A Logic of Agent Programs

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

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

Abstract

We present a sound and complete logic for reasoning about SimpleAPL programs. SimpleAPL is a fragment of the agent programming language 3APL 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.

Introduction

The verification of agent programs is a key problem in agent research and development. This focus stems both from the researcher's desire to check that agent architectures and programming languages conform to general principles of rational agency and the developer's need to check that a particular agent program will achieve the agent's goals. A logic used for reasoning about the execution of agent programs must be grounded in the computation of the agent (in the sense of (van der Hoek & Wooldridge 2003)). In addition, for the researcher, it is beneficial if the logic allows specification of key aspects of the agent's architecture, such as its execution cycle (e.g., to explore commitment under different program execution strategies). For both the developer and the researcher the logic should admit a fully automated verification procedure.

In this paper, we present a sound and complete logic for an APL-like (Dastani *et al.* 2004; Bordini *et al.* 2005) agent programming language which admits a fully automated verification procedure using theorem proving. Our logic is a variant of Propositional Dynamic Logic (PDL) (Fischer & Ladner 1979) and allows the specification of safety and liveness properties of agent programs. Moreover, our approach allows us to capture the agent's execution strategy in the logic.

The remainder of the paper is organised as follows. In the next section, we introduce SimpleAPL and give its operational semantics for two example program execution strategies. We then give the syntax and semantics of the logic and its sound and complete axiomatization and prove a correspondence between the operational semantics of SimpleAPL and the models of the logic for both program 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 a theorem prover.

An Agent Programming Language

In this section we present the syntax and semantics of SimpleAPL, a fragment of the logic based agent-oriented programming language 3APL (Dastani *et al.* 2004; Bordini *et al.* 2005). SimpleAPL contains the core features of 3APL and allows the implementation of agents with beliefs, goals, actions, plans, and planning rules. The main features of 3APL not present in SimpleAPL are a first order language for beliefs and goals, and rules for dropping goals and for revising plans. We have omitted these features in order to simplify the presentation; they do not present a significant technical challenge for our approach.

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). For simplicity, we only allow the agent's beliefs and goals to be 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, room 2 is not clean and its battery is charged:

```
Beliefs: room1, -clean2, 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.

Basic Actions Basic actions specify the capabilities an agent can use to achieve its goals. There are three types of basic actions: 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 derivable from

an 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 derivable from 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, and can be executed if its pre-condition is derivable from the agent's current beliefs. Executing the action adds its postcondition to the agent's beliefs. Belief update actions maintain consistency of the agent's beliefs, i.e., if p is in the belief set and $-p$ is added as a postcondition of an action, p is replaced by $-p$. For example, the following belief update specifications

```
BeliefUpdates:
  {room1}          moveR {-room1, room2}
  {room1, battery} suck  {clean1, -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". 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. 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 there, 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 the body of the rule. The goal query specifies what the plan is good for; 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 plan which forms the body of the rule. 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.

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 $\langle aliteral \rangle$ to denote the name of a belief update action and $\langle bliteral \rangle$ and $\langle gliteral \rangle$ to denote belief and goal literals.

$\langle APL\_Prog \rangle$ ::= `"BeliefUpdates:"` $\langle updatespecs \rangle$
| `"Beliefs:"` $\langle beliefs \rangle$
| `"Goals":` $\langle goals \rangle$
| `"PG rules:"` $\langle pgrules \rangle$

$\langle updatespecs \rangle$ ::= [$\langle updatespec \rangle$ `","` $\langle updatespec \rangle$)*]

$\langle updatespec \rangle$ ::= `"{"` $\langle beliefs \rangle$ `"}"` $\langle aliteral \rangle$ `"{"` $\langle beliefs \rangle$ `"}"`

$\langle beliefs \rangle$ ::= [$\langle bliteral \rangle$ `","` $\langle bliteral \rangle$)*]

$\langle goals \rangle$ ::= [$\langle gliteral \rangle$ `","` $\langle gliteral \rangle$)*]

$\langle plan \rangle$ ::= $\langle baction \rangle$ | $\langle sequenceplan \rangle$ | $\langle ifplan \rangle$ | $\langle whileplan \rangle$

$\langle baction \rangle$ ::= $\langle aliteral \rangle$ | $\langle testbelief \rangle$ | $\langle testgoal \rangle$

$\langle testbelief \rangle$ ::= $\langle bquery \rangle$ `"?"`

$\langle testgoal \rangle$ ::= $\langle gquery \rangle$ `"!"`

$\langle bquery \rangle$ ::= $\langle bliteral \rangle$ | $\langle bquery \rangle$ `"and"` $\langle bquery \rangle$ | $\langle bquery \rangle$ `"or"` $\langle bquery \rangle$

$\langle gquery \rangle$ ::= $\langle gliteral \rangle$ | $\langle gquery \rangle$ `"and"` $\langle gquery \rangle$ | $\langle gquery \rangle$ `"or"` $\langle gquery \rangle$

$\langle sequenceplan \rangle$ ::= $\langle plan \rangle$ `";"` $\langle plan \rangle$

$\langle ifplan \rangle$ ::= `"if"` $\langle bquery \rangle$ `"then {"` $\langle plan \rangle$ `"}"` [`"else {"` $\langle plan \rangle$ `"}"`]

$\langle whileplan \rangle$ ::= `"while"` $\langle bquery \rangle$ `"do {"` $\langle plan \rangle$ `"}"`

$\langle pgrules \rangle$ ::= [$\langle pgrule \rangle$ `","` $\langle pgrule \rangle$)*]

$\langle pgrule \rangle$ ::= [$\langle gquery \rangle$] `"<-"` $\langle bquery \rangle$ `"|"` $\langle plan \rangle$

Operational Semantics

We define the formal semantics of the agent programming language in terms of a transition system. Each transition corresponds to a single execution step and takes the system from one configuration to another. Configurations consist of the beliefs, goals, and plans of the agent. 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 operational semantics, one for an agent which executes a single plan to completion before choosing another plan, and another for an execution strategy which interleaves the execution of multiple plans with the adoption of new plans.

Definition 1 *The configuration of an individual agent is defined as* $\langle \sigma, \gamma, \Pi \rangle$ *where* σ *is a set of literals representing the agent's beliefs,* γ *is a set of literals representing the agent's goals, and* Π *is a set of plan entries representing the agent's current active plans.*

An agent's initial beliefs and goals are specified by its program, and Π is initially empty. Executing the agent's program modifies its initial configuration in accordance with the transition rules presented below. We first give the transitions for the non-interleaved execution strategy and then show how these are modified for interleaved execution.

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 its precondition is entailed by the agent's beliefs, i,e., $\sigma \models \phi$. Executing the action adds the literals in the postcondition to the agent's beliefs and removes any existing beliefs which are inconsistent with the postcondition.

$$(1)\quad \frac{T(\alpha,\sigma)=\sigma' \qquad \gamma'=\gamma\setminus\{\phi\in\gamma \mid \sigma'\models\phi\}}{\langle\sigma,\gamma,\{\alpha\}\rangle\longrightarrow\langle\sigma',\gamma',\{\}\rangle}$$

T is a partial function that takes a belief update action α and a belief set σ, and returns the modified belief set if the precondition of the action is entailed by σ. Note that executing a belief update action causes the agent to drop any goals it believes to be achieved as a result of the update.

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

$$(2)\quad \frac{\sigma\models\beta}{\langle\sigma,\gamma,\{\beta?\}\rangle\longrightarrow\langle\sigma,\gamma,\{\}\rangle}$$

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

$$(3)\quad \frac{\gamma\models\kappa}{\langle\sigma,\gamma,\{\kappa!\}\rangle\longrightarrow\langle\sigma,\gamma,\{\}\rangle}$$

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

$$(4)\quad \frac{\langle\sigma,\gamma,\{\pi_1\}\rangle\rightarrow\langle\sigma',\gamma',\{\}\rangle \;\; \langle\sigma',\gamma',\{\pi_2\}\rangle\rightarrow\langle\sigma'',\gamma'',\{\}\rangle}{\langle\sigma,\gamma,\{\pi_1;\pi_2\}\rangle\longrightarrow\langle\sigma'',\gamma'',\{\}\rangle}$$

$$(5)\quad \frac{\sigma\models\phi}{\langle\sigma,\gamma,\{\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\}\rangle\longrightarrow\langle\sigma,\gamma,\{\pi_1\}\rangle}$$

$$(6)\quad \frac{\sigma\not\models\phi}{\langle\sigma,\gamma,\{\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\}\rangle\longrightarrow\langle\sigma,\gamma,\{\pi_2\}\rangle}$$

$$(7)\quad \frac{\sigma\models\phi}{\langle\sigma,\gamma,\{\texttt{while}\ \phi\ \texttt{do}\ \pi\}\rangle\longrightarrow\langle\sigma,\gamma,\{\pi;\texttt{while}\ \phi\ \texttt{do}\ \pi\}\rangle}$$

$$(8)\quad \frac{\sigma\not\models\phi}{\langle\sigma,\gamma,\{\texttt{while}\ \phi\ \texttt{do}\ \pi\}\rangle\longrightarrow\langle\sigma,\gamma,\{\}\rangle}$$

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.

$$(9)\quad \frac{\gamma\models\kappa \quad \sigma\models\beta}{\langle\sigma,\gamma,\{\}\rangle\longrightarrow\langle\sigma,\gamma,\{\pi\}\rangle}$$

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. The transitions for an interleaved execution strategy are:

$$(1^i)\quad \frac{\alpha\in\Pi \quad T(\alpha,\sigma)=\sigma' \quad \gamma'=\gamma\setminus\{\phi\in\gamma \mid \sigma'\models\phi\}}{\langle\sigma,\gamma,\Pi\rangle\longrightarrow\langle\sigma',\gamma',\Pi\setminus\{\alpha\}\rangle}$$

$$(2^i)\quad \frac{\beta?\in\Pi \quad \sigma\models\beta}{\langle\sigma,\gamma,\Pi\rangle\longrightarrow\langle\sigma,\gamma,\Pi\setminus\{\beta?\}\rangle}$$

The interleaved version of (3) is analogous to (2^i), and the interleaved versions of (4) is as follows:

$$(4^i)\frac{\alpha\in\Pi \;\; \langle\sigma,\gamma,\Pi\rangle\rightarrow\langle\sigma',\gamma',\Pi\setminus\{\alpha\}\rangle \quad \alpha;\pi\in\Pi'}{\langle\sigma,\gamma,\Pi'\rangle\longrightarrow\langle\sigma',\gamma',(\Pi'\setminus\{\alpha;\pi\})\cup\{\pi\}\rangle}$$

The interleaved versions of (5) – (8) are obtained by adding a set of plans Π in all configurations, analogously to (1^i) – (4^i). In (9) the planning goal rule is applicable only if the agent's set of plan entries is empty. For the interleaved version, this requirement is dropped:

$$(9^i)\quad \frac{\gamma\models\kappa \quad \sigma\models\beta}{\langle\sigma,\gamma,\Pi\rangle\longrightarrow\langle\sigma,\gamma,\{\pi\}\cup\Pi\rangle}$$

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 basic 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 all the 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.

States and transitions Let P denote the set of propositional variables used to describe agent's beliefs and goals. A state s is a pair $\langle\sigma,\gamma\rangle$, where:

- σ is a set of beliefs $\{(-)p_1,\ldots,(-)p_n : p_i\in P\}$. We assume that belief states are consistent, i.e., for no $p\in P$ both p and $-p\in\sigma$.
- γ is a set of goals $\{(-)u_1,\ldots,(-)u_n : u_i\in P\}$. The set of goals does not have to be consistent, but it has to be disjoint from σ: no element of γ is in σ (i.e., is already believed).

Let the set of basic actions be $\texttt{Ac}=\{\alpha_1,\ldots,\alpha_m\}$. We associate with each $\alpha_i\in\texttt{Ac}$ a set of pre- and postconditions of the form $\{(-)p_1\in\sigma,\ldots,(-)p_n\in\sigma\}$, $\{(-)q_1\in\sigma',\ldots,(-)q_k\in\sigma'\}$ (where ps and qs are not necessarily disjoint) which mean: if α_i is executed in a state with belief set σ which satisfies the precondition then the resulting state s' has the belief set σ' which satisfies the postcondition (including replacing p with $-p$ if necessary to restore consistency), *the rest of σ' is the same as σ*, and the goal set $\gamma'=\gamma\setminus\{(-)p : (-)p\in\sigma'\}$.

Executing an action α_i in different configurations may give different results. For each α_i, we denote the set of pre- and postcondition pairs $\{(\texttt{prec}_1, \texttt{post}_1), \ldots, (\texttt{prec}_l, \texttt{post}_l)\}$ by $C(\alpha_i)$. We assume that $C(\alpha_i)$ is finite, that preconditions $\texttt{prec}_j, \texttt{prec}_k$ are mutually exclusive if $j \neq k$, and that each precondition has exactly one associated postcondition. We denote the set of all pre- and postconditions by $\mathbf{C}$.

Language Assume that we can make PDL program expressions ρ out of basic actions α_i 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. The language L for talking about the agent's beliefs, goals and plans is the language of PDL extended with belief operator B and goal operator G. A formula of L is defined as follows: if $p \in P$, then $B(\text{–})p$ and $G(\text{–})p$ are formulas; if 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.

Semantics A model for L is a structure $M = (S, \{R_{\alpha_i} : \alpha_i \in \texttt{Ac}\}, V)$, 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 such that for $s = \langle\sigma, \gamma\rangle$, $V_b(s) = \sigma$ and $V_g(s) = \gamma$.
- R_{α_i}, for each $\alpha_i \in \texttt{Ac}$, is a relation on S such that $(s, s') \in R_{\alpha_i}$ iff for some $(\texttt{prec}_j, \texttt{post}_j) \in C(\alpha_i)$, $\texttt{prec}_j(s)$ and $\texttt{post}_j(s')$, i.e., for some pair of pre- and postconditions of α_i, the precondition holds for s and the corresponding postcondition holds for s'. Note that this implies two things: first, an α_i transition can only originate in a state s which satisfies one of the preconditions for α_i; second, since pre-conditions are mutually exclusive, every such s satisfies exactly one pre-condition, and all α_i-successors of s satisfy the matching post-condition.

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

- $\tau(\alpha_i) = \{(s, s') : R_{\alpha_i}(s, s')\}$
- $\tau(\phi?) = \{(s, s) : M, s \models \phi\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1; \rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, where $\circ$ is concatenation of paths, such that $z_1 \circ z_2$ is only defined if z_1 ends in the state where z_2 starts
- $\tau(\rho^*)$ is the set of all paths consisting of zero or finitely many concatenations of paths in $\tau(\rho)$ (same condition on concatenation as above)

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

- $M, s \models B(\text{–})p$ iff $(\text{–})p \in V_b(s)$
- $M, s \models G(\text{–})p$ iff $(\text{–})p \in V_g(s)$
- $M, s \models \neg\phi$ 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 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 paths $\tau(\rho)$ starting in s, the end state s' of the path satisfies $M, s' \models \phi$.

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 basic actions).

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

To axiomatise the set of models defined above relative to $\mathbf{C}$ we need:

CL classical propositional logic

PDL axioms of PDL (see, e.g., (Harel, Kozen, & Tiuryn 2000))

A1 consistency of beliefs: $\neg(Bp \wedge B{-}p)$

A2 beliefs are not goals: $B(\text{–})p \rightarrow \neg G(\text{–})p$

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

$$f_b(\texttt{prec}_j) \wedge \Phi \rightarrow [\alpha_i](f_b(\texttt{post}_j) \wedge \Phi)$$

this is essentially a frame axiom for basic actions.

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

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

where $\top$ is a tautology.

A5 for every action α_i and every precondition $\texttt{prec}_j$ in $C(\alpha_i)$, $f_b(\texttt{prec}_j) \rightarrow \langle\alpha_i\rangle\top$

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 complete for the class of models* $\mathbf{M_C}$.

The proof is omitted due to lack of space.

Verification

In this section we show how to define exactly the set of paths in the transition system generated by the operational semantics by 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.

Agent Programs in PDL

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 $B(\text{–})p$ and $G(\text{–})q$ 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 Λ).

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

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: $\rho^+ = \rho; \rho^*$. According to this expression, each rule can be applied zero or more times (but at least one of the rules 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:

Safety properties:
$\phi \rightarrow [\, (\bigcup_{r_i \in \Lambda} (f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+ \,]\psi$

Liveness properties:
$\phi \rightarrow \langle\, (\bigcup_{r_i \in \Lambda} (f_g(\kappa_i) \wedge f_b(\beta_i))?; f_p(\pi_i))^+ \,\rangle \psi$

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.

Let $\mathbf{C}$ be a set of pre- and postconditions of belief updates, Λ an agent program, and TS the corresponding transition system defined by the operational semantics for an agent using a non-interleaved execution strategy (all possible configurations $\langle \sigma, \gamma, \{\Pi\} \rangle$ and transitions between them, given Λ and $\mathbf{C}$). Finally, let $M \in \mathbf{M_C}$ be a PDL model.

Theorem 2 *For every two agent configurations $\langle \sigma, \gamma, \{\} \rangle$ and $\langle \sigma', \gamma', \{\} \rangle$, there is a path between $\langle \sigma, \gamma, \{\} \rangle$ and $\langle \sigma', \gamma', \{\} \rangle$ in TS if, and only if, in M there is a corresponding path described by $\xi(\Lambda)$ between $\langle \sigma, \gamma \rangle$ and $\langle \sigma', \gamma' \rangle$.*

The proof is omitted due to lack of space.

Expressing the interleaved strategy For an agent with an interleaved execution strategy, we need a version of PDL with an additional interleaving operator, $\parallel$ (Abrahamson 1980).

Recall that each ρ is interpreted as a set of paths $\tau(\rho)$. Each path consists of zero or finitely many steps (s, s') which are in R_{α_i} for some atomic action α_i or in $\tau(\phi?)$ for some formula ϕ. The inductive clause for the interleaving operator $\rho_1 \parallel \rho_2$ is as follows:

$\tau(\rho_1 \parallel \rho_2)$ is the set of all paths obtained by interleaving atomic actions and tests from $\tau(\rho_1)$ and $\tau(\rho_2)$. Only legal computational sequences are considered here, namely whenever $(s_1, s_2)(s_3, s_4)$ is a subword of a sequence, then $s_2 = s_3$.

In this extended language, we can define paths in the execution of an agent with an interleaved execution strategy and planning goal rules $\Lambda = \{r_i | r_i = \kappa_i \leftarrow \beta_i | \pi_i\}$ by the following program expression:

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

Let $\mathbf{C}$, Λ and M be as in theorem 2, and TS be a transition system defined by the operational semantics for the interleaved execution strategy.

Theorem 3 *For every two agent configurations $\langle \sigma, \gamma, \{\} \rangle$ and $\langle \sigma', \gamma', \{\} \rangle$, there is a path between $\langle \sigma, \gamma, \{\} \rangle$ and $\langle \sigma', \gamma', \{\} \rangle$ in TS if, and only if, in M there is a corresponding path described by $\xi^i(\Lambda)$ between $\langle \sigma, \gamma \rangle$ and $\langle \sigma', \gamma \rangle$.*

The proof is omitted due to lack of space.

Example

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 $\texttt{clean}_\texttt{i}$, r_i for $\texttt{room}_\texttt{i}$, b for `battery`, s for `suck`, c for `charge`, r for `moveR`, l for `moveL`. The agent's program is:

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, this program corresponds to the following PDL program expression:

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

Given appropriate pre- and postconditions for the actions in the example program (such as the pre- and postconditions

of moveR and suck given earlier in the paper), some of the instances of A3–A5 are:

A3r $Br_1 \wedge Gc_2 \rightarrow [r](Br_2 \wedge Gc_2)$

A3s $Br_1 \wedge Bb \wedge Gc_2 \rightarrow [s](Bc_1 \wedge Br_1 \wedge Gc_2)$

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 (Hustadt & Schmidt 2000) (for properties without $*$) or PDL-TABLEAU (Schmidt 2003), and instances of axioms A1-A5 such as those above, we can prove a liveness property that if the agent has goals clean1, clean2 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.

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:

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

Discussion

In this paper, we proposed a sound and complete logic which allows the specification and verification of safety and liveness properties of SimpleAPL agents.

There has been a considerable amount of work on verifying properties of agents implemented in a variety of agent programming languages such as AgentSpeak, ConGolog and 3APL. Shapiro et al. (2002), 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 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 (Fisher 2006), 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 on the one hand and verification logics on the other, e.g., (Hindriks & Meyer 2007), have yet to result in an automated verification procedure.

Another important strand of work in automated verification of agents is based on model-checking, e.g., (Benerecetti, Giunchiglia, & Serafini 1998; Bordini *et al.* 2006; Lomuscio & Raimondi 2006). While our logic can be used to express properties of agents for model-checking, in this paper we chose to focus on the relationship between specific properties of programs and the axioms describing general properties of the system, as we believe this gives us a better insight into the reasons why a property holds or fails and which fundamental properties of the agent it depends on.

Acknowledgements We would like to thank to Renate Schmidt for help with MSPASS and PDL-TABLEAU. Natasha Alechina and Brian Logan were supported by EPSRC grant no. EP/E031226.

References

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

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

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

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

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

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.* 18(2):194–211.

Fisher, M. 2006. MetateM: The story so far. In *Proc. ProMAS 2005*, volume 3862 of *LNCS*, 3–22. Springer.

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

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

Hustadt, U., and Schmidt, R. A. 2000. MSPASS: Modal reasoning by translation and first-order resolution. In *Proc. TABLEAUX 2000*, volume 1847 of *LNCS*, 67–71. Springer.

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

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

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

van der Hoek, W., and Wooldridge, M. 2003. Towards a logic of rational agency. *Logic Journal of the IGPL* 11(2):133–157.