The agent programming language Meta-APL

Thu Trang Doan, Natasha Alechina, and Brian Logan

University of Nottingham, Nottingham NG8 1BB, UK
{ttd,nza,bsl}@cs.nott.ac.uk

Abstract. We describe a novel agent programming language, Meta-APL, and give its operational semantics. Meta-APL allows both agent programs and their associated deliberation strategy to be encoded in the same programming language. We define a notion of equivalence between programs written in different agent programming languages based on the notion of weak bisimulation equivalence. We show how to simulate (up to this notion of equivalence) programs written in other agent programming languages by programs of Meta-APL. This involves translating both the agent program and the deliberation strategy under which it is executed into Meta-APL.

1 Introduction

In this paper we sketch the agent programming language Meta-APL. Its distinguishing feature is that the agent's deliberation strategy can be encoded as part of the agent program. Meta-APL is designed to form part of a platform for verifying multi-agent systems where the agents are implemented in different (BDI-based) agent programming languages. As part of the verification process, the agent programs will be translated into Meta-APL and verification tools designed for Meta-APL will be used to verify the system. Similar approaches have been proposed before, see for example [5]. The distinguishing feature of our approach is that Meta-APL is itself an agent programming language (rather than a special purpose library as in [5]) and that the deliberation strategy of the target agent can be expressed in the Meta-APL program along with the agent program itself. It is clearly necessary to encode the deliberation strategy correctly, as executing the same program under different strategies may give very different results. Other agent programming languages have also been used to program agent deliberation, see for example [2]. However Meta-APL is specifically designed with this capability in mind, for example it is possible to write rules in Meta-APL which match against the contents of the agent's plan base.

This paper describes the first step towards this long term goal, concentrating on the design and the operational semantics of Meta-APL itself. We also define a notion of equivalence or bisimulation between programs written in different agent programming languages. We sketch how programs written in other agent programming languages plus their deliberation strategies can be translated in Meta-APL so that the resulting Meta-APL program is equivalent to the original program together with its deliberation strategy.

The remainder of this paper is organised as follows. In section 2 we introduce the syntax of Meta-APL. In section 3 we define its operational semantics. In section 4 we define the notion of equivalence between programs and show how to simulate programs written in 3APL [3]. We conclude and outline directions for future work in section 5.

2 Syntax of Meta-APL

The language of beliefs and goals in Meta-APL is similar to that of other BDI-based agent programming languages, for example, 3APL [3], but we assume a propositional language for ease of presentation (to avoid extra notation to do with substitutions etc. The actual implementation has Prolog-like syntax for beliefs).

We assume that beliefs and goals are built using propositional atoms from a finite set *Prop*. Beliefs are either atoms p, or Horn clauses $p \texttt{:-} q_1, \ldots, q_n$. A belief base is a finite set of beliefs. A belief query ϕ is defined as $\phi ::= p \mid \texttt{not}\ p \mid \phi_1\ \texttt{and}\ \phi_2 \mid \phi_1\ \texttt{or}\ \phi_2$, where `not` is negation as failure. Goals are atoms or conjunctions of atoms. A goal base is a finite set of goals.

Before we introduce plans, we define the notion of a plan body. A plan body is a finite sequence of basic actions, test actions, meta-actions and sub-goals, where

- A basic action has the form of $\#a$ where the symbol $\#$ is used to indicate that this is a basic action, and a is the name of the basic action.
- A test action has the form of $?\varphi$ where ? is used to indicate that this is a test action, and φ is a belief query.
- A sub-goal has the form of $!g$ where ! is used to indicate that this is a sub-goal, and g is a goal.
- Meta-actions are actions that allow the agent to add and delete beliefs and goals, and to delete and execute applicable plans. The set of meta-actions is as follows:
  - `add-bel`(d) is for adding a belief d into a belief base.
  - `del-bel`(d) is for deleting a belief d from a belief base.
  - `add-goal`(d) is for adding a goal d into a goal base.
  - `del-goal`(d) is for deleting a goal d from a goal base.
  - `del-plan`(i) is for deleting an applicable plan i from a plan base.
  - `exec`(i) is for executing an applicable plan i.
  - `step`(i) is for executing a single step of an applicable plan i.

A plan in Meta-APL represents the triggering conditions and execution state of a plan body. A plan is a tuple of the form (g, b, π, x, π') where:

- g is a goal,
- b is a context query (defined below),
- π is an initial plan-body,
- x is a flag for specifying the state of the plan which can have one of the following values: a (to say that it is active), ex (to say that it is executed), na (to say that it is not active).
- π' is a partially executed plan-body.

A plan base is a finite set of plans.

Context queries are evaluated against the agent's belief and plan bases. In order to define context queries, we first need to introduce the notion of plan-body terms, goal terms and flag terms. Informally, a plan-body term is a plan-body except that variables may occur where plan-bodies normally are. Given a set *Vars* of variables, the syntax of plan-body terms is as follows:

$$t_\pi ::= X \mid !s \mid ?\varphi \mid \#a \mid \texttt{ma}(d) \mid \texttt{mb}(t_\pi) \mid t_\pi; t'_\pi$$

where $X \in$ *Vars*, $!s$ is a sub-goal, $?\varphi$ is a test action, $\#a$ is a basic action, $\texttt{ma}(d)$ is a meta-action with $\texttt{ma} \in \{\texttt{add-bel}, \texttt{del-bel}, \texttt{add-goal}, \texttt{del-goal}\}$, d is a belief, and $\texttt{mb}(t_\pi)$ is also a meta-action with $\texttt{mb} \in \{\texttt{del-plan}, \texttt{exec}, \texttt{step}\}$.

A goal term is either a variable or a goal. A flag term is either a variable or a flag.

Then, context queries are defined by the following syntax:

$$b ::= X \mid t_1 = t_2 \mid t_1 \neq t_2 \mid \varphi \mid p(t_g, b, t_\pi, t_x, t_{\pi'}) \mid \neg p(t_g, b, t_\pi, t_x, t_{\pi'}) \mid b \& b'$$

where $X \in$ *Vars*, t_1, t_2, t_π and $t_{\pi'}$ are plan-body terms, φ is a belief query, t_g is a goal term, and t_x is a flag term. Evaluation of context queries will be defined in the next section when we define the operational semantics of Meta-APL. Informally, a belief query is evaluated against the belief base in a standard way, $t_1 = t_2$ is used to check if two terms t_1 and t_2 are unifiable, $p(t_g, b', t_\pi, t_s, t_{\pi'})$ means that there is a plan in the plan base that can unify with $(t_g, b', t_\pi, t_s, t_{\pi'})$; and $\neg p(t_g, b', t_\pi, t_s, t_{\pi'})$ means there is no applicable plan in the plan base which can unify with $(t_g, b', t_\pi, t_s, t_{\pi'})$.

A plan (g, b, π, x, π') is effectively identified by the first three components g, b and π. It can not happen that two applicable plans have the same first three components in a plan base at the same time. During the existence of the plan, its components g, b, π stay unchanged.

In Meta-APL, plans are generated by means of rules. A rule has the form:

$$g, b \rightarrow \pi.$$

where g is a goal and is optional, b is a context query where variables are not allowed to occur outside the scope of the atom $p(\ldots)$, and π is a plan-body. When a rule has no goal, we define that its "hidden" goal is $\top$.

3 Operational semantics of Meta-APL

An agent program consists of an initial belief base, an initial goal base and a set of rules. The initial plan base is empty.

A configuration is a tuple of the form $\langle \sigma, \gamma, \Pi, D \rangle$ where σ is a belief base, γ is a goal base, Π is a plan base and D is a phase indicator of the deliberation cycle which can have one of the following values to indicate in which phase the configuration is: UpdatePercept, ApplyRule, and Exec.

Informally, an agent runs by repeatedly performing a deliberation cycle. In the deliberation cycle, there are three main phases: updating percepts (UpdatePercept), matching and applying rules (ApplyRule), and executing executable intentions (Exec). At the beginning of a deliberation cycle, the agent first updates its percepts where the belief base of the agent is updated according to the percepts collected from the environment.

In this phase, the agent also updates its goal base by adding new goals which are received from outside and dropping goals which become achieved after the belief base is updated. In the next phase, the agent looks for applicable rules from the set of rules against the belief base, the goal base and the plan base. Then, an arbitrary applicable rule is applied to add a new applicable plan into the plan base. Notice that this newly added applicable plan may enable or disable the applicability of other rules. The agent then repeats looking for applicable rules and applying them, one by one, until no more are found. This is when the agent switches to the next phase where it executes executable intentions. In this phase, an executable applicable plan is selected and executed. After a plan is executed, the flag of the plan is set to be "ex". The phase is continued until all executable plans are marked with "ex". Then, the phase is changed to Update-Percept for starting a new deliberation cycle and all "ex" applicable plans are changed to "a".

In the following, we discuss each phase of the deliberation cycle in more detail and define the operational semantics by describing transition rules which transform one configuration to another.

First we give a definition of evaluation of beliefs, goals and belief queries. For a belief base σ and a belief or goal d, we say that $\sigma \models_{Pr} d$ iff d is propositionally entailed by σ. For a goal base γ and a goal d, we say that $\gamma \models_g d$ iff d propositionally follows from one of the goals in γ. Finally, for belief query ϕ and a belief base σ, we define $\sigma \models_{naf} \phi$ as follows:

$\sigma \models_{naf} p$ iff $\sigma \models_{Pr} p$.
$\sigma \models_{naf} \neg p$ iff $\sigma \not\models_{Pr} p$.
$\sigma \models_{naf} \phi_1$ `and` ϕ_2 iff $\sigma \models_{naf} \phi_1$ and $\sigma \models_{naf} \phi_2$.
$\sigma \models_{naf} \phi_1$ `or` ϕ_2 iff $\sigma \models_{naf} \phi_1$ or $\sigma \models_{naf} \phi_2$.

Next we define how to evaluate a context query, where variables can occur only within the scope of the literal $p(\ldots)$, against a configuration $\langle \sigma, \gamma, \Pi, D \rangle$. We write $t = g \mid \theta$ to say that two terms t and g are unifiable by the most general unifier (mgu) θ. When two terms t and g fail to unify, we write $t \neq g$. We evaluate a context query against a configuration $\langle \sigma, \gamma, \Pi, D \rangle$ inductively as follows:

- $\langle \sigma, \gamma, \Pi, D \rangle \models \varphi \mid \emptyset$ iff $\sigma \models_{naf} \varphi$ where $\emptyset$ is used to denote an empty substitution
- $\langle \sigma, \gamma, \Pi, D \rangle \models t_1 = t_2 \mid \theta$ iff the two terms t_1 and t_2 are unifiable by the mgu θ, that is $t_1 = t_2 \mid \theta$.
- $\langle \sigma, \gamma, \Pi, D \rangle \models t_1 \neq t_2 \mid \emptyset$ iff the two terms t_1 and t_2 are not unifiable.
- $\langle \sigma, \gamma, \Pi, D \rangle \models p(t_g, b', t_\pi, t_s, t_{\pi'}) \mid \theta$ iff there exists an applicable plan $i \in \Pi$ such that $p(t_g, b', t_\pi, t_s, t_{\pi'}) = i \mid \theta$.
- $\langle \sigma, \gamma, \Pi, D \rangle \models \neg p(t_g, b', t_\pi, t_s, t_{\pi'}) \mid \emptyset$ iff for all applicable plans $i \in \Pi$, we have that $p(t_g, b', t_\pi, t_s, t_{\pi'}) \neq i$.
- $\langle \sigma, \gamma, \Pi, D \rangle \models b_1 \& b_2 \mid \theta$ iff $\langle \sigma, \gamma, \Pi, D \rangle \models b_1 \mid \theta$ and $\langle \sigma, \gamma, \Pi, D \rangle \models b_2 \mid \theta$

Given a plan base Π, in order to determine which plans can be executed, we define the relation ">" over plans in Π as follows. Given

$$i_1 = (g_1, b_1, \pi_1, s_1, \pi'_1)$$
$$i_2 = (g_2, b_2, \pi_2, s_2, \pi'_2)$$

we say that $i_1 > i_2$ iff $p(g_2, b_2, \pi_2, \_, \_)$ occurs in b_1 (is a subformula of b_1). We use the notation underscore _ as in Prolog for representing any value. This means i_1 is created because of the existence of i_2 and we shall call i_1 to be the meta plan (of i_2). In each deliberation cycle of the agent, we only execute the maximal meta applicable plan, which could indirectly lead to the execution of the lower meta plans through the help of the meta-actions for executing intentions.

Then, we define the set of active plans (those with the flag to be a), the set of roots (the most meta applicable plan), the set of leafs (the least meta applicable plans or the object applicable plans), and the set of executable applicable plans (only roots which are not leafs are allowed to execute), respectively, with respect to a plan base Π as follows:

$$\begin{aligned} active(\Pi) &= \{(g, b, \pi, a, \pi') \in \Pi\} \\ root(\Pi) &= \{i \in active(\Pi) \mid \nexists i' \in active(\Pi) : i' > i\} \\ leaf(\Pi) &= \{i \in active(\Pi) \mid \nexists i' \in active(\Pi) : i > i'\} \\ executable(\Pi) &= root(\Pi) \setminus leaf(\Pi) \end{aligned}$$

Before defining transition rules for the operational semantics of Meta-APL, let us model an environment by two functions `env_percept` and `env_perform`. The effect of each function is as follows:

- `env_percept`(σ, γ, e) takes a belief base σ, a goal base γ, and the environment e as arguments. This function returns a pair of an updated belief base and an updated goal base. The current implementation assumes that percepts are atomic formulas, and the updated belief base is obtained by adding new beliefs and removing incorrect beliefs according to the percepts from the environment. Likewise, the updated goal base is obtained by adding new goals and removing achieved goals, also, according to the percepts from the environment.
- `env_action`(α, e) takes a basic action and the environment as argument. This function performs the action on the environment. For the moment, we assume that this function returns the value *true* or *false* where it only returns *true* iff the basic action is supported by the environment.

At the beginning of a deliberation cycle, an agent always updates its belief base and goal base. The transition rule for the phase of updating percepts is as follows.

$$\frac{\texttt{env\_percept}(\sigma, \gamma) = (\sigma', \gamma')}{\langle \sigma, \gamma, \Pi, \textsf{UpdatePercept} \rangle \rightarrow \langle \sigma', \gamma', \Pi, \textsf{ApplyRule} \rangle} \tag{1}$$

The transition above expresses the phase UpdatePercept where the belief base and goal base are updated with the percepts. Apart from the belief base and the goal base being updated, the phase indicator also changes from UpdatePercept to ApplyRule so that the agent can start the next phase.

We say that a rule $g, b \rightarrow \pi$ is applicable with respect to a configuration $\langle \sigma, \gamma, \Pi, D \rangle$ by a substitution θ iff the following conditions hold:

- $\gamma \models_g g$,

– $\langle \sigma, \gamma, \mathit{\Pi}, D \rangle \models b \mid \theta$,
– There is no applicable plan $(g, b\theta, \pi\theta, X, \pi') \in \mathit{\Pi}$ where $\pi' \neq \epsilon$.

Let $\mathit{Applicable}(\langle \sigma, \gamma, \mathit{\Pi}, D \rangle)$ be the set of pairs of (ρ, θ) where ρ is an applicable rule with respect to $\langle \sigma, \gamma, \mathit{\Pi}, D \rangle$ and θ is the corresponding substitution. Moreover, the last condition is for avoiding the case when a rule may be fired more than once to produce the same applicable plan. The transitions rules for the phase ApplyRule are as follows:

$$\frac{\exists((g, b \to \pi), \theta) \in \mathit{Applicable}(\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{ApplyRule} \rangle)}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{ApplyRule} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \cup \{(g, b\theta, \pi\theta, a, \pi\theta)\}, \mathsf{ApplyRule} \rangle} \quad (2)$$

The phase will change to the next one when there are no more applicable rules.

$$\frac{\mathit{Applicable}(\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{ApplyRule} \rangle) = \emptyset}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{ApplyRule} \rangle \to \langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle} \quad (3)$$

Notice that programmers have the responsibility to make sure that the loop of applying applicable rules terminates. A neglectful design of rules can easily cause the phase ApplyRule to run forever, for example if one of the rules is $p(G, B, \mathit{\Pi}, a, X) \to$ $\mathtt{exec}(G, B, \mathit{\Pi}, a, X)$.

In the phase of executing applicable plans, we execute the first step of every executable plan (those which are active, root and not leaf). Let us define transition rules corresponding to each type of the first step as follows.

The following rule is for executing a basic action.

$$\frac{i = (g, b, \pi, a, \#\alpha; \pi') \in \mathit{executable}(\mathit{\Pi}) \text{ and } \mathtt{env\_action}(\alpha) = \mathit{true}}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \quad (4)$$

When the basic action is not allowed (or supported) by the environment, the applicable plan is put into the inactive state as follows:

$$\frac{i = (g, b, \pi, a, \#\alpha; \pi') \in \mathit{executable}(\mathit{\Pi}) \text{ and } \mathtt{env\_action}(\alpha) = \mathit{false}}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, na, \#\alpha; \pi')\}, \mathsf{Exec} \rangle} \quad (5)$$

The test action simply checks if the belief query is true against the belief base.

$$\frac{i = (g, b, \pi, a, ?\varphi; \pi') \in \mathit{executable}(\mathit{\Pi}) \text{ and } \sigma \models_{naf} \varphi}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \quad (6)$$

When it is not, the test action is not removed from the applicable plan.

$$\frac{i = (g, b, \pi, a, ?\varphi; \pi') \in \mathit{executable}(\mathit{\Pi}) \text{ and } \sigma \not\models_{naf} \varphi}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, na, ?\varphi; \pi')\}, \mathsf{Exec} \rangle} \quad (7)$$

We execute a sub-goal by simply leaving it there and the programmer needs to define a suitable rule to process the subgoal.

$$\frac{i = (g, b, \pi, a, !h; \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \to \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, !h; \pi')\}, \mathsf{Exec} \rangle} \quad (8)$$

The following rule is for the case of the meta-action for deleting beliefs:

$$\frac{i = (g, b, \pi, a, \texttt{del-belief}(d); \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma \setminus \{d\}, \gamma, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{9}$$

The effect of the above rule is to remove the belief d from the belief base. Similarly, we have the following rules for the case for adding a new belief.

$$\frac{i = (g, b, \pi, a, \texttt{add-belief}(d); \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma \cup \{d\}, \gamma', \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{10}$$

where $\gamma' = \gamma \setminus \{g \in \gamma \mid \sigma \cup \{d\} \models g\}$. Besides the effect of adding new beliefs, we also remove achieved goals from the goal base. The following rule is for deleting a goal from the goal base:

$$\frac{i = (g, b, \pi, a, \texttt{del-goal}(d); \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma \setminus \{d\}, \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{11}$$

Similarly, we have the following rule for adding a new goal into the goal base:

$$\frac{i = (g, b, \pi, a, \texttt{add-goal}(d); \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma', \mathit{\Pi} \setminus \{i\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{12}$$

where $\gamma' = \gamma \cup \{d\}$ iff $\sigma \not\models d$; otherwise, $\gamma' = \gamma$. This means d is added into the goal base only when it is not an achieved goal.

The next transition rule is for deleting an applicable plan.

$$\frac{i = (g, b, \pi, a, \texttt{del-plan}(i'); \pi') \in \mathit{executable}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma, \mathit{\Pi} \setminus \{i, i'\} \cup \{(g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{13}$$

Then, we define the transition rules for the meta-actions for executing applicable plans. In principle, the "exec" meta-action makes similar effect as executing the first step of an intention.

For convenience, we also define a different transition rule, denoted as $\rightarrow^i$ for executing an active applicable plan i which is not required to be a root, but has to be active (i.e. the flag is a). The definitions are the repetition of those above for the execution phase where we replace the condition $i \in \mathit{executable}(\mathit{\Pi})$ by $i \in \mathit{active}(\mathit{\Pi})$. Then for every transition rule of the form $\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma', \gamma', \mathit{\Pi} \setminus \{i\} \cup \{i'\}, \mathsf{Exec} \rangle$, we also define:

$$\frac{i \in \mathit{active}(\mathit{\Pi})}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow^i \langle \sigma', \gamma', \mathit{\Pi} \setminus \{i\} \cup \{i'\}, \mathsf{Exec} \rangle} \tag{14}$$

Notice that the transition rules $\rightarrow^i$ are not the operational semantics of Meta-APL but we use them as auxiliary transition rules for defining the operational semantics of the meta actions `exec` and `step` of Meta-APL. We shall define the transition rule for the meta-action $\mathit{exec}(i')$ based on $\rightarrow^{i'}$ as follows:

$$\frac{\begin{array}{c} i = (g, b, \pi, a, \texttt{exec}(i'); \pi') \in \mathit{executable}(\mathit{\Pi}) \text{ and} \\ \langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow^{i'} \langle \sigma, \gamma', \mathit{\Pi} \setminus \{i'\} \cup \{i''\}, \mathsf{Exec} \rangle \end{array}}{\langle \sigma, \gamma, \mathit{\Pi}, \mathsf{Exec} \rangle \rightarrow \langle \sigma', \gamma, \mathit{\Pi} \setminus \{i, i'\} \cup \{i'', (g, b, \pi, ex, \pi'')\}, \mathsf{Exec} \rangle} \tag{15}$$

where $\pi'' = \texttt{exec}(i''); \pi'$ if i'' is not an empty plan; otherwise $\pi'' = \pi'$. The above rule means that if we have an executable applicable plan which starts with $\texttt{exec}(i')$, and i' can be executed by means of $\rightarrow^{i'}$ to become i'', then $\texttt{exec}(i')$ means to execute i' and to change to $\texttt{exec}(i'')$. The semantics $\texttt{step}(i')$ is similar to $\texttt{exec}(i')$ except that the action $\texttt{step}(i')$ does not execute the remainder i'' of i'. The transition rule for this meta-action is as follows:

$$\frac{\begin{array}{c} i = (g, b, \pi, a, \texttt{step}(i'); \pi') \in \mathit{executable}(\Pi) \text{ and} \\ \langle \sigma, \gamma, \Pi, \mathsf{Exec} \rangle \rightarrow^{i'} \langle \sigma', \gamma', \Pi \setminus \{i\} \cup \{i''\}, \mathsf{Exec} \rangle \end{array}}{\langle \sigma, \gamma, \Pi, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{i, i'\} \cup \{i'', (g, b, \pi, ex, \pi')\}, \mathsf{Exec} \rangle} \tag{16}$$

In both cases of the transition rules for exec and step, if $i' \notin \Pi$ or is inactive, then i also becomes an inactive plan.

$$\frac{i = (g, b, \pi, a, \texttt{exec}(i'); \pi') \in \mathit{executable}(\Pi) \text{ and } i' \notin \mathit{active}(\Pi)}{\langle \sigma, \gamma, \Pi, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{i\} \cup \{(g, b, \pi, na, \texttt{exec}(i'); \pi')\}, \mathsf{Exec} \rangle} \tag{17}$$

We also have:

$$\frac{i = (g, b, \pi, a, \texttt{step}(i'); \pi') \in \mathit{executable}(\Pi) \text{ and } i' \notin \mathit{active}(\Pi)}{\langle \sigma, \gamma, \Pi, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma, \Pi \setminus \{i\} \cup \{(g, b, \pi, na, \texttt{step}(i'); \pi')\}, \mathsf{Exec} \rangle} \tag{18}$$

Notice that the new definition of the transition rules $\rightarrow$ for the meta-actions `exec` and `step` also implicitly gives extra definitions of the transition rule $\rightarrow^{i}$. This helps us to define further transition rules for nested meta-actions `exec` and `step`.

Then, when there is no more executable plans, the phase turns back to UpdatePercept for a new deliberation cycle. All plans have the flag "ex" are also changed to "a" for further execution in the next deliberation cycle. We also have:

$$\frac{\mathit{executable}(\Pi) = \emptyset}{\langle \sigma, \gamma, \Pi, \mathsf{Exec} \rangle \rightarrow \langle \sigma, \gamma, \Pi', \mathsf{UpdatePercept} \rangle} \tag{19}$$

where $\Pi' = (\Pi \setminus \{(g, b, \pi, ex, \pi') \in \Pi\}) \cup \{(g, b, \pi, a, \pi') \mid (g, b, \pi, ex, \pi') \in \Pi \wedge \pi' \neq \epsilon\} \cup \{(g, b, \pi, na, \epsilon) \mid (g, b, \pi, ex, \epsilon) \in \Pi\}$.

In the above transition rule, when transiting from the phase Exec back to Update-Percept in the deliberation cycle, all applicable plans in the plan base with the flag being ex are changed back to a so that they are ready for further execution, except those have an empty plan (denoted as ϵ) which are changed to the state inactive.

4 Simulating 3APL

In this section we show how to translate agent programs of 3APL into Meta-APL. Both languages share similar features, but have different deliberation cycles. We show how meta rules can be used to simulate deliberation cycle of other languages in Meta-APL.

First we define what we mean by simulating one program by another program. We use the concept of *weak bisimulation* [6]. We treat transitions other that basic actions as *internal* or τ actions. By a *run of a program* we will mean a sequence

$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ where s_i are agent's configurations and a_i are transitions of the agent's operational semantics which are either basic actions or other internal τ transitions. Two runs $r = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots$ and $r' = s'_0 \xrightarrow{a'_0} s'_1 \xrightarrow{a'_1} s'_2 \ldots$ are *equivalent* if there is a symmetric relation R (later referred to as equivalence) between the configurations in r_1 and r_2 such that:

– $R(s_0, s'_0)$
– if $R(s, s')$ then the agent's beliefs about environment in s and s' are the same
– if $R(s, s')$ and $s \xrightarrow{\tau^*} t_1 \xrightarrow{a} t_2$ in r, then $s' \xrightarrow{\tau^*} t'_1 \xrightarrow{a} t'_2$ in r' and $R(t_2, t'_2)$, where τ^* is a sequence of 0 or finitely many internal transitions, and a is the first basic action occurring after s

Intuitively, $R(s, s')$ means that in configurations s and s' the agent has the same beliefs and goals.

We say that a program p_1 simulates another program p_2 iff:

– For every run r_1 of p_1, there is a run r_2 of p_2 such that r_1 and r_2 are equivalent.
– For every run r_2 of p_2, there is a run r_1 of p_1 such that r_2 and r_1 are equivalent.

In this paper we show how to translate a program p_2 of some agent programming language into a program p_1 of Meta-APL so that p_1 simulates p_2.

4.1 3APL

In this paper, we refer to 3APL as the agent programming language which was presented in [3]. Since the version of Meta-APL introduced in this paper for simplicity only allows propositional beliefs and goals, we show how to simulate propositional 3APL programs, but the extension to full 3APL beliefs and goals is straightforward. Moreover, we also slightly modify 3APL in order to omit the plan constructs `if-then-else` and `while-do`.

An agent in 3APL can have three types of rules:

– PG (plan generation) rule: $g \leftarrow b \mid \pi$
– GR (goal revision) rule: $g \leftarrow b \mid g'$
– PR (plan revision) rule: $\pi \leftarrow b \mid \pi'$

Where g and g' are goals, b is a belief query, π and π' are plans. A plan π is a sequence of basic actions, test actions and abstract plans. Moreover, test actions are of the form $B(\varphi)$ only. Branching and looping constructs (if-then-else and while-do) in a plan are not allowed as they can be translated into rules in 3APL using abstract plans. Note that omitting `if-then-else` and `while-do` in plans does not reduce the expressiveness of 3APL as they can be represented by abstract plans by using PG and PR rules. For example, the following PG rule of 3APL:

$$g \leftarrow b \mid \pi; \texttt{if } b' \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ end-if}; \pi'.$$

is translated into an abstract plan by one PG rule and two PR rules as follows:

$$\begin{aligned} g \leftarrow b \mid \pi; abs; \pi'. \\ abs \leftarrow b' \mid \pi_1. \\ abs \leftarrow \neg b' \mid \pi_2. \end{aligned}$$

Similarly, the `while-do` construct

$$g \leftarrow b \mid \pi; \texttt{while}\ b'\ \texttt{do}\ \pi_1\ \texttt{end-while}; \pi'.$$

is translated into abstract plans as follows

$$\begin{aligned} g &\leftarrow b \mid \pi; abs; \pi'. \\ abs &\leftarrow b' \mid \pi_1; abs. \\ abs &\leftarrow \neg b' \mid \epsilon. \end{aligned}$$

In order to prove the correctness of the simulation, we need to show the equivalence between runs in 3APL and those by the simulation. Firstly, we specify a deliberation cycle of 3APL. A basic deliberation cycle of 3APL is as follows:

1. Apply applicable rules.
2. Execute plans.

However, we have not defined in detail each of the above two stages. In the stage of applying applicable rules, two extreme approaches are either to apply only one rule, or to apply all the rules until no more are applicable (more precisely, compute a set of applicable rules, if it is not empty, choose a rule and apply it, recompute the set of applicable rules, etc. until the set of applicable rules is empty). In the stage of plan execution, a common approach is to pick a plan and to execute one step. If plans for executing a step are selected randomly, we call this an interleaved execution method. Otherwise, if a selected plan is executed completely before any other plan's steps are selected, we called this a non-interleaved execution method.

In order to demonstrate how to simulate 3APL by means of Meta-APL, we choose the following approach to define how the rule application is done:

1. Apply all applicable PG rules (in any order).
2. Continuously pick an applicable PR rule and apply until no more are applicable.

The two different methods of plan executions are considered in the next sections.

4.2 Simulating interleaved deliberation cycle of 3APL

Firstly, we translate a plan π in 3APL into Meta-APL by translating each element of π. Let us denote the translated plan as $tr(\pi)$.

We translate each type of rules into the corresponding ones in Meta-APL as follows:

– A PG rule $g \leftarrow b \mid \pi$ is translated into: $g, b \rightarrow tr(\pi)$.
– A GR rule $g \leftarrow b \mid g'$ is translated into the following rule:

$$g, b \ \&\ \neg p(g, \_, \texttt{del-goal}(g); X, a, \_) \rightarrow \texttt{del-goal}(g); \texttt{add-goal}(g').$$

The above rule is for replacing a goal g in the goal base with another goal g'. Comparing to the original rule in 3APL, the guard of the translated rule has an extra condition $\neg p(\ldots)$ which is for preventing from applying the same rule and other rules for revising the same goal g once this rule has been fired.

– A PR rule $\pi \leftarrow b \mid \pi'$ is translated into the following rule:

$$\begin{aligned}&p(G, B, P, a, tr(\pi)) \mathbin{\&} b \mathbin{\&}\\ &\neg p(G', p(G, B, P, a, tr(\pi)) \mathbin{\&} B', P', a, \texttt{del-plan}(G, B, P, a, tr(\pi)); X)\\ &\rightarrow \texttt{del-plan}(G, B, P, a, tr(\pi)); tr(\pi').\end{aligned}$$

The above rule is for revising a plan $tr(\pi)$ in the plan base with another plan $tr(\pi')$. Similar to the case of GR rules, the guard of the translated rule also has an extra condition $\neg p(\ldots)$ which is for preventing from applying the same rule and other rules for revising the same plan $tr(\pi)$ once this rule has been fired.

Then, we implement the interleaved deliberation cycle. Below are the rules for implementing the interleaved deliberation cycle of 3APL:

$$\begin{aligned}&\neg p(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, Y)) \mathbin{\&} p(G3, B3, P3, a, X)\\ &\qquad \rightarrow \texttt{step}(G3, B3, P3, a, X).\end{aligned}$$

$$\begin{aligned}&p(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, X))\\ &\mathbin{\&} p(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\ &\qquad \rightarrow \texttt{del-plan}(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, X)).\end{aligned}$$

$$\begin{aligned}&p(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, X))\\ &\mathbin{\&} p(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\ &\qquad \rightarrow \texttt{step}(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X').\end{aligned}$$

$$\begin{aligned}&p(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, X))\\ &\mathbin{\&} X \neq \texttt{del-plan}(G3, B3, P3, a, Y); Y'\\ &\mathbin{\&} \neg p(G4, B4, P4, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\ &\mathbin{\&} p(G5, B5, P5, a, \texttt{del-plan}(G6, B6, P6, a, Z); Z')\\ &\qquad \rightarrow \texttt{step}(G5, B5, P5, a, \texttt{del-plan}(G6, B6, P6, a, Z); Z').\end{aligned}$$

$$\begin{aligned}&p(G1, B1, P1, a, \texttt{step}(G2, B2, P2, a, X))\\ &\mathbin{\&} X \neq \texttt{del-plan}(G3, B3, P3, a, Y); Y'\\ &\mathbin{\&} \neg p(G4, B4, P4, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\ &\mathbin{\&} p(G5, B5, P5, a, \texttt{exec}(G6, B6, P6, a, Z))\\ &\qquad \rightarrow \texttt{step}(G5, B5, P5, a, \texttt{exec}(G6, B6, P6, a, Z)).\end{aligned}$$

We also have the following rule for executing the rules which translate GR rules:

$$p(G1, B1, P1, a, \texttt{del-goal}(G); Z) \rightarrow \texttt{exec}(G1, B1, P1, a, \texttt{del-goal}(G); Z).$$

The first rule selects an arbitrary plan in the plan base for execution but of only one step. Then, once applied, the rule is not applicable any more in that cycle, hence, we can prevent selecting another plan to apply at the same cycle. When a plan is selected to execute but it is also selected to be revised by another rule, we transfer the selection to the newly revised plan by means of the next two rules. Finally, the last two rules are there blocking the above selection when there are the application of rules to revise plans or goals where they generate plans starting with either `del-plan` or `exec`.

Let us revisit the example in [4] as an illustration of how the translation from 3APL into Meta-APL works. The example is about moving blocks on a floor to a desired configuration by an agent which has the power to put a block which has nothing on the top on the floor or on top of other block which also has no block on top. In the example, there are three blocks namely a, b and c. The initial setting is that a and b are on the floor while c is on top of a. The desired setting is that c is on the floor, b is on c and a is on b.

In 3APL, the program of the agent is as follows:

- Belief base: $on(a, floor)$, $on(b, floor)$, $on(c, a)$
- Goal base: $on(c, floor) \wedge on(b, c) \wedge on(a, b)$
- Rule base:

$$\begin{aligned} on(X,Y) &\leftarrow \neg on(X,Y) \mid clear(X); clear(Y); move(X,Y). \\ clear(X); Z &\leftarrow on(Y,X) \wedge X \neq floor \mid clear(Y); move(Y, floor); Z. \\ clear(X); Z &\leftarrow \neg on(Y,X) \mid Z. \\ clear(floor); Z &\leftarrow \top \mid Z. \end{aligned}$$

The set of rules above is translated into Meta-APL as follows:

$$\begin{aligned} & on(X,Y), \neg on(X,Y) \\ & \qquad \rightarrow !clear(X); !clear(Y); \#move(X,Y). \end{aligned}$$

$$\begin{aligned} & p(G,B,P,a,!clear(X);Z) \;\&\; on(Y,X) \;\&\; X \neq floor \\ & \;\&\; \neg p(G', p(G,B,P,a,!clear(X);Z) \;\&\; B', P', a, \\ & \qquad\qquad\qquad\qquad \texttt{del-plan}(G,B,P,a,!clear(X);Z);Z') \\ & \qquad \rightarrow \texttt{del-plan}(G,B,P,a,!clear(X);Z); !clear(Y); \#move(Y, floor); Z. \end{aligned}$$

$$\begin{aligned} & p(G,B,P,a,!clear(X);Z) \;\&\; \neg on(Y,X) \\ & \;\&\; \neg p(G', p(G,B,P,a,!clear(X);Z) \;\&\; B', P', a, \\ & \qquad\qquad\qquad\qquad \texttt{del-plan}(G,B,P,a,!clear(X);Z);Z') \\ & \qquad \rightarrow \texttt{del-plan}(G,B,P,a,!clear(X);Z); Z. \end{aligned}$$

$$\begin{aligned} & p(G,B,P,a,!clear(floor);Z) \\ & \;\&\; \neg p(G', p(G,B,P,a,!clear(floor);Z) \;\&\; B', P', a, \\ & \qquad\qquad\qquad\qquad \texttt{del-plan}(!clear(floor);Z);Z') \\ & \qquad \rightarrow \texttt{del-plan}(!clear(floor);Z); Z. \end{aligned}$$

Notice that the above rules are in the abbreviated form.

We sketch here the proof that one cycle in 3APL corresponds to one or more cycles in the simulation by Meta-APL (hence the runs of the two programs are equivalent). There are two cases:

1. Consider a cycle in 3APL, if there are no PR and GR rules applicable, at the end of the cycle, a plan is selected for execution. The corresponding run in Meta-APL also contains only a single cycle, as no rules into which PR and GR rules are translated are applicable, there is no plan starting with `del-plan` or `exec`, hence, a plan which is selected for execution of one step is not blocked. The corresponding cycle is chosen by selecting the corresponding plan in the case of 3APL.
2. Consider a cycle in 3APL where some PR or GR rules are applicable, at the stage of applying PR and GR rules, it is repeated until no more PR and GR rules are applicable. We define sequences of PR rule application which are sequences of plans $\pi_0, \ldots, \pi_k$ where π_i is obtained by appling some PR rule to revise $\pi_{i-1}$for all $i \geq 1$. We also define a sequence of GR rule application as a sequence of goals $g_0, \ldots, g_m$ where g_i is replaced by applying some GR rule for all $i \geq 0$. Then, let $n + 1$ be the length of the longest sequence among sequences of PR rule applications, then we construct a run in Meta-APL containing $n + m + 1$ cycles where the starting and ending configurations are equivalent to configurations in 3APL before and after the cycle, respectively. In the first cycle, we apply all translated PR rules which are applicable by following the order of PR rule application in 3APL (ignore those which is not applicable yet). Of course, since some PR rule is applied, any plan which is selected for execution is blocked. We repeat this again and again and after n cycles, we must reach a configuration where no more PR rules are applicable. Then, the next cycles are for applying GR rules in the order of the corresponding to the sequence of GR rule application. The final cycle is just for selecting the corresponding plan in 3APL for execution (and it is not blocked as no more PR and GR rules are applicable).

The reverse direction can be shown similarly.

4.3 Simulating non-interleaved deliberation cycle of 3APL

In this section, we simulate the non-interleaved deliberation cycle of 3APL. In this deliberation cycle, we keep executing a plan and any plans which revise this plan until it becomes empty. The implementation of the non-interleaved deliberation cycle is quite similar to the case of interleaved deliberation cycle as in the previous section except that we interchange the use of the meta-actions `step` and `exec`.

In particular, we keep the function tr to translate plans, the translation of PG, GR and PR rules from 3APL to Meta-APL unchanged. The only difference comparing to the case of the interleaved deliberation cycle of 3APL is the implementation of the non-interleaved deliberation cycle. The rules to implement the non-interleaved deliberation cycle of 3APL are as follows:

$$\neg p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, Y)) \ \&\ p(G3, B3, P3, a, X)$$

$$\rightarrow \texttt{exec}(G3, B3, P3, a, X).$$

$$\begin{aligned}
&p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X))\\
&\ \&\, p(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\
&\qquad \rightarrow \texttt{del-plan}(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X)).
\end{aligned}$$

$$\begin{aligned}
&p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X))\\
&\ \&\, p(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\
&\qquad \rightarrow \texttt{exec}(G3, B3, P3, a, \texttt{del-plan}(G2, B2, P2, a, X); X').
\end{aligned}$$

$$\begin{aligned}
&p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X))\\
&\ \&\, X \neq \texttt{del-plan}(G3, B3, P3, a, Y); Y'\\
&\ \&\, \neg p(G4, B4, P4, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\
&\ \&\, p(G5, B5, P5, a, \texttt{del-plan}(G6, B6, P6, a, Z); Z')\\
&\qquad \rightarrow \texttt{step}(G5, B5, P5, a, \texttt{del-plan}(G6, B6, P6, a, Z); Z').
\end{aligned}$$

$$\begin{aligned}
&p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X))\\
&\ \&\, X \neq \texttt{del-plan}(G3, B3, P3, a, Y); Y'\\
&\ \&\, \neg p(G4, B4, P4, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\
&\ \&\, p(G5, B5, P5, a, \texttt{step}(G6, B6, P6, a, \texttt{add-goal}(G); Z))\\
&\qquad \rightarrow \texttt{step}(G5, B5, P5, a, \texttt{step}(\texttt{add-goal}(G); Z)).
\end{aligned}$$

$$\begin{aligned}
&p(G1, B1, P1, a, \texttt{exec}(G2, B2, P2, a, X))\\
&\ \&\, X \neq \texttt{del-plan}(G3, B3, P3, a, Y); Y'\\
&\ \&\, \neg p(G4, B4, P4, a, \texttt{del-plan}(G2, B2, P2, a, X); X')\\
&\ \&\, p(G5, B5, P5, a, \texttt{step}(G6, B6, P6, a, \texttt{del-goal}(G); Z)))\\
&\qquad \rightarrow \texttt{step}(G5, B5, P5, a, \texttt{step}(G6, B6, P6, a, \texttt{del-goal}(G); Z))).
\end{aligned}$$

Similar to the implementation of the interleaved deliberation cycle of 3APL, the first rule is also to select a plan to execute by using the meta action `exec`. Since `exec` is used, this selection of the plan to execute is still kept in the next deliberation cycle if it does not become empty. We also have the next two rules is for changing the selection of a plan to its parents when it is revised by some rule. Finally, the last three rules are also for blocking the selected plan from being executed if some plans or goals are revised. Notice that we have more than one rule comparing to the implementation of the interleaved deliberation cycle of 3APL.

In order to execute the rules which translate GR rules, we have the following rules:

$$p(G1, B1, P1, a, \texttt{del-goal}(G); Z) \rightarrow \texttt{step}(G1, B1, P1, a, \texttt{del-goal}(G); Z).$$
$$p(G1, B1, P1, a, \texttt{add-goal}(G); Z) \rightarrow \texttt{step}(G1, B1, P1, a, \texttt{add-goal}(G); Z).$$

It is straightforward to prove that the translated program in Meta-APL simulates the non-interleaved deliberation cycle of 3APL. The proof is similar to that of the interleaved case in the previous section.

5 Conclusions and future work

We have introduced the syntax and operational semantics of Meta-APL. We have sketched how it can be used to simulate programs written in other agent programming languages together with their operational semantics. In our future work, we plan to develop automatic methods for producing provably equivalent translations of agent programs in Meta-APL and a set of tools for automatically verifying properties of agent systems implemented in Meta-APL.

References

1. N. Alechina, M. Dastani, BS Logan, and J. Meyer. A Logic of Agent Programs. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 795. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
2. M. Dastani, F. de Boer, F. Dignum, and J.J. Meyer. Programming agent deliberation: an approach illustrated using the 3APL language. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 97–104. ACM Press New York, NY, USA, 2003.
3. M. Dastani, F. Dignum, and J.J. Meyer. 3APL: A Programming Language for Cognitive Agents. *ERCIM News, European Research*, 2000.
4. M. Dastani, M.B. van Riemsdijk, F. Dignum, and J.J.C. Meyer. A Programming Language for Cognitive Agents Goal Directed 3APL. *LECTURE NOTES IN COMPUTER SCIENCE.*, pages 111–130, 2003.
5. L.A. Dennis, B. Farwer, R.H. Bordini, and M. Fisher. A flexible framework for verifying agent programs. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1303–1306. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, 2008.
6. R.J. Van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM (JACM)*, 43(3):555–600, 1996.