Agent Logics as Program Logics: Grounding KARO

Koen V. Hindriks[1] and John-Jules Ch. Meyer[2]

[1] Nijmegen Institute for Cognition and Information, Radboud University Nijmegen, The Netherlands, k.hindriks@nici.ru.nl
[2] Utrecht University, Department of Information and Computing Sciences, The Netherlands, jj@cs.uu.nl

Abstract. Several options are available to relate agent logics to computational agent systems. Among others, one can try to find useful executable fragments of an agent logic or use a model checking approach. In this paper, an alternative approach is explored based on the view that *an agent logic is a program logic*. Using the same starting point, one of the established agent logics, we ask instead if it is possible to construct a programming language for that agent logic. We show that the programming language and the agent logic are formally related by constructing a denotational semantics. As a result, the agent logic can be used as as a design tool to specify and verify the corresponding agent programs.
In particular, we construct an agent programming language that is formally related to the KARO agent logic. The KARO logic is an agent logic that builds on top of dynamic logic. The approach is based on mapping worlds in the modal semantics for KARO onto a state-based semantics. The state-based semantics can be used to define an operational semantics for KARO programs. In this way, we obtain a computationally grounded semantics for a significant part of the KARO logic, including the operators for knowledge or beliefs, motivational attitudes and belief revision actions of a rational KARO agent.

1 Introduction

Various agent logics have been proposed as models of so-called intelligent or rational agents. Some of the more influential ones have been those of [1–3]. These agent logics have played a guiding role in the research on the so-called strong notion of intelligent agent.

At the same time, the use of modal agent logics for engineering rational agents has been questioned (see for an extensive discussion also [4]). Agent logics have not been as useful as was hoped for in the specification and verification of agent systems. Consequently, the development of agent architectures and progamming languages has been inspired by operational systems such as the PRS system, which motivated the definition of AgentSpeak(L) [5].

The issue concerning the relation between agent logics and systems goes both ways: (i) can a suitable logical framework for reasoning about existing agent systems be constructed? and (ii) can a suitable operational framework for engineering agent systems be constructed that is related to an existing agent logic "in

the right way"? These questions have been referred to as the gap between theory and practice. In this paper, we will not pursue the first question (see e.g. [6, 7]) but only consider the second. There are good reasons to continue the search for a solution to bridge the gap. Agent systems are inherently complex due to the many different components and mechanisms such systems consist of. To be able to build such complex systems, at least two conditions should be met. First, it should be unambiguously clear how these components and mechanisms operate. One of the proven approaches to provide such an unambiguous interpretation is to provide a mathematical semantics that specifies the operation of an agent system. Second, tools for the design, specification and verification of agent systems should be available. This additionally requires the development of a design method including proof techniques for agent systems. Both conditions require that a precise relation between agent logics and systems is established.

As discussed in [4], various approaches to demonstrate the applicability of agent logics are available. Each of these approaches proposes techniques for relating agent logics to computational agent systems. For example, one method is to apply techniques for *directly executing* the logic. The goal of such methods is to find fragments that can be executed *efficiently*. Relevant work in this area is, for example, that of [8, 9]. A drawback, however, is that these techniques are applicable only to relatively small fragments of agent logics.

In this paper, an alternative approach is proposed. Using the same starting point, one of the established agent logics, we ask instead if it is possible to construct an agent programming framework for that agent logic. The idea is to construct an agent programming framework for engineering agent systems that is related to an agent logic in a way such that the logic can be used to prove properties of the agent system. The idea promoted here is that *agent logics are program logics*. Program logics are used as a design tool to specify and verify agent programs instead as directly executable frameworks. It is shown that this approach can be successfully applied to close the gap between theory and practice by constructing a programming framework for the KARO logic [3].

2 Grounding Agent Logics

Agent logics typically are *modal* logics. Since the associated possible world semantics is abstract, in [4], the issue has been framed as the question whether it is possible to *ground* the semantics of agent logics. As the authors explain, "there is usually no precise relationship between the abstract accessibility relations that are used to characterize an agent's state, and any concrete computational model." Such a precise relationship, in a mathematical sense, is exactly what we will be looking for in this paper. To achieve this objective, we briefly clarify what we mean by (i) an agent logic, (ii) a concrete computational system, and (iii) the relation between the two.

What is an Agent Logic? Any logic that is explicitly constructed as a tool for modeling rational agents and is rich enough to model agents that derive their

choice of action from their beliefs and motivations is an agent logic. In this paper, agent logics are single agent modal logics. Some of the better known agent logics such as [1–3] fit this definition and are reference examples. Agent logics based on *dynamic logic* (e.g. [3]) can be distinguished from agent logics based on *temporal logic* (e.g. [2]). Since in dynamic logic programs are explictly represented, agent logics based on dynamic logic provide a good starting point for our purposes.

What is a Concrete Computational System? To be able to construct a concrete computational system that relates to an agent logic in the right way, we start with providing a rather abstract definition and then move on to provide a concrete instance of this definition.

The main assumption that we introduce here is that *computational systems are state-based*. We take this to mean that the possible behavior of such a system can be uniquely predicted given its current state. Moreover, states are *extensional* and do not have an intensional flavour. That is, computational systems behave identically whenever they are in the same state at different times.

In fact, we will be more concrete and take a computational system to be a system that is programmed using a particular *programming language*. A programming language is a set of programming constructs to perform operations on specific data structures. Here, we are particularly interested in the set of data structures of agent systems, i.e. agent states, and associated agent programming frameworks for dynamically changing such states.

Programming frameworks typically have features for inspecting states of a computational system. For example, the language AgentSpeak(L) includes tests on the beliefs of an agent. The power that such tests have, however, may vary considerably. We distinguish so-called *poor tests* from *rich tests* (cf. [10]). Poor tests only allow inspection of the *current* state of a system whereas rich tests allow inspection of potential *future* states as well. Rich tests thus presuppose capabilities, called *look ahead facilities*, to perform tests on future states. In general, it is not clear how to provide a computational interpretation for rich tests. The second assumption we introduce is that *computational systems do not have look ahead facilities*. It will turn out, in fact, that the latter assumption will require the most effort in constructing a suitable agent programming framework. Computations thus are *local* in the sense that actions and tests are performed on the current state and do not require additional resources. The approach to define a state-based semantics is inspired by [11], but differs in its aim to derive a programming language from an agent logic. Any computational framework that introduces programming constructs to build computational systems and satisfies the two assumptions discussed is called a *programming framework*. As far as we know, all agent programming languages in the literature are programming frameworks in this sense (e.g. [5, 12, 13]).

How are Agent Logics and Computational Systems Related? The view promoted here is that agent logics are program logics for the specification and verification of agent programs. Agent logics provide declarative specifications of *what* an agent program should compute, whereas agent programming languages provide

operational specifications *how* to execute an agent program. The semantics of the first is provided by *possible worlds semantics. Structural operational semantics* is used here to define computation steps that provide an interpretation of the operations of an agent program [14]. The precise relation between the two is established by proving that both semantics are equivalent. Formally, a *denotational* semantics for programs is derived from the logical semantics of an agent logic and is shown to be equivalent to the operational semantics. This is a standard technique in programming theory to show that a logic can be used to verify (partial correctness) properties of programs. The approach differs from directly executing an agent logic since the logic itself is only used to *verify properties of an executable agent program.* It differs from a model checking approach in that the logic is not used to check whether execution traces of a program satisfy a specification, but instead is used to *axiomatically verify program properties based on the program text.*

3 Grounding KARO

To demonstrate the approach discussed in the previous section, an agent programming language for KARO is presented. The exposition of KARO is based on [3]. For additional explanation about the logic and some of the choices made in modeling rational agents in KARO the reader is referred to this paper. For an example application of KARO to specify agents see e.g. [9]. KARO is an integrated logical framework for modeling rational agents that offers a logical theory of how actions, information and motivation of agents are related. All of these notions are formalised in a modal logic that is a blend of dynamic and epistemic logic extended with operators that model several motivational attitudes of agents.

It is shown that a substantial fragment of KARO is the corresponding program logic for a particular agent programming language. KARO agent programs ground the KARO logic, and, consequently, the KARO logic can be applied as a specification and verification tool for KARO agent systems that are built using that programming language.

KARO is a very expressive logic in which several concepts are defined using non-local constraints. Such constraints refer to potential future states of a system and do not naturally fit into a state-based approach. In particular, KARO introduces three non-local constraints:

- In the definition of the concept of *ability*, a constraint is included to verify the ability of an agent in potential future states. Complex abilities of an agent are defined in terms of a dynamic operator $\langle do(\pi) \rangle$ that models the opportunities and results achieved by actions. Because of the dynamic operators in the definition of abilities that involve tests on potential future states, the ability operator $\mathsf{A}\pi$ has been excluded from the fragment that is discussed here. It is not clear by inspection of the logical semantics *how* the abilities of an agent change and a computational interpretation is not obtained by providing a state-based semantics for KARO.

– In the definition of the concept of a *goal*, quantification over actions is used to verify that a goal can be achieved via the execution of some plan by the agent. This verification involves reference to potential future states and some mechanism to test the possibility of performing an action in those states. It is not clear *how* to implement such look ahead facilities in a state-based approach. Instead of this non-local definition of goals in the KARO framework, a slightly weaker notion is defined that is implied by the KARO definition of goals, but not vice versa.
– In the definition of *commitments*, a test whether a plan can be executed is included. The actions `commit_to` to make a commitment and `uncommit` to remove a commitment similarly are defined in terms of the possibility to execute a plan. These definitions presuppose that KARO agents have look ahead facilities which do not straightforwardly translate into a computational semantics. Therefore, the commitment operator $\mathsf{Com}\pi$ is excluded here.

The computational complexity of KARO has been located primarily in the non-local constraints that are made use of in the definition of some operators. The logical semantics does not provide a clue on how to operationalize such constraints. It is an interesting question whether these constraints can somehow be operationalized in a state-based approach.

In the remainder, it is shown that the KARO fragment excluding operators that are defined by non-local constraints can be grounded. This fragment includes the modal operators $[do(\pi)]$ to represent the actions, $\mathsf{B}^k, \mathsf{B}^o$ to represent innate and observational knowledge, W to represent the wishes or desires, and C to represent the choices or goals of an agent. It also includes the informational actions `expand` φ and `contract` φ to add or remove a proposition φ from the observational knowledge base, and the action `select` φ to select φ as goal. Additionally, program constructs for tests, written as `confirm`φ, conditional composition `if_then_else_` and repetition `while_do_` are part of the KARO language. Due to space restrictions, the latter is not discussed. The label KARO will be used below to refer to this fragment.

A distinguishing feature of KARO is the distinction of various belief clusters. Due to space restrictions, we only discuss two of the four clusters. First, built-in knowledge represents the fixed, objective options an agent considers possible. Second, observational knowledge is based on perceptual sources and may change through time. The built-in knowledge is restricted to objective propositional formulae. This seems reasonable since built-in knowledge is supposed to pertain to the external world and not to states of the agent itself. Moreover, this restriction will allow us to model the relation between built-in knowledge and observational knowledge in the state-based semantics. In this context, we do not allow an agent to have wishes about the fixed built-in knowledge, and allow an agent only to have wishes to obtain knowledge through its perceptual apparatus.

Definition 1. (*KARO* Propositions)
Let $\mathcal{L}_0$ *be a classical propositional language, built from an infinite set At of* propositional atoms*, the connectives* $\neg, \wedge$ *and let* $\mathcal{L}_{obs}$ *be the standard extension*

of $\mathcal{L}_0$ to an epistemic modal language with epistemic operator B^o. Let Act be a set of atomic actions. *Then:*

The KARO language $\mathcal{L}$ is defined by:
– $At \subseteq \mathcal{L}$,
– if $\varphi, \psi \in \mathcal{L}$,$\chi \in \mathcal{L}_0$, then $\neg\varphi, \varphi \wedge \psi, \mathsf{B}^k\chi, \mathsf{B}^o\varphi \in \mathcal{L}$,
– if $\varphi \in \mathcal{L}$ and $\pi \in \Pi$, then $[do(\pi)]\varphi \in \mathcal{L}$,
– if $\varphi \in \mathcal{L}_{obs}$, then $\mathsf{W}\varphi \in \mathcal{L}$, $\mathsf{C}\varphi \in \mathcal{L}$.

The set of KARO programs Π is defined by:
– $Act \subseteq \Pi$,
– if $\varphi \in \mathcal{L}_0$, then $\texttt{expand}\,\varphi, \texttt{contract}\,\varphi \in \Pi$,
– if $\varphi \in \mathcal{L}_{obs}$, then $\texttt{select}\,\varphi \in \Pi$,
– if $\varphi \in \mathcal{L}, \pi_1, \pi_2 \in \Pi$, then $\texttt{confirm}\,\varphi$,
$\texttt{if}\,\varphi\,\texttt{then}\,\pi_1\,\texttt{else}\,\pi_2\,, \pi_1; \pi_2 \in \Pi$.

The semantics of KARO is defined as usual by Kripke structures.

Definition 2. (KARO Structure)
A KARO structure *M is a tuple $\langle W, R, B^k, B^o, D, C, V\rangle$ with:*
– W a non-empty set of worlds, typically denoted by w,
– R a partial function such that for each $a \in Act$, $R_a : W \rightharpoonup W$,
– $B^k, B^o \subseteq W \times W$ equivalence relations, such that $B^o \subseteq B^k$,
– $D \subseteq W \times W$,
– $C : W \to \wp(\mathcal{L}_{obs})$, a mapping of worlds to subsets of $\mathcal{L}_{obs}$, and
– V a truth function such that (s.t.) $V(p, w) \in \{1, 0\}$ for $p \in At$.

The knowledge operators are modal S5 operators as usual in agent logics. Knowledge obtained by perception always extends the agent's built-in knowledge since $B^o \subseteq B^k$. Observe that wishes modeled by the relation D may be inconsistent.

Definition 3. (KARO Semantics)
Let $M = \langle W, R, B^k, B^o, D, C, V\rangle$, $w \in W$ and $x \in \{k, o\}$.
The truth conditions for KARO propositions are defined by:
– $M, w \models p$ iff $V(p, w) = 1$,
– $M, w \models \neg\varphi$ iff $M, w \not\models \varphi$,
– $M, w \models \varphi \wedge \psi$ iff $M, w \models \varphi$ and $M, w \models \psi$,
– $M, w \models \mathsf{B}^x\varphi$ iff $M, w' \models \varphi$, $\forall w'$ s.t. wB^xw',
– $M, w \models [do(\pi)]\varphi$ iff $M, w' \models \varphi$, $\forall w'$ s.t. $wR_\pi w'$,
– $M, w \models \mathsf{W}\varphi$ iff $M, w' \models \varphi$, $\forall w'$ s.t. wDw',
– $M, w \models \mathsf{C}\varphi$ iff $\varphi \in C(w)$.
The meaning of KARO programs is defined by:
R_a,

$$
\begin{aligned}
R_{\texttt{confirm}\,\varphi} &= \{(w, w) \mid M, w \models \varphi\},\\
R_{\texttt{if}\,\varphi\,\texttt{then}\,\pi_1\,\texttt{else}\,\pi_2} &= (R_{\pi_1} \cap ([\![\varphi]\!] \times W)) \cup\\
&\quad\ (R_{\pi_2} \cap ([\![\neg\varphi]\!] \times W)),\\
R_{\pi_1;\pi_2} &= R_{\pi_1} \circ R_{\pi_2}.
\end{aligned}
$$

where $[\![\varphi]\!]_M = \{w \mid M, w \models \varphi\}$*, written* $[\![\varphi]\!]$ *when the structure* M *is clear from the context, and* $R \circ S = \{(a, c) \mid \exists b(aRb \wedge bSc)\}$.

Note that all KARO programs are deterministic since the transition relation R for actions is a function. In KARO, complex motivational attitudes such as goals are defined in terms of the basic motivational operators W and C. KARO defines goals as those wishes that are (a) *selected* by the agent, (b) *not (yet) fulfilled* and (c) *implementable*. Condition (c) is defined by quantifying over actions and involves tests on potential future states. It does not satisfy the constraints on state-based systems. A weaker version, by dropping (c), can be defined, however. The goal operator G is then defined as: $\mathsf{G}\varphi \equiv \mathsf{W}\varphi \wedge \neg\varphi \wedge \mathsf{C}\varphi$. Note that tautologies cannot be goals and neither are goals closed under implication.

Changing One's Mind One interesting feature of KARO is that it incorporates specific instances of atomic actions to change the knowledge or motivations of an agent. KARO thus not only formalizes the logic of propositional attitudes but also provides a theory about how an agent can change its knowledge or motivations by performing mental "actions".

Since mental actions do not change the (external) world, a natural way in modal semantics to model these actions is to change the knowledge or motivational components B^k, B^o, D, C in a KARO structure instead of a world w (see for extensive discussion [3]). This type of action semantics extends the component R in a structure to apply to pairs (M, w) as well as worlds w. Actions thus are interpreted as structure transformers. Some notation is introduced to facilitate the semantic definition of the KARO actions. Given an equivalence relation S, the equivalence class of w is defined as: $[w]_S = \{w' \mid wSw'\}$. Due to space restrictions, the semantics of `contract` φ is not discussed.

Definition 4. (Semantics of KARO Actions)
Let $M = \langle W, R, B^k, B^o, D, C, V\rangle$ *be a structure and* $w \in W$*. R is extended to a structure transforming semantics as follows:*

$$R_a(M, w) = (M, R_a(w)),$$

$$R_{\texttt{expand}\,\varphi}(M, w) = \begin{cases} (\langle W, R, B^k, B^{o'}, D, C, V\rangle, w) \text{ such that:} & \\ \quad [w']_{B^{o'}} = [w']_{B^o} \cap [\![\varphi]\!], \forall w' \in [w]_{B^k} \cap [\![\varphi]\!], & \\ \quad [w']_{B^{o'}} = [w']_{B^o} \cap [\![\neg\varphi]\!], \forall w' \in [w]_{B^k} \cap [\![\neg\varphi]\!], \text{ and} & \\ \quad [w']_{B^{o'}} = [w']_{B^o}, \forall w' \notin [w]_{B^k}, & \text{if } M, w \models \varphi, \\ \emptyset & \text{otherwise.} \end{cases}$$

$$R_{\texttt{select}\,\varphi}(M, w) = \begin{cases} (\langle W, R, K, D, C', V\rangle, w) \text{ such that:} & \\ \quad C'(w) = C(w) \cup \{\varphi\}, & \\ \quad C'(w') = C(w'), \forall w' \neq w, & \text{if } M, w \models \mathsf{W}\varphi, \\ \emptyset & \text{otherwise.} \end{cases}$$

The extension for compound constructs can be defined analogously.

The semantics for `expand` φ ensures that an agent after (successfully) performing it knows φ, i.e. $\mathsf{B}^o\varphi$, still knows what it knew before, and doesn't change

anything when the agent already knew φ. This semantics validates the propositions $\varphi \rightarrow [do(\texttt{expand}\ \varphi)]\mathsf{B}^o\varphi$, $\mathsf{B}^o\psi \rightarrow [do(\texttt{expand}\ \varphi)]\mathsf{B}^o\psi$, and $\mathsf{B}^o\varphi \rightarrow (\mathsf{B}^o\psi \leftrightarrow [do(\texttt{expand}\ \varphi)]\mathsf{B}^o\psi)$, for $\varphi, \psi \in \mathcal{L}_0$, which are expected properties of adding true information to ones knowledge.

A State-Based Semantics for KARO A programming language with KARO as its associated program logic must account for each of the components in a KARO structure. Except for the relation R all components will be represented in the state of a KARO agent program. These states, additionally, must be extensional, i.e. they can be uniquely identified by their syntactic content.

To avoid overly complex states, an additional axiom on top of the KARO axioms is introduced to relate the knowledge and wishes of an agent. As for knowledge, we assume that an agent is also able to introspect its motivational attitudes. If an agent has a wish, consequently, it knows it has a wish. Formally, we introduce the axioms $\mathsf{B}^o\mathsf{W}\varphi \leftrightarrow \mathsf{W}\varphi$ and $\mathsf{B}^o\neg\mathsf{W}\varphi \leftrightarrow \neg\mathsf{W}\varphi$. From now on, we will assume that these axioms are part of the KARO logic.

The assignment function V and possible worlds W in the modal semantics are replaced by so-called *world states* $v \subseteq At$ representing the external world in the state-based semantics. A corresponding world state can be defined for each $w \in W$ by $v = \{p \mid V(p, w) = 1\}$. The informational components B^k and B^o are replaced by *knowledge bases*. Corresponding knowledge and observational bases can be defined for each $w \in W$ respectively by $k = \{\varphi \in \mathcal{L}_0 \mid M, w \models \mathsf{B}^k\varphi\}$ and $o = \{\varphi \in \mathcal{L}_0 \mid M, w \models \mathsf{B}^o\varphi\}$. The motivational components D and C are replaced by a set of wishes $d \subseteq \mathcal{L}_{obs}$ closed under logical consequence and a set of choices $c \subseteq \mathcal{L}_{obs}$. Finally, the function R that provides the meaning of actions is quite straightforwardly replaced by a similar function R^c defined on KARO states instead. A KARO state can be viewed as the agent's internal, mental state.

Definition 5. (State-Based KARO Structure)
A state-based KARO structure M^c *is a tuple* $\langle W^c, R^c\rangle$ *with:*

- W^c *a set of states of the form* (v, k, o, d, c)*, with* v *a world state,* k, o *knowledge bases such that* $\models_v k$*,* $\models_v o$ *and* $o \models k$*,* $d, c \subseteq \mathcal{L}_{obs}$ *a set of wishes and choices respectively, and such that* W^c *satisfies the following closure conditions: if* $(v, k, o, d, c) \in W^c$*, then:*
  - $(v', k, o, d, c) \in W^c$ *for all* v' *such that* $\models_{v'} k$*,*
  - $(v', k, o, d, c) \in W^c$ *for all* v' *such that* $\models_{v'} o$*,*
  - $(v', k, o', d, c) \in W^c$ *for all* v', o' *such that* $\models_{v',o'} d$*.*
- R^c *a partial function such that for each* $a \in Act$ *or* $a \in \{\texttt{expand}\ \varphi, \texttt{select}\ \varphi\}$*,* $R^c_a : W^c \rightharpoonup W^c$*.*

The definition of states clarifies the nature of the states in the state-based semantics. States are tuples of various databases which are the data structures that a KARO program operates on. To ensure a proper relation with the modal semantics, these components need to be related in the right way, which explains

the various constraints on states. These constraints correspond to e.g. the relations between the accessibility relations in the modal semantics.

The truth conditions using state-based structures are defined next. The semantic clauses for atomic actions, tests and compound actions are the same as in definition 3 and are not repeated. Typically, states (v, k, o, d, c) are denoted by s, s' and we write $s[v'/v]$, $s[k'/k]$, ... to denote the state that results from replacing v by v', k by k', etc. Note the subscript c to distinguish $\models_c$ from the standard relation $\models$.

Definition 6. (State-Based Semantics for KARO)
Let $M^c = \langle W^c, R^c \rangle$ and $s = (v, k, o, d, c) \in W^c$. Then:
The truth conditions for KARO propositions φ are defined by:
$- M^c, s \models_c p$ *iff* $p \in v$,
$- M^c, s \models_c \neg\varphi$ *iff* $M^c, s \not\models_c \varphi$,
$- M^c, s \models_c \varphi \wedge \psi$ *iff* $M^c, s \models_c \varphi$ *and* $M^c, s \models_c \psi$,
$- M^c, s \models_c [do(\pi)]\varphi$ *iff* $M^c, s' \models_c \varphi$, $\forall s'$ *s.t.* $sR^c_\pi s'$,
$- M^c, s \models_c \mathsf{B}^k\varphi$ *iff* $M^c, s[v'/v] \models_c \varphi$, $\forall v'$ *s.t.* $\models_{v'} k$,
$- M^c, s \models_c \mathsf{B}^o\varphi$ *iff* $M^c, s[v'/v] \models_c \varphi$, $\forall v'$ *s.t.* $\models_{v'} o$,
$- M^c, s \models_c \mathsf{W}\varphi$ *iff* $M^c, s[v'/v, o'/o] \models_c \varphi$,
$\forall v', o'$ *s.t.* $\models_{v',o'} d$,
$- M^c, s \models_c \mathsf{C}\varphi$ *iff* $\varphi \in c$.
where $\models_v \varphi$ is $v \models \varphi$ and $\models_{v,o} \varphi$ is defined by the first three clauses above and the clause for B^o.
The semantics of KARO programs π is defined by:
$R^c_{\mathtt{expand}\,\varphi} = \{(s, s[exp(o, \varphi)/o']) \mid M^c, s \models \varphi\}$,
$R^c_{\mathtt{select}\,\varphi} = \{(s, s[c \cup \{\varphi\}/c]) \mid M^c, s \models \mathsf{W}\varphi\}$
$R^c_{\mathtt{confirm}\,\varphi} = \{(s, s) \mid M^c, s \models \varphi\}$.
with $exp(o, \varphi)$ defined as $\{\psi \mid \forall v'(\models_{v'} o \wedge \varphi \Rightarrow \models_{v'} \psi)\}$.

The modal and state-based semantics are equivalent, i.e. the expressive power is not reduced by introducing a state-based semantics.

Theorem 7. (Equivalence of $\models$ and $\models_c$ for KARO)
The standard and the state-based semantics for KARO are equivalent. That is, assuming the set of propositional atoms At is infinite, for any $\varphi \in \mathcal{L}$:

$$\models \varphi \quad \textit{iff} \quad \models_c \varphi$$

Proof. We give a sketch of the proof, the full proof is available in the full paper. First, observe that the structure-transforming action semantics in the standard modal semantics can be replaced with a standard Kripke semantics, by defining a super structure with worlds w_M for each pair (M, w). The right to left implication then is proved by a straightforward mapping from state-based structures to standard structures. For the left to right implication, use the finite model property for the (super structure) Kripke semantics to show the equivalence with the state-based semantics. We need to prove that if $M^c, s \not\models \varphi$, then also $M, w \not\models \varphi$. Since the truth of φ can only depend on a finite number of propositional atoms, an

infinite number of atoms remains that can be used as names for possible worlds in the standard structure to keep track of these worlds in a state-based structure. Using this observation, then set up a correspondence between possible worlds and states to define a state-based structure and show their equivalence. □

Note that no restrictions are imposed on the KARO language in theorem 7. The state-based semantics is defined for arbitrary formulae of the KARO language. To provide an operational interpretation, as discussed, however, we need to restrict the tests that are allowed in programs. The KARO language is a rich test version of a dynamic agent logic. To avoid the introduction of undecidable look ahead facilities into the programming framework, a poor test variant of KARO is introduced. The tests that can be allowed are those that can be evaluated in the current state. Consequently, propositions without occurrences of dynamic operators $[do(\pi)]$, called *intentional propositions* (since they refer to intentional or mental states), can be used as tests since KARO states contain the information needed to evaluate such propositions. We use $\mathcal{L}_i$ to denote the set of intentional propositions; note that $\mathcal{L}_{obs} \subset \mathcal{L}_i$. The fragment of KARO with restricted tests $\texttt{confirm}\,\varphi$ such that φ is an intentional proposition is called *poor test* KARO and denoted by $\mathcal{L}^p$. $\mathcal{L}^p$ is strictly less expressive as $\mathcal{L}$.

The computational interpretation for intentional propositions, and, consequently, poor tests, is provided by the state-based semantic clauses for the non-dynamic operators. The definition below provides a computational interpretation since it identifies concrete data structures on which operations can be performed by a computer. Moreover, the computational interpretation is state-based and thus fits our definition of a computational system.

Definition 8. (Computational Interpretation of Poor Tests)
Let $s = (v, k, o, d, c)$ be a state such that $\models_v k$, $\models_v o$, and $o \models k$. Then the truth conditions for intentional propositions are defined by:
$- \models_s p$ *iff* $p \in v$,
$- \models_s \neg\varphi$ *iff* $\not\models_s \varphi$,
$- \models_s \varphi \wedge \psi$ *iff* $\models_s \varphi$ *and* $\models_s \psi$,
$- \models_s \mathsf{B}^k\varphi$ *iff* $\models_{s[v'/v]} \varphi$, $\forall v'$ *s.t.* $\models_{v'} k$,
$- \models_s \mathsf{B}^o\varphi$ *iff* $\models_{s[v'/v]} \varphi$, $\forall v'$ *s.t.* $\models_{v'} o$,
$- \models_s \mathsf{W}\varphi$ *iff* $\models_{s[v'/v,,o'/o]} \varphi$, $\forall v', o'$ *s.t.* $\models_{v',o'} d$,
$- \models_s \mathsf{C}\varphi$ *iff* $\varphi \in c$.

In case an intentional proposition is of the form $X\varphi$ with X some non-dynamic operator and $\varphi \in \mathcal{L}_0$ a state proposition, we have $\models_{v,k,o,d,c} \varphi$ iff $x \models \varphi$ for $x = k, o, d, c$ respectively. For example, it is easy to show that $\models_{v,k,o,d,c} \mathsf{W}\varphi$ iff $d \models \varphi$.

Lemma 9. *Let $\varphi \in \mathcal{L}_0$ be a state proposition. Then we have:*

$$\begin{array}{ll} \models_{v,k,o,d,c} \mathsf{B}^o\varphi & \textit{iff } o \models \varphi, \\ \models_{v,k,o,d,c} \mathsf{W}\varphi & \textit{iff } d \models \varphi, \\ \models_{v,k,o,d,c} \mathsf{C}\varphi & \textit{iff } \varphi \in c. \end{array}$$

Proof. The last item follows immediately from the semantic definition of C*. The proof of the other three statements is similar. We prove the case for* $\mathsf{W}\varphi$*. By definition, we have that* $\models_{v,k,o,d,c} \mathsf{W}\varphi$ *iff for all* v', k', o' *such that* $\models_{v',k',o'} d$ *we have* $\models_{v',k',o',d,c} \varphi$*. Since* $\varphi \in \mathcal{L}_0$ *is a state proposition, its truth evaluation depends only on the world state component* v'*. Now suppose that* $d \not\models \varphi$*. Then there is a model of* d *in which* d *is true, but* φ *is not true. I.e., there are* v', k', o' *such that* $\models_{v',k',o'} d$ *and* $\not\models_{v',k',o'} \varphi$*, contrary to the assumption that* $\models_{v,k,o,d,c} \mathsf{W}\varphi$*.* □

In the remainder of this section, we use the computational interpretation of intentional propositions to provide a transition semantics for KARO programs. A transition semantics, defined in terms of a computation step relation $\longrightarrow$, provides a computational semantics for KARO programs.

Interestingly, in the transition semantics for KARO programs transition rules are required for the specific actions `expand` φ and `select` φ. Agents that explicitly represent their knowledge and motivational attitudes in their mental states need capabilities to modify these mental structures. In this respect, KARO contributes to an understanding of such capabilities, for both informational as well as motivational attitudes. Of course, agent logics allow for the *specification* of additional actions and the fact that KARO can be used as a program logic for KARO programs, proven below, shows that such specifications can be usefully applied to build agent programs.

In the transition semantics a transition function $\mathcal{T}(a, s)$ maps actions and a state to their successor state. This function must respect the constraints on states from definition 5. This translates into the following condition on transition functions $\mathcal{T}$: if $\mathcal{T}(a, v, k, o, d, c) = (v', k', o', d', c')$, then we have $\models_{v'} k', \models_{v'} o'$ and $o' \models k'$. KARO states in state-based structures are tuples (v, k, o, d, c) such that $\models_v k, \models_v o$ and $o \models k$ to ensure that beliefs of an agent are always true in the current world state and the belief clusters k and o are properly related. In the transition semantics, this relation between world states and knowledge bases also needs to be enforced.

If a KARO program starts in an initial state that satisfies this condition, then the constraint on transition relations ensures that states during the execution of the program invariantly satisfy this condition. Next, the transition semantics is defined. The rules for sequential composition and the `if_then_else_` are standard and not included here (cf. [14]).

Definition 10. (Transition Semantics for KARO Programs)
Let s, s' *be states,* $\mathcal{T}$ *a transition function,* $\varphi \in \mathcal{L}_i$*,* $\psi \in \mathcal{L}_0$ *and* $\chi \in \mathcal{L}_{obs}$*. We use the symbol* E *to denote successful program termination. Then the transition semantics for KARO programs is defined by:*

$$\frac{\mathcal{T}(a,s) = s'}{\langle a, s\rangle \longrightarrow \langle E, s'\rangle} \qquad \frac{\models_s \varphi}{\langle \texttt{confirm}\,\varphi, s\rangle \longrightarrow \langle E, s\rangle}$$

$$\frac{\models_s \psi}{\langle \texttt{expand}\,\psi, s\rangle \longrightarrow \langle E, s[exp(o,\psi)/o]\rangle}$$

$$\frac{\models_s \mathsf{W}\chi}{\langle \texttt{select}\,\chi, s\rangle \longrightarrow \langle E, s[c \cup \{\chi\}/c]\rangle}$$

The operational semantics for KARO programs defines the input-output relation on KARO states for arbitrary KARO programs in terms of the transitive closure $\longrightarrow^*$ of the transition relation $\longrightarrow$.

Definition 11. (Operational Semantics for KARO Programs)
The operational semantics for KARO programs is defined by:

$$\mathcal{O}(\pi)(s) = s' \text{ for } s' \text{ such that } \langle \pi, s\rangle \longrightarrow^* \langle E, s'\rangle$$

Note that the definition of the operational semantics is well-defined since KARO programs are deterministic.

The denotational semantics for KARO programs is derived from the logical semantics for KARO. In the definition of the denotational semantics the state-based semantics for KARO is used, which is justified by theorem 7. To provide a definition of the denotational semantics, we need to fix an interpretation of atomic actions. It will be convenient to ensure that this interpretation is equivalent to the one that is fixed in the transition semantics by the transition function $\mathcal{T}$ since we need this later on to prove the equivalence of both types of semantics.

Definition 12. ($\mathcal{T}$-compatible)
Let $M^c = \langle W^c, R^c\rangle$ be a state-based KARO structure. We say that M^c and the accessibility relation R^c are $\mathcal{T}$-compatible *if the following condition is satisfied: $sR^c_a s'$ iff $\mathcal{T}(a,s) = s'$ for all KARO actions.*

The denotational semantics can now be defined using the concept of a $\mathcal{T}$-compatible KARO structure; that is, a unique, well-defined and compositional semantic function can be defined for KARO programs.

Definition 13. (Denotational Semantics for KARO Programs)
Let $M^c = \langle W^c, R^c\rangle$ be a state-based, $\mathcal{T}$-compatible KARO structure. Then the denotational semantics for KARO programs is defined by:

$$
\begin{aligned}
[\![a]\!](s) &= \begin{cases} s' & \text{, if } sR_a^c s', \\ \textit{undefined} & \text{, otherwise,} \end{cases} \\
[\![\texttt{expand}\ \varphi]\!](s) &= \begin{cases} s[exp(o,\varphi)/o'] & \text{, if } M^c, s \models_c \varphi, \\ \textit{undefined} & \text{, otherwise} \end{cases} \\
[\![\texttt{select}\ \varphi]\!](s) &= \begin{cases} s[c \cup \{\varphi\}/c] & \text{, if } M^c, s \models_c \mathsf{W}\varphi, \\ \textit{undefined} & \text{, otherwise} \end{cases} \\
[\![\texttt{confirm}\,\varphi]\!](s) &= \begin{cases} s & \text{, if } M^c, s \models_c \varphi \\ \textit{undefined} & \text{, otherwise} \end{cases} \\
[\![\texttt{if}\ \varphi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\,]\!](s) &= \begin{cases} [\![\pi_1]\!](s) & \text{, if } M^c, s \models_c \varphi, \\ [\![\pi_2]\!](s) & \text{, otherwise} \end{cases} \\
[\![\pi_1; \pi_2]\!](s) &= [\![\pi_2]\!]([\![\pi_1]\!](s)),
\end{aligned}
$$

It is easy to show that $[\![\_]\!]$ is well-defined. The next step is to show that the denotational and operational semantics are equivalent. This provides the precise relationship of the computation step relation $\longrightarrow$ and the logical semantics of KARO that we were looking for and shows that the logic can be used as a program logic to verify properties of KARO programs.

Theorem 14. (Denotational Equivalent to Operational Semantics)
The denotational and operational semantics of KARO programs are equivalent, i.e. $[\![\pi]\!](s) = \mathcal{O}(\pi)(s)$.

Proof. Use induction on the structure of programs. □

The equivalence of the denotational and operational semantics shows that KARO has an application in the verification of KARO programs, in particular, to prove *partial correctness properties* of KARO programs. This fact is expressed mathematically in the following corollary:

Corollary 15. (Proving Partial Correctness Properties)
Let π *be a poor test KARO program,* $\varphi, \psi \in \mathcal{L}_i$, *and* $\mathcal{M}_{\mathcal{T}}$ *be the set of all* $\mathcal{T}$*-compatible KARO structures. Then we have:*

$$
\begin{gathered}
\forall s, s' : \ \textit{if} \ \models_s \varphi \ \textit{and} \ \langle \pi, s \rangle \longrightarrow^* \langle E, s' \rangle \ \textit{, then} \ \models_{s'} \psi \\
\textit{iff} \\
\models^{\mathcal{M}_{\mathcal{T}}} \varphi \rightarrow [\pi]\psi
\end{gathered}
$$

Proof. Immediate from theorem 7 and 14. □

By using the techniques explored in the previous section, we have been able to define a state-based programming framework that corresponds to a substantial fragment of the original KARO logic. In other words, the KARO logic is a program logic for the KARO programming framework introduced in this section.

The KARO programming framework introduced can be compared to existing agent programming languages in the literature. It is instructive, for example, to compare the transition semantics of the KARO programming framework with

similar approaches. For example, this style of semantics has been proposed for the closely related programming languages AgentSpeak(L), ConGolog, and 3APL (cf. e.g. [15]). Even though the KARO programming framework introduced here does not include all aspects of these languages (notably concurrency is absent), it also includes features that are not present in one of the mentioned programming languages. The most important distinguishing features are the presence of *declarative* motivational attitudes and the definition of specific actions to change the agent's mind. For example, in AgentSpeak(L) the structures called intentions or plans are similar to the KARO programs and do not have a declarative interpretation. Instead, the motivational components in KARO program states are declarative. Another way to illustrate the same point is the observation that the corresponding program logic for 3APL proposed in [7] includes an operator for beliefs but not for motivational operators as the KARO logic does.

4 Conclusion

In this paper, we have explored formal techniques for relating agent logics to agent programming frameworks. We showed that this is a viable approach which clarifies the use of agent logics in the practice of agent engineering as specification and verification tools. The approach has been illustrated by constructing a programming framework for KARO, an agent logic that extends dynamic logic.

One of the benefits of our approach is that it explores the space of programming languages from a logical point of view. Taking an agent logic as our starting point, we showed what a programming language related to that logic looks like. This clarifies at least partly which agent logics are related to which agent programming languages.

A precise relationship between agent logics and agent programming frameworks will clarify what an agent programming language should be like from a logical agent perspective. The precise analysis of states of agents and the associated operations, moreover, facilitates a comparison between various agent logics as well as between various agent programming frameworks.

In the paper, it is shown that the approach based on the view that agent logics are useful as program logics resolves at least part of the gap between theory and practice. As discussed in [4], various approaches to show the usefulness of agent logics are available. These approaches are not each as practical as the other. Directly executing agent logics, for example, has to face the high computational complexity of agent logics. Model checking approaches suffer from other problems, as highlighted in [4]. By taking another view and viewing agent logics not as executable frameworks but as program logics we were able to circumvent some of these problems.

There have been other attempts to provide a computational grounding of KARO. In [9] a reduction approach is presented, based on translating KARO to first order logic. Alternatively, a translation of a fragment of the KARO logic to a combination of branching time logic CTL and a modal S5 logic, has been proposed (cf. [9]). In this approach, the *core* - as it is called - of the KARO frame-

work is first translated into another logical formalism, to obtain an executable fragment. However, the fragment that can be translated into executable form is smaller and does not include multiple belief clusters, the wishes and choices of an agent, nor the specific KARO actions that are included here.

References

1. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence **42** (1990) 213–261
2. Rao, A., Georgeff, M.: Decision Procedures for BDI Logics. Journal of Logic and Computation **8**(3) (1998) 293–343
3. van der Hoek, W., van Linder, B., Meyer, J.-J.Ch.: An Integrated Modal Approach to Rational Agents. In Wooldridge, M., ed.: Foundations of Rational Agency. Kluwer, Dordrecht (1999) 133–168
4. van der Hoek, W., Wooldridge, M.: Towards a Logic of Rational Agency. Logic Journal of the IGPL **11**(2) (2003) 133–157
5. Rao, A.S.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In van der Velde, W., Perram, J., eds.: Agents Breaking Away. Number 1038 in LNAI, Springer (1996) 42–55
6. de Boer, F., Hindriks, K., van der Hoek, W., Meyer, J.-J.Ch.: A Verification Framework for Agent Programming with Declarative Goals. Accepted for the Journal of Applied Logic (2006)
7. Hindriks, K., de Boer, F., van der Hoek, W., Meyer, J.-J.Ch.: A Programming Logic for part of the Agent Language 3APL. In Rash, J., ed.: Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems. Number 1871 in LNCS, Springer (2001) 78–89
8. Fisher, M.: A Survey of Concurrent MetateM. In: Proceedings of the First International Conference on Temporal Logic (ICTL). Number 827 in LNCS, Springer (1994) 480–505
9. Hustadt, U., Dixon, C., Schmidt, R., Fisher, M., Meyer, J.-J.Ch., van der Hoek, W.: Reasoning about Agents in the KARO Framework. In Bettini, C., Montanari, A., eds.: Proc. of the Eighth Int. Symposium on Temporal Representation and Reasoning. (2001) 206–213
10. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
11. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Knowledge-based programs. Distributed Computing **10**(4) (1997) 199–225
12. Bordini, R., Dastani, M., Dix, J., Seghrouchni, A.E.F., eds.: Multi-Agent Programming: Languages, Platforms and Applications. Springer (2005)
13. de Giacomo, G., Lespérance, Y., Levesque, H.J.: ConGolog, a Concurrent Programming Language Based on the Situation Calculus. Artificial Intelligence **121**(1-2) (2000) 109–169
14. Plotkin, G.D.: A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University (1981)
15. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.Ch.: A Formal Embedding of AgentSpeak(L) in 3APL. In Antoniou, G., Slaney, J., eds.: Advanced Topics in Artificial Intelligence. Number 1502 in LNAI. Springer (1998) 155–166