A Programming Language for Cognitive Agents Goal Directed 3APL

Mehdi Dastani M. Birna van Riemsdijk Frank Dignum
John-Jules Ch. Meyer

Institute of Information and Computing Sciences
Utrecht University
The Netherlands

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 [9,15]. 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 [5,12,20]. 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 [19]. 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, [14]), intentions (AgentSpeak(L), [10]) and goals (3APL, [6]). All of these languages however lacked an important element of BDI ([11]) or KARO ([16]) 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 [17], 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 [6]), 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 [4]. 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 [3]. 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 [3].

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 beliefs of 3APL agents are specified by its belief base, which contains information the agent believes about the world as well as information that is internal to the agent. The *goals* of the agent on the other hand, denote the situation the agent wants to realize. It is specified by an agent's goal base, which contains information about its preferences. The beliefs and goals of 3APL agents can be specified in terms of a base language which is a first-order language. The terms of the base language represent the domain objects and its formulae represent the relations between the domain objects. In the sequel, a language defined by inclusion is the smallest set containing the specified elements.

Definition 1. *(base language) Let $Var, Func$, and $Pred$ be the sets of domain variables, functions and predicates, respectively. Let $n \geq 0$. The terms of the base language, $Term$, are defined as follows, where functions with no arguments are constants:*

- *if $x \in Var$, then $x \in Term$,*
- *if $f \in Func$ and $t_1, \ldots, t_n \in Term$, then $f(t_1, \ldots, t_n) \in Term$.*

The base language L contains only atomic formulae and is defined as follows:

- *if $p \in Pred$ and $t_1, \ldots, t_n \in Term$, then $p(t_1, \ldots, t_n) \in L$,*

In the following, we will use the standard notion of a *ground formula*. This is a formula not containing variables. Furthermore, a *closed formula* is a formula in which all variables are bound by a quantifier. The belief and goal bases are defined in terms of the expressions of the base language.

Definition 2. *(belief and goal base language) Let $\psi, \psi_1, \ldots, \psi_n \in L$ be ground formulae and let $\phi, \phi_1, \ldots, \phi_n \in L$. The belief base language, BB, and the goal base language, GB, of a 3APL agent are sets of formulae defined on the base language L as follows:*

- *ψ, $\forall_{x_1,\ldots,x_n}(\phi_1 \wedge \ldots \wedge \phi_n \rightarrow \phi)\ \in BB$,*
- *$\psi_1 \wedge \ldots \wedge \psi_m \in GB$*

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

In the rules which will be defined in the sequel, one needs to be able to refer to formulae that are derivable from the belief base or goal base. Therefore, we define the following belief query and goal query languages on top of the base language.

Definition 3. *(belief and goal queries) Let L be the base language. Then, the belief query language L_B with typical formula β and the goal query language L_G with typical formula κ are defined as follows:*

- *if $\phi_1, \ldots, \phi_n \in L$, then $\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n), \neg\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n) \in Disjunction$,*
- *$\top \in Disjunction$,*
- *if $\delta, \delta' \in Disjunction$, then $\delta \overrightarrow{\vee} \delta' \in Disjunction$,*
- *if $\delta \in Disjunction$, then $\delta \in L_B$,*
- *if $\beta, \beta' \in L_B$, then $\beta \overrightarrow{\wedge} \beta' \in L_B$,*

- *if $\phi_1, \ldots, \phi_n \in L$, then $\mathbf{G}(\phi_1 \wedge \ldots \wedge \phi_n) \in L_G$,*
- *$\top \in L_G$,*
- *if $\kappa, \kappa' \in L_G$, then $\kappa \overrightarrow{\wedge} \kappa' \in L_G$.*

The belief query language is a kind of conjunctive normal form, where formulas of the form $\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n)$ are the "atoms". As will become clear in the sequel when we define the semantics of belief and goal queries (see definition 15), the "disjunction" and "conjunction" operators are not commutative. To indicate this, we use the special symbol $\overrightarrow{\vee}$ and $\overrightarrow{\wedge}$, respectively.

The goal query language does not include negation. The main reason for this is the use of such query expressions in the 3APL language. In particular, such a query can occur in the goal revision rules, which intuitively modify existing goals, i.e. they modify only goal expressions without negation.

2.2 Plans

In order to reach its goals, a 3APL agent adopts *plans*. A plan is built from basic elements. The basic elements can be *basic actions*, *tests* on the belief base or *abstract plans* (sometimes called achievement goals [6]).

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 the execution of a basic action is not a change in the world, but a change in the belief base of the agent.

A test action checks if a certain formula is derivable from the beliefbase.

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.

As abstract plans can be transformed into basic actions and basic actions are executed in a domain, both basic actions and abstract plans can be parameterized with terms that denote the domain objects. To be more specific, abstract plans are plan names which can be parameterized with terms (denoting domain objects). We thus use a set of plan names $PName = \{q_1, q_2, \ldots\}$ which are used to define the set of abstract plans $AP = \{q(t_1, \ldots, t_n) \mid q \in PName, t_1, \ldots, t_n \in Term, n \geq 0\}$ with typical element ρ. 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 $Act = \{a(t_1, \ldots, t_n) \mid a \in AName, t_1, \ldots, t_n \in Term, n \geq 0\}$ with typical element α.

Definition 4. *(plans) Let $\beta \in L_B$. The plan language L_P consists of the following elements:*

- *basic action: $Act \subseteq L_P$,*
- *test: $\beta? \in L_P$,*
- *abstract plan: $AP \subseteq L_P$,*
- *if $\beta?, \pi \in L_P, \alpha \in Act, \rho \in AP$, then $\alpha;\pi$, $\beta?;\pi$, $\rho;\pi$ $\in L_P$,*
- *composite plans: if $\pi, \pi_1, \pi_2 \in L_P$, then*
  `if` β `then` π_1 `else` π_2 `fi`, `if` β `then` π_1 `else` π_2 `fi`;π $\in L_P$, *and*
  `while` β `do` π_1 `od`, `while` β `do` π_1 `od`;π $\in L_P$.

We use E to denote the empty plan, which is an empty list and we identify $E; \pi$ with π. In the sequel, we will use $\circ$ to indicate that a plan is a sequential composition of two plans, i.e. $\pi_1 \circ \pi$ denotes a plan in which π_1 is a plan followed by the second plan π (π_1 is the prefix of the plan $\pi_1 \circ \pi$).

2.3 Rules

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

Definition 5. *(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 to revise goals and plans, and to select plans. These rules are called goal revision rules (GR), plan revision rules (PR), and plan selection rules (PS), respectively.*

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

The *goal revision rules* are used to revise, generate or drop goals. For example, the goal revision 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 revision 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,
- $\kappa_h \leftarrow \beta \mid \kappa_h \overset{\rightarrow}{\wedge} \kappa_b$ 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 revision rules of the form $\top \leftarrow \top \mid \kappa$.

The *plan selection rules* are used to generate plans to achieve goals. They are similar to the goal rules of Dribble. For example, the plan selection 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. Plan selection rules can also be used to model reactive behavior with rules of the form $\top \leftarrow \beta \mid \pi$.

Finally, the *plan revision rules*, which are similar to the practical reasoning rules of 3APL, are used to revise and drop plans For example, the plan revision rule $move(x,y,z) \leftarrow \neg\mathbf{B}(clear(x)) \mid on(u,x)?; move(u,x,Fl); move(x,y,z)$ 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.

2.4 Plan Safety and Rule Safety

In this subsection, we will explain and define the concepts of plan safety and rule safety, which will be used to ensure that applying rules cannot result in an unground goalbase or beliefbase (containing atomic formulae with variables that are not bound), and ill-defined plans (containing actions with variables that are not bound). As these notions are syntactic, they are defined in this section. The definitions in this section do not have to be mastered by the reader in order for him/her to understand the general ideas of the semantics.

In the sequel, we will assume that all rules are *safe*. The idea of the requirement of rule safety starts with the concept of *plan safety* (definition 8). The intuition behind the notion of a safe plan is that variables in the basic actions occurring in the plan should either be bound by a substitution or they should be preceded by a test through which a binding will be computed. The reason for this requirement is that it is not clear what it means to execute a basic action with variables without a binding. For example, it is not clear how to specify the semantics of the basic action $move(x, y)$ in a sensible way if x and y are variables without a binding. We require the same condition on variables that occur in abstract plans, as these can be transformed into basic actions using plan revision rules.

In order to define the concept of a safe plan, we define a function yielding the so called safe variables of a belief query. This function takes a belief query formula and yields the variables that will be bound by any substitution under which the query would succeed (see definition 11 for the definition of a (ground) substitution and see definition 15 for the semantics of a belief query under a substitution). The idea thus is, that this function returns those variables of a belief query, that will definitely get a "value" if the query succeeds.

The reason that not all variables in a belief query will get a value if the query succeeds, is that we can pose queries such as $\neg\mathbf{B}(p(x))$. Informally, this query will succeed if the agent does not believe that $p(x)$ holds, i.e. there is no possible value a for x, such that the agent believes $p(a)$. A query such as $\mathbf{B}(p(x))$ however, will always return a value for x, if the query succeeds. We can furthermore pose queries such as $\mathbf{B}(p(x) \overrightarrow{\vee} q(y))$, which will succeed if either a value a for x can be found such that the agent believes $p(a)$, or alternatively, if a value b for y can be found such that the agent believes $p(b)$. The definition of the function $safeVar$ for belief queries reflects these considerations.

Below, we also specify functions yielding the safe variables of goal queries and of plans. These are needed in the definition of the notion of a safe rule and will be explained later in more detail.

Definition 6. *(variables, safe variables) We define $Var_f(e)$[1] to be the set of variables occurring in the syntactic expression e. Moreover, the function $safeVar$:*

[1] Note that we used Var in definition 1 to denote the set of variables of the base language. In this definition, Var is a function yielding the set of variables occurring in some expression. We use the subscript f to denote that we are referring to the *function Var*.

$L_B \rightarrow \wp(Var)$ is then defined as follows.

$$\begin{aligned}
safeVar(\mathbf{B}(\phi)) &= Var_f(\phi)\\
safeVar(\neg\mathbf{B}(\phi)) &= \emptyset\\
safeVar(\beta \overset{\rightarrow}{\vee} \beta') &= safeVar(\beta) \cap safeVar(\beta')\\
safeVar(\beta \overset{\rightarrow}{\wedge} \beta') &= safeVar(\beta) \cup safeVar(\beta')
\end{aligned}$$

We assume a similar function $safeVar : L_G \rightarrow \wp(Var)$ for goal queries. For plans, we define the following function $safeVar : L_P \rightarrow \wp(Var)$ with $\alpha \in Act$, $\rho \in AP$ and $\pi, \pi' \in L_P$, where π' is not of the form $\alpha;\pi$ or $\rho;\pi$.

$$\begin{aligned}
safeVar(\alpha;\pi) &= Var_f(\alpha) \cup safeVar(\pi)\\
safeVar(\rho;\pi) &= Var_f(\rho) \cup safeVar(\pi)\\
safeVar(\pi') &= \emptyset
\end{aligned}$$

The function can be generalized, yielding a function $safeVar : \wp(L_B \cup L_G \cup L_P) \rightarrow \wp(Var)$ as follows: $safeVar(Expr) = \bigcup_{e \in Expr} safeVar(e)$.

In order to be able to define the notion of a safe plan, we also need the concept of free variables of a plan. The free variables of a plan π are those variables occurring in abstract plans or basic actions in π, that are not "preceded" by a test through which these variables will be bound for certain. In the specification of the free variables of a plan, we thus use the definition of the safe variables of a belief query. For example, in the plan $\mathbf{B}(p(x))?; do(x)$ the variable x in the basic action *do* will have a value after the execution of the test preceding this action, since x is in the set of safe variables of $\mathbf{B}(p(x))$. Below, we define a function yielding the free variables of a plan.

Definition 7. *(free variables of a plan) Let $\alpha \in Act$, $\rho \in AP$, $\pi, \pi_1, \pi_2 \in L_P$ and $\beta \in L_B$. Let the functions Var_f and $safeVar$ be as in definition 6. The function $Free : L_P \rightarrow \wp(Var)$ is then defined as follows.*

$$\begin{aligned}
Free(E) &= Var_f(E)\\
Free(\alpha;\pi) &= Var_f(\alpha) \cup Free(\pi)\\
Free(\rho;\pi) &= Var_f(\rho) \cup Free(\pi)\\
Free(\beta?;\pi) &= Free(\pi) \setminus safeVar(\beta)\\
Free(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi}; \pi) &= (Free(\pi_1) \setminus safeVar(\beta)) \cup\\
&\quad\ Free(\pi_2) \cup Free(\pi)\\
Free(\texttt{while } \beta \texttt{ do } \pi_1 \texttt{ od}; \pi) &= Free(\pi_1) \cup Free(\pi)
\end{aligned}$$

A safe plan now is defined as a plan without free variables.

Definition 8. *(safe plan) Let $\pi \in L_P$ be a plan. The plan π is safe if and only if $Free(\pi) = \emptyset$.*

As plans can be transformed using plan revision rules, we have to add a requirement on these rules, making sure that plan safety is preserved under plan revision. Furthermore, new plans can be adopted using plan selection rules. Therefore, we need a requirement on these rules as well, ascertaining that only safe plans are adopted. Finally, we also need a requirement on goal revision rules for the following reason. The goals in the body of goal revision rules are added to the goal base. The goal base should be ground, so we have to make sure that all variables in the goals that will be added, are substituted by a value. This is reflected in the safety requirement for goal revision rules.

Definition 9. *(safe rules)*

- *A goal revision rule* $\kappa_h \leftarrow \beta \mid \kappa_b \ \in \ GR$ *is safe, if* $Var_f(\kappa_b) \subseteq safeVar(\{\kappa_h, \beta\})$.
- *A plan selection rule* $\kappa \leftarrow \beta \mid \pi \ \in \ PS$ *is safe, if* $Free(\pi) \subseteq safeVar(\{\kappa, \beta\})$.
- *A plan revision rule* $\pi_h \leftarrow \beta \mid \pi_b \ \in \ PR$ *is safe, if* $Free(\pi_b) \subseteq safeVar(\{\pi_h, \beta\})$.

2.5 A 3APL configuration

Above, the beliefs, goals, plans, and reasoning rules of a 3APL agent were defined. To program a 3APL agent means to specify its initial beliefs, goals, and plans, and to write sets of goal revision rules, plan selection rules and plan revision rules. This is formalized in the specification of a 3APL agent.

Definition 10. *(3APL agent) A 3APL agent is a tuple* $\langle \sigma_0, \gamma_0, \Pi_0, GR, PS, PR \rangle$ *where* σ_0 *is the initial beliefbase,* γ_0 *is the initial goalbase,* Π_0 *is the initial planbase,* GR *is a set of goal revision rules,* PS *is a set of plan selection rules, and* PR *is a set of plan revision rules.*

The beliefs, goals, and plans are the elements that change during the execution of the agent while the reasoning rules remain unchanged during the execution of the agent. Together with a fourth *substitution* component, these elements constitute a 3APL configuration. This substitution part of the configuration is used to store values or bindings associated with first order variables.

Definition 11. *((ground) substitution) A substitution* θ *is a finite set of the form* $\{x_1/t_1, \ldots, x_n/t_n\}$, *where* $x_i \in Var$ *and* $t_i \in Term$ *and* $\forall i \neq j : x_i \neq x_j$. θ *is called a ground substitution if all* t_i *are ground terms.*

Definition 12. *(binding, domain, free variables) Let* $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ *be a ground substitution. Each element* x_i/t_i *is called a binding for* x_i. *The set of variables* $\{x_1, \ldots, x_n\}$ *is the domain of* θ *and will be denoted by* $dom(\theta)$. *The variables occurring in some syntactic expression* e *that are not bound by some substitution* θ, *i.e. that are not in* $dom(\theta)$, *are called the free variables of* e *and this will be denoted by* $Free_\theta(e)$.

Below, we define what it means to apply a substitution to a syntactic expression. We will need this in the sequel.

Definition 13. *(application of substitution) Let* e *be a syntactic expression and let* θ *be a ground substitution. Then* $e\theta$ *denotes the expression where all occurrences of variable* x *in* e *for which* $x/t \in \theta$ *are simultaneously replaced by* t.

Definition 14. *(configuration) A configuration of a 3APL agent is a tuple $\langle \sigma, \gamma, \Pi, \theta \rangle$, where $\sigma \subseteq BB$ is the belief base of the agent, $\gamma \subseteq GB$ is the goal base of the agent, $\Pi \subseteq L_P \times L_G$ is the plan base of the agent*[2] *and θ represents a ground substitution that binds domain variables to domain terms. Finally, the goal base in a configuration is such that for any goal $\phi \in \gamma$ it holds that $\sigma \not\models \phi$, i.e. the goal ϕ is not entailed by the agent's beliefs.*

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 condition on the 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.

3 Semantics

We define an operational semantics for 3APL in terms of a transition system ([8]). 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 queries.

Definition 15. *(semantics of belief and goal queries) Let $\langle \sigma, \gamma, \Pi, \theta \rangle$ be an agent configuration, $\delta, \delta' \in Disjunction$, $\mathbf{B}\phi, \beta, \beta' \in L_B$ and $\mathbf{G}\phi, \kappa, \kappa' \in L_G$. Let*

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

τ, τ_1, τ_2 *be ground substitutions.*

$$
\begin{array}{lll}
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\emptyset \top & & \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \mathbf{B}\phi & \Leftrightarrow & \sigma \models \phi\tau \\
& & \textit{where } Var_f(\phi) = dom(\tau) \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\emptyset \neg\mathbf{B}\phi & \Leftrightarrow & \neg\exists\tau : \langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \mathbf{B}\phi \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta \overrightarrow{\vee} \delta' & \Leftrightarrow & \langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta \textit{ or} \\
& & (\forall\tau' : \langle \sigma, \gamma, \Pi, \theta \rangle \not\models_{\tau'} \delta \textit{ and } \langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \delta') \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \beta \overrightarrow{\wedge} \beta' & \Leftrightarrow & \exists\tau_1, \tau_2 : \langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_1} \beta \textit{ and } \langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_2} \beta'\tau_1 \\
& & \textit{where } \tau_1 \cup \tau_2 = \tau
\end{array}
$$

$$
\begin{array}{lll}
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\emptyset \top & & \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \mathbf{G}\phi & \Leftrightarrow & \gamma \models \phi\tau \textit{ and } \sigma \not\models \phi\tau \\
& & \textit{where } Var_f(\phi) = dom(\tau) \\
\langle \sigma, \gamma, \Pi, \theta \rangle \models_\tau \kappa \overrightarrow{\wedge} \kappa' & \Leftrightarrow & \exists\tau_1, \tau_2 : \langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_1} \kappa \textit{ and } \langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_2} \kappa'\tau_1 \\
& & \textit{where } \tau_1 \cup \tau_2 = \tau
\end{array}
$$

Belief and goal queries can be posed in a configuration. The result of these queries is, like in logic programming, not just "succeeded" or "failed", but the query will also return a substitution τ (if it succeeds under this τ). A belief or goal query formula can thus hold in a configuration under some substitution τ. We will now explain the semantics of the belief and goal queries in more detail.

A formula of the form $\mathbf{B}\phi$ holds in a configuration with belief base σ under a substitution τ, iff ϕ with τ applied to it, follows from σ. We require that τ is such, that it binds all and nothing but the variables in ϕ. Suppose for example that $\phi = p(x, y)$ and that (only) $p(a, b)$ follows from σ. We then want our substitution to return, for instance, the binding a for x *and* the binding b for y. We furthermore do not want τ to bind any variables that do not occur in ϕ.

A formula of the form $\neg\mathbf{B}\phi$ holds in a configuration with belief base σ, iff there is no possible substitution τ such that $\mathbf{B}\phi$ follows from the configuration. If for example the formula $\neg\mathbf{B}(p(x))$ holds, it should not be possible to substitute some value a for x, such that $p(a)$ follows from σ. The evaluation of a negative "literal" will thus always yield an empty substitution.

A formula of the form $\delta \overrightarrow{\vee} \delta'$ holds in a configuration under a substitution τ, iff δ or otherwise δ' holds under τ. The idea is, that if for example a query $\mathbf{B}(p(x)) \overrightarrow{\vee} \mathbf{B}(q(y))$ is posed, the left part of the formula, i.e. $\mathbf{B}(p(x))$, is checked first. If this query $\mathbf{B}(p(x))$ succeeds, we conclude that the orginial query succeeds and we do not have to check the second part of the original query. If the first part fails however, we need to then check the second part. This definition of the semantics of $\overrightarrow{\vee}$ renders it a non-commutative operator. Take for example the formula $\mathbf{B}(p(x)) \overrightarrow{\vee} \mathbf{B}(q(y))$ and suppose that $\sigma = \{p(a), q(b)\}$. The formula $\mathbf{B}(p(x)) \overrightarrow{\vee} \mathbf{B}(q(y))$ holds in a configuration with belief base σ under $\tau = \{x/a\}$. The formula $\mathbf{B}(q(y)) \overrightarrow{\vee} \mathbf{B}(p(x))$ on the other hand, holds under $\tau = \{y/b\}$. They both fail under $\tau = \{y/b\}$

A formula of the form $\beta \overrightarrow{\wedge} \beta'$ holds in a configuration under a substitution τ, iff β holds under some substitution τ_1 and β' with τ_1 applied to it, holds under some substitution τ_2. The operator $\overrightarrow{\wedge}$ is therefore not commutative. These τ_1 and τ_2 should be such that together they form substitution τ. For example, let $\sigma = \{p(a), q(b,c)\}$ and suppose that we evaluate the following formula in a configuration with belief base σ: $\neg\mathbf{B}(p(x)) \overrightarrow{\wedge} \mathbf{B}(q(x,y))$. We first evaluate $\neg\mathbf{B}(p(x))$, which means that there should exists no substitution τ such that $\mathbf{B}(p(x))$ holds under this τ. However, $\mathbf{B}(p(x))$ holds under $\tau = \{x/a\}$ and the query thus fails. Now take the formula $\mathbf{B}(q(x,y)) \overrightarrow{\wedge} \neg\mathbf{B}(p(x))$. We first evaluate $\mathbf{B}(q(x,y))$, which holds under $\tau_1 = \{x/b, y/c\}$. We then apply τ_1 to $\neg\mathbf{B}(p(x))$, yielding $\neg\mathbf{B}(p(b))$. Now we evaluate $\neg\mathbf{B}(p(b))$, which holds if there is no substitution τ such that $\mathbf{B}(p(b))$ holds under τ. There is indeed no substitution under which this formula holds, so the query succeeds with $\tau = \{x/b, y/c\}$.

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 ([7]).

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 revision rules GR, a set of plan revision rules PR, and a set of plan selection rules PS.

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 16. *(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. Let* $Free_\theta : \wp(L_P \times L_G) \to \wp(Var)$ *be the generalization of the function* $Free_\theta$ *of definition 12 to sets of plan-goal pairs and let* $V = Free_\theta(\Pi)$*. Then, the derivation for the execution of a set of plans is specified in terms of the execution of individual plans as follows.*

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

Transitions for individual plans are parameterized by the set of free variables V, i.e. those not bound by θ, 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 the function $args : L_G \to \wp(\{\phi_1 \wedge \ldots \wedge \phi_n \mid \phi_1, \ldots, \phi_n \in L\})$ that removes the $\mathbf{G}$ modalities from a goal formula returning its goals from L, with $args(\top) = \emptyset$. For example, $args(\mathbf{G}(p(x)) \wedge \mathbf{G}(q(y))) = \{p(x), q(y)\}$. 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 revision rules (see definition 22).

Definition 17. *(basic action execution) Let $\alpha \in Act$ and let $\mathcal{T} : (Act \times BB) \rightarrow BB$ 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:*

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

where $\gamma' = \gamma \backslash \{\phi \in \gamma \mid \ \sigma' \models \phi\}$.

The substitution θ is used to instantiate free variables in the basic action α. Furthermore, by definition 23, we know that κ must be ground. We can therefore specify that κ should hold under the empty substitution, as no variables need to be bound.

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 14). 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 18. *(test execution) Let $\beta \in L_B$ and let τ be a ground substitution.*

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

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 19. *(execution of composite plans) Let τ be a ground substitution. The following transitions specify the execution of different types of composite* *plans.*

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

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

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

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

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

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 revision 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 revision rule only affects the goal base of the agent, i.e. the goal base of the agent is revised according to the goal revision rule.

Definition 20. *(goal revision rule application) Let the rule $\kappa_h \leftarrow \beta \mid \kappa_b$ be a safe goal revision rule from GR and τ_1, τ_2 be ground substitutions. Then the transition rule for this safe goal revision rule is defined as follows:*

$$\frac{\langle \sigma, \gamma, \Pi, \theta\rangle \models_{\tau_1} \kappa_h \;\&\; \langle \sigma, \gamma, \Pi, \theta\rangle \models_{\tau_2} \beta\tau_1 \;\&\; \forall\phi \in args(\kappa_b) : \sigma \not\models \phi\tau_1\tau_2}{\langle \sigma, \gamma, \Pi, \theta\rangle_V \rightarrow \langle \sigma, \gamma', \Pi, \theta\rangle}$$

where $\gamma' = (\gamma \backslash \{\phi \in \gamma \mid \phi' \in args(\kappa_h) \textit{ and } \phi \equiv \phi'\tau_1\tau_2\}) \cup \{\phi\tau_1\tau_2 \mid \phi \in args(\kappa_b)\}$.

The effect of the application of the safe goal revision rule on the goal base is that it removes goals that are syntactically equal to the goal in the head (modulo the **G** operator). Furthermore, it adds the goals in the body of the rule (for all possible substitutions τ_1 and τ_2).

Note that we first check if the head of the rule (a goal query) is derivable from the agent configuration under a substitution τ, and then we check if the guard of the rule (a belief query) to which τ is applied, is derivable from the agent configuration. Doing the checks in this order (and not first belief query and then goal query) allows more goal rules to be applicable. To illustrate this, consider the rule $\mathbf{G}(g(x)) \leftarrow \neg\mathbf{B}(p(x)) \mid \kappa$ and suppose that $\sigma = \{p(a)\}$ and $\gamma = \{g(c)\}$. The proposed order of checks allows this rule to be applied while the reverse order does not. Note that we do not require a variant of the goal revision rule since the (updating) goals are ground.

A plan revision rule $\pi_h \leftarrow \beta \mid \pi_b$ is applicable if its head π_h unifies with the prefix of an agent's plan and its condition β is derivable from the 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 revision rule only affects the plan base of the agent, i.e. the plan to which the plan revision rule is applied, is revised. We first define the concept of a most general unifier.

Definition 21. *(most general unifier) Let $\pi, \pi' \in L_P$. A unifier for the pair (π, π') is a substitution θ such that $\pi\theta \equiv \pi'\theta$, i.e. such that the two plans are syntactically equal. A unifier θ is called the most general unifier for the pair, if for each unifier θ' of the pair, there exists a substitution τ such that $\theta' = \theta\tau$.*

Definition 22. *(plan revision rule application) Let $\pi_h \leftarrow \beta \mid \pi_b$ be a variant of a safe plan revision rule from PR such that no free variables in the rule occur in V or $dom(\theta)$. Let η be a most general unifier for π and π_h and let τ be a ground substitution.*

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

The effect of the application of the safe plan revision rule on the plan base is that the plan π is replaced by the body π_b of the plan revision rule instantiated with the substitution η, which results from matching the head of the rule with the plan, and with the substitution τ, which results from matching the condition of the rule with the belief base. Note that the substitution θ is not updated by the substitutions τ or η 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 or η are about new variables that occur only in the body of the rule. τ or η can therefore be applied directly to π_b. Note also that plan revision rules revise the prefix of plans.

A safe plan selection 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. A plan selection rule only affects the plan base of the agent.

Definition 23. *(plan selection rule application) Let $\kappa \leftarrow \beta \mid \pi$ be a variant of a safe plan selection rule from PS such that no free variables in the rule (plan part of the rule) occur in V or $dom(\theta)$. Let also τ_1, τ_2 be ground substitutions.*

$$\frac{\langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_1} \kappa \ \& \ \langle \sigma, \gamma, \Pi, \theta \rangle \models_{\tau_2} \beta\tau_1}{\langle \sigma, \gamma, \Pi, \theta \rangle_V \rightarrow \langle \sigma, \gamma, \Pi \cup \{(\pi\tau_1\tau_2, \kappa\tau_1)\}, \theta \rangle}$$

Note that the goal $\kappa\tau_1$ that should be achieved by the plan $\pi\tau_1\tau_2$ 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 $\pi\tau_1\tau_2$ is not executed yet; the goals of agents may change only after execution of plans. We do not add substitutions τ_1, τ_2 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 24. *(computation run) A computation run* $\texttt{CR}(s_0)$ *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.*

Definition 25. *(semantics of a 3APL agent) The semantics of a 3APL agent* $\langle \sigma_0, \gamma_0, \Pi_0, GR, PR, PS \rangle$ *is defined iff the plans in* Π_0 *and the rules* GR, PR *and* PS *are safe. The semantics then is the set of computation runs* $\texttt{CR}(\langle \sigma_0, \gamma_0, \Pi_0, \emptyset \rangle)$.

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 [3].
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 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 revision 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 [11]. Some more moderate alternatives are also possible. E.g. create a plan for a goal (using an plan selection 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 revision 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. It seems not unreasonable to allow only one plan at the time for each goal, which coincides with the idea that we try different plans consecutively and not in parallel, because this might lead to a lot of unnecessary interventions between plans and also a waist of resources. If we allow only one current plan for each goal, the plans in the plan base will

all be for different goals.
Also 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 [3] 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

In this section we will discuss an example to illustrate the actual performance of a 3APL agent from its (formal) initial state to its final state. 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 or the floor (Fl) ($move(x, y, z)$). 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 as discussed in the previous section. Plan selection rules can thus only be applied if the relevant plan of the agent is empty. Let

$$\begin{aligned} \sigma_0 &= \{on(A, Fl) \wedge on(B, Fl) \wedge on(C, A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl)\} \\ \gamma_0 &= \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\} \\ \Pi_0 &= \emptyset. \end{aligned}$$

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

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

The plan selection rule is used to derive the $\mathit{move}(x,y,z)$ action that should be executed to fulfil a goal $\mathit{on}(x,z)$. 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 revision rules can then be used to create a configuration in which this action *can* be executed. Note that the plan selection rule is used to select an action to fulfil a goal of the form $\mathit{on}(x,z)$. The initial goal base however contains a conjunction of $\mathit{on}(x,z)$ predicates. The plan selection 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 revision rule p_1 can be applied to an action $\mathit{move}(x,y,z)$ 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 revision rules with head $\mathit{move}(x,y,z)$ 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 $\mathit{move}(x,y,z)$ action should be replaced by the plan $\mathbf{B}(\mathit{on}(u,x))?; \mathit{move}(u,x,Fl); \mathit{move}(x,y,z)$: 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 plan selection 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 $\mathit{move}(A,Fl,B)$, $\mathit{move}(B,Fl,C)$ or $\mathit{move}(C,A,Fl)$). Suppose the first substitution is chosen. After application of this plan selection rule, the plan of the agent becomes the plan in the consequent of the rule after application of τ. The goal $\mathit{on}(A,B)$ is moreover associated with the plan, resulting in the following plan base (other components of the initial configuration do not change):

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

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

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

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 $\mathit{move}(C,A,Fl)$ is executed (the substitution θ is applied to the action). The modified components of the agent's configuration are as follows:

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

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:

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

During the execution, a substitution θ_F is computed with $v \in dom(\theta_F)$. We assume variable u of plan revision 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 plan selection rule.

In this example, we did not use any goal revision rule in order to keep it simple. However, in a domain where blocks for instance have weights, a goal revision rule could be added to drop goals involving blocks which are too heavy. Suppose the belief base of an agent contains a formula $\forall x, n : weight(x,n) \wedge (n > 3) \rightarrow tooHeavy(x)$ 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 revision 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).

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

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

6 Related Work

Declarative goals were already added to a propositional version of 3APL in [17]. In this paper, we added declarative goals to the first-order version of 3APL. Moreover, plan selection rules were introduced to generate plans for declarative goals and goal rules are introduced to reason with goals.

Although the notion of declarative goals has been investigated many times in theoretic research on cognitive agents (see for example [2,9]), it has received less attention in agent programming languages. An example of a programming language in which the notion of declarative goals is incorporated is the language GOAL [7]. However, this language lacks the notion of plans, which makes this language unsuitable to serve as a programming language. It nevertheless served as the starting point for the 3APL extensions discussed in [17].

A few other programming languages have also claimed to incorporate the notion of declarative goals, which we will review below briefly. In research on AgentSpeak(L), the issue has been addressed in [1], in which a BDI logic for AgentSpeak(L) is defined. In this logic, the notion of desire (or goal, as these concepts are often identified) is defined in terms of achievement goals. These achievement goals however are part of the procedural part of an AgentSpeak(L) agent, i.e. they can occur in a sequence of actions or in a plan as we call it. Achievement goals are not a separate component of an AgentSpeak(L) agent. In this way, it is difficult to decouple plan failure (or execution) from goal failure (or achievement) (see [18]). The interesting aspect of the use of declarative goals is precisely this decoupling: all kinds of unpredictable things can happen during plan execution and the execution of the plan might not have the desired result. In 3APL, the goal will remain in the goal base in this case and a new plan can be selected to try to achieve the goal once more. This is the reason why it *is* interesting for agents to explicitly incorporate desired end-states and it *is not* for normal procedural programs. A normal procedural program does not operate in a dynamic and unpredictable environment and if programmed right, the execution of it will have the desired result.

In the context of the agent programming language Golog, declarative goals are discussed in [13]. In this paper, the issue of acting rationally in the presence of prioritized goals is discussed. A set of prioritized goals and a high-level non-deterministic Golog program are used to produce a ready-to-execute plan, whose execution will respect both the given program and the set of goals. These goals are thus used to guide the generation of an executable low-level plan from a high-level plan. In 3APL on the contrary, goals are used to *select* a high- or low-level plan through the application of plan selection rules. In case of the selection of a high-level plan, this plan is transformed into a low-level plan while it is being executed, using plan revision rules. If the plan fails to reach the goal, the goal will remain in the goal base. It will probably depend on the characteristics of the problem at hand (dynamic versus static environment etc.) which approach is more suitable.

The issue of goal revision during execution has, as far as we know, not been addressed in the languages discussed above.

7 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.

Another issue for future research could perhaps be to relate the Golog approach to declarative goals ([13]) to our approach.

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.

References

1. R. H. Bordini and A. F. Moreira. Proving the asymmetry thesis principles for a BDI agent-oriented programming language. *Electronic Notes in Theoretical Computer Science*, 70(5), 2002. http://www.elsevier.nl/gej-ng/31/29/23/125/23/29/70.5.008.pdf.
2. P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
3. 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)*, pages 97–104, Melbourne, 2003.
4. 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.
5. D. Dennet. *The intentional stance*. The MIT Press, Cambridge, 1987.
6. 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.
7. 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.
8. G. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, Computer Science Department, 1981.
9. 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.
10. 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.
11. 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.
12. 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.
13. S. Sardina and S. Shapiro. Rational action in agent programs with prioritized goals. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 417–424, Melbourne, 2003.
14. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

15. 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.
16. B. van Linder, W. van der Hoek, and J.-J. Ch. Meyer. Formalizing abilities and opportunities of agents. *Fundamenta Informaticae*, 34(1,2):53–101, 1998.
17. 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 the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 393–400, Melbourne, 2003.
18. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.
19. M. Wooldridge. *An introduction to multiagent systems.* John Wiley and Sons, LTD, West Sussex, 2002.
20. 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.