A Programming Language for Cognitive Agents Goal Directed 3APL

Mehdi Dastani Birna van Riemsdijk Frank Dignum John-Jules Ch. Meyer
Institute of Information and Computing Sciences
Utrecht University
The Netherlands
{ mehdi , birna , dignum , jj }@cs.uu.nl

ABSTRACT

This paper presents the specification of a programming language for cognitive agents. This programming language is an extension of 3APL (An Abstract Agent Programming Language) and allows the programmer to implement agents' mental attitudes like beliefs, goals, plans, and actions, and agents' reasoning rules by means of which agents can modify their mental attitudes. The formal syntax and semantics of this language is presented as well as a discussion on the deliberation cycle and an example.

1. INTRODUCTION

In research on agents, besides architectures, the areas of agent theories and agent programming languages are distinguished. Theories concern descriptions of (the behavior of) agents. Agents are often described using logic [8, 13]. Concepts that are commonly incorporated in such logics are for instance knowledge, beliefs, desires, intentions, commitments, goals and plans.

It has been argued in the literature that it can be useful to analyze and specify a system in terms of these concepts [4, 11, 17]. If the system would however then be implemented using an arbitrary programming language, it will be difficult to verify whether it satisfies its specification: if we cannot identify what for instance the beliefs, desires and intentions of the system are, it will be hard to check the system against its specification expressed in these terms. This is referred to by Wooldridge as the problem of ungrounded semantics for agent specification languages [16]. It will moreover be more difficult to go from specification to implementation if there is no clear correspondence between the concepts used for specification and those used for implementation.

To support the practical development of intelligent agents, several programming languages have thus been introduced that incorporate some of the concepts from agent logics. First there is a family of languages that use actions as their starting point to define commitments (Agent-0, [12]), intentions (AgentSpeak(L), [9]) and goals (3APL, [5]). All of these languages however lacked an important element of BDI ([10]) or KARO ([14]) like (declarative) logics, which incorporate a declarative notion of goals. Having the notion of goals separate from structures built from actions, has the advantage that one can describe pro-active behavior of an agent. To bridge this gap, in [15], the language Dribble was proposed which constitutes a synthesis between the declarative and the procedural approaches, combining both notions in one and the same programming language. Dribble is however a propositional language without variables, which severely limits its programming power. In this paper, we propose an extension of the language 3APL, inspired by Dribble, with declarative goals *and* first order features. Furthermore, whereas in Dribble one can use goals for plan selection only, in this extension of 3APL we add rules for reasoning with goals. We will refer to the extension of 3APL presented in this paper, simply with the same name 3APL.

In the extended version of 3APL we consider the notion of procedural goals (used in [5]), to be reduced to that of plans, which are selected to achieve declarative goals. So, this version of 3APL provides formal constructs to implement an agent's beliefs, goals and plans. Of course, to solve the problem of ungrounded semantics for 3APL agents one should be able to implement an agent's intentions as well. However, in this paper for simplicity reasons we concentrate only on declarative goals. A discussion on the notion of intention and how to incorporate it in 3APL is discussed in [3]. In order to implement the dynamic behavior of 3APL agents, one needs formal constructs by means of which goals and plans are selected, plans executed, reasoning and planning rules are applied, etc. The language which is needed to implement such issues is called the deliberation language. The behavior of 3APL agents can be implemented by means of a deliberation cycle which is an expression of the deliberation language. More details on the formal specification of the deliberation language can be found in [2].

In the next section we introduce the syntax of the extended version of 3APL and indicate some of the important (new) features. In section 3 we describe the operational semantics of 3APL using state transitions. In section 4 we indicate a number of issues to be dealt with at the deliberation level of goal directed agents. In section 5 we give an example to illustrate the use of the various programming constructs of 3APL. We give some conclusions and areas for further research in section 6.

2. SYNTAX

2.1 Beliefs and goals

The *beliefs* of a 3APL agent describe the situation the agent is in. The belief base can contain information the agent believes about the world and it can contain information that is internal to the agent. The *goals* of the agent on the other hand, denote the situation the agent wants to realize. The beliefs and goals can be represented by a first-order domain language. The terms of the domain language represent the domain objects and its formulae represent the relations between domain objects. In the sequel, a language defined by inclusion shall be the smallest language containing the specified elements. The canonical tautology $\top$ shall be an element of the domain language and the belief and goal languages which are defined below. It is defined to mean $\phi \vee \neg\phi$ where ϕ is a formula of one of the above-mentioned languages. The canonical contradiction $\bot$ is defined to be $\neg\top$.

DEFINITION 1. *(domain language) Let VAR, $FUNC$, and $PRED$ be the sets of domain variables, functions (functions with no arguments are constants), and predicates, respectively, and $n \geq 0$. The terms T_L of the domain language L are defined as follows:*

- *if $x \in VAR$, then $x \in T_L$,*
- *if $f \in FUNC$ and $t_1, \ldots, t_n \in T_L$, then $f(t_1, \ldots, t_n) \in T_L$.*

The formulae of the domain language L consist of the following expressions:

- *if $p \in PRED$ and $t_1, \ldots, t_n \in T_L$, then $p(t_1, \ldots, t_n) \in L$ for $n \geq 0$,*
- *if $\phi, \phi' \in L$, then $\neg\phi, \phi \wedge \phi' \in L$,*
- *if $x_1, \ldots, x_n \in VAR$ and $x_1, \ldots, x_n$ occur in ϕ, then $\forall x_1, \ldots, x_n \phi \in L$*

A 3APL agent has a belief base and a goal base, both being sets of formulas from this domain language. In the rules which will be defined in the sequel, one needs to be able to refer to sentences from the belief base or goal base. Therefore, we define the following belief language and goal language on top of the domain language.

DEFINITION 2. *(beliefs and goals) Let L be the domain language. Then, the belief language L_B and the goal language L_G are defined as follows:*

- *The belief language L_B:*
  - *if $\phi \in L$, then $\mathbf{B}\phi \in L_B$,*
  - *if $\beta, \beta' \in L_B$, then $\beta \wedge \beta' \in L_B$*
- *The goal language L_G:*
  - *if $\phi \in L$, then $\mathbf{G}\phi \in L_G$,*
  - *if $\kappa, \kappa' \in L_G$, then $\kappa \wedge \kappa' \in L_G$*

The belief and goal languages do not include negation. Main reason is to avoid a number of the problems occurring with substitutions and unifications concerning negation. They are very similar to those occurring in logic programs with negation. Although some (partial) solutions thus could be used from that area they would complicate the current paper unnecessary. We therefore left negation as a separate issue to be dealt with in a subsequent paper.

2.2 Plans

In order to reach its goals, a 3APL agent adopts *plans*. A plan is a sequence built from basic elements. The basic elements can be *basic actions*, *tests* on the belief base or *abstract plans* (sometimes called achievement goals [5]). As in the languages GOAL and 3APL, basic actions specify the capabilities with which an agent should achieve a certain state of affairs. The effect of execution of a basic action is not a change in the world, but a change in the belief base of the agent. An abstract plan cannot be executed directly in the sense that it updates the belief base of an agent. Abstract plans serve as an abstraction mechanism like procedures in imperative programming. If a plan consists of an abstract plan, this abstract plan could be transformed into basic actions through reasoning rules. Abstract plans can be represented using plan names. Plans are ultimately sequences of actions that are executed in a domain. To refer to objects in this domain, abstract plans are plan names which can be parameterized with these domain objects. We thus use a set of plan names $PNAME = \{\rho_1, \rho_2, \ldots\}$ which can be used to define the set of abstract plans AP: $AP = \{\rho(t_1, \ldots, t_n) \mid \rho \in PNAME, t_1, \ldots, t_n \in T_L, n \geq 0\}$. Moreover, we assume a set of basic action names $ANAME = \{a_1, a_2, \ldots\}$ which are used to define the set of basic actions $ACT = \{a(t_1, \ldots, t_n) \mid a \in ANAME, t_1, \ldots, t_n \in T_L, n \geq 0\}$. Finally, we use E to denote the empty plan.

DEFINITION 3. *(plans) Let $\beta \in L_B$. The plan language L_P consists of the following elements:*

- *empty plan: $E \in L_P$,*
- *basic action: $ACT \subseteq L_P$,*
- *test: $\beta? \in L_P$,*
- *abstract plan: $AP \subseteq L_P$,*
- *composite plans: if $\pi_1, \pi_2 \in L_P$, then $\pi_1; \pi_2$, `if` β `then` π_1 `else` π_2 `fi` , `while` β `do` π_1`od` $\in L_P$*

Moreover, if E, $\pi \in L_P$ it holds that $E; \pi = \pi; E = \pi$.

2.3 Rules

We propose various rules to reason with goals, plans, and their interactions. These rules are conditionalized by beliefs.

DEFINITION 4. *(rules) Let $\beta \in L_B$, $\kappa, \kappa_h, \kappa_b \in L_G$, and $\pi, \pi_h, \pi_b \in L_P$. We define sets of reasoning rules for goals GR, plans PR, and their interactions IR as follows:*

- *$\kappa_h \leftarrow \beta \mid \kappa_b \in GR$,*
- *$\kappa \leftarrow \beta \mid \pi \in IR$,*
- *$\pi_h \leftarrow \beta \mid \pi_b \in PR$.*

The *goal rules* are used to revise, generate or drop goals. For example, the goal rule $\mathbf{G}(on(x,y)) \leftarrow \mathbf{B}(tooHeavy(x) \wedge notHeavy(z)) \mid \mathbf{G}(on(z,y))$ can be used to revise one of an agent's goals: it informally means that if the agent desires to have block x on block y, but it believes that x is too heavy while z is not heavy, then it should revise its goal and aim to have block z on block y. The goal rules can also be used to generate, extend or drop goals by using the following general forms, respectively:

- $\top \leftarrow \beta \mid \kappa_b$ for goal generation,
- $\kappa_h \leftarrow \beta \mid \kappa_h \wedge \kappa_b$ for goal extension,
- $\kappa_h \leftarrow \beta \mid \top$ for dropping goals,

It is important to note that maintenance of goals can be modelled by goal rules of the form $\top \leftarrow \top \mid \kappa$.

The *interaction rules* are used to generate plans to achieve goals. They are similar to the goal rules of Dribble. For example, the interaction rule $\mathbf{G}(on(x,z)) \leftarrow \mathbf{B}(on(x,y)) \mid move(x,y,z)$ states that if the agent desires to have block x on block z, but it believes that x is on block y, then it plans to move x from y and put it on z. The belief condition thus indicates when the plan could be selected to achieve the specified goal. Interaction rules can also be used to model reactive behavior with rules of the form $\top \leftarrow \beta \mid \pi$.

Finally, the *plan rules*, which are similar to the practical reasoning rules of 3APL, are used to revise and drop plans For example, the plan rule $move(x,y,z) \leftarrow \mathbf{B}(\neg clear(x)) \mid on(u,x)?; move(u,x,Fl); move(x,y,z)$ informally means that if the agent plans to move block x from block y onto block z, but it cannot move x because (it believes that) there is a block on x, then the agent should revise its plan by finding out which block (u) is on x, moving u onto the floor, and finally moving x from y onto z. Plan rules are not used to generate plans. This is because we assume that a plan is only generated to achieve a certain goal. The generation of plans is therefore restricted to the interaction rules. The general forms of the rules for modifying and dropping plans are similar to those of the goal rules.

2.4 A 3APL configuration

Above, the beliefs, goals and plans of a 3APL agent were defined. These are the elements that change during the execution of the agent. Together with a fourth *substitution* component, they constitute a 3APL configuration. The substitution is used to store values or bindings associated with first order variables. In the sequel, we will use $Free(e)$ to indicate the set of free variables (not bound by a quantifier) that occur in the syntactic element e and $dom(\theta)$ to denote the set of variables that form the domain of the substitution θ.

DEFINITION 5. *(configuration) A configuration of a 3APL agent is a tuple $\langle \sigma, \gamma, \Pi, \theta \rangle$, where $\sigma \subseteq L$ is the belief base of the agent, $\gamma \subseteq L$ is the goal base of the agent, $\Pi \subseteq L_P \times L_G$ is the plan base of the agent*[1]*, and θ represents the substitution that binds domain variables to domain terms, i.e. $\theta \subseteq \{ [x_i/t_i] \mid x_i \in VAR\ ,\ t_i \in T_L, x_i \notin Free(t_j), \forall i \neq j\ x_i \neq x_j \}$. Moreover, the belief and goal bases are assumed to be grounded, i.e. they consist of formulae in which no free variables occur. Finally, the belief base and the goal base of agents are such that for any goal $\phi \in \gamma$ it holds:*

- *$\sigma \not\models \phi$, i.e. the goal ϕ is not entailed by the agent's beliefs,*
- *$\phi \not\models \bot$, i.e. the goal ϕ is consistent,*
- *$\sigma \not\models \bot$, i.e. the agent's beliefs are consistent.*

[1]Note that with each plan the (initial) goal to be achieved by the plan is associated.

In this definition, we have defined Π as consisting of plan-goal formula pairs. The goal for which a plan is selected is recorded with the plan, because this for instance provides the possibility to drop a plan of which the goal is reached. Furthermore, goals may be revised or dropped and one might want to remove a plan associated with a goal which has been dropped, from the plan base.

The rationale behind the conditions on belief base and goal base is the following. The beliefs of an agent describe the state the agent is in and the goals describe the state the agent wants to realize. If an agent believes ϕ is the case, it cannot have the goal to achieve ϕ, because the state of affairs ϕ is already realized. Furthermore, the individual goals in the goals base must be consistent: an agent cannot desire a state which is not logically possible. The entire goal base does not have to be consistent, because two inconsistent goals could both be realized, although not at the same time. A consequence of this is, that an agent cannot adopt the conjunction of two separate goals in the goal base as a new goal, for this could lead to inconsistent goals (see definition 7). Finally, the belief base must be consistent, because an agent with an inconsistent belief base would believe everything and could therefore hardly function.

2.5 A 3APL agent

To program a 3APL agent means to specify its initial beliefs and goals and to write sets of goal rules, interaction rules and plan rules. This is formalized in the specification of a 3APL agent.

DEFINITION 6. *(3APL agent) A 3APL agent is a tuple $\langle \sigma_0, \gamma_0, GR, IR, PR \rangle$ where $\langle \sigma_0, \gamma_0, \emptyset, \emptyset \rangle$ is the initial configuration, GR is a set of goal rules, IR is a set of interaction rules and PR is a set of plan rules.*

The plan base of the agent is empty in the initial mental state. This choice reflects the idea that an agent should start to act because it has certain goals or because it reacts to a certain situation using a reactive interaction rule.

3. SEMANTICS

We define an operational semantics for 3APL in terms of a transition system ([7]). A transition system is a set of derivation rules for deriving transitions. A transition is a transformation of one configuration into another and it corresponds to a single computation step.

3.1 Semantics of belief and goal formulae

In order to define the semantics of the various rules, we first need to define the semantics of the belief and goal formulae.

DEFINITION 7. *(semantics of belief and goal formulae) Let $\langle \sigma, \gamma, \Pi, \theta \rangle$ be an agent configuration, ϕ, $\phi' \in L$ and φ, $\varphi' \in L_B \cup L_G$.*

- $\langle \sigma, \gamma, \Pi, \theta \rangle \models \mathbf{B}\phi \Leftrightarrow \sigma \models \phi$
- $\langle \sigma, \gamma, \Pi, \theta \rangle \models \mathbf{G}\phi \Leftrightarrow \exists \phi' \in \gamma : \phi' \models \phi$ *and* $\sigma \not\models \phi$
- $\langle \sigma, \gamma, \Pi, \theta \rangle \models \varphi \wedge \varphi' \Leftrightarrow$
  $\langle \sigma, \gamma, \Pi, \theta \rangle \models \varphi$ *and* $\langle \sigma, \gamma, \Pi, \theta \rangle \models \varphi'$

As explained above, the semantics of $\mathbf{G}\phi$ is defined in terms of separate goals, as opposed to defining it in terms of the entire goal base. The idea is, that all logical consequences of a particular goal are also goals, but only if they are not believed ([6]).

3.2 Transition system

In the following, a set of derivation rules is proposed that specifies the semantics of various ingredients of 3APL. These rules specify the semantics of a 3APL agent with a set of goal rules GR, a set of plan rules PR and a set of interaction rules IR. In the rules, variables and their values play an important role and substitutions are applied to syntactic elements. We assume the usual notions of ground substitutions and application of a substitution as also defined in [5].

The first derivation rule specifies the execution of the plan base of a 3APL agent. The plan base of the agent is a set of plan-goal pairs. This set can be executed by executing one of the constituent plans. The execution of a plan can change the agent's configuration.

DEFINITION 8. *(plan base execution) Let* $\Pi = \{(\pi_1, \kappa_1), \ldots, (\pi_i, \kappa_i), \ldots, (\pi_n, \kappa_n)\} \subseteq L_P \times L_G$ *and* $\Pi' = \{(\pi_1, \kappa_1), \ldots, (\pi'_i, \kappa_i), \ldots, (\pi_n, \kappa_n)\} \subseteq L_P \times L_G$ *be plan bases,* θ, θ' *be ground substitutions, and* $V = Free(\Pi)$*. Then, the derivation for the execution of a set of plans is specified in terms of the execution of individual plans as follows:*

$$\frac{\langle \sigma, \gamma, (\pi_i, \kappa_i), \theta \rangle_V \rightarrow \langle \sigma', \gamma', (\pi'_i, \kappa_i), \theta' \rangle}{\langle \sigma, \gamma, \Pi, \theta \rangle \rightarrow \langle \sigma', \gamma', \Pi', \theta' \rangle}$$

Transitions for individual plans are parameterized by the set of free variables V of the entire plan base Π. This is necessary because in the transition rules for individual plans, sometimes reference needs to be made to this set.

In the following, we use *Goal* as a function of type $L_G \rightarrow \wp(L)$ that removes the $\mathbf{G}$ modalities from a goal formula returning its goals from L. For example, $Goal(\mathbf{G}(p(a)) \wedge \mathbf{G}(q(b))) = \{p(a), q(b)\}$. Now we will introduce the derivation rules for the execution of individual plans. We introduce derivation rules for two types of basic elements of plans: basic actions and tests. We do not introduce derivation rules for abstract plans, because abstract plans cannot be executed. They can only be transformed using plan rules.

DEFINITION 9. *(basic action execution) Let* $\alpha \in ACT$ *and let* $\mathcal{T}$ *be a function that specifies the belief update resulting from the execution of basic actions, then the execution of a single action is specified as follows:*

$$\frac{\mathcal{T}(\alpha\theta, \sigma) = \sigma' \ \&\ \langle \sigma, \gamma, \{(\alpha, \kappa)\}, \theta \rangle \models \kappa}{\langle \sigma, \gamma, (\alpha, \kappa), \theta \rangle_V \rightarrow \langle \sigma', \gamma', (E, \kappa), \theta \rangle}$$

where $\gamma' = \gamma \backslash \{\phi \in \gamma \mid \ \phi \in Goal(\kappa) \ \&\ \sigma' \models \phi\}$.

Note that the condition $\langle \sigma, \gamma, \{(\alpha, \kappa)\}, \theta \rangle \models \kappa$ guarantees that the action can only be executed if the goal for which α was selected is still entailed by the current configuration. This condition might be considered too strong. An alternative is, to remove the condition from this transition rule. The decision of whether to execute plans of which the goal is not entailed by the current configuration, could then be lifted to the deliberation cycle (see section 4). The function $\mathcal{T}$ is assumed to preserve consistency of the belief base (see definition 5).

The substitution θ is used to instantiate free variables in the basic action α. Note also that the effect of the execution of basic actions is first of all a belief update. If goals in the goal base are realized through the execution of the action, these goals are removed from the goal base.

The derivation rule for the execution of the test can bind the free variables that occur in the test formula for which no bindings have been computed yet.

DEFINITION 10. *(test execution) Let* $\beta \in L_B$ *and let* τ *be a ground substitution such that* $dom(\tau) = Free(\beta\theta)$*, then*

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\theta\tau}{\langle \sigma, \gamma, (\beta?, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (E, \kappa), \theta\tau \rangle}$$

Having explained the test execution, we can now state the following assumption about configurations. Let $z \in ACT \cup AP$. For each configuration $\langle \sigma, \gamma, \Pi, \theta \rangle$ we assume that for each plan π with $(\pi, \kappa) \in \Pi$ and for each z occurring in π, one of the following properties holds for all $x \in Free(z)$:
- $x \in dom(\theta)$,
- z is preceded by a test β? where $x \in Free(\beta\theta)$.

The idea is, that variables in basic actions or abstract plans should either be bound by a substitution or they should be preceded by a test through which this binding can be computed. The reason for this assumption is, that the meaning is not clear of for instance the execution of a basic action with variables without a binding. The same goes for abstract plans.

In the semantics of composite plans and rules, we will need the notion of a variant. A syntactic element e is a variant of another element e' in case e can be obtained from e' by renaming of variables. We will use variants of plans or rules to avoid unwanted bindings between variables in those plans or rules and variables in the plan base (V) or in $dom(\theta)$.

The derivation rules for the execution of composite plans are defined recursively in the standard way below.

DEFINITION 11. *(execution of composite plans) Let* τ *be a ground substitution such that* $dom(\tau) = Free(\beta\theta)$*. Let* π' *be a variant of* π *such that no free variables in* π' *occur in* V *or* $dom(\theta)$*. The following transitions specify the execution of different types of composite plans.*

$$\frac{\langle \sigma, \gamma, (\pi_1, \kappa), \theta \rangle_V \rightarrow \langle \sigma', \gamma', (\pi'_1, \kappa), \theta' \rangle}{\langle \sigma, \gamma, (\pi_1; \pi_2, \kappa), \theta \rangle_V \rightarrow \langle \sigma', \gamma', (\pi'_1; \pi_2, \kappa), \theta' \rangle}$$

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\ \theta\tau}{\langle \sigma, \gamma, (\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (\pi_1\tau, \kappa), \theta \rangle}$$

$$\frac{\neg\exists\tau : \langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\ \theta\tau}{\langle \sigma, \gamma, (\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (\pi_2, \kappa), \theta \rangle}$$

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\ \theta\tau}{\langle \sigma, \gamma, (\texttt{while}\ \beta\ \texttt{do}\ \pi\ \texttt{od}, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (\pi\tau; \texttt{while}\ \beta\ \texttt{do}\ \pi'\ \texttt{od}, \kappa), \theta \rangle}$$

$$\frac{\neg\exists\tau : \langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\ \theta\tau}{\langle \sigma, \gamma, (\texttt{while}\ \beta\ \texttt{do}\ \pi\ \texttt{od}, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (E, \kappa), \theta \rangle}$$

Note that the goal associated with some plan is passed on unchanged through the transitions modifying this plan.

We will now define the transition rules for the reasoning rules. A goal rule $\kappa_h \leftarrow \beta \mid \kappa_b$ is applicable if its head

is derivable from the agent's goal base and its condition is derivable from the agent's belief base. The application of the goal rule only affects the goal base of the agent, i.e. the goal base of the agent is revised according to the goal rule.

DEFINITION 12. *(goal rule application) Let $\kappa_h, \kappa_b \in L_G$, $\beta \in L_B$, η, τ be ground substitutions such that $dom(\eta) = Free(\kappa_h)$ and $dom(\tau) = Free(\beta\eta)$. Let the rule $\kappa_h \leftarrow \beta \mid \kappa_b$ be a variant of a goal rule $\in GR$ such that no free variables in the rule occur in $dom(\theta)$ and $Free(\kappa_b) \subseteq Free(\kappa_h) \cup Free(\beta)$. Then the transition rule for the goal rule $\kappa_h \leftarrow \beta \mid \kappa_b$ is defined as follows:*

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \kappa_h\eta \ \&\ \langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\eta\tau \ \&\ \forall \phi'' \in Goal(\kappa_b) : \sigma \not\models \phi''\eta\tau}{\langle \sigma, \gamma, \Pi, \theta \rangle_V \rightarrow \langle \sigma, \gamma', \Pi, \theta \rangle}$$

where $\gamma' = (\gamma \backslash \{\phi \in \gamma \mid \phi' \in Goal(\kappa_h) \text{ and } \phi \models \phi'\eta\}) \cup \{\phi''\eta\tau \mid \phi'' \in Goal(\kappa_b)\}$.

The effect of the application of the goal rule on the goal base is that it removes goals of which a goal in the head of the rule is a logical consequence (for all substitutions τ) from the goal base, and adds the goals in the body of the goal rule to the goal base (for any substitutions τ and η).

A plan rule $\pi_h \leftarrow \beta \mid \pi_b$ is applicable if its head π_h unifies with a plan of the agent and its condition β is derivable from agent's beliefs. We assume that the revised plan π_b is designed to achieve the same goal. Therefore, the goal associated with plan π_h in the plan base will be associated with the revised plan π_b as well. The application of a plan rule only affects the plan base of the agent, i.e. the plan to which the plan rule is applied, is revised.

DEFINITION 13. *(plan rule application) Let $\pi_h, \pi_b \in L_P, \kappa \in L_G, \beta \in L_B$, $\pi_h \leftarrow \beta \mid \pi_b$ be a variant of a plan rule $\in PR$ such that no free variables in the rule occur in V or $dom(\theta)$, η be a most general unifier for π and π_h, and θ, τ be ground substitution such that $dom(\tau) = Free(\beta\eta)$. Then,*

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\eta\tau \ \&\ \langle \sigma, \gamma, \{(\alpha, \kappa)\}, \theta \rangle \models \kappa}{\langle \sigma, \gamma, (\pi, \kappa), \theta \rangle_V \rightarrow \langle \sigma, \gamma, (\pi_b\eta\tau, \kappa), \theta \rangle}$$

The effect of the application of the plan rule on the plan base is that the plan π is replaced by the body π_b of the plan rule instantiated with the substitution η, that resulted from matching the head of the rule with the revised plan, and with the substitution τ that resulted from matching the condition of the rule with the belief base. Note that the substitution θ is not updated by the substitution τ because the body of the rule is a variant and does not contain any variable occurring in Π or $dom(\theta)$. This implies that all bindings in τ are about new variables that occur only in the body of the rule. τ can therefore be applied directly to π_b.

An interaction rule $\kappa \leftarrow \beta \mid \pi$ specifies that the goal κ can be achieved by plan π if β is derivable from the agent's beliefs. An interaction rule only affects the plan base of the agent.

DEFINITION 14. *(interaction rule application) Let $\kappa \in L_G$, $\beta \in L_B$, $\pi \in L_P$, $\kappa \leftarrow \beta \mid \pi$ be a variant of an interaction rule $\in IR$ such that no free variables in the rule occur in V or $dom(\theta)$, and η, τ be ground substitutions such that $dom(\eta) = Free(\kappa)$ and $dom(\tau) = Free(\beta\eta)$. Then,*

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models \kappa\eta \ \&\ \langle \sigma, \gamma, \Pi, \theta \rangle \models \beta\eta\tau}{\langle \sigma, \gamma, \Pi, \theta \rangle_V \rightarrow \langle \sigma, \gamma, \Pi \cup \{(\pi\eta\tau, \kappa\eta)\}, \theta \rangle}$$

Note that the goal $\kappa\eta$ that should be achieved by the plan $\pi\eta\tau$ is associated with it. It is only this rule that associates goals with plans.

The goal base of the agent does not change because the plan $\pi\eta\tau$ is not executed yet; the goals of agents change only after execution of plans. We do not add substitution τ to θ since this substitution should only influence the new plan π.

3.3 Semantics of a 3APL agent

The semantics of a 3APL agent is derived directly from the transition relation $\rightarrow$. The meaning of a 3APL agent consists of a set of so called computation runs.

DEFINITION 15. *(computation run) A computation run $\mathtt{CR}(s_0)$ for a 3APL agent is a finite or infinite sequence $s_0, \ldots, s_n$ or $s_0, \ldots$ where s_i are configurations, and $\forall_{i>0}$: $s_{i-1} \rightarrow s_i$ is a transition in the transition system for the 3APL agent.*

The meaning of a 3APL agent $\langle \sigma_0, \gamma_0, GR, PR, IR \rangle$ is the set of computation runs $\mathtt{CR}(\langle \sigma_0, \gamma_0, \emptyset, \emptyset \rangle)$. Note that the first state of the computation runs is the initial mental state of the 3APL agent.

4. DELIBERATION CYCLE

In the previous sections we have described the syntax and semantics of 3APL. However, in order to run 3APL we also need an interpreter that determines the order in which rules are applied, when actions should be performed, when belief updates should be made, etc. This interpreter is not fixed in 3APL but is itself a program again. This deliberation module for 3APL without the declarative goals was described already in [1].

The addition of declarative goals will, however, substantially influence the deliberation cycle. Although a complete discussion of all issues falls outside the scope of this paper (and no space is available for such a discussion) we describe some of the prominent topics to be dealt with during the deliberation.

First of all one has to make choices about which types of rules to apply at what moment in time. Do we apply goal rules (changing current goals) whenever applicable or do we only invoke those rules when it seems the current goals are not reachable using any possible plan and using any possible planning rule. The latter leads to what is called "blindly committed" agents in [10]. Some more moderate alternatives are also possible. E.g. create a plan for a goal (using an interaction rule) and use the planning rules in order to perform this plan. If this leads to a stage where no planning rule can be used any more and the goal is not reached, then one can change the goal using a goal rule. So, this leads to a strategy where one plan is tried completely (including all possible rewrites depending on the situation) and if it fails the goal is abandoned.

At the deliberation level we also have to check the relation between plans and goals. Although we check whether a goal still exists during the plan execution and thus avoid continuing with a plan while a goal is reached (or dropped), we still keep the plan itself. It is up to the deliberation module to perform a kind of "garbage collection" and remove a left-over plan for a goal that no longer exists. If this would not be done the left-over plan would become active again as soon as the goal would be established at any later time.

The last issue that we will describe in this paper is that of having multiple (parallel) goals and/or plans. First one should decide whether only one or more plans can be derived for the same goal at any time. If we allow only one current plan for each goal, the plans in the plan base will all be for different goals.
In this case one has to determine whether the plans will be executed interleaved or consecutively. Interleaving might be beneficial, but can also lead to resource contention between plans in a way that no plan executes successfully anymore. E.g. a robot needs to go to two different rooms that lay in opposite directions. If it has a plan to arrive in each room and interleaves those two plans it will keep oscillating around its starting position indefinitely. Many of the existing work on concurrent planning can, however, be applied straight away in this setting to avoid most problems in this area.

Although many issues arise at this level, they can all be reduced to determining the order in which the rules are applied. In [1] the basic constructs needed to program this level were indicated . The same constructs can be used to write programs to tackle the issues indicated above.

The semantics of a 3APL agent was specified in section 3.3. This definition could be extended to include a certain programmed deliberation cycle. The resulting semantics should then define a subset of the traces of the most general semantic specification of section 3.3. As we however did not formally specify the constructs with which the deliberation cycle can be programmed, we cannot formulate this extension of the definition.

5. EXAMPLE

Our example agent has to solve the problem of building a tower of blocks. The blocks have to be stacked in a certain order: block C has to be on the floor, B on C and block A on B. Initially, the blocks A and B are on the floor, while C is on A. The only action an agent can perform, is to move a block x from some block y onto another block z ($move(x, y, z)$). The action is enabled only if the block to be moved (x) and the block onto which x is moved (z) are clear. The result of the action is, that x is on z and not on y, block y becomes clear and block z is not clear anymore (assuming that z is not the floor, because the floor is always clear). In this example, we assume the agent only has one plan in its plan base regarding this task. Otherwise, different plans for this task could interfere with each other in unwanted ways (this problem could be solved on the deliberation level, but this is outside the scope of this article). Interaction rules can thus only be applied if the relevant plan of the agent is empty. Let

$$\begin{array}{lcl}\sigma_0 & = & \{on(A, Fl) \wedge on(B, Fl) \wedge on(C, A) \wedge clear(B) \\ & & \wedge\ clear(C) \wedge clear(Fl), \\ & & on(x, y) \wedge y \neq Fl \rightarrow \neg clear(y)\}, \\ \gamma_0 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}.\end{array}$$

A 3APL agent can solve the tower building problem with the following rules ($i \in IR, p_1, p_2 \in PR$).

$$\begin{array}{ll} i: & \mathbf{G}(on(x, z)) \leftarrow \mathbf{B}(on(x, y)) \mid move(x, y, z) \\ p_1: & move(x, y, z) \leftarrow \mathbf{B}(\neg clear(x)) \mid \\ & on(u, x)?; move(u, x, Fl); move(x, y, z) \\ p_2: & move(x, y, z) \leftarrow \mathbf{B}(\neg clear(z)) \mid \\ & on(u, z)?; move(u, z, Fl); move(x, y, z)\end{array}$$

The interaction rule is used to derive the $move(x, y, z)$ action that should be executed to fulfil a goal $on(x, z)$. The preconditions of the move action are not checked in this rule, so it is possible that the derived action cannot be executed in a particular configuration. The plan rules can then be used to create a configuration in which this action *can* be executed. Note that the interaction rule is used to select an action to fulfil a goal of the form $on(x, z)$. The initial goal base however contains a conjunction of $on(x, z)$ predicates. The interaction rule is applicable to this conjunction, because a formula $\mathbf{G}\phi$ is true if ϕ is a logical consequence of a goal in the goal base, but only if ϕ is not believed by the agent.

Plan rule p_1 can be applied to an action $move(x, y, z)$ if the condition that x is clear is not satisfied which means that the action cannot be executed. Rule p_2 can be applied if z is not clear. The plan rules with head $move(x, y, z)$ construct a plan to create a configuration in which the move action can be executed. Rule p_1 for example specifies that if x is not clear, a $move(x, y, z)$ action should be replaced by the plan $on(u, x)?; move(u, x, Fl); move(x, y, z)$: first bind u to the block that is on top of x, then clear x by moving u, then move x.

In the initial configuration of the agent $\langle \sigma_0, \gamma_0, \emptyset, \emptyset \rangle$, three possible substitutions of interaction rule i can be computed: $\tau = \{x/A,\ y/Fl,\ z/B\}$ or $\{x/B,\ y/Fl,\ z/C\}$ or $\{x/C,\ y/A,\ z/Fl\}$ (yielding $move(A, Fl, B), move(B, Fl, C)$ or $move(C, A, Fl)$). Suppose the first substitution is chosen. After application of this interaction rule, the plan of the agent becomes the plan in the consequent of the rule after application of τ. The goal $on(A, B)$ is moreover associated with the plan, resulting in the following plan base (other components of the initial configuration do not change):

$$\Pi = \{(move(A, Fl, B), \mathbf{G}(on(A, B)))\}.$$

The plan cannot be executed because the preconditions of the action are not satisfied in this configuration (block A is not clear). The interaction rule cannot be applied because the plan of the agent is not empty. The only applicable rule is the plan rule p_1 where $\eta = \{x/A, y/Fl, z/B\}$, resulting in the following plan base:

$$\begin{array}{lcl}\Pi & = & \{(on(u, A)?; move(u, A, Fl); move(A, Fl, B), \\ & & \mathbf{G}(on(A, B)))\}.\end{array}$$

The only option is to execute the test. The substitution $\tau = \{u/C\}$ is computed and added to the empty substitution of the current configuration: $\theta = \{u/C\}$. Then the action $move(C, A, Fl)$ is executed (the substitution θ is applied to the action). The modified components of the agent's configuration are as follows:

$$\begin{array}{lcl}\sigma & \models & on(A, Fl) \wedge on(B, Fl) \wedge on(C, Fl) \wedge clear(A) \wedge \\ & & clear(B) \wedge clear(C) \wedge clear(Fl), \\ \Pi & = & \{(move(A, Fl, B), \mathbf{G}(on(A, B)))\}, \\ \theta & = & \{u/C\}.\end{array}$$

In the above configuration, the action $move(A, Fl, B)$ is executed. After a number of other test and action executions and rule applications, the agent reaches the final configuration. In this configuration, the goal is reached and thus removed from the goal base:

$$\begin{array}{lcl}\sigma_F & \models & on(A, B) \wedge on(B, C) \wedge on(C, Fl) \wedge clear(A) \wedge \\ & & clear(Fl), \\ \gamma_F & = & \emptyset, \\ \Pi_F & = & \emptyset, \\ \theta_F & = & \{u/C, v/A\}.\end{array}$$

During the execution, a substitution θ_F is computed with

$v \in dom(\theta_F)$. We assume variable u of plan rule p_1 was renamed to v in the creation of a variant of p_1. The example execution shows that the 3APL agent can reach its initial goal. The agent will however not always take the shortest path. The length of the path depends on which choices are made if multiple substitutions can be computed for the interaction rule.

In this example, we did not use any goal rule in order to keep it simple. However, in a domain where blocks for instance have weights, a goal rule could be added to drop goals involving blocks which are too heavy. Suppose the belief base of an agent contains a formula $weight(x, n) \wedge (n > 3) \rightarrow tooHeavy(x)$ to indicate that a block x is too heavy for this agent if its weight exceeds 3 and suppose it contains the formula $weight(A, 5)$. The following goal rule could then be used to drop for instance a goal $on(A, B) \wedge on(B, C)$ (a second rule would of course have to be added for the y-part of an $on(x, y)$ formula).

$g: \quad \mathbf{G}(on(x, y)) \leftarrow \mathbf{B}(tooHeavy(x)) \mid \top$

The substitution $\eta = \{x/A, y/B\}$ is computed and goals of which $on(A, B)$ is a logical consequence, are dropped.

6. CONCLUSION AND FUTURE RESEARCH

In this paper we have described the syntax and semantics of an agent programming language that includes all the classical elements of the theory of agents. I.e. beliefs, goals and plans (or intentions). We thus conjecture that it should be possible to verify whether a 3APL program satisfies a given specification in terms of beliefs, goals and plans. It should moreover be easier to go from analysis and specification in terms of these concepts to implementation. These are however issues that remain for future research.

An interpreter for the basic form of 3APL is already implemented and extensions are currently being programmed. The interpreter will enable us to evaluate the effectiveness of the language for problems of realistic complexity.

In this paper we only sketched a number of issues for the deliberation cycle of 3APL agents. Especially determining the balance between reactive and pro-active behavior and how to capture this in programming structures on the deliberative level will be an important issue for further research.

In the version of the paper to be submitted for the post-proceedings, we will include a comparison with other languages and architectures, such as PRS, DMARS and AgentSpeak(L).

7. REFERENCES

[1] M. Dastani, F. de Boer, F. Dignum, and J. C. Meyer. Programming agent deliberation. In *Proceedings of AAMAS*, Melbourne, Australia, July 2003.

[2] M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation: An approach illustrated using the 3apl language. In *Proceedings of The Second Conference on Autonomous Agents and Multi-agent Systems (AAMAS'03)*, Melbourne, Australia, 2003.

[3] M. Dastani, F. Dignum, and J.-J. Meyer. Autonomy and agent deliberation. In *Proceedings of The First International Workshop on Computatinal Autonomy - Potential, Risks, Solutions (Autonomous 2003)*, Melbourne, Australia, 2003.

[4] D. Dennet. *The intentional stance*. The MIT Press, Cambridge, 1987.

[5] K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Int. J. of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[6] K. Hindriks, F. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In N. Jennings and Y. Lesperance, editors, *Intelligent Agents VI - Proceedings of ATAL'2000*, LNAI-1757. Springer, Berlin, 2001.

[7] G. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, Computer Science Department, 1981.

[8] A. Rao and M. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann, 1991.

[9] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.

[10] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.

[11] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[12] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[13] W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. An integrated modal approach to rational agents. In M. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*, Applied Logic Series 14, pages 133–168. Kluwer, Dordrecht, 1998.

[14] B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Formalizing abilities and opportunities of agents. *Fundameta Informaticae*, 34(1,2):53–101, 1998.

[15] M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in Dribble: from beliefs to goals with plans. In *Proceedings of AAMAS*, Melbourne, Australia, July 2003.

[16] M. Wooldridge. *An introduction to multiagent systems*. John Wiley and Sons, LTD, West Sussex, 2002.

[17] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1994.