Goal Types in Agent Programming

Mehdi Dastani and **M. Birna van Riemsdijk** and **John-Jules Ch. Meyer** [1]

Abstract. This paper presents three types of declarative goals: perform goals, achieve goals, and maintain goals. The integration of these goal types in a simple but extendable logic-based agent-oriented programming language is discussed and motivated. The computational semantics for each goal type is presented by means of a transition system. It is shown that the presented semantics of the goal types ensure some desirable and expected properties.

1 Introduction

An essential characteristic of autonomous intelligent agents is their pro-active behaviour [7]. These agents are assumed to have goals for which they (pro-actively) decide actions to perform. Different logics have been proposed to characterize goals, to represent and reason about them, and to specify their relations to other agent concepts such as beliefs and actions [2, 3, 5]. These logics allow the specification of various agent types, i.e., agents that have a certain attitude towards their goals.

Motivated by these logics, various agent-oriented programming languages have been proposed [1]. In order to allow the implementation of various goal related agent types (i.e., agents that can have different attitudes towards their goals) existing programming languages provide constructs to implement various types of goals. For example, JACK provides programming constructs to implement, among others, test, achieve, insist, and maintain goals, Jason has achieve and test goals, and Jadex has achieve, query, perform and maintain goals [1]. The way in which goals are treated by these programming languages differs. In Jadex, goals are represented in XML in terms of a label/name and a number of other parameters. In Jason and JACK, goals are particular types of events. Further, neither JACK nor Jadex provides the formal semantics of their goal types. A detailed comparison between the goal types of the various languages is beyond the scope of this paper.

In this paper, we focus on three types of goals: perform goals, achieve goals, and maintain goals. These goals are represented as logical formulas having formal semantics. Perform goals can be used to generate plans without demanding that the plans *must* reach the states denoted by the goals. The attitude of an agent toward a perform goal is thus to generate relevant plans after which the goal will be dropped. For example, consider "`FC and CC`" as a perform goal of an agent where `FC` stands for having-fuel-in-car and `CC` for having-clean-car. The goal "`FC and CC`" can be used to generate, e.g., a plan to refuel at the gas station `gs1` and a plan to clean the car in the car wash `cw1`. After generating the plans, the agent drops the goal entirely regardless of the plans' effects. However, if the agent has only means to generate the plan to refuel, then it will only generate the refuel plan after which the entire goal is dropped. Dropping goals after generating relevant plans is a practical idea that has been implemented in most agent-oriented programming languages, although sometimes under different names. In JACK and Jason this type of goal is known as achieve goal. As in Jadex, we associate a *redo* flag with a perform goal. If this flag is false, then the agent will behave as described above. Otherwise, if the flag is true, the agent will not drop its perform goal but apply relevant planning rules repeatedly and indefinitely. For example, a vacuum cleaner that has to clean a number of rooms repeatedly without the ability to check if a room is clean can be modelled as having a perform goal with a true redo flag.

The idea of an achieve goal is to reach the state denoted by it. Similar to perform goals, an agent with an achieve goal will apply relevant planning rules to generate and execute plans. If the achieve goal is not reached after the execution of these plans, it applies the planning rules again, hopefully generating different plans, since the circumstances might have changed. Once the achieve goal is reached, it will stop generating and executing plans for this goal. For example, consider an agent that has `FC` as an achieve goal and that can generate two plans, i.e., to refuel at gas stations `gs1` and `gs2`. The agent can generate and execute the plan to refuel at `gs1`. If the plan is successful, the achieve goal will be dropped. Otherwise, it will generate and execute the second plan to refuel at `gs2`. If both plans do not achieve the goal, then it will apply the rules again. As suggested in [6] and implemented in Jadex, we assign a failure condition to each achieve goal to indicate when the agent should stop trying to achieve the goal and thus drop the goal. For example, `no-fuel-at-gs1-and-gs2` can be the failure condition of the achieve goal `FC`. A similar type of goal is introduced in Jadex and in JACK, though in JACK under the name *insist* goal.

Finally, the idea of maintain goal is to ensure that a state holds and continues to hold. Plans should be generated and executed if the state denoted by the maintain goal is *threatened* not to hold, rather than waiting and taking action once the state does not hold. The condition under which the maintain goal is threatened not to hold will be called the maintain condition. The agent starts to generate and execute plans when the maintain condition becomes true. For example, consider an agent with `FC` as a maintain goal. This means that the agent wants to maintain having a fueled car. The maintain condition is the illuminated lamp warning of a shortage of fuel. The agent should generate and execute a refuel plan if the lamp is illuminated. If the maintain condition continues to hold after the execution of the plan, because for example the tank station had no fuel, the agent may try to generate and execute another plan, e.g., to go to another tank station to refuel. If all plans are generated and the maintain condition still holds, then there are two options. The agent can either stop generating and executing plans since it has tried all plans, or it can continue to apply the planning rules once again, hopefully generating new plans this time. In order to allow both options, we add a *retry* flag, like in Jadex [1], to the maintain goals. If the flag is true, the agent will try to re-apply planning rules, otherwise it does not.

[1] Utrecht University, The Netherlands, Email: {mehdi,birna,jj}@cs.uu.nl

2 Syntax

In order to implement different types of goals, we propose a simple but extendable logic-based agent-oriented programming language. We assume a propositional language L and the propositional entailment relation $\models$. A perform goal consists of a propositional formula and a *redo* flag that indicates whether the goal should be performed repeatedly. An achieve goal consists of two propositional formulas. The first formula denotes the state to be achieved and the second formula represents a failure condition. Finally, a maintain goal consists of two propositional formulas and a *retry* flag. The first formula denotes the state to be maintained and the second formula represents the maintain (trigger) condition. The retry flag indicates whether to repeat performing plans as long as the maintain trigger condition holds.

Definition 1 *(belief and goal languages) The belief language L_σ, the perform goal language L_{γ_p}, the achieve goal language L_{γ_a}, and the maintain goal language L_{γ_m} are defined as follows:*
- $L_\sigma = L$
- $L_{\gamma_p} = \{(\phi, f) \mid \phi \in L \;\&\; f \in \{\top, \bot\}\}$
- $L_{\gamma_a} = \{(\phi, fc) \mid \phi, fc \in L\}$
- $L_{\gamma_m} = \{(\phi, mc, f) \mid \phi, mc \in L \;\&\; f \in \{\top, \bot\}\}$

A plan is considered as a sequence of basic actions which update the beliefs of an agent when executed. The plan language can be extended with other actions such as test and communication, which can be composed by if-then-else and while constructs [4, 1].

Definition 2 *(plan) Let* $\mathsf{BasicAction}$ *with typical element a be the set of basic actions. The set of plans* Plan *with typical element π is defined as: $\pi ::= a \mid \pi_1; \pi_2$. We use ϵ to denote the empty plan.*

Planning rules are used for selecting an appropriate plan for a goal under a certain belief condition. A planning rule is of the form $\beta, \kappa \Rightarrow \pi$ and indicates to select plan π for the goal κ, if the agent believes β. In order to be able to check whether an agent has a certain belief or goal, we use propositional formulas from L to represent belief and goal query expressions.

Definition 3 *(plan selection rule) The set of plan selection rules $\mathcal{R}_{\mathsf{PG}}$ is defined as: $\mathcal{R}_{\mathsf{PG}} = \{\beta, \kappa \Rightarrow \pi \mid \beta, \kappa \in L, \pi \in \mathsf{Plan}\}$.*
In the following, we use $\mathbf{G}(r)$ and $\mathbf{B}(r)$ to indicate the goal condition κ and the belief condition β, respectively, that occur in the head of the planning rule $r = (\beta, \kappa \Rightarrow \pi)$.

Given these languages, an agent can be implemented by programming four sets of propositional formulas (representing the agent's beliefs, perform goals, achieve goals, and maintain goals), one set of planning rules, and an ordering on the set of planning rules to indicate the order in which the planning rules should be applied.

Definition 4 *(agent program) An agent program is a tuple $(\sigma, \gamma_p, \gamma_a, \gamma_m, \mathsf{PG}, <)$ where $\sigma \subseteq L_\sigma, \gamma_p \subseteq L_{\gamma_p}, \gamma_a \subseteq L_{\gamma_a}, \gamma_m \subseteq L_{\gamma_m}$, $\mathsf{PG} \subseteq \mathcal{R}_{\mathsf{PG}}$, and $<$ is a strict order on* PG.

3 Semantics

The semantics of the programming language is defined by means of a transition system. A transition system for a programming language consists of a set of axioms and derivation rules for deriving transitions for this language. A transition is a transformation of one configuration into another and it corresponds to a single computation step. A configuration represents the state of an agent at each point during computation.

For the purpose of this paper, a configuration consists of a belief base σ representing the agent's beliefs, a goal base γ representing the agent's goals, a plan base Π containing the generated plans, a set of planning rules PG, and a set T to administrate the set of planning rules that have already been tried for application. These rules will not be applied anymore because either they have been applied already or the goals that occur in their heads are achieved. In the rest of this paper, we will call these rules *tried rules*.

Definition 5 *(agent configuration) Let $\Sigma = \{\sigma \mid \sigma \subseteq L_\sigma, \sigma \not\models \bot\}, \gamma_p \subseteq \{(\phi, f) \in L_{\gamma_p} \mid \phi \not\models \bot\}, \gamma_a \subseteq \{(\phi, fc) \in L_{\gamma_a} \mid \phi \not\models \bot\}$, and $\gamma_m \subseteq \{(\phi, mc, f) \in L_{\gamma_m} \mid \phi \not\models \bot\}$. An agent configuration is a tuple $\langle \sigma, \gamma, \Pi, \mathsf{PG}, <, T\rangle$ where $\sigma \in \Sigma$ is the belief base, $\gamma = (\gamma_p, \gamma_a, \gamma_m)$ is the goal base, $\Pi \subseteq (L \times L \times \mathsf{Plan})$ is the plan base, $\mathsf{PG} \subseteq \mathcal{R}_{\mathsf{PG}}$ is a set of planning rules, $<$ is a strict order on PG, and $T \subseteq \{\langle \phi, R\rangle \mid \phi \in L, R \subseteq \{r \mid r \in \mathsf{PG} \;\&\; \phi \models \mathbf{G}(r)\}\}$ administrates the sets of tried planning rules.*

The goal base γ in this definition is a 3-tuple consisting of three goal bases $\gamma_p, \gamma_a, \gamma_m$. These goal bases represent the agent's perform goals, achieve goals, and maintain goals, respectively. Moreover, the elements of the plan base are defined as 3-tuples consisting of a plan and two goal formulas that indicate the reasons for generating the plan. More specifically, a plan can be generated by applying a planning rule $\beta, \kappa \Rightarrow \pi$, if κ is a subgoal of an agent's goal ϕ, i.e., if $\phi \models \kappa$. The two formulas associated with a plan in a 3-tuple are the agent's goal ϕ and its subgoal κ based on which a planning rule is applied and the plan is generated. This means that we have $\forall (\phi, \kappa, \pi) \in \Pi : \phi \models \kappa$. Finally, the set T is added to the agent configuration in order to administrate the planning rules R that have been tried to be applied for each goal ϕ of the agent. This information will be used to avoid applying the same planning rules repeatedly.

The initial configuration of an agent can be built based on the agent program (definition 4) that specifies the initial beliefs, goals, planning rules, and the ordering on the rules. As noted, an agent does not have initial plans for simplicity reasons. This implies that the plan base is empty in the initial configuration. Also, because none of the planning rules have been tried to be applied before an agent starts to execute, the set of applied planning rules for each goal in the initial configuration is empty.

Definition 6 *(initial configuration) Let $(\sigma, \gamma_p, \gamma_a, \gamma_m, \mathsf{PG}, <)$ be an agent program. The initial configuration of the agent is $\langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, \mathsf{PG}, <, T\rangle$, where $\Pi = \emptyset$, and $T = \{\langle \phi, \emptyset\rangle \mid (\phi, f) \in \gamma_p$ or $(\phi, fc) \in \gamma_a$ or $(\phi, mc, f) \in \gamma_m\}$.*

In the sequel, we omit the set of planning rules PG and the ordering $<$ in the agent configuration for reasons of presentation. Moreover, in the following we use $\models_b$ which is defined as follows: $(\langle \sigma, \gamma, \Pi, T\rangle \models_b \phi) \Leftrightarrow (\sigma \models \phi)$.

When executing an agent, planning rules will be selected and applied based on its goals, beliefs, and the ordering on the planning rules. The application of planning rules generates plans which can be selected and performed. Before introducing the transition rules to specify possible agent execution steps, we need to define what it means to perform a plan. For simplicity reasons, we assume here that the performance of a plan affects only the belief base. The effect of plans on the belief base is captured through an update operator τ, which takes the belief base and a basic action and generates the updated belief base. This update operator can be as simple as adding/deleting atoms to/from the belief base.

Definition 7 *(plan performance) Let* $\mathsf{BasicAction}$ *be the set of basic actions, $\tau : \mathsf{BasicAction} \times \Sigma \to \Sigma$ be a partial function implement-*

ing a belief update operator, and $a;\pi$ be a plan consisting of action a followed by the plan π. The performance of a plan with respect to a belief base is defined by the function $Perform : \Sigma \times \mathsf{Plan} \to \Sigma$ that updates the belief base consecutively by the sequence of actions involved in the plan, i.e., $Perform(\sigma, a) = \tau(\sigma, a)$ and $Perform(\sigma, a; \pi) = Perform(Perform(\sigma, a), \pi)$.

Note that $Perform$ is a partial function since the belief update operator is a partial function: we assume that the function $Perform$ evaluates to $undefined$ if it is applied to an undefined input element. Given the function $Perform$, the following two transition rules (called EP_1 and EP_2) define plan execution.

$$\frac{(\phi, \kappa, \pi) \in \Pi \;\&\; \pi \neq \epsilon \;\&\; Perform(\sigma, \pi) = \sigma'}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma', \gamma, (\Pi \setminus \{(\phi, \kappa, \pi)\}) \cup \{(\phi, \kappa, \epsilon)\}, T\rangle}$$

$$\frac{(\phi, \kappa, \pi) \in \Pi \;\&\; \pi \neq \epsilon \;\&\; Perform(\sigma, \pi) = undefined}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma, \gamma, \Pi \setminus \{(\phi, \kappa, \pi)\}, T\rangle}$$

The first transition rule (EP_1) captures the case where the plan π is successfully performed. The resulting configuration contains the empty plan and the belief base is updated appropriately. The second transition rule (EP_2) captures the case that the performance of the plan has failed. Note that in this case, the failed plan (ϕ, ψ, π) will be removed from the plan base.

In order to define the transitions for different types of goals, we first define the notion of *relevant* and *applicable* planning rules w.r.t. an agent's goal, and an auxiliary function called *next*. Intuitively, a planning rule is relevant for an agent's goal if it can contribute to the agent's goal, i.e., if the goal that occurs in the head of the planning rule is a subgoal of the agent's goal. A planning rule is applicable to an agent's goal if it is not applied yet, if it is relevant for that goal, and if the belief condition of the rule is entailed by the agent's configuration. Finally, the next function selects the *first applicable planning rule* for an agent's goal if there exists one, evaluates to *nil* if all relevant planning rules have been tried to be applied, and evaluates to undefined otherwise.

Definition 8 *(relevant, applicable, next planning rules) Let $C = \langle \sigma, \gamma, \Pi, \mathsf{PG}, <, T\rangle$ be an agent configuration and $\langle \phi, R\rangle \in T$. For the given configuration C and goal ϕ, the set of relevant and applicable planning rules, and the next function are defined as follows:*
- $rel(\phi, C) = \{r \in \mathsf{PG} \mid \phi \models \mathbf{G}(r)\}$
- $app(\phi, C) = \{r \in rel(\phi, C) \mid r \notin R, \sigma \models \mathbf{B}(r)\}$
- $next(\phi, C)$
$= r \quad if\ r \in app(\phi, C) \;\&\; \forall r' \in app(\phi, C) : r \neq r' \to r > r'$
$= nil \quad if\ rel(\phi, C) = R$
$= undefined \quad otherwise$

In the following transition rules, we omit the reference to configuration C and assume that $rel(\pi)$, $app(\phi)$, and $next(\phi)$ are used in the context of the left-hand side configuration of the transition.

Proposition 1 *Let $\langle \sigma, \gamma, \Pi, \mathsf{PG}, <, T\rangle$ be an agent configuration where $\langle \phi, R\rangle \in T$. Then, $next(\phi) = undefined$ iff $rel(\phi) \neq \emptyset \wedge \forall r \in rel(\phi) : (\sigma \not\models \mathbf{B}(r) \wedge r \notin R)$.*

In order to compare the behaviours of an agent for different goal types, we need to define an agent's execution. An execution of an agent is the sequence of configurations that can be generated by applying transition rules. Agent executions can be generated by a cyclic procedure where in each cycle each transition rule gets the opportunity to be applied, i.e., in each cycle each transition rule is selected once and, if possible, applied. Such a cyclic procedure is in fact a round robin scheduling technique for selecting and applying transition rules. We call such a procedure for selecting and applying transition rules a *round robin procedure*.

Definition 9 *(agent execution) An execution of an agent is a finite or infinite sequence $\langle C_1, C_2, \ldots\rangle$, where $C_i \to C_{i+1}$ is a transition derived from the transition system for $i \in N$. A fair agent execution is an execution that is generated by a round robin procedure.*

3.1 Perform Goals

The idea of perform goals is to allow the generation of plans by applying planning rules. A planning rule can be applied if the goal in its head is logically entailed by one of the agent's goals. We use thus logical formulas to allow reasoning about the goals to decide which planning rule to apply. The following transition rule (called P_1) selects and applies the first (w.r.t. the ordering $<$ on the rules) applicable planning rule. This transition rule can be applied if there are no plans generated for the subgoal that occurs in the head of the planning rule and if the subgoal is not achieved yet. The latter condition is for efficiency.

$$\frac{\begin{array}{c}(\phi_p, f) \in \gamma_p \;\&\; next(\phi_p) = (\beta, \kappa \Rightarrow \pi) \;\&\\ \nexists \pi' \in \mathsf{Plan} : (\phi_p, \kappa, \pi') \in \Pi \;\&\; \sigma \not\models \kappa\end{array}}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma, \gamma, \Pi \cup \{(\phi_p, \kappa, \pi)\}, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_p, R\rangle\}) \cup \{\langle \phi_p, R \cup \{\beta, \kappa \Rightarrow \pi\}\rangle\}$.

Below is the second transition rule for a perform goal (called P_2) to administrate planning rules as being tried if the subgoal that occurs in their heads has already been achieved. As noted this is for efficiency reasons.

$$\frac{(\phi_p, f) \in \gamma_p \;\&\; r \in \mathsf{PG} \;\&\; \sigma \models \mathbf{G}(r)}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma, \gamma, \Pi, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_p, R\rangle\}) \cup \{\langle \phi_p, R \cup \{r\}\rangle\}$.

The third transition rule for a perform goal (called P_3) removes all empty plans from the plan base. This transition rule makes it possible to apply transition rule P_1 once again to generate an alternative plan for the perform goal.

$$\frac{(\phi_p, f) \in \gamma_p \;\&\; (\phi_p, \psi, \epsilon) \in \Pi}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma, \gamma, \Pi \setminus \{(\phi_p, \psi, \epsilon)\}, T\rangle}$$

The perform goal is enriched with a *redo* flag which can be true or false. If all relevant planning rules for a perform goal have been applied (i.e., if the next function evaluates to nil) and the *redo* flag is false, then the perform goal will be removed from the goal base and the set of tried planning rules for this perform goal is emptied (reset). However, when the *redo* flag is true, the perform goal remains in the goal base while the set of planning rules will be emptied. This ensures that the plans associated with the perform goal will be generated and performed again. These two cases are captured by the below two transition rules (called P_4 and P_5).

$$\frac{\begin{array}{c}(\phi_p, \bot) \in \gamma_p \;\&\; next(\phi_p) = nil \;\&\\ \nexists \phi \in L \;\nexists \pi \in \mathsf{Plan} : (\phi_p, \phi, \pi) \in \Pi\end{array}}{\langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, T\rangle \to \langle \sigma, (\gamma_p', \gamma_a, \gamma_m), \Pi, T \setminus \{\langle \phi_p, R\rangle\}\rangle}$$

where $\gamma_p' = \gamma_p \setminus \{(\phi_p, \bot)\}$.

$$\frac{\begin{array}{c}(\phi_p, \top) \in \gamma_p \;\&\; next(\phi_p) = nil \;\&\\ \nexists \phi \in L \;\nexists \pi \in \mathsf{Plan} : (\phi_p, \phi, \pi) \in \Pi\end{array}}{\langle \sigma, \gamma, \Pi, T\rangle \to \langle \sigma, \gamma, \Pi, (T \setminus \{\langle \phi_p, R\rangle\}) \cup \{\langle \phi_p, \emptyset\rangle\}\rangle}$$

The above transition rules determine how an agent processes its perform goals. The following proposition shows that an agent with a perform goal to which a false redo flag is assigned, can drop the goal if the relevant planning rules have tautologies as belief conditions. It shows that such an agent will drop the goal if its behaviour is generated by a fair execution procedure.

Proposition 2 *Let $C = \langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, T\rangle$ be an initial agent configuration, where $(\phi_p, \bot) \in \gamma_p$ and $\forall r \in \mathsf{PG} : \mathbf{B}(r) = \top$. There exists a finite execution that removes $(\phi_p, \bot)$ from the goal base. All fair executions eventually remove $(\phi_p, \bot)$ from the goal base.*

3.2 Achieve Goals

For an achieve goal plans should be generated to reach the state denoted by it. If the generated and performed plans do not achieve the desired state, then the achieve goal remains in the goal base. The first transition rule below (called A_1) is designed to apply planning rules in order to achieve the subgoals of the achieve goals. A planning rule can be applied if the goal in the head of the rule is not achieved yet, if there is no plan for the same subgoal in the plan base, and if the failure condition does not hold. The application of a planning rule will add the plan of the planning rule to the plan base and administrate it as a tried rule.

$$\frac{(\phi_a, fc) \in \gamma_a \;\&\; next(\phi_a) = (\beta, \kappa \Rightarrow \pi) \;\& \\ \nexists \pi' \in \mathsf{Plan} : (\phi_a, \kappa, \pi') \notin \Pi \;\&\; \sigma \not\models \kappa \;\&\; \sigma \not\models fc}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi \cup \{(\phi_p, \kappa, \pi)\}, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_a, R\rangle\}) \cup \{\langle \phi_a, R \cup \{(\beta, \kappa \Rightarrow \pi)\}\rangle\}$.

If the head of a planning rule is already achieved, then the rule will be considered as tried and will be administrated accordingly. This is realized by the next transition rule (called A_2).

$$\frac{(\phi_a, fc) \in \gamma_a \;\&\; r \in \mathsf{PG} \;\&\; \sigma \models \mathbf{G}(r)}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_a, R\rangle\}) \cup \{\langle \phi_a, R \cup \{(\beta, \kappa \Rightarrow \pi)\}\rangle\}$.

The next transition rule (A_3) removes all empty plans, which are generated for the achieve goals, from the plan base. Note that this transition rule is similar to P_3. We did not generalize this rule to be applicable to all goal types since the empty plans for the maintain goals should be removed differently (see next section).

$$\frac{(\phi_a, fc) \in \gamma_a \;\&\; (\phi_a, \psi, \epsilon) \in \Pi}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{(\phi_a, \psi, \epsilon)\}, T\rangle}$$

An achieve goal can be dropped under two circumstances: either when the failure condition, which is assigned to it, becomes true, or when the state it denotes is reached. The transition rule A_4 below captures these two cases of dropping the achieve goal. In both cases, besides the removal of the achieve goal, the plans associated with it will be removed and the set of tried planning rules will be set to empty.

$$\frac{(\phi_a, fc) \in \gamma_a \;\&\; (\sigma \models \phi_a \;or\; \sigma \models fc)}{\langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, T\rangle \rightarrow \langle \sigma, (\gamma_p, \gamma_a', \gamma_m), \Pi', T \setminus \{\langle \phi_a, R\rangle\}\rangle}$$

where $\gamma_a' = \gamma_a \setminus \{(\phi_a, fc)\}$ and $\Pi' = \Pi \setminus \{(\phi_a, \kappa, \pi) \mid \kappa \in L, \pi \in \mathsf{Plan}\}$.

When all planning rules that could achieve a subgoal of an achieve goal have been applied, all generated plans have been performed, but the state denoted by the achieve goal is not reached, then the set of tried rules will be set to empty in order to enable a new round of rule applications and plan performance. This is realized by the following transition rule A_5.

$$\frac{(\phi_a, fc) \in \gamma_a \;\&\; next(\phi_a) = nil \;\& \\ \sigma \not\models \phi_a \;\&\; \sigma \not\models fc \;\&\; \nexists \kappa \in L \; \nexists \pi \in \mathsf{Plan} : (\phi_a, \kappa, \pi) \in \Pi}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi, (T \setminus \{\langle \phi_a, R\rangle\}) \cup \{\langle \phi_a, \emptyset\rangle\}\rangle}$$

Given the transition rules $A_1, \ldots, A_5$ for the achieve goals, we can now show that an achieve goal will not be removed from the goal base unless the state denoted by it is reached.

Proposition 3 *Let $\langle C_1, C_2, \ldots\rangle$ be an agent execution, where $C_i = \langle \sigma^i, (\gamma_p^i, \gamma_a^i, \gamma_m^i), \Pi^i, T^i\rangle$ and $(\phi, \bot) \in \gamma_a^1$. Then,*

$$\forall k \in N : (\sigma^1 \not\models \phi \wedge \ldots \wedge \sigma^k \not\models \phi) \rightarrow (\phi, \bot) \in \gamma_a^k$$

The achievement of an achieve goal depends on many factors among which whether the planning rules are designed correctly and whether the plans of different rules interfere with each other. In order to define whether a planning rule is designed correctly, we need to specify the consequence of performing a plan in an agent configuration. The following function XP maps an agent configuration C to another agent configuration C' such that C' can be reached from configuration C by performing successfully the plan π (from the plan base of C), i.e., $C \rightarrow C'$ is a transition derived by applying the transition rule EP_1.

Definition 10 *(XP function) Let $\mathcal{C}$ be the set of agent configurations. The eXecute Plan function $XP : \mathcal{C} \times (L \times L \times \mathsf{Plan}) \rightarrow \mathcal{C}$ is defined as follows: $XP(C, (\phi, \kappa, \pi)) = C'$ iff $C \rightarrow C'$ is derivable by applying transition rule EP_1 through which (ϕ, κ, π) from the plan base of C is executed, i.e., where $C = \langle \sigma, \gamma, \Pi, T\rangle$, $C' = \langle \sigma', \gamma, \Pi', T\rangle$, $(\phi, \kappa, \pi) \in \Pi$, $\Pi' = (\Pi \setminus \{(\phi, \kappa, \pi)\}) \cup \{(\phi, \kappa, \epsilon)\}$, and $Perform(\sigma, \pi) = \sigma'$.*

A correct planning rule is a rule for which the performance of its plan in an agent configuration achieves the goal occuring in its head.

Definition 11 *(correct planning rules) Let XP be the eXecute Plan function. A planning rule $\beta, \kappa \Rightarrow \pi$ is correct iff for all configurations C and $\phi, \psi \in L$ it holds that: $XP(C, (\phi, \psi, \pi)) \models_b \kappa$*

The applications of planning rules generate a set of plans that can be performed in an arbitrary order. If plans are not designed carefully, then their order of performance can have undesirable effects. In order to guarantee that a set of plans reach certain states independent of their order of performance, we employ a notion of non-interfering plans.

Definition 12 *(non-interfering plans) Let $C = \langle \sigma, \gamma, \Pi, T\rangle$ be an agent configuration, $p_1 = (\phi_1, \psi_1, \pi_1)$, $p_2 = (\phi_2, \psi_2, \pi_2)$, and $p_1, p_2 \in \Pi$. The plans π_1 and π_2 are non-interfering iff*

$$XP(XP(C, p_1), p_2) = XP(XP(C, p_2), p_1)$$

If all planning rules are correct and they can always be applied (the belief condition of the rules are tautologies), then all fair executions reach eventually the state denoted by an achieve goal that is equivalent with the goals occurring in the head of the planning rules.

Proposition 4 *Let $C_1 = \langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, T\rangle$ be an initial configuration, $(\phi, fc) \in \gamma_a$, all planning rules in PG be correct, the plans in these rules be non-interfering, and there exists $\{(\top, \kappa_1 \Rightarrow \pi_1), \ldots, (\top, \kappa_n \Rightarrow \pi_n)\} \subseteq \mathsf{PG}$ such that $\{\kappa_1, \ldots, \kappa_n\} \models \phi$ and $\phi \models \kappa_i$ for $1 \leq i \leq n$. Then, there exists a configuration C_n in every fair agent execution $\langle C_1, C_2, \ldots\rangle$ such that $C_n \models_b \phi$.*

The transition rules for the perform and achieve goals show close similarities. In order to examine and compare an agent's behaviours with respect to different goal types, we define goal-equivalent agent executions. The idea is that the consecutive agent configurations in these agent executions are identical in all components except their goal bases.

Definition 13 *(goal-equivalent executions) Let* $e_1 = \langle C_1, C_2, \ldots\rangle$ *and* $e_2 = \langle C'_1, C'_2, \ldots\rangle$ *be agent executions, where for* $i \in N$, $C_i = \langle \sigma_i, \gamma_i, \Pi_i, T_i\rangle$ *and* $C'_i = \langle \sigma'_i, \gamma'_i, \Pi'_i, T'_i\rangle$ *are agent configurations. The executions* e_1 *and* e_2 *are goal-equivalent iff* $\sigma_i = \sigma'_i$, $\Pi_i = \Pi'_i$, *and* $T_i = T'_i$ *for* $i \in N$.

Note that the goal-equivalent relation is an equivalence relation. We can now show that under certain conditions the achieve goal of an agent can be replaced with a perform goal. In particular, an agent with an achieve goal such that the state denoted by it cannot be reached and its failure condition never becomes true (e.g., the goal to have fuel in the car `FC` while there is no fuel anywhere, with the failure condition `no-fuel-in-world` assuming that the agent cannot come to conclude this belief), can be replaced by a perform goal with a true redo flag (e.g., the goal `FC` with a flag $\top$ while there is no fuel anywhere).

Proposition 5 *Let* $C_1 = \langle \sigma, (\gamma_p, \gamma_a, \gamma_m), \Pi, T\rangle$ *be an agent configuration. If there exists no execution* $\langle C_1, C_2, \ldots\rangle$ *with* $C_i \models_b \phi$ *for some* $i \in N$, *then for every execution starting from* $\langle \sigma, (\gamma_p, \gamma_a \cup \{(\phi, \bot)\}, \gamma_m), \Pi, T\rangle$ *there exists a goal-equivalent execution starting at* $\langle \sigma, (\gamma_p \cup \{(\phi, \top)\}, \gamma_a, \gamma_m), \Pi, T\rangle$.

3.3 Maintain Goals

The state denoted by a maintain goal should hold at any moment during the agent execution. In order to maintain the denoted state, plans should be generated and performed. The question is when exactly plans should be generated and performed. A maintain goal is enriched with a triggering condition which, if it becomes true, indicates that plans should be generated and performed. This triggering condition can be considered as an alarm to act in order to ensure the maintenance of the state denoted by the maintain goal. It should be noted that there might be no logical relation between the maintain goals and their triggering conditions. This relation depends usually on the application domain. However, we assume that in all agent configurations, if the triggering condition does not hold, then the maintain goal holds: $\forall \sigma \in \Sigma, \forall \phi_m, mc \in L : (\sigma \not\models mc) \rightarrow (\sigma \models \phi_m)$. We associate a *retry* flag to maintain goals. If this flag is set to false, all planning rules have been applied, the generated plans have been performed, and the maintain triggering condition still holds, then no new rounds of actions should be taken. This flag can thus be used to avoid new attempts to maintain a goal.

The first transition rule M_1 is designed to allow the application of planning rules when the maintain triggering condition becomes true. The applied rule should be such that there is no plan already in the plan base for the subgoal that occurs in the head of the selected rule.

$$\frac{\begin{array}{c}(\phi_m, mc, f) \in \gamma_m \;\&\; next(\phi_m) = (\beta, \kappa \Rightarrow \pi) \;\& \\ \nexists \pi' \in \mathsf{Plan} : (\phi_m, \kappa, \pi') \in \Pi \;\&\; \sigma \models mc\end{array}}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi \cup \{(\phi_m, \kappa, \pi)\}, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_m, R\rangle\}) \cup \{\langle \phi_m, R \cup \{(\beta, \kappa \Rightarrow \pi)\}\rangle\}$.

In order to maintain a goal, applicable planning rules should be applied independently of whether the goal in their heads, which are subgoals of the maintain goals, are achieved. But this means that all planning rules that have the same subgoal in their heads will be applied if they are applicable. This can, however, make the executions inefficient. For example, suppose that the maintain goal of an agent is to have fuel in the car `FC` and that the agent has several planning rules that indicate several plans to refuel. It is undesirable to allow the agent to apply all planning rules to generates all possible ways to refuel. Therefore, we administrate all planning rules with the same goal in their heads as tried rules when one plan for this subgoal is successfully performed. This idea is captured by the transition rule M_2.

$$\frac{(\phi_m, mc, f) \in \gamma_m \;\&\; \{(\phi_m, \kappa, \epsilon)\} \in \Pi}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{(\phi_m, \kappa, \epsilon)\}, T'\rangle}$$

where $T' = (T \setminus \{\langle \phi_m, R\rangle\}) \cup \{\langle \phi_m, R \cup \{r \in \mathsf{PG} \mid \mathbf{G}(r) = \kappa\}\rangle\}$.

If there are no plans to perform, no rules to apply, and either the goal is maintained (maintain condition does not hold) or the retry flag is true, the administration of applied rules is reset. The reset makes it possible to re-apply rules and re-perform plans if needed. Note that if the maintain condition holds (actions should be taken to maintain the goal) and the retry flag is false, then the goal is not maintainable and the agent cannot do anything to maintain the goal.

$$\frac{\begin{array}{c}(\phi_m, mc, f) \in \gamma_m \;\&\; next(\phi_m) = nil \;\& \\ (f = \top \textit{ or } \sigma \not\models mc) \;\&\; \nexists \kappa \in L \; \nexists \pi \in \mathsf{Plan} : (\phi_a, \kappa, \pi) \in \Pi\end{array}}{\langle \sigma, \gamma, \Pi, T\rangle \rightarrow \langle \sigma, \gamma, \Pi, (T \setminus \{\langle \phi_m, R\rangle\}) \cup \{\langle \phi_m, \emptyset\rangle\}\rangle}$$

Given these transition rules for the maintain goal, the perform goal $(\phi, \bot)$ and the maintain goal $(\phi, \top, \bot)$ are identical.

Proposition 6 *Let* σ *be a belief base, for all* $r \in \mathsf{PG} : \sigma \not\models \mathbf{G}(r)$, *all planning rules are correct, and plans are non-interfering. Then, for every execution starting from* $C_1 = \langle \sigma, (\gamma_p, \gamma_a, \gamma_m \cup \{(\phi, \top, \bot)\}), \Pi, T\rangle$ *there exists a goal-equivalent execution starting from* $C'_1 = \langle \sigma, (\gamma_p \cup \{(\phi, \bot)\}, \gamma_a, \gamma_m), \Pi, T\rangle$.

4 Conclusion

In this paper, we have introduced three types of declarative goals for which we argued that they should be integrated in logic-based agent-oriented programming languages. We presented a simple agent-oriented programming language that allows the implementation of these goal types. The formal semantics of the programming language, and thus of the goal types, is given and it is shown that this semantics has desirable properties (because of the space limit we could not present the proofs). The presented programming language is simplified for the purpose of this paper. The language can be extended by using a (computational) subset of a first-order predicate language instead of using a propositional language. In this way, declarative goals become more expressive. Also, the plan language can be extended to allow more complex plans. We have started implementing an interpreter for an extended version of the presented programming language which we hope will be available soon. For this implementation, we have decided to generate a specific fair execution of agents by means of a deliberation cycle. In future work, we will investigate possible deliberation cycles and use temporal logics to prove properties of different deliberation cycles and to compare them.

REFERENCES

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

[2] C. Boutilier, 'Toward a logic for qualitative decision theory', in *Proceedings of the KR'94*, pp. 75–86, (1994).

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

[4] M. Dastani and L. van der Torre, 'Programming BOID-Plan agents: deliberating about conflicts among defeasible mental attitudes and plans', in *Proc. of AAMAS'04*, pp. 706–713, New York, USA, (2004).

[5] Anand S. Rao and Michael P. Georgeff, 'Modeling rational agents within a BDI-architecture', in *Proc. of KR'91*, (1991).

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

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