Using theorem proving to verify properties of agent programs

N. Alechina, M. Dastani, F. Khan, B. Logan and J.-J. Ch. Meyer

Abstract We present a sound and complete logic for automatic verification of SimpleAPL programs. SimpleAPL is a fragment of agent programming languages such as 3APL and 2APL designed for the implementation of cognitive agents with beliefs, goals and plans. Our logic is a variant of PDL, and allows the specification of safety and liveness properties of agent programs. We prove a correspondence between the operational semantics of SimpleAPL and the models of the logic for two example program execution strategies. We show how to translate agent programs written in SimpleAPL into expressions of the logic, and give an example in which we show how to verify correctness properties for a simple agent program using theorem-proving.

1 Introduction

The specification and verification of agent architectures and programs is a key problem in agent research and development. Formal verification provides a degree of certainty regarding system behaviour which is difficult or impossible to obtain using conventional testing methodologies, particularly when applied to autonomous systems operating in open environments. For example, the use of appropriate specification and verification techniques can allow agent researchers to check that agent architectures and programming languages conform to general principles of rational agency, or agent developers to check that a particular agent program will achieve the agent's goals in a given range of environments. Ideally, such techniques should

N. Alechina, F. Khan, B. Logan
University of Nottingham, School of Computer Science, UK e-mail: {nza,afk,bsl}@cs.nott.ac.uk

M. Dastani, J.-J. Ch. Meyer
Universiteit Utrecht, Department of Information and Computing Sciences, The Netherlands e-mail: {mehdi,jj}@cs.uu.nl

allow specification of key aspects of the agent's architecture such as its execution cycle (e.g., to explore commitment under different program execution strategies), and should admit a fully automated verification procedure. However, while there has been considerable work on the formal verification of software systems and on logics of agency, it has proved difficult to bring this work to bear on verification of agent programs. On the one hand, it can be difficult to specify and verify relevant properties of agent programs using conventional formal verification techniques, and on the other, standard epistemic logics of agency (e.g., [17]) fail to take into account the computational limitations of agent implementations.

Since an agent program is a special kind of program, logics intended for the specification of conventional programs can be used for specifying agent programming languages. In this approach we have some set of propositional variables or predicates to encode the agent's state, and, for example, dynamic or temporal operators for describing how the state changes as the computation evolves. However, for agents based on the Belief-Desire-Intention model of agency, such an approach fails to capture important structure in the agent's state which can be usefully exploited in verification. For example, we could encode the fact that the agent has the belief that p as the proposition u_1, and the fact that the agent has the goal that p as the proposition u_2. However such an encoding obscures the key logical relationship between the two facts, making it difficult to express general properties such as 'an agent cannot have as a goal a proposition which it currently believes'. It therefore seems natural for a logical language intended for reasoning about agent programs to include primitives for beliefs and goals the agent, e.g., where Bp means that the agent believes that p, and Gp means that the agent has a goal that p.

The next natural question is, what should the semantics of these operators be? For example, should the belief operator satisfy the KD45 properties? In our view, it is critical that the properties of the agent's beliefs and goals should be grounded in the computation of the agent (in the sense of [22]). If the agent implements a full classical reasoner (perhaps in a restricted logic), then we can formalise its beliefs as closed under classical inference. However if the agent's implementation simply matches belief literals against a database of believed propositions without any additional logical reasoning, we should not model its beliefs as closed under classical consequence.

In this paper, we present an approach to specification and verification which is tailored to the requirements of BDI-based agent programming languages [7]. Our approach is grounded in the computation of the agent and admits an automated verification procedure based on theorem proving. The use of theorem proving rather than model checking is motivated by the current state of the art regarding available verification frameworks and tools for PDL. In particular, to the best of our knowledge there is no model checking framework for PDL, while theorem proving techniques for this logic are readily available [23, 28]. We develop our approach in the context of SimpleAPL, a simplified version of the logic-based agent programming languages 3APL [15, 7] and 2APL [13, 11]. We present a sound and complete variant of PDL [18] for SimpleAPL which allows the specification of safety and liveness properties of SimpleAPL programs. Our approach allows us to capture the

agent's execution strategy in the logic, and we prove a correspondence between the operational semantics of SimpleAPL and the models of the logic for two example execution strategies. Finally, we show how to translate agent programs written in SimpleAPL into expressions of the logic, and give an example in which we verify correctness properties of a simple agent program using the PDL theorem prover MSPASS [23]. While we focus on APL-like languages and consider only single agent programs, our approach can be generalised to other BDI-based agent programming languages and the verification of multi-agent systems.

2 An Agent Programming Language

In this section we present the syntax and semantics of SimpleAPL, a simplified version of logic based agent-oriented programming languages 3APL [15, 7] and 2APL [13, 11]. SimpleAPL contains the core features of 3APL and 2APL and allows the implementation of agents with beliefs, goals, actions, plans, and planning rules. The main features of 3APL/2APL not present in SimpleAPL are a first order language for beliefs and goals[1], a richer set of actions (e.g., abstract plans, communication actions) and a richer set of rule types (e.g., rules for revising plans and goals and for processing events).

2.1 SimpleAPL

Beliefs and Goals The *beliefs* of an agent represent its information about its environment, while its *goals* represent situations the agent wants to realize (not necessary all at once). The agent's beliefs are represented by a set of positive literals and its goals by a set of arbitrary literals. The initial beliefs and goals of an agent are specified by its program. For example, a simple vacuum cleaner agent might initially believe that it is in room 1 and its battery is charged:

```
Beliefs: room1, battery
```

and may initially want to achieve a situation in which both room 1 and room 2 are clean:

```
Goals: clean1, clean2
```

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, and if an agent does not believe that p, it will not have $-p$ as a goal.

[1] In 3APL and 2APL, an agent's beliefs are implemented as a set of first-order Horn clauses and an agent's goals are implemented as a set of conjunctions of ground atoms.

Basic Actions Basic actions specify the capabilities an agent can use to achieve its goals. There are three types of basic action: those that update the agent's beliefs and those which test its beliefs and goals. A *belief test action* tests whether a boolean belief expression is entailed by the agent's beliefs, i.e., it tests whether the agent has a certain belief. A *goal test action* tests whether a boolean goal expression is entailed by the agent's goals, i.e., it tests whether the agent has a certain goal. *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 one of its pre-conditions is entailed by the agent's current beliefs. Executing the action updates the agent's beliefs to make the corresponding postcondition entailed by the agent's belief base. While the belief base of the agent contains only positive literals, belief and goal expressions appearing in belief and goal test actions can be complex, and the pre- and postconditions of belief update actions may contain negative literals. We will define the notion of 'entailed' formally below. Informally, a pre-condition of an action is entailed by the agent's belief base if all positive literals in the precondition are contained in the agent's belief base, and for every negative literal $-p$ in the precondition, p is not in the belief base (i.e., we use entailment under the closed world assumption). After executing a belief update action, all positive literals in the corresponding postcondition are added to the belief base, and for every negative literal $-p$ in the postcondition, p is removed from the agent's belief base. For example, the following belief update specifications:

```
BeliefUpdates:
  {room1}           moveR  {-room1, room2}
  {room1, battery}  suck   {clean1, -battery}
  {room2, battery}  suck   {clean2, -battery}
  {room2}           moveL  {-room2, room1}
  {-battery}        charge {battery}
```

can be read as "if the agent is in room 1 and moves right, it ends up in room 2", and "if the agent is in room 1 and its battery is charged, it can perform a 'suck' action, after which room 1 is clean and its battery is discharged". Note that performing a 'suck' action in a different state, e.g., in the state where the agent is in room 2, has a different result. Belief update actions are assumed to be deterministic, i.e., the pre-conditions of an action are assumed to be mutually exclusive.

Updating the agent's beliefs may result in achievement of one or more of the agent's goals. Goals which are achieved by the postcondition of an action are dropped. For example, if the agent has a goal to clean room 1, executing a 'suck' action in room 1 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, so the agent's beliefs describe the state of the real world. This assumption can be relaxed in a straightforward way by including the state of the environment in the models.

Plans In order to achieve its goals, an agent adopts *plans*. A plan consists of basic 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` $\{\pi_1\}$ `else` $\{\pi_2\}$ and `while` ϕ `do` $\{\pi\}$ respectively. The condition ϕ is evaluated with respect to the agent's current beliefs. For example, the plan:

```
if room1 then {suck} else {moveL; suck}
```

causes the agent to clean room 1 if it's currently in room 1, otherwise it first moves to room 1 and then cleans it.

Planning Goal Rules *Planning goal rules* are used by the agent to select a plan based on its current goals and beliefs. A planning goal rule consists of three parts: an (optional) goal query, a belief query, and a plan. The goal query specifies which goal(s) the plan achieves, and the belief query characterises the situation(s) in which it could be a good idea to execute the plan. Firing a planning goal rule causes the agent to adopt the specified plan. For example, the planning goal rule:

```
clean2 <- battery |
  if room2 then {suck} else {moveR; suck}
```

states that "if the agent's goal is to clean room 2 and its battery is charged, then the specified plan may be used to clean the room". Note that an agent can generate a plan based only on its current beliefs as the goal query is optional. This allows the implementation of *reactive* agents (agents without any goals). For example, the reactive rule:

```
<- -battery |
  if room2 then {charge} else {moveR; charge}
```

states "if the battery is low, the specified plan may be used to charge it". 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.

2.2 SimpleAPL syntax

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 and ⟨*literal*⟩ (⟨*pliteral*⟩) to denote belief and goal literals (positive literals).

⟨*APL_Prog*⟩ ::= `"BeliefUpdates:"` ⟨*updatespecs*⟩
| `"Beliefs:"` ⟨*pliterals*⟩
| `"Goals":` ⟨*literals*⟩
| `"PG rules:"` ⟨*pgrules*⟩

⟨*updatespecs*⟩ ::= [⟨*updatespec*⟩ `","` ⟨*updatespec*⟩)*]

⟨*updatespec*⟩ ::= `"{"` ⟨*literals*⟩ `"}"` ⟨*aliteral*⟩ `"{"`⟨*literals*⟩`"}"`

⟨*pliterals*⟩ ::= [⟨*pliteral*⟩ `","` ⟨*pliteral*⟩)*]

⟨*literals*⟩ ::= [⟨*literal*⟩ `","` ⟨*literal*⟩)*]

⟨*plan*⟩ ::= ⟨*baction*⟩ | ⟨*seqplan*⟩ | ⟨*ifplan*⟩ | ⟨*whileplan*⟩
⟨*baction*⟩ ::= ⟨*aliteral*⟩ | ⟨*testbelief*⟩ | ⟨*testgoal*⟩
⟨*testbelief*⟩ ::= ⟨*bquery*⟩ `"?"`
⟨*testgoal*⟩ ::= ⟨*gquery*⟩ `"!"`
⟨*bquery*⟩ ::= ⟨*literal*⟩ | ⟨*bquery*⟩ `"and"` ⟨*bquery*⟩ | ⟨*bquery*⟩ `"or"` ⟨*bquery*⟩
⟨*gquery*⟩ ::= ⟨*literal*⟩ | ⟨*gquery*⟩ `"or"` ⟨*gquery*⟩
⟨*seqplan*⟩ ::= ⟨*plan*⟩ `";"` ⟨*plan*⟩
⟨*ifplan*⟩ ::= `"if"` ⟨*bquery*⟩ `"then {"` ⟨*plan*⟩ `"}"` [`"else {"` ⟨*plan*⟩ `"}"`]
⟨*whileplan*⟩ ::= `"while"` ⟨*bquery*⟩ `"do {"` ⟨*plan*⟩ `"}"`
⟨*pgrules*⟩ ::= [⟨*pgrule*⟩ (`","` ⟨*pgrule*⟩)*]
⟨*pgrule*⟩ ::= [⟨*gquery*⟩] `"<-"` ⟨*bquery*⟩ `"|"` ⟨*plan*⟩

3 Operational Semantics

We define the operational semantics of SimpleAPL in terms of a transition system. A transition system is a graph where the nodes are configurations of an agent program and the edges (transitions) are given by a set of transition rules. The configuration of a SimpleAPL agent program consists of the beliefs, goals and plan(s) of the agent. Each transition corresponds to a single computation step. Which transitions are possible in a configuration depends on the agent's execution strategy. Many execution strategies are possible and we do not have space here to describe them all in detail. Below we give two versions of the operational semantics, one for an agent which executes a single plan to completion before choosing another plan (non-interleaved execution), and another for an execution strategy which interleaves the execution of multiple plans with the adoption of new plans (interleaved execution).

These strategies were chosen as representative of deliberation strategies found in the literature and in current implementations of BDI-based agent programming languages. However neither of these strategies (or any other single strategy) is clearly best for all agent task environments. For example, the non-interleaved 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 interleaved 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.

Agent Configuration An agent configuration is a 3-tuple $\langle \sigma, \gamma, \Pi \rangle$ where σ is a set of positive 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 representing the agent's currently

executing plans.[2] In the initial configuration the agent's initial beliefs and goals are those specified by its program, and Π is empty. Executing the agent's program modifies its initial configuration in accordance with the transition rules presented below. We first present the transition rules for the non-interleaved execution strategy and then those for interleaved execution.

For the formulation of the operational semantics we need to formalize 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:

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

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:

$$\begin{array}{ll}
\gamma \models_g p & \Leftrightarrow p \in \gamma \\
\gamma \models_g -p & \Leftrightarrow -p \in \gamma \\
\gamma \models_g \phi \texttt{ or } \psi & \Leftrightarrow \gamma \models_g \phi \text{ or } \gamma \models_g \psi
\end{array}$$

Note that "$\gamma \models_g \phi \texttt{ and } \psi \;\Leftrightarrow\; \gamma \models_g \phi$ and $\gamma \models_g \psi$" does not hold since ϕ and ψ may be entailed by two different goals γ_1 and γ_2 from γ, but there may be no $\gamma_i \in \gamma$ which entails both ψ and ϕ. In fact, in SimpleAPL there are no non-trivial conjunctive goal queries (that is, not of the form p `and` p) which may be entailed by the goal base, since the goal base consists of literals.

We assume that each belief update action α has a set of preconditions $\texttt{prec}_1(\alpha)$, $\ldots$, $\texttt{prec}_k(\alpha)$. Each $\texttt{prec}_i(\alpha)$ is a finite set of belief literals, and any two preconditions for an action α, $\texttt{prec}_i(\alpha)$ and $\texttt{prec}_j(\alpha)$ $(i \neq j)$, are mutually exclusive (i.e., for any belief base σ, if $\sigma \models_{cwa} \texttt{prec}_i(\alpha)$ and $\sigma \models_{cwa} \texttt{prec}_j(\alpha)$ then $i = j$). For each $\texttt{prec}_i(\alpha)$ there is a unique corresponding postcondition $\texttt{post}_i(\alpha)$, which is also a finite set of literals. A belief update action α can be executed if the current set of agent's beliefs σ entails some precondition $\texttt{prec}_j(\alpha)$ of α with respect to $\models_{cwa}$. This holds when all positive literals p in $\texttt{prec}_j(\alpha)$ are in σ and $\sigma \cap \{p : -p \in \texttt{prec}_j(\alpha)\} = \emptyset$. The effect of updating a set of beliefs σ with α is given by $T_j(\alpha, \sigma) = \sigma \cup (\{p : p \in \texttt{post}_j(\alpha)\} \setminus \{p : -p \in \texttt{post}_j(\alpha)\})$, (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).

[2] As an agent's planning goal rules do not change during the execution of the agent's program, we do not include them in the agent configuration.

3.1 Non-interleaved execution

By non-interleaved execution we mean the following execution strategy: when in a configuration with no plan, choose a planning goal rule non-deterministically, apply it, execute the resulting plan; repeat.

Belief Update Actions A belief update action α can be executed if one of its preconditions is entailed by the agent's beliefs, i,e., $\sigma \models_{cwa} \phi$. Executing the action adds the literals in the corresponding postcondition to the agent's beliefs and removes any existing beliefs which are inconsistent with the postcondition, and causes the agent to drop any goals it believes to be achieved as a result of the update.

$$(1) \quad \frac{\sigma \models_{cwa} \mathtt{prec}_j(\alpha) \qquad T_j(\alpha,\sigma) = \sigma'}{\langle \sigma, \gamma, \{\alpha;\pi\}\rangle \longrightarrow \langle \sigma', \gamma', \{\pi\}\rangle}$$

where $\gamma' = \gamma \setminus (\{p : p \in \sigma'\} \cup \{-p : p \notin \sigma'\})$ and T_j is the function that determines the effect of a belief update action on a belief base as defined above. Note that in this and in rules (2)-(7) below, π may be empty, in which case α; is identical to α.[3]

Belief and Goal Test Actions A belief test action $\beta?$ can be executed if β is entailed by the agent's beliefs.

$$(2) \quad \frac{\sigma \models_{cwa} \beta}{\langle \sigma, \gamma, \{\beta?;\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

The execution of a belief test action $\beta?$ in a configuration where β is not entailed by the agent's beliefs causes execution of the plan to block. In the case of non-interleaved execution, this causes the whole agent to block.

A goal test action $\kappa!$ can be executed if κ is entailed by the agent's goals.

$$(3) \quad \frac{\gamma \models_{g} \kappa}{\langle \sigma, \gamma, \{\kappa!;\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

Similar to the belief test action, the execution of a goal test action $\kappa!$ in a configuration where κ is not entailed by the agent's goals, blocks.

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

$$(4) \quad \frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{(\mathtt{if}\ \phi\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2);\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1;\pi\}\rangle}$$

$$(5) \quad \frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{(\mathtt{if}\ \phi\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2);\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_2;\pi\}\rangle}$$

$$(6) \quad \frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{(\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1);\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1;(\mathtt{while}\ \phi\ \mathtt{do}\ \pi_1);\pi\}\rangle}$$

[3] This avoids introducing an additional transition rule for the sequence operator ;

$$(7)\quad \frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{(\texttt{while } \phi \texttt{ do } \pi_1);\pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

Planning Goal Rules A planning goal rule $\kappa \leftarrow \beta|\pi$ can be applied if κ is entailed by the agent's goals and β is entailed by the agent's beliefs. Applying the rule adds π to the agent's plans.

$$(8)\quad \frac{\gamma \models_g \kappa \quad \sigma \models_{cwa} \beta}{\langle \sigma, \gamma, \{\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

3.2 Interleaved execution

By interleaved execution we mean the following execution strategy: either apply a planning goal rule, or execute the first step in any of the current plans; repeat. Interleaved execution strategies are characteristic of many 'event-driven' agent programming languages such as AgentSpeak(L) [25] and its derivatives [10], where the agent may adopt a new intention at each processing cycle and can pursue multiple intentions in parallel. To simplify the presentation of the operational semantics of the interleaved strategy, we associate a unique name with each planning goal rule $r_i = \kappa_i \leftarrow \beta_i|\pi_i$, and add to each plan entry in the plan base the name of the planning goal rule whose application generated the plan entry, (i.e., an entry $r_i : \pi$ in the plan base indicates that the plan π was generated by applying the planning goal rule $r_i : \kappa_i \leftarrow \beta_i \mid \pi_i$). Note that, in a particular configuration, the actual plan π in the plan base may be different from the π_i generated by applying the planning goal rule r_i if some prefix of π_i has already been executed.

The transitions for an interleaved execution strategy are:

Belief updates Similar to transition rule (1), the following rule specifies the execution of belief update action in a configuration where the plan base can contain more than one plan entry.

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

where $\gamma' = \gamma \setminus \sigma'$. We stipulate that $\Pi \cup \{r_i : \ \} = \Pi$. As before, π may be empty.

Belief and goal tests Again, similar to transition rules (2) and (3), the following rules specify the execution of belief and goal test actions in a configuration where the plan base can contain more than one plan entry.

$$(2^i)\quad \frac{r_i : \beta?;\pi \in \Pi \qquad \sigma \models_{cwa} \beta}{\langle \sigma, \gamma, \Pi\rangle \longrightarrow \langle \sigma, \gamma, (\Pi \setminus \{r_i : \beta?;\pi\}) \cup \{r_i : \pi\}\rangle}$$

$$(3^i)\quad \frac{r_i : \kappa!;\pi \in \Pi \qquad \gamma \models_g \kappa}{\langle \sigma, \gamma, \Pi\rangle \longrightarrow \langle \sigma, \gamma, (\Pi \setminus \{r_i : \kappa!;\pi\}) \cup \{r_i : \pi\}\rangle}$$

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

$$(4^i)\quad \frac{r_i : (\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2); \pi \in \Pi \quad \sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi_1; \pi\} \rangle}$$

$$(5^i)\quad \frac{r_i : (\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2); \pi \in \Pi \quad \sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi_2; \pi\} \rangle}$$

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

$$(6^i)\quad \frac{r_i : (\texttt{while } \phi \texttt{ do } \pi_1); \pi \in \Pi \quad \sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : (\pi_1; \texttt{while } \phi \texttt{ do } \pi_1); \pi\} \rangle}$$

$$(7^i)\quad \frac{r_i : (\texttt{while } \phi \texttt{ do } \pi_1); \pi \in \Pi \quad \sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi' \cup \{r_i : \pi\} \rangle}$$

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

Planning goal rules 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 (possibly partially executed) plan generated by applying r_i. Applying the rule r_i adds π_i to the agent's plans.

$$(8^i)\quad \frac{\gamma \models_g \kappa_i \quad \sigma \models_{cwa} \beta_i \quad r_i : \pi \notin \Pi}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \cup \{r_i : \pi_i\} \rangle}$$

The transition system TS for the agent's program is generated by the initial configuration c_0 if it consists of c_0 and all configurations which can be reached by applying the above mentioned transition rules. Recall that the initial configuration always has an empty plan base.

4 Logic

In this section, we introduce a logic which allows us to specify properties of SimpleAPL agent programs.

We begin by defining transition systems which capture the *capabilities* of agents as specified by their belief update actions. These transition systems are more general than both versions of the operational semantics presented above, in that they do not describe a particular agent program or execution strategy, but all possible basic transitions between the possible belief and goal states of an agent. We then show how to interpret a variant of Propositional Dynamic Logic (PDL) with belief and goal operators in this semantics, and give a sound and complete axiom system for the logic. In section 4.4 we show how the beliefs, goals and plans of an agent can be translated into our logic.

4.1 Preliminary

The models of the logic are defined relative to an agent program with a set of planning goal rules Λ and a set of pre- and postconditions for all belief update actions $\mathbf{C}$. Let P denote the set of propositional variables occurring in Λ. A state s corresponds to a pair $\langle \sigma, \gamma \rangle$, where:

- $\sigma \subseteq P$ is a set of beliefs, and
- γ is a set of goals $\{(-)u_1, \ldots, (-)u_n : u_i \in P\}$; no goal in γ should be entailed (with respect to $\models_{cwa}$) by σ.

We model states as points, and beliefs and goals are assigned to a state by two assignments, V_b and V_g. Let the set of belief update actions be $\mathtt{Ac} = \{\alpha_1, \ldots, \alpha_m\}$. Executing an action α_i in different configurations may give different results so that for each $\alpha_i \in \mathtt{Ac}$ we have an associated set of pre- and postcondition pairs $\{(\mathtt{prec}_1, \mathtt{post}_1), \ldots, (\mathtt{prec}_k, \mathtt{post}_k)\}$ denoted by $C(\alpha_i)$. We assume that $C(\alpha_i)$ is finite, that different preconditions are mutually exclusive, and that each precondition has exactly one associated postcondition. If $\sigma \models_{cwa} \mathtt{prec}_j(\alpha)$ and $T_j(\alpha, \sigma) = \sigma'$, then in the models of our logic there will be a transition R_α from a state $s = (\sigma, \gamma)$ to a state $s' = (\sigma', \gamma')$ where $\gamma' = \gamma \setminus (\{p : p \in \sigma'\} \cup \{-p : p \notin \sigma'\})$.

4.2 Language

Assume that we can make PDL program expressions ρ out of belief update actions $\alpha_i \in \mathtt{Ac}$ by using sequential composition ;, test on formulas ?, union $\cup$, and finite iteration $*$. The formulas on which we can test are any formulas of the language L defined below, although to express SimpleAPL plans we only need tests on beliefs and goals. Let Υ be the set of program expressions constructed in this way.

The language L for talking about the agent's beliefs, goals and plans is the language of PDL extended with a belief operator B and a goal operator G. A formula of L is defined as follows: if $p \in P$, then Bp and $G(-)p$ are formulas; if ρ is a program expression and ϕ a formula, then $\langle \rho \rangle \phi$ and $[\rho]\phi$ are formulas; and L is closed under the usual boolean connectives. In the following, we will refer to the sublanguage of L which does not contain program modalities $\langle \rho \rangle$ and $[\rho]$ as L_0.

4.3 Semantics

A model for L is a structure $M = (S, \{R_\rho : \rho \in \Upsilon\}, V)$, where

- S is a set of states.
- $V = (V_b, V_g)$ is the evaluation function consisting of belief and goal valuation functions V_b and V_g; each state s can be identified with a pair (σ, γ), where $V_b(s) = \sigma$ and $V_g(s) = \gamma$.

- We define R_ρ for $\rho \in \Upsilon$ inductively by the following clauses:
  - R_α, for each belief update action $\alpha \in \mathtt{Ac}$, is a relation on S such that for any $s, s' \in S$ we have that $R_\alpha(s,s')$ iff for some $(\mathtt{prec}_j, \mathtt{post}_j) \in C(\alpha)$, $T_j(\alpha, V_b(s)) = V_b(s')$ and $V_g(s') = V_g(s) \setminus (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\}$. Note that this implies two things: first, an α transition can only originate in a state s which satisfies one of the preconditions for α; second, since pre-conditions are mutually exclusive, every such s satisfies exactly one pre-condition, and all α-successors of s satisfy the matching post-condition.
  - $R_{\rho_1;\rho_2} = R_{\rho_1} \circ R_{\rho_2} = \{(s_1,s_2) \ : \ s_1, s_2 \in S \ , \ \exists s_3 \in S(R_{\rho_1}(s_1,s_3) \wedge R_{\rho_2}(s_3,s_2))\}$
  - $R_{\phi?} = \{(s,s) \ : \ M,s \models \phi\}$, for each formula $\phi \in L$
  - $R_{\rho_1 \cup \rho_2} = R_{\rho_1} \ \cup \ R_{\rho_2}$
  - $R_{\rho^*} = (R_\rho)^*$, the reflexive transitive closure of R_ρ.

The 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 $(-)p \in V_g(s)$
- $M,s \models \neg\phi$ 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 exists a $s' \in S$ such that $R_\rho(s,s')$ and $M,s' \models \phi$.
- $M,s \models [\rho]\phi$ iff for all $s' \in S$ such that $R_\rho(s,s')$ we have that $M,s' \models \phi$.

Let the class of transition systems defined above be denoted $\mathbf{M_C}$ (note that $\mathbf{M}$ is parameterised by the set $\mathbf{C}$ of pre- and postconditions of belief update actions).

4.4 Axiomatisation

The beliefs, goals and plans of agent programs can be translated into PDL expressions as follows.

- Translation of belief formulas: let $p \in P$ and ϕ, ψ be belief query expressions (i.e., $\langle bquery \rangle$) of SimpleAPL
  - $f_b(p) = Bp$
  - $f_b(-p) = \neg Bp$
  - $f_b(\phi \ \mathtt{and} \ \psi) = f_b(\phi) \wedge f_b(\psi)$
  - $f_b(\phi \ \mathtt{or} \ \psi) = f_b(\phi) \vee f_b(\psi)$

  Observe that negative queries are translated using the closed world assumption: an agent is assumed to believe that p is false if it does not have p in its belief base.
- Translation of goal formulas:
  - $f_g(p) = Gp$
  - $f_g(-p) = G{-}p$

- $f_g(\phi \texttt{ or } \psi) = f_g(\phi) \vee f_g(\psi)$

- 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 (i.e., $\langle plan \rangle$s) of SimpleAPL
  - $f_p(\alpha_i) = \alpha_i$
  - $f_p(\phi?) = f_b(\phi)?$
  - $f_p(\psi!) = f_g(\psi)?$
  - $f_p(\pi_1;\pi_2) = f_p(\pi_1);f_p(\pi_2)$
  - $f_p(\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2) = (f_b(\phi)?;f_p(\pi_1)) \cup (\neg f_b(\phi)?;f_p(\pi_2))$
  - $f_p(\texttt{while } \phi \texttt{ do } \pi) = (f_b(\phi)?;f_p(\pi))^*;\neg f_b(\phi)?$

Proposition 1. *For all states $s = (\sigma, \gamma)$ and for all belief formulae β and goal formulae κ we have that:*

1. $M,s \models f_b(\beta) \Leftrightarrow \sigma \models_{cwa} \beta$
2. $M,s \models f_g(\kappa) \Leftrightarrow \gamma \models_g \kappa$

Proof. We prove these two propositions by induction on the complexity of formulas β and κ, respectively.

1. $M,s \models f_b(\beta) \Leftrightarrow \sigma \models_{cwa} \beta$
   - *Base case*: Let $\beta = p$ or $\beta = -p$. Then, we have:
     $M,s \models f_b(p) \Leftrightarrow M,s \models Bp \Leftrightarrow p \in V_b(s) \Leftrightarrow \sigma \models_{cwa} p$.
     $M,s \models f_b(-p) \Leftrightarrow M,s \models \neg Bp \Leftrightarrow p \notin V_b(s) \Leftrightarrow \sigma \models_{cwa} -p$.
   - *Inductive case*: Let $\beta = \beta_1 \texttt{ and } \beta_2$. Then, $M,s \models f_b(\beta_1 \texttt{ and } \beta_2) \Leftrightarrow M,s \models f_b(\beta_1)$ and $M,s \models f_b(\beta_2) \Leftrightarrow$ (by the inductive hypothesis) $\sigma \models_{cwa} \beta_1$ and $\sigma \models_{cwa} \beta_2 \Leftrightarrow$ (by the definition of $\models_{cwa}$) $\sigma \models_{cwa} \beta_1 \texttt{ and } \beta_2$. The case for $\beta = \beta_1 \texttt{ or } \beta_2$ similarly follows from the inductive hypothesis and $\sigma \models_{cwa} \beta_1$ or $\sigma \models_{cwa} \beta_2$ if and only if $\sigma \models_{cwa} \beta_1 \texttt{ or } \beta_2$.

2. $M,s \models f_g(\kappa) \Leftrightarrow \gamma \models_g \kappa$
   - *Base case*: $\kappa = (-)p$
     $M,s \models f_g((-)p) \Leftrightarrow M,s \models G(-)p \Leftrightarrow (-)p \in V_g(s) \Leftrightarrow \sigma \models_g p$.
   - *Inductive case:* $\kappa = \kappa_1 \texttt{ or } \kappa_2$
     $M,s \models f_g(\kappa_1 \texttt{ or } \kappa_2) \Leftrightarrow M,s \models f_g(\kappa_1) \vee f_g(\kappa_2) \Leftrightarrow M,s \models f_g(\kappa_1)$ or $M,s \models f_g(\kappa_2) \Leftrightarrow$ (by the inductive hypothesis) $\gamma \models_g \kappa_1$ or $\gamma \models_g \kappa_2 \Leftrightarrow$ (by the definition of $\models_g$ $\gamma \models_g \kappa_1 \texttt{ or } \kappa_2$.

Note that for every pre- and postcondition pair $(\texttt{prec}_j, \texttt{post}_j)$ we can describe states satisfying $\texttt{prec}_j$ and states satisfying $\texttt{post}_j$ by formulas of L. More formally, we define a formula $f_b(X)$ corresponding to a pre- or postcondition X as follows: $f_b(\{\phi_1, \ldots, \phi_n\}) = f_b(\phi_1) \wedge \ldots \wedge f_b(\phi_n)$. This allows us to axiomatise pre- and postconditions of belief update actions.

To axiomatise the set of models defined above relative to **C** we need:

CL classical propositional logic

PDL axioms of PDL (see, e.g., [20])

A1 beliefs are not goals (positive): $Bp \rightarrow \neg Gp$

A2 beliefs are not goals (negative): $G-p \rightarrow Bp$

A3 for every belief update action α_i and every pair of pre- and postconditions $(\mathtt{prec}_j, \mathtt{post}_j)$ in $C(\alpha_i)$ and formula Φ not containing any propositional variables occurring in $\mathtt{post}_j$:

$$f_b(\mathtt{prec}_j) \wedge \Phi \rightarrow [\alpha_i](f_b(\mathtt{post_j}) \wedge \Phi).$$

This is essentially a frame axiom for belief update actions.

A4 for every belief update action α_i where all possible preconditions in $C(\alpha_i)$ are $\mathtt{prec}_1, \ldots, \mathtt{prec}_k$:

$$\neg f_b(\mathtt{prec}_1) \wedge \ldots \wedge \neg f_b(\mathtt{prec}_k) \rightarrow \neg\langle\alpha_i\rangle\top$$

where $\top$ is a tautology. This axiom ensures that belief update actions cannot be performed in states that do not satisfy any of its preconditions.

A5 for every belief update action α_i and every precondition $\mathtt{prec}_j$ in $C(\alpha_i)$, $f_b(\mathtt{prec}_j) \rightarrow \langle\alpha_i\rangle\top$. This axiom ensures that belief update actions can be performed successfully when one of their preconditions holds.

Let us call the axiom system above $\mathbf{Ax_C}$ where, as before, $\mathbf{C}$ is the set of pre- and postconditions of basic actions.

Theorem 1. $\mathbf{Ax_C}$ *is sound and (weakly) complete for the class of regular models* $\mathbf{M_C}$.

Proof. Since our logic includes PDL, we cannot prove strong completeness (for every set of formulas Γ and formula ϕ, if $\Gamma \models \phi$ then $\Gamma \vdash \phi$) because PDL is not compact. Instead, we can prove weak completeness: every valid formula ϕ is derivable ($\models \phi \ \Rightarrow \vdash \phi$).

The proof of soundness is by straightforward induction on the length of a derivation. All axioms are clearly sound, and the inference rules are standard.

The proof of completeness is standard as far as the PDL part is concerned, see for example [6]. Take a consistent formula ϕ; we are going to build a finite satisfying model $M \in \mathbf{M_C}$ for ϕ.

We define the closure, $CL(\Sigma)$ of a set of formulas of our language based on the usual definition of the Fischer-Ladner closure under single negations of Σ. However we assume a special definition of subformula closure under which we do not permit the inclusion of propositional variables, e.g., if $Bp \in \Sigma$, then we do not allow p in the subformula closure of Σ, since we do not have bare propositional variables in our language. We also have an extra condition that if an action α occurs in ϕ, then $CL(\phi)$ contains $f_b(\psi)$ for all pre- and postconditions ψ for α.

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 α be a basic action. We define $R_\alpha(A,B)$ to hold if and only if the conjunction of formulas in A, $\wedge A$, is consistent with $\langle\alpha\rangle \wedge B$ (conjunction of formulas in B preceded by $\langle\alpha\rangle$).

The relations corresponding to complex programs ρ are defined inductively on top of the relations corresponding to basic actions using unions, compositions, and reflexive transitive closures, as is the case with regular models.

We define the assignment V 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 $G(-)p \in CL(\phi)$.

The truth lemma follows easily on the basis of the PDL completeness proof given in [6]; so we have that 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.

Clearly, beliefs and goals are disjoint because the states are consistent with respect to the axioms **A1** and **A2**.

All that remains to show is that this model M also satisfies the pre- and postconditions of the actions which occur in the formula ϕ: an action α is not applicable if none of its preconditions are satisfied, and if it is applied in a state s where one of its preconditions holds (recall that the preconditions are disjoint), then the corresponding postcondition holds in all states s' accessible from s by α.

First, consider an action α and state A such that A does not satisfy any of the preconditions of α. Then, by axiom **A4**, $\wedge A$ implies $[\alpha]\bot$, so there is no maximal consistent set B such that $\wedge A \wedge \langle\alpha\rangle \wedge B$ is consistent, so there is no α-transition from A.

Now suppose that A satisfies one of the preconditions $\texttt{prec}_j$ of α. Then $f_b(\texttt{prec}_j) \in A$ (recall that $CL(\phi)$ contains $f_b(\texttt{prec}_j)$, so we can use the truth lemma) and $\wedge A$ implies $[\alpha]f_b(\texttt{post}_j)$ by **A3**. For any B such that $\wedge A \wedge \langle\alpha\rangle \wedge B$ is consistent, B has to contain $f_b(\texttt{post}_j)$ since $f_b(\texttt{post}_j)$ is in $CL(\phi)$ and $\wedge A \wedge \langle\alpha\rangle \neg f_b(\texttt{post}_j)$ is not consistent, and such a successor B exists by **A5**. So every α-successor of A satisfies the postcondition. Similarly, we can show that for every literal q in A (in $CL(\phi)$), which does not occur in the postcondition $f_b(\texttt{post}_j)$, it is not consistent to assume that its value has changed in a state accessible by α (e.g. $Bq \wedge \langle\alpha\rangle \neg Bq$ is inconsistent), because of **A3**; so all literals in the state A which do not occur in the postcondition $f_b(\texttt{post}_j)$ do not change their value in the state accessible by α. Note that all other literals which do not occur in $CL(\phi)$ and by construction do not occur in the postconditions of any action occurring in $CL(\phi)$ are always absent in all states, so their value trivially does not change. □

5 Verification

In this section we show how to define exactly the set of paths in the transition system generated by the operational semantics which correspond to a PDL program expression. This allows us to verify properties of agent programs, such as 'all executions

of a given program result in a state satisfying property ϕ'. More precisely, we would like to express that, given the initial beliefs and goals of the agent, the application of its planning goal rules and the execution of the resulting plans reach states in which the agent has certain beliefs and goals.

We distinguish two types of properties of agent programs: safety properties and liveness properties. Let $\phi \in L_0$ denote the initial beliefs and goals of an agent and $\psi \in L_0$ denote states in which certain beliefs and goals hold (i.e., ϕ, ψ are formulas of L_0 containing only Bp and $G(-)q$ atoms). The general form of safety and liveness properties is then: $\phi \rightarrow [\ \xi(\Lambda)\ ]\psi$ and $\phi \rightarrow \langle \xi(\Lambda) \rangle \psi$, respectively, where $\xi(\Lambda)$ describes the execution of the agent's program with a set of planning goal rules Λ.

5.1 Expressing the non-interleaved strategy

The application of a set of planning goal rules $\Lambda = \{r_i | r_i = \kappa_i \leftarrow \beta_i | \pi_i\}$ for an agent with a non-interleaved execution strategy is translated as follows:

$$\xi(\Lambda) = (\bigcup_{r_i \in \Lambda} (f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+$$

where $^+$ is the strict transitive closure operator: $\rho^+ = \rho;\rho^*$. This states that each planning goal rule is be applied zero or more times (but at least one planning goal rule will be applied).

Using this definition of $\xi(\Lambda)$, the general schema of safety and liveness properties for an agent with an interleaved execution strategy are then:

$\phi \rightarrow [\ (\bigcup_{r_i \in \Lambda} (f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+\ ]\psi$ for safety properties; and
$\phi \rightarrow \langle\ (\bigcup_{r_i \in \Lambda} (f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+\ \rangle \psi$ for liveness properties.

Below we show that the translation above is faithful, namely the PDL program expression which is the translation of the agent's program corresponds to the set of paths in the transition system generated by the operational semantics for that agent program. But first we need a few extra definitions.

A model generated by a state s_0 consists of all possible states which can be recursively reached from s_0 by following the basic relations. A state s and a configuration $c = \langle \sigma, \gamma, \Pi \rangle$ are matching if they have the same belief and goal bases, that is $V_b(s) = \sigma$ and $V_g(s) = \gamma$. We denote this as $s \sim c$. Let $\mathbf{C}$ be a set of pre- and postconditions of belief update actions and Λ a set of planning goal rules. Let TS be a transition system defined by the operational semantics for an agent using the non-interleaved execution strategy (all possible configurations $\langle \sigma, \gamma, \Pi \rangle$ and transitions between them, given Λ and $\mathbf{C}$) and M a model belonging to $\mathbf{M_C}$. TS and M are called matching if they are generated by c_0 and s_0, respectively, such that $s_0 \sim c_0$.

We now prove a theorem which will allow us to verify properties of reachability in TS by evaluating formulas $\langle \xi(\Lambda) \rangle \phi$ at s_0.

Theorem 2. *Assume that TS is a transition system defined by the operational semantics for an agent with a set of planning goal rules Λ with pre- and postconditions for basic actions $\mathbf{C}$ using a non-interleaved execution strategy, and M is a model in $\mathbf{M_C}$. Then if TS and M match, then a configuration c with an empty plan base is reachable from the initial configuration c_0 in TS iff a state s matching c is reachable from the initial state s_0 (matching c_0) along a path described by $\xi(\Lambda)$, i.e., $(s_0,s) \in R_{\xi(\Lambda)}$.*

Before proving the theorem, we need the following lemma:

Lemma 1. *For all $s = \langle \sigma, \gamma \rangle$, $s' = \langle \sigma', \gamma' \rangle$, and plans π and π', we have: $\langle \sigma, \gamma, \{\pi; \pi'\} \rangle \longrightarrow \langle \sigma', \gamma', \{\pi'\} \rangle$ in TS iff $R_{f_p(\pi)}(s,s')$ in M.*

Proof of Lemma 1. By induction on the length of π.

Basis of induction: We prove that the lemma holds for belief update actions, and belief and goal test actions. Clearly, with respect to the belief update actions α, $\langle \sigma, \gamma, \{\alpha; \pi'\} \rangle \longrightarrow \langle \sigma', \gamma', \{\pi'\} \rangle$ in TS iff $R_\alpha(s,s')$ in M, by the operational semantics rule (1) and the definition of R_α in terms of pre- and postconditions. For belief tests $\phi?$ and goal tests $\psi!$, the relations $R_{f_b(\phi)?}$ and $R_{f_g(\psi)?}$ hold for exactly the same pairs (s,s) for which belief and goal test transitions hold by rules (2) and (3). This follows from Proposition 1.

Inductive step: Assume the lemma holds for the sub-plans of π.

Let $\pi =$ `if` ϕ `then` π_1 `else` π_2. Let us assume that $\sigma \models_{cwa} \phi$ and there is a transition from $\langle \sigma, \gamma, \{\pi; \pi'\} \rangle$ to $\langle \sigma, \gamma, \{\pi_1; \pi'\} \rangle$ by rule (4) in TS (the `else` case is similar). Then by Proposition 1, $M, s \models f_b(\phi)$, so $R_{f_b(\phi)?}(s,s)$. By the inductive hypothesis, there is a path from $\langle \sigma, \gamma, \{\pi_1; \pi'\} \rangle$ to $\langle \sigma', \gamma', \{\pi'\} \rangle$ iff $R_{f_p(\pi_1)}(s,s')$. Hence, $R_{f_b(\phi)?;f_p(\pi_1)}(s,s')$ and $R_{f_p(\pi)}(s,s')$. The other direction is similar. Assume that $R_{f_p(\pi)}(s,s')$ and $R_{f_b(\phi)?}(s,s)$ (the case of $R_{\neg f_b(\phi)?}(s,s)$ is identical). Then by Proposition 1, $\sigma \models_{cwa} \phi$ so by rule (4), there is a transition from $\langle \sigma, \gamma, \{\pi; \pi'\} \rangle$ to $\langle \sigma, \gamma, \{\pi_1; \pi'\} \rangle$ and from there by executing π_1 to $\langle \sigma', \gamma', \{\pi'\} \rangle$ (by the inductive hypothesis).

Let $\pi =$ `while` ϕ `do` π_1. Assume that there is a path in TS between $\langle \sigma, \gamma, \{\pi; \pi'\} \rangle$ and $\langle \sigma', \gamma', \{\pi'\} \rangle$. Note that from the rules (6) and (7) we can conclude that $\sigma' \not\models_{cwa} \phi$. By Proposition 1, $M, s' \models \neg f_b(\phi)$, so $R_{\neg f_b(\phi)}(s,s)$. Consider the path in TS; it is a sequence of configurations $\langle \sigma_1, \gamma_1, \{\pi; \pi'\} \rangle$, $\langle \sigma_2, \gamma_2, \{\pi_1; \pi; \pi'\} \rangle$, $\ldots$, $\langle \sigma_n, \gamma_n, \{\pi'\} \rangle$, where $(\sigma_1, \gamma_1) = (\sigma, \gamma)$, $(\sigma_n, \gamma_n) = (\sigma', \gamma')$ and one of the two cases holds. Either $n = 2$, so the path is of the form $\langle \sigma, \gamma, \{\pi; \pi'\} \rangle$, $\langle \sigma, \gamma, \{\pi'\} \rangle$. In this case (σ, γ) and (σ', γ') are the same (that is, $s = s'$), $\sigma \not\models_{cwa} \phi$ (rule 7) and $R_{\neg f_b(\phi)?}(s,s')$.

Or, $n > 2$, so there is a chain of configurations connected by paths corresponding to the executions of π_1. In this case, for each $i < n$ it holds that $\langle \sigma_i, \gamma_i, \{\pi_1; \pi'\} \rangle$ has a path to $\langle \sigma_{i+1}, \gamma_{i+1}, \{\pi'\} \rangle$. But then by the inductive hypothesis, $R_{f_p(\pi_1)}(s_i, s_{i+1})$ and $R_{f_p(\pi_1)^*}(s,s')$, hence $R_{f_p(\pi)}(s,s')$. The other direction is similar.

This completes the proof that all paths corresponding to an execution of a plan π in TS are described by $f_p(\pi)$ in M. □ (of Lemma)

Proof of Theorem 2. Observe that in the operational semantics for the non-interleaved execution strategy, any path between two configurations with an empty plan base

consists of one or more cycles of executing one of the goal planning rules followed by the execution of the corresponding plan. We will prove the theorem by induction on the number of such cycles on the path between two configurations with empty plan bases, $\langle \sigma, \gamma, \{\} \rangle$ and $\langle \sigma', \gamma', \{\} \rangle$. We use the lemma above for the special case when π' is empty.

Basis of induction: the path involves one cycle. Suppose there is such a path in TS. This means that some planning goal rule $\phi \leftarrow \psi | \pi$ matched (so ϕ and ψ are true in $\langle \sigma, \gamma, \{\} \rangle$) and π was adopted, resulting in a configuration $\langle \sigma, \gamma, \{\pi\} \rangle$, and from that configuration there is a path to $\langle \sigma', \gamma', \{\} \rangle$. By the lemma, this means that there is a corresponding path in M from (σ, γ) to (σ', γ') described by $f_p(\pi)$. Since ϕ and ψ are true in (σ, γ), there is a path from (σ, γ) to itself by $(f_g(\psi) \wedge f_b(\phi))?$, from which follows that there is a path in M from (σ, γ) to (σ', γ') described by $(f_g(\psi) \wedge f_b(\phi))?; f_p(\pi)$.

The other direction: assume that in M, there is a path from (σ, γ) to (σ', γ') described by $(f_g(\psi) \wedge f_b(\phi))?; f_p(\pi)$. We need to show that in TS, there is a path from $\langle \sigma, \gamma, \{\} \rangle$ to $\langle \sigma', \gamma', \{\} \rangle$. Since in M, there exists a transition from (σ, γ) to itself along $(f_g(\psi) \wedge f_b(\phi))?$, this means that ϕ and ψ are entailed by the agent's belief and goal base by Proposition 1. This means that the corresponding planning goal rule will be applied in $\langle \sigma, \gamma, \{\} \rangle$ resulting in adoption of π (transition to the configuration $\langle \sigma, \gamma, \{\pi\} \rangle$). By the lemma, there is a path from $\langle \sigma, \gamma, \{\pi\} \rangle$ to $\langle \sigma', \gamma', \{\} \rangle$.

Inductive step: assume that any path of length $k-1$ between two configurations with empty plan bases has a corresponding path in M described by a path in $\xi(\Lambda)$, which means that it is described by an $k-1$-long concatenation of expressions of the form $(f_g(\psi) \wedge f_b(\phi))?; f_p(\pi)$, for example $(f_g(\psi_1) \wedge f_b(\phi_1))?; f_p(\pi_1); \ldots; (f_g(\psi_{k-1}) \wedge f_b(\phi_{n-1}))?; f_p(\pi_{k-1})$. By the argument in the basis step, the last (kth) segment corresponds to a path described by $(f_g(\psi_k) \wedge f_b(\phi_k))?; f_p(\pi_k)$. Hence, the whole path is described by

$$(f_g(\psi_1) \wedge f_b(\phi_1))?; f_p(\pi_1); \ldots; (f_g(\psi_k) \wedge f_b(\phi_k))?; f_p(\pi_k),$$

which is in $\xi(\Lambda)$. □ (of Theorem)

5.2 Expressing the interleaved strategy

For an agent with an interleaved execution strategy, we need a version of PDL with an additional interleaving operator, $\|$ [1]. Strictly speaking, the interleaving operator does not increase the expressive power of PDL, but it makes the language more concise (every formula containing the interleaving operator has an equivalent formula without, however the size of that formula may be doubly exponential in the size of the original formula, see [1]).

Note that we are able to view regular models $M = (S, \{R_\rho : \rho \in \Upsilon\}, V)$ as models of the form $M' = (S, \tau, V)$ where $\tau(\alpha_i) \subseteq (S \times S)$ gives us the set of state transitions for α_i such that $(s, s') \in \tau(\alpha_i)$ iff $R_{\alpha_i}(s, s')$. We can extend this inductively to give us

a set of *paths* $\tau(\rho) \subseteq (S \times S)^*$ in M corresponding to any PDL program expression ρ, including expressions with the interleaving operator $\rho_1 \parallel \rho_2$. By a path we mean a sequence $(s_1,s_2),(s_3,s_4),\ldots,(s_{n-1},s_n)$ $(n \geq 2)$ of pairs of states, where each pair is connected by an atomic action transition or a test transition. By a *legal* path we mean a path where for every even $i < n$ (the target of the transition), $s_i = s_{i+1}$ (the source of the next transition). Otherwise a path is called *illegal*. For example, $(s_1,s_2),(s_2,s_3)$ is a legal path and $(s_1,s_2),(s_3,s_4)$ where $s_2 \neq s_3$ is an illegal path.

Paths corresponding to PDL program expressions are defined as follows:

- $\tau(\phi?) = \{(s,s) : M,s \models \phi\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1;\rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, where $\circ$ is concatenation of paths; here we allow illegal paths $p_1 \circ p_2$ where $p_1 = (s_0,s_1)...(s_n,s_{n+1})$ and $p_2 = (t_0,t_1)...(t_m,t_{m+1})$, with $s_{n+1} \neq t_0$.
- $\tau(\rho^*)$ 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 paths from $\tau(\rho_1)$ and $\tau(\rho_2)$.

The reason why we need illegal paths for PDL with the interleaving operator can be illustrated by the following example. Let $(s_1,s_2) \in \tau(\alpha_1)$ and $(s_3,s_4) \in \tau(\alpha_2)$, with $s_2 \neq s_3$). Then the illegal path $(s_1,s_2),(s_3,s_4) \in \tau(\alpha_1;\alpha_2)$. Let $(s_2,s_3) \in \tau(\alpha_3)$. Then $(s_1,s_2),(s_2,s_3),(s_3,s_4)$ is obtained by interleaving a path from $\tau(\alpha_1;\alpha_2)$ and $\tau(\alpha_3)$, and it is a legal path in $\tau(\alpha_1;\alpha_2 \parallel \alpha_3)$. Note that if the paths above are the only paths in $\tau(\alpha_1),\tau(\alpha_2)$ and $\tau(\alpha_3)$, then using an illegal path in $\tau(\alpha_1;\alpha_2)$ is the only way to define a legal interleaving in $\tau(\alpha_1;\alpha_2 \parallel \alpha_3)$.

We define the relation $\models$ of a formula being true in a state of a model as:

- $M,s \models Bp$ iff $p \in V_b(s)$
- $M,s \models G(-)p$ iff $(-)p \in V_g(s)$
- $M,s \models \neg\phi$ 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 $M,s' \models \phi$.
- $M,s \models [\rho]\phi$ iff for all legal paths $\tau(\rho)$ starting in s, the end state s' of the path satisfies ϕ: $M,s' \models \phi$.

In this extended language, we can define paths in the execution of an agent with an interleaved execution strategy and planning goal rules as

$$\xi^i(\Lambda) = \bigcup_{\Lambda' \subseteq \Lambda, \Lambda' \neq \emptyset} \|_{r_i \in \Lambda'} ((f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+$$

Theorem 3. *Assume that TS is a transition system defined by the operational semantics for an agent with a set of planning goal rules Λ with pre- and postconditions for basic actions* **C** *using an interleaved execution strategy, and M is a model in* $\mathbf{M_C}$. *Then if TS and M match, then a configuration c with an empty plan base is*

reachable from the initial configuration c_0 in TS iff a state s matching c is reachable from the initial state s_0 (matching c_0) along a path in $\tau(\xi^i(\Lambda))$.

Proof. In order to prove the theorem, we need to show a correspondence between finite paths in TS and M. By a path in TS from c_0 to c, we will mean a legal path $(c_0,c_1),(c_1,c_2),\ldots,(c_{n-1},c_n)$ where $c_n = c$, such that for every pair (c_i,c_i+1) on the path, there is a transition from c_i to c_{i+1} described by one of the operational semantics rules (1^i) - (8^i). By a path in M from s_0 to s we will mean a legal path $(s_0,s_1),(s_1,s_2)\ldots,(s_{n-1},s_n)$ where $s_n = s$ such that for each step $p_i = (s_i,s_{i+1})$ on the path, s_i and s_{i+1} are in some R_α or $R_{\phi?}$ relation (in the latter case $s_i = s_{i+1}$). We will refer to α or $\phi?$ as the label of that step and denote it by $label(p_i)$. By a path in M from s_0 to s which is in $\tau(\xi^i(\Lambda))$ we will mean a path from s_0 to s where the labels on the path spell a word in $(\xi^i(\Lambda))$.

We will denote the steps on the path by $p_0,p_1,\ldots,p_{n-1}$, and refer to the first component of the pair p_i as p_i^0 and to the second component as p_i^1. If $p_i = (s_i,s_{i+1})$, then $p_i^0 = s_i$ and $p_i^1 = s_{i+1}$. Note that the same state can occur in different steps, and we want to be able to distinguish those occurrences. Since the path is legal, for all i, $p_i^1 = p_{i+1}^0$.

As an auxiliary device in the proof, we will associate with each component p_i^j ($j \in \{0,1\}$) of each step p_i on the path in M a history $\rho(p_i^j)$ and a set of 'execution points' $E(p_i^j)$.

A history is a PDL program expression which is a concatenation of labels of the previous steps on the path. For example, consider a path $p_0 = (s_0,s_0), p_1 = (s_0,s_2)$ where $R_{\phi?}(s_0,s_0)$ and $R_\alpha(s_0,s_2)$. Then the history $\rho(p_0^0)$ is an empty string, $\rho(p_0^1) = \rho(p_1^0) = \phi?$ and the history $\rho(p_1^1) = \phi?;\alpha$. For an arbitrary point p_i^j on a path in $\tau(\xi^i(\Lambda))$, the history describes a prefix of a path in $\tau(\xi^i(\Lambda))$. Note that $\xi^i(\Lambda)$ is a union of $\|_{r_i\in\Lambda'}\ ((f_g(\kappa_i)\wedge f_b(\beta_i))?;f_p(\pi_i))^+$, so the history will consist of an interleaving of tests and actions which come from tracing expressions of the form $(f_g(\kappa_i)\wedge f_b(\beta_i))?;f_p(\pi_i)$ for $r_i \in \Lambda' \subseteq \Lambda$, some of them repeated several times. At the step where all the plan expressions $f_p(\pi_i)$ have been traced to their ends, the history describes a path in $\tau(\xi^i(\Lambda))$. Conversely, if the history in the last step of the path describes a path in $\tau(\xi^i(\Lambda))$, then the path is in $\tau(\xi^i(\Lambda))$.

A set of execution points $E(p_i^j)$ will contain execution points, which are PDL plan expressions of the form $f_p(\pi)$, where π is either a translation of some complete plan π_i for some $r_i \in \Lambda$, or of a suffix of such a plan. Intuitively, they correspond to a set of (partially executed) plans in the plan base corresponding to p_i^j, and are called execution points because they tell us where we are in executing those plans. We annotate p_i^j with sets of execution points using the following simple rule. When we are in p_i^0 and $E(p_i^0) = \{f_p(\pi_1),\ldots,f_p(\pi_k)\}$, then exactly one of the following three options apply. Either $label(p_i) = \alpha$, and one of the $f_p(\pi_j)$ is of the form $f_p(\alpha;\pi')$, in which case in $E(p_i^1)$, $f_p(\alpha;\pi')$ is replaced by $f_p(\pi')$. Or, $label(p_i) = \phi?$, and then one of the two possibilities apply. Either $\phi? = f_b(\beta?)$ and one of the $f_p(\pi_j)$ is of the form $f_p(\beta?;\pi')$, in which case in $E(p_i^1)$, $f_p(\beta?;\pi')$ is replaced by $f_p(\pi')$. Or, $\phi? = (f_g(\kappa_m!)\wedge f_b(\beta_m?))?$ (the test corresponding to a planning goal rule r_m) and $E((p_i^1) = E(p_i^0)\cup\{f_p(\pi_m)\}$. Essentially we take the first step α or $\phi?$ in tracing

one of the expressions in $E(p_i^0)$, remove it, and append it to the history in the next state. It is clear that if the sets of execution points along the path correspond to plan bases in the operational semantics, then the histories correspond to prefixes of paths in $\tau(\xi^i(\Lambda))$. Also, if on such a path $E(p_i^j) = \emptyset$ for $i > 0$, then $\rho(p_i^j)$ describes a path in $\tau(\xi^i(\Lambda))$.

We say that a set of execution points $E(p_i^j)$ and a plan base Π match if the execution points in $E(p_i^j)$ are the translations of plans in Π, that is $\Pi = \{\pi_1, \ldots, \pi_k\}$ and $E(p_i^j) = \{f_p(\pi_1), \ldots, f_p(\pi_k)\}$.

The idea of the proof is as follows. We show that

($TS \Rightarrow M$) For every path in TS from c_0 to a configuration c with an empty plan base, we can trace, maintaining a set of execution points and a history, a path in M from s_0 to a state s such that $s \sim c$ and the last step on the path has an empty set of execution points and a history in $\tau(\xi^i(\Lambda))$. This will show one direction of the theorem, that every path in TC has a corresponding path in M.

($M \Rightarrow TS$) States on every path in M from s_0 to s which is in $\tau(\xi^i(\Lambda))$ can be furnished with sets of execution points which correspond to plan bases of configurations on a corresponding path from c_0 to c such that $s \sim c$. This shows another direction of the theorem, that if we have a path in $\tau(\xi^i(\Lambda))$, we can find a corresponding path in TS.

To prove ($TS \Rightarrow M$), we first note that s_0 and c_0 have a matching set of execution points and plan base (empty). Then we consider a pair $s \sim c$ where the plan base in c and the set of execution points in $p_{n-1}^1 = s$ match, and show that if we can make a transition from c, then we can make a step $p_n = (s, s')$ from s, and end up with a matching state-configuration pair $s' \sim c'$ where $E(p_n^1)$ matches Π'. Note that if we match the plan base and the set of execution points at each step, and update the execution step according to the rule, then the history in guaranteed to be a prefix of a path in $\tau(\xi^i(\Lambda))$ (or a path in $\tau(\xi^i(\Lambda))$ if the set of execution points is empty).

Let us consider possible transitions from c. Either, a planning goal rule is applied, or one of the plans in Π is chosen and a suitable transition rule applied to execute its first step.

If a planning goal rule r_m is applied, then clearly the belief and goal conditions of r_m hold in c so by assumption they hold in s, hence in s, there is a transition by $R_{(f_g(\kappa_m) \wedge f_b(\beta_m))?}$ to the same s with the same belief and goal base. The transition in TS goes to c' with the same belief and goal base as c and Π' extended with π_m. In M, we make a step along the path to $s' = s$ and add $f_p(\pi_m)$ to the set of execution points $E(p_n^1)$. Clearly, Π' and $E(p_n^1)$ match.

If the transition rule corresponds to executing one of the plans π_i in Π, then we have the following cases.

- $\pi_i = \alpha; \pi$: we execute a belief update action α and transit to c'. This is possible only if in M there is an R_α transition to s' such that $s' \sim c'$. In c' in the plan base Π' we have π instead of $\alpha; \pi$. In the set of execution points for the next step $p_n^1 = s'$ we have $f_p(\pi)$ instead of $\alpha; f_p(\pi)$. Clearly, Π' and $E(p_n^1)$ match.

- $\pi_i = \beta?;\pi$ or $\pi_i = \kappa!;\pi$. A transition from c to a configuration c' where the plan base contains π instead of π_i is possible if and only if the test succeeds in σ, so by Proposition 1 if and only if in M there is a corresponding $R_{f_b(\beta)?}$ or $R_{f_g(\kappa)?}$ transition from s to itself, so $p_n = (s,s)$, $s \sim c'$, the execution point $E(p_n^1)$ contains $f_p(\pi)$ instead of $f_p(\pi_i)$, so Π' and $E(p_n^1)$ match.
- $\pi_i = (\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2);\pi$ and in $E(p_n^0)$ we have $(f_b(\phi)?;f_p(\pi_1)) \cup (\neg f_b(\phi)?;\ f_p(\pi_2));f_p(\pi)$. Since $s \sim c$ either ϕ is true in both s and c or $\neg\phi$. Assume that ϕ is true (the case for ϕ false is analogous). In TS we transit to c' with $\pi_1;\pi$ in the plan base and in M we transit to s' by executing the test on the left branch of $\cup$ and replace $f_p(\pi_i)$ in the set of execution points with $f_p(\pi_1);f_p(\pi) = f_p(\pi_1;\pi)$.
- $\pi_i = \texttt{while}\ \phi\ \texttt{do}\ \pi_1;\pi$ and in $E(p_n^0)$ we have $(f_b(\phi)?;f_p(\pi_1))^*;\neg f_b(\phi)?;f_p(\pi)$. Since $s \sim c$ we have ϕ either true or false in both. If it is false then we transit in TS to c' with π in the plan base, and in M there is a $\neg f_b(\phi)?$ transition to s itself, but now we replace $(f_b(\phi)?;f_p(\pi_1))^*;\neg f_b(\phi)?;f_p(\pi)$ in the set of execution points with $f_p(\pi)$. If ϕ is true, then by the rule (6^i) we go to c' with Π' containing $\pi_1;\texttt{while}\ \phi\ \texttt{do}\ \pi_1;\pi$ and by $f_b(\phi)?$ in M we go to s' with the set of execution points containing $f_p(\pi_1);(f_b(\phi)?;f_p(\pi_1))^*;\neg f_b(\phi)?;f_p(\pi)$. Note that the new set of execution points and Π' match because $f_p(\pi_1);(f_b(\phi)?;f_p(\pi_1))^*;\neg f_b(\phi)?;f_p(\pi)$ is the same as $f_p(\pi_1;\texttt{while}\ \phi\ \texttt{do}\ \pi_1;\ \pi)$.

This achieves the desired correspondence in the direction from the existence of a path in the operational semantics to the existence of a corresponding path in the model; it is easy to check that the path to s' is described by its history, and that when we reach a state s corresponding to a configuration with an empty plan base, its set of execution points is also empty and the history describes a path in $\tau(\xi^i(\Lambda))$.

Let us consider the opposite direction ($M \Rightarrow TS$). Suppose we have a path in M from s_0 to s which is in $\tau(\xi^i(\Lambda))$. We need to show that there is a corresponding path in the operational semantics. For this direction, we only need to decorate each components of a step on the path in M with a set of execution points corresponding to the plan base in the matching configuration c'. (We do not need the history because we already know that the path in M is in $\tau(\xi^i(\Lambda))$.)

Clearly, in the initial state s_0, the set of execution points is empty, and $s_0 \sim c_0$. Now we assume that we have reached s and c such that $s \sim c$ and $E(p_{n-1}^1)$ and Π match, where p_{n-1} is the last step on the path and $p_{n-1}^1 = s$. Now we have to show how for the next step along the path in M to a state s' we can find a transition in TS to a configuration c' so that $s' \sim c'$ and the set of execution points in s' matches Π'.

Our task is made slightly harder by the fact that if we encounter, for example, a test transition on a path in M, we do not necessarily know whether it is a test corresponding to firing a new planning goal rule, or is an `if` or a `while` test in one of the plans (there could be several plan expressions in $E(p_n^0)$ starting with the same test $f_b(\phi?)$, for example). Similarly, if we have an α transition on the path, several plan expressions in $E(p_n^0)$ may start with an α (for example $E(p_n^0) = \{\alpha;\alpha_1,\ \alpha;\alpha_2\}$), so the question is how do we find a corresponding configuration in the operational semantics. In the example above, the plan base could be either $\Pi_1' =$

$\{\alpha_1,\ \alpha;\alpha_2\}$ or $\Pi'_2 = \{\alpha;\alpha_1,\ \alpha_2\}$. However, note that the path we are trying to match is in $\tau(\xi^i(\Lambda))$, so it contains transitions corresponding to a complete interleaved execution of all plans currently in the set of execution points until the end. So we can look ahead at the rest of the path and annotate ambiguous transitions with the corresponding plan indices. In the example above, if the next transition is α_1, we know that the ambiguous α belongs to the first plan; if the next transition is α_2, we annotate the ambiguous α with the index of the second plan; if the next transition is α, then there are two possible matching paths in TS, and we can pick one of them non-deterministically, as both α's have the same effect on the belief and goal base, for example annotate the first α with the index of the first plan and the second α with the index of the second plan.

Once we have indexed the ambiguous transitions, finding the corresponding path in TS can be done in exactly the same way as in the proof for $(TS \Rightarrow M)$. □

6 Example of using theorem proving to verify properties of an agent program

In this section we briefly illustrate how to prove properties of agents in our logic, using the vacuum cleaner agent as an example. We will use the following abbreviations: c_i for `clean`$_i$, r_i for `room`$_i$, b for `battery`, s for `suck`, c for `charge`, r for `moveR`, l for `moveL`. The agent has the following planning goal rules:

c_1 `<-` b `|` `if` r_1 `then` $\{s\}$ `else` $\{l;\ s\}$
c_2 `<-` b `|` `if` r_2 `then` $\{s\}$ `else` $\{r;\ s\}$
`<-` $-b$ `|` `if` r_2 `then` $\{c\}$ `else` $\{r;\ c\}$

Under the non-interleaved execution strategy, these planning goal rules can be translated as the following PDL program expression:

$$\begin{aligned}\texttt{vac} =_{df}\ & ((Gc_1 \wedge Bb)?;(Br_1?;s) \cup (\neg Br_1?;l;s)) \cup \\ & ((Gc_2 \wedge Bb)?;(Br_2?;s) \cup (\neg Br_2?;r;s)) \cup \\ & (\neg Bb?;(Br_2?;c) \cup (\neg Br_2?;r;c))\end{aligned}$$

Given appropriate pre- and postconditions for the belief update actions in the example program (such as the pre- and postconditions of `moveR`, `moveL`, `charge` and `suck` given earlier in the paper), some of the instances of A3–A5 are:

A3r $Bc_1 \wedge Br_1 \wedge \neg Bb \wedge Gc_2 \rightarrow [r](Br_2 \wedge Bc_1 \wedge \neg Bb \wedge Gc_2)$
A3s1 $Gc_1 \wedge Gc_2 \wedge Br_1 \wedge Bb \rightarrow [s](Bc_1 \wedge \wedge Gc_2 \wedge Br_1 \wedge \neg Bb)$
A3s2 $Gc_1 \wedge Gc_2 \wedge Br_2 \wedge Bb \rightarrow [s](Gc_1 \wedge \wedge Bc_2 \wedge Br_2 \wedge \neg Bb)$
A3c $Br_2 \wedge Bc_1 \wedge \neg Bb \wedge Gc_2 \rightarrow [c](Br_2 \wedge Bc_1 \wedge Bb \wedge Gc_2)$
A4r $\neg Br_1 \rightarrow \neg\langle r\rangle\top$
A5s $Br_1 \wedge Bb \rightarrow \langle s\rangle\top$.

Using a PDL theorem prover such as MSPASS [23] (for properties without $*$) or PDL-TABLEAU [28], and instances of axioms A1-A5 such as those above, we can prove a liveness property that if the agent has goals to clean rooms 1 and 2, and starts in the state where its battery is charged and it is in room 1, it can reach a state where both rooms are clean, and a safety property that it is guaranteed to achieve its goal:

$$Gc_1 \wedge Gc_2 \wedge Bb \wedge Br_1 \rightarrow \langle \mathtt{vac}^3 \rangle (Bc_1 \wedge Bc_2)$$

$$Gc_1 \wedge Gc_2 \wedge Bb \wedge Br_1 \rightarrow [\mathtt{vac}^3] (Bc_1 \wedge Bc_2)$$

where $\mathtt{vac}^3$ stands for vac repeated three times. The MSPASS encoding of the first property is given in Appendix 9.1. Its verification using the web interface to MSPASS is virtually instantaneous.

We can also prove, using PDL-TABLEAU, a version of a blind commitment property which states that an agent either keeps its goal or believes it has been achieved:

$$Gc_1 \rightarrow [\mathtt{vac}^+] (Bc_1 \vee Gc_1)$$

In the appendix we split the proof of this property into two parts to simplify the PDL-TABLEAU encoding: first we prove using MSPASS $Bc_1 \vee Gc_1 \rightarrow [\mathtt{vac}](Bc_1 \vee Gc_1)$ (see Appendix 9.2) and then prove blind commitment using this as a lemma (see Appendix 9.3).

The theorem prover encodings given in the appendix are produced by hand, but this process can be automated at the cost of making the encoding larger (at worst, exponential in the size of the agent's program). In the remainder of this section we sketch a naive approach to automating the encoding. Axioms A1, A2, A4 and A5 are straightforward. For every literal l occurring in the agent program, we can generate an instance of axioms A1 and A2. For every belief update action α occurring in the agent's program, we generate an instance of A4, and for each precondition of α, we generate an instance of A5. The difficult case is the axiom schema A3, which is a kind of frame axiom. It includes a formula Φ which intuitively encodes the information about the state which *does not* change after executing an action α. To generate a sufficient number of instances of A3 automatically, we have to use all possible complete state descriptions for Φ (more precisely, all combinations of the agent's beliefs and goals which are not affected by α). Then the instances of A3 will say, for every complete description of the agent's beliefs and goals, what the state of the agent after the performance of α will be. Clearly, this makes the encoding exponential in the number of possible beliefs and goals of the agent. In the vacuum cleaner example, the properties of the agent's state are: its belief about its location, its beliefs about the cleanliness of the two rooms, its belief about its battery, and two possible goals. Even with the domain axioms $\neg(Br_1 \wedge Br_2)$ and $Br_1 \vee Br_2$ (the agent is never in both rooms at the time and it is always in one of the rooms) which reduce the number of possible beliefs about the agent's location to 2, the number of possible belief states is $2^4 = 16$ and the number of possible combined belief and goal states is $2^6 = 64$, requiring 64 instances of A3 for every action α. Note that this naive approach to automatisation of encoding more or less reduces the theorem proving

task to a model-checking task (we use A3, A4 and A5 to specify the transition relation, and the number of instances of A3 is equal to the number of entries in the transition table). However, various ways of reducing the number of axiom instances can be used. For example, we may use an approach similar to slicing [9], or adopt a more efficient way of expressing frame conditions, following the work in situation calculus [27] or, for a language without quantification over actions, [16].

The example above illustrates verification of a program under the non-itnterleaved execution strategy. For verifying program under the interleaved strategy, a PDL theorem prover would need to be adapted to accept program expressions which contain the interleaving operator. Alternatively, the program expression containing the interleaving operator would need to be translated into PDL without interleaving.

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 [30, 29] 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 [19], 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., [21, 14, 12], have yet to result in an automated verification procedure.

There has also been considerable work on the automated verification of multi-agent systems using model-checking [5, 24]. For example, in [9, 8], 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 semantics 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 [26, 4, 31]. 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 an extended and revised version of [2]. It is also closely related to our previous work on using theorem proving techniques to verify agent deliberation strategies [3]. However in that work a fixed general execution strategy is constrained by the execution model to obtain different execution strategies,

rather than different execution strategies being specified by different PDL program expressions as in this paper.

8 Conclusion

In this paper, we proposed a sound and complete logic which allows the specification of safety and liveness properties of SimpleAPL agent programs as well as their verification using theorem proving techniques. Our logic is a variant of PDL, and allows the specification of safety and liveness properties of agent programs. Our approach allows us to capture the agent's execution strategy in the logic, and we proved a correspondence between the operational semantics of SimpleAPL and the models of the logic for two example execution strategies. We showed how to translate agent programs written in SimpleAPL into expressions of the logic, and gave an example in which we show how to verify correctness properties for a simple agent program using theorem-proving. While we focus on APL-like languages and consider only single agent programs, our approach can be generalised to other BDI-based agent programming languages and the verification of multi-agent systems.

In future work, we would like to develop this verification framework further to deal with agent programming languages extended with plan revision mechanisms.

Acknowledgements

We would like to thank to Renate Schmidt for help with MSPASS and PDL-TABLEAU. Natasha Alechina and Brian Logan were supported by the Engineering and Physical Sciences Research Council [grant number EP/E031226].

References

1. Abrahamson, K.R.: Decidability and expressiveness of logics of processes. Ph.D. thesis, Department of Computer Science, University of Washington (1980)
2. Alechina, N., Dastani, M., Logan, B., Meyer, J.J.C.: A logic of agent programs. In: Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007), pp. 795–800. AAAI Press (2007)
3. Alechina, N., Dastani, M., Logan, B., Meyer, J.J.C.: Reasoning about agent deliberation. In: G. Brewka, J. Lang (eds.) Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), pp. 16–26. AAAI, Sydney, Australia (2008)
4. Astefanoaei, L., Dastani, M., de Boer, F., Meyer, J.J.C.: A verification framework for normative multi-agent systems. In: In the proceedings of The 11th Pacific Rim International Conference on Multi-Agents (PRIMA 2008), *LNCS*, vol. 5357. Springer (2008)
5. Benerecetti, M., Giunchiglia, F., Serafini, L.: Model checking multiagent systems. J. Log. Comput. **8**(3), 401–423 (1998)

6. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, *Cambridge Tracts in Theoretical Computer Science*, vol. 53. Cambridge University Press (2001)
7. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A.: Multi-Agent Programming: Languages, Platforms and Applications. Springer, Berlin (2005)
8. Bordini, R.H., Dennis, L.A., Farwer, B., Fisher, M.: Directions for agent model checking. In: this volume, Chapter 4
9. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multi-Agent Systems **12**(2), 239–256 (2006)
10. Bordini, R.H., Hübner, J.F., Vieira, R.: ***Jason*** and the Golden Fleece of agent-oriented programming. In: R.H. Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni (eds.) Multi-Agent Programming: Languages, Platforms and Applications, chap. 1. Springer-Verlag (2005)
11. Dastani, M.: 2APL: a practical agent programming language. Autonomous Agents and Multi-Agent Systems **16**(3), 214–248 (2008)
12. Dastani, M., Meyer, J.J.: Correctness of multi-agent programs: A hybrid approach. In: this volume, Chapter 6
13. Dastani, M., Meyer, J.J.C.: A practical agent programming language. In: M. Dastani, A.E. Fallah-Seghrouchni, A. Ricci, M. Winikoff (eds.) Proceedings of the Fifth International Workshop on Programming Multi-agent Systems (ProMAS'07), *LNCS*, vol. 4908, pp. 107–123. Springer (2008)
14. Dastani, M., van Riemsdijk, B., Meyer, J.J.C.: A grounded specification language for agent programs. In: Proc. of AAMAS 2007. ACM Press (2007). (to appear)
15. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.J.C.: A programming language for cognitive agents: Goal directed 3APL. In: Proc. ProMAS 2003, *LNCS*, vol. 3067, pp. 111–130. Springer (2004)
16. van Ditmarsch, H.P., Herzig, A., De Lima, T.: Optimal regression for reasoning about knowledge and actions. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007), pp. 1070–1075. AAAI Press (2007)
17. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge, Mass. (1995)
18. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18**(2), 194–211 (1979)
19. Fisher, M.: MetateM: The story so far. In: Proc. ProMAS 2005, *LNCS*, vol. 3862, pp. 3–22. Springer (2006)
20. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press (2000)
21. Hindriks, K., Meyer, J.J.C.: Agent logics as program logics: Grounding KARO. In: Proc. 29th German Conference on AI (KI 2006), *LNAI*, vol. 4314. Springer (2007)
22. van der Hoek, W., Wooldridge, M.: Towards a logic of rational agency. Logic Journal of the IGPL **11**(2), 133–157 (2003)
23. Hustadt, U., Schmidt, R.A.: MSPASS: Modal reasoning by translation and first-order resolution. In: Proc. TABLEAUX 2000, *LNCS*, vol. 1847, pp. 67–71. Springer (2000)
24. Lomuscio, A., Raimondi, F.: Mcmas: A model checker for multi-agent systems. In: Proc. TACAS 2006, pp. 450–454 (2006)
25. Rao, A.S.: Agentspeak(l): BDI agents speak out in a logical computable language. In: MAAMAW'96: Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world: agents breaking away, pp. 42–55. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1996)
26. van Riemsdijk, M., de Boer, F., Dastani, M., Meyer, J.J.C.: Prototyping 3apl in the maude term rewriting language. In: Proceedings of the seventh International Workshop on Computational Logic in Multi-Agent Systems (CLIMA 2006), *LNAI*, vol. 4371. Springer (2007)
27. Scherl, R.B., Levesque, H.J.: Knowledge, action, and the frame problem. Artif. Intell. **144**(1-2), 1–39 (2003)
28. Schmidt, R.A.: PDL-TABLEAU (2003). `http://www.cs.man.ac.uk/~schmidt/pdl-tableau`
29. Shapiro, S., Lesperance, Y., Levesque, H.: The cognitive agent specification language and verification environment. In: this volume, Chapter 9

30. Shapiro, S., Lespérance, Y., Levesque, H.J.: The cognitive agents specification language and verification environment for multiagent systems. In: Proc. AAMAS 2002, pp. 19–26. ACM Press (2002)
31. van Riemsdijk, M.B., Astefanoaei, L., de Boer, F.S.: Using the Maude term rewriting language for agent development with formal foundations. In: this volume, Chapter 11

9 Appendix: Encodings of properties in MSPASS

9.1 MSPASS encoding of the example

```
begin_problem(PDL_vacuum_cleaner_example 1).
list_of_descriptions.
 name(* PDL vacuum cleaner example 1 *).
 author(*N. Alechina, M. Dastani, B. Logan, and J.-J. Ch. Meyer *).
 description(* A formula which says that if the vacuum cleaner agent
       starts in room 1 with charged battery and its goal is to clean
       room 1 and room 2, then it will achieve its goals.
       *).
end_of_list.

list_of_symbols.
% Rr - moveRight, Rl - moveLeft, Rs - suck, Rc - charge,
% br1 - believes that in room1, br2 - in room2, bb - battery charged,
% bc1 - believes room1 clean, bc2 - room2 clean,
% gc1 - goal to clean room1, gc2 - clean room2.
predicates[ (Rr,2), (r,0), (Rl,2), (l,0), (Rs,2), (s,0), (Rc,2), (c,0),
           (br1,0), (br2,0), (bb,0), (bc1,0), (bc2,0), (gc1,0),
           (gc2,0) ].
% The following interprets dia(r,...) as accessible by Rr,
% dia(l,...) as accessible by Rl, etc.
translpairs[ (r,Rr), (l,Rl), (s, Rs), (c,Rc) ].
end_of_list.

list_of_special_formulae(axioms, eml).
% instances of A3
prop_formula(
 implies(and(gc1, gc2, br1, bb), box(s, and(bc1, gc2, br1, not(bb))))
).
prop_formula(
 implies(and(bc1, br1, not(bb), gc2),
             box(r, and(br2, bc1, not(bb), gc2)))
).
prop_formula(
 implies(and(br2, bc1, not(bb), gc2), box(c, and(br2, bc1, bb, gc2)))
).
prop_formula(
 implies(and(br2, bc1, bb, gc2), box(s, and(br2, bc1,not(bb), bc2)))
).

% instances of A5
```

```
prop_formula(
 implies(bb, dia(s, true))
).
prop_formula(
 implies(br1, dia(r, true))
).
prop_formula(
 implies(br2, dia(l, true))
).
prop_formula(
 implies(and(br2, not(bb)), dia(c, true))
).
end_of_list.

% The formula we want to prove below is
% gc1 & gc2 & br1 & bb -> <vac><vac><vac> (bc1 & bc2)
% where vac is the vacuum cleaner's program:
% (gc1?; bb?; (br1?;s) U ((not br1)?;l;s)) U
% (gc2?; bb?; (br2?;s) U ((not br2)?;r;s)) U
% ( (not bb)?; (br2?;c) U ((not br2)?;r;c)))

list_of_special_formulae(conjectures, EML).
prop_formula(
 implies(
   and (gc1, gc2, br1, bb),
   dia(
     or(
       % rule1
       comp(test(gc1),
            comp(test(bb),
                 or(comp(test(br1), s),
                    comp(test(not (br1)), comp(l,s))))),
       % rule2
       comp(test(gc2),
            comp(test(bb),
                 or(comp(test(br2), s),
                    comp(test(not (br2)), comp(r,s))))),
       % rule 3
       comp(test(not(bb)),
            or(comp(test(br2), c),
               comp(test(not(br2)), comp(r, c))))
     ), % end first vac or
     dia(
       or(
         % rule1
         comp(test(gc1),
              comp(test(bb),
                   or(comp(test(br1), s),
                      comp(test(not (br1)), comp(l,s))))),
         % rule2
         comp(test(gc2),
              comp(test(bb),
                   or(comp(test(br2), s),
                      comp(test(not (br2)), comp(r,s))))),
```

```
         % rule 3
         comp(test(not(bb)),
              or(comp(test(br2), c),
                 comp(test(not(br2)), comp(r, c))))
        ), % end second vac or
        dia(
          or(
            % rule1
            comp(test(gc1),
                 comp(test(bb),
                      or(comp(test(br1), s),
                         comp(test(not (br1)), comp(l,s))))),
            % rule2
            comp(test(gc2),
                 comp(test(bb),
                      or(comp(test(br2), s),
                         comp(test(not (br2)), comp(r,s))))),
            % rule 3
            comp(test(not(bb)),
                 or(comp(test(br2), c),
                    comp(test(not(br2)), comp(r, c))))
          ), % end third vac or
          and(bc1, bc2))))
 ) % end implies
).
end_of_list.

end_problem.
```

9.2 MSPASS encoding of a lemma for the proof of the blind commitment property of the vacuum cleaner agent

```
begin_problem(PDL_vacuum_cleaner_example3).
list_of_descriptions.
 name(* PDL example 3 *).
 author(*N. Alechina, M. Dastani, B. Logan and J.-J. Ch. Meyer*).
 description(* A formula which says that if we start with bc1 or gc1,
               then after each iteration of the program, bc1 or gc1.
       *).
end_of_list.

list_of_symbols.
% Rr - moveRight, Rl - moveLeft, Rs - suck, Rc - charge,
% br1 - believes that in room1, br2 - in room2, bb - battery charged,
% bc1 - believes room1 clean, bc2 - room2 clean,
% gc1 - goal to clean room1, gc2 - clean room2.
predicates[ (Rr,2), (r,0), (Rl,2), (l,0), (Rs,2), (s,0), (Rc,2), (c,0),
           (br1,0), (br2,0), (bb,0), (bc1,0), (bc2,0), (gc1,0), (gc2,0)].
% The following interprets dia(r,...) as accessible by Rr,
% dia(l,...) as accessible by Rl, etc.
```

```
translpairs[ (r,Rr), (l,Rl), (s, Rs), (c,Rc) ].
end_of_list.

list_of_special_formulae(axioms, eml).
% world axioms
prop_formula(
 not(and(br1,br2))
).
prop_formula(
 or(br1,br2)
).
% instances of A2
prop_formula(
 not(and(gc1,bc1))
).
prop_formula(
 not(and(gc2,bc2))
).
% instances of A3
prop_formula(
 implies(and(gc1, bb), box(s, or(bc1, gc1)))
).
prop_formula(
 implies(and(bc1, bb), box(s, or(bc1, gc1)))
).
prop_formula(
 implies(and(bc1, br1), box(r, and(bc1, br2)))
).
prop_formula(
 implies(and(gc1, br1), box(r, and(gc1, br2)))
).
prop_formula(
 implies(and(bc1, br2,not(bb)), box(c, and(bc1, br2, bb)))
).
prop_formula(
 implies(and(gc1, br2,not(bb)), box(c, and(gc1, br2, bb)))
).
prop_formula(
 implies(and(gc1, br2), box(l, and(gc1, br1)))
).
prop_formula(
 implies(and(bc1, br2), box(l, and(bc1, br1)))
).
% instances of A4
prop_formula(
 implies(not(bb), not(dia(s, true)))
).
prop_formula(
 implies(not(br1), not(dia(r, true)))
).
prop_formula(
 implies(not(br2), not(dia(l, true)))
).
prop_formula(
```

```
 implies(not(and(br2, not(bb))), not(dia(c, true)))
).

% instances of A5
prop_formula(
 implies(bb, dia(s, true))
).
prop_formula(
 implies(br1, dia(r, true))
).
prop_formula(
 implies(br2, dia(l, true))
).
prop_formula(
 implies(and(br2, not(bb)), dia(c, true))
).
end_of_list.

% The formula we want to prove below is
% bc1 v gc1 -> [vac] (bc1 v gc1)
% where vac is the vacuum cleaner's program:
% (gc1?; bb?; (br1?;s) U ((not br1)?;l;s)) U
% (gc2?; bb?; (br2?;s) U ((not br2)?;r;s)) U
% ( (not bb)?; (br2?;c) U ((not br2)?;r;c)))
list_of_special_formulae(conjectures, EML).
prop_formula(
 implies(
   or(bc1,gc1),
   box(
      or(
       % rule1
       comp(test(gc1),
            comp(test(bb),
                 or(comp(test(br1), s),
                    comp(test(not (br1)), comp(l,s))))),
        % rule2
        comp(test(gc2),
             comp(test(bb),
                  or(comp(test(br2), s),
                     comp(test(not (br2)), comp(r,s))))),
        % rule 3
        comp(test(not(bb)),
             or(comp(test(br2), c),
                comp(test(not(br2)), comp(r, c))))
      ), % end first vac or
          or(bc1,gc1) )
 ) % end implies
).
end_of_list.

end_problem.
```

9.3 PDL-TABLEAU *encoding of the blind committment property*

```
prove(
%axioms
[
 implies(and(gc1, br1, bb), box(s, and(bc1, br1, not(bb)))),
 implies(and(gc1, br2, bb), box(s, and(gc1, bc2, br2, not(bb)))),
 implies(and(bc1, br1, bb), box(s, and(bc1, br1, not(bb)))),
 implies(and(bc1, br2, bb), box(s, and(gc1, bc2, br2, not(bb)))),
 implies(and(bc1, br1), box(r, and(bc1, br2))),
 implies(and(gc1, br1), box(r, and(gc1, br2))),
 implies(and(bc1, br2,not(bb)), box(c, and(bc1, br2, bb))),
 implies(and(gc1, br2,not(bb)), box(c, and(gc1, br2, bb))),
 implies(and(gc1, br2), box(l, and(gc1, br1))),
 implies(and(bc1, br2), box(l, and(bc1, br1))),
 implies(not(bb), not(dia(s, true))),
 implies(not(br1), not(dia(r, true))),
 implies(not(br2), not(dia(l, true))),
 implies(not(and(br2, not(bb))), not(dia(c, true)))
],
% The formula we want to prove below is
% gc1 -> [vac*] (bc1 v gc1)
% where vac is the vacuum cleaner's program:
% (gc1?; bb?; (br1?;s) U ((not br1)?;l;s)) U
% (gc2?; bb?; (br2?;s) U ((not br2)?;r;s)) U
% ( (not bb)?; (br2?;c) U ((not br2)?;r;c)))
 implies(
   gc1,
   box(star(
      or(
       % rule1
       comp(test(gc1),
            comp(test(bb),
                 or(comp(test(br1), s),
                    comp(test(not (br1)), comp(l,s))))),
        % rule2
        comp(test(gc2),
             comp(test(bb),
                  or(comp(test(br2), s),
                     comp(test(not (br2)), comp(r,s))))),
        % rule 3
        comp(test(not(bb)),
             or(comp(test(br2), c),
                comp(test(not(br2)), comp(r, c))))
      )),
         or(bc1,gc1)))
). % end prove
```