A Grounded Specification Language for Agent Programs

Mehdi Dastani
Utrecht University
The Netherlands
mehdi@cs.uu.nl

M. Birna van Riemsdijk
LMU, Munich
Germany
riemsdijk@pst.ifi.lmu.de

John-Jules Ch. Meyer
Utrecht University
The Netherlands
jj@cs.uu.nl

ABSTRACT

This paper studies the relation between agent specification and agent programming languages. In particular, it shows that an agent programming language obeys some desirable properties expressed in an agent specification language, i.e., that any agent implemented by the programming language satisfies the desirable property expressed in the specification language. We study this relation by defining and aligning the semantics of an agent specification language and implementation language, and prove that certain properties expressed in the specification language are satisfied by the implementation language.

Categories and Subject Descriptors

F.3.1 [**Logics and Meaning of Programs**]: Specifying and Verifying and Reasoning about Programs—*Logics of programs, Specification techniques*

Keywords

Agent Programming Language, Agent Specification language

1. INTRODUCTION

In the area of agent theory and agent-oriented software systems, various logics have been proposed to characterize the behavior of rational agents. The most cited logics to specify agents' behavior are the BDI logics [3, 10, 11, 8]. These logics are multi-modal logics consisting of temporal and epistemic modal operators. In the BDI logics, the behaviour of an agent is specified in terms of the temporal evolution of its mental attitudes (i.e., beliefs, desires, and intentions) and their interactions. These logics are characterized by means of axioms and inference rules to capture the desired static and dynamic properties of agents' behaviour. In particular, the axioms establish the desired properties of the epistemic and temporal operators as well as the rational balance between them. For example, the axioms to capture the desired static properties of beliefs are $KD45$ (the standard weak $S5$ system), for desires and intentions are the K and D axioms, and for the rational balance between beliefs and desires the various versions of the *realism* axioms. Moreover, some desired dynamic properties of agents' behaviour are captured through axioms that implement various versions of the *commitment strategies*. These axioms are defined using temporal operators expressing when and under which conditions the goals and intentions of agents can be dropped. For example, an agent can be specified either to hold its goals until it has achieved it (blindly-committed agent type), or to drop the goal if it believes that it can *never* achieve it (single-minded agent type) [3, 10].

Motivated by these logics, various agent-oriented programming languages have been proposed [1, 4] to implement rational agents. These programming languages provide specific constructs to implement the concepts that are used in the BDI logics (e.g., beliefs, goals, and intentions). Some of the agent programming languages are designed and developed from scratch and come with an explicit formal semantics, while others are extensions of existing programming languages such as Java, an do not have an explicit formal semantics. A main concern in designing and developing agent-oriented programming languages is to provide programming constructs in such a way that their executions generate the agent behaviours having the same desirable properties as in their specifications. This implies that the semantics of the programming languages (or the extensions) should be defined in such a way to satisfy (some of) the desirable properties captured by means of the axioms in the BDI logics. It should be noted that arbitrary programming languages can be used to implement agents. However, it will become difficult to verify whether they satisfy their specifications as it is hard to identify what for instance the beliefs, desires and intentions of the agents are. This is referred to by Wooldridge as the problem of *ungrounded semantics* for agent specification languages [14].

The main issue addressed in this paper is how agent specification logics, which are used to specify the agents' behaviour, are related to agent-oriented programming languages, which are used to implement the agents. In this paper, we show that an agent programming language obeys some desirable properties expressed in an agent specification language, i.e., that any individual agent implemented by the programming language satisfies the desirable property expressed in the specification language. We study this relation by defining and aligning the semantics of an agent specification language and implementation language, and prove that certain properties expressed in the specification language are satisfied by the implementation language. It is important to realize that this problem is different from the model checking problem. In model checking, the problem is to verify if a certain property, expressed in a specification language such as the language of a BDI logic, holds for one specific agent implementation (and not for any agent that is implemented by the same programming language). Of course, the properties that we focus on in this paper are general properties of individual agents such as commitment strategies. Thus, in contrast to model checking, where one verifies that a particular agent program satisfies a particular property expressed in a specification language, we are here interested only in certain general properties of individual agents such as *an agent will not drop its goals until it believes it has achieved them.*

In this paper, we show a possible connection between an agent specification language and an agent-oriented programming language. In particular, we first present a simple but extendable agent-oriented programming language that provides mental concepts such as beliefs, goals, and plans. The syntax and semantics of this programming language will be presented in section 2. We then present in section 3 an agent specification language which is closely related to BDI specification languages. This language consists of (modal) operators for beliefs, goals, and time such that properties such as commitment strategies can be expressed. In section 4 we show that all agents that are implemented in the proposed agent programming language satisfy the desirable properties that are specified in the specification language. Finally, we conclude the paper and discuss future research directions.

2. AN AGENT-ORIENTED PROGRAMMING LANGUAGE *APL*

In this section, we propose the syntax and semantics of a simple logic-based agent-oriented programming language that provides programming constructs to implement cognitive agents. In order to focus on the relation between programming and specification languages and without loss of generality, we ignore some aspects that may be relevant or even necessary for a practical logic-based agent-oriented programming language. This language is based on the existing agent-oriented programming languages (e.g., 3APL and Jason [5, 1]) such that our approach can be used to relate them to a specification language as well.

2.1 Syntax of *APL*

The presented agent-oriented programming language will have beliefs, goals, plans, and planning rules. We use a propositional language to represent an agent's beliefs and goals while plans are assumed as operations that update the agent's beliefs base. It is important to note that these simplifications do not limit the applicability of the proposed model. In order to study various desirable properties of this programming language, we will consider two types of goals: achievement and maintenance goals. Elsewhere [5] we have discussed these goal types and elaborated on their detailed semantics.

The idea of an achievement goal is to reach the state denoted by it. An agent is then expected to generate and execute plans to achieve its achievement goal. The emphasis here is that the goal will not be dropped until the state denoted by it is achieved. An example of an achievement goal is to have fuel in car (`fuel`) for which the agent can generate either a plan to fuel at gas station 1 (`gs1`) or a plan to fuel at gas station 2 (`gs2`). The achievement goal will be dropped as soon as the agent believes that it has fuel in its car, i.e., as soon as it believes `fuel`. As suggested in [12] and implemented in Jadex, we assign a failure condition to each achievement goal to indicate when the agent should stop trying to achieve the goal and thus drop the goal. For example, the failure condition of the achievement goal `fuel` can be set as `fuelNoWhere` indicating that there is neither fuel at gas station 1 nor at gas station 2.

The idea of a maintenance goal is to ensure that a state holds and continues to hold. An agent is expected to generate and execute plans only if the state denoted by the maintenance goal is *threatened* not to hold (rather than waiting and taking action once the state does not hold). The condition under which the maintenance goal is threatened not to hold will be called the *maintain condition*. The agent starts to generate and execute plans when the maintain condition becomes true. For example, consider now an agent with `fuel` as a maintain goal. This means that the agent wants to maintain having a fueled car. In such a case, the maintain condition is the illuminated lamp warning of a shortage of fuel. The agent should in this case generate and execute a refuel plan until the maintain condition holds again.

DEFINITION 1. *(belief and goal languages) Let L_p be a propositional language (used also in the rest of the paper). The belief language L_σ, the achievement goal language L_{γ_a}, and the maintenance goal language L_{γ_m} are defined as follows:*
- $L_\sigma = L_p$
- $L_{\gamma_a} = \{(\phi, \psi) \mid \phi, \psi \in L_p\}$
- $L_{\gamma_m} = \{(\phi, \psi) \mid \phi, \psi \in L_p\}$

An achievement goal is thus represented as a tuple (ϕ, ψ) of propositions, where ϕ denotes the desirable state to achieve and ψ is its corresponding failure condition. In our running example, a car driving agent can believe that he is in position 1 (`pos1`) and have the achievement goal `(fuel, fuelNoWhere)`. A maintenance goal is also represented as a tuple (ϕ, ψ) of propositions, where ϕ denote the state to maintain and ψ is its corresponding maintain condition.

For the purpose of this paper, we assume a set of plans Plan that can be executed by the agents atomically, i.e., a plan can be executed in one computation step. As plans are executed atomically, their internal structure can be ignored. The empty plan will be denoted by ϵ. Moreover, agents are assumed to generate their plans based on their beliefs and goals. The planning rules indicate which plans are appropriate to decide when the agent has a certain goal and beliefs. A planning rule is of the form $\beta, \kappa \Rightarrow \pi$ and indicates to select plan π for the goal κ, if the agent believes β. In order to be able to check whether an agent has a certain belief or goal, we use propositional formulas from L_p to represent belief and goal query expressions. Note that the planning rules are generic and can be used to generate plans for both achievement and maintenance goals.

DEFINITION 2. *(planning rule) Let* Plan *be the set of plans that an agent can use. The set of planning rules $\mathcal{R}_{\mathsf{PL}}$ is defined as:*

$$\mathcal{R}_{\mathsf{PL}} = \{\beta, \kappa \Rightarrow \pi \mid \beta, \kappa \in L_p, \pi \in \mathsf{Plan}\}$$

In the following, we use **Goal**(r) *and* **Bel**(r) *to indicate the goal condition κ and the belief condition β, respectively, that occur in the head of the planning rule $r = (\beta, \kappa \Rightarrow \pi)$.*

In our running example, the agent can have two planning rules `pos1,fuel⇒gs1` and `pos2,fuel⇒gs2`. The first (second) planning rule indicates that the agent should fuel at gas station 1 (2) if he want to fuel and believes that he is in position `pos1` (`pos2`). Given these languages, an agent can be implemented by programming three sets of propositional formulas (representing the agent's beliefs, achievement goals and maintenance goals), and one set of planning rules. For the purpose of this paper, we assume that agents cannot have initial plans, but generate them during their executions.

DEFINITION 3. *(agent program) An agent program is a tuple $(\sigma, \gamma_a, \gamma_m, \mathsf{PL})$ where $\sigma \subseteq L_\sigma, \gamma_a \subseteq L_{\gamma_a}, \gamma_m \subseteq L_{\gamma_m}$ and $\mathsf{PL} \subseteq \mathcal{R}_{\mathsf{PL}}$.*

The agent program for our running example is then a tuple where $\sigma = \{$`pos1`$\}$, $\gamma_a = \{$`(fuel,fuelNoWhere)`$\}$, $\gamma_m = \emptyset$, and
PL $= \{$`pos1,fuel⇒gs1 , pos2,fuel⇒gs2`$\}$. Note that the agent believes it is in position 1 and has no maintenance goal.

2.2 Semantics of *APL*

In the previous section, we described the constructs of an agent-oriented programming language and explained their intuitive meanings. In this section, we present the operational semantics of the programming language 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 state into another and it corresponds to a single computation step. For the semantics of the agent programming language a transition is a transformation of one agent configuration (state) into another.

2.2.1 Agent Configuration

An agent's configuration denotes the state of the agent at one moment in time. It is determined by its mental attitudes, i.e., by its beliefs, goals, plans, and reasoning rules.

DEFINITION 4. *(agent configuration) Let* $\models_p$ *be the propositional entailment relation (used also in the rest of the paper). Let* $\Sigma = \{\sigma \mid \sigma \subseteq L_\sigma, \sigma \not\models_p \bot\}$ *be the set of possible consistent belief bases,* $\Gamma_a = \{(\phi, \psi) \in L_{\gamma_a} \mid \phi \not\models_p \bot\}$ *be the set of achievement goals, and* $\Gamma_m = \{(\phi, \psi) \in L_{\gamma_m} \mid \phi \not\models_p \bot \& \ \psi \not\models_p \bot\}$ *be the set of maintenance goals. An agent configuration is a tuple* $\langle \sigma, (\gamma_a, \gamma_m), \Pi, \mathsf{PL} \rangle$ *where* $\sigma \in \Sigma$ *is the belief base,* $\gamma_a \subseteq \Gamma_a$ *and* $\gamma_m \subseteq \Gamma_m$ *are respectively the achievement and maintain goal bases,* $\Pi \subseteq (L_p \times L_p \times \mathsf{Plan})$ *is the plan base, and* $\mathsf{PL} \subseteq \mathcal{R}_{\mathsf{PL}}$ *is a set of planning rules.*

In the above definition, it is assumed that the belief base of an agent is consistent since otherwise the agent can believe everything which is not a desirable property. Also, each achievement goal (not the corresponding failure condition) is assumed to be consistent since otherwise an agent should achieve an impossible state. The failure condition can be $\bot$ which means that the agent should not drop its achievement goal until it believes that the state denoted by it is reached. Moreover, a maintenance goal (and its corresponding maintain condition) is assumed to be consistent since otherwise an agent should maintain an impossible state (and will never start generating a plan). Finally, the elements of the plan base are defined as 3-tuples ($L_p \times L_p \times \mathsf{Plan}$) consisting of a plan and two goals (propositional formulas) that indicate the reasons for generating the plan. More specifically, (ϕ, κ, π) is added to an agent's plan base if the planning rule $\beta, \kappa \Rightarrow \pi$ is applied because κ is a subgoal of the agent's goal ϕ. Note that the planning rule can be applied only if κ is a subgoal of an agent's goal ϕ. This means that we have $\forall(\phi, \kappa, \pi) \in \Pi : \ \phi \models_p \kappa$. This information will be used to avoid applying a planning rule if it is already applied and the generated plan is not fully executed. In our running example, the agent can apply both planning rules (depending on the agent's beliefs) since the goals of these rules (i.e., `fuel`) is a subgoal of the agent's goal `(fuel,fuelNoWhere)`, i.e., since `fuel`$\models$`fuel`. Note that these rules could also be applied if the agent had more complex goals such as `(fuel`$\wedge$`cw,FuelNoWhere)`; `cw` stands for car wash.

The initial configuration of an agent can be defined based on the agent program (definition 3) that specifies the initial beliefs, goals, and planning rules. As noted, for the purpose of this paper, we assume that an agent does not have initial plans, i.e., the initial plan base is empty.

DEFINITION 5. *(initial configuration) Let* $(\sigma, \gamma_a, \gamma_m, \mathsf{PL})$ *be an agent program. Then, the initial configuration of the agent is* $\langle \sigma, (\gamma_a, \gamma_m), \Pi, \mathsf{PL} \rangle$ *where* $\Pi = \emptyset$.

In the following, for reasons of presentation we do not include the set $\mathcal{R}_{\mathsf{PL}}$ in the agent configuration when possible (as they do not change during the agent executions). Also, we will write γ to represent the whole goal base (γ_a, γ_m) when possible. So, we use $\langle \sigma, \gamma, \Pi \rangle$ or $\langle \sigma, (\gamma_a, \gamma_m), \Pi \rangle$ instead of $\langle \sigma, (\gamma_a, \gamma_m), \Pi, \mathsf{PL} \rangle$.

2.2.2 Transition System $\mathcal{T}$

This subsection presents the transition system $\mathcal{T}$ which consists of transition rules (also called derivation rules) for deriving transitions between an agent's configurations. Each transition rule has the following form indicating that the configuration C can be transformed to configuration C' if the condition of the rule holds.

$$\frac{condition}{C \rightarrow C'}$$

In order to define the operational semantics of the application of planning rules, we define the notions of *relevant* and *applicable* planning rules w.r.t. an agent's goal and its configuration. Intuitively, a planning rule is relevant for an agent's goal if it can contribute to the agent's goal, i.e., if the goal that occurs in the head of the planning rule is a subgoal of the agent's goal. A planning rule is applicable to an agent's goal if it is relevant for that goal and its belief condition is entailed by the agent's configuration.

DEFINITION 6. *(relevant, applicable) Let* $C = \langle \sigma, \gamma, \Pi, \mathsf{PL} \rangle$ *be an agent configuration. For the given configuration C and goal* ϕ*, the set of relevant and applicable planning rules are defined as follows:*
- $rel(\phi, C) = \{r \in \mathsf{PL} \mid \phi \models_p \mathbf{Goal}(r)\}$
- $app(\phi, C) = \{r \in rel(\phi, C) \mid \sigma \models_p \mathbf{Bel}(r)\}$

In the following transition rules we write $\frac{app(\phi)}{C \rightarrow C'}$ instead of $\frac{app(\phi, C)}{C \rightarrow C'}$.

When executing an agent, planning rules will be selected and applied based on its beliefs, goals and plans. The application of planning rules generates plans which can subsequently be selected and executed. Before introducing the transition rules to specify possible agent execution steps, we need to define what it means to perform a plan. For simplicity reasons, we assume here that the performance of a plan affects only the belief base. The effect of plans on the belief base is captured by an update operator `update`, which takes the belief base and a plan and generates the updated belief base. This update operator can be as simple as adding/deleting atoms to/from the belief base. We assume a partial function `update` : $(\mathsf{Plan} \times \Sigma) \rightarrow \Sigma$ that takes a plan and a belief base, and yields the belief base resulting from the execution of the plan on the input belief base (if the update is not successful, the update operation is undefined).

The first transition rule (R_1) captures the case where the plan π is successfully performed. The resulting configuration contains the empty plan and the belief base is updated appropriately.

Rule R_1 (Plan execution rule)

$$\frac{(\phi, \kappa, \pi) \in \Pi \ \& \ \pi \neq \epsilon \ \& \ \mathtt{update}(\sigma, \pi) = \sigma'}{\langle \sigma, \gamma, \Pi \rangle \rightarrow \langle \sigma', \gamma, (\Pi \setminus \{(\phi, \kappa, \pi)\}) \cup \{(\phi, \kappa, \epsilon)\} \rangle}$$

The second transition rule (R_2) captures the case that the performance of the plan has failed. Note that in this case, the failed plan (ϕ, ψ, π) will be removed from the plan base.

Rule R_2 (Plan execution rule)

$$\frac{(\phi, \kappa, \pi) \in \Pi \ \& \ \pi \neq \epsilon \ \& \ \mathtt{update}(\sigma, \pi) = undefined}{\langle \sigma, \gamma, \Pi \rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{(\phi, \kappa, \pi)\} \rangle}$$

For an achievement goal, plans should be generated to reach the state denoted by it. If the generated and performed plans do not achieve the desired state, then the achievement goal remains in the goal base. The first transition rule below (called R_3) is designed to apply planning rules in order to generate plans the execution of which may achieve the subgoals of the achievement goals. A planning rule can be applied if the goal in the head of the rule is not achieved yet, if there is no plan for the same subgoal in the plan base (in order to avoid applying rules if it is already applied), and if the failure condition does not hold. The application of a planning rule will add the plan of the planning rule to the plan base.

Rule R_3 (apply planning rules)

$$\frac{(\phi,\psi) \in \gamma_a \ \& \ (\beta,\kappa \Rightarrow \pi) \in app(\phi) \ \& \\ \nexists \pi' \in \mathsf{Plan} : (\phi,\kappa,\pi') \in \Pi \ \& \ \sigma \not\models \kappa \ \& \ \sigma \not\models \psi}{\langle \sigma,\gamma,\Pi \rangle \rightarrow \langle \sigma,\gamma,\Pi \cup \{(\phi,\kappa,\pi)\}\rangle}$$

The next transition rule (R_4) removes all empty plans, which are generated for the achievement or maintenance goals, from the plan base. Note that this rule is for both achievement and maintenance goals.

Rule R_4 (remove empty plans)

$$\frac{(\phi,\psi) \in \gamma_a \cup \gamma_m \ \& \ (\phi,\kappa,\epsilon) \in \Pi}{\langle \sigma,\gamma,\Pi \rangle \rightarrow \langle \sigma,\gamma,\Pi \setminus \{(\phi,\kappa,\epsilon)\}\rangle}$$

An achievement goal can be dropped under two circumstances: either when its failure condition becomes true, or when the state it denotes is reached. The transition rule R_5 below captures these two cases of dropping the achievement goal. In both cases, besides the removal of the achievement goal, the plans associated with it will be removed.

Rule R_5 (achieved or not achievable goals)

$$\frac{(\phi,\psi) \in \gamma_a \ \& \ (\sigma \models \phi \ or \ \sigma \models \psi)}{\langle \sigma,(\gamma_a,\gamma_m),\Pi \rangle \rightarrow \langle \sigma,(\gamma_a',\gamma_m),\Pi' \rangle}$$

where $\gamma_a' = \gamma_a \setminus \{(\phi,\psi)\}$ and $\Pi' = \Pi \setminus \{(\phi,\kappa,\pi) \mid \kappa \in L_p, \pi \in \mathsf{Plan}\}$.

The state denoted by a maintenance goal should hold at any moment during an agent's execution. In order to maintain the denoted state, plans should be generated and performed. The question is when exactly plans should be generated and performed. A maintenance goal ϕ is enriched with a maintain condition ψ which, if it becomes true, indicates that plans should be generated and executed. It should be noted that there might be no logical relation between the maintenance goals and their maintain conditions. This relation depends usually on the application domain. However, we assume that in all agent configurations, if the maintain condition does not hold, then the maintenance goal holds: $\forall \sigma \in \Sigma, \forall \phi, \psi \in L : (\sigma \not\models \psi) \rightarrow (\sigma \models \phi)$.

The transition rule R_6 is designed to allow the application of planning rules when the maintain condition becomes true. The applied rule should be such that there is no plan already in the plan base for the subgoal that occurs in the head of the selected rule.

Rule R_6 (apply planning rules)

$$\frac{(\phi,\psi) \in \gamma_m \ \& \ (\beta,\kappa \Rightarrow \pi) \in app(\phi) \ \& \\ \nexists \pi' \in \mathsf{Plan} : (\phi,\kappa,\pi') \in \Pi \ \& \ \sigma \models \psi}{\langle \sigma,\gamma,\Pi \rangle \rightarrow \langle \sigma,\gamma,\Pi \cup \{(\phi,\kappa,\pi)\}\rangle}$$

Note that there is no transition rule that removes a maintenance goal from an agent's goal base. This is because an agent's maintain goal should remain in the agent's goal base during its execution. Note also that the removal of empty plans of the maintenance goals is already captured by the transition rule R_4, and that the plans which are generated to maintain goals, are based on the same set of planning rules as used for achieve goals.

2.2.3 *Agent Execution*

In order to define the behaviours of an agent and compare them with each other, we need to define what it means to execute an agent. Given a transition system consisting of a set of transition rules, the execution of an agent is a set of transitions generated by applying the transition rules to the initial configuration of an agent. Thus, the execution of an agent starts with the initial configuration of an agent and generates subsequent configurations that can be reached from the initial configuration by applying a transition rule. Note that the execution of an agent forms a graph in which the nodes are the configurations and the edges indicate the application of a transition rule. In the following, we define the execution of an agent A by first defining the set of all possible transitions $\mathcal{R}_\mathcal{T}$ for all possible agents in the transition system $\mathcal{T}$, and then take the subset of those transitions that can be reached from the initial configuration of agent A.

DEFINITION 7. *(agent execution) Let C be the set of all agent configurations. Then, the set of transitions that are derivable from the transition system $\mathcal{T}$, denoted as $\mathcal{R}_\mathcal{T}$, is defined as follows:*

$\mathcal{R}_\mathcal{T} =$
$\{(c_i, c_j) \mid c_i \Rightarrow c_j$ *is a transition derivable from* $\mathcal{T}$ & $c_i, c_j \in C\}$

Given an agent program A with corresponding initial agent configuration c_0, the execution of A is the smallest set $\mathcal{E}_\mathcal{T}(A)$ of transitions derivable from $\mathcal{T}$ starting from A, i.e., it is the smallest subset $\mathcal{E}_\mathcal{T}(A) \subseteq \mathcal{R}_\mathcal{T}$ such that:

- *if $(c_0, c_1) \in \mathcal{R}_\mathcal{T}$, then $(c_0, c_1) \in \mathcal{E}_\mathcal{T}(A)$ for $c_1 \in C$*
- *if $(c_i, c_j) \in \mathcal{E}_\mathcal{T}(A)$ and $(c_j, c_k) \in \mathcal{R}_\mathcal{T}$, then $(c_j, c_k) \in \mathcal{E}_\mathcal{T}(A)$ for $c_i, c_j, c_k \in C$*

3. CTL_{APL}: A SPECIFICATION LANGUAGE FOR AGENT PROGRAMS

In order to specify (not implement) an agent's behaviour, we will use an instantiation of the BDI_{CTL} logic [3, 10, 11]. This is a multimodal logic consisting of temporal modal operators to specify the evolution of agents' configurations (the agents' execution) through time and epistemic modal operators to specify agents' mental state (beliefs and goals) in each configuration.

For the purpose of this paper to relate the specification and implementation languages, we do not allow the nesting of epistemic operators. This is because the beliefs and goals in the agent programming language presented in this paper are propositional rather than modal formulas. This is, however, not a principle limitation as the representation of beliefs and goals in agent programming languages can be extended to modal formulas [13]. In order to specify the mental state of agents, we will define the language L consisting of non-nested belief and goal formulas.

DEFINITION 8 (SPECIFICATION LANGUAGE L). *The language L for the specification of agents' mental attitudes consists of non-nested belief and goal formulas, defined as follows:*

- *if $\phi \in L_p$, then $B(\phi) \in L$*
- *if $\phi, \psi \in L_p$, then $G_a(\phi,\psi)$ and $G_m(\phi,\psi) \in L$*

3.1 CTL_{apl} Syntax

The behaviour of an agent, generated by the execution of the agent, is a temporal structure overs its mental states. In order to specify the behaviour of agents, we use the standard CTL^* logic [6] in which the primitive propositions are formulas from the language

L defined in Definition 8. The resulting language will be called CTL_{apl} defined as follows.

DEFINITION 9 (SPECIFICATION LANGUAGE CTL_{apl}). *The state and path formulas are defined by the following S and P clauses, respectively.*

S1 Each formula from L is a state formula.

S2 If ϕ and ψ are state formulas, then $\phi \wedge \psi$ and $\neg\phi$ are also state formulas.

S3 If ϕ is a path formula, then $E\phi$ and $A\phi$ are state formulas.

P1 Any state formula is a path formula.

P2 If ϕ and ψ are path formulas, then $\neg\phi$, $\phi \wedge \psi$, $X\phi$, $\diamond\phi$, and $\phi\ U\ \psi$ are path formulas.

Using the CTL_{apl} language, one can for example express that if an agent has an achievement goal, then it will not drop the goal until it believes the goal is achieved or believes that its failure condition holds, i.e., $G_a(\phi,\psi) \rightarrow A(G_a(\phi,\psi)\ U\ (B(\phi) \vee B(\psi)))$.

3.2 CTL_{apl} Semantics

The semantics of the specification language CTL_{apl} is defined on a Kripke structure $M = \langle C, R, V\rangle$, where the set of states C is the set of configurations of agents implemented in the agent programming language APL (definition 4), and the temporal relation R is specified by the transition system $\mathcal{T}$ of the agent programming language APL (definition 7). In particular, a transition between two agent configurations is derivable from the transition system $\mathcal{T}$ if there exists a temporal relation between these two configurations in the Kripke structure. Finally, the valuation function $V = (V_b, V_g^a, V_g^m)$ of the Kripke structure is defined on agent configurations and consists of different valuation functions each with respect to a specific mental attitude of agents' configurations.

More specifically, we define a valuation function V_b that valuates the belief formulas in terms of agents' beliefs, a valuation function V_g^a for the agents' achievement goals, and a valuation function V_g^m for the agents' maintenance goals. The belief valuation function V_b maps an agent configuration c to a set of propositions $V_b(c)$ that are derivable from the agent's belief base. An agent believes a proposition if and only if the proposition is included in $V_b(c)$. The valuation function V_g^a for achievement goals is defined in such a way that all subgoals of an agent's achievement goal are also considered as a goal. The valuation function V_g^a maps an agent's configuration to a set of sets of pairs. Each set of pairs contains all subgoals of an achievement goal. An agent wants to achieve a proposition if and only if the proposition is included in a set of pairs. The valuation function V_g^m is defined in similar way.

The semantics of the CTL_{apl} expressions are defined in the same way as the semantics of CTL^* expressions except that the valuation functions are defined with respect to the agent's belief and goal bases. We omit the semantics of negation and conjunction for reasons of space. These are defined as usual.

DEFINITION 10. *Let $M = \langle C, R, V\rangle$ be a Kripke structure specified by the execution of the transition system $\mathcal{T}$, where:*

- *C is a set of configurations (states) of the following form: $\langle\sigma, (\gamma_a, \gamma_m), \Pi, \mathsf{PL}, \mathcal{T}\rangle$.*
- *$R \subseteq C \times C$ is a serial binary relation, such that for each $(c, c') \in R$, we have $(c, c') \in \mathcal{R}_{\mathcal{T}}$, or $c = c'$.*
- *$V = (V_b, V_g^a, V_g^m)$ are the belief and goal (achieve and maintain) evaluation functions, i.e.,*
  $V_b \colon C \rightarrow 2^{L_p}$, *s.t.,*
  $V_b(\langle\sigma, (\gamma_a, \gamma_m), \Pi, T\rangle) = \{\phi \mid \sigma \models_p \phi\}$
  $V_g^a : C \rightarrow 2^{2^{L_p \times L_p}}$, *s.t.,*
  $V_g^a(\langle\sigma,(\gamma_a, \gamma_m), \Pi, T\rangle) = \{\{(\phi', \psi') \mid \phi \models_p \phi' \ \&\ \psi \equiv \psi'\} \mid (\phi, \psi) \in \gamma_a\}$
  $V_g^m : C \rightarrow 2^{2^{L_p \times L_p}}$, *s.t.,*
  $V_g^m(\langle\sigma, (\gamma_a, \gamma_m), \Pi, T\rangle) = \{\{(\phi', \psi') \mid \phi \models_p \phi' \ \&\ \psi \equiv \psi'\} \mid (\phi, \psi) \in \gamma_m\}$

A fullpath is an infinite sequence $x = c_0, c_1, c_2, \ldots$ of configurations such that $\forall i : (c_i, c_{i+1}) \in R$. We use x^i to indicate the i-th state of the path x.

(S1) $M, c \models B(\phi) \ \Leftrightarrow\ \phi \in V_b(c)$

(S1) $M, c \models G_a(\phi,\psi) \ \Leftrightarrow\ \exists s \in V_g^a(c) : (\phi,\psi) \in s$

(S1) $M, c \models G_m(\phi,\psi) \ \Leftrightarrow\ \exists s \in V_g^m(c) : (\phi,\psi) \in s$

(S3) $M, c \models E\phi \ \Leftrightarrow$
$\exists$ *fullpath* $x = c, c_1, c_2, \ldots \in M : M, x \models \phi$

(S3) $M, c \models A\phi \ \Leftrightarrow$
$\forall$ *fullpath* $x = c, c_1, c_2, \ldots \in M : M, x \models \phi$

(P1) $M, x \models \phi \ \Leftrightarrow\ M, x^0 \models \phi$ *for ϕ is a state formula*

(P2) $M, x \models X\phi \ \Leftrightarrow\ M, x^1 \models \phi$

(P2) $M, x \models \diamond\phi \ \Leftrightarrow\ M, x^n \models \phi$ *for some $n \geq 0$*

(P2) $M, x \models \phi\ U\ \psi \ \Leftrightarrow$
a) $\exists k \geq 0$ *such that*
$M, x^k \models \psi$ *and for all* $0 \leq j < k :\ M, x^j \models \phi$, *or,*
b) $\forall j \geq 0 :\ M, x^j \models \phi$

Note that the two options in clause $P6$ of the above definition capture two interpretations of the until operator. The first (strong) interpretation is captured by the option a and requires that the condition ψ of the until expression should hold at once. The second (weak) interpretation is captured by the option $a + b$ and requires the formula ϕ should hold forever.

In the above definition, the CTL_{apl} state formulas are evaluated in the Kripke model M with respect to a configuration c. As we are interested in expressing that a certain property holds in all agent configurations, we will define the notion of satisfaction in a model.

DEFINITION 11. *(satisfaction in model) A formula ϕ is satisfied in the model M if and only if ϕ holds in all configurations in M, i.e., $M \models \phi \Leftrightarrow_{def} M, c \models \phi$ for every $c \in C$.*

In the following, we model the execution of an agent with the initial configuration A and based on the transition system $\mathcal{T}$ as the Kripke model $M_{\mathcal{T}}^A = \langle C_{\mathcal{T}}^A, R_{\mathcal{T}}^A, V\rangle$ based on which the CTL_{apl} expressions (i.e., properties to be checked) can be evaluated. The accessibility relation $R_{\mathcal{T}}^A$ is defined as the set of executions of the agent A extended with a reflexive accessibility for all end configurations. This is to guarantee the seriality property of the accessibility relation $R_{\mathcal{T}}^A$. Moreover, the set $C_{\mathcal{T}}^A$ of configurations will be defined in terms of configurations that occur in the execution of the agent.

DEFINITION 12. *(agent model) Let A be an agent program and let $\mathcal{E}_{\mathcal{T}}(A)$ be the execution of A. Then the model corresponding with agent A, which we call an* agent model*, is defined as $M_{\mathcal{T}}^A =$*

$\langle C^A_{\mathcal{T}}, R^A_{\mathcal{T}}, V\rangle$, where the accessibility relation $R^A_{\mathcal{T}}$ and the set of configurations $C^A_{\mathcal{T}}$ are defined as follows:

$$\begin{aligned} R^A_{\mathcal{T}} &= \mathcal{E}_{\mathcal{T}}(A) \cup \\ &\quad \{(c_n, c_n) \mid \exists (c_{n-1}, c_n) \in \mathcal{E}_{\mathcal{T}}(A) : \neg\exists (c_n, c_{n+1}) \in \mathcal{E}_{\mathcal{T}}(A)\} \\ C^A_{\mathcal{T}} &= \{c \mid (c, c') \in R^A_{\mathcal{T}}\} \end{aligned}$$

Note that agent models are Kripke structures in the sense of Definition 10.

In section 4, we prove that certain properties hold for any agent program that is implemented in the *APL* programming language. As the above definition of model $M^A_{\mathcal{T}}$ is based on one specific agent A, we need to quantify over all agent programs. Since the binary relation $R^A_{\mathcal{T}}$ (derived from the transition system $\mathcal{T}$, which is the semantics of the agent programs) has to be the same in all Kripke models, a quantification over agent programs means a quantification over models $M^A_{\mathcal{T}}$. This implies that we need to define the notion of validity of a property as being true for all agents and thus all models $M^A_{\mathcal{T}}$.

DEFINITION 13. *(validity) A property $\phi \in CTL_{apl}$ holds for the execution of an arbitrary agent A based on the transition system $\mathcal{T}$, expressed as $\models_{\mathcal{T}}$, if and only if ϕ holds in all agent models $M^A_{\mathcal{T}}$, i.e., $\models_{\mathcal{T}} \phi \Leftrightarrow_{def} \forall A : M^A_{\mathcal{T}} \models \phi$.*

Note that this notion of validity is the same as the notion of validity in modal logic since it is defined at the level of frames, i.e., at the level of states and relation and not valuations, which is in our case defined in terms of specific agents.

4. PROPERTIES

Given the semantics of the programming language *APL* and the specification language CTL_{apl}, we can prove that certain properties expressed in CTL_{apl} hold for agents programmed in *APL*.

4.1 Proving the Properties

In this section, we present a number of desired properties and prove that they hold for arbitrary agents. In section 4.2, we discuss these properties in more detail. The first property expresses that an agent may not drop its goals until it believes it has achieved them, or believes the failure condition holds.

PROPOSITION 1. *(single-minded commitment)*

$$\models_{\mathcal{T}} G_a(\phi, \psi) \rightarrow A(G_a(\phi, \psi)\ U\ (B(\phi) \vee B(\psi)))$$

PROOF. Using Definitions 13 and 11, we derive that we have to prove $M^A_{\mathcal{T}}, c \models G_a(\phi, \psi) \Leftrightarrow M^A_{\mathcal{T}}, c \models A(G_a(\phi, \psi)\ U\ (B(\phi) \vee B(\psi)))$.

Assume $M^A_{\mathcal{T}}, c \models G_a(\phi, \psi)$. Then to prove $\forall$ fullpath $x = c, c_1, c_2, \ldots \in M^A_{\mathcal{T}} : M^A_{\mathcal{T}}, x \models G_a(\phi, \psi)\ U\ (B(\phi) \vee B(\psi))$ (Definition 10). Let $x = c, c_1, \ldots$ an arbitrary fullpath starting in c. Then to prove $M^A_{\mathcal{T}}, x \models G_a(\phi, \psi)\ U\ (B(\phi) \vee B(\psi))$.

We can prove that $G_a(\phi, \psi)$ holds until $B(\phi) \vee B(\psi)$ on the path x, by proving for each configuration x^i on the path x the following:

$$\text{if } M^A_{\mathcal{T}}, x^i \models G_a(\phi, \psi) \quad \text{then} \quad \begin{array}{ll} \text{(I)} & M^A_{\mathcal{T}}, x^{i+1} \models G_a(\phi, \psi) \text{ or} \\ \text{(II)} & M^A_{\mathcal{T}}, x^{i+1} \models B(\phi) \vee B(\psi). \end{array} \tag{1}$$

That is, if $G_a(\phi, \psi)$ holds, then this keeps on holding in the next state, or else $B(\phi) \vee B(\psi)$ holds. If the former is continuously the case, then we have that $\forall j : M^A_{\mathcal{T}}, x^j \models G_a(\phi, \psi)$, i.e., case b of clause ($P6$) of Definition 10. If the latter is the case for some x^k, then we have that $M^A_{\mathcal{T}}, x^k \models B(\phi) \vee B(\psi)$, and for all $0 \leq j < k$ we have $M^A_{\mathcal{T}}, x^j \models G_a(\phi, \psi)$, since $M^A_{\mathcal{T}}, c \models G_a(\phi, \psi)$ by assumption, i.e., we have case a of clause ($P6$).

Assume $M^A_{\mathcal{T}}, x^i \models G_a(\phi, \psi)$, i.e., $\exists s \in V^a_g(x^i) : (\phi, \psi) \in s$, i.e., $\exists (\phi', \psi') \in \gamma^i_a : \phi' \models_p \phi$ and $\psi' \equiv \psi$, where γ^i_a is the achievement goal base of x^i. Given these properties of the configuration x^i, we have to show that any next configuration x^{i+1} has the stated desired properties. Since the accessibility relation of $M^A_{\mathcal{T}}$, $R^{\mathcal{T}}_A$, is based on $\mathcal{T}$, we know that $x^i \rightarrow x^{i+1}$ should be derivable in $\mathcal{T}$, or $x^i \rightarrow x^{i+1}$ is a reflexive edge. In the latter case, we have (I). In the former case, we can see that the only transition rule of $\mathcal{T}$ through which the goal base can be changed is R_5.

Assume that $x^i \rightarrow x^{i+1}$ is a transition derived using transition rule R_5. This means that some goal (ϕ'', ψ'') has been removed from γ^i_a. If $(\phi'', \psi'') \neq (\phi', \psi')$, then we have $(\phi', \psi') \in \gamma^{i+1}_a$, and therefore $M^A_{\mathcal{T}}, x^{i+1} \models G_a(\phi, \psi)$ (I). If $(\phi'', \psi'') = (\phi', \psi')$, then it must be the case that $\sigma^i \models_p \phi'$ or $\sigma^i \models_p \psi'$, where σ^i is the belief base of x^i. Since R_5 does not update the belief base, it must be the case that $\sigma^{i+1} \models_p \phi'$ or $\sigma^{i+1} \models_p \psi'$. Since $\phi' \models_p \phi$ and $\psi' \equiv \psi$, we have that $\sigma^{i+1} \models_p \phi$ or $\sigma^{i+1} \models_p \psi$. Therefore, we have that $M^A_{\mathcal{T}}, x^{i+1} \models B(\phi) \vee B(\psi)$ (II), by Definition 10.

Otherwise, goals do not change and we have case (I), yielding the desired result. $\square$

The next property specifies that if the failure condition of an achievement goal is $\perp$, the agent does not drop this goal until it believes it has achieved the goal.

PROPOSITION 2. *(blind commitment)*

$$\models_{\mathcal{T}} G_a(\phi, \perp) \rightarrow A(G_a(\phi, \perp)\ U\ B(\phi))$$

PROOF. By Proposition 1, we have $\models_{\mathcal{T}} G_a(\phi, \perp) \rightarrow A(G_a(\phi, \perp)\ U\ (B(\phi) \vee B(\perp)))$. By Definition 4, we have that belief bases are consistent, i.e., $\neg\exists M^A_{\mathcal{T}}, c : M^A_{\mathcal{T}}, c \models B(\perp)$. We thus have that $B(\perp) \equiv \perp$, and as $\varphi \vee \perp \equiv \varphi$ for any formula φ, we have $B(\phi) \vee B(\perp) \equiv B(\phi)$, yielding the desired result. $\square$

We now proceed to give a definition of intention, and show how intentions defined in this way are related to an agent's goals. We define that an agent intends κ, if κ follows from the second component of one of the plans in an agent's plan base. The second component of a plan specifies the subgoal for which the plan was selected, and it is these subgoals for which the agent is executing the plans, that we define to form the agent's intentions. This is analogous to the way the semantics of intention is defined in [2].

DEFINITION 14. *(intention) Let $V_i : C \rightarrow 2^{2^{L_p}}$ be defined as $V_i(\langle\sigma, (\gamma_a, \gamma_m), \Pi\rangle) = \{\{\kappa' \mid \kappa \models_p \kappa'\} \mid (\phi, \kappa, \pi) \in \Pi\}$. Then $M, c \models I(\kappa)$ is defined as $\exists s \in V_i(c) : \kappa \in s$.*

Given this definition, we prove that an agent's intentions are a "subset" of the agent's goals, i.e., we prove the following proposition.

PROPOSITION 3. *(intentions)*

$$\models_{\mathcal{T}} I(\kappa) \rightarrow G_a(\kappa, \psi) \textit{ for some } \psi$$

PROOF. In order to prove the proposition, we prove something stronger. Informally, if we can prove for every configuration that if a plan is in the plan base, then there is also a corresponding goal in the goal base, we have the desired result. Formally, we need to show $\forall c : \forall (\phi, \kappa, \pi) : ((\phi, \kappa, \pi) \in \Pi_c \Rightarrow \exists \psi : (\phi, \psi) \in \gamma^c_a)$ where Π_c and γ^c_a are the plan base and goal base of configuration c, respectively. We prove this result using induction.

Let c be an initial agent configuration, i.e., $\neg\exists c' : (c', c) \in R^A_{\mathcal{T}}$. Then the plan base of c is empty, which means that $\neg\exists (\phi, \kappa, \pi) : (\phi, \kappa, \pi) \in \Pi_c$. This yields the desired result in the base case.

Let c be an arbitrary configuration and let $(c, c') \in R^A_{\mathcal{T}}$. We then have to show:

$$\forall(\phi, \kappa, \pi) : \quad ((\phi, \kappa, \pi) \in \Pi_c \Rightarrow \exists\psi : (\phi, \psi) \in \gamma^c_a) \Rightarrow$$
$$((\phi, \kappa, \pi) \in \Pi_{c'} \Rightarrow \exists\psi : (\phi, \psi) \in \gamma^{c'}_a)$$

Take an arbitrary plan (ϕ, κ, π). Assume $(\phi, \kappa, \pi) \in \Pi_c \Rightarrow \exists\psi : (\phi, \psi) \in \gamma^c_a$. Then to prove $((\phi, \kappa, \pi) \in \Pi_{c'} \Rightarrow \exists\psi : (\phi, \psi) \in \gamma^{c'}_a)$. We distinguish two cases, i.e., the case in which $(\phi, \kappa, \pi) \in \Pi_c$, and the case in which $(\phi, \kappa, \pi) \notin \Pi_c$:

Assume $(\phi, \kappa, \pi) \in \Pi_c$. Assume $\neg\exists\psi : (\phi, \psi) \in \gamma^{c'}_a$ (contraposition). Then to prove: $(\phi, \kappa, \pi) \notin \Pi_{c'}$. We then have $\exists\psi : (\phi, \psi) \in \gamma^c_a$. As we have $\neg\exists\psi : (\phi, \psi) \in \gamma^{c'}_a$, we know that a goal of the form (ϕ, ψ) was removed from γ^c_a over the transition to c'. Goals can only be removed using transition rule R_5. This rule removes only one goal at a time. We thus know that there was only one goal with ϕ as its first element in γ^c_a. Transition rule R_5 specifies that if a goal (ϕ, ψ) is removed from the goal base, all plans of the form (ϕ, κ', π') have to be removed from the plan base. We thus have in particular that $(\phi, \kappa, \pi) \notin \Pi_{c'}$, yielding the desired result in this case. The case for $(\phi, \kappa, \pi) \notin \Pi_c$ is proven in a similar way using transition rule R_3. □

Note that the opposite of Proposition 3 does not hold, as it can be the case that an agent has a goal for which it has not yet selected a plan.

PROPOSITION 4 (MAINTENANCE GOALS). $\models^{\mathcal{T}}_A G_m(\phi, \psi) \rightarrow A\Box G_m(\phi, \psi)$

PROOF. The property can be proven using the fact that there is no transition rule in $\mathcal{T}$ that drops a maintenance goal. □

4.2 Discussion

4.2.1 Relation with BDI logic

The properties we have proven in the previous section, are not just arbitrary properties. They are closely related to properties that have been presented as desired properties of goals and intentions in the influential BDI logics [3, 10]. The properties we prove, however, are not exactly the corresponding properties discussed in [3, 10]. In this section, we discuss how the properties proven in the previous section differ from those specified mainly in [10]. Note that we have proven the properties by reasoning about the semantics of the CTL_{apl} language, i.e., we did not define a deductive system. Also, the properties we have proven are properties that hold for all agents in the *APL* language. Properties of particular *APL* agents, such as liveness properties, would typically be proven using model checking or some deductive system.

Property 1 is a variant of what has been termed "single-minded commitment" in [10]. The property of single-minded commitment expresses a certain level of persistency of or commitment towards goals, i.e., an agent should not drop its goals until it believes it has achieved them, or believes they are unachievable. Property 1 differs from single-minded commitment of [10] in several ways. First, Rao and Georgeff define single-minded commitment for intentions, rather than goals. Goals are also present in their framework, but are contrasted with intentions in that the agent is not necessarily committed to achieving its goals (but is committed in some way to achieving its intentions).

In agent programming frameworks that incorporate goals and some form of intentions, commitment strategies such as single-minded commitment are generally attributed to goals, rather than to intentions (see, e.g., [7, 12]). The kind of entity in agent programming frameworks that is sometimes referred to as "intention", is generally some kind of procedural entity (a plan) [9], and does not have the level of commitment of goals. On the contrary, a plan is generally removed after its execution, regardless of the effect of the plan. It is precisely the goals that are used for providing the agent with a desired level of persistency in behavior. One could thus argue that goals as used in agent programming frameworks are actually intentions as addressed in BDI logics, and that a component corresponding with the goals in BDI logics, i.e., a component from which intentions are selected, is missing in agent programming frameworks.

Another difference is that where in [10] single-minded commitment defines that an agent cannot drop a goal until it believes it has achieved it or *it believes it is unachievable*, Proposition 1 specifies that an agent cannot drop a goal until it believes it has achieved it or *it believes that the failure condition associated with the goal holds*. In agent programming frameworks, the intuition behind the failure condition is that it expresses a property of the state of the agent (or of the environment) from which it is not possible to reach the goal, i.e., the idea is that if the failure condition holds, the goal is unachievable [12]. The advantage of using such a failure condition that can be evaluated on the current state only, is that the agent does not have to reason about its future courses of action, checking whether a goal is achievable or not. The disadvantage is that it puts a burden on the programmer, who has to formulate these failure conditions.

A further difference is that single-minded commitment in [10] is defined regarding intentions about the future. That is, the property is formulated as $\text{INTEND}(A \diamond \phi) \rightarrow A(\text{INTEND}(A \diamond \phi)\ U\ (B(\phi) \vee \neg B(E \diamond \phi)))$. In this BDI logic, the semantics of INTEND is defined using an accessibility relation that is defined over a temporal structure. Here, it is intuitive that intentions express desired properties of future states. In agent programming frameworks such as the one presented in this paper, however, the $G_a(\phi, \psi)$ operator is not evaluated over a temporal structure, but the semantics is defined on the basis of the goal base of the current state. Nevertheless, the idea of achievement goals in agent programming frameworks is, of course, that these represent a desired future state of affairs.

Proposition 2 is a variant of what has been termed "blind commitment" in [10], and what are called "persistent goals" in [3]. This property expresses that an agent should not drop its goals before it believes to have achieved them. Regarding this property, we remark the following. In [10], it is suggested that the various kinds of commitments yield various kinds of agents, i.e., an agent can, e.g., be blindly committed. In the agent programming framework presented in this paper, the level of commitment can vary from goal to goal. That is, blind commitment can be modeled as a certain kind of single-minded commitment using $\perp$ as the failure condition. We thus did not have to introduce an additional kind of goal with a "blind commitment semantics". This could be considered an advantage.

Proposition 3 intuitively specifies that intentions are a subset of the goals. This is a variant of a property of the BDI logic of [10]. In [10], the property is defined for so-called O-formulas, i.e., κ is then an O-formula. An O-formula is a formula containing no positive occurrences of the temporal A operator outside the scope of the modal operators for belief, goal, and intention. Intuitively, this property holds for the programming language of this paper, since "intentions" or plans are generated on the basis of goals, i.e., a plan cannot be created without a corresponding achievement goal. Moreover, if a goal is removed, its corresponding plans are also removed. Note that while the commitment strategies were defined for intentions in [10] and hold for goals in our framework, the property of the BDI logic that relates goals and intentions *does* map directly to goals and (what we have defined as) intentions in our framework.

4.2.2 *Robustness of the Properties*

The fact that the properties of Section 4.1 hold for agents programmed in the presented programming language, depends on how we have defined the semantics of the programming language, and also on how we have defined the semantics of the CTL_{apl} operators. In this section, we discuss whether the proven properties continue to hold if we make certain changes to the semantics of the programming language and/or the semantics of CTL_{apl}.

An important issue has to do with whether goals may already be believed to be achieved. An alternative semantics of the operator $G_a(\phi, \psi)$ could be obtained by adding the condition "$\phi \notin V_b(c)$ and $\psi \notin V_b(c)$" to the semantics, expressing that the formula $G_a(\phi, \psi)$ holds if ϕ or ψ is not already believed to be achieved. Such a semantics is often defined for goal operators in agent programming languages (see, e.g., [7]). If we would have defined the semantics of the operator in our CTL_{apl} language in this way, the property of single-minded commitment would still hold. The formula $G_a(\phi, \psi)$ would then not hold anymore as soon as either ϕ or ψ are believed, which is exactly what we want in order to achieve single-minded commitment. We have, however, not defined the semantics of CTL_{apl} in this way for reasons of simplicity.

Another issue has to do with the semantics of the U operator. The semantics of the until operator is defined as the so-called "weak until". That is, in order for $\phi\ U\ \psi$ to hold, it is allowed that ϕ keeps holding indefinitely. If the semantics of the until operator would have been defined as a strong until, it would have to be the case that ψ holds at some point. Under the strong until, the property of single-minded commitment does not hold. The reason is that it is not guaranteed that the agent will drop its goal. It can be the case that a goal (ϕ, ψ) remains in the goal base without ϕ or ψ ever becoming true. This is in contrast with properties attributed to goals in [3, 10], as in those logics there can be no so-called "infinite deferral", meaning that the agent has to drop its goal at some point in time.

Further, we make a remark regarding the property of Proposition 3. This property holds, partly because plans are removed from the plan base as soon as a corresponding goal is removed from the goal base. Therefore it cannot be the case that there is a plan in the plan base without there being a corresponding goal in the goal base. In, e.g., the semantics of 3APL as defined in [1], plans remain in the plan base, even if a corresponding goal is removed. Under such a semantics of the programming language, Proposition 3 thus does not hold.

5. CONCLUSION

In this paper, we showed how a BDI-based agent-oriented programming language can be related to a BDI specification language. The relation allows us to prove that certain properties expressed in the specification language hold for the agent programming language, and thus for all individual agents that are implemented in this agent programming language. We used here a very simple agent programming language which can be extended in many different ways. In fact, we have already developed a BDI-based agent-oriented programming languages with first-order belief and goal expressions, complex plans and reasoning rules. For this agent programming language we aim at proving desirable properties similar to those discussed in this paper. Doing this we can ensure that the semantics of our agent programming language is defined and implemented correctly. This work can thus be considered as a first step in that direction. In [2], a comparable effort is undertaken for the agent programming language AgentSpeak. The specification language in that work is, however, not a temporal logic, and the properties proven are different (not related to dynamics of goals). At this moment our agent programming language is not expressive enough to support the implementation of composite mental attitudes such as beliefs about beliefs and goals (or goals to have certain beliefs). This was the reason why we modified and simplified the BDI_{CTL} specification language. Extending the agent programming language with operators that can be nested makes it possible to prove also these kinds of properties for the programming language. It should be noted that our focus in this paper was on individual agent properties. Future research is needed to study the multi-agent system properties. Finally, we are currently developing a logic for an agent programming language (similar to one presented in this paper) with sound and complete axiomatisation that allows reasoning about agent programs. Such a logic can be used to verify safety and liveness properties of particular agent programs that are executed according to a certain execution strategy, e.g., executing multiple plans in parallel and interleaving mode or executing single plans atomically.

6. REFERENCES

[1] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.

[2] Rafael H. Bordini and Alvaro F. Moreira. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. *Electronic Notes in Theoretical Computer Science*, 70(5), 2002.

[3] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42, 1990.

[4] M. Dastani, D. Hobo, and J.-J. Ch Meyer. Practical extensions in agent programming languages. In *Proceedings of AAMAS 2007*. ACM Press, 2007.

[5] Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch. Meyer. Goal types in agent programming. In *Proceedings of ECAI'06*, 2006.

[6] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[7] K.V. Hindriks, F.S. de Boer, W. van der Hoek, and J.-J. Ch Meyer. Agent programming with declarative goals. In *Proceedings of ATAL'2000)*, LNAI.

[8] J.-J. Ch Meyer, W. van der Hoek, and B. van Linder. A logical approach to the dynamics of commitments. *Arificial Intelligence*, 113:1–40, 1999.

[9] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J.W. Perram, editors, *Agents Breaking Away (LNAI 1038)*, 1996.

[10] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of KR'91*, pages 473–484, 1991.

[11] K. Schild. On the relationship between BDI-logics and standard logics of concurrency. *Autonomous agents and multi-agent systems*, 3:259–283, 2000.

[12] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proc. of KR'02*, 2002.

[13] Laurens Winkelhagen, Mehdi Dastani, and Jan Broersen. Beliefs in agent implementation. In *Proceedings DALT 2005, LNCS 3904*. Springer, 2006.

[14] Michael Woolridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., 2002.