \Phi$ (eventually), $\Diamond \Phi := \neg \Box \Phi$ (globally) and $F_1 \lor F_2 := \neg F_1 \lor \neg F_2$.

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

- $J, i \models G \iff (I_i, W_i) \models G$,
- $J, i \models \neg \Phi \iff J, i \not\models \Phi$,
- $J, i \models F_1 \land F_2 \iff J, i \models F_1$ and $J, i \models F_2$,
- $J, i \models X \Phi \iff J, i+1 \models \Phi$,
- $J, i \models F_1 \lor F_2 \iff \exists k \geq i : J, k \models F_2$ and $\forall j, i \leq j < k : J, j \models F_1$.

Now, we are ready to define the verification problem.

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

\[ \text{pick}(x_1, \ldots, x_n) : (\psi_1 \lor \cdots \lor \psi_m) \land \psi' \]

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

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 $\text{Ind} \subseteq \Delta$ be a fixed finite set of individuals, including all individuals occurring in $P$.

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

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 $P$. Intuitively, the agent won’t be able to
Lemma 10 gives rise to a rewriting of epistemic subconcepts and roles, which yields the so-called knowledge closure of the context. Let $\hat{C}(\mathcal{P})$ 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 $I$ is then given by:

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

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

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 $\Phi$ an $\mathcal{ALCOK}$-LTL formula. The problem whether $\Phi$ is valid in $\mathcal{P}$ or not is decidable in $\mathcal{2EXPSPACE}$.

Since the size of $\hat{C}(\mathcal{P})$ 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 $\mathcal{2EXPSPACE}$ 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 $\mathcal{K}$ operators within concept atoms, or role atoms of the form $\mathcal{K} \mathcal{r}(z,z)$. 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 $\mathcal{EXPTIME}$-complete, and $\mathcal{PSPACE}$-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 $\mu$-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 ALCOK. We showed that under suitable restrictions, verifying LTL properties over possibly epistemic 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


