Auton Agent Multi-Agent Syst (2011) 22:356–381
DOI 10.1007/s10458-010-9129-2

Reasoning about agent deliberation

N. Alechina · M. Dastani · B. S. Logan ·
J.-J. Ch. Meyer

Published online: 23 April 2010

Abstract We present a family of sound and complete logics for reasoning about deliberation strategies for SimpleAPL programs. SimpleAPL is a fragment of the agent programming language 3APL designed for the implementation of cognitive agents with beliefs, goals and plans. The logics are variants of PDL, and allow us to prove safety and liveness properties of SimpleAPL agent programs under different deliberation strategies. We show how to axiomatise different deliberation strategies for SimpleAPL programs, and, for each strategy we prove a correspondence between the operational semantics of SimpleAPL and the models of the corresponding logic. We illustrate the utility of our approach with an example in which we show how to verify correctness properties for a simple agent program under different deliberation strategies.

Keywords Agent programming languages · Agent logics · Reasoning about agent programs · Verification of agent programs · Agent deliberation and agent executions

1 Introduction

The design and development of software agents have become important and challenging topics of research. However there remains a gap between theory and practice in this area, in particular when the design of cognitive, BDI-based agents is concerned. For this kind of

N. Alechina · B. S. Logan
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
e-mail: nza@cs.nott.ac.uk

B. S. Logan
e-mail: bsl@cs.nott.ac.uk

M. Dastani (✉) · J.-J. Ch. Meyer
Intelligent Systems Group, Utrecht University, Utrecht, The Netherlands
e-mail: mehdi@cs.uu.nl

J.-J. Ch. Meyer
e-mail: jj@cs.uu.nl

advanced software agent, methods are needed to verify whether their implementation conforms to their specification. In this paper we pursue our investigations in this direction in the sense that we aim at verifying (agent) programs written in a BDI-based agent programming language. In particular we focus here on logical means to reason about the agent's deliberation strategy.

The deliberation strategy, also called the deliberation process, is the core building block of the interpreters of the agent programming languages. The deliberation strategy determines which goals the agent will attend to and when, and how the agent's plans to achieve these goals are executed. Even if the agent's program is capable in principle of achieving a particular goal in a given situation, a particular deliberation strategy may mean that the relevant actions never get executed, or are executed in such a way as not to achieve the goal.

For some existing BDI-based agent programming languages [6] the deliberation strategy forms integral part of their semantics. However most agent platforms provide customisation mechanisms that allow the agent developers to influence aspects of the deliberation strategy, for example, Jason [8], 3APL [12], 2APL [9,10] and Jadex [21]. In some cases, a BDI-based agent programming language gives the agent developer complete control of the deliberation strategy. For example, PRS [15] allows an agent's deliberation strategy to be tailored to a particular problem through the use of 'Meta-Acts'—plans which can determine which goals or events give rise to intentions and the order in which the currently intended plans are executed. In our opinion, such control over (aspects of) program execution and the deliberation process in particular is extremely important in allowing the agent developer to tailor the execution of an agent's program to the requirements of a particular problem, e.g., by varying the balance between reactive and deliberative behaviour, or varying the number of goals an agent will attend to simultaneously.

As an agent's behaviour is determined by both the agent's program and the deliberation strategy used, it is important for the agent developers to verify properties of programs in the context of a particular deliberation strategy. Of course, one can ignore the impact of any particular deliberation strategy and examine the properties of an agent program that are valid under *all* deliberation strategies. However, we believe that most interesting properties of agent programs, e.g., goal attainment, depend critically on the chosen deliberation strategy, and that the correctness of agent programs can only be examined if one is able to reason about deliberation strategies. While there has been considerable research on reasoning about and verification of BDI agents, e.g., [4,7,17,18,23], there has been much less work on deliberation strategies. An exception is the work of Mulder et al. [20] who present a model of the execution of PRS in an executable temporal logic, MML. Agent plans are represented as temporal formulas and deliberation strategies are represented by sets of MML rules. The rules define the behaviour of a meta-interpreter operating on terms which are names for temporal formulas. The MML model allows the direct specification and verification (via execution in concurrent MetateM) of agent properties.

In this paper we explore a different approach. We present a family of PDL-like logics for reasoning about deliberation strategies; deliberation strategies are expressed as axioms in the logics, and the logics are complete and decidable. Declarative agent programs can naturally be expressed in the form of logical axioms (compared to, say, a model-checking approach). We believe such axiomatisations helps in understanding the semantics of different deliberation strategies. We consider deliberation strategies in the context of a simple APL-like [6,11] agent programming language, SimpleAPL introduced in [2]. We sketch the syntax of SimpleAPL, give its operational semantics, and define four alternative deliberation strategies for SimpleAPL programs which are typical of those used in BDI-based agent programming languages. We then introduce the syntax and semantics of the logics to reason about safety and liveness properties of SimpleAPL programs under these deliberation strategies. We provide sound and complete axiomatisations of the logics, and prove a correspondence between the operational semantics of SimpleAPL and the models of the logics for the program deliberation strategies we consider. Finally, we show how to translate agent programs written in SimpleAPL into logical expressions, and, using a simple example program, show how the agent's deliberation strategy can determine whether a given program will achieve a particular goal.

In contrast to previous work, e.g., [2] where two basic deliberation strategies, interleaved and non-interleaved, were 'hard-coded' into the translation of an agent program or model-checking based approaches, the approach presented here uses a single fixed translation of the agent program together with an axiomatisation of the agent's deliberation strategy. The modularity of our approach makes it easy to verify a property under different deliberation strategies—rather than developing a new encoding for the combined program and deliberation strategy, we simply need to replace the axioms which define the deliberation strategy. Moreover, although we focus on a particular agent programming language and a small number of deliberation strategies, our methodology is general enough to accommodate any deliberation strategy that can be formulated in terms of distinct phases such as plan selection and plan execution phases, and the kinds of operations that can be performed in each phase. As such, we believe it represents a significant advance on previous work, both in the ease with which meta reasoning strategies can be expressed and in more clearly characterising their properties.

2 SimpleAPL

SimpleAPL is a fragment of the agent-oriented programming language 3APL [6,11]. SimpleAPL contains the core features of 3APL, and allows the implementation of agents with beliefs, goals, actions, plans, and planning rules. The main features of 3APL we have omitted are a first order language for beliefs and goals, belief and goal test actions, and some basic actions such as actions for adopting/dropping goals and beliefs. We have omitted these features in order to simplify the presentation; they do not present a significant technical challenge for our approach. 3APL assumes finite domains and can be reduced to a propositional language by considering all possible substitutions. Belief and goal test actions were considered in [2] and the omission of actions to adopt/drop subgoals, while an important practical issue, does not result in a reduction in expressive power. SimpleAPL retains the declarative goals of 3APL and the agent chooses which plan to adopt to achieve a goal using planning goal rules (see below).

In SimpleAPL, an agent's state is specified in terms of its beliefs and goals and its program by a set of plans. The *beliefs* of an agent represent the agent's information about its environment, while its *goals* represent situations the agent wants to realise (not necessary all at once). For simplicity, we only allow the agent's beliefs to be a set of atoms (positive literals) and goals to be a set of literals. For beliefs we assume the closed-world assumption, i.e., the agent believes $-p$ if and only if p is not a belief. For example, an agent might believe that it is at home and that it is raining:

```
Beliefs: home, raining
```

and its goals may be to have breakfast and go to work:

```
Goals: breakfast, work
```

The beliefs and goals of an agent are related to each other: if an agent believes p, then it will not pursue p as a goal. In other words, in each state, the agent's beliefs and goals are disjoint.

Belief update actions change the beliefs of the agent. A belief update action is specified in terms of its pre- and postconditions (which are sets of literals), and can be executed if the belief literals in one of its preconditions are entailed by the agent's current set of beliefs (under the closed-world assumption). Executing the action updates the agent's beliefs with the belief literals in the action's postcondition. For example, the following belief update specification

```
BeliefUpdates:
  {home} walk_work {-home, work}
```

can be read as "if the agent is at home it can walk to work, after which it is at work (and not at home)". Belief update actions maintain consistency of the agent's beliefs, i.e., if p is in the belief set and the belief set is updated by $-p$, then p is removed from the belief set. Goals which are achieved by the postcondition of an action are dropped. For example, if the agent is at home and has a goal of being at work, executing a `walk_work` action will cause it to drop the goal. For simplicity, we assume that the agent's beliefs about its environment are always correct and its actions in the environment are always successful. This assumption can be relaxed, which would require including the state of the environment in the models, so that we can talk about properties of the environment and about agent's beliefs about those properties. Pre- and post-conditions of actions would then be expressed in terms of propositions about the environment, and the agent may have incorrect beliefs concerning preconditions or results of actions.

In order to achieve its goals, an agent adopts *plans*. A plan consists of belief update actions composed by sequence, conditional choice and conditional iteration operators. The sequence operator '`;`' takes two plans as arguments and indicates that the first plan should be performed before the second plan. The conditional choice and conditional iteration operators allow branching and looping and generate plans of the form '`if` ϕ `then` π_1 `else` π_2' and '`while` ϕ `do` π' respectively. The condition ϕ is evaluated with respect to the agent's current beliefs. For example, the plan

```
if raining then take_umbrella else take_sunglasses ;
walk_work
```

causes the agent to take an umbrella if it is raining and sunglasses if it is not, and then walk to work.

To select appropriate plans, the agent uses *planning goal rules*. A planning goal rule consists of three parts: an (optional) goal query specifying the goal(s) the plan achieves, a belief query characterising situation(s) in which it could be a good idea to adopt the plan, and the body of the rule. Applying a planning goal rule causes the agent to adopt the plan which forms the body of the rule. For example, the planning goal rule:

```
work <- home | if raining then take_umbrella
               else take_sunglasses ;
               walk_work
```

states that "if the agent's goal is to be at work and it believes it is at home, then it will adopt the specified plan". For simplicity, we assume that agents do not have initial plans, i.e., plans can only be generated during the agent's execution by planning goal rules.

The syntax of SimpleAPL is given below in EBNF notation. We assume a set of belief update actions and a set of propositions, and use ⟨*aliteral*⟩ to denote the name of a belief update action, ⟨*atom*⟩ to denote an atom, and ⟨*literal*⟩ to denote a literal.

```
⟨APL_Prog⟩       ::=  "BeliefUpdates:" ⟨updatespecs⟩
                      "Beliefs:" ⟨beliefs⟩
                      "Goals": ⟨goals⟩
                      "PG rules:" ⟨pgrules⟩
⟨updatespecs⟩    ::=  [⟨updatespec⟩ ("," ⟨updatespec⟩)* ]
⟨updatespec⟩     ::= "{" ⟨literals⟩ "}"  ⟨aliteral⟩  "{"⟨literals⟩"}"
⟨beliefs⟩        ::=  [⟨atom⟩ ("," ⟨atom⟩)*]
⟨goals⟩          ::=  [⟨literals⟩]
⟨plan⟩           ::=  ⟨baction⟩ | ⟨sequenceplan⟩ | ⟨ifplan⟩ | ⟨whileplan⟩
⟨baction⟩        ::=  ⟨aliteral⟩
⟨sequenceplan⟩   ::=  ⟨plan⟩ ";" ⟨plan⟩
⟨ifplan⟩         ::=  "if" ⟨query⟩ "then {" ⟨plan⟩ "}" ["else {" ⟨plan⟩ "}"]
⟨whileplan⟩      ::=  "while" ⟨query⟩ "do {" ⟨plan⟩ "}"
⟨pgrules⟩        ::=  [⟨pgrule⟩ ("," ⟨pgrule⟩)* ]
⟨pgrule⟩         ::=  [⟨literal⟩] "<-" ⟨query⟩ "|" ⟨plan⟩
⟨query⟩          ::=  ⟨literal⟩ | ⟨query⟩ "and" ⟨query⟩ | ⟨query⟩ "or" ⟨query⟩
⟨literals⟩       ::=  [⟨literal⟩ ("," ⟨literal⟩)*]
```

We will use the following simple agent program as a running example in the remainder of the paper:

```
BeliefUpdates:
  {home} take_umbrella {umbrella}
  {home} take_sunglasses {sunglasses}
  {home} walk_work {-home, work}
  {home} eat_breakfast {breakfast}

Beliefs:
  home, raining

Goals:
  breakfast, work

PG rules:
  r1: work <- home |
        if raining then take_umbrella
        else take_sunglasses;
        walk_work

  r2: breakfast <- home | eat_breakfast
```

This program implements an agent that initially believes it is at home and it is raining, and wants to have breakfast and to be at work. It is important to note that the agent does not necessarily need to achieve these goals at the same time. The first planning goal rule can be applied to get the agent from home to work. The application of this rule causes the agent to take an umbrella if it rains or otherwise take sunglasses, after which it will walk to work. The second planning goal rule can be applied to get the agent to have breakfast when it is at home. As already explained, the belief updates specify when the agent's actions can be performed and what are their effects.

3 Operational semantics

We define the formal semantics of SimpleAPL in terms of a transition system. Each transition corresponds to a single execution step and takes the system from one configuration (defined as the agent's current beliefs, goals and plans) to another. We assume that the execution of basic actions and the application of planning goal rules are atomic operations.

Definition 1 The configuration of an agent is defined as $\langle \sigma, \gamma, \Pi \rangle$ where σ is a set of atoms representing the agent's beliefs, γ is a set of literals representing the agent's goals, and Π is a set of plan entries $r_i : \pi$ representing the agent's current active plans, where π is a plan (possibly partially executed) and r_i the planning goal rule which caused the agent to adopt this plan. An agent's initial beliefs and goals are specified by its program, and Π is initially empty.

For the formulation of the operational semantics we need to formalise some basic assumptions. In particular, we use the notion of belief entailment based on the closed-world assumption. This notion of entailment, which we denote by $\models_{cwa}$, is defined as follows:

$$\begin{array}{ll}
\sigma \models_{cwa} p & \Leftrightarrow p \in \sigma \\
\sigma \models_{cwa} -p & \Leftrightarrow p \notin \sigma \\
\sigma \models_{cwa} \phi \texttt{ and } \psi & \Leftrightarrow \sigma \models_{cwa} \phi \text{ and } \sigma \models_{cwa} \psi \\
\sigma \models_{cwa} \phi \texttt{ or } \psi & \Leftrightarrow \sigma \models_{cwa} \phi \text{ or } \sigma \models_{cwa} \psi \\
\sigma \models_{cwa} \{\phi_1, \ldots, \phi_n\} & \Leftrightarrow \forall 1 \le i \le n \;\; \sigma \models_{cwa} \phi_i
\end{array}$$

The notion of goal entailment, denoted by $\models_g$, corresponds to a formula being classically entailed by one of the goals in the goal base γ, and is defined as follows:

$$\begin{array}{ll}
\gamma \models_g p & \Leftrightarrow p \in \gamma \\
\gamma \models_g -p & \Leftrightarrow -p \in \gamma
\end{array}$$

Note that these are the only goal queries allowed by the EBNF definition of the language above.

Each *belief update action* α has a set of preconditions $\texttt{prec}_\texttt{1}(\alpha)$, ..., $\texttt{prec}_\texttt{k}(\alpha)$. Each $\texttt{prec}_\texttt{i}(\alpha)$ is a finite set of belief literals, and any two preconditions for an action α, $\texttt{prec}_\texttt{i}(\alpha)$ and $\texttt{prec}_\texttt{j}(\alpha)$ $(i \neq j)$, are mutually exclusive (both sets of propositional variables cannot be satisfied simultaneously). For each $\texttt{prec}_\texttt{i}(\alpha)$ there is a unique corresponding postcondition $\texttt{post}_\texttt{i}(\alpha)$, which is also a finite set of literals. A belief update action α can be executed if $\sigma \models_{cwa} \texttt{prec}_\texttt{j}(\alpha)$ for some precondition j. The effect of updating a set of beliefs σ with α is given by $T_j(\alpha, \sigma) = \sigma \cup (\{p : p \in \texttt{post}_\texttt{j}(\alpha)\} \backslash \{p : -p \in \texttt{post}_\texttt{j}(\alpha)\})$, (i.e., executing the belief update action α adds the positive literals in its postcondition to the agent's beliefs and removes any existing beliefs if their negations are in the postcondition).

Executing the agent's program modifies its (initial) configuration in accordance with the transition rules presented below.

The successful execution of a belief update action α in a configuration where the plan $r_i : \alpha; \pi$ is in the set of the agent's current plans is given by:

$$\frac{r_i : \alpha; \pi \in \Pi \quad \sigma \models_{cwa} \texttt{prec}_j(\alpha) \quad T_j(\alpha, \sigma) = \sigma'}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma', \gamma', (\Pi \backslash \{r_i : \alpha; \pi\}) \cup \{r_i : \pi\} \rangle} \tag{1a}$$

where $\gamma' = \gamma \backslash \{\phi \in \gamma \mid \sigma' \models_{cwa} \phi\}$ (executing a belief update action causes the agent to drop any goals it believes to be achieved as a result of the update). In this and the following transition rules, the plan π in the sequence plan $\alpha; \pi$ can be empty in which case $\alpha; \pi$ is identical to α. Moreover, we stipulate that $\Pi \cup \{r_i : \;\}= \Pi$.

If an agent has a plan $r_i : \alpha; \pi$ but none of the preconditions of α hold, then attempting to execute α removes the plan from the plan base and does not change the agent's beliefs and goals:

$$\frac{r_i : \alpha; \pi \in \Pi \quad \forall j\ \sigma \not\models_{cwa} \mathtt{prec_j}(\alpha)}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \backslash \{r_i : \alpha; \pi\} \rangle} \tag{1b}$$

Composite plans. The following transition rules specify the effect of executing the conditional choice and conditional iteration operators, respectively.

$$\frac{r_i : (\mathtt{if}\ \phi\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2); \pi \in \Pi \quad \sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi_1; \pi\} \rangle} \tag{2a}$$

$$\frac{r_i : (\mathtt{if}\ \phi\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2); \pi \in \Pi \quad \sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi_2; \pi\} \rangle} \tag{2b}$$

where $\Pi' = \Pi \backslash \{r_i : (\mathtt{if}\ \phi\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2); \pi\}$.

$$\frac{r_i : (\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1); \pi \in \Pi \quad \sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : (\pi_1; \mathtt{while}\ \phi\ \mathtt{do}\ \pi_1); \pi\} \rangle} \tag{3a}$$

$$\frac{r_i : (\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1); \pi \in \Pi \quad \sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi\} \rangle} \tag{3b}$$

where $\Pi' = \Pi \backslash \{r_i : (\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1); \pi\}$. Note that the sequence operator is specified implicitly by the other rules which specify how to execute the first operation in the sequence.

A *planning goal rule* $r_i = \kappa_i \leftarrow \beta_i | \pi_i$ can be applied if κ_i is entailed by the agent's goals and β_i is entailed by the agent's beliefs, and if the plan base does not already contain a (partially executed) plan added by r_i. Applying the rule r_i adds π_i to the agent's plans.

$$\frac{\gamma \models_g \kappa_i \quad \sigma \models_{cwa} \beta_i \quad r_i : \pi \notin \Pi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \cup \{r_i : \pi_i\} \rangle} \tag{4}$$

3.1 Specifying deliberation strategies

The transition rules presented above define the most general model of agent execution in which any atomic operation can be interleaved with any other. More precisely, this *fully-interleaved* deliberation strategy (which we denote by **(i)**) can be defined as: "either apply a planning goal rule, or execute the first action in any of the current plans; repeat". Particular deliberation strategies are restrictions of this fully-interleaved deliberation which prohibit certain execution paths. For example, a simple *non-interleaved* deliberation strategy which executes a single plan to completion before choosing another plan, i.e., "when in a configuration with no plan, choose a planning goal rule non-deterministically, apply it, execute the resulting plan; repeat".

Many deliberation strategies are possible and it is impossible to consider them all in detail. Instead we characterise some typical deliberation strategies in terms of the execution paths they admit. We focus on the non-interleaved strategy (which we denote by **(ni)**) and two simple 'alternating' strategies: one which first applies a planning goal rule and then executes a single basic action of one of the agent's current plans (which we denote **(as)**); and another which first applies a planning goal rule and then executes a single basic action from each of the agent's current plans (which we denote **(am)**). These strategies were chosen as representative of deliberation strategies found in the literature and in current implementations of BDI-based agent programming languages. However none of these strategies (or any other single strategy) is clearly "best" for all agent task environments. For example, the **(ni)** strategy is appropriate in situations where a sequence of actions must be executed 'atomically' in order to ensure the success of a plan. However it means that the agent is unable to respond to new goals until the plan for the current goal has been executed. Conversely, the **(as)** and **(am)** strategies allow an agent to pursue multiple goals at the same time, e.g., allowing an agent to respond to an urgent, short-duration task while engaged in a long-term task. However they can increase the risk that actions in different plans will interfere with each other. It is therefore important that the agent developer has the freedom to choose the strategy which is most appropriate to a particular problem.

To define the deliberation strategies, we assume that the following control actions are available:

`apply_rule(`Π, Λ`)`	if the conditions of transition rule (4) are satisfied for some planning goal rule r_i in Λ, returns $\Pi \cup \{r_i : \pi_i\}$ where π_i is the plan which forms the body of r_i; otherwise returns Π
`choose_plan(`Π`)`	choose a plan π_i from Π
`execute_step(`Π, π_i`)`	if the appropriate conditions of transition rules (1a)–(3b) are satisfied, executes the next step in $\pi_i \in \Pi$, updates the configuration accordingly and returns the updated plan base; otherwise (if the next step in π is not executable) returns $\Pi \backslash \{\pi_i\}$.

Note that the `apply_rule(`Π, Λ`)` control action constitutes to the plan selection phase (transition rule (4)). Similarly, the `choose_plan(`Π`)` control action followed by one or more `execute_step(`Π, π_i`)` control actions together form the plan execution phase (transition rules (1)–(3)).

The non-interleaved strategy (**ni**) can then be defined as:

```
repeat
  $\Pi$ := apply_rule($\Pi$, $\Lambda$)
  $\pi_i$ := choose_plan($\Pi$)
  while ($\Pi$ != {})
    $\Pi$ := execute_step($\Pi$, $\pi_i$)
```

The alternating (single action) strategy (**as**) can be defined as:

```
repeat
  $\Pi$ := apply_rule($\Pi$, $\Lambda$)
  $\pi_i$ := choose_plan($\Pi$)
  $\Pi$ := execute_step($\Pi$, $\pi_i$)
```

and the alternating (multi-action) strategy (**am**) as:

```
repeat
  $\Pi$ := apply_rule($\Pi$, $\Lambda$)
  foreach $\pi_i$ in $\Pi$
    $\Pi$ := execute_step($\Pi$, $\pi_i$)
```

Other strategies can be defined in a similar way. For example, by changing the definition of the `apply_rule` control action we can prioritise adopting plans for particular types of goals or which are triggered by particular beliefs (e.g., high priority beliefs or goals). Similarly, by changing the definition of the `choose_plan` control action, we can preferentially return plans which achieve high priority goals etc.

Consider the planning goal rules from the example program:

```
r1: work <- home |
      if raining then take_umbrella
      else take_sunglasses;
      walk_work

r2: breakfast <- home | eat_breakfast
```

In a state where both rules are applicable and it is raining, the non-interleaved execution strategy results in one of two possible executions. The first one will apply rule r1 first - the resulting plan base will contain `if raining then take_umbrella else take_sunglasses; walk_work` and executing the plan. Note that after getting to work, the second rule is not applicable. Informally, for brevity, we will represent the sequence of steps in this execution as

```
r1, rain?, take_umbrella, walk_work
```

Another possible sequence of steps is

```
r2, eat_breakfast, r1, rain?, take_umbrella, walk_work
```

However, if the agent adopts the alternating single step strategy, we may get

```
r1, rain?, r2, take_umbrella, (no rule), walk_work
```

(note that at the action execution stage, the agent has to execute the next step of *some* plan in the plan base, so it may always execute the next step of the plan asserted by r1. After the agent performed `walk_work`, the next step of the second plan, namely `eat_breakfast`, is not executable). An example of program execution under the alternating multi-step strategy would be

```
  r1, rain?, r2, take_umbrella, eat_breakfast, (no rule),
walk_work
```

Note that after `eat_breakfast` is executed, no rule is applicable, so we again execute the first step of a plan.

4 Logic

In this section we introduce a series of logics to describe transition systems corresponding to the **(i)**, **(ni)**, **(as)** and **(am)** deliberation strategies. The language of our logic is based on (test-free) Propositional Dynamic Logic PDL (see, e.g., [16]). PDL is a logic to reason about programs. The language of test-free PDL language is defined with respect to a set of propositional variables and a set of atomic programs. Complex program expressions are built from atomic programs, sequential composition ‘;’ ($\rho_1; \rho_2$ means program ρ_1 followed by ρ_2), union ‘$\cup$’ ($\rho_1 \cup \rho_2$ means executing either ρ_1 or ρ_2), and finite iteration ‘$*$’ (ρ^* means executing ρ 0 or finitely many times). For each program expression ρ, the language contains a modality $\langle\rho\rangle$. PDL formulas are defined as follows: $p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle\rho\rangle\phi$ and interpreted on labelled transition systems, where labels are atomic programs. A formula $\langle\rho\rangle\phi$ is true in a state s if there exists a state reachable from s by a path described by ρ, which satisfies ϕ; intuitively, if there is a possible execution of program ρ which results in a state satisfying ϕ.

4.1 Language

We extend the standard language of PDL with belief and goal operators, and with an interleaving program constructor $\|$ [1], where $\rho_1 \parallel \rho_2$ means the interleaved execution of actions of ρ_1 and actions of ρ_2.[1] The belief modality B and goal modality G are introduced to indicate the modality of a proposition, i.e., to indicate whether a proposition should be evaluated with respect to an agent's beliefs or to its goals. They are not interpreted using accessibility relations but as properties of the agent's state. Essentially, the agent believes a propositional variable if it is in the agent's belief base, and the agent has a literal as a goal if it is in the agent's goal base. We have chosen this approach since it has an obvious correspondence with the semantics of SimpleAPL.

We define the language of our logic relative to a set of planning goal rules Λ with plans $\Pi(\Lambda)$ and pre- and post conditions for belief updates $C(\Lambda)$. Let $\Lambda = \{r_1, \ldots, r_n\}$ be the set of planning goal rules, each of which is of the form $r_i = \kappa_i \leftarrow \beta_i \mid \pi_i$. Let $\Pi(\Lambda) = \{\pi_1, \ldots \pi_n\}$ be the set of plans occurring in the rules, and $Ac(\Lambda)$ the finite set of belief update actions occurring in those plans. Let P be the set of atoms, i.e., positive belief and goal literals, occurring in Λ. For each belief update α, we have a set of pre- and postcondition pairs $C(\alpha) = \{(\mathtt{prec}_1(\alpha), \mathtt{post}_1(\alpha)), \ldots, (\mathtt{prec}_k(\alpha), \mathtt{post}_k(\alpha))\}$. We denote the set of all pre- and postconditions for all belief updates in Λ by $C(\Lambda)$, that is, $C(\Lambda) = \{C(\alpha) : \alpha \in Ac(\Lambda)\}$.

We identify key phases in the deliberation cycle by propositional flags, and then write axioms which capture the possible transitions between phases. For the **(i), (ni), (as)** and **(am)** strategies we consider, the flags are: $start_i$, which indicates that plan π_i has started execution; $step_i$, which indicates that the next step of π_i has been executed; and $fail_i$, which indicates that π_i has failed, namely the next belief update action in π_i cannot be executed because its preconditions do not hold.

The set of atomic propositions of our logic consists of:

– a set of propositional variables P
– a set of boolean flags $P_c = \{start_i, fail_i, step_i : r_i \in \Lambda\}$;

The set of 'atomic programs' of our logic consists of:

– for every rule $r_i \in \Lambda$, an atomic action $\delta_{r\,i}$ for apply r_i
– a set of atomic actions $Ac_{ind}(\Lambda) = \{\alpha_i \mid \alpha \in Ac(\Lambda)$ and α appears in $\pi_i \in \Pi(\Lambda)\}$ (i.e. we introduce a new atomic action α_i for every plan π_i in which the belief update action α appears)
– for every test formula ϕ (appearing in an if- or while- plan) in a plan $\pi_i \in \Pi(\Lambda)$, a test action $t_i((\neg)\phi)$. Essentially this is a PDL test operator restricted to tests expressible in SimpleAPL, however unlike PDL tests, it can change the state (set $step_i$ to true, for example)
– for each plan π_i, an atomic action e_i. This action is introduced for technical reasons, and will be used in our translation of Λ as a PDL expression. We append it after each plan to reset the control flags after the plan has finished executing.

A formula of L is defined as follows: if $p \in P$, then Bp and $G(\neg)p$ are formulas (note that B and G operators cannot be nested and that the B operator can only be applied to positive literals). If $p \in P_c$, then p is a formula; if ρ is a program expression and ϕ a formula, then $\langle\rho\rangle\phi$ is a formula; and L is closed under the usual boolean connectives. We define $[\rho]\phi$ as $\neg\langle\rho\rangle\neg\phi$ and use the abbreviation $\langle[\rho]\rangle\phi$ for $[\rho]\phi \wedge \langle\rho\rangle\phi$. (As will be clear from the next section, in general $[\rho]\phi$ does not entail $\langle\rho\rangle\phi$.)

[1] Note that every formula with the interleaving operator can be rewritten without the interleaving operator, however the resulting formula may be doubly exponentially larger [1].

4.2 Semantics

A model M is defined as $(W, \{R_{\alpha_i} : \alpha_i \in Ac_{ind}(\Lambda)\}, \{R_{\delta_{ri}} : r_i \in \Lambda\}, R_{e_i}, V)$ where

- W is a non-empty set of states
- $V = (V_b, V_g, V_c)$ is the evaluation function consisting of belief and goal valuation functions V_b and V_g and control valuation function V_c such that for every $s \in W$,
  $V_b(s) = \{p_1, \ldots, p_m : p_i \in P\}$ is the agent's beliefs in s
  $V_g(s) = \{(-)u_1, \ldots, (-)u_n : u_i \in P\}$ is the agent's goals in s (note that V_g assigns literals rather than propositional variables)
  $V_c(s) \subseteq P_c$ are the control variables true in s
- $R_{\alpha_i}, R_{t_i(\phi)}, R_{\delta_{ri}}, R_{e_i}$ are binary relations on W; R_{α_i} corresponds to belief updates, $R_{t_i(\phi)}$ to belief tests, $R_{\delta_{ri}}$ to firing a rule, and R_{e_i} corresponds to executing the e_i action (intuitively, removing a plan from the plan base).

The conditions on R_{α_i}, $R_{\delta_{ri}}$ and R_{e_i} depend on the deliberation strategy and are defined below.

Given the relations corresponding to atomic programs in M, we can define sets of paths in the model corresponding to any PDL program expression ρ in M. A set of paths $\tau(\rho) \subseteq (W \times W)^*$ is defined inductively:

- $\tau(\alpha_i) = \{(s, s') : R_{\alpha_i}(s, s')\}$
- $\tau(t_i(\phi)) = \{(s, s') : R_{t_i(\phi)}(s, s')\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1; \rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, where $\circ$ is concatenation of paths.
- $\tau(\rho^*)$ is the set of all paths consisting of zero or finitely many concatenations of paths in $\tau(\rho)$
- $\tau(\rho_1 \parallel \rho_2)$ is the set of all paths obtained by interleaving atomic actions and tests from $\tau(\rho_1)$ and $\tau(\rho_2)$.

By an interleaving of two sequences of atomic programs $a_1; \ldots; a_n$ and $b_1; \ldots; b_m$ we mean any sequence of as and bs such that the order within as and bs is preserved; namely a_i should precede a_j in the sequence if $i < j$, and the same for bs. The set of all interleavings of $a_1; a_2$ and b_1 is $\{a_1; a_2; b,\ a_1; b; a_2,\ b; a_1; a_2\}$. However, in the definition of $\tau(\rho_1 \parallel \rho_2)$ we talk about interleavings of *paths* from $\tau(\rho_1)$ and $\tau(\rho_2)$, where paths are sequences of pairs of states. In order to be able to define all possible interleavings of paths inductively, we allow 'illegal paths' of the form $(s_0, s_1), (s_2, s_3)$ in $\tau(\rho)$, where $s_1 \neq s_2$; in other words, concatenation $z_1 \circ z_2$ is defined for paths z_1 and z_2 even when the last state of z_1 is not the same as the first state of z_2. This is different from standard PDL. To see why this is necessary in the presence of the interleaving operator, consider the following example. A path (s_0, s_1),(s_1, s_2), (s_2, s_3) where $(s_0, s_1) \in \tau(\alpha_1)$, $(s_1, s_2) \in \tau(\alpha_3)$ and $(s_2, s_3) \in \tau(\alpha_2)$ should be in $\tau(\alpha_1; \alpha_2 \parallel \alpha_3)$ but this means that an illegal path (s_0, s_1),(s_2, s_3) should be in $\tau(\alpha_1; \alpha_2)$. We will call paths without such 'jumps' *legal paths*. Only legal paths are used in evaluating PDL modalities (see the truth definition below).

The satisfaction relation $\models$ of a formula being true in a state of a model is defined inductively as follows:

- $M, s \models Bp$ iff $p \in V_b(s)$
- $M, s \models G(-)p$ iff $(-)p \in V_g(s)$
- $M, s \models p$ iff $p \in V_c(s)$, where $p \in P_c$
- $M, s \models \neg\phi$ iff $M, s \not\models \phi$
- $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$

– $M, s \models \langle \rho \rangle \phi$ iff there is a legal path in $\tau(\rho)$ starting in s which ends in a state s' such that $M, s' \models \phi$.

We use the $start_i$ flags to signal that plan π_i has started executing; it is set to true when the planning goal rule r_i is applied and prevents repeated application of r_i. We use the $step_i$ flags to say that a single step of plan π_i has been executed. If a belief update action α_i of plan π_i cannot be executed, the $fail_i$ flag is set. Finally, the special action e_i, which is appended to the end of plan π_i in our translation of the agent program, resets the $start_i$, $step_i$ and $fail_i$ flags to false.

Models for all deliberation strategies satisfy the following conditions, for all $s \in W$:

C1 $V_g(s) \cap V_b(s) = \emptyset$ and $\{p : -p \in V_g(s)\} \subseteq V_b(s)$ (Beliefs and goals are disjoint.)

C2 If $fail_i \in V_c(s)$, then for every u_i where u_i is an action α_i or test $t_i(\phi)$ of plan π_i, $R_{u_i}(s, s)$ and for no $s' \neq s$, $R_{u_i}(s, s')$.
(If the $fail_i$ flag has been set, this causes all subsequent actions in π_i to be 'consumed' without changing the state, mimicking the deletion of the remaining steps of π_i.)

C3 $R_{e_i}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \setminus \{start_i, fail_i, step_i\}$.
(e_i sets $start_i$, $fail_i$ and $step_i$ to false.)

C4 If ϕ and $fail_i$ are false in s, then there is no s' such that $R_{t_i(\phi)}(s, s')$ (strictly speaking, if the translation of ϕ in our logical language, $f_b(\phi)$, which is defined in the next section, is false in s).

5 Axiomatisation

Different deliberation strategies require different conditions on applicability of actions and rules and can be characterised by different sets of axioms. In order to specify different conditions and axiomatisations for different strategies, we first explain how different components of agent programs can be translated into PDL expressions.

The beliefs, goals and plans of agent programs are translated into PDL expressions using translation functions f_b, f_g and f_p as follows:

– translation of belief formulas: let $p \in P$ and ϕ, ψ be belief query expressions of SimpleAPL (boolean combinations of literals)

  - $f_b(p) = Bp$
  - $f_b(\phi \texttt{ and } \psi) = f_b(\phi) \wedge f_b(\psi)$
  - $f_b(\phi \texttt{ or } \psi) = f_b(\phi) \vee f_b(\psi)$

  In addition,

  - $f_b(\neg\phi) = \neg f_b(\phi)$ (we need this clause for translating expressions of the form $t_i(\neg\phi)$ in translations of if- and while- plans below)
  - $f_b(X) = \bigwedge_{p \in X} Bp \wedge \bigwedge_{-p \in X} \neg Bp$ where X is a set of literals (this is for translating pre- and post-conditions of a belief update)

– translation of goal formulas: let $p \in P$

  - $f_g(p) = Gp$
  - $f_g(-p) = G{-p}$

– translation of plan expressions: let α be a belief update action, ϕ a belief query expression, and π, π_1, π_2 be plan expressions of SimpleAPL, all occurring in a plan i:

- $f_p(\alpha) = \alpha_i$
- $f_p(\pi_1; \pi_2) = f_p(\pi_1); f_p(\pi_2)$
- $f_p(\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2) = (t_i(\phi); f_p(\pi_1)) \cup (t_i(\neg\phi); f_p(\pi_2))$
- $f_p(\texttt{while}\ \phi\ \texttt{do}\ \pi) = (t_i(\phi); f_p(\pi))^*; t_i(\neg\phi)$

Different deliberation strategies require different conditions on models. We now state these conditions and provide complete axiomatisations for the corresponding classes of models.

5.1 Conditions on models for the fully-interleaved strategy **(i)**

Models corresponding to the fully-interleaved deliberation strategy **(i)** in addition conform to the following constraints.

C5 If $f_b(\texttt{prec}_j(\alpha))$ and $\neg fail_i$ are true in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$, $V_g(s') = V_g(s) \backslash (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ and $V_c(s') = V_c(s)$.
(Corresponds to transition (1a): when action α is successfully executed, transit to a state where the beliefs and goals are modified according to the action specification.)

C6 If $f_b(\phi)$ and $\neg fail_i$ are true in s, then $R_{t_i(\phi)}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s)$. (Corresponds to evaluating the test formula in transitions (2a) and (3a).)

C7 If $\vee_j f_b(\texttt{prec}_j(\alpha))$ and $fail_i$ are false in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = V_b(s)$ and $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{fail_i\}$.
(Corresponds to transition (1b): if an action of π_i is not executable (i.e., none of its preconditions hold) transit to a state where $fail_i$ is true.)

C8 If $r_i = \kappa_i \leftarrow \beta_i \mid \pi_i$ is a PG rule, then $R_{\delta_{r_i}}(s, s')$ iff $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ are true in s and $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and $V_c(s') = V_c(s) \cup \{start_i\}$. (Corresponds to transition (4): r_i can be fired if, and only if, π_i has not started and the belief and goal conditions of r_i are true.)

Let the class of transition systems defined above be denoted $\mathbf{M}(\Lambda, \mathbf{i})$.

5.2 Axiomatisation for the fully-interleaved strategy **(i)**

CL classical propositional logic
PDL axioms of PDL (see, e.g., [16]) excluding interleaving since it is expressible

A1 $Bp \to \neg Gp$ (corresponds to C1)

A2 $G-p \to Bp$ (corresponds to C1)

A3 $fail_i \wedge \phi \to \langle[u_i]\rangle(fail_i \wedge \phi)$ where ϕ is any formula and u_i is either α_i or $t_i(\phi)$ (corresponds to C2)

A4 $\phi \to \langle[e_i]\rangle(\phi \wedge \neg start_i \wedge \neg fail_i \wedge \neg step_i)$ for any formula ϕ not containing $start_i$ and $fail_i$ (corresponds to C3).

A5 $\neg f_b(\phi) \wedge \neg fail_i \to [t_i(\phi)]\bot$ (corresponds to C4: test is not executable if the formula is false and the plan has not failed)

A6 $f_b(\texttt{prec}_j(\alpha)) \wedge \neg fail_i \wedge \phi \to \langle[\alpha_i]\rangle(f_b(\texttt{post}_j(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_j(\alpha)$ (corresponds to C5)

A7 $f_b(\phi) \wedge \psi \to \langle[t_i(\phi)]\rangle\psi$ (corresponds to C6: tests are executable and don't change the state if the test formula is true)

A8 $\bigwedge_j \neg f_b(\texttt{prec}_j(\alpha)) \wedge \neg fail_i \wedge \phi \to \langle[\alpha_i]\rangle(fail_i \wedge \phi)$ where ϕ does not contain $fail_i$ (corresponds to C7)

A9 $\neg start_i \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge \phi \to \langle[\delta_{r\,i}]\rangle(start_i \wedge \phi)$, where ϕ does not contain $start_i$ (corresponds to C8; ϕ encodes the frame condition that the state does not change apart from setting the $start_i$ flag to true)

A10 $start_i \vee \neg(f_g(\kappa_i) \wedge f_b(\beta_i)) \to [\delta_{r\,i}]\bot$ (corresponds to C8 'only if')

Let us call the axiom system above $\mathbf{Ax}(\Lambda, \mathbf{i})$.

Theorem 1 $\mathbf{Ax}(\Lambda, \mathbf{i})$ *is sound and (weakly) complete for the class of models* $\mathbf{M}(\Lambda, \mathbf{i})$.

Proof The proof of soundness is by straightforward induction on the length of a derivation. All axioms are clearly sound (since they closely correspond to conditions on models), and the inference rules are standard.

The proof of completeness is standard as far as the PDL part is concerned, see for example [5]. Take a consistent formula ϕ. As the building blocks in our construction we will use a set $CL(\phi)$ which includes subformulas of ϕ and a finite number of other formulas specified below. First of all, we define the set of subformulas of ϕ in the usual way, but considering subformulas of the form Bp and $G(-)p$ as atomic formulas (that is, p and $-p$ are not included in the set of subformulas). Then we require that $CL(\phi)$ is closed under subformulas and in addition satisfies the usual conditions for the Fischer–Ladner closure and closure under single negations:

- if $\langle\rho_1; \rho_2\rangle\psi \in CL(\phi)$ then $\langle\rho_1\rangle\langle\rho_2\rangle\psi \in CL(\phi)$
- if $\langle\rho_1 \cup \rho_2\rangle\psi \in CL(\phi)$ then $\langle\rho_1\rangle\psi \vee \langle\rho_2\rangle\psi \in CL(\phi)$
- if $\langle\rho^*\rangle\psi \in CL(\phi)$ then $\langle\rho\rangle\langle\rho^*\rangle\psi \in CL(\phi)$
- if $\psi \in CL(\phi)$ and ψ is not of the form $\neg\chi$, then $\neg\psi \in CL(\phi)$.

plus the following extra conditions:

- $start_i, step_i, fail_i \in CL(\phi)$
- if an action α_i occurs in ϕ, then $CL(\phi)$ contains f_b translations of all pre- and postconditions for α, e.g. if one of α's preconditions is $\{p, -q\}$ then $Bp, \neg Bq \in CL(\phi)$
- if $\langle t(\phi')\rangle\psi \in CL(\phi)$ then $f_b(\phi') \in CL(\phi)$

The states of the satisfying model M will be all maximal consistent subsets of $CL(\phi)$. Let A, B be such maximal consistent sets, and a be any of α_i, $t_i(\phi)$, $\delta_{r\,i}$ or e_i. Then $R_a(A, B)$ holds if and only if the conjunction of formulas in A, $\hat{A}$, is consistent with $\langle a\rangle\hat{B}$ (conjunction of formulas in B preceded by $\langle a\rangle$). Similarly for accessibility relations corresponding to complex programs ρ: $R_\rho(A, B)$ iff $\hat{A} \wedge \langle\rho\rangle\hat{B}$ is consistent. By the standard PDL proof, R_ρ so defined does in fact correspond to the relation in a regular model, for example $R_{\rho_1\cup\rho_2} = R_{\rho_1} \cup R_{\rho_2}$, similarly for ; and $*$.

We define the assignments V_b, V_g and V_c in an obvious way:

- $p \in V_b(A)$ iff $Bp \in A$, where $Bp \in CL(\phi)$;
- $(-)p \in V_g(A)$ iff $G(-)p \in A$, where $G(-)p \in CL(\phi)$;
- $p \in V_c(A)$ iff $p \in A$.

The truth lemma follows easily: for every $\psi \in CL(\phi)$,

$$\psi \in A \;\Leftrightarrow\; M, A \models \psi$$

Since our formula ϕ is consistent, it belongs to at least one maximal consistent set A, so it is satisfied in some state in M.

Now we have to show that the model we constructed satisfies conditions on $\mathbf{M}(\Lambda, (\mathbf{x}))$ for the interleaved strategy. First we show that the conditions common to all strategies hold for the model we constructed:

C1 Clearly, since the states are consistent with respect to the axiom schemas A1 and A2, and by the truth lemma, beliefs are consistent, and beliefs and goals are disjoint.

C2 Let A be a maximal consistent set containing $fail_i$. By axiom A3, if $fail_i \wedge \hat{A}$ is consistent, then $fail_i \wedge \hat{A} \wedge \langle u_i \rangle \hat{A}$ is consistent, so $R_{u_i}(A, A)$ holds. Observe that for any $B \neq A$, $R_{u_i}(A, B)$ does not hold because by A3 again, $fail_i \wedge \hat{A} \rightarrow [u_i]\hat{A}$ so all the states accessible by R_{u_i} should satisfy all the formulas in A. Since the states are maximal, this means that the only accessible state is A.

C3 Let $R_{e_i}(A, B)$. Let us denote by A^b (B^b) the set of all formulas in A (B) starting with the belief operator. Since $\hat{A^b}$ does not contain $start_i$, $fail_i$ and $step_i$, by axiom A4, $\hat{A^b} \rightarrow [e_i]\hat{A^b}$, so since $\hat{A} \wedge \langle e_i \rangle \hat{B}$ is consistent, so is $\hat{A^b} \wedge \hat{B}$, therefore $B^b = A^b$ and $V_b(B) = V_b(A)$. Similarly for the goal formulas and control flags other than $start_i$, $fail_i$ and $step_i$. Finally, since $\hat{A} \rightarrow [e_i](\neg start_i \wedge \neg fail_i \wedge \neg step_i)$, $V_c(B) = V_c(A) \backslash \{start_i, fail_i, step_i\}$. Similarly, using the $\langle e_i \rangle$ version of A4 we can show that for any B which differs from A only in its assignment of false to $start_i$, $fail_i$ and $step_i$, $R_{e_i}(A, B)$ holds.

C4 If the test formula $f_b(\phi)$ is not true in A, and $\neg fail_i$ is true in A, then there is no $t_i(\phi)$ transition out of A by A5.

For the conditions specific to **i** strategy, the proof also exploits the close correspondence between the axioms and conditions on models:

C5 If a state A does not contain a control flag $fail_i$ indicating that some action of plan π_i is not executable, and preconditions of an action α_i hold, axiom A6 ensures that all states B for which $\hat{A} \wedge \langle \alpha_i \rangle \hat{B}$ is consistent, that is $R_{\alpha_i}(A, B)$ holds, satisfy the postconditions of α_i and are otherwise the same as A.

C6 Similar condition for test actions is enforced by axiom A7.

C7 If A contains none of the preconditions of α_i, then the only R_{α_i} transition out of A is to a state which is the same as A but contains $fail_i$, by the axiom A8.

C8 Let A contain $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ (which are the conditions for firing a planning goal rule r_i). Then by the axiom A9 ($\langle \delta_{r_i} \rangle$ part), $\hat{A} \wedge \langle \delta_{r_i} \rangle \hat{B}$ is consistent, where B has the same formulas as A apart from $start_i$ instead of $\neg start_i$. By the $[\delta_{r_i}]$ part of the same axiom, all Bs such that $R_{\delta_{r_i}}(A, B)$ holds are like that, so they have the same assignment of belief and goal formulas and the only difference in control flags is assigning true to $start_i$. Axiom A10 ensures that if a state does not satisfy the conditions for firing a planning goal rule, there is no δ_{r_i} transition from that state. □

5.3 Conditions on models for the non-interleaved strategy **(ni)**

Models corresponding to the non-interleaved strategy **(ni)** satisfy conditions C1–C7 above. C8 is replaced with

C9 If $\bigwedge_j \neg start_j$, $f_g(\kappa_i)$ and $f_b(\beta_i)$ are true in s, then $R_{\delta_{r_i}}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{start_i\}$.
(This strengthens C8 to require that no other planning rule has been fired (not just r_i) to ensure that the agent has only one plan at a time.)

Let the class of transition systems defined above be denoted $\mathbf{M}(\Lambda, \mathbf{ni})$.

5.4 Axiomatisation of the non-interleaved strategy **(ni)**

CL, PDL, A1–A8 as above:

A11 $f_g(\kappa_i) \wedge f_b(\beta_i) \wedge \bigwedge_j \neg start_j \wedge \phi \rightarrow \langle[\delta_{ri}]\rangle(start_i \wedge \phi)$, where ϕ does not contain $start_i$

A12 $\neg(f_g(\kappa_i) \wedge f_b(\beta_i)) \vee \bigvee_j start_j \rightarrow [\delta_{ri}]\bot$.

A11 and A12 replace A9 and A10 from the fully-interleaved strategy and correspond to C9.

Let us call the axiom system above **Ax**$(\Lambda,$ **ni**$)$.

Theorem 2 **Ax**$(\Lambda,$ **ni**$)$ *is sound and (weakly) complete for the class of models* **M**$(\Lambda,$ **ni**$)$.

The proof of soundness and completeness is similar to the proof of Theorem 1. The only difference is in the condition on models for PG rule transitions $R_{\delta_{ri}}$, which corresponds to the axioms A11 and A12.

5.5 Conditions on models for the **(as)** strategy

Recall that the **(as)** strategy assumes the application of one planning goal rule followed by the execution of one action of one plan. We use boolean flags $step_i$ to say that a single step of plan π_i has been executed; when this flag is true for some i, all actions are disabled and we must apply a planning goal rule. Rule application sets all $step_i$ flags to false, re-enabling action execution and disabling rule application. If some $step_i$ is true, but no rules are applicable, we continue to execute actions; conversely, if all $step_i$ are false but all current plans have failed, we re-enable rule application.

To make the conditions more readable, we introduce several abbreviations:

– execution phase: $\mathtt{x} = \bigwedge_{r_i \in \Lambda} \neg step_i$
– plan base is empty:
  $\mathtt{noplans} = \bigwedge_{r_i \in \Lambda}(start_i \rightarrow fail_i)$
– no rules are applicable:
  $\mathtt{norules} = \bigwedge_{r_i \in \Lambda}(start_i \vee \neg(f_g(\kappa_i) \wedge f_b(\beta_i)))$

Models corresponding to **(as)** strategy satisfy C1–C4 above and in addition conform to the following constraints.

C10 If $f_b(\mathtt{prec}_j(\alpha))$, $\neg fail_i$ and $\mathtt{x} \vee \mathtt{norules}$ are true in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$,
$V_g(s') = V_g(s) \backslash (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ and $V_c(s') = V_c(s) \cup \{step_i\}$.
(Corresponds to transition (1a) for **(as)**: an action can be executed if one of its preconditions holds and either action execution is enabled or no rules are applicable.)

C11 If $f_b(\phi)$, $\neg fail_i$ and $\mathtt{x} \vee \mathtt{norules}$ are true in s, then $R_{t_i(\phi)}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{step_i\}$.
(Corresponds to transitions (2a) and (3a) for **(as)**: a test action can be executed if one of its preconditions holds and either action execution is enabled or no rules are applicable.)

C12 If $\bigvee_j f_b(\mathtt{prec}_j(\alpha))$ and $fail_i$ are false in s, and $\mathtt{x} \vee \mathtt{norules}$ is true, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = V_b(s)$ and $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{fail_i, step_i\}$.
(Corresponds to transition (1b): if an action is not executable (i.e., none of its preconditions hold), transit to a state where $fail_i$ and $step_i$ are true, enabling rule execution.)

C13 If $\bigvee_j step_j$ and $\neg fail_i$ are true in s and $\mathtt{norules}$ is false in s, then $R_{u_i}(s, s')$ holds for no u_i (where u_i is α_i or $t_i(\phi)$) (it is not possible to execute the next step of any plan if a step of some plan has been executed, and there are applicable rules).

C14 $R_{\delta_{ri}}(s, s')$ iff $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ and $\neg \mathtt{x} \vee \mathtt{noplans}$ are true in s and $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{start_i\} \cup \{\neg step_j : r_j \in \Lambda\}$.
(Corresponds to transition (4) for the **(as)** strategy: a rule is applicable if the corresponding plan is not in the plan base, the belief and goal conditions of the rule hold, and either rule execution is enabled or all the plans in the plan base have failed.)

Let the class of transition systems defined above be denoted $\mathbf{M}(\Lambda, \mathbf{as})$.

5.6 Axiomatisation of the **(as)** strategy

CL, PDL, A1–A5 as above

A13 $f_b(\mathtt{prec_j}(\alpha)) \wedge \neg fail_i \wedge (\mathtt{x} \vee \mathtt{norules}) \wedge \phi \to \langle[\alpha_i]\rangle(step_i \wedge f_b(\mathtt{post_j}(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\mathtt{post_j}(\alpha)$ and $step_i$. (Corresponds to C10.)

A14 $f_b(\psi) \wedge \neg fail_i \wedge (\mathtt{x} \vee \mathtt{norules}) \wedge \phi \to \langle[t_i(\psi)]\rangle(step_i \wedge \phi)$, where ϕ does not contain $step_i$. (Corresponds to C11.)

A15 $\bigwedge_j \neg f_b(\mathtt{prec_j}(\alpha)) \wedge \neg fail_i \wedge (\mathtt{x} \vee \mathtt{norules}) \wedge \phi \to \langle[\alpha_i]\rangle(fail_i \wedge step_i \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_i$. (Corresponds to C12.)

A16 $\bigvee_j step_j \wedge \neg fail_i \wedge \neg \mathtt{norules} \to [u_i]\bot$ where u_i is either α_i or $t_i(\phi)$. (Corresponds to C13.)

A17 $\neg start_i \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge (\neg \mathtt{x} \vee \mathtt{noplans}) \wedge \phi \to \langle[\delta_{ri}]\rangle(start_i \wedge \bigwedge_j \neg step_j \wedge \phi)$ where ϕ does not contain $start_i$ and $step_j$ for any j. (Corresponds to C14.)

A18 $\neg(f_g(\kappa_i) \wedge f_b(\beta_i)) \vee (x \wedge \neg \mathtt{noplans}) \to [\delta_{ri}]\bot$. (Corresponds to C14, only if.)

Let us call the axiom system above $\mathbf{Ax}(\Lambda, \mathbf{as})$.

Theorem 3 $\mathbf{Ax}(\Lambda, \mathbf{as})$ *is sound and (weakly) complete for the class of models* $\mathbf{M}(\Lambda, \mathbf{as})$.

Again the proof is very similar to the proof of Theorem 1. It exploits the same close correspondence between conditions on models and axioms. The main difference is in the use of an extra type of control flag $step_i$ in conditions and axioms.

5.7 Conditions on models for the **(am)** strategy

Recall that **(am)** strategy assumes the application of one planning goal rule followed by the execution of one action of each plan in the plan base. Below we use the following abbreviation:

– planning phase: $\mathtt{p} = \bigwedge_{r_i \in \Lambda}(start_i \to step_i \vee fail_i)$

Models corresponding to the **(am)** strategy satisfy C1–C4 above and in addition

C15 If $f_b(\mathtt{prec}_j(\alpha))$, $\neg fail_i$ and $\neg step_i$ are true in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$, $V_g(s') = V_g(s) \backslash (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ and $V_c(s') = V_c(s) \cup \{step_i\}$.
(Corresponds to transition (1a) with the additional requirement that π_i has not yet executed the next step; executing α_i sets $step_i$ to true.)

C16 If $f_b(\mathtt{prec}_j(\alpha))$, $\neg fail_i$ and $\mathtt{p} \wedge \mathtt{norules}$ are true in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$,
$V_g(s') = V_g(s) \backslash (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ and $V_c(s') = V_c(s) \backslash \{step_j : j \neq i\}$.
(Corresponds to transition (1a) with the additional requirement that no planning rules are applicable; in such a case every current plan gets to execute one more step.)

C17 If $f_b(\phi)$, $\neg fail_i$ and $\neg step_i$ are true in s, then $R_{t_i(\phi)}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{step_i\}$.

C18 If $f_b(\phi), \neg fail_i, \texttt{p}$ and $\texttt{norules}$ are true in s, then $R_{t_i(\phi)}(s, s')$ iff $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \backslash \{step_j : j \neq i\}$.

C19 If $\bigvee_j f_b(\texttt{prec}_j(\alpha))$ and $fail_i$ are false in s, and $\neg step_i$ is true, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = V_b(s)$ and $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{fail_i, step_i\}$.
(Corresponds to transition (1b) for the case when π_i has not performed a step.)

C20 If $\bigvee_j f_b(\texttt{prec}_j(\alpha))$ and $fail_i$ are false in s, and $\texttt{p} \wedge \texttt{norules}$ is true, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = V_b(s)$ and $V_g(s') = V_g(s)$ and $V_c(s') = (V_c(s) \cup \{fail_i\}) \backslash \{step_j : j \neq i\}$
(Corresponds to transition (1b) when π_i has performed a step, but no rules are applicable.)

C21 If $step_i$ and $\neg fail_i$ are true in s and $(\texttt{p} \wedge \texttt{norules})$ is false in s, then $R_{u_i}(s, s')$ holds for no u_i (where u_i is α_i or $t_i(\phi)$) (it is not possible to execute the next step of a plan if a step of this plan has been executed, and there are applicable rules).

C22 $R_{\delta_{ri}}(s, s')$ iff $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ and $\texttt{p}$ are true in s and $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{start_i\} \cup \{\neg step_j : r_j \in \Lambda\}$.
(A rule can be applied if the corresponding plan has not started, the belief and goal conditions of the rule hold, and all current plans have performed a step or failed.)

Let the class of transition systems defined above be denoted $\mathbf{M}(\Lambda, \mathbf{am})$.

5.8 Axiomatisation of the **(am)** strategy

CL, PDL, A1–A5 as above

A19 $f_b(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge \neg step_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(step_i \wedge f_b(\texttt{post}_\texttt{j}(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_\texttt{j}(\alpha)$ and $step_i$. (Corresponds to C15.)

A20 $f_b(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge step_i \wedge \texttt{p} \wedge \texttt{norules} \wedge \phi \rightarrow$ $\langle[\alpha_i]\rangle(f_b(\texttt{post}_\texttt{j}(\alpha)) \wedge \bigwedge_{j \neq i} \neg step_j \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_\texttt{j}(\alpha)$ and $step_j$ for all $j \neq i$. (Corresponds to C16.)

A21 $f_b(\psi)) \wedge \neg fail_i \wedge \neg step_i \wedge \phi \rightarrow \langle[t_i(\psi)]\rangle(step_i \wedge \phi)$, where ϕ does not contain $step_i$. (Corresponds to 17.)

A22 $f_b(\psi) \wedge \neg fail_i \wedge \texttt{p} \wedge \texttt{norules} \wedge \phi \rightarrow \langle[t_i(\psi)]\rangle(\bigwedge_{j \neq i} \neg step_j \wedge \phi)$, where ϕ does not contain $step_j$ for all $j \neq i$. (Corresponds to C18.)

A23 $\bigwedge_j \neg f_b(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge \neg step_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge step_i \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_i$. (Corresponds to C19.)

A24 $\bigwedge_j \neg f_b(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge \texttt{p} \wedge \texttt{norules} \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge \bigwedge_{j \neq i} \neg step_j \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_j$ for all $j \neq i$. (Corresponds to C20.)

A25 $step_i \wedge \neg fail_i \wedge \neg(\texttt{p} \wedge \texttt{norules}) \rightarrow [u_i]\bot$ where u_i is either α_i or $t_i(\phi)$. (Corresponds to condition C21).

A26 $\neg start_i \wedge G\kappa_i \wedge B\beta_i \wedge \texttt{p} \wedge \phi \rightarrow \langle[\delta_{ri}]\rangle(start_i \wedge \bigwedge_j \neg step_j \wedge \phi)$ where ϕ does not contain $start_i$ and $step_j$ for any j. (Corresponds to C22.)

A27 $\neg(G\kappa_i \wedge B\beta_i) \vee \neg start_i \vee \neg \texttt{p} \rightarrow [\delta_{ri}]\bot$ (Corresponds to C22 only-if.)

Let us call the axiom system above $\mathbf{Ax}(\Lambda, \mathbf{am})$.

Theorem 4 $\mathbf{Ax}(\Lambda, \mathbf{am})$ *is sound and (weakly) complete for the class of models* $\mathbf{M}(\Lambda, \mathbf{am})$.

The proof is again similar to the proof of Theorem 1 and exploits close correspondence between conditions on models and axioms.

6 Verifying agent programs

Our aim is to verify properties of the agent such as ‘in all states (or in some state) reachable by a path corresponding to the execution of the agent’s program, property ϕ holds’. In this section we show how to translate the agent’s program into an expression of L which does not depend on the agent’s deliberation strategy but which describes exactly the paths corresponding to the agent’s execution under a given deliberation strategy in the models for this strategy.

The basic building blocks of our translation are expressions of the form $\delta_{r\,i}; f_p(\pi_i); e_i$ which correspond to firing a rule, executing the corresponding plan, and resetting the boolean flags for this plan. Before the agent fires the rule r_i again, it has to finish executing the plan (or the plan has to fail). The agent may also interleave this plan execution with firing other rules and executing the corresponding plans. It may also be that several consecutive executions of $\delta_{r\,i}; f_p(\pi_i); e_i$, that is $(\delta_{r\,i}; f_p(\pi_i); e_i)^+$, may be interleaved with several consecutive executions of $\delta_{r\,j}; f_p(\pi_j); e_j$, that is, $(\delta_{r\,j}; f_p(\pi_j); e_j)^+$. Note that the agent does not have to and probably will not be able to execute all of its rules and plans.

This gives rise to the following translation of the agent program:

$$\xi(\Lambda) = \bigcup_{\Lambda' \subseteq \Lambda, \Lambda' \neq \emptyset} \|_{r_i \in \Lambda'} \; (\delta_{r\,i}; f_p(\pi_i); e_i)^+$$

that is, the interleaving of one or more repetitions of all possible subsets of the agent’s plans.

We are interested in safety and liveness properties of agent programs, namely properties of the form $\phi_0 \rightarrow [\xi(\Lambda)]\phi$ and $\phi_0 \rightarrow \langle \xi(\Lambda) \rangle \phi$ where ϕ_0 is the description of the initial state and ϕ is the property of interest (such as achievement of a goal). To prove properties of the agent program under a particular deliberation strategy we need to show that the property is derivable from the corresponding axioms. For example, to show that an agent with program Λ, initial belief p and goal q is guaranteed to achieve its goal under the interleaved deliberation strategy, we need to derive $Bp \wedge Gq \wedge init \rightarrow [\xi(\Lambda)]Bq$ from $\mathbf{Ax}(\Lambda, \mathbf{i})$ (where $init = \bigwedge_{r_i \in \Lambda}(\neg start_i \wedge \neg fail_i \wedge \neg step_i)$ describes the initial configuration).

To prove such properties, we must ensure that there is a correspondence between paths in the operational semantics plus a deliberation strategy and paths in the PDL models satisfying the axioms for this strategy. If a path exists in the operational semantics, then there is a corresponding path in the PDL model. Note that the converse is not true; for example, in the PDL model from any state there is a transition by a belief update action, and in the operational semantics this only holds if the belief update is the first action of some plan which is in the plan base in that state. However, we can prove that if a there is a path in the PDL model which is described by $\xi(\Lambda)$, then there is a corresponding path in the operational semantics.

Before we state the theorems precisely, we need to introduce some definitions. For each deliberation strategy, we define what it means for configurations of an agent and states in the models of the logic to correspond to each other. First we define this correspondence for the **(i)** deliberation strategy. Given a configuration $c = \langle \sigma, \gamma, \Pi = \{r_1 : \pi'_1, \ldots, r_n : \pi'_n\} \rangle$, a state s is in the correspondence relation $\sim_{(\mathtt{i})}$ to c, $s \sim_{(\mathtt{i})} c$, if:

- $V_b(s) = \sigma$, $V_g(s) = \gamma$ (beliefs and goals are the same in c and s),
- $start_i \in V_c(s)$ iff $r_i : \pi \in \Pi$ ($start_i$ means that a plan has been added to the plan base by r_i)
- $fail_i \notin V_c(s)$ for any $r_i \in \Lambda$ (only the states where $fail_i$ is false for all plans correspond to ‘real’ configurations).

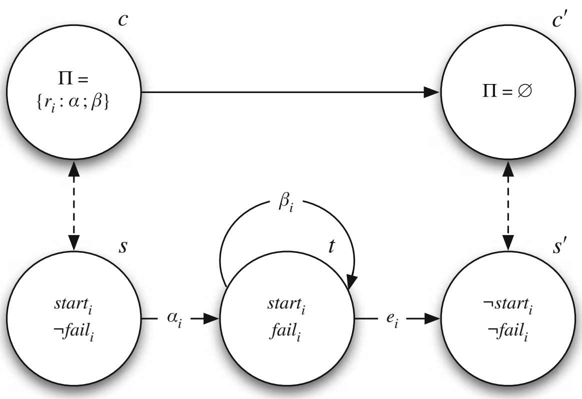

Fig. 1 Correspondence between operational semantics and models (α not executable)

By a path in an operational semantics transition system S, we will mean a sequence of configurations $c_1, label_1, c_2, label_2, \ldots, c_m$ where c_{j+1} is obtained from c_j by one of the transition rules (1a)–(4). For convenience, we label each transition by the corresponding operation; a (1a) transition executing an update action α by 'execute α in π_i', a (1b) transition by 'fail α in π_i', a (2a) transition by 'test if ϕ in π_i', a (2b) transition by 'test if $\neg\phi$ in π_i', similarly for (3a) and (3b), and a (4) transition of firing a rule r_i by 'fire r_i'. We claim that if there is a path $c = c_1, \ldots, c_n = c'$ in S with a certain sequence of labels, then there is a corresponding path $s = s_1, \ldots, s_k = s'$ in M such that $s \sim_{(\mathtt{i})} c$ and $s' \sim_{(\mathtt{i})} c'$. It remains to define what we mean by a 'corresponding path'. For each single step $c_j, label_j, c_{j+1}$ on a path in S, the corresponding path in M is as follows:

(1a): c_j, 'execute α in π_i', c_{j+1}: the corresponding path is s_j, t, s_{j+i} or s_j, s_{j+1} depending on whether α is the last action in π_i, where $s_j \sim_{(\mathtt{i})} c_j$ and $s_{j+1} \sim_{(\mathtt{i})} c_{j+1}$. If α is the last action in π_i, then $R_{\alpha_i}(s_j, t)$ and $R_{e_i}(t, s_{j+1})$. If α is not the last action, then $R_{\alpha_i}(s_j, s_{j+1})$.

(1b): c_j, 'fail α in π_i', c_{j+1}: the corresponding path is $s_j, t, \ldots, t, s_{j+1}$ where $s_j \sim_{(\mathtt{i})} c_j$, $s_{j+1} \sim_{(\mathtt{i})} c_{j+1}$, $R_{\alpha_i}(s_j, t)$, t satisfies $fail_i$ and has otherwise the same assignments as s_j, and $R_{e_i}(t, s_{j+1})$. Intuitively, the path contains as many t loops as there are update actions remaining in the plan when it failed, and the last step on the path is along the e_i action which resets the $start_i$ and $fail_i$ flags to false and leaves the rest of the assignments the same. Figure 1 illustrates this point; we assume that action α in a plan generated by a PG rule r_i is not executable, so in the operational semantics the plan is removed from the plan base, while in the model the rest of the plan is 'consumed' in a state where $fail_i$ flag is set to true. The e_i transition resets the $start_i$ and $fail_i$ flags to false.

(2a)–(3b): the corresponding path is s_j, s_{j+1} where $s_j \sim_{(\mathtt{i})} c_j, s_{j+1} \sim_{(\mathtt{i})} c_{j+1}$ and $s_{j+1} = s_j$.

(4): the corresponding path is s_j, s_{j+1} where $s_j \sim_{(\mathtt{i})} c_j$, $s_{j+1} \sim_{(\mathtt{i})} c_{j+1}$ and $R_{\delta_{r_i}}(s_j, s_{j+1})$.

Theorem 5 *Let Λ be the program of an agent using the* **(i)** *deliberation strategy. Let c_0 be an initial configuration in a operational semantics transition system S for this agent. Let $M \in \mathbf{M}(\Lambda, \mathbf{i})$ be generated by $s_0 \sim_{(\mathtt{i})} c_0$. There exists a path from c_0 to c in S, if and only if, there is a path in M described by $\xi(\Lambda)$ from s_0 to a state $s \sim_{(\mathtt{i})} c$.*

To prove the theorem, we need the following two lemmas. S and M in the lemmas refer to S and M in Theorem 5.

Lemma 1 *For any two configurations $c = \langle\sigma, \gamma, \Pi\rangle$ and $c' = \langle\sigma', \gamma', \Pi'\rangle$ in S, if there is a path between them in S, then there is a corresponding path in M between a state $s \sim_{(\mathtt{i})} c$ and a state $s' \sim_{(\mathtt{i})} c'$.*

Proof By induction on the number of labels in the path in S, using the preconditions of the transitions of the operational semantics, the definition of the deliberation strategy cycle, and conditions on $\mathbf{M}(\Lambda, \mathbf{i})$. We show that for every configuration c, the set of transitions possible in c is included in the set of transitions possible in a state $s \sim_{(\mathtt{i})} c$, and moreover the configurations reachable from c are in the relation $\sim_{(\mathtt{i})}$ with the states reachable by the corresponding transitions from s.

Under the interleaved execution strategy, the possible transitions from $c = \langle\sigma, \gamma, \Pi = \{r_1 : \pi'_1, \ldots, r_n : \pi'_n\}\rangle$ are (1a)–(4), namely the agent can fire an applicable rule r_i which is not in $\{r_1, \ldots, r_n\}$, or apply transition rules (1a)–(3) with respect to one of its plans $\{\pi'_1, \ldots, \pi'_n\}$. Let $s \sim_{(\mathtt{i})} c$.

(1a): if some plan in Π is of the form $r_i : \alpha; \pi$ and $\sigma \models_{cwa} \mathtt{prec}_j(\alpha)$, then there is a transition to c' where the belief base is $\sigma' = T_j(\alpha, \sigma)$, the goal base is the same apart from removing goals which became true, and instead of $r_i : \alpha; \pi$ the plan base contains $r_i : \pi$. By the condition C5, $R_{\alpha_i}(s, s')$ where $V_b(s') = T_j(\alpha, V_b(s))$, $V_g(s') = V_g \backslash (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ and control flags do not change. In other words, $s' \sim_{(\mathtt{i})} c'$.

(1b): if some plan in Π is of the form $r_i : \alpha; \pi$ and none of the preconditions of α holds in σ, there is a transition to c' with the same belief and goal base but the plan base $\Pi' = \Pi \backslash \{r_i : \alpha; \pi\}$. By C6, $R_{\alpha_i}(s, t)$ where t has the same beliefs and goals but satisfies $fail_i$. By C3, $R_{e_i}(t, s')$ where s' has the same beliefs and goals, but $fail_i$ is false and $start_i$ is false. So, $s' \sim_{(\mathtt{i})} c'$.

(2a), (2b), (3a), (3b): if ϕ is true in c, then $f_b(\phi)$ is true in s, so $R_{t_i(\phi)}(s, s)$ and $s \sim_{(\mathtt{i})} c'$; otherwise $f_b(\neg\phi)$ is true in c, and $R_{t_i(\neg\phi)}(s, s)$ and $s \sim_{(\mathtt{i})} c'$.

(4): if there is some rule r_i which is not in $\{r_1, \ldots, r_n\}$, and its belief and goal conditions β_i and κ_i hold in c, then there is a reachable configuration c' which has the same belief and goal base, and contains the plan $r_i : \pi_i$ in its plan base. Then by condition C4, $R_{\delta_{ri}}(s, s')$ where beliefs and goals are the same as in s and $start_i$ is set to true. Therefore, $s' \sim_{(\mathtt{i})} c'$. □

Lemma 2 *For every pair of states s and s′ in M, which have a corresponding configuration with an empty plan base in S, there exists a path between s and s′ described by $\xi(\Lambda)$ iff there is a corresponding path between c and c′, where $s \sim_{(\mathtt{i})} c$ and $s' \sim_{(\mathtt{i})} c'$.*

Proof The ‘only if’ direction is easy to show by an argument similar to the previous lemma. For the ‘if’ direction, assume that there is a path between s and s' which is described by $\xi(\Lambda)$. We want to show that a corresponding path exists between c and c'. Imagine that we have two markers, one for states in M and another for configurations in S. The first marker starts at s and the second at c. We move the first marker along the path in M, sometimes several steps at a time, and the second marker along the corresponding transition in S, so that when the markers are on s_j and c_j, $s_j \sim_{(\mathtt{i})} c_j$. If such a move is always possible, we will find a corresponding path in S, because by the time the first marker reaches s', the second one is on c' such that $s' \sim_{(\mathtt{i})} c'$. Since the path in M is fixed, we always know what is the next move in M and hence what should be the answering move in S. The existence of the corresponding transition in S follows from the fact that the conditions enabling a transition in M match exactly the conditions for corresponding configurations in S, given the history of the corresponding configuration. For example, if the next transition from s_j is α_i, this means that earlier on the path there was an δ_{ri} transition, followed by transitions corresponding to the statements preceding α in π_i (this is because the path is described by $\xi(\Lambda)$). So we can assume that c_j has $r_i : \alpha; \pi'_i$ in its plan base, and the preconditions of α hold in c_j; the first marker moves to a state s_{j+1} such that $R_{\alpha_i}(s_j, s_{j+1})$ and the second marker to a configuration c_{j+1} such that $s_{j+1} \sim_{(\texttt{i})} c_{j+1}$ where the plan base contains $r_i : \pi'_i$. □

Theorem 5 follows immediately from the two lemmas. Correspondence proofs for other deliberation strategies are similar.

Theorem 6 *Let Λ be the program of an agent using the **(ni)** deliberation strategy. Let S be the transition system generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\Lambda, \mathbf{ni})$ be generated by $s_0 \sim_{(\texttt{i})} c_0$. There exists a path from c_0 to c if, and only if, in M there is a path described by $\xi(\Lambda)$ from s_0 to a state $s \sim_{(\texttt{i})} c$.*

The proof is similar to the proof of Theorem 5 but uses the restrictions imposed by **(ni)** strategy on transition rules, and corresponding conditions on $\mathbf{M}(\Lambda, \mathbf{ni})$.

Correspondence for the alternating strategies needs to take into account the $step_i$ flags. We define $s \sim_{(\texttt{as})} c$ to hold if $s \sim_{(\texttt{i})} c$ and in addition the following condition holds:

- $step_i \in V_c(s)$ iff this configuration has been obtained by transition (1a) (executing a belief update for plan π_i).

Theorem 7 *Let Λ be the program of an agent using the **(as)** deliberation strategy. Let S be the transition system generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\Lambda, \mathbf{as})$ be generated by $s_0 \sim_{(\texttt{as})} c_0$. There exists a path from c_0 to c if, and only if, in M there is a path described by $\xi(\Lambda)$ from s_0 to a state $s \sim_{(\texttt{as})} c$.*

The proof is similar to the proof of Theorem 5 but uses the restrictions imposed by **(as)** strategy on transition rules, and corresponding conditions on $\mathbf{M}(\Lambda, \mathbf{as})$.

Correspondence between states and configurations for **(am)** is defined as: $s \sim_{(\texttt{am})} c$ if $s \sim_{(\texttt{i})} c$ and in addition the following condition holds:

- $step_i \in V_c(s)$ iff on the path leading to c, since the last execution of a planning rule, a belief update in π_i was executed.

Theorem 8 *Let Λ be the program of an agent using the **(am)** deliberation strategy. Let S be the transition system generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\Lambda, \mathbf{am})$ be generated by $s_0 \sim_{(\texttt{am})} c_0$. There exists a path from c_0 to c if, and only if, in M there is a path described by $\xi(\Lambda)$ to a state $s \sim_{(\texttt{am})} c$.*

The proof is similar to the proof of Theorem 5 but uses the restrictions imposed by **(am)** strategy on transition rules, and corresponding conditions on $\mathbf{M}(\Lambda, \mathbf{am})$.

6.1 Complexity of the verification problem

Given an agent program Λ, the size of its translation $\xi(\Lambda)$ is linear in Λ. The axiomatisation of any execution strategy involves schema axioms such as A5 which hold for arbitrary formulas ϕ (essentially such axioms serve as frame axioms and state that if ϕ was true before an action was executed, and ϕ does not mention any effects of the action, then ϕ remains true). A naive automatic generation of all possible frame axioms is exponential in the number of literals in Λ (since it has to talk about all possible states). Therefore, the set of axioms $Ax(\Lambda)$ required for proving properties of Λ may be exponential in the size of Λ. Once we have $Ax(\Lambda)$, the problem of checking whether it entails a property ϕ of Λ has the same complexity as satisfiability problem for PDL with interleaving [19]: double exponential in the size of Λ and ϕ. In other words, the problem is decidable but triple exponential in the size of Λ.

6.2 Example

In this section, we briefly illustrate how to prove properties of agents in our logic using the running example.

Let us abbreviate `home` as h, `work` as o (for “office”), `breakfast` as b, `raining` as r, `take_umbrella` as u, `take_sunglasses` as s, `walk_work` as w, and `eat_` $athttbreakfast$ as t.

The translation of the agent’s program $\Lambda = \{r_1; r_2\}$ is

$$\begin{aligned}&\xi(\Lambda) = \\ &\quad (\delta_{r1}; ((t_1(r); u_1) \cup (t_1(\neg r); s_1)); w_1; e_1)^+ \cup \\ &\quad (\delta_{r2}; t_2; e_2)^+ \cup \\ &\quad ((\delta_{r1}; ((t_1(r); u_1) \cup (t_1(\neg r); s_1)); w_1; e_1)^+ \parallel (\delta_{r2}; t_2; e_2)^+)\end{aligned}$$

The expression $\xi(\Lambda)$ has an equivalent interleaving-free form which can be generated automatically, and we can use a PDL theorem prover such as PDL- TABLEAU [22] to automatically verify properties of the agent program. For example, the agent is guaranteed to achieve both its goals under the **(am)** strategy. Namely, the following formula:

$$init \wedge Bh \wedge Br \wedge Gb \wedge Go \rightarrow \langle[\xi(\Lambda)]\rangle(Bb \wedge Bo)$$

where $init = \bigwedge_{i=1,2}(\neg start_i \wedge \neg fail_i \wedge \neg step_i)$, is derivable in $\mathbf{Ax}(\Lambda, \mathbf{am})$ from the axioms such as the following instances of A26, A21, A5, A25:

$\neg start_1 \wedge \neg start_2 \wedge Go \wedge Gb \wedge Bh \wedge Br \rightarrow \langle[\delta_{r1}]\rangle(start_1 \wedge \neg start_2 \wedge \neg step_1 \wedge Go \wedge Gb \wedge Bh \wedge Br)$
$start_1 \wedge \neg start_2 \wedge \neg step_1 \wedge Go \wedge Gb \wedge Bh \wedge Br \rightarrow \langle[t_1(r)]\rangle(start_1 \wedge \neg start_2 \wedge step_1 \wedge Go \wedge Gb \wedge Bh \wedge Br)$
$start_1 \wedge \neg start_2 \wedge \neg step_1 \wedge Go \wedge Gb \wedge Bh \wedge Br \rightarrow [t_1(\neg r)]\bot$
$start_1 \wedge \neg start_2 \wedge step_1 \wedge Go \wedge Gb \wedge Bh \wedge Br \rightarrow [u_1]\bot$

Under other strategies, the agent is not guaranteed to achieve both its goals. As a simple counterexample, consider the following path which is possible from the start state under **(i)** and **(ni)**: $\delta_{r1}; t_1(r); u_1; w_1; e_1$. In the state reachable by this path, δ_{r2} cannot be applied since its belief condition h fails. Therefore, from that state it is impossible to reach a state where Bb is true by following $\delta_{r2}; t_2; e_2$. Similarly, for **(as)**, a sequence $\delta_{r1}; t_1(r); u_1; \delta_{r2}; w_1; t_2$ does not reach the goal, because w_1 destroys the preconditions of t_2, so although, there are states reachable by this sequence under **(as)**, execution of t_2 fails and does not make its postcondition true.

7 Related work

There has been a considerable amount of work on verifying properties of agent programs implemented in other agent programming languages such as ConGolog, MetateM, 3APL, 2APL, and AgentSpeak. Shapiro et al. in [23] describe CASLve, a framework for verifying properties of agents implemented in ConGolog. CASLve is based on the higher-order theorem prover PVS and has been used to prove, e.g., termination of bounded-loop ConGolog programs. However, its flexibility means that verification requires user interaction in the form of proof strategies. Properties of agents implemented in programming languages based on executable temporal logics such as MetateM [14], can also easily be automatically verified.

However these languages are quite different from languages like SimpleAPL, in that the agent program is specified in terms of temporal relations between states rather than branching and looping constructs. Other related attempts to bridge the gap between agent programs such as 3APL and 2APL on the one hand and verification logics on the other, e.g., [13,17], have yet to result in an automated verification procedure.

In [20] Mulder et al. present a model of the execution of PRS in an executable temporal logic, MML. Agent plans are represented as temporal formulas and deliberation strategies are represented by sets of MML rules. The rules define the behaviour of a meta-interpreter operating on terms which are names for temporal formulas. The MML model allows the direct specification and verification (via execution in concurrent MetateM) of agent properties.

There has also been considerable work on the automated verification of multi-agent systems using model-checking [4,18]. For example, in [7], Bordini et al. describe work on verifying programs written in Jason, an extension of AgentSpeak(L). In this approach, agent programs together with the semantics of Jason are translated into either Promela or Java, and verified using Spin or JPF model checkers respectively. There has also been work on using model checking techniques to verify agent programming languages similar to SimpleAPL [3,24]. In this approach agent programs and execution strategies are encoded directly into the Maude term rewriting language, allowing the use of the Maude LTL model checking tool to verify temporal properties describing the behaviour of agent programs.

The work reported here is closely related to our previous work on using theorem proving techniques to verify agent deliberation strategies [2]. However in that work, different execution strategies were specified using different PDL program expressions, rather than in terms of a fixed general execution strategy which is constrained by the execution model to obtain different execution strategies, as in this paper.

8 Conclusion

In this paper we analysed the implications of an agent's deliberation strategy in determining the behaviour of BDI-based agent programs. In order to illustrate the problem, we presented a simple agent programming language, SimpleAPL, and explored some of its possible deliberation strategies. We proposed a family of logics to reason about deliberation strategies of SimpleAPL programs and showed how these can be used to verify the correctness of agent programs. Using a simple example program, we illustrated how the choice of deliberation strategy can determine whether a given program will achieve a particular goal. Although we investigated only a small number of deliberation strategies, our approach of associating propositions with phases in the agent's deliberation cycle and using these transitions to axiomatise the possible transitions between phases is general enough to accommodate any deliberation strategy that can be formulated in terms of distinct phases of execution and the kinds of operations that can be performed in each phase. The axiomatisations share significant structure, concisely characterising the similarities and differences between strategies, and facilitating the formalisation of new strategies.

In future work we plan to investigate other deliberation strategies. For example, it would also be interesting to investigate strategies which prioritise particular goals and the plans that achieve them. Another direction for future work is extending the programming language, e.g., to introduce variables in the language of beliefs, goals, plans and planning goal rules, and to extend the setting to include additional phases in the agent's cycle, such as events or sensing, and actions performed in an external environment.

Acknowledgements Natasha Alechina and Brian Logan were supported by the Engineering and Physical Sciences Research Council [grant number EP/E031226]. We would like to thank Fahad Khan and the anonymous reviewers for useful comments and suggestions.

References

1. Abrahamson, K. R. (1980). *Decidability and expressiveness of logics of processes*. PhD thesis, Department of Computer Science, University of Washington.
2. Alechina, N., Dastani, M., Logan, B., & Meyer, J.-J. Ch. (2007). A logic of agent programs. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007)* (pp. 795–800). AAAI Press.
3. Astefanoaei, L., Dastani, M., de Boer, F. S., & Meyer, J.-J. Ch. (2008). A verification framework for normative multi-agent systems. In *The Proceedings of the 11th Pacific Rim International Conference on Multi-Agents (PRIMA 2008)*. LNCS (Vol. 5357). Springer.
4. Benerecetti, M., Giunchiglia, F., & Serafini, L. (1998). Model checking multiagent systems. *Journal of Logic and Computation, 8*(3), 401–423.
5. Blackburn, P., de Rijke, M., & Venema, Y. (2001). *Modal logic*. Cambridge Tracts in Theoretical Computer Science (Vol. 53). Cambridge, UK: Cambridge University Press.
6. Bordini, R. H., Dastani, M., Dix, J., & Seghrouchni, A. E. F (2005). *Multi-agent programming—languages, platforms and applications*. Berlin: Springer.
7. Bordini, R. H., Fisher, M., Visser, W., & Wooldridge, M. (2006). Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems, 12*(2), 239–256.
8. Bordini, R. H., Hübner, J. F., & Vieira, R. (2005). Jason and the golden fleece of agent-oriented programming. In R. H. Bordini, M. Dastani, J. Dix, & A. E. F. Seghrouchni (Eds.), *Multi-agent programming—languages, platforms and applications.* Heidelberg: Springer.
9. Dastani, M. (2008). 2APL: A practical agent programming language. *International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 16*(3), 214–248, Special Issue on Computational Logic-based Agents. F. Toni, & J. Bentahar (Eds.).
10. Dastani, M., & Meyer, J. J. Ch. (2007). A practical agent programming language. In *The Proceedings of the Fifth International Workshop on Programming Multi-agent Systems (ProMAS'07).* LNAI (Vol. 4908, pp. 107–123). Springer.
11. Dastani, M., van Riemsdijk, M. B., Dignum, F., & Meyer, J.-J. Ch. (2004). A programming language for cognitive agents: Goal directed 3APL. In *Proceedings of ProMAS 2003*. LNCS (Vol. 3067, pp. 111–130). Springer.
12. Dastani, M., van Riemsdijk, M. B., & Meyer, J.-J. Ch. (2005). Programming multi-agent systems in 3APL. In R. H. Bordini, M. Dastani, J. Dix, A. E. F. Seghrouchni (Eds.), *Multi-agent programming—languages, platforms and applications*. Berlin: Springer.
13. Dastani, M., van Riemsdijk B., & Meyer J.-J. Ch. (2007). A grounded specification language for agent programs. In *The Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'07)* (pp. 578–585). ACM Press.
14. Fisher, M. (2006). Metate M: The story so far. In *The Proceedings of the Third International Workshop on Programming Multi-agent Systems (ProMAS'05).* LNAI (Vol. 3862, pp. 3–22). Springer.
15. Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87* (pp. 677–682).
16. Harel, D., Kozen, D., & Tiuryn, J. (2000). *Dynamic logic*. Cambridge: MIT Press.
17. Hindriks K. V., & Meyer J.-J. Ch. (2007). *Agent logics as program logics: Grounding KARO*. In *Proceedings of the 29th German Conference on AI (KI 2006).* LNAI (Vol. 4314). Heidelberg: Springer.
18. Lomuscio, A., & Raimondi, F. (2006). MCMAS: A tool for verifying multi-agent systems. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS06).* LNCS (Vol. 3920, pp. 450–454). Vienna, Austria: Springer.
19. Mayer, A. J., & Stockmeyer, L. J. (1996). The complexity of PDL with interleaving. *Theoretical Computer Science, 161*(1&2), 109–122.
20. Mulder, M., Treur, J., & Fisher, M. (1997). Agent modelling in METATEM and DESIRE. In M. P.Singh, A. S. Rao, & M. Wooldridge (Eds.), *Intelligent agents IV, agent theories, architectures, and languages, 4th international workshop (ATAL'97).* Lecture Notes in Computer Science (Vol. 1365, pp. 193–207). Springer.

21. Pokahr, A., Braubach, L., & Lamersdorf, W. (2005). Jadex: A BDI reasoning engine. In: R. H. Bordini, M. Dastani, J. Dix, & A. E. F. Seghrouchni (Eds.), *Multi-agent programming—languages, platforms and applications*. Heidelberg: Springer.
22. Schmidt, R. A. (2003). pdl-tableau. http://www.cs.man.ac.uk/~schmidt/pdltableau.
23. Shapiro, S., Lespérance, Y., & Levesque, H. J. (2002). The cognitive agents specification language and verification environment for multiagent systems. In *The Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)* (pp. 19–26). ACM Press.
24. van Riemsdijk, M. B., de Boer, F. S., Dastani, Mehdi., & Meyer J.-J. Ch. (2006). Prototyping 3APL in the Maude term rewriting language. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)* (pp. 1279–1281). New York, NY, USA: ACM.