Reasoning About Agent Deliberation

N. Alechina[1] **M. Dastani**[2]
[1] School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
{nza,bsl}@cs.nott.ac.uk

B. S. Logan[1] **J.-J. Ch. Meyer**[2]
[2] Deparment of Information and Computing Sciences
Universiteit Utrecht
3584CH Utrecht, The Netherlands
{mehdi,jj}@cs.uu.nl

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 axiomatize different deliberation strategies for SimpleAPL programs, and, for each strategy we consider, 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.

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 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 (Bordini *et al.* 2005) the deliberation strategy forms integral part of their semantics, e.g., Jason (Bordini, Hübner, & Vieira 2005). However most agent platforms provide customization mechanisms that allow the agent developers to influence aspects of the deliberation strategy, for example, 3APL (Dastani, van Riemsdijk, & Meyer 2005), 2APL (Dastani & Meyer 2007) and Jadex (Pokahr, Braubach, & Lamersdorf 2005). In some cases, a BDI-based agent programming language gives the agent developer complete control of the deliberation strategy. For example, PRS (Georgeff & Lansky 1987) 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 behavior, or varying the number of goals an agent will attend to simultaneously.

As an agent's behavior 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., (Hindriks & Meyer 2007; Benerecetti, Giunchiglia, & Serafini 1998; Bordini *et al.* 2006; Lomuscio & Raimondi 2006; Shapiro, Lespérance, & Levesque 2002), there has been much less work on deliberation strategies. An exception is the work of (Mulder, Treur, & Fisher 1997) 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. We are not aware of any published results on the complexity of MML, however it is likely to be high.

In this paper 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. We consider deliberation strategies in the context of a simple APL-like (Dastani *et al.* 2004; Bordini *et al.* 2005) agent programming language, SimpleAPL introduced in (Alechina *et al.* 2007). We sketch the syntax of SimpleAPL, give its operational semantics, and define various 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 axiomatizations 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., (Alechina *et al.* 2007) where two basic deliberation strategies, interleaved and non-interleaved, were 'hard-coded' into the translation of an agent program, the approach presented here uses a single fixed translation of the agent program together with an axiomatization of the agent's deliberation strategy. As an example, we axiomatize four deliberation strategies. 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 of execution 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.

SimpleAPL

SimpleAPL is a fragment of the agent-oriented programming language 3APL (Dastani *et al.* 2004; Bordini *et al.* 2005). 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 (Alechina *et al.* 2007) 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 realize (not necessary all at once). For simplicity, we only allow the agent's beliefs and goals to be literals. 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 set of agent's beliefs and the set of goals are disjoint.

Belief update actions change the beliefs of the agent. A belief update action is specified in terms of its pre- and post-conditions (which are sets of literals), and can be executed if the belief literals in one of its preconditions are in the agent's current set of beliefs. Executing the action adds the belief literals in the action's postcondition to the agent's beliefs. 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". Belief update actions maintain consistency of the agent's beliefs, i.e., if p is in the belief set and its negation $\neg p$ is added as a postcondition of an action, p is replaced by $\neg p$. 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 in a straightforward way by including the state of the environment in the models.

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

```
π = 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 characterizing situation(s) in which it could be a good idea to adopt the plan, and the body of the rule. Firing 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 | π
```

states that "if the agent's goal is to be at work and it is at home, then it will adopt the plan π". 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 in EBNF notation can be found in (Alechina *et al.* 2007).

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

Each *belief update action* α has a set of preconditions $\mathtt{prec_1}(\alpha)$, ..., $\mathtt{prec_k}(\alpha)$. Each $\mathtt{prec_i}(\alpha)$ is a finite set of belief literals, and any two preconditions for an action α, $\mathtt{prec_i}(\alpha)$ and $\mathtt{prec_j}(\alpha)$ $(i \neq j)$, are mutually exclusive (both sets of propositional variables cannot be satisfied simultaneously). For each $\mathtt{prec_i}(\alpha)$ there is a unique corresponding postcondition $\mathtt{post_i}(\alpha)$, which is also a finite set of literals. A belief update action α can be executed if $\mathtt{prec_j}(\alpha) \subseteq \sigma$ for some precondition j, and the effect of updating σ with α in the resulting configuration is given by $T_j(\alpha, \sigma) = \sigma \cup \mathtt{post_j}(\alpha) \setminus (\{p : \neg p \in \mathtt{post_j}(\alpha)\} \cup \{\neg p : p \in \mathtt{post_j}(\alpha)\})$, i.e., executing the belief update action α adds the literals in its postcondition to the agent's beliefs and removes any existing beliefs which are inconsistent with the postcondition.

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 then:

$$(1a) \quad \frac{r_i : \alpha; \pi \in \Pi \quad \mathtt{prec}_j(\alpha) \subseteq \sigma \quad T_j(\alpha, \sigma) = \sigma'}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma', \gamma', (\Pi \setminus \{r_i : \alpha; \pi\}) \cup \{r_i : \pi\} \rangle}$$

where $\gamma' = \gamma \setminus \{\phi \in \gamma \mid \phi \in \sigma'\}$ (executing a belief update action causes the agent to drop any goals it believes to be achieved as a result of the update). 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:

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

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

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

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

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

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

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

where $\Pi' = \Pi \setminus \{r_i : (\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1); \pi\}$. 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 provided that 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.

$$(4) \quad \frac{\gamma \models \kappa_i \quad \sigma \models \beta_i \quad r_i : \pi \notin \Pi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \cup \{r_i : \pi_i\} \rangle}$$

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 nondeterministically, apply it, execute the resulting plan; repeat".

Many deliberation strategies are possible and we do not have space to consider them all in detail. Instead we characterize 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 **(am)** strategy allows 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 it 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:

`choose_rule(`Λ`, condition)` returns a planning goal rule r_i from Λ which satisfies *condition*; if no rule satisfies *condition*, returns an arbitrary rule from Λ

`apply(`Π`, `r_i`)` if the conditions of transition rule (4) are satisfied for planning goal rule r_i, returns $\Pi \cup \{r_i : \pi_i\}$, where π_i is the plan produced by r_i; otherwise returns Π

`choose_plan(`Π`, condition)` returns a plan π_i from Π which satisfies *condition*; if no plan satisfies *condition*, returns an arbitrary plan from Π

`next(`Π`, `π_i`)` if the appropriate conditions of transition rules (1a)–(3b) are satisfied, executes the next action in $\pi_i \in \Pi$ (and any preceding if and while tests), updates the configuration accordingly and returns the updated plan base; otherwise returns $\Pi \setminus \{\pi_i\}$.

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

```
repeat
  r_i = choose_rule(Λ, applicable)
  Π = apply(Π, r_i)
  π_i = choose_plan(Π, true)
  while (Π != {})
    Π = next(Π, π_i)
```

The *applicable* condition of the `choose_rule` control action is true for a planning goal rule r_i if the belief and goal conditions of r_i are true in the current configuration and no plan associated with r_i is in the plan base (i.e., it mirrors the conditions of transition rule (4)), and has the effect of causing the agent to adopt plans which are relevant to its current beliefs and goals.

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

```
repeat
  r_i = choose_rule(Λ, applicable)
  Π = apply(Π, r_i)
  π_i = choose_plan(Π, true)
  Π = next(Π, π_i)
```

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

```
repeat
  r_i = choose_rule(Λ, applicable)
  Π = apply(Π, r_i)
  foreach π_i in Π
    Π = next(Π, π_i)
```

Other strategies can be defined in a similar way. For example, by changing the *true* condition of the `choose_plan` control action to be *executable*, we can delay discarding plans which are currently not executable (i.e., where the next action α in the plan would fail if executed in the current context) in the hope that an action in an executable plan will make the preconditions of α true.

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 PDL (see, e.g., (Harel, Kozen, & Tiuryn 2000)). Standard PDL is a logic to reason about programs. Its 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, tests on formulas '?' ($\phi?$ is executable in a state if ϕ is true in that state), 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.

We extend the standard language of PDL with belief and goal operators, and an interleaving program constructor $\|$ (Abrahamson 1980), where $\rho_1 \parallel \rho_2$ means interleave executing actions of ρ_1 with executing actions of ρ_2.[1] We define the language of our logic relative to an agent program (set of rules) Λ with a given set of 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 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, $C(\Lambda) = \{C(\alpha) : \alpha \in Ac(\Lambda)\}$.

[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 (Abrahamson 1980).

We represent 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 (basic action) of π_i has been executed; and $fail_i$, which indicates that π_i has failed.

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 δ_{ri} 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 each plan π_i, an atomic action e_i. This action is introduced for technical reasons, and will be used in our translation of the agent program Λ as a PDL expression. We append it after each plan to reset the control flags after the plan has finished executing.

The language L is the language of PDL with interleaving, extended with a belief operator B and a goal operator G. A formula of L is defined as follows: if $p \in P$, then $B(\text{–})p$ and $G(\text{–})p$ are formulas; 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 $[\rho]\phi \wedge \langle\rho\rangle\phi$.

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)
  - $f_b((\text{–})p) = B(\text{–})p$
  - $f_b(\phi \texttt{ and } \psi) = f_b(\phi) \wedge f_b(\psi)$
  - $f_b(\phi \texttt{ or } \psi) = f_b(\phi) \vee f_b(\psi)$
- translation of goal formulas: analogous to beliefs, with ϕ, ψ replaced by goal query expressions, B replaced by G and f_b replaced by f_g
- translation of plan expressions: let α_i be a belief update action, ϕ and ψ be belief and goal query expressions, and π, π_1, π_2 be plan expressions of SimpleAPL
  - $f_p(\alpha_i) = \alpha_i$
  - $f_p(\pi_1; \pi_2) = f_p(\pi_1); f_p(\pi_2)$
  - $f_p(\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2) = (f_b(\phi)?; f_p(\pi_1)) \cup (\neg f_b(\phi)?; f_p(\pi_2))$
  - $f_p(\texttt{while } \phi \texttt{ do } \pi) = (f_b(\phi)?; f_p(\pi))^*; \neg f_b(\phi)?$

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

Conditions on models

A model $M = (W, \{R_{\alpha_i} : \alpha_i \in Ac_{ind}(\Lambda)\}, \{R_{\delta_{r_i}} : r_i \in \Lambda\}, R_{e_i}, V)$ 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 $s \in W$,
  $V_b(s) = \{(\text{–})p_1, \ldots, (\text{–})p_m : p_i \in P\}$ is a set of agent's beliefs in s (note that V_b assigns literals rather than propositional variables)
  $V_g(s) = \{(\text{–})u_1, \ldots, (\text{–})u_n : u_i \in P\}$ is a set of agent's goals in s
  $V_c(s) \subseteq P_c$ is a set of control variables true in s
- R_{α_i}, $R_{\delta_{r_i}}$, R_{e_i} are binary relations on W; R_{α_i} correspond to belief updates, $R_{\delta_{r_i}}$ to firing a rule, and R_{e_i} corresponding to executing the e_i action.

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

Given the relations corresponding to basic actions 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:

- $\tau(\alpha_i) = \{(s, s') : R_{\alpha_i}(s, s')\}$
- $\tau(\phi?) = \{(s, s) : M, s \models \phi\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1; \rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, 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 $\tau(\rho_2)$.

In order to be able to define all possible interleavings of paths, we allow 'illegal paths' of the form $(s_0, s_1), (s_2, s_3)$, 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. To see why this is necessary, 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 (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 relation $\models$ of a formula being true in a state of a model is defined inductively as follows:

- $M, s \models B(\text{–})p$ iff $(\text{–})p \in V_b(s)$, where $p \in P_b$
- $M, s \models G(\text{–})p$ iff $(\text{–})p \in V_g(s)$, where $p \in P_g$
- $M, s \models p$ 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 $start_i$ flag to signal that plan π_i has started executing; it is set to true when the planning goal rule r_i is fired and prevents repeated firing of r_i. 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 $start_i$ and $fail_i$ flags to false.

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

C1 $V_b(s) \cap V_g(s) = \emptyset$ and $V_b(s)$ is consistent, i.e., for no $p \in P_b$ both p and $\neg p \in V_b(s)$.

(Beliefs and goals are disjoint and beliefs are consistent.)

C2 If $fail_i \in V_c(s)$, then for every action α_i of plan π_i, $R_{\alpha_i}(s,s)$ and for no $s' \neq s$, $R_{\alpha_i}(s,s')$.

(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 $V_c(s') = V_c(s) \setminus \{start_i, fail_i\}$.

(e_i sets $start_i$ and $fail_i$ to false.)

In addition, different strategies require different conditions on applicability of actions and rules.

Conditions on models for (i)

Models corresponding to the **(i)** strategy in addition conform to the following constraints.

C4 If $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ 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 $V_c(s') = V_c(s) \cup \{start_i\}$.

(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.)

C5 If $f_b(\mathtt{prec}_j(\alpha))$ and $\neg fail_i$ are true in s, then $R_{\alpha_i}(s,s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$, $V_g(s') = V_g(s) \setminus V_b(s')$ and $V_c(s') = V_c(s)$.

(Corresponds to transition (1a).)

C6 If $\vee_j f_b(\mathtt{prec}_j(\alpha))$ 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., its preconditions don't hold) transit to a state where $fail_i$ is true.)

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

Axiomatization for (i)

CL classical propositional logic

PDL axioms of PDL (see, e.g., (Harel, Kozen, & Tiuryn 2000)) excluding interleaving since it is expressible

A1 $\neg(Bp \wedge B{-}p)$ (corresponds to C1)

A2 $B(\text{–})p \rightarrow \neg G(\text{–})p$ (corresponds to C1)

A3 $fail_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge \phi)$ where ϕ is any formula (corresponds to C2)

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

A5 $\neg start_i \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge \phi \rightarrow \langle[\delta_{r_i}]\rangle(start_i \wedge \phi)$, where ϕ does not contain $start_i$ (corresponds to C4; ϕ encodes the frame condition that the state does not change apart from setting the $start_i$ flag to true)

A6 $start_i \vee \neg(f_g(\kappa_i) \wedge f_b(\beta_i)) \rightarrow [\delta_{r_i}]\bot$ (corresponds to C4 'only if')

A7 $f_b(\mathtt{prec}_j(\alpha)) \wedge \neg fail_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(f_b(\mathtt{post}_j(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\mathtt{post}_j(\alpha)$ (corresponds to C5)

A8 $\bigwedge_j \neg f_b(\mathtt{prec}_j(\alpha)) \wedge \neg fail_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge \phi)$ where ϕ does not contain $fail_i$ (corresponds to C6)

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

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

Conditions on models for (ni)

Models corresponding to the **(ni)** strategy satisfy conditions C1-C3, C5 and C6 above, and

C7 If $\wedge_j \neg start_j$, $f_g(\kappa_i)$ 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 C4 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 $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$.

Axiomatization for (ni)

CL, PDL, A1, A2, A3, A4, A7, A8 as before;

A9 $\bigwedge_j \neg start_j \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge \phi \rightarrow \langle[\delta_{r_i}]\rangle(start_i \wedge \phi)$, where ϕ does not contain $start_i$

A10 $\bigvee_j start_j \vee \neg(f_g(\kappa_i) \wedge f_b(\beta_i)) \rightarrow [\delta_{r_i}]\bot$.

A9 and A10 replace A5 and A6 and correspond to C7. Let us call the axiom system above $\mathbf{Ax}(\mathbf{\Lambda}, \mathbf{ni})$.

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

Conditions on models for (as)

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:
  $\texttt{norules} = \bigwedge_{r_i \in \Lambda}(start_i \vee \neg(f_g(\kappa_i) \wedge f_b(\beta_i)))$

C8 If $\neg start_i$, $f_g(\kappa_i)$, $f_b(\beta_i)$ and $\neg\texttt{x} \vee \texttt{noplans}$ 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 $V_c(s') = V_c(s) \cup \{start_i\} \cup \{\neg step_j : r_j \in \Lambda\}$.

(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.)

C9 If $f_b(\texttt{prec}_j(\alpha))$, $\neg fail_i$ and $\texttt{x} \vee \texttt{norules}$ are true in s, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = T_j(\alpha, V_b(s))$, $V_g(s') = V_g(s) \setminus V_b(s')$ 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.)

C10 If $\vee_j f_b(\texttt{prec}_j(\alpha))$ and $fail_i$ are false in s, and $\texttt{x} \vee \texttt{norules}$ is true, then $R_{\alpha_i}(s, s')$ iff $V_b(s') = V_b(s)$ and $V_g(s') = V_g(s)$ and $V_c(s') = V_c(s) \cup \{fail_i, step_i\}$.

(Corresponds to transition (1b): if an action is chosen to be executed but none of its preconditions hold, the plan is removed and rule execution is enabled.)

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

Axiomatisation for (as)

CL, PDL, A1, A2, A3, A4, A6 as before;

A11 $\neg start_i \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge (\neg\texttt{x} \vee \texttt{noplans}) \wedge \phi \rightarrow \langle[\delta_{ri}]\rangle(start_i \wedge \bigwedge_j \neg step_j \wedge \phi)$ where ϕ does not contain $start_i$ and $step_j$ for any j.

A12 $f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge (\texttt{x} \vee \texttt{norules}) \wedge \phi \rightarrow \langle[\alpha_i]\rangle(step_i \wedge f_p(\texttt{post}_\texttt{j}(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_\texttt{j}(\alpha)$ and $step_i$

A13 $\bigwedge_j \neg f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge (\texttt{x} \vee \texttt{norules}) \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge step_i \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_i$

A11–A12 correspond to the conditions C8–C10, respectively.

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

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

Conditions on models for (am)

Below we use the following abbreviation:

- planning phase: $\texttt{p} = \bigwedge_{r_i \in \Lambda}(start_i \rightarrow step_i \vee fail_i)$

Models corresponding to the **(am)** strategy satisfy C1-C3 above and

C11 As C8 but with $\neg\texttt{x} \vee \texttt{noplans}$ replaced with p.

(A rule can be fired 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.)

C12 As C9 but with $\texttt{x} \vee \texttt{norules}$ replaced with $\neg step_i$.

(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.)

C13 As C9 but with $\texttt{x} \vee \texttt{norules}$ replaced with $step_i \wedge \texttt{norules}$ and the condition on $V_c(s')$ changed to $V_c(s') = V_c(s) \setminus \{step_j : j \neq i\}$.

(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.)

C14 As C10 but with $\texttt{x} \vee \texttt{norules}$ replaced with $\neg step_i$.

(Corresponds to transition (1b) for the case when π_i has not performed a step.)

C15 As C10 but with $\texttt{x} \vee \texttt{norules}$ replaced with $step_i \wedge \texttt{norules}$ and the condition for $V_c(s')$ changed to $V_c(s') = (V_c(s) \cup \{fail_i\}) \setminus \{step_j : j \neq i\}$.

(Corresponds to transition (1b) when π_i has performed a step, but no rules are applicable.)

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

Axiomatization for (am)

CL, PDL, A1, A2, A3, A4, A6 as before;

A14 $\neg start_i \wedge G\kappa_i \wedge B\beta_i \wedge \texttt{p} \wedge \phi \rightarrow \langle[\delta_{ri}]\rangle(start_i \wedge \bigwedge_j \neg step_j \wedge \phi)$ where ϕ does not contain $start_i$ and $step_j$ for any j.

A15 $f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge \neg step_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(step_i \wedge f_p(\texttt{post}_\texttt{j}(\alpha)) \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_\texttt{j}(\alpha)$ and $step_i$

A16 $f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge step_i \wedge \texttt{norules} \wedge \phi \rightarrow \langle[\alpha_i]\rangle(f_p(\texttt{post}_\texttt{j}(\alpha)) \wedge \bigwedge_{j \neq i} \neg step_j \wedge \phi)$, where ϕ does not contain variables from $\texttt{post}_\texttt{j}(\alpha)$ and $step_j$ for all $j \neq i$

A17 $\bigwedge_j \neg f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge \neg step_i \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge step_i \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_i$

A18 $\bigwedge_j \neg f_p(\texttt{prec}_\texttt{j}(\alpha)) \wedge \neg fail_i \wedge step_i \wedge \texttt{norules} \wedge \phi \rightarrow \langle[\alpha_i]\rangle(fail_i \wedge \bigwedge_{j \neq i} \neg step_j \wedge \phi)$ where ϕ does not contain $fail_i$ and $step_j$ for all $j \neq i$

A14–A18 correspond to C11–C15, respectively. Let us call the axiom system above $\mathbf{Ax}(\mathbf{\Lambda}, \mathbf{am})$.

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

Proof sketch for Theorems 1 - 4.

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 (Blackburn, de Rijke, & Venema 2001). Take a consistent formula ϕ; we are going to build a finite satisfying model M for ϕ. We take a Fisher-Ladner closure of the set of subformulas of ϕ; we add an

extra condition that if an action α_i occurs in ϕ, then the closure $CL(\phi)$ should contain all pre- and postconditions for α_i. 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 a basic action α_i, δ_{ri} 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 $B(-)p \in A$, where $B(-)p \in CL(\phi)$;
- $(-)p \in V_g(A)$ 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 $\mathbf{M}(\mathbf{\Lambda},(\mathbf{x}))$ for $\mathbf{x} \in \{\mathbf{i}, \mathbf{ni}, \mathbf{as}, \mathbf{am}\}$. Here we show the common conditions:

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 $fail_i\wedge\hat{A}\wedge\langle\alpha_i\rangle\hat{A}$ is consistent, so $R_{\alpha_i}(A,A)$ holds. Observe that for any $B \neq A$, $R_{\alpha_i}(A,B)$ does not hold because by A3 again, $fail_i \wedge \hat{A} \rightarrow [\alpha_i]\hat{A}$ so all the states accessible by R_{α_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 $start_i$ and $fail_i$, 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 $start_i$ and $fail_i$. Finally, since $\hat{A} \rightarrow [e_i](\neg start_i \wedge \neg fail_i)$, $V_c(B) = V_c(A)\setminus\{start_i, fail_i\}$. Similarly, using the $\langle e_i\rangle$ version of A4 we can show that for any B which differs from A at most in its assignment to $start_i$ and $fail_i$, $R_{e_i}(A,B)$ holds.

For each of the deliberation strategies, there is a similar close correspondence between conditions on models and axioms. □

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 agent's execution, 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 $\delta_{ri}; f_p(\pi_i); e_i$ 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 $\delta_{ri}; f_p(\pi_i); e_i$, that is $(\delta_{ri}; f_p(\pi_i); e_i)^+$, may be interleaved with several consecutive executions of $\delta_{rj}; f_p(\pi_j); e_j$, that is, $(\delta_{rj}; f_p(\pi_j); e_j)^+$. 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:

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

that is, the interleaving of one or more repetitions of some subset 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 $\mathbf{Ax}(\mathbf{\Lambda},\mathbf{i})$ (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).

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.

(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 $M \in \mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$ 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 sketch. 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 $\mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$. 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 $\mathtt{prec}_j(\alpha) \subseteq \sigma$, 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 $V_b(s') = T_j(\alpha, V_b(s), V_g(s') = V_g \backslash V_b(s')$ 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 $\Pi' = \Pi \setminus \{r_i : \alpha; \pi\}$. 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 $R_{f_b(\phi)?}(s, s)$ and $s \sim_{(\mathtt{i})} c'$; otherwise $\neg\phi$ is true in c, and $R_{\neg f_b(\phi)?}(s, s)$ and $s \sim_{(\mathtt{i})} c'$.

(4): if there is some rule r_i which is not among $\{r_1, \ldots, r_n\}$, 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 $r_i : \pi_i$ will in its plan base. Then by condition C4, $R_{\delta_{r_i}}(s, s')$ 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 sketch. 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 $\delta_{r\,i}$ 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 systems generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$ 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 $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$.

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 systems generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\mathbf{\Lambda}, \mathbf{as})$ 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 $\mathbf{M}(\mathbf{\Lambda}, \mathbf{as})$.

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 systems generated by the operational semantics for this agent with initial configuration c_0. Let $M \in \mathbf{M}(\mathbf{\Lambda}, \mathbf{am})$ 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 $\mathbf{M}(\mathbf{\Lambda}, \mathbf{am})$.

Example

As an example, we briefly illustrate how to prove properties of agents in our logic using a simple agent program. The agent has two goals, to have breakfast and to be at work. The agent's program is:

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

r2: breakfast <- home | eat_breakfast
```

The pre- and postconditions of the agent's updates are:

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

and its initial beliefs and goals are given by:

```
Beliefs: home, raining
Goals: breakfast, work
```

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_breakfast` as t.

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

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

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 (Schmidt 2003) 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 $\mathbf{Ax}(\mathbf{\Lambda}, \mathbf{am})$ from the axioms such as the following instances of A14 and A15:

(r1) $\neg start_1 \wedge Go \wedge Bh \wedge Br \wedge Gb \wedge (start_2 \rightarrow step_2 \vee fail_2) \rightarrow \langle [\delta_{r1}] \rangle (start_1 \wedge Bh \wedge Br \wedge Gb)$

(r2) $\neg start_2 \wedge Gb \wedge Bh \wedge (Br \wedge Go) \wedge (start_1 \rightarrow step_1 \vee fail_1) \rightarrow \langle [\delta_{r2}] \rangle (start_2 \wedge Bh \wedge Br \wedge Go)$

(u) $Bh \wedge \neg fail_1 \wedge \neg step_1 \wedge Go \wedge Gb \rightarrow \langle [u_1] \rangle (step_1 \wedge Bh \wedge Go \wedge Gb)$

(w) $Bh \wedge \neg fail_1 \wedge \neg step_1 \rightarrow \langle [w_1] \rangle (step_1 \wedge Bo \wedge B{-}h)$

(t) $Bh \wedge \neg fail_2 \wedge \neg step_2 \wedge Go \rightarrow \langle [t_2] \rangle (step_2 \wedge Bb \wedge Bh \wedge Go)$

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)**: $\delta_{r1}; u_1; w_1; e_1$. 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 $\delta_{r1}; u_1; \delta_{r2}; w_1; t_2$ 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.

Conclusion

In this paper we analyzed the implications of an agent's deliberation strategy in determining the behavior 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 execution 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 axiomatize 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 axiomatizations share significant structure, concisely characterising the similarities and differences between strategies, and facilitating the formalization of new strategies.

In future work we plan to investigate other deliberation strategies. For example, it would also be interesting to investigate strategies which prioritize 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

Abrahamson, K. R. 1980. *Decidability and expressiveness of logics of processes*. Ph.D. Dissertation, Department of Computer Science, University of Washington.

Alechina, N.; Dastani, M.; Logan, B.; and Meyer, J.-J. C. 2007. A logic of agent programs. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007)*, 795–800. AAAI Press.

Benerecetti, M.; Giunchiglia, F.; and Serafini, L. 1998. Model checking multiagent systems. *J. Log. Comput.* 8(3):401–423.

Blackburn, P.; de Rijke, M.; and Venema, Y. 2001. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

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

Bordini, R. H.; Fisher, M.; Visser, W.; and Wooldridge, M. 2006. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems* 12(2):239–256.

Bordini, R. H.; Hübner, J.; and Vieira, R. 2005. Jason and the golden fleece of agent-oriented programming. In Bordini, R. H.; Dastani, M.; Dix, J.; and Seghrouchni, A. E. F., eds., *Multi-Agent Programming - Languages, Platforms and Applications*. Springer.

Dastani, M., and Meyer, J. 2007. A Practical Agent Programming Language. In *Proc. of PROMAS*.

Dastani, M.; van Riemsdijk, M. B.; Dignum, F.; and Meyer, J.-J. C. 2004. A programming language for cognitive agents: Goal directed 3APL. In *Proc. ProMAS 2003*, volume 3067 of *LNCS*, 111–130. Springer.

Dastani, M.; van Riemsdijk, M. B.; and Meyer, J.-J. C. 2005. Programming multi-agent systems in 3APL. In Bordini, R. H.; Dastani, M.; Dix, J.; and Seghrouchni, A. E. F., eds., *Multi-Agent Programming - Languages, Platforms and Applications*. Springer.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, 677–682.

Harel, D.; Kozen, D.; and Tiuryn, J. 2000. *Dynamic Logic*. MIT Press.

Hindriks, K., and Meyer, J.-J. C. 2007. Agent logics as program logics: Grounding KARO. In *Proc. 29th German Conference on AI (KI 2006)*, volume 4314 of *LNAI*. Springer.

Lomuscio, A., and Raimondi, F. 2006. MCMAS: A model checker for multi-agent systems. In *Proc. TACAS 2006*, 450–454.

Mulder, M.; Treur, J.; and Fisher, M. 1997. Agent modelling in METATEM and DESIRE. In Singh, M. P.; Rao, A. S.; and Wooldridge, M., eds., *Intelligent Agents IV, Agent Theories, Architectures, and Languages, 4th International Workshop (ATAL'97)*, volume 1365 of *Lecture Notes in Computer Science*, 193–207. Springer.

Pokahr, A.; Braubach, L.; and Lamersdorf, W. 2005. Jadex: A BDI reasoning engine. In Bordini, R. H.; Dastani, M.; Dix, J.; and Seghrouchni, A. E. F., eds., *Multi-Agent Programming - Languages, Platforms and Applications*. Springer.

Schmidt, R. A. 2003. PDL-TABLEAU. `http://www.cs.man.ac.uk/~schmidt/pdl-tableau`.

Shapiro, S.; Lespérance, Y.; and Levesque, H. J. 2002. The cognitive agents specification language and verification environment for multiagent systems. In *Proc. AAMAS 2002*, 19–26. ACM Press.