Decidable Verification of Knowledge-Based Programs over Description Logic Actions with Sensing [⋆]

(Extended Abstract [⋆⋆])

Benjamin Zarrieß[1] and Jens Claßen[2]

[1] Theoretical Computer Science, TU Dresden, Germany,
[2] Knowledge-Based Systems Group, RWTH Aachen University, Germany

1 Introduction

Since the Golog [5, 10] 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 received increasing attention [4, 7]. Both the Golog language itself and the underlying Situation Calculus [11, 13] 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 [6, 15].

In this extended abstract 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 in the epistemic Situation Calculus were proposed by Reiter [14] and later by Claßen and Lakemeyer [3].

Here we review our work on a new epistemic action formalism based on the basic Description Logic (DL) $\mathcal{ALC}$ obtained by combining and extending earlier proposals for DL action formalisms [1] and epistemic DLs [8]. 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. More precisely, in our setting a knowledge-based programs for the control of a single agent consists of the following ingredients: 1. an (objective) $\mathcal{ALC}$-TBox and ABox representing the *initial static knowledge* of the agent about the world.; 2. a set of *primitive actions* describing the basic abilities of an agent to change the world and to gain new information from the environment and 3. a *program expression* defining the possible courses of action by combining primitive actions and subjective conditions formulated in the epistemic DL $\mathcal{ALCOK}$ (an extension of $\mathcal{ALC}$ with nominals ($\mathcal{O}$) and an epistemic constructor ($\mathcal{K}$)) using programming constructs

[⋆] Supported by DFG Research Unit FOR 1513, (http://www.hybrid-reasoning.org)
[⋆⋆] See [2] for the long versions of the paper and [16] for the technical report.

for sequencing, iteration and nondeterministic choice. Desired properties of such a program can be expressed in LTL over $\mathcal{ALC}$-concept inclusions and $\mathcal{ALCOK}$-ABox assertions - a logic we call $\mathcal{ALCOK}$-LTL. The *verification problem* asks whether or not all runs of a given knowledge-based program satisfy a given $\mathcal{ALCOK}$-LTL formula.

Verifying knowledge-based programs with this language yields multiple advantages. First, under reasonable restrictions 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 [12] solution to the frame problem and a reasoning mechanism resembling Levesque and Lakemeyer's [9] Representation Theorem where reasoning about knowledge is reduced to reasoning in the standard DL $\mathcal{ALCO}$.

2 Example

As an example consider a mobile robot in a factory whose task it is to detect faulty gears and do the necessary repairs before turning them on. The agent is equipped with the following KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ representing its initial knowledge about the world:

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

The first concept inclusion (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.

To represent conditional effects of primitive actions and axioms whose truth can be sensed we use boolean combinations of *atoms*, i.e. ABox assertions where in place of individuals also variables are allowed. An *effect* is of the form φ/γ, where φ is a boolean combination of atoms and γ is a literal of the form $A(z), \neg A(z), r(z, z')$ and $\neg r(z, z')$. A *primitive action* is a pair of the finite sets eff and sense, where eff is a set of effects and sense a finite set of boolean combinations of atoms. For example consider the following actions:

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

$\texttt{turn-on}(x)$ with variable 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. No sensing result is provided. $\texttt{sense-on}(x)$ is a pure sensing action that represents the agent's ability to perceive whether $\mathit{On}(x)$ is true in the real world.

Semantically, a primitive ground action induces a binary relation on *epistemic interpretations* $(\mathcal{I}, \mathcal{W})$ which allow us to explicitly distinguish changes affecting the real world, represented by the interpretation $\mathcal{I}$, and changes to the knowledge state $\mathcal{W}$, which is a set of interpretations (i.e., possible worlds) over a common countably infinite domain. In our semantics we also assume that the agent knows the physical effects of its primitive actions. For instance, assume $\mathcal{K}$ as given above is all the agent knows initially about the world. Thus, it is initially 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, i.e. there is a least one possible world satisfying $\mathcal{K}$ where *gear* is an instance of $\exists \mathit{has\text{-}f}.\mathit{CritFault}$ and one where this is not the case. Consequently, the actual outcome of executing $\texttt{turn-on}(\mathit{gear})$ in $\mathcal{K}$ is also unknown. This can be expressed by the epistemic ABox assertion $\neg\mathsf{K}\mathit{On} \sqcap \neg\mathsf{K}\neg\mathit{On}(\mathit{gear})$, where the K is used here as a concept constructor, intuitively denoting the *known* instances. 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 epistemic ABox assertions $\mathsf{K}\exists \mathit{has\text{-}f}.\mathit{CritFault} \sqcup \mathsf{K}\neg\exists \mathit{has\text{-}f}.\mathit{CritFault}(\mathit{gear})$ and $\mathsf{K}\mathit{On} \sqcup \mathsf{K}\neg\mathit{On}(\mathit{gear})$ come to hold. A knowledge-based program describing the behaviour of an agent is then given as follows, where $\texttt{sense-f}(\mathit{gear}, x)$ is an additional sensing action for checking if *gear* has fault x and $\texttt{repair}(\mathit{gear}, x)$ an action for removing fault x of *gear*.

```
while $\neg\mathsf{K}(\forall \mathit{has\text{-}f}.\neg\mathsf{K}\mathit{Fault})(\mathit{gear})$
  pick$(x)$ : $\mathsf{K}\mathit{Fault}(x) \wedge \neg\mathsf{K}\mathit{has\text{-}f}(\mathit{gear}, x)? \wedge \neg\mathsf{K}\neg\mathit{has\text{-}f}(\mathit{gear}, x).\texttt{sense-f}(\mathit{gear}, x)$;
  if $\mathsf{K}\mathit{has\text{-}f}(\mathit{gear}, x)$ then $\texttt{repair}(\mathit{gear}, x)$ else continue;
end; $\texttt{turn-on}(\mathit{gear})$; $\texttt{sense-on}(\mathit{gear})$
```

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 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. 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. This can be expressed by the following $\mathcal{ALCOK}$-LTL formula:

$$\exists \mathit{has\text{-}f}.(\mathit{CritFault} \sqcap \neg\mathsf{K}\mathit{Fault})(\mathit{gear}) \rightarrow \Diamond\mathsf{K}\exists \mathit{has\text{-}f}.(\mathit{CritFault} \sqcap \neg\mathsf{K}\mathit{Fault})(\mathit{gear}).$$

In our semantics of actions it is not guaranteed that the TBox given in the initial KB always holds. However persistence of a TBox $\mathcal{T}$ in a program can be verified by checking validity of the $\mathcal{ALCOK}$-LTL formula $\Box\left(\bigwedge_{\varrho\in\mathcal{T}} \varrho\right)$.

3 Results

Unfortunately, it turns out that the verification problem is undecidable for an already quite small subset of our formalism. In our setting a state of the program consists of an epistemic interpretation and a program expression representing the program that remains to be executed. Thus, we end up with an infinite state transition system. As the source of undecidability we have identified the pick-operator for non-deterministic choice of argument, which may range over the whole countably infinite domain. However, we also have the positive result that decidability of the verification problem can be retained for a syntactically restricted fragment of the formalism where pick operators are extended with epistemic guards such that the agent is only allowed to choose an argument among the *known* individuals. We have devised an algorithm with a 2ExpSpace upper bound.

References

1. Baader, F., Lutz, C., Miličić, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. In: Proc. of AAAI 2005
2. Zarrieß, B., Claßen, J.: Verification of knowledge-based programs over description logic programs. In: Proc. of IJCAI-15 (2015)
3. Claßen, J., Lakemeyer, G.: Foundations for knowledge-based programs using ES. In: Proc. of KR 2006
4. Claßen, J., Lakemeyer, G.: A logic for non-terminating Golog programs. In: Proc. of KR 2008
5. De Giacomo, G., Lespérance, Y., Levesque, H.J.: ConGolog, a concurrent programming language based on the situation calculus. AIJ 121(1–2), 109–169 (2000)
6. De Giacomo, G., Lespérance, Y., Patrizi, F.: Bounded situation calculus action theories and decidable verification. In: Proc. of KR 2012
7. De Giacomo, G., Lespérance, Y., Pearce, A.R.: Situation calculus based programs for representing and reasoning about game structures. In: Proc. of KR 2010
8. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. AIJ 100(1-2), 225–274 (1998)
9. Levesque, H.J., Lakemeyer, G.: The Logic of Knowledge Bases. MIT Press (2001)
10. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming 31(1–3), 59–83 (1997)
11. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence 4, pp. 463–502. (1969)
12. Reiter, R.: 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 pp. 359–380 (1991)
13. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
14. Reiter, R.: On knowledge-based programming with sensing in the situation calculus. ACM Trans. Comput. Log. 2(4), 433–457 (2001)
15. Zarrieß, B., Claßen, J.: Verifying CTL* properties of Golog programs over local-effect actions. In: Proc. of ECAI 2014
16. Zarrieß, B., Claßen, J.: Verification of knowledge-based programs over description logic actions. LTCS-Report 15-10,
    See `http://lat.inf.tu-dresden.de/research/reports.html`.