

Beliefs in Agent Implementation

Laurens Winkelhagen, Mehdi Dastani and Jan Broersen

{lwinkelh,mehdi,broersen}@cs.uu.nl
Institute of Information and Computing Sciences
Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Abstract. This paper extends a programming language for implementing cognitive agents with the capability to explicitly represent beliefs and reason about them. In this programming language, the beliefs of agents are implemented by modal logic programs, where beliefs are represented by explicit modal operators. A distinction is made between a belief base language that can be used to represent an agent's beliefs, and a belief query language that can be used to express queries to the agent's belief base. We adopt and modify a proof procedure that decides if a belief query formula is derivable from the belief base of an agent. We show that the presented proof procedure is sound.

1 Introduction

This paper presents an extension of the agent programming language 3APL [1]. This programming language provides data structures such as beliefs, goals, plans and reasoning rules, as well as programming constructs to manipulate these data structures. Examples of such constructs are updating the beliefs, planning a goal and executing a plan. In multi-agent settings, agents are assumed to have the ability to communicate. Several specifications have been proposed to facilitate agent communication, amongst these, the FIPA¹ standards are important to 3APL. According to FIPA, agents can communicate by sending each other messages that contain, amongst other things, a communicative act and the message content. The communicative acts specified by FIPA [2] require that a message with a certain performative can only be sent if certain belief formulae, the preconditions of the act, hold. This necessitates capabilities in the agent programming language to implement agents that can reason with beliefs. These beliefs can be about the beliefs of the agent that wants to send the message, or about the beliefs of the receiver of the message. For example, the INFORM act specifies that the sender believes that receiver has no beliefs about the message content.

The programming language 3APL implements agent belief in terms of a set of Horn clause formulae and uses a Prolog-engine to verify if the agent has a certain belief. However, 3APL lacks (1) the possibility to represent beliefs of agents about their own beliefs and the beliefs of other agents. It also lacks (2) the possibility to reason with these beliefs.

¹ the Foundation for Intelligent Physical Agents: <http://www.fipa.org/>

In this paper we show how we can extend the language of 3APL and the underlying Prolog mechanism in order to implement these two features. We take existing work [3, 4] on adding modal reasoning capabilities to logic programming, and investigate how to use this in combination with 3APL. We extend the programming language with an explicit modal operator of belief and provide a proof method that allows 3APL agents to function correctly with this modal operator. We show that this proof method is sound.

In sections 2 and 3 we give a quick introduction to 3APL and modal logic programming respectively. The bulk of the paper is section 4 in which we combine one approach to modal logic programming with the 3APL programming language, first discussing the syntactical changes, then the semantical interpretations and finally giving a soundness proof for these semantics. Section 5 is the conclusion, in which we will also point out some areas for further research.

2 3APL

Like other agent programming languages, 3APL provides data structures and programming constructs to manipulate them. Since 3APL is designed to implement cognitive agents, its data structures represent cognitive concepts such as beliefs, goals, plans, and reasoning rules. These data structures can be modified by programming constructs, also called deliberation operations, such as selecting a goal, applying a planning rule to it, or executing a plan. These operations constitute the deliberation process of individual agents which can be viewed as the agent interpreter. The formal syntax and semantics of 3APL are given in [1]. In this section, we will explain the ingredients of this programming language and give the formal definition of only those ingredients that are relevant for the research problem of this paper, i.e. the belief of 3APL agents.

The beliefs and goals are logical formulae representing the current and desirable state of the world, respectively. The goals of the agents are represented as logical formulae which are conjunction of atomic ground² formulae. The beliefs of 3APL agents can be specified by formulae in the following belief base language:

Definition 1. (*base language and belief base language*) Let Var , $Func$ and $Pred$ be the sets of domain variables, functions and predicates, respectively. Let $Term$ be the set of terms constructed from variables and functions in usual way. The base language \mathcal{L} is defined as the set of atomic formulae built on terms $Term$ and predicates $Pred$ in the usual way. Let $\psi \in \mathcal{L}$ be ground (atomic) formulae of the base language and let $\phi, \phi_1, \dots, \phi_n \in \mathcal{L}$. The belief base language \mathcal{L}_{BB} , which represents the beliefs of agents, is defined as follows.

$$\psi, \forall_{x_1, \dots, x_n} (\phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi) \in \mathcal{L}_{BB}$$

where $\forall_{x_1, \dots, x_n}(\varphi)$ denotes the universal closure of the formula φ for every variable x_1, \dots, x_n occurring in φ .

² Throughout this paper we will use such standard terminology. In case of doubt, we will use the same terminology as used in [1].

In order to reach its goals, a 3APL agent adopts plans. A plan is built from basic elements that can be composed by sequence operators, if-then-else constructs, and while-do loops. The basic elements can be basic actions, test actions, or abstract plans. A test action checks whether a certain formula is derivable from the belief base. An abstract plan is an abstract representation of a plan which can be instantiated with a plan during execution. Thus, an abstract plan cannot be executed directly and should be rewritten into another plan, possibly (and even probably) containing executable basic actions, through application of reasoning rules (see below).

There are three types of basic actions. The first type of basic action is the mental action. This action modifies the beliefs of the agents and is specified in terms of pre- and post-conditions in the form of belief formulae. A mental action can be performed if the pre-condition is derivable from the agent beliefs after which the post-condition must be derivable. The external actions can be performed in the environment of the agents. The effect of these actions is determined by the environment and can be perceived by the agent through sensing. The communication actions pass messages to another agent. A message contains the name of the receiver of the message, the speech act or performative (e.g. inform, request, etc.) of the message, and the content. The content of the message is a belief formula.

In order to reason with goals and plans, 3APL has two types of reasoning rules: goal planning rules and plan revision rules. A goal planning rule, which is a tuple consisting of a goal formula, a belief formula and a plan, indicates that the state represented by the goal formula can be reached by the plan if the belief formula holds. Such a rule is applicable when the agent has a goal unifiable with the goal formula of the rule and when the belief formula of the rule is derivable from the belief base. Application of such a rule will add the plan to the set of plans of the agent.

A plan revision rule, which is a tuple consisting of a belief formula and two plans, indicates that the first plan can be replaced by the second plan if the belief formula holds. Such a rule is applicable when the agent has a plan unifiable with the first plan of the rule and when the belief formula of the rule is derivable from the belief base. The application of the rule will replace the unifiable plan of the agent with the second plan. The plan revision rules are powerful way to handle failed or blocked plans as well as adapting and removing plans of the agent.

A 3APL agent starts its deliberation with a number of goals to achieve. Planning rules are then applied to generate plans for the goals after which the plans are executed to achieve the goals. Plans may start with mental actions for which the pre-conditions are not true. In this case, the plan revision rules are applied to generate alternative plans. During the execution of 3APL agents, there are four cases where it is checked if a certain formula is derivable from the beliefs of the agent. These cases are related to the execution of *test actions* and *mental actions*, and to the application of the *goal planning rules* and *plan revision rules*. In fact, the content of test actions, the pre-condition of mental actions, and the guard of the rules are logical formulae which should be derivable

before these actions and rules can be executed or applied. We define the language of these formulae, the *belief query language* (\mathcal{L}_B).

Definition 2. (*belief query language*) Let \mathcal{L} be the base language. Then, the belief query language \mathcal{L}_B with typical formula β is defined as follows:

- if $\phi \in \mathcal{L}$, then $\mathbf{B}(\phi), \neg\mathbf{B}(\phi) \in \text{Disjunction}$,
- $\top \in \text{Disjunction}$,
- if $\delta, \delta' \in \text{Disjunction}$, then $\delta \vee \delta' \in \text{Disjunction}$,
- if $\delta \in \text{Disjunction}$, then $\delta \in \mathcal{L}_B$,
- if $\beta, \beta' \in \mathcal{L}_B$, then $\beta \wedge \beta' \in \mathcal{L}_B$.

The bold ‘ \mathbf{B} ’ is not a modal operator, but is used to represent a query expression. For example $\mathbf{B}\varphi$ represents the query whether φ is derivable from the belief base. We use $\neg\mathbf{B}\varphi$ to represent the query whether φ is *not* derivable from the belief base. This interpretation of \neg corresponds with the interpretation of negation as failure in logic programming. The \mathbf{B} operator prevents the distribution of this negation over the belief query.

The 3APL semantics is based on an operational semantics which is defined in terms of a transition system. A transition system is a set of derivation rules for deriving transitions. A transition is a transformation of one configuration (or state) into another and it corresponds to a single computation step. In the case of the operational semantics for 3APL agents, the configurations are the mental states of the 3APL agents defined in terms of the belief base, goal base, plan base, and a substitution which assigns terms to variables.

Definition 3. (*configuration*) Let \mathcal{L}_{GB} be the goal language and \mathcal{L}_P be the plan language³. A configuration of an individual 3APL agent is a tuple $\langle \iota, \sigma, \gamma, \Pi, \theta \rangle$, where ι is an agent identifier, $\sigma \subseteq \mathcal{L}_{BB}$ is the belief base of the agent, $\gamma \subseteq \mathcal{L}_{GB}$ is the goal base of the agent, $\Pi \subseteq \mathcal{L}_P$ is the plan base of the agent and θ is a ground substitution that binds domain variables to domain terms.

In order to check whether an agent in a certain state has a certain belief or not, one must check if the corresponding belief query formula is derivable from the agent configuration that represents the state of the agent. In the 3APL transition semantics, this is formally expressed through an entailment relation \models_τ ⁴. In particular, to check if agent ι believes $\beta \in \mathcal{L}_B$ (a belief query formula) in state $\langle \iota, \sigma, \gamma, \Pi, \theta \rangle$ is specified as $\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta$.

The entailment relation \models_τ is defined recursively for the compound formulae. At the level of atomic formulae the satisfaction relation is defined in terms of propositional satisfaction relation. In particular, if β is of the form $\mathbf{B}\phi$, the satisfaction relation is defined as follows:

$$\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta \Leftrightarrow \sigma \models \phi\tau$$

³ Since the focus of this paper is the beliefs of the 3APL agents, the goal and plan language are not presented. For a detailed specification of these languages, see [1]

⁴ The subscript τ is a substitution under which a formula is derivable from the state.

The details of this semantics are explained in [1].

In the current implementation of 3APL, we use the Prolog engine for the propositional satisfaction relation \models in order to check the derivability of a belief query formula from the agent belief. In this paper, we will extend the beliefs of agents with modal (belief) formula by adding the modal belief operator to the belief language and belief query language, and use a reasoning engine to check the derivability of belief query formulae from the agent beliefs.

3 PROLOG-based Approaches to Modal Reasoning

If one wants to use modal logic in logic programming there are generally two approaches. One can translate the modal operators to a non-modal logic and apply the theory of classical logic programming (the translational approach, exemplified by Nonnengart [5]). Alternatively one can maintain the modal syntax and adapt the logic programming theory to the modal extension (the direct approach [4]). It makes sense to take a closer look at the latter approach, as it allows for an explicit introduction of modal operators in 3APL.

In this section, we will take a closer look at MProlog (by Linh Anh Nguyen [6, 4]) and NemoLOG (by Matteo Baldoni [3, 7, 8]), two existing systems for modal-logic programming that use the direct approach. These systems are generic in the sense that they can be used to model a variety of multi-modal logics. They allow for different modal operators as well as for different axiom systems. Both MProlog and NemoLOG ultimately operate in a similar manner, using goal-resolution, which is very much like Prolog and which can be illustrated by considering a simple example in *KD45* modal logic.

Example 1. A simple modal logic programming example. Let us examine how MProlog and NemoLOG solve a modal logic program consisting of the following goal, rule and fact.

-
1. (**goal**) $\Box\varphi$
 2. (**rule**) $\Box(\Box\psi \rightarrow \Box\varphi)$
 3. (**fact**) $\Box\psi$
-

Both systems will prove the goal (1) from the fact (3) using the rule (2), after they have determined that the modalities in the goal, the rule and the fact are in correspondence with each other. In the terminology of MProlog and NemoLOG, this correspondence is called ‘direct consequence’ and ‘derivation’ respectively. The methods used by MProlog and NemoLOG differ slightly in various ways. For example while MProlog allows both universal and existential modal operators, NemoLOG disallows existential modal operators [8]. They also differ in the way they define their semantics. In the case of MProlog, an SLD-resolution method is defined that allows the use of specific rules to modify the modalities in goal-clauses, in order to be able to apply a rule to the goal-clause. On the other hand, NemoLOG introduces a goal-directed proof procedure which utilizes an external derivation mechanism for modalities. We will discuss this mechanism shortly.

Finally, there is a difference in the implementation. MProlog is developed as an interpreter while, for reasons of efficiency, NemoLOG is specifically designed not to be an interpreter. As a result, programs in MProlog need to have every rule and fact designated as such beforehand, while programs in NemoLOG look just like normal Prolog programs, except for the fact that each rule makes a call to a derivation-rule to check if the modal operators allow execution of the rule.

NemoLOG's separation of the logic governing modalities and the logic concerning program-clauses seems to make modal logic programming more clear, more intuitive and more easily adaptable to our purposes. In this paper we aim to give 3APL the instruments for accommodating a logic of belief. We will do so by blending in some of the ideas of NemoLOG. Specifically, we look at the goal-directed proof procedure.

A key part of this proof procedure is the external derivation mechanism for modalities. Modalities are stripped from the goal-clause and remembered in the so-called *Modal Context*. A modal context is an environment in which goals can be proven. To prove a goal in a given modal context we may use formulae that are valid in specific other modal contexts. These modal contexts are determined by a so called 'matching relation'. In this manner, the matching relation defines which modal logical framework we use. Syntactically modal contexts form a data structure used by the recursive goal-resolution procedure to prove modal goals. Semantically, they resemble the possible worlds well-known from Kripke semantics and the matching relation defines the properties of the modal accessibility relation [9].

When applying a rule to a goal-clause, the goal-directed proof procedure checks whether the modalities in the rule can be derived from the modal context. Consecutively the rule is applied to the goal and the modalities in the modal context are adjusted to represent the new environment.

Our approach to modal logic programming, based NemoLOG, will be explained in detail in section 4.3.

4 Integrating Modal Belief Properties in 3APL

In the previous sections we have taken a look at 3APL and have investigated modal logic programming, now we are ready to integrate both. It should be clear that this integration focusses on the belief base and belief query language in 3APL as well as on the (Prolog) inference mechanism 3APL uses.

4.1 The addition of modal operators in 3APL

In this paper we are primarily interested in installing a notion of *belief* into 3APL agents, so therefore we will only add one type of modal operator to the base language: modal operators of belief. These modal operators follow the axioms of a widely accepted system for belief, namely *KD45*. Later, in the section 4.2, we give an interpretation for this belief operator.

Even as in the original 3APL, we refrain from introducing direct negation to the base language. The belief base of an agent in 3APL is a Prolog program, and therefore it is based on the closed world assumption. Negation in a belief base is modelled as negation-as-failure, meaning that formulae are negated by merely not being in the belief base.

The addition of a modal operator for belief does not change this very much, though it does present us with some new possibilities regarding negation. For one, there is an intuitive difference between not believing that something is the case and believing that something is not the case. If we want to be able to distinguish these, we should add the possibility of using classical negation within the scope of a belief modality to 3APL. We leave this possibility for further research.

Because 3APL is a programming language for implementing Multi Agent Systems, it makes sense to introduce multiple modal operators of belief. Therefore we will introduce for each agent $i \in \mathit{Agents}$ a modal operator B_i .

Definition 4. (*extended base language*) Let \mathcal{L} be the basic language as defined in 1. Let B_i be a modal operator for an agent $i \in \mathit{Agents}$. The base language \mathcal{L} is extended with the modal belief operator by adding the following clause: if $\phi \in \mathcal{L}$, then $B_i\phi \in \mathcal{L}$.

The revised belief base language is defined in terms of the expressions of the extended base language. The belief base of an agent contains only a finite number of expressions from the respective language, possibly none. Note that, in the base language, we can have $B_iB_jB_i\phi$ where $i, j \in \mathit{Agents}$.

Definition 5. (*extended belief base language*) Let \mathcal{L} be the extended base language. Let $\psi, \psi_1, \dots, \psi_m \in \mathcal{L}$ be ground (atomic) formulae, let $\phi, \phi_1, \dots, \phi_m \in \mathcal{L}$ and let $i \in \mathit{Agents}$. The belief base language, \mathcal{L}_{BB} is a set of formulae defined on the extended base language as follows:

- $B_i\psi \in \mathcal{L}_{BB}$
- $\forall_{x_1, \dots, x_n} B_i(\phi_1 \wedge \dots \wedge \phi_m \rightarrow \phi) \in \mathcal{L}_{BB}$

where $\forall_{x_1, \dots, x_n}(\varphi)$ denotes the universal closure of the formula φ for every variable x_1, \dots, x_n occurring in φ .

According to the above definition, formulae in the belief base language can have two forms: $B_i\psi$ and $\forall_{x_1, \dots, x_n} B_i(\phi_1 \wedge \dots \wedge \phi_m \rightarrow \phi)$. We will call the first form the fact-form and the second the rule-form. Note that every formula in the belief base language \mathcal{L}_{BB} is preceded by a modality B_i . i.e. every expression in the belief base language is explicitly believed.

As one can see, the above definition for \mathcal{L}_{BB} is very similar to the one used to define the corresponding 3APL language. However, there are some notable differences. We have imposed the restriction that everything in the belief base of an agent (i) should be preceded by a belief modality (B_i) to make explicit that the agent believes whatever is in its belief base⁵. This as opposed to 3APL

⁵ Note that the argument of B_i is a belief formula that can contain modal operators B_i or B_j for other agents j .

where no modal operator is allowed (i.e. in 3APL everything in the belief base is *implicitly* believed). The explicit instead of implicit use of the belief modality will turn out to be especially useful when we come to the semantics of the belief base language.

Now we will redefine the belief query language. In the following it is important to note the difference between \mathbf{B} and B_i . The latter is used as a modal operator to indicate a belief of an agent i and can be applied to modal belief formulae in which B_i can occur again. The former is used to indicate that a formula is a belief formula and cannot be applied to a formula that contains the \mathbf{B} operator. Note that another use of the bold ' \mathbf{B} ' is to prevent the distribution of negation over its argument, as explained in our section on 3APL.

Definition 6. (*extended belief query language*) Let \mathcal{L} be the extended base language and let ψ be of the form $B_i\varphi$ where $\varphi \in \mathcal{L}$ and $i \in \text{Agents}$. We shall call ψ a formula in the inner belief query language (\mathcal{L}_I). The extended belief query language, \mathcal{L}_B , with typical formula β is defined as follows:

- $\mathbf{B}(\psi)$, $\neg\mathbf{B}(\psi) \in \text{Disjunction}$,
- $\top \in \text{Disjunction}$,
- if $\delta, \delta' \in \text{Disjunction}$ then $\delta \vee \delta' \in \text{Disjunction}$,
- if $\delta \in \text{Disjunction}$, then $\delta \in \mathcal{L}_B$,
- if $\beta, \beta' \in \mathcal{L}_B$, then $\beta \wedge \beta' \in \mathcal{L}_B$.

Note this definition is almost exactly the same as definition 2. The only difference lies in the arguments of the basic queries, which are of the form $\mathbf{B}(\varphi)$.

4.2 The semantics of modal operators in 3APL

In section 2, we have explained that during the execution of 3APL agents, there are four cases where it should be tested if a belief query formula is derivable from the agent belief. These test occur at the execution of *test actions* and *mental actions* and at the application of *goal planning rules* and *plan revision rules*. In the original 3APL, the agent typically uses a belief query for this purpose, and this has not changed with the addition of modal operators and the extra restrictions on both the belief base and belief query language. One thing that has changed is that the reasoning mechanism must now take modal operators into account. This expands our belief base language (\mathcal{L}_{BB}) to a Horn clause fragment of first order modal logic and enriches our belief query language (\mathcal{L}_B) with modal operators.

In order to define the derivability of queries from a belief base we will introduce a *superset language* \mathcal{L}^s . Relying on the belief base language or the belief query language alone does not suffice, because the languages differ from each other. We embed both the belief base language and the inner belief query language (\mathcal{L}_I) as subsets of the superset language to study the derivation relation in a standard modal logic setting. This embedding is illustrated in figure 1. This situation allows us to extend both the belief base and belief query language to

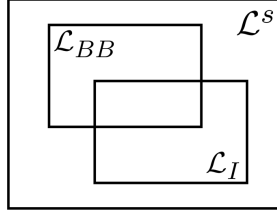


Fig. 1. *The relation between the languages.*

form a larger fragment of the superset language if we so desire. An example of such an extension could be the inclusion of negation in various forms.

The alphabet of the superset language consists of the same sets \mathcal{Vars} , \mathcal{Func} and \mathcal{Pred} as the base language (definition 1) combined with the classical logical connectives and quantifiers $\neg, \vee, \wedge, \rightarrow, \forall$ and \exists . The syntax of the superset language is defined as usual in first-order modal logic.

As usual in first-order modal logic, we will use Kripke interpretations to define the semantics of this superset language. These interpretations make use of possible worlds. Each possible world has a first-order domain of quantification. For the sake of simplicity we will assume constant domains (i.e. the domains do not vary between possible worlds) and rigid designators (i.e. constants refer to the same domain element, regardless of the possible world it is in). The semantics of the superset language are defined in terms of Kripke interpretations as follows:

Definition 7. (*Kripke interpretations*) Let M be a Kripke interpretation $M = \langle W, R, D, \pi \rangle$ where:

- W is a nonempty set of worlds
- $R = \{R_{a_1}, \dots, R_{a_n}\}$ for $a_1, \dots, a_n \in \text{Agents}$ where each $R_i \in R$ is a binary relation on W which is serial, transitive and Euclidean (the accessibility relation associated with B_i)
- D is a (nonempty) set of domain elements
- π is an interpretation of constant symbols function symbols and predicate symbols such that
 - for each n -ary function symbol f of \mathcal{L}^s (including constants of \mathcal{L}^s), $\pi(f)$ is a function from D^n to D
 - for each n -ary predicate symbol p and each world $w \in W$, $\pi(p, w)$ is an n -ary relation on D

A variable assignment V w.r.t. a Kripke interpretation M is a function that maps each variable to an element of the domain D of M . Interpretation for terms in the domain is defined as usual from the interpretation of constants and function symbols.

Definition 8. (*Kripke semantics*) Let \models be a relation between $w \in W$ and closed formulae of \mathcal{L}^s satisfying, for all $w \in W$, the following conditions:

- $M, V, w \models \top$
- $M, V, w \models p(t_1, \dots, t_n)$ iff $\langle (V(t_1), \dots, V(t_n)) \rangle \in \pi(p, w)$
- $M, V, w \models \neg\phi$ iff $M, V, w \not\models \phi$
- $M, V, w \models \phi \wedge \psi$ iff $M, V, w \models \phi$ and $M, V, w \models \psi$
- $M, V, w \models \phi \rightarrow \psi$ iff $M, V, w \not\models \phi$ or $M, V, w \models \psi$
- $M, V, w \models \forall x\phi$ iff for each $c \in D$, $M, V, w, \models \phi\{x/c\}$
- $M, V, w \models \exists x\phi$ iff for some $c \in D$, $M, V, w, \models \phi\{x/c\}$
- $M, V, w \models B_i\phi$ iff for all $w' \in W$ such that $(w, w') \in R_i$, $M, V, w' \models \phi$

A closed formula (i.e. a formula where all variables are bound by a quantifier) ϕ of the language \mathcal{L}^s is satisfiable if there is a Kripke model $M = \langle W, R, D, \pi \rangle$ and a world $w \in W$ for some variable assignment V , such that $M, V, w \models \phi$. ϕ is a valid formula ($\models \phi$) if $\neg\phi$ is not satisfiable.

We will lift the above definition for satisfiability to groups of formulae: $M, V, w \models \Phi$ iff $M, w \models \phi$ for all $\phi \in \Phi$. We will define modal logical entailment between groups of formulae $\Phi \models_{\mathcal{L}^s} \Psi$ as $M, V, w \models \Phi$ implies $M, V, w \models \Psi$.

We can use this definition of entailment to establish whether a set of formulae in \mathcal{L}^s follows from a second set of formulae in \mathcal{L}^s . This of course also holds for formulae in clausal fragments of \mathcal{L}^s such as the belief base language.

We can now give a definition of the semantics of belief queries based on [1]. We will do so by defining what it means for belief queries to be satisfied by an agent configuration. It is important to stress that this entailment (denoted by \models_τ) relation is an entirely different relation than the one defining modal logical entailment (denoted by \models) from definition 8. In particular, we will use the entailment relation defined for modal logic (\models) to define the entailment relation for belief queries (\models_τ).

Definition 9. (semantics of belief queries) Let $\langle \iota, \sigma, \gamma, \Pi, \theta \rangle$ be the agent configuration of agent ι , $\delta, \delta' \in \text{Disjunction}$ and $\mathbf{B}\phi, \beta, \beta' \in \mathcal{L}_B$. Let τ, τ_1, τ_2 be ground substitutions. The semantics of belief queries is defined as follows:

$$\begin{aligned}
\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\emptyset \top \\
\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \mathbf{B}\phi &\Leftrightarrow \sigma \models \phi\tau \text{ where } \text{Var}_f(\phi) \subseteq \text{dom}(\tau) \\
\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\emptyset \neg\mathbf{B}\phi &\Leftrightarrow \neg\exists\tau : \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \mathbf{B}\phi \\
\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta \vee \delta' &\Leftrightarrow \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta \text{ or} \\
&\quad (\forall\tau' : \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \not\models_{\tau'} \delta \text{ and } \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta') \\
\langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta \wedge \beta' &\Leftrightarrow \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta \text{ and } \langle \iota, \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta'\tau
\end{aligned}$$

Here the entailment relation \models on the righthand side of the second clause is defined in definition 8.

Again, except for the use of an entailment function for first order modal logic, this is quite like the semantics specified in the original 3APL [1].

4.3 A goal-directed proof procedure

In this section we will give a goal-directed proof procedure. This procedure will allow us to compute whether a belief query is entailed by a belief base. The

procedure is based on NemoLOG [3] proposed by Baldoni, which is adapted to the needs and specifications of 3APL. We will first introduce the concept of a modal context, which will form the basis of our matching⁶ relation ($\overset{*}{\Rightarrow}$) and thus for the whole of the proof procedure.

Definition 10. (*Modal Context*) Let $M_1, \dots, M_n \in \{B_1, \dots, B_m\}$ where $1, \dots, m \in \text{Agents}$. We will then define a modal context \mathcal{MC} as a finite sequence of modal operators denoted by $M_1 \dots M_n$. We will denote an empty modal context with ε . A modal context records the modalities in front of goals during the execution of our goal-directed proof procedure. We will denote the set of all possible modal contexts by \mathcal{MC}^* .

Intuitively, a modal context can be seen as a name for a set of possible worlds. We say that we evaluate a formula in a certain modal context \mathcal{MC} when we evaluate it in each possible world named by that modal context instead of in the actual world. In our goal-directed proof procedure, we will use the modal context to be able to recognize syntactically if a Program Clause is applicable to the Goal or not. From this viewpoint, the modal context denotes in which worlds we are evaluating the Goal. The proof procedure itself will, when appropriate, update the modal context to reflect the correct worlds.

To avoid problems with variable renaming and substitutions we will denote by $[P]$ the set containing the set of all ground instances of clauses from a belief base P . Also, each formula $F \in \mathcal{L}$ which is in $[P]$ will be transformed into a formula of the form $\top \rightarrow F$. We do this to make all semantical information in $[P]$ available in a uniform way, i.e. in the form of implications, the rule-form. This will be useful when we get to the inductive definition of the goal-directed proof procedure described in definition 13.

Definition 11. (*Program Clauses*) Let P be a set of clauses $\in \mathcal{L}_{BB}$ and $\Gamma \in \mathcal{MC}^*$. Define $[P]$ (the Program Clauses) to be the set of formulae obtained by transforming all clauses in P using the following rules:

- a) if $C' \in P$ and $C' \in \mathcal{L}$ then $\top \rightarrow C' \in [P]$.
- b) if $\forall x_1, \dots, x_n \Gamma C' \in P$ and $\forall x_1, \dots, x_n \Gamma C' \notin \mathcal{L}$ then $\Gamma C' \{t/x\} \in [P]$
for each $x \in x_1, \dots, x_n$ for all ground terms t that can occur in P .

Let G , which we shall call a Goal, be either \top or of the form $\phi_1 \wedge \dots \wedge \phi_n$, where $\phi_1, \dots, \phi_n \in \mathcal{L}$. We will call $[P]$, where P is the contents of the belief base, the Program. By definition 8 it can easily be seen that in $[P]$ every bit of semantical information in the belief base P is present in clauses of the general form $\Gamma_b(G \rightarrow \Gamma_h A)$, where Γ_b is ε or a single belief modality, Γ_h is an arbitrary, possibly empty, sequence of belief modalities and $A \in \mathcal{L}$. This is important because the inference rules of our goal-directed proof procedure can only make use of Program Clauses written in this general form.

In the following, we will give a proof procedure that applies formulae in the program $[P]$ to prove a goal G in a modal context \mathcal{MC} . The applicability of a

⁶ As mentioned in the previous chapter, Baldoni often uses the word derivation.

formula $F \in [P]$ depends on the modalities in F and on the modal context. The sequences of modalities in F must *match* the sequence of modal operators that is \mathcal{MC} . We will now give the definition of this matching relation.

Definition 12. (*matching relation*) For $1, \dots, n \in \text{Agents}$, let \mathcal{M} be the set of modal operators $\{B_1, \dots, B_n\}$. We define two auxiliary relations. Let the relation \Rightarrow be defined as $\Rightarrow = \{(M_i, M_i M_i) \mid M_i \in \mathcal{M}\}$. Let the relation \Rightarrow' be defined as $\Rightarrow' = \{(\Gamma_1 \Gamma_2, \Gamma_1 \Gamma' \Gamma_2) \mid \Gamma, \Gamma', \Gamma_1, \Gamma_2 \in \mathcal{MC}^* \ \& \ \Gamma \Rightarrow \Gamma'\}$ ⁷. Now we can define our matching relation (\Rightarrow^*) as the equivalency closure of \Rightarrow' over the space \mathcal{MC}^* .

Now we will give a set of inference rules. These rules constitute our goal-directed proof procedure.

Definition 13. (*goal-directed proof procedure*) Let M be a single modal operator. Let $A \in \mathcal{L}$ be without modal operators. We define the procedure to find a proof for a closed Goal G from a modal context $\mathcal{MC} = \Gamma$ and a program $[P]$ by induction on the structure of G as follows:

1. $[P], \Gamma \vdash \top$;
2. $[P], \Gamma \vdash A$ if there is a clause $\Gamma_b(G \rightarrow \Gamma_h A) \in P$ and a modal context Γ'_b such that
 (1) $[P], \Gamma'_b \vdash G$, (2) $\Gamma_b \xrightarrow{*} \Gamma'_b$, and (3) $\Gamma'_b \Gamma_h \xrightarrow{*} \Gamma$.
3. $[P], \Gamma \vdash G_1 \wedge G_2$ if
 $[P], \Gamma \vdash G_1$ and
 $[P], \Gamma \vdash G_2$;
4. $[P], \Gamma \vdash \exists x G$ if for some ground term t
 $[P], \Gamma \vdash \{x/t\}G$;
5. $[P], \Gamma \vdash MG$ if
 $[P], \Gamma M \vdash G$.

Given a program $[P]$ and a closed goal G , we say that G is provable from $[P]$ if $[P], \varepsilon \vdash G$ can be derived by applying the above rules 1. - 5.

Using Goal-Resolution and Backwards chaining, this proof procedure is able to prove modal goals in the same way Prolog is able to prove non-modal goals. Implementation of the proof procedure builds on the current 3APL implementation and is a matter of writing a simple Prolog module to handle the modalities.

The above definitions (11-13) can together be seen as a proof method for the $KD45_n$ quantified modal logic that we want our agents to use. The working of the proof procedure is very simple: The rules 1, 3, and 4 deal with query formulae that are always true, conjunctions or existentially qualified, respectively. If a goal is prefixed by a modality, then rule 5 takes the modality away from the goal while updating the modal context. Using definition 11 we ensure that goals cannot be of a form other than the ones dealt with above, so this suffices. Finally, rule 2 allows for the application of a Program Clause (rule) and specifies the resulting Goal and modal context. Rule 2 makes use of the special matching relation $\xrightarrow{*}$ as defined in definition 12. The matching relation is intended as a means to

⁷ We denote by $\Gamma_1 \Gamma_2$ the concatenation of the modal contexts Γ_1 and Γ_2 .

establish if a certain sequence of modal operators can be used to access a formula involving (or if you will; guarded by) another sequence of modal operators.

Our matching relation is a specialized version of the derivation relation defined by Baldoni [3], which is based on *inclusion axioms*⁸. That derivation relation $\overset{*}{\Rightarrow}$ is defined as the transitive and reflexive closure of $\Gamma\Gamma_1\Gamma' \Rightarrow \Gamma\Gamma_2\Gamma'$ for each *inclusion axiom* $\Gamma_1 \rightarrow \Gamma_2$. Here $\Gamma, \Gamma_1, \Gamma_2$ and Γ' are all sequences of modalities.

An example of an inclusion axiom is $\Box_i \rightarrow \Box_i\Box_i$, describing transitivity. A derivation relation based on this axiom makes $\Box_2\Box_1\Box_2 \overset{*}{\Rightarrow} \Box_2\Box_1\Box_1\Box_2$ valid. Other inclusion axiom will introduce other properties. Our own derivation relation (definition 12) gives our programs transitivity, seriality and euclidity. However this can easily be adapted to other logic systems following [3].

4.4 Soundness of the proof-procedure

The proof-procedure given in the previous section allows us to derive a belief query from an agent configuration. We first transform the belief base of the agent configuration into Program Clauses and the belief query into a query usable by the goal-directed proof procedure defined in definition 13. Given this goal-directed proof procedure and given the Kripke semantics for the superset language \mathcal{L}^s (given in definition 8) we show that the proof procedure is sound.

In order to prove soundness, we must prove that every belief query proven by our procedure (definition 13) is valid using the semantics the superset language (\mathcal{L}^s). Formally: $[P] \vdash \beta \Rightarrow P \models_{\mathcal{L}^s} \beta$.

With our soundness proof we deviate from the approach of Baldoni, who gives a correctness proof based on fixpoint semantics relating to both the operational semantics and the Kripke semantics [3]. We instead attempt to give a soundness proof by directly relating the goal directed proof procedure to the Kripke semantics of our superset language.

The matching relation $\overset{*}{\Rightarrow}$ plays an important role in the proof procedure, therefore it makes sense to establish some important properties for this relation. First, because the matching relation determines the properties of the modal logic of the system, we will prove that these properties are the desired ones. As a consequence we prove that the validity of modal logical formulae is preserved within matching modal contexts. This allows us to prove the soundness of our proof procedure.

The matching relation imposes certain restrictions on the relations between modal contexts, and thus on B_i . We will prove that these restrictions correspond to those described in definition 8 (seriality, transitivity and euclidity) by constructing a Kripke frame over \mathcal{MC}^* (the set of all modal contexts) and the relation R_i . We will then apply the restrictions imposed by the matching relation and prove that the frame is a $KD45_n$ -frame, which means that seriality, transitivity and euclidity apply.

⁸ inclusion axioms can be used to describe modal logic properties. For example $\Box_i \rightarrow \Box_i\Box_i$ describes transitivity.

A $KD45_n$ -frame is a non-empty set of worlds with accessibility relations between those worlds that are serial, transitive and euclidean, with respect to multiple agents [10]. Our Kripke interpretation of definition 7 is based on a $KD45_n$ -Kripke frame (the worlds W and relations R of the interpretation). What we prove is that, if one imposes restrictions on the relations in accordance with the *matching* relation, then the desired transitivity, seriality and euclidity properties of the relations $R_i \in R$ hold.

Lemma 1. (*Semantics of the matching relation*) *The matching relation $\overset{*}{\Rightarrow}$ over \mathcal{MC}^* defines a $KD45_n$ -frame.*

Proof. Let \mathcal{F}' be a frame $\langle W, R \rangle$ where W is an infinite set of worlds such that $W = \{w_\Gamma | \Gamma \in \mathcal{MC}^*\}$ and R is a set of relations $\{R_1, \dots, R_n\}$ for $1, \dots, n \in \mathcal{Agents}$ where the relation R_i is defined as $(w_\Gamma, w_{\Gamma'})$ where $\Gamma' = \Gamma M_i$. Here ΓM_i is the modal context obtained by the concatenation of Γ and the single modality M_i . Let \mathcal{F} be \mathcal{F}' modulo $\overset{*}{\Rightarrow}$, i.e. if $\Gamma \overset{*}{\Rightarrow} \Gamma'$ then $w_\Gamma \in W$ and $w_{\Gamma'} \in W$ are equivalent. We will prove that the accessibility relations $R_i \in R$ are all serial, transitive and euclidean and that therefore \mathcal{F} is a $KD45_n$ -frame.

seriality Every $\Gamma \in \mathcal{MC}^*$ corresponds to a world $w_\Gamma \in W$. Furthermore if $\Gamma \in \mathcal{MC}^*$ then, by the infinite nature of \mathcal{MC}^* , $\Gamma M_i \in \mathcal{MC}^*$. Therefore there always is a world $w_{\Gamma'} \in W$ that corresponds with ΓM_i . Since R_i is defined as $(w_\Gamma, w_{\Gamma'})$ we have that for every $w_\Gamma \in W$ $(w_\Gamma, w_{\Gamma'}) \in R_i$.

transitivity Let $w_\Gamma, w_{\Gamma'}$ and $w_{\Gamma''}$ correspond to the modal contexts $\Gamma, \Gamma M_i$ and $\Gamma M_i M_i$ respectively. Since $\Gamma M_i \overset{*}{\Rightarrow} \Gamma M_i M_i$ and because \mathcal{F} is \mathcal{F}' modulo $\overset{*}{\Rightarrow}$, we have $w_{\Gamma'} = w_{\Gamma''}$. Moreover, since $(w_\Gamma, w_{\Gamma'}) \in R_i$ we also have $(w_\Gamma, w_{\Gamma''}) \in R_i$.

euclidity If $(w_\Gamma, w_{\Gamma'}) \in R_i$ and $(w_\Gamma, w_{\Gamma''}) \in R_i$ then if w_Γ corresponds with the modal context Γ then $w_{\Gamma'}$ and $w_{\Gamma''}$ corresponds with the modal context ΓM_i . Because the relation R_i is also serial there must be a relation $(w_{\Gamma'}, w_{\Gamma'''}) \in R_i$ such that $w_{\Gamma'''}$ corresponds with the modal context $\Gamma M_i M_i$. Because $\Gamma M_i M_i \overset{*}{\Rightarrow} \Gamma M_i$, we have $w_{\Gamma'''} = w_{\Gamma''}$ and thus $(w_{\Gamma'}, w_{\Gamma''}) \in R_i$. \dashv

Lemma 2. *If $\Gamma\varphi$ is valid (in the logic of the superset language (definition 8)), and $\Gamma \overset{*}{\Rightarrow} \Gamma'$, then $\Gamma'\varphi$ is also valid.*

Proof. This is a direct consequence of the previous lemma (1) saying that the matching relation $\overset{*}{\Rightarrow}$ defines a $KD45_n$ -frame over modal contexts. \dashv

Theorem 1. (*soundness: $[P] \vdash \beta \Rightarrow P \vDash_{\mathcal{L}^s} \beta$*) *Every belief query made true by the goal-directed proof procedure (\vdash) is also valid in the Kripke semantics (\vDash).*

Proof. We have to prove that $[P] \vdash \beta \Rightarrow P \vDash_{\mathcal{L}^s} \beta$. If the goal directed proof procedure stops, the result is a proof tree, where any node is of the form $[P], \Gamma \vdash \beta$, the root is of the form $[P], \epsilon \vdash G$, and all leaves have the form $[P], \Gamma \vdash \top$. With every node $[P], \Gamma \vdash \beta$ of the tree, we define an associated \mathcal{L}^s formula of the form $P \rightarrow \Gamma\beta$. Then we prove that validity of the associated formulas $P \rightarrow \Gamma\beta$ is preserved when going up (or 'down', depending on whether

you really want to picture the tree as a tree) from the leaves of the tree to the root. The associated formulae for the leaves are all of the form $P \rightarrow \Gamma \top$, which are trivially valid in the superset logic (Definition 8). Then by induction over the proof tree structure, we get that $P \rightarrow \beta$ is valid (ε is the empty modal context), which (for all standard notions of entailment) results in $P \vDash_{\mathcal{L}^s} \beta$.

To complete the proof, we now prove the preservation property for each individual rule in the goal-directed proof procedure.

Rule 1: Trivial.

Rule 2: We have to prove that under the conditions (1) $\Gamma_b(G \rightarrow \Gamma_h A) \in P$, (2) $\Gamma'_b \Gamma_h \xrightarrow{*} \Gamma$, and (3) $\Gamma_b \xrightarrow{*} \Gamma'_b$, validity of $P \rightarrow \Gamma'_b G$ implies validity of $P \rightarrow \Gamma A$. If $\Gamma_b(G \rightarrow \Gamma_h A)$ is not valid, the implication holds trivially, so we only have to consider the case where $\Gamma_b(G \rightarrow \Gamma_h A)$ is valid. Thus, we have to prove that under conditions 2 and 3, the validity of $P \rightarrow \Gamma'_b G$ and $\Gamma_b(G \rightarrow \Gamma_h A)$ implies the validity of $P \rightarrow \Gamma A$. For this we make use of the previous lemma (2). With this lemma together with condition 2, we conclude that from the validity of $\Gamma_b(G \rightarrow \Gamma_h A)$ we may conclude the validity of $\Gamma'_b(G \rightarrow \Gamma_h A)$. From the fact that we deal with *normal* modal reasoning, we may conclude to the validity of $\Gamma'_b G \rightarrow \Gamma'_b \Gamma_h A$ (applying the K-property of normal modal logic). From this, together with the validity of $P \rightarrow \Gamma'_b G$, we may conclude to the validity of $P \rightarrow \Gamma'_b \Gamma_h A$. Applying the previous lemma (2) one more time gives us the desired $P \rightarrow \Gamma A$.

Rule 3: We have to prove that if $P \rightarrow \Gamma G_1$ and $P \rightarrow \Gamma G_2$ are valid, also $P \rightarrow \Gamma(G_1 \wedge G_2)$ is valid. This follows directly from the fact that we deal with *normal* modal logic (i.e. Kripke structures), for which $\Box\varphi \wedge \Box\psi \rightarrow \Box(\varphi \wedge \psi)$.

Rule 4: This rule replaces a ground term by an existentially quantified variable. From the semantics of existential quantification, it immediately follows that this preserves validity.

Rule 5: Trivial. ⊣

5 Conclusion and Further Research

We have extended the agent programming language 3APL, giving agents a means to explicitly reason about the beliefs of themselves and the beliefs of other agents. We have done so by adding modal operators of belief to the syntax of the belief base and belief query languages of 3APL. The corresponding semantics have been adjusted to correspond with a $KD45_n$ -type modal logic, often used to represent beliefs, governing the behavior of these modal operators. In the final section we have given a method for checking the derivability of a belief query from a belief base, providing the functionality we sought. This method is proven to be sound.

The next step will be to implement a working version of this method, and to test it with communication. This implementation can be build upon the existing 3APL implementation. This would only require programming a Prolog interpreter that can work with the modalities involved.

Another interesting possibility is the addition of negation to the belief base and belief query languages. This may dramatically increase the expressional

power of 3APL. With the right restrictions, we believe that negation can be introduced problem-free, however this is left for further research. Finally, we plan to show that our proof method is complete with respect to possible belief queries and belief base content. This can be done by induction on the form of the belief queries.

References

1. M. Dastani, B. van Riemsdijk, F. Dignum, J.J. Meyer: A programming language for cognitive agents: Goal directed 3apl. In Mehdi Dastani, Juergen Dix, A.E.F.S., ed.: *Programming Multi-Agent Systems: First International Workshop, PROMAS 2003*, Melbourne, Australia, July 15, 2003, Selected Revised and Invited papers. Volume 3067 of *Lecture Notes in Computer Science.*, Springer (2004) 111–130
2. The Foundation for Intelligent Physical Agents: (Fipa communicative act library specification)
3. Baldoni, M.: *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, University of Turin, Italy (1998)
4. Nguyen, L.A.: The modal logic programming system MProlog. In Alferes, J.J., Leite, J.A., eds.: *Proceedings of JELIA 2004*. Volume 3229 of *Lecture Notes in Computer Science.*, Springer (2004) 266–278
5. Nonnengart, A.: How to use modalities and sorts in prolog. In MacNish, C., Pearce, D., Pereira, L.M., eds.: *Logics in Artificial Intelligence*. Springer (1994) 365–378
6. Nguyen, L.A.: *Multimodal logic programming and its applications to modal deductive databases*. Technical report (2003)
7. M. Baldoni, L. Giordano, A. Martelli: A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning. In: *Journal of Logic and Computation*. Volume 8. Oxford University Press (1998) 597–635
8. M. Baldoni, L. Giordano, A. Martelli: A framework for modal logic programming. In Maher, M., ed.: *Proc. of the Joint International Conference and Symposium on Logic Programming, JICSLP'96*. MIT Press (1996) 52–66
9. M. Baldoni, L. Giordano, A. Martelli: Translating a modal language with embedded implication into horn clause logic. In: *Extensions of Logic Programming*. (1996) 19–33
10. Blackburn, P., de Rijke, M., Venema, Y.: *Modal Logic*. Volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press (2001)