Reasoning about plan revision in BDI agent programs

Natasha Alechina[a], Mehdi Dastani[b], Brian Logan[a], John-Jules Ch. Meyer[b]

[a] *School of Computer Science*
University of Nottingham
Nottingham NG8 1BB, UK
[b] *Department of Information and Computing Sciences*
Universiteit Utrecht
3584CH Utrecht, The Netherlands

Abstract

Facilities for handling plan execution failures are essential for agents which must cope with the effects of nondeterministic actions, and some form of failure handling can be found in most mature agent programming languages and platforms. While such features simplify the development of more robust agents, they make it hard to reason about the execution of agent programs, e.g., to verify their correctness. In this paper, we present an approach to the verification of agent programs which admit exceptional executions. We consider executions of the BDI-based agent programming language 3APL in which plans containing non-executable actions can be revised using plan revision rules, and present a logic for reasoning about normal and exceptional executions of 3APL programs. We provide a complete axiomatisation for the logic and, using a simple example, show how to express properties of 3APL programs as formulas of the logic.

Keywords: Formalisms, logics

1. Introduction

3APL [1, 2] is a BDI-based agent-oriented programming language designed to facilitate the implementation of cognitive agents. In addition to programming constructs such as beliefs, goals, and plans common to many BDI-based agent programming languages, 3APL provides support for *plan revision rules* which allow the agent's plans to be revised at run time. Plan revision rules are intended to deal with an important problem in agent programming, namely the need to cope with the effects of nondeterministic action execution. For example, if an action which is intended to establish a precondition of a subsequent action in the same plan fails to do so, the agent needs to detect this and initiate some recovery behaviour.

Some form of 'failure handling' facility can be found in most mature BDI-based agent languages and platforms. For example, 2APL [3, 4] provides plan revision rules which can be applied to revise plans whose executions have failed, Jason [5, 6] provides "clean up" plans triggered by the abandonment of a goal, JACK [7] and SPARK

[8] provide failure methods and/or meta-procedures which are triggered when plan execution fails, and in [9, 10] features are proposed for aborting and suspending tasks in the context of the CAN abstract agent programming language. However, while such features simplify the development of more robust agents, they make it hard to reason about the execution of agent programs, e.g., to verify their correctness. As a result, formal accounts of program execution in the literature have typically been limited to the specification of the operational semantics of a particular failure handling construct as in [9] or to simplified versions of the language which do not admit exceptional executions. For example, [11] considers SimpleAPL, a version of 3APL without nondeterministic actions, plan revision rules, or any mechanism for dropping plans whose actions are unexecutable or when the goal which led to the adoption of the plan has been achieved, and [12] considers alternative execution strategies but again in the context of a simplified language and setting. Extending these accounts to model plan revision presents significant challenges. In addition to reasoning about the agent's beliefs, goals and plans, we need to model the current state of plan execution, and the evolution of the agent's program at run time in response to interactions between the actual (as opposed to intended) effects of the agent's actions in its environment and its plan revision rules.

In this paper we present an approach to reasoning about 3APL programs which include plan revision rules. We consider external actions with nondeterministic outcomes which may lead to unexpected achievement of a goal, or an inability to execute subsequent actions in the plan. We show how such exceptional executions can be formalised in a logic incorporating modalities to represent the agent's beliefs, goals and plans, and how this logic can be used to state and verify properties of the agent program. The main technical contribution of this work is in extending Propositional Dynamic Logic (PDL) [13] with explicit operators for 'having a plan' and axiomatising the interaction of belief, goal and plan modalities and standard PDL program modalities.

The remainder of the paper is organised as follows. In the next section we define the syntax of 3APL and give its operational semantics under both 'intended' and 'exceptional' executions. We then introduce the syntax and semantics of a logic to reason about safety and liveness properties of 3APL programs. We provide a sound and complete axiomatisation of the logic, and prove a correspondence between the operational semantics of 3APL and the models of the logic. Finally, we show how to translate agent programs written in 3APL into expressions of the logic, and, using a simple example, show how to verify a safety property for an agent program which allows plan revision.

2. 3APL

In this section we summarise the syntax and semantics of 3APL. 3APL is a BDI-based agent-oriented programming language which allows the implementation of agents with beliefs, (declarative) goals,[1] actions, plans, and rules for adopting and revising plans. We consider 3APL as defined in [2] with some minor modifications to simplify the presentation. The main change is that we assume a propositional language

[1] In the BDI literature, the terms 'goal' and 'desire' are often used interchangeably to denote the motivational attitude of an agent.

for beliefs and goals in this paper whereas [2] uses a first order language for beliefs and goals. However, 3APL assumes finite domains, and under this assumption all first order formulas can be encoded as propositional formulas. Other differences from [2] include restricting beliefs to atoms and goals to literals, and the omission of the *send* communication action. We also adopt a more precise formalisation of external actions which allows us to express when an external action can be executed and its possible effects. Finally, we follow the 2APL [4, 3] approach to plan revision rules in only applying plan revision rules to non-executable plans. These differences are discussed in more detail in the relevant sections below. We illustrate our presentation of 3APL with a simple running example loosely based on the example in [9]. In the example, an agent has a goal to attend a conference. To achieve this goal, it can adopt a plan of writing and submitting a paper (which may or may not be accepted). However before it can submit the paper it must obtain clearance from its organisation. If the clearance is not given, the agent can revise the paper and apply for clearance again. Writing a paper, applying for clearance and revising a paper take indeterminate amounts of time, with the result that they may not complete before the deadline for paper submission is past.

In 3APL, an agent's *state* is specified in terms of its beliefs, goals and plans, and its *program* by its initial state and its rules for adopting and revising plans. The agent's *beliefs* represent information about its environment while its *goals* represent situations the agent wants to bring about. Note that although the agent's beliefs are assumed to be consistent with each other, an agent's goals may conflict with each other as they do not all have to be achieved at the same time, i.e., goals that are inconsistent are assumed to be achieved in sequence. For example, an agent may want to attend a conference, and to write a proposal and not attend a conference. Although these two goals are conflicting in the sense that they cannot be achieved at the same time, the agent can achieve one after the other. For simplicity, we restrict the agent's beliefs to be a set of atoms[2] and assume the closed-world assumption, i.e., an agent believes $-p$ if p is not in its beliefs. The agent's goals are a set of (possibly inconsistent) literals. For example, an agent might have a goal to attend a conference and believe that fly is an option to travel:

```
BeliefBase: fly
GoalBase: attendConference
```

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

[2]In [2] an agent's beliefs can also include (definite) Horn clauses that can be used to infer new literals from existing atoms. This complicates the axiomatisation of belief update actions, since there may be several different ways of restoring consistency following a belief update. As belief update is not the main focus of this paper, we consider only belief atoms here.

```
writePaper; requestClearance; submitPaper;
  if accepted then travelToConference
```

causes the agent to write and submit a paper, and, if the paper is accepted, to travel to the conference.

Basic actions define the capabilities an agent can use to achieve its goals. There are two types of basic actions: *belief test actions* and *belief update actions*. We first explain the intuitive idea of basic actions before giving their formal definitions.

A belief test action, denoted by $\phi?$, tests whether the formula ϕ is derivable from an agent's beliefs. If ϕ is derivable from the agent's beliefs, execution of the plan continues. If ϕ is not derivable from the agent's beliefs, the belief test action is not executable and execution of the plan containing the belief test action blocks. Belief test actions play a role somewhat analogous to 'assert' statements in imperative programming languages. For example, they can be used to test if the execution of an external action had the effect intended by the programmer (see below) and if not, block further execution of the plan.

Belief update actions are the primitive operations an agent can use to change its environment. In contrast to the version of 3APL described in [2], we extend the notion of a belief update action to include both changing the agent's environment and updating the agent's beliefs to reflect the effects of the action. In [2] a distinction is made between (deterministic) mental actions which affect only the agent's beliefs and (nondeterministic) actions which affect the external environment and which are implemented as Java methods. For simplicity, we consider these two types of action together here, and use the term belief update action for both. We extend the notion of a 'capability', used in [2] to define mental actions, to allow the specification of nondeterministic external actions. The effects of a belief update action are defined in terms of pre- and postconditions. Each action has a set of preconditions specifying the situations in which the action is executable, and, for each precondition there is a set of postconditions, one for each possible outcome of executing the action in an environment in which the precondition holds (pre- and postconditions are sets of literals).[3] One or more postconditions correspond to the effect(s) of the action intended by the programmer, while the others correspond to possible but unintended outcomes (in the sense that they fail to achieve the agent's goal or make it impossible to execute subsequent actions in the plan). An action is executable if the belief literals in one of its preconditions are in the agent's current beliefs. When the action is executed, one of the postconditions associated with the precondition is chosen to modify the agent's beliefs. Execution of external actions is therefore nondeterministic, with the effects of the action being 'chosen' by the agent's environment. If the postcondition chosen by the environment corresponds to an unintended outcome, we say the action fails. Executing an action may take time, and agent execution resumes when the action terminates. For example, the belief update action

```
{paper, clearance, -deadlinePast}
```

[3]The deterministic mental actions of [2] therefore correspond to belief update actions with a single postcondition in our formulation.

```
  submitPaper
{{accepted}, {rejected}}
```

can be read as "if the agent has a paper and has obtained clearance, and the submission deadline has not passed, then the paper can be submitted, which results in one of two possible outcomes—in the first the paper is accepted (the intended outcome) and in the second it is rejected (an unintended outcome)". If none of the preconditions of an action is in the agent's current beliefs, the action is not executable and execution of the plan containing the action blocks. For example, if the agent does not have clearance, the `submitPaper` action is not executable. If execution of a plan blocks because the first step in the plan is a non-executable basic (belief test or belief update) action, we say plan execution fails (see [4, 3] for more detailed discussion of plan execution failure).

Our model of belief update actions effectively combines the execution of an external action with sensing to determine the effect of the action. For simplicity, we assume that belief update actions always terminate in one of the possible postconditions of the action, and that the agent's beliefs about the resulting state of environment are always correct. Execution of belief update actions maintains consistency of the agent's beliefs, i.e., if p is in an agent's belief base before executing an action α and the postcondition of α chosen by the environment contains $-p$, p is removed from the agent's belief base. Goals which are achieved by the postcondition of an action are dropped.[4] If a goal for which a plan was selected is achieved while the plan is still being executed, the plan is also dropped.

Unlike basic actions, *abstract plans* cannot be directly executed by the agent. Abstract plans provide an abstraction mechanism (similar to procedures in imperative programming) which are expanded into basic actions using plan revision rules (see below). If the first step of a plan π is an abstract plan $\bar{\alpha}$, execution of π blocks.

To adopt or revise plans, the agent uses rules. To adopt appropriate plans, the agent uses *planning goal rules*. A planning goal rule is of the form $\gamma \leftarrow \beta \mid \pi$ and consists of three parts: an (optional) goal query γ specifying the goal the plan is intended to achieve, a belief query β characterising situation(s) in which it could be a good idea to adopt the plan, and π the plan to be adopted. A planning goal rule is applicable if the goal and belief queries which form the head of the rule are derivable from the agent's goal and belief bases respectively, and the agent does not currently have a plan for this goal in its plan base. Applying a planning goal rule causes the agent to adopt the specified plan. The agent is committed to goals for which it has adopted a plan, and these goals correspond to what are termed 'intentions' in the BDI literature. For example, the planning goal rule:

```
attendConference <- -paper and -deadlinePast |
  writePaper; requestClearance; submitPaper;
  if accepted then travelToConference
```

states that "if the agent's goal is to attend the conference, and the agent believes it has no paper and the submission deadline is not past, then it may adopt the specified plan".

[4] In [2] dropping plans is part of the agent's deliberation strategy; for simplicity, we incorporate it into the basic language.

Note that the goal query part of a planning goal rule is optional so that an agent can generate a plan based only on its current beliefs. This allows the implementation of *reactive* agents (agents whose behaviour is partly or wholly triggered by their beliefs).

Other agent programming languages, such as AgentSpeak(L) [14] and Jason [5, 6], utilise a similar approach to adopting plans. However there are important differences between such languages and 3APL. A planning goal rule in 3APL is applicable if the goal query in the head of the rule is derivable from the agent's goal base (and the belief query evaluates to true). In contrast, in AgentSpeak(L) and Jason, a plan is applicable when a belief or goal change event matches the triggering event of the plan. This means that in 3APL a planning goal rule can be reapplied as long as the corresponding goal has not been achieved (i.e., as long as its corresponding goal is not derivable from the agent's belief base), while in Jason and AgentSpeak(L) a plan is adopted at most once in response to a triggering event.

Note that adopting a plan by applying a planning goal rule does not guarantee that the plan can be successfully executed. The belief query that characterises the situation(s) in which it could be a good idea to adopt the plan is only a heuristic, and in general cannot capture the preconditions of all the actions in the plan. In many cases, whether an action α in a plan can be successfully executed depends critically on one or more preceding actions in the plan having their intended effects in order to establish the appropriate precondition for α. However, as belief update actions are nondeterministic, executing an action may not have the intended effect of establishing a precondition of an action later in the plan. We say a plan is *not executable* if the first step in the plan is either a belief test action which evaluates to false or a belief update action which is not executable (i.e., none of its preconditions are derivable from the agent's current beliefs) or an abstract plan. (Non executable plans are called *blocked* in [15].)

Plan revision rules can be used by the agent to revise non-executable plans. A plan revision rule is of the form $\pi \leftarrow \beta \mid \pi'$ and consists of three parts: a plan to be revised π, a belief query β characterising the situation(s) in which it may be a good idea to adopt this revision, and a new plan π'. Plan revision rules are applicable when the plan to be revised is in the agent's plan base, the belief query is derivable from the agent's belief base and the next step in the plan is not executable. Applying a plan revision rule causes the agent to replace π in its plan base with π'. For example, assume that while the `requestClearance` action may result in the agent obtaining clearance, on some occasions it does not.

```
{paper, -clearance, -deadlinePast}
 requestClearance
{{clearance}, { }}
```

If the agent does not obtain clearance, the agent is unable to execute the next step in its plan (since obtaining clearance is a precondition of submitting a paper). To handle such situations, the agent can utilise a plan revision rule such as:

```
submitPaper; π <- paper and -clearance and -deadlinePast |
  revisePaper; requestClearance; submitPaper; π
```

where $\pi =$ `if accepted then travelToConference`.

The revised plan resulting from the application of a plan revision rule may simply replace the non executable action with a new action or sequence of actions followed by the rest of the original plan, or it may replace the original plan in its entirety. In particular, a plan may be replaced by the empty plan, allowing the agent to abandon a failed plan and use its planning goal rules to select a new plan to achieve the goal which is more appropriate to the current belief context. For example, if writing the paper takes until after the submission deadline is past, the agent can utilise a plan revision rule such as:

```
requestClearance; π <- deadlinePast | ϵ
```

where $\pi =$ `submitPaper; if accepted then travelToConference` and ϵ is the null plan, to drop the plan. Note that the agent still has the goal of attending the conference, and, if it has other applicable planning goal rules, it can adopt an alternative plan to attend the conference.

Plan revision rules give a 3APL developer considerable flexibility in determining how plan execution failures are handled. For example, if a particular action α is likely to fail but can be safely retried in the resulting environment, the developer can simply test for the intended postcondition ϕ and, if it doesn't hold, repeat the action: `while` $-\phi$ `do` α. However if recovering from the failure of α depends on which of the unintended outcomes of α actually results, we can include a belief test action after α which tests if ϕ has been achieved, e.g., $\alpha_1 \,;\, \phi? \,;\, \alpha_2$, and rely on the belief queries in the corresponding plan revision rules to select the appropriate recovery action(s). Alternatively, if an action α_1 is unlikely to fail and establishes the precondition of a subsequent action α_k, and recovering from the failure is easy even if the intervening actions $\alpha_2 \,;\, \ldots \,;\, \alpha_{k-1}$ have been executed, then the developer can simply rely on the non-executable action α_k triggering plan revision. Similar tradeoffs between how many and which kinds of failures to anticipate are also found in conditional planning and conventional imperative programming.

In addition to recovering from plan execution failures, in 3APL plan revision rules are also used to implement *abstract plans*. An abstract plan $\bar{\alpha}$ is expanded to a (more concrete) plan π' using a plan revision rule of the form $\bar{\alpha}; \pi \leftarrow \beta \mid \pi'; \pi$. The use of plan revision rules allows the selection of subplans to be context sensitive, i.e., conditional on the agent's beliefs. For example, if `travelToConference` is an abstract action, and the agent believes it should fly to the conference, it can use a plan revision rule such as:

```
travelToConference <- fly |
  buyPlaneTicket; flyToConference
```

to expand the abstract action.

We say the execution of a 3APL program is successful if it achieves the agent's goals. Conversely, we say that execution of a 3APL program fails if the agent is not able to achieve its goals, either because it has no planning goal rule to adopt an appropriate plan for a goal in the current belief context, or an adopted plan is not executable in the current belief context and the agent has no appropriate plan revision rule to repair it.

2.1. 3APL Syntax

The syntax of 3APL programs is given below in EBNF notation. It is defined relative to a set of propositions (atoms), belief update actions, and abstract plans. Following EBNF notation, we use `[]` brackets to indicate optional elements of the language.

Definition 1 (3APL Program)**.** *Let $\langle uaction \rangle$ denote the name of a belief update action, $\langle absplan \rangle$ denote the name of an abstract plan, $\langle literal \rangle$ denote a (belief or goal) literal, and $\langle atom \rangle$ denote a proposition. Then the syntax of 3APL is defined as follows:*

$\langle 3APL\_Prog \rangle$ ::= `"BeliefUpdates:"` $\langle updatespecs \rangle$
| `"BeliefBase:"` $\langle beliefs \rangle$
| `"GoalBase:"` $\langle goals \rangle$
| `"PG-rules:"` $\langle pgrules \rangle$
| `"PR-rules:"` $\langle prrules \rangle$

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

$\langle updatespec \rangle$ ::= $\langle preconditions \rangle$ $\langle uaction \rangle$ $\langle postconditions \rangle$

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

$\langle goals \rangle$ ::= [$\langle literals \rangle$]

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

$\langle baction \rangle$ ::= $\langle uaction \rangle$ | $\langle testbelief \rangle$

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

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

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

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

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

$\langle pgrule \rangle$ ::= [$\langle literal \rangle$] `"<-"` $\langle query \rangle$ `"|"` $\langle plan \rangle$

$\langle prrules \rangle$ ::= [$\langle prrule \rangle$ (`","` $\langle prrule \rangle$)*]

$\langle prrule \rangle$ ::= $\langle plan \rangle$ `"<-"` $\langle query \rangle$ `"|"` $\langle plan \rangle$

$\langle query \rangle$ ::= $\langle literal \rangle$ | $\langle query \rangle$ `"and"` $\langle query \rangle$ | $\langle query \rangle$ `"or"` $\langle query \rangle$

$\langle preconditions \rangle$ ::= $\langle condition \rangle$ [(`","` $\langle condition \rangle$)*]

$\langle postconditions \rangle$ ::= $\langle conditions \rangle$ [(`","` $\langle conditions \rangle$)*]

$\langle conditions \rangle$ ::= `"{"` $\langle condition \rangle$ [(`","` $\langle condition \rangle$)*] `"}"`

$\langle condition \rangle$ ::= `"{"` $\langle literals \rangle$ `"}"`

$\langle literals \rangle$ ::= [$\langle literal \rangle$ (`","` $\langle literal \rangle$)*]

2.2. Operational Semantics

In this section, we present a slightly modified version of the operational semantics for 3APL given in [2], focusing on individual agents. We define the formal semantics of 3APL in terms of a transition system [16]. Each transition corresponds to a single execution step and takes the system from one configuration to another.

Definition 2 (3APL Configuration)**.** *The configuration of an agent is defined as $\langle \sigma, \gamma, \Pi \rangle$ where σ is a set of atoms representing the agent's beliefs, γ is a set of literals representing the agent's goals, and Π is a set of plans.*

An agent's initial beliefs and goals are specified by its program and Π is initially empty. We assume that the agent adopts a *non-interleaved* execution strategy: "when in a configuration with no plan, choose a planning goal rule nondeterministically, apply it, execute the resulting plan to completion; repeat". If there are no applicable planning goal rules, the agent halts. Plan revision rules are only applied when the next step in the agent's plan cannot be executed: i.e., "when in a configuration with a non-executable plan, choose a plan revision rule nondeterministically, apply it, and execute the revised plan". If there are no applicable plan revision rules for a non-executable plan, the agent halts. With a non-interleaved execution strategy, at any point in the agent's execution, Π is either empty or contains a single plan. We annotate the plan base with the goal query, κ, of the planning goal rule that resulted in the adoption of the current plan π, denoted $\{\pi \triangleright \kappa\}$.

In formalizing the operational semantics, we use a notion of belief entailment based on the closed-world assumption and a notion of goal entailment based on set inclusion. The belief entailment relation is used to determine whether a formula is entailed by a belief base (which is a set of atoms) and the goal entailment relation is used to determined whether a formula is entailed by a goal base (which is a set of literals).

Definition 3 (Belief and Goal Entailment)**.** *Let σ be a belief base, γ be a goal base, p be an atom, l be a literal, and ϕ, ϕ_i, ψ be queries (see Definition 1). Then the belief and goal entailment relations, denoted respectively by $\sigma \models_{cwa} \phi$ and $\gamma \models_g l$, are defined as follows:*

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

Executing an agent's program modifies its initial configuration $\langle \sigma, \gamma, \Pi \rangle$ in accordance with the following transition rules.

Basic Actions 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 (both sets of propositional variables cannot be satisfied simultaneously). For each precondition $\texttt{prec}_i(\alpha)$ there is a set of matching postconditions $\texttt{post}_{i,1}(\alpha)$, ..., $\texttt{post}_{i,k}(\alpha)$, one for each possible outcome of an action. Each postcondition is also a finite set of literals. A belief update action α can be executed if $\sigma \models_{cwa} \texttt{prec}_i(\alpha)$. Executing α adds the positive literals in one of the postconditions $\texttt{post}_{i,1}(\alpha), \ldots, \texttt{post}_{i,k}(\alpha)$ corresponding to $\texttt{prec}_i(\alpha)$ to the agent's beliefs and removes any existing atoms from the agent's beliefs if their negations are in the postcondition.

Definition 4 (Belief Update)**.** *Let α be a belief update action, $\texttt{prec}_i$ be a precondition of α, $\texttt{post}_{i,j}$ be one of the corresponding postconditions of $\texttt{prec}_i$, and σ be a belief base such that $\sigma \models_{cwa} \texttt{prec}_i$. Then the outcome $T_{i,j}(\alpha, \sigma)$ of updating σ with α is*

defined as follows:

$$T_{i,j}(\alpha, \sigma) = (\sigma \cup \{p : p \in \texttt{post}_{i,j}(\alpha)\}) \setminus \{p : -p \in \texttt{post}_{i,j}(\alpha)\}$$

Observe that $T_{i,j}(\alpha, \sigma) \models_{cwa} \texttt{post}_{i,j}(\alpha)$ for all belief bases σ.

The successful execution of a belief update action α in a configuration where $\Pi = \{\alpha; \pi \triangleright \kappa\}$ (where π may be null) is then:

$$(1a) \quad \frac{\sigma \models_{cwa} \texttt{prec}_i(\alpha) \quad T_{i,j}(\alpha, \sigma) = \sigma' \quad \gamma' = \gamma \setminus \{\phi \in \gamma \mid \sigma' \models_{cwa} \phi\} \quad \sigma' \not\models_{cwa} \kappa}{\langle \sigma, \gamma, \{\alpha; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma', \gamma', \{\pi \triangleright \kappa\}\rangle}$$

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. If the goal condition for the current plan, κ, is dropped, the plan is also dropped:

$$(1b) \quad \frac{\sigma \models_{cwa} \texttt{prec}_i(\alpha) \quad T_{i,j}(\alpha, \sigma) = \sigma' \quad \gamma' = \gamma \setminus \{\phi \in \gamma \mid \sigma' \models_{cwa} \phi\} \quad \sigma' \models_{cwa} \kappa}{\langle \sigma, \gamma, \{\alpha; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma', \gamma', \{\,\}\rangle}$$

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

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

If a belief update action α is not executable or a belief test action β is not derivable from the agents beliefs, execution of the plan containing the belief update or test action blocks, and the plan must be revised by applying a plan revision rule as explained below.

Conditional Plans The following transition rules specify the effect of executing the conditional choice and conditional iteration operators, respectively. Note that the sequence operator, '`;`', is specified implicitly by the other rules which specify how to execute the first operation in the sequence.

$$(3a) \quad \frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1; \pi \triangleright \kappa\}\rangle}$$

$$(3b) \quad \frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_2; \pi \triangleright \kappa\}\rangle}$$

$$(4a) \quad \frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{\texttt{while } \phi \texttt{ do } \pi_1; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1; \texttt{while } \phi \texttt{ do } \pi_1; \pi \triangleright \kappa\}\rangle}$$

$$(4b) \quad \frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{\texttt{while } \phi \texttt{ do } \pi_1; \pi \triangleright \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi \triangleright \kappa\}\rangle}$$

Rules A planning goal rule $r_i = \kappa_i \leftarrow \beta_i \mid \pi_i$ can be applied if κ_i is entailed by the agent's goals, β_i is entailed by the agent's beliefs, and the agent's plan base is empty. Applying the rule r_i adds π_i to the agent's plans.

$$(5) \quad \frac{\gamma \models_g \kappa_i \quad \sigma \models_{cwa} \beta_i}{\langle \sigma, \gamma, \{\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_i \triangleright \kappa_i\}\rangle}$$

A plan revision rule $p_j = \pi_j \leftarrow \beta_j \mid \pi'_j$ can be applied if π_j is in the plan base, β_j is entailed by the agent's beliefs and π_j is not executable, i.e., the first action of π_j is either a belief update or belief test action which is not executable in the current belief state, or an abstract plan. (Note that this means that in order for a plan revision rule to be applicable, the plan base must be non-empty.)

$$(6a) \quad \frac{\forall i\ \sigma \not\models_{cwa} \mathtt{prec}_i(\alpha) \quad \sigma \models_{cwa} \beta_j}{\langle \sigma, \gamma, \{\pi_j = \alpha; \pi \rhd \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi'_j \rhd \kappa\}\rangle}$$

$$(6b) \quad \frac{\sigma \not\models_{cwa} \beta \qquad \sigma \models_{cwa} \beta_j}{\langle \sigma, \gamma, \{\pi_j = \beta?; \pi \rhd \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi'_j \rhd \kappa\}\rangle}$$

$$(6c) \quad \frac{\sigma \models_{cwa} \beta_j}{\langle \sigma, \gamma, \{\pi_j = \bar{\alpha}; \pi \rhd \kappa\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi'_j; \pi \rhd \kappa\}\rangle}$$

where $\bar{\alpha}$ is the name of an abstract plan.

Definition 5 (Transition Rules for the Operational Semantics of 3APL)**.** *The only transitions allowed by the operational semantics of 3APL are rules (1a)–(6c) listed above.*

3. Logic

In this section, we show how to describe state transition systems corresponding to the operational semantics of 3APL in an extension of Propositional Dynamic Logic (PDL) [13] with belief, goal and plan operators, which we call PDL-3APL.

The models of the logic look quite similar to the transition systems generated by the operational semantics of the agent. Each state has an assignment of beliefs, goals and of at most a single plan. Transitions between states correspond to executing a basic action, applying a planning goal rule, etc. For technical reasons, which will become clear after the precise statement of correspondence between the operational semantics and the PDL-3APL models, some states are marked by a special symbol x. Intuitively, those are the states where the goal corresponding to the agent's current plan has been achieved or the current plan is not executable.

The models of the logic are defined relative to an agent program. Therefore, we present the ingredients of an agent program which are used in defining the models before presenting the PDL-3APL language. Note that the definition below generalises the definition of a 3APL program (Definition 1). Namely, we abstract from the initial belief and goal bases. We also explicitly include the set of all possible plans which may occur in the execution of a program.

Definition 6 (Signature of an Agent Program)**.** *The signature of an agent program R is defined as*

$$R = \langle \mathcal{P}, PG, PR, Ac, \bar{A}c, Act, Plan \rangle$$

where:

- *$\mathcal{P}$ is a set of belief and goal atoms*

- *PG is a set of planning goal rules. We will denote elements of PG by r_i, where $r_i = \kappa_i \leftarrow \beta_i \mid \pi_i$*
- *PR is a set of plan revision rules. We will denote elements of PR by p_j, where $p_j = \pi_j \leftarrow \beta_j \mid \pi'_j$*
- *Ac is a set of belief update actions occurring in the plans of PG and PR rules*
- *$\bar{Ac}$ is a set of abstract plans occurring in the plans of PG and PR rules*
- *Act is the set of specifications for belief update actions Ac. For each element $\alpha \in Ac$, Act includes a set of mutually exclusive preconditions $\texttt{prec}_1(\alpha), \ldots, \texttt{prec}_m(\alpha)$ and for each precondition $\texttt{prec}_i(\alpha)$, a set of postconditions $\texttt{post}_{i1}(\alpha), \ldots, \texttt{post}_{ik}(\alpha)$*
- *$Plan$ is the set of all possible $\pi \triangleright \kappa$ pairs where κ is one of the agent's goals and π is a plan occurring in PG and PR rules or a suffix of such a plan (see Definition 7). ϵ denotes the null plan.*

The last clause in the above definition specifies the set of all plans that can occur during the execution of an agent program. Since plans are inserted into a configuration by PG and PR rules, and the execution of plan steps removes expressions from the beginning of the plans, the set of all possible plans that can occur during the agent's execution is the set of all suffixes of plans in the PG and PR rules. The following definition specifies a suffix of a plan.

Definition 7 (Plan Suffix)**.** *A suffix of a plan π, or otherwise a partial execution of π, is defined as follows:*

- *If $\pi = \pi_1; \pi_2$ then π_2 is a suffix of π;*
- *If $\pi = \texttt{if } \phi' \texttt{ then } \pi_1 \texttt{ else } \pi_2; \pi'$ then $\pi_1; \pi'$ and $\pi_2; \pi'$ are suffixes of π;*
- *If $\pi = (\texttt{while } \phi' \texttt{ do } \pi_1); \pi'$ then $\pi_1; (\texttt{while } \phi' \texttt{ do } \pi_1); \pi'$ is a suffix of π;*
- *a suffix of a suffix of π is a suffix of π.*

Note that if the sets of PG and PR rules are finite, then the set $Plan$ is finite too. This is due the fact that all plans in $Plan$ are either occur in a rule or are a suffix of such a plan and each plan has a finite number of suffixes.

3.1. Language

The language L of PDL-3APL is similar to that of PDL in that it contains modal operators corresponding to program expressions, or labels of transitions in the state transition system.

Definition 8 (PDL-3APL)**.** *Let $R = \langle \mathcal{P}$, PG, PR, Ac, $\bar{Ac}$, Act, $Plan\rangle$ be the signature of an agent program. Let ϕ be either a formula of type $\langle query\rangle$ (see Definition 1) or its negation, δ_{r_i} be the action of applying a planning goal rule r_i from PG, δ_{p_j} be the action of applying a plan revision rule p_j from PR, $p \in \mathcal{P}$, $\pi \in Plan$, κ be κ_i in*

$r_i \in PG$, *and* x *be a boolean flag. Then a program expression* ρ *of* L *is defined as follows:*

$$\rho ::= \alpha \in Ac \mid t(\phi) \mid \bar{a} \in \bar{Ac} \mid \delta_{ri} \mid \delta_{p_j} \mid \rho_1; \rho_2 \mid \rho_1 \cup \rho_2 \mid \rho^*$$

and a formula ψ *of PDL-3APL relative to* R *is defined as follows:*

$$\psi ::= Bp \mid Gp \mid G\text{–}p \mid \ x \mid P^{\kappa}\pi \mid P\epsilon \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \langle\rho\rangle\psi$$

In the definition of program expressions, $\rho_1; \rho_2$ is a sequential composition of ρ_1 and ρ_2, $\rho_1 \cup \rho_2$ corresponds to executing either ρ_1 or ρ_2, and ρ^* corresponds to executing ρ zero or finitely many times. The reason we use $t(\phi)$ rather than the PDL test operator $\phi?$ (for any PDL-3APL formula ϕ) is that unlike $\phi?$, the execution of $t(\phi)$ changes the state. Namely, the agent's plan is part of the state, and evaluating a test changes the agent's plan (for example, if in the current state the plan is `if` ϕ `then` π_1 `else` π_2 and the test formula ϕ evaluates to true, then as the result of the test transition, the plan in the next state is π_1). In the translation of conditional choice and conditional iteration as PDL program expressions we need to use a test action $t(\neg\phi)$, hence the extension of the set of test formulas to $\neg\langle query\rangle$.

In the definition of PDL-3APL, Bp stands for the agent believes that p, Gl for the agent has the goal that l, x is a boolean flag indicating a state where the current plan has to be dropped (the goal has been achieved) or revised (the plan is blocked), $P^{\kappa}\pi$ stands for the agent has a plan π for achieving goal κ, $P\epsilon$ stands for the plan base is empty, ρ is a program expression denoting a transition, and $\langle\rho\rangle\psi$ stands for 'there exists a state reachable by a transition denoted by ρ which satisfies ψ'. The dual modality is defined as $[\rho]\psi = \neg\langle\rho\rangle\neg\psi$ and means that all states reachable by a ρ transition satisfy ψ. We will sometimes use an abbreviation $\langle[\rho]\rangle\psi$ for $\langle\rho\rangle\psi \wedge [\rho]\psi$, and the usual $\vee, \rightarrow$ definitions.

3.2. Models

The models of PDL-3APL are labelled state transition systems. To relate these models to the operational semantics of 3APL, a number of conditions will be imposed on the states and transitions. To simplify the presentation, the models are defined with only references to the conditions. The conditions themselves will be specified directly after the definition of PDL-3APL models.

Definition 9 (PDL-3APL Models)**.** *Let* $R = \langle \mathcal{P}$*, PG, PR, Ac,* $\bar{Ac}$*, Act,* $Plan\rangle$ *be the signature of an agent program. A PDL-3APL model* M *relative to* R *is defined as*

$$M = (W, V, \mathcal{R}_{\alpha}, \mathcal{R}_{t(\phi)}, \mathcal{R}_{\bar{\alpha}}, \mathcal{R}_{\delta_{ri}}, \mathcal{R}_{\delta_{p_j}})$$

where

- W *is a non-empty set of states.*
- $V = (V_b, V_g, V_c, V_p)$ *is an evaluation function consisting of belief and goal valuation functions* V_b *and* V_g *satisfying condition* **C1** *(see below), control valuation function* V_c *and plan valuation function* V_p *such that for every* $s \in W$*:*

- $V_b(s) = \{p_1, \ldots, p_m : p_i \in \mathcal{P}\}$ *is the set of the agent's beliefs in* s*;*
- $V_g(s) = \{(-)u_1, \ldots, (-)u_n : u_i \in \mathcal{P}\}$ *is the set of the agent's goals in* s *(note that* V_g *assigns literals rather than propositional variables);*
- $V_c(s)$ *is either an empty set or* $\{x\}$*; if* x *is in* $V_c(s)$ *this means that either the goal corresponding to the agent's plan has been achieved or the agent's plan is not executable in* s*;*
- $V_p(s)$ *is either the empty set or a singleton set* $\{\pi \triangleright \kappa\}$*, where* π *is the agent's plan in* s *and* κ *is the goal(s) achieved by this plan.*

- $\mathcal{R}_\alpha$, $\mathcal{R}_{t(\phi)}$, $\mathcal{R}_{\bar{\alpha}}$, $\mathcal{R}_{\delta_{r_i}}$, $\mathcal{R}_{\delta_{p_i}}$ *are sets of binary relations on* W *defined as follows:*
  - $\mathcal{R}_\alpha = \{R_\alpha : \alpha \in Ac\}$*, where* R_α *is a set of pairs of states connected by a transition corresponding to a belief update action.* $\mathcal{R}_\alpha$ *is the largest relation satisfying conditions* **C2, C4***, and* **C8** *below.*
  - $\mathcal{R}_{t(\phi)} = \{R_{t(\phi)} : \phi \in \neg \langle query \rangle\}$*, where* $R_{t(\phi)}$ *is a set of pairs of states connected by a transition corresponding to testing whether* ϕ *is true.* $\mathcal{R}_{t(\phi)}$ *is the largest relation satisfying conditions* **C3, C5, C7***, and* **C8** *below.*
  - $\mathcal{R}_{\bar{\alpha}} = \{R_{\bar{\alpha}} : \alpha \in \bar{Ac}\}$*, where* $R_{\bar{\alpha}}$ *is a set of pairs of states connected by a transition corresponding to an abstract plan.* $\mathcal{R}_{\bar{\alpha}}$ *is the largest relation satisfying conditions* **C6** *and* **C8** *below.*
  - $\mathcal{R}_{\delta_{r_i}} = \{R_{\delta_{r_i}} : r_i \in PG\}$*, where* $R_{\delta_{r_i}}$ *is a set of pairs of states connected by a PG rule firing transition.* $\mathcal{R}_{\delta_{r_i}}$ *is the largest relation satisfying condition* **C9** *below.*
  - $\mathcal{R}_{\delta_{p_i}} = \{R_{\delta_{p_j}} : p_j \in PR\}$*, where* $R_{\delta_{p_j}}$ *is a set of pairs of states connected by a PR rule firing transition.* $\mathcal{R}_{\delta_{p_i}}$ *is the largest relation satisfying condition* **C10** *below.*

The conditions **C1–C10** enforce a correspondence between the operational semantics of 3APL and the models of PDL-3APL. The first condition **C1** restricts the states to those where beliefs and goals are disjoint while **C2–C10** restrict transitions between states. Only transitions satisfying these conditions exist in a model. We define each condition below. Note that, unless otherwise stated, $V_y(s') = V_y(s)$ where y is b, g, c or p, and the plan π in the conditions below may be null.

C1 (Beliefs and goals) Beliefs and goals are disjoint:

1. $V_g(s) \cap V_b(s) = \emptyset$ (p can never be both a belief and a goal)
2. $\{p : -p \in V_g(s)\} \subseteq V_b(s)$ (if $-p$ is a goal, then $-p$ does not follow by the closed world assumption from the agent's beliefs)

The intuition behind conditions **C2–C10** on relations R_α, $R_{t(\phi)}$, $R_{\bar{\alpha}}$, $R_{\delta_{r_i}}$ and $R_{\delta_{p_i}}$ is that in non-x states, transition relations in M correspond to the transitions possible in the operational semantics: for example, R_α is possible if α is the first step in the current plan and a precondition of α holds. When a goal is achieved, or when the next step in the plan is not executable, there is instead a transition to a x-state which in turn

may have a transition by a PG or PR rule to a non-x state again. So, while all transitions in non-x states correspond to the transitions in the operational semantics, only some of the transitions in x states have a corresponding transition in the operational semantics. We need x-states to 'consume' the rest of a PDL program expression when its corresponding plan in a state s (denoted by $V_p(s)$) is being dropped, in the sense which will become clear in the following section.

C2 (Execution of belief update actions)

If $V_p(s) = \{\alpha; \pi \triangleright \kappa\}$, $V_b(s) \models_{cwa} \texttt{prec}_i(\alpha)$ and $x \notin V_c(s)$ then there is an R_α transition to a state s' where

1. $V_b(s') = T_{i,j}(\alpha, V_b(s))$ ($V_b(s')$ is the result of updating $V_b(s)$ with some postcondition of α)
2. $V_g(s') = V_g(s) \setminus (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$ (goals which have been achieved by executing α are dropped)
3. if $V_b(s') \not\models_{cwa} \kappa$, $V_p(s') = \{\pi \triangleright \kappa\}$ (if the goal of the current plan has not been achieved, α is removed from the current plan). This corresponds to rule (1a) of the operational semantics.
4. if $V_b(s') \models_{cwa} \kappa$, $x \in V_c(s')$ and $V_p(s') = \{\}$ (if the goal of the current plan has been achieved, the plan is dropped and we transit to an x state). This corresponds to rule (1b) of the operational semantics.

C3 (Execution of tests)

If $V_p(s) = \{\phi?; \pi \triangleright \kappa\}$, $V_b(s) \models_{cwa} \phi$, and $x \notin V_c(s)$, then there is a $R_{t(\phi)}$ transition to a state s' where $V_p(s') = \{\pi \triangleright \kappa\}$. This corresponds to rule (2) of the operational semantics.

C4 (Non-executable belief update actions)

If $V_p(s) = \{\alpha; \pi \triangleright \kappa\}$, $V_b(s) \not\models_{cwa} \texttt{prec}_i(\alpha)$, and $x \notin V_c(s)$, then there is an R_α transition to a state s' where $x \in V_c(s')$. This corresponds to the 'first half' of the operational semantics rule (6a); the 'second half', which transits back to a non-x state, corresponds to condition **C10** below.

C5 (Non-executable tests)

If $V_p(s) = \{\phi?; \pi \triangleright \kappa\}$, $V_b(s) \not\models_{cwa} \phi$, and $x \notin V_c(s)$, then there is a $R_{t(\phi)}$ transition to a state s' where $x \in V_c(s')$. This corresponds to the 'first half' of the operational semantics rule (6b); the 'second half', which transits back to a non-x state, corresponds to condition **C10** below.

C6 (Expansion of abstract plans) If $V_p(s) = \{\bar{\alpha}; \pi \triangleright \kappa\}$ where $\bar{\alpha}$ is an abstract plan and $x \notin V_c(s)$, then there is an $R_{\bar{\alpha}}$ transition to a state s' where $x \in V_c(s')$. This corresponds to the 'first half' of the operational semantics rule (6c); the 'second half', which transits back to a non-x state, corresponds to condition **C10** below.

C7 (Execution of conditional plans)

1. If $V_p(s) = \{\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2; \pi \triangleright \kappa\}$, $V_b(s) \models_{cwa} \phi$, and $x \notin V_c(s)$, then there is an $R_{t(\phi)}$ transition to a state s' where $V_p(s') = \{\pi_1; \pi \triangleright \kappa\}$. This corresponds to rule (3a) of the operational semantics.

2. If $V_p(s) = \{\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2;\ \pi \triangleright \kappa\}$, $V_b(s) \not\models_{cwa} \phi$, and $x \notin V_c(s)$, then there is a $R_{t(\neg\phi)}$: transition to a state s' where $V_p(s') = \{\pi_2; \pi \triangleright \kappa\}$. This corresponds to rule (3b) of the operational semantics.
3. If $V_p(s) = \{\texttt{while}\ \phi\ \texttt{do}\ \pi_1;\ \pi \triangleright \kappa\}$, $V_b(s) \models_{cwa} \phi$, and $x \notin V_c(s)$, then there is an $R_{t(\phi)}$ transition to a state s' where $V_p(s') = \{\pi_1;\ \texttt{while}\ \phi\ \texttt{do}\ \pi_1;\ \pi \triangleright \kappa\}$. This corresponds to rule (4a) of the operational semantics.
4. If $V_p(s) = \{\texttt{while}\ \phi\ \texttt{do}\ \pi_1;\ \pi \triangleright \kappa\}$, $V_b(s) \not\models_{cwa} \phi$, and $x \notin V_c(s)$, then there is an $R_{t(\neg\phi)}$ transition to a state s' where $V_p(s') = \{\pi \triangleright \kappa\}$. This corresponds to rule (4b) of the operational semantics.

C8 **(Execution in x-states)** If $x \in V_c(s)$ then there are R_α, $R_{\bar{\alpha}}$ and $R_{t(\phi)}$ transitions from state s to itself. This condition ensures that there is a path from any x-state to itself, labelled by the remaining actions of the PDL program expression, so that the remainder of the PDL program expression is 'consumed' without changing the state.

C9 **(PG rules)** If $V_p(s) = \{\}$, $V_g(s) \models_g \kappa_i$, $V_b(s) \models_{cwa} \beta_i$, then there is a $R_{\delta_{r_i}}$ transition to a state s' where $V_p(s') = \{\pi_i \triangleright \kappa_i\}$ and $x \notin V_c(s')$ (where $r_i = \kappa_i \leftarrow \beta_i \,|\, \pi_i$).

Note that PG rules are applicable only if the plan base is empty, and the states in which they are applicable may be x-states. If a plan achieves its goal, even by the last action of the plan, this results in transiting to an x-state with an empty plan base. Firing a PG rule results in transition to a non-x state, with the same beliefs and goals, and a new plan in the plan base. This corresponds to the rule (5) of the operational semantics. As in the operational semantics, if no PG rule is applicable, the program halts.

C10 **(PR rules)** If $x \in V_c(s)$, $V_p(s) = \{\pi_j \triangleright \kappa\}$, $V_b(s) \models_{cwa} \beta_j$, then there is a $R_{\delta_{p_j}}$ transition to a state s' where $V_p(s') = \{\pi'_j \triangleright \kappa\}$ and $x \notin V_c(s')$ (where $p_j = \pi_j \leftarrow \beta_j \,|\, \pi'_j$).

This means that PR rules are only applicable in x-states with a non-empty plan base which matches the condition of the PR rule. Firing a PR rule results in transiting to a non-x state with a modified plan base. This corresponds to the 'second half' of rules (6a)–(6c) of the operational semantics. As in the operational semantics, if current plan blocks and no PR rule is applicable, the program halts.

The primitive transitions for belief update action, test action, abstract plan, and rule applications can be composed by standard PDL operators, i.e., choice, sequence, and iteration operators.

Definition 10 (Transition Relation Composition)**.** *Given the basic relations R_u (where u is α, $t(\phi)$, $\bar{\alpha}$, δ_{r_i} or δ_{p_j}) defined in the model, we can define relations R_ρ corresponding to complex program expressions ρ inductively as follows:*

- $R_{\rho_1 \cup \rho_2} = R_{\rho_1} \cup R_{\rho_2}$
- $R_{\rho_1;\rho_2} = R_{\rho_1} \circ R_{\rho_2}$ *where $\circ$ is composition of relations*

- $R_{\rho^*} = (R_\rho)^*$ *(reflexive transitive closure of* R_ρ*)*

The satisfaction relation for PDL-3APL is a slight modification of the standard definition of the satisfaction relation for PDL.

Definition 11 (Satisfaction Relation $\models$)**.** *The relation* $\models$ *of a PDL-3APL formula* ϕ *being true in a state* s *of a model* M*, denoted as* $M, s \models \phi$*, is defined inductively as follows:*

- $M, s \models Bp$ *iff* $p \in V_b(s)$
- $M, s \models Gp$ *iff* $p \in V_g(s)$
- $M, s \models G{-}p$ *iff* $-p \in V_g(s)$
- $M, s \models x$ *iff* $x \in V_c(s)$
- $M, s \models P^\kappa \pi$ *iff* $V_p(s) = \{\pi \triangleright \kappa\}$
- $M, s \models P\epsilon$ *iff* $V_p(s) = \{\}$
- $M, s \models \neg\psi$ *iff* $M, s \not\models \psi$
- $M, s \models \psi_1 \wedge \psi_2$ *iff* $M, s \models \psi_1$ *and* $M, s \models \psi_2$
- $M, s \models \langle\rho\rangle\psi$ *iff there exists* s' *such that* $R_\rho(s, s')$ *and* $M, s' \models \psi$*.*

We denote the class of models satisfying the conditions **C1–C10** for an agent program with signature R as $\mathbf{M}_{3APL}(R)$.

3.3. Axiomatisation

To axiomatise this class of models, we first need to explain how pre- and postconditions of transitions can be expressed in the language of PDL-3APL.

Definition 12 (Translation Functions f_b, f_g, f_p)**.** *The beliefs, goals and plans of agent programs can be translated into PDL-3APL expressions using translation functions* f_b*,* f_g *and* f_p *defined as follows:*

- f_b*: Let* $p \in \mathcal{P}$ *and* ϕ, ψ *be belief query expressions (i.e.,* $\langle query\rangle$*) of 3APL:* $f_b(p) = Bp$*;* $f_b(\phi \texttt{ and } \psi) = f_b(\phi) \wedge f_b(\psi)$*;* $f_b(\phi \texttt{ or } \psi) = f_b(\phi) \vee f_b(\psi)$*. For negations of formulas in* $\langle query\rangle$*, we also need the case* $f_b(\neg\phi) = \neg f_b(\phi)$*. For translating pre- and postconditions of actions, we need to extend* f_b *to sets of literals; if* X *is a set of literals,* $f_b(X) = \bigwedge_{p\in X} Bp \wedge \bigwedge_{-p\in X} \neg Bp$*.*
- $f_g(p) = Gp$*;* $f_g(-p) = G{-}p$*.*
- f_p*: Let* α *be a belief update action,* ϕ *and* ψ *be belief query expressions,* $\bar{\alpha}$ *an abstract plan, and* π, π_1, π_2 *be plan expressions (i.e.,* $\langle plan\rangle$*) of 3APL:*
  - $f_p(\alpha) = \alpha$
  - $f_p(\phi?) = t(\phi)$

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

We can now axiomatise the class of models $\mathbf{M}_{3APL}(R)$ relative to an agent program with signature R. Note that PDL-3APL contains belief, goal and plan modalities and several kinds of 'atomic programs' (belief update actions, belief test actions, abstract plans, planning goal rules and plan revision rules). Each of those ingredients requires one or two axioms, hence the length of the axiomatisation. In the following definition, we use labels **PDLi** for standard PDL axioms, **Ai** for state axioms, **BAi** for basic action axioms, **CPi** for composite plan axioms, **PGi** for planning goal rule axioms, and **PRi** for plan revision rule axioms.

Definition 13 (PDL-3APL Axiomatic System)**.** *Let ψ_{np} be any formula not containing plan expressions, ψ_{npx} be any formula not containing plan expressions or x, and ψ_{nx} be any formula not containing x. Let π be a plan expression (if π is null, $P^\kappa\pi$ is $P\epsilon$). Then the following axioms and inference rules constitute the axiomatic system of PDL-3APL.*

CL *all tautologies of classical propositional logic*

PDL1 $[\rho](\phi \rightarrow \psi) \rightarrow ([\rho]\phi \rightarrow [\rho]\psi)$

PDL2 $\langle\rho_1; \rho_2\rangle\phi \leftrightarrow \langle\rho_1\rangle\langle\rho_2\rangle\phi$

PDL3 $\langle\rho_1 \cup \rho_2\rangle\phi \leftrightarrow \langle\rho_1\rangle\phi \vee \langle\rho_2\rangle\phi$

PDL4 $\langle\rho^*\rangle\phi \leftrightarrow \phi \vee \langle\rho\rangle\langle\rho^*\rangle\phi$

PDL5 $[\rho^*](\phi \rightarrow [\rho]\phi) \rightarrow (\phi \rightarrow [\rho^*]\phi)$

MP $\dfrac{\phi,\ \phi{\rightarrow}\psi}{\psi}$

N $\dfrac{\phi}{[\rho]\phi}$

A1 $Bp \rightarrow \neg Gp$

A2 $G{-}p \rightarrow Bp$

A3a $P^\kappa\pi \rightarrow \neg P^{\kappa'}\pi'$ *where* $\pi' \neq \pi$ *or* $\kappa' \neq \kappa$

A3b $P\epsilon \vee \bigvee_{\pi \triangleright \kappa \in Plan} P^\kappa\pi$

BA1 $\neg x \wedge P^\kappa(\alpha;\pi) \wedge f_b(\texttt{prec}_i(\alpha)) \wedge \psi \wedge \psi' \rightarrow \langle\alpha\rangle($
$(f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^\kappa\pi \wedge \psi) \vee (f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi'))$

where ψ, ψ' are any formulas not containing plan expressions or literals in $f_b(\texttt{post}_{ij}(\alpha))$, and in addition ψ' does not contain x

BA2a $\neg x \wedge P^{\kappa}\pi \rightarrow [u]\bot$ *where* $\pi \neq u;\pi'$ *and* $u \in Ac \cup \bar{A}c$

BA2b $\neg x \wedge P^{\kappa}\pi \rightarrow [t(\phi)]\bot$ *if* π *does not start with a belief test action* $\phi?$ *or a conditional plan test on* ψ *where* $\phi = \psi$ *or* $\phi = \neg\psi$

BA3 $\neg x \wedge P^{\kappa}(\alpha;\pi) \wedge f_b(\texttt{prec}_i(\alpha)) \wedge \bigwedge_j \psi_j \wedge \bigwedge_j \psi'_j \rightarrow [\alpha]($
$\bigvee_j (\, f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi_j) \vee$
$\bigvee_j (\, f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi'_j))$
where ψ_j *and* ψ'_j *are any formulas not containing plan expressions or literals in* $f_b(\texttt{post}_{ij}(\alpha))$*, and in addition* ψ'_j *does not contain* x

BA4 $\neg x \wedge P^{\kappa}(\phi?;\pi) \wedge f_b(\phi) \wedge \psi_{np} \rightarrow \langle [t(\phi)] \rangle (P^{\kappa}\pi \wedge \psi_{np})$

BA5 $\neg x \wedge P^{\kappa}(\alpha;\pi) \wedge \bigwedge_i \neg f_b(\texttt{prec}_i(\alpha)) \wedge \psi_{nx} \rightarrow \langle [\alpha] \rangle (x \wedge \psi_{nx})$

BA6 $\neg x \wedge P^{\kappa}(\phi?;\pi) \wedge \neg f_b(\phi) \wedge \psi_{nx} \rightarrow \langle [t(\phi)] \rangle (x \wedge \psi_{nx})$

BA7 $\neg x \wedge P^{\kappa}(\bar{\alpha};\pi) \wedge \psi_{nx} \rightarrow \langle [\bar{\alpha}] \rangle (x \wedge \psi_{nx})$

BA8 $x \wedge \psi \rightarrow \langle [u] \rangle \psi$ *where* u *is* α*,* $t(\phi)$ *or* $\bar{\alpha}$

CP1 $\neg x \wedge P^{\kappa}(\pi_{if};\pi) \wedge f_b(\phi) \wedge \psi_{np} \rightarrow \langle [t(\phi)] \rangle (P^{\kappa}\pi_1;\pi \wedge \psi_{np})$*, where* π_{if} *is of the form* $\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2$

CP2 $\neg x \wedge P^{\kappa}(\pi_{if};\pi) \wedge \neg f_b(\phi) \wedge \psi_{np} \rightarrow \langle [t(\neg\phi)] \rangle (P^{\kappa}\pi_2;\pi \wedge \psi_{np})$*, where* π_{if} *is as in* **CP1**

CP3 $\neg x \wedge P^{\kappa}(\pi_{wh};\pi) \wedge f_b(\phi) \wedge \psi_{np} \rightarrow \langle [t(\phi)] \rangle (P^{\kappa}\pi_1;\pi_{wh};\pi \wedge \psi_{np})$*, where* π_{wh} *is of the form* $\texttt{while}\ \phi\ \texttt{do}\ \pi_1$

CP4 $\neg x \wedge P^{\kappa}(\pi_{wh};\pi) \wedge \neg f_b(\phi) \wedge \psi_{np} \rightarrow \langle [t(\neg\phi)] \rangle (P^{\kappa}\pi \wedge \psi_{np})$*, where* π_{wh} *is as in* **CP3**

CP5 $\neg x \wedge (P^{\kappa}\pi_{if} \vee P^{\kappa}\pi_{wh}) \wedge \neg f_b(\phi) \rightarrow [t(\phi)]\bot$ *where* π_{if} *and* π_{wh} *are as above*

PG1 $P\epsilon \wedge f_g(\kappa_i) \wedge f_b(\beta_i) \wedge \psi_{npx} \rightarrow \langle [\delta_{r\,i}] \rangle (\neg x \wedge P^{\kappa_i}\pi_i \wedge \psi_{npx})$

PG2 $\neg P\epsilon \vee \neg f_g(\kappa_i) \vee \neg f_b(\beta_i) \rightarrow [\delta_{r\,i}]\bot$

PR1 $x \wedge P^{\kappa}\pi_j \wedge f_b(\beta_j) \wedge \psi_{npx} \rightarrow \langle [\delta_{p\,j}] \rangle (\neg x \wedge P^{\kappa}\pi'_j \wedge \psi_{npx})$

PR2 $\neg x \vee \neg P^{\kappa}\pi_j \vee \neg f_b(\beta_j) \rightarrow [\delta_{p\,j}]\bot$

Theorem 1. *PDL-3APL is sound with respect to* $\mathbf{M}_{3APL}(R)$.

The proof of soundness is by straightforward induction on the length of a derivation. It is obvious that the inference rules derive valid conclusions from valid premises. It remains to show that the axioms are valid. In the following, we only show validity of some of the axioms that are characteristic for our framework. The validity proofs of other axioms are either the same as the corresponding axioms of PDL or they are similar to those we prove here. We first formulate the following proposition which will be used in the validity proofs of several axioms.

Proposition 1. *Let M be a model, s be a state in M, V_b be the belief valuation function of M, and X be a set of literals. Then,*

$$M, s \models f_b(X) \text{ iff } V_b(s) \models_{cwa} X$$

Proof. $M, s \models f_b(X) \overset{def.}{\Leftrightarrow}{}^{12} M, s \models \bigwedge_{p\in X} Bp \wedge \bigwedge_{-p\in X} \neg Bp \overset{def.}{\Leftrightarrow}{}^{11} M, s \models \bigwedge_{p\in X} Bp$ and $M, s \models \bigwedge_{-p\in X} \neg Bp \overset{def.}{\Leftrightarrow}{}^{11} \bigwedge_{p\in X} p \in V_b(s)$ and $\bigwedge_{-p\in X} p \notin V_b(s) \overset{def.}{\Leftrightarrow}{}^{3}$ $\forall p \in X : V_b(s) \models_{cwa} p$ and $\forall -p \in X : V_b(s) \models_{cwa} -p \overset{def.}{\Leftrightarrow}{}^{3} V_b(s) \models_{cwa} X$. □

The following propositions establish the validity of axioms **BA1**, **BA2a**, **BA3**, **CP1**, **CP4**, **PG1**, **PG2**, **PR1,** and **PR2**.

Proposition 2. *Axiom **BA1** is valid.*

Proof. $M, s \models \neg x \wedge P^{\kappa}(\alpha;\pi) \wedge f_b(\texttt{prec}_i(\alpha)) \wedge \psi \wedge \psi' \overset{def.}{\Leftrightarrow}{}^{11} M, s \models \neg x$ and $M, s \models P^{\kappa}(\alpha;\pi)$ and $M, s \models f_b(\texttt{prec}_i(\alpha))$ and $M, s \models \psi$ and $M, s \models \psi'$. From Definition 11 and Proposition 1 we can conclude $x \notin V_c(s)$ and $V_p(s) = \{\alpha; \pi \triangleright \kappa\}$ and $V_b(s) \models_{cwa} \texttt{prec}_i(\alpha)$. Then, condition **C2** ensures that there exists a state s' such that $R_\alpha(s, s')$, where either clauses 1, 2 and 3, or clauses 1, 2 and 4 below hold.

1 $V_b(s') = T_{i,j}(\alpha, V_b(s))$ for some postcondition j of α
2 $V_g(s') = V_g(s) \setminus (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$
3 $V_b(s') \not\models_{cwa} \kappa$ and $V_p(s') = \{\pi \triangleright \kappa\}$
4 $V_b(s') \models_{cwa} \kappa$, $x \in V_c(s')$ and $V_p(s') = \{\}$

From Proposition 1, Definitions 11 and 12, and that $T_{i,j}$ ensures $V_b(s') \models_{cwa} \texttt{post}_{ij}(\alpha)$ (see Definition 4), and the clauses above, we have that:

1 $M, s' \models f_b(\texttt{post}_{ij}(\alpha))$ for some postcondition j of α
2 $V_g(s') = V_g(s) \setminus (\{p : p \in V_b(s')\} \cup \{-p : p \notin V_b(s')\})$
3 $M, s' \models \neg f_b(\kappa)$ and $M, s' \models P^{\kappa}\pi$
4 $M, s' \models f_b(\kappa)$, and $M, s' \models x$, and $M, s' \models P\epsilon$

Moreover, from clauses 1 and 2 we can conclude that for all ψ not in the postcondition of action α and not a plan formula $M, s \models \psi$ iff $M, s' \models \psi$. Similarly for all formulas ψ' which do not contain the boolean flag x. Thus there exists a state s' such that $R_\alpha(s, s')$ and either $M, s' \models f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi$ or $M, s' \models f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi'$. In other words, $M, s \models \langle\alpha\rangle(f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi) \vee (f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi')$. □

Proposition 3. *Axiom **BA2a** is valid.*

Proof. Suppose $M, s \models \neg x \wedge P^{\kappa}\pi$ where $\pi \neq u;\pi'$, but $M, s \models \neg[u]\bot$ which is equivalent to $M, s \models \langle u\rangle\top$. Following Definition 11 there exists a state s' such that $R_u(s, s')$. We consider the two possible cases where $u \in Ac$ and $u \in \bar{Ac}$. Case 1 ($u \in Ac$): from Definition 9 only those pairs of states that satisfy conditions **C2, C4,** and **C8** are in R_u. However, the existence of an R_u edge is not implied by **C2** and

C4 because these require $M, s \models P^{\kappa}u; \pi'$ which is contrary to our assumption[5] that $M, s \models P^{\kappa}\pi$ where $\pi \neq u; \pi'$. An R_u edge is also not implied by **C8** because this requires $M, s \models x$ which is again contrary to our assumption that $M, s \models \neg x$. Case 2 follows a similar line of reasoning with respect to conditions **C6** and **C8**. □

Proposition 4. *Axiom **BA3** is valid.*

Proof. Assume the antecedent holds in M, s, but not the consequent. From Definition 11 the negation of the consequent is $M, s \models \langle\alpha\rangle \bigwedge_j \neg(\, f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi_j) \wedge \bigwedge_j \neg(\, f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi'_j)) \overset{def.11}{\Leftrightarrow} \exists s' : R_{\alpha}(s, s')$ and $M, s \models \bigwedge_j \neg(\, f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi_j) \wedge \bigwedge_j \neg(\, f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P\epsilon \wedge \psi'_j))$. Definition 9 states that only those R_{α} transitions exist that satisfy conditions **C2, C4** and **C8**. Only **C2** can be satisfied since **C2** requires $V_b(s) \models_{cwa} \texttt{prec}_i(\alpha)$ which is satisfied by the assumption $M, s \models f_b(\texttt{prec}_i(\alpha))$ using Proposition 1; other conditions are clearly not satisfied. However, **C2** requires $V_b(s') = T_{i,j}(\alpha, V_b(s))$ for some postcondition j of α. Since $T_{i,j}(\alpha, V_b(s)) \models \texttt{post}_{ij}(\alpha)$ (see Definition 4), we have $M, s' \models f_b(\texttt{post}_{ij}(\alpha))$ for some postcondition j of α. Moreover, **C2** requires either ($V_b(s') \not\models_{cwa} \kappa$ and $V_p(s') = \{\pi \triangleright \kappa\}$) or ($V_b(s') \models_{cwa} \kappa,\ x \in V_c(s')$ and $V_p(s') = \{\}$). Finally, clauses 1,2, and 3 of **C2** require that only the valuation of formulas containing plan expressions or those involved in $\texttt{post}_{ij}(\alpha)$ can be changed, and clauses 1,2, and 4 require that the valuation of only formulas containing plan expressions, x, and those involved in $\texttt{post}_{ij}(\alpha)$ can be changed; the valuation of all other formulas remains unchanged. Altogether, **C2** requires either $M, s' \models (\, f_b(\texttt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^{\kappa}\pi \wedge \psi\,)$ or $(\, M, s' \models f_b(\texttt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge x \wedge P^{\kappa}\epsilon \wedge \psi'\,)$, for some postcondition j of α, ψ not containing plan expressions or propositions from $\texttt{post}_{ij}(\alpha)$, and ψ' not containing plan expressions, execution flags, or those from $\texttt{post}_{ij}(\alpha)$. This is contrary to our assumption that the consequent does not hold. □

Proposition 5. *Axiom* **CP1** *is valid.*

Proof. Assume $M, s \models \neg x \wedge P^{\kappa}(\pi_{if}; \pi) \wedge f_b(\phi) \wedge \psi_{np}$. From Definition 11 and Proposition 1, we have $x \notin V_c(s),\ V_p(s) = \{\pi_{if}; \pi \triangleright \kappa\},\ V_b(s) \models_{cwa} \phi$, and $M, s \models \psi_{np}$. Condition **C7**, clause 1, requires there is an $R_{t(\phi)}$ transition to state s' where $V_p(s') = \{\pi_1; \pi \triangleright \kappa\}$, and hence according to Definition 11 and Proposition 1, $M, s' \models \langle t(\phi)\rangle P^{\kappa}\pi_1; \pi$. Moreover, as the only expressions for which the valuation are changed as a result of this transition are plan expressions, we have for all other expressions ψ_{np} if $M, s \models \psi_{np}$ then $M, s' \models \psi_{np}$. Putting together $M, s' \models \langle t(\phi)\rangle P^{\kappa}\pi_1; \pi \ \wedge\ \psi_{np}$ for all non-plan expressions ψ_{np}, Definition 10 states that only those $R_{t(\phi)}$ transitions exist that satisfy conditions **C3, C5, C7** and **C8**. However, since only **C7** applies (note that **C3** and **C5** require $M, s \models P^{\kappa}\phi?; \pi$ and **C8** requires $M, s \models x$ which are contrary to our assumption), we also have $M, s' \models [t(\phi)]P^{\kappa}\pi_1; \pi \ \wedge\ \psi_{np}$ for all non-plan expressions ψ_{np}. □

Proposition 6. *Axiom* **CP4** *is valid.*

[5]Note there there exists only one plan and this plan does not start with $u \in Ac$

Proof. Assume $M, s \models \neg x \wedge P^{\kappa}(\pi_{wh}; \pi) \wedge \neg f_b(\phi) \wedge \psi_{np}$. From Definition 11 and Proposition 1, we have $x \notin V_c(s)$, $V_p(s) = \{\pi_{wh}; \pi \triangleright \kappa\}$, $V_b(s) \not\models_{cwa} \phi$, and $M, s \models \psi_{np}$. Condition **C7**, clause 4, requires there is an $R_{t(\neg\phi)}$ transition to state s' where $V_p(s') = \{\pi \triangleright \kappa\}$, and hence according to Definition 11 and Proposition 1, $M, s' \models \langle t(\neg\phi)\rangle P^{\kappa}\pi$. Moreover, since the only expressions for which the valuation is changed are plan expressions, we have for all other expressions ψ_{np} if $M, s \models \psi_{np}$ then $M, s' \models \psi_{np}$. Putting together $M, s \models \langle t(\neg\phi)\rangle P^{\kappa}\pi \ \wedge \ \psi_{np}$ for all non-plan expressions ψ_{np}. Definition 10 states that only those $R_{t(\neg\phi)}$ transitions exist that are implied by conditions **C3, C5, C7** and **C8**. However, since only **C7** applies (note that **C3** and **C5** require $M, s \models P^{\kappa}\phi?; \pi$ and **C8** requires $M, s \models x$ which are contrary to our assumption), we also have $M, s' \models [t(\neg\phi)]P^{\kappa}\pi \wedge \psi_{np}$ for all non-plan expressions ψ_{np}. □

Proposition 7. *Axiom* **PG1** *is valid.*

Proof. Assume $M, s \models P\epsilon \ \wedge \ f_g(\kappa_i) \ \wedge f_b(\beta_i) \ \wedge \psi_{npx}$. From Definition 11 and Proposition 1, we have $V_p(s) = \{\}, V_g(s) \models \kappa_i$, $V_b(s) \models_{cwa} \beta_i$, and $M, s \models \psi_{npx}$. According to condition **C9**, there exists an $R_{\delta_{r_i}}$ transition to a state s' where $V_p(s') = \{\pi_i \triangleright \kappa_i\}$ and $x \notin V_c(s') \overset{def.}{\Leftrightarrow}{}^{11} \exists s' : R_{\delta_{r_i}}(s, s')$ such that $M, s' \models P^{\kappa_i}\pi_i \wedge \neg x \overset{def.}{\Leftrightarrow}{}^{11}$ $M, s' \models \langle\delta_{r_i}\rangle P^{\kappa_i}\pi_i \wedge \neg x$. Moreover, as only formulas containing plan expressions and x change as a result of this transition, we have for all other formulas ψ_{npx}, if $M, s \models \psi_{npx}$ then $M, s' \models \psi_{npx}$. Putting together $M, s' \models \langle\delta_{r_i}\rangle P^{\kappa_i}\pi_i \wedge \neg x \wedge \psi_{npx}$ for all formulas ψ_{npx} not containing plan expressions or x. Finally, since only **C9** applies, we have also $M, s' \models [\delta_{r_i}]P^{\kappa_i}\pi_i \wedge \neg x \wedge \psi_{npx}$ for all formula ψ_{npx} not containing plan expressions or execution flag. □

Proposition 8. *Axiom* **PG2** *is valid.*

Proof. Assume $M, s \models \neg P\epsilon \vee \neg f_g(\kappa_i) \vee \neg f_b(\beta_i)$, but $\neg[\delta_{r_i}]\bot \Leftrightarrow \langle\delta_{r_i}\rangle\top$. Then there exists an $R_{\delta_{r_i}}$ transition to a state s'. However, according to Definition 9, the only $R_{\delta_{r_i}}$ transitions are those implied by condition **C9** which requires $V_p(s) = \{\}, V_g(s) \models \kappa_i, V_b(s) \models_{cwa} \beta_i$, and hence according to Definition 11 and Proposition 1, $M, s \models P\epsilon \wedge f_g(\kappa_i) \wedge f_b(\beta_i)$. However, this contradicts our assumption. □

Proposition 9. *Axiom* **PR1** *is valid.*

Proof. Assume $M, s \models x \wedge P^{\kappa}\pi_j \wedge f_b(\beta_j) \wedge \psi_{npx}$. From Definition 11 and proposition 1, we have $x \in V_c(s), V_p(s) = \{\pi_j \triangleright \kappa\}, V_b(s) \models_{cwa} \beta_j$, and $M, s \models \psi_{npx}$. According to condition **C10**, there exists an $R_{\delta_{p_j}}$ transition to a state s' where $V_p(s') = \{\pi'_j \triangleright \kappa\}$ and $x \notin V_c(s') \overset{def.}{\Leftrightarrow}{}^{11} \exists s' : R_{\delta_{p_j}}(s, s')$ such that $M, s' \models P^{\kappa}\pi'_j \wedge \neg x \overset{def.}{\Leftrightarrow}{}^{11}$ $M, s' \models \langle\delta_{p_j}\rangle P^{\kappa}\pi'_j \wedge \neg x$. Moreover, since only formulas containing plan expressions and x change as a result of this transition, we have for all other formulas ψ_{npx}, if $M, s \models \psi_{npx}$ then $M, s' \models \psi_{npx}$. Putting together $M, s' \models \langle\delta_{p_j}\rangle P^{\kappa}\pi'_j \wedge \neg x \wedge \psi_{npx}$ for all formulas ψ_{npx} not containing plan expressions or boolean flag x. Finally, since only **C10** can imply $R_{\delta_{p_j}}$ transitions, we also have $M, s' \models [\delta_{p_j}]P^{\kappa}\pi'_j \wedge \neg x \wedge \psi_{npx}$ for all formula ψ_{npx} not containing plan expressions or x. □

Proposition 10. *Axiom* **PR2** *is valid.*

Proof. Assume $M, s \models \neg x \vee \neg P^{\kappa}\pi_j \vee \neg f_b(\beta_j)$, but $\neg[\delta_{p_j}]\bot$ which is equivalent to $\langle\delta_{p_j}\rangle\top$. Then there exists a $R_{\delta_{p_j}}$ transition to a state s'. However, according to Definition 9, the only $R_{\delta_{p_j}}$ transitions are those implied by condition **C10** which requires $x \notin V_c(s), V_p(s) = \{\pi_j \triangleright \kappa\}, V_b(s) \models_{cwa} \beta_j$, and hence according to Definition 11 and Proposition 1, $M, s \models \neg x \wedge P^{\kappa}\pi_j \wedge f_b(\beta_i)$. However, this contradicts our assumption. □

This concludes our discussion of the soundness of the axiomatic system for PDL-3APL. The rest of this section will be devoted to the proof of weak completeness of the axiomatisation.

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 prove weak completeness: every valid formula ϕ is derivable ($\models \phi \Rightarrow \vdash \phi$).

The proof is based on the standard completeness proof for PDL, see for example [17]. We show that any PDL-3APL consistent formula ϕ has a satisfying model M^{ϕ}. The construction proceeds as follows. Given a formula ϕ, we define a set of formulas $Closure(\phi)$ (defined below) which is similar to a set used in the standard PDL completeness proof, but has some extra conditions specific to PDL-3APL. The satisfying model M^{ϕ} for ϕ is constructed using $Closure(\phi)$, namely the states of M^{ϕ} are maximal consistent subsets of $Closure(\phi)$. We define the assignments and binary relations for M^{ϕ} similarly to the standard PDL proof. We omit the proofs of the standard PDL-specific lemmas. Instead, we concentrate on the PDL-3APL specific part of the proof which shows that M^{ϕ} satisfies conditions **C1**-**C10** on $\mathbf{M}_{3APL}(R)$ models.

The set $Closure(\phi)$ used in the construction of M^{ϕ} includes subformulas of ϕ and a finite number of other formulas specified below. First of all, we define the set $Subf(\phi)$ of subformulas of ϕ in the usual way, but considering subformulas of the form Bp, Gp and $G-p$ as atomic formulas (that is, p and $-p$ are not included in the set of subformulas). Conditions 1–4 below correspond to the Fischer-Ladner closure conditions used in the standard completeness proof for PDL [18], condition 5 corresponds to closure under single negations, and conditions 6–12 are specific to the proof for PDL-3APL.

1. $Subf(\phi) \subseteq Closure(\phi)$
2. if $\langle\rho_1;\rho_2\rangle\psi \in Closure(\phi)$ then $\langle\rho_1\rangle\langle\rho_2\rangle\psi \in Closure(\phi)$
3. if $\langle\rho_1 \cup \rho_2\rangle\psi \in Closure(\phi)$ then $\langle\rho_1\rangle\psi \vee \langle\rho_2\rangle\psi \in Closure(\phi)$
4. if $\langle\rho^*\rangle\psi \in Closure(\phi)$ then $\langle\rho\rangle\langle\rho^*\rangle\psi \in Closure(\phi)$
5. if $\psi \in Closure(\phi)$ and ψ is not of the form $\neg\chi$, then $\neg\psi \in Closure(\phi)$
6. if $G-p \in Closure(\phi)$, then $Bp \in Closure(\phi)$
7. $x \in Closure(\phi)$
8. if an action α occurs in ϕ, then $Closure(\phi)$ contains f_b translations of all pre- and postconditions for α, e.g., if one of α's preconditions is $\{p, -q\}$ then $Bp, \neg Bq \in Closure(\phi)$
9. $P\epsilon \in Closure(\phi)$

10. $P\pi \in Closure(\phi)$ for all $\pi \in Plan$
11. if $\langle t(\phi')\rangle\psi \in Closure(\phi)$ then $f_b(\phi') \in Closure(\phi)$
12. if $P^\kappa\pi \in Closure(\phi)$ then for all formulas $\phi' \in \langle query\rangle$ which occur in a plan expression in $Closure(\phi)$, $f_b(\phi') \in Closure(\phi)$

Given a consistent formula ϕ, we define M^ϕ as follows:

Definition 14 (Canonical PDL-3APL model for ϕ)**.** *A canonical PDL-3APL model for ϕ, M^ϕ is defined as*

$$M^\phi = (W^\phi, V^\phi, \mathcal{R}^\phi_\alpha, \mathcal{R}^\phi_{t(\phi)}, \mathcal{R}^\phi_{\bar{\alpha}}, \mathcal{R}^\phi_{\delta_{ri}}, \mathcal{R}^\phi_{\delta_{pj}})$$

where:

W^ϕ is the set of all maximal consistent subsets of $Closure(\phi)$

V^ϕ is defined as follows:

- $p \in V^\phi_b(A)$ *iff* $Bp \in A$, *where* $Bp \in Closure(\phi)$;
- $(-)p \in V^\phi_g(A)$ *iff* $G(-)p \in A$, *where* $G(-)p \in Closure(\phi)$;
- $V^\phi_p(A) = \{\pi \triangleright \kappa\}$ *iff* $P^\kappa\pi \in A$, *where* $P^\kappa\pi \in Closure(\phi)$; $V^\phi_p(A) = \emptyset$ *iff* $P\epsilon \in A$.

The transition relations $\mathcal{R}^\phi_\alpha, \mathcal{R}^\phi_{t(\phi)}, \mathcal{R}^\phi_{\bar{\alpha}}, \mathcal{R}^\phi_{\delta_{ri}}, \mathcal{R}^\phi_{\delta_{pj}}$ are defined as follows. Let A, B be maximal consistent subsets of $Closure(\phi)$, and ρ be a program expression. We first define auxiliary relations $S_\rho(A, B)$ for each program expression ρ as follows. Let us denote by $\wedge A$ the conjunction of all formulas in A. Then $S_\rho(A, B)$ holds if and only if $\wedge A$ is consistent with $\langle\rho\rangle \wedge B$ (the conjunction of formulas in B preceded by $\langle\rho\rangle$).

Now using the auxiliary relations, we define relations R^ϕ_ρ:

- *for every u= α, $t(\phi)$, $\bar{\alpha}$, δ_{ri}, δ_{pj}: $R^\phi_u = S_u$;*
- $R^\phi_{\rho_1;\rho_2} = R^\phi_{\rho_1} \circ R^\phi_{\rho_2}$ *where $\circ$ is relational composition;*
- $R^\phi_{\rho_1\cup\rho_2} = R^\phi_{\rho_1} \cup R^\phi_{\rho_2}$;
- $R^\phi_{\rho^*} = (R^\phi_\rho)^*$.

The following three lemmas have exactly the same proof as lemmas 4.88, 4.89 and 4.90 in [17]:

Lemma 1 (Inclusion Lemma)**.** *For every ρ, $S_\rho \subseteq R^\phi_\rho$.*

Lemma 2 (Existence lemma)**.** *Let A be a maximal consistent set and ρ a program expression. Then for all $\langle\rho\rangle\psi \in Closure(\phi)$, $\langle\rho\rangle\psi \in A$ iff there is a maximal consistent set A' such that $R^\phi_\rho(A, A')$ and $\psi \in A'$.*

Lemmas 1 and 2 are used in the proof of the following crucial lemma:

Lemma 3 (Truth lemma)**.** *Let $\psi \in Closure(\phi)$. Then for every maximal consistent set A: $M^\phi, A \models \psi$ iff $\psi \in A$.*

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^ϕ.

The last component for the completeness proof is the following proposition:

Proposition 11. *M^ϕ satisfies conditions **C1–C10**.*

Proof. Since every A is consistent with respect to **A1**, A cannot contain Bp and Gp so it is not possible that $p \in V_b^\phi(A)$ and $p \in V_g^\phi(A)$. Similarly, if $G{-}p \in A$, then $Bp \in A$ by **A2**, so condition **C1** holds in M^ϕ. By **A3a** and **A3b** each state contains a unique plan expression.

Let us prove that **C2** holds in M^ϕ. Suppose $V_p^\phi(A) = \{\alpha; \pi \triangleright \kappa\}$, $V_b^\phi(A)$ entails $\mathtt{prec}_i(\alpha)$ and $x \notin V_c^\phi(A)$. We need to show that for every $\mathtt{post}_{ij}(\alpha)$ there exists a maximal consistent set B in M^ϕ such that

1. $V_b(B) = T_{i,j}(\alpha, V_b(A))$
2. $V_g(B) = V_g(A) \setminus (\{p : p \in V_b(B)\} \cup \{-p : p \notin V_b(B)\})$
3. if $V_b(B) \not\models_{cwa} \kappa$, then $V_p(B) = \{\pi \triangleright \kappa\}$
4. if $V_b(B) \models_{cwa} \kappa$, then $x \in V_c(B)$ and $V_p(B) = \{\}$

Given the construction of M^ϕ, the conditions on B above are equivalent to the following:

1. $f_b(\mathtt{post}_{ij}(\alpha)) \in B$ and for every p which does not occur in $f_b(\mathtt{post}_{ij}(\alpha))$, $Bp \in B$ iff $Bp \in A$
2. if $p \in \mathtt{post}_{ij}(\alpha)$, $Gp \notin B$; if $-p \in \mathtt{post}_{ij}(\alpha)$, $G{-}p \notin B$; for all other p, $G(-)p \in B$ iff $G(-)p \in A$
3. if $\neg f_b(\kappa) \in B$, then $P^\kappa\pi \in B$
4. if $f_b(\kappa) \in B$, then $x \in B$ and $P\epsilon \in B$

Recall that for any set B, $R_\alpha^\phi(A, B)$ if $\wedge A \wedge \langle\alpha\rangle \wedge B$ is consistent. We need to show that there is a maximal consistent set of formulas B which satisfies this condition and also either conditions 1, 2 and 3 or 1, 2 and 4 above.

From $V_p^\phi(A) = \{\alpha; \pi \triangleright \kappa\}$ and the model definition we conclude that $P^\kappa\alpha; \pi \in A$. Since $x, f_b(\mathtt{prec}_i(\alpha)) \in Closure(\phi)$, $f_b(\mathtt{prec}_i(\alpha)) \in A$ and $\neg x \in A$. So $\wedge A$ implies $P^\kappa\alpha; \pi \wedge f_b(\mathtt{prec}_i(\alpha)) \wedge \neg x$. Let F_j be the set of p which do not occur in $\mathtt{post}_{ij}(\alpha)$. Then $\wedge A$ also implies the following formula ψ_A, describing the set of beliefs and goals true in A which are not going to be affected by executing α if the outcome is described by the postcondition j:

$$\psi_A = \bigwedge_{p \in F_j, Bp \in A} Bp \wedge \bigwedge_{p \in F_j, G(-)p \in A} G(-)p$$

Consider an instance of the axiom **BA1** where for the formula ψ we use $\psi_A \wedge \neg x$ and for the formula ψ' we use ψ_A. Clearly, A implies its antecedent:

$$\neg x \wedge P^\kappa\alpha; \pi \wedge f_b(\mathtt{prec}_i(\alpha)) \wedge (\psi_A \wedge \neg x) \wedge \psi_A$$

This means that $\wedge A$ implies the consequent of the axiom. Since A is consistent, this means that $\wedge A$ is consistent either with

$$\langle\alpha\rangle(f_b(\mathtt{post}_{ij}(\alpha)) \wedge \neg f_b(\kappa) \wedge P^\kappa\pi \wedge \neg x \wedge \psi_A)$$

or with

$$\langle\alpha\rangle(f_b(\mathtt{post}_{ij}(\alpha)) \wedge f_b(\kappa) \wedge P\epsilon \wedge x \wedge \psi_A)$$

Note that because of axioms **A1** and **A2** if $\wedge A$ is consistent with $\langle\alpha\rangle(Bp \wedge \neg Bq)$ it is also consistent with $\langle\alpha\rangle(Bp \wedge \neg Gp \wedge \neg Bq \wedge \neg G-q)$, so all negative statements about goals which are dropped because of executing α can also be added. Note that those statements only involve variables which are not in F_j. In either case, we have that $\wedge A \wedge \langle\alpha\rangle\psi$ is consistent, where ψ is a formula which describes the conditions on the desired state B. The only problem is that ψ does not contain, for every single formula from $Closure(\phi)$, this formula or its negation as a conjunct; in other words, it does not yet describe a maximal consistent set. However, using a standard technique called 'forcing choices' we can extend ψ to such a set. Namely, we enumerate all formulas from $Closure(\phi)$: $\chi_1, \ldots, \chi_n$. Set $\psi_0 = \psi$. For each formula χ_i either $\wedge A \wedge \langle\alpha\rangle(\psi_{i-1} \wedge \chi_i)$ or $\wedge A \wedge \langle\alpha\rangle(\psi_{i-1} \wedge \neg\chi_i)$ is consistent, provided $\wedge A \wedge \langle\alpha\rangle\psi_{i-1}$ is consistent (from basic modal logic). Construct a conjunction containing every formula from $Closure(\phi)$ or its negation by extending ψ while maintaining consistency. Finally, ψ_n will give us a conjunction of formulas in a maximal consistent set B which satisfies the conditions of **C2**.

This argument shows that R_α transitions required by the model conditions exist in M^ϕ. We also need to show that only those R_α transitions do exist from non-x states. Axiom **BA2a** makes sure that there are no R_α transitions from the states where the plan does not start with α (if the conjunction of formulas in A implies $[\alpha]\bot$, this means that this conjunction is not consistent with any formula of the form $\langle\alpha\rangle\psi$). Axiom **BA3** ensures that all states reachable by α are as described in **C2**.

Next consider condition **C3**. Axiom **BA2b** ensures that $R_{t(\phi')}$ transitions do not exist from the states where the plan expression does not start with a test whether ϕ' holds, and **BA4** ensures that if A contains $P^\kappa(\phi'?;\pi)$ and $f_b(\phi')$, then there exists a $R_{t(\phi')}$ transition to a state B which is identical to A except that the plan expression is $P^\kappa(\pi)$, and only to such a state B.

C4 and **C5** require that if x is false and the next step in the current plan is not executable (preconditions of a belief update action do not hold or the test formula in a belief test action is false) then the transition by this step should lead to a state which is identical to the current one except that x holds there. This is ensured by axioms **BA5** and **BA6**.

The model satisfies condition **C6** because by **BA7** a state A can have an abstract plan $\bar{\alpha}$ link to another state B if B is exactly like A apart from being an x-state. Axiom **BA2a** makes sure that an $\bar{\alpha}$ transition is only possible if the current plan has $\bar{\alpha}$ as its first step.

If A contains $P^\kappa\pi;\pi'$ where π is a conditional plan with a test on ϕ', axioms **CP1** — **CP4** ensure that there exist $R_{t((\neg)\phi')}$ transitions required by condition **C7**, and the only states reachable by them are as described in the condition. **CP5** disables an $R_{t(\phi')}$

transition if π tests on ϕ' and $f_b(\phi')$ is false, and vice versa, disables $R_{t(\neg\phi')}$ transition if π tests on ϕ' and $f_b(\phi')$ is true.

In x-states, all actions and tests should be executable without changing the state. In other words every A such that $x \in A$ should have an R_u transition to itself for every $u = \alpha$, $t(\phi)$, $\bar{\alpha}$. Clearly the conjunction of formulas in A is consistent with $\langle u \rangle \wedge A$, otherwise it will be inconsistent with axiom **BA8**. So condition **C8** holds.

To satisfy condition **C9**, we need to ensure that first, the required transitions by $R_{\delta_{r_i}}$ exist, and second, that only those transitions exist. Axiom **PG1** ensures that any A such that $P\epsilon \in A$, where A satisifies the belief and goal conditions of the PG rule r_i, has a transition to a state which is identical to A apart from replacing $P\epsilon$ with the new plan expression and possibly not containing x. If the conjunction of formulas in A is not consistent with the description of the resulting state, then A is not consistent with **PG1**. **PG1** also makes sure that *all* $R_{\delta_{r_i}}$ transitions lead to such a state. Finally, **PG2** makes sure that other states (which do not have an empty plan base, or do not satisfy the belief and goal conditions of r_i) do not have any outgoing $R_{\delta_{r_i}}$ transitions.

Similarly for condition **C10**: **PR1** ensures that if $x \in A$ and A contains formulas which mean that the plan and belief condition of a PR rule p_j match, then there is a $R_{\delta_{p_j}}$ transition to a state which is identical to A apart from replacing the plan expression with the new one, and not containing x, and only such $R_{\delta_{p_j}}$ exist. To make sure there are no $R_{\delta_{p_j}}$ transitions from any other states, we have axiom **PR2**. □

Lemma 3 and Proposition 11 give the proof of the weak completeness of PDL-3APL:

Theorem 2. *PDL-3APL is weakly complete with respect to* $\mathbf{M}_{3APL}(R)$.

Proof. We have shown that every formula ϕ consistent with respect to PDL-3APL axioms has a model M^ϕ (since by Lemma 3, ϕ is satisfied in at least one state of M^ϕ). By Proposition 11, M^ϕ satisfies conditions **C1**–**C10**, in other words, it is a $\mathbf{M}_{3APL}(R)$ model. We have shown that if $\nvdash \neg\phi$, then ϕ's negation is not true in all models: $\not\models \neg\phi$. By contraposition, if $\neg\phi$ is valid, namely if $\models \neg\phi$, then its negation is provable: $\vdash \neg\phi$. Since every PDL-3APL formula is equivalent to its double negation, we have $\models \phi$ implies $\models \neg\neg\phi$ implies $\vdash \neg\neg\phi$ implies $\vdash \phi$. □

The definitions of PDL-3APL models and the set of axioms are quite complex. The models have transitions corresponding to the various components of 3APL programs (belief update actions, tests, abstract plans, planning goal rules and plan revision rules) and the states have the beliefs, goals and plans of the agent. Each of those components requires a relatively small set of conditions (motivated by the operational semantics) and a couple of axioms. However, due to the complexity of 3APL, the complete list of axioms is rather long. On the positive side, this does provide us with a complete system for reasoning about the whole language, rather than just a fragment of it as in [11]. In the remainder of the paper, we explain how 3APL programs can be translated into PDL-3APL and how to verify 3APL programs using theorem-proving in PDL-3APL.

4. Translating 3APL Programs into PDL-3APL

In this section, we give a translation of a complete 3APL agent program into a PDL-3APL program expression. In this and in subsequent sections, we will use R as a name both for a program (and the agent's initial configuration) and its signature (when we talk about the models of the logic corresponding to this signature).

Recall that in the initial configuration, the agent has an empty plan base, and, with a non-interleaved execution strategy, execution of the agent's program proceeds by the adoption and execution of a single appropriate plan (as determined by the current beliefs and goals of the agent). If the goal for the currently executing plan is achieved, or the plan becomes non-executable, the plan may be dropped or revised by a plan revision rule. The execution of an agent program R can therefore be translated into PDL-3APL program expressions corresponding to the application of a PG rule and the execution of the corresponding plan, possibly interleaved with the application of a PR rule and the consequent revision of the plan base

$$(\cup_i(\delta_{ri}; f_p(\pi_i)) \bigcup \cup_j(\delta_{p_j}; f_p(\pi'_j)))^+$$

where i ranges over all PG rules and j ranges over all PR rules in the program. We will refer to this expression as $tr(R)$. $tr(R)$ picks out exactly those paths in a model which correspond to an execution of the program.

To show that our translation is faithful with respect to the operational semantics, we prove a correspondence theorem relating transition systems generated by the operational semantics for program R and models in $\mathbf{M}_{3APL}(R)$. First of all, note that there is a clear correspondence relation between configurations in the operational semantics and states in $\mathbf{M}_{3APL}(R)$ models. Namely, given a configuration $c = (\sigma, \gamma, \Pi)$, we say that $s \sim c$ for a state s in a model M if s has the same belief, goal and plan assignments as c.

We will also talk about matching paths in the operational semantics and in $\mathbf{M}_{3APL}(R)$ models. By a path in an operational semantics transition system $\mathcal{S}$, we mean a sequence of configurations $c_1, label_1, c_2, label_2, \ldots, c_m$ where c_{j+1} is obtained from c_j by one of the transition rules (1a)-(6c). For convenience, we label each transition by the corresponding operation; the labels are, (1a, α), (1b, α), (2, $t(\beta)$), (3a, $t(\phi)$), (3b, $t(\neg\phi)$), (4a, $t(\phi)$), (4b, $t(\neg\phi)$), (5, δ_{ri}), (6a, $\delta_{p_j}(\alpha)$), (6b, $\delta_{p_j}(t(\beta))$), (6c, $\delta_{p_j}(\bar{\alpha})$). We claim that if there is a path $c = c_1, \ldots, c_n = c'$ in $\mathcal{S}$ with a certain sequence of labels, then there is a corresponding path $s = s_1, \ldots, s_k = s'$ in M such that $s \sim c$ and $s' \sim c'$. It remains to define what we mean by a 'corresponding path'.

Definition 15. *Let $c_j, label_j, c_{j+1}$ be a single step on a path in the 3APL operational semantics transition system $\mathcal{S}$. The corresponding path $s_j, \ldots, s_{j+i}$ in M depends on $label_j$ and is defined as follows.*

- *c_j, (1a,α), c_{j+1}: the corresponding path is s_j, s_{j+1} where $s_j \sim c_j$ and $s_{j+1} \sim c_{j+1}$ and $R_\alpha(s_j, s_{j+1})$.*
- *c_j, (1b,α), c_{j+1}: the corresponding path is $s_j, s_{j+1}, \ldots, s_{j+1}$ where $s_j \sim c_j$ and $s_{j+1} \sim c_{j+1}$ and $R_\alpha(s_j, s_{j+1})$, s_{j+1} satisfies x and has an empty plan*

base, and is repeated as many times as there are steps remaining in the plan when the goal has been achieved; the transitions from s_{j+1} to s_{j+1} on the path correspond to the remaining steps in the plan.[6]

- c_j, *(2,* $t(\beta)$*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\beta)}(s_j, s_{j+1})$.
- c_j, *(3a,* $t(\phi)$*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\phi)}(s_j, s_{j+1})$.
- c_j, *(3b,* $t(\neg\phi)$*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\neg\phi)}(s_j, s_{j+1})$.
- c_j, *(4a,* $t(\phi)$*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\phi)}(s_j, s_{j+1})$.
- c_j, *(4b,* $t(\neg\phi)$*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\neg\phi)}(s_j, s_{j+1})$.
- c_j, *(5,* δ_{r_i}*),* c_{j+1}*: the corresponding path is* s_j, s_{j+1} *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{\delta_{r_i}}(s_j, s_{j+1})$.
- c_j, *(6a,* $\delta_{p_k}(\alpha)$*),* c_{j+1}*: the corresponding path is* $s_j, t, \ldots, t, s_{j+1}$ *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_\alpha(s_j, t)$, $R_{\delta_{p_k}}(t, s_{j+1})$, t *is an* x*-state, and the transitions from* t *to* t *on the path correspond to the remaining steps in the original plan.*
- c_j, *(6b,* $\delta_{p_k}(t(\beta))$*),* c_{j+1}*: the corresponding path is* $s_j, t, \ldots, t, s_{j+1}$ *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{t(\beta)}(s_j, t)$, $R_{\delta_{p_k}}(t, s_{j+1})$, t *is an* x*-state, and the transitions from* t *to* t *on the path correspond to the remaining steps in the original plan.*
- c_j, *(6c,* $\delta_{p_k}(\bar{\alpha})$*),* c_{j+1}*: the corresponding path is* $s_j, t, \ldots, t, s_{j+1}$ *where* $s_j \sim c_j$ *and* $s_{j+1} \sim c_{j+1}$ *and* $R_{\bar{\alpha}}(s_j, t)$, $R_{\delta_{p_k}}(t, s_{j+1})$, t *is an* x*-state, and the transitions from* t *to* t *on the path correspond to the remaining steps in the original plan.*

The following theorem states that if a transition system $\mathcal{S}$ and a PDL-3APL model M are generated by the same initial configuration, then the set of configurations with an empty plan base reachable from the initial state in $\mathcal{S}$ is the same as the set of states

[6] We require this to ensure that states with empty plan bases are reachable by a path in $tr(R)$, where paths are the executions of PDL program expressions (which correspond to every plan in states that achieves its goal) to completion. For example, if in configuration c with an empty belief base, a goal base containing p, and a plan $\alpha_1; \alpha_2$ with its associated goal p, executing α_1 achieves p, then in the resulting configuration c' the belief base contains p, the goal base and the plan base are empty. In the model, we go from a state s matching c to an x-state s' with the same belief, goal and plan assignment as in c', but we make an extra step to 'consume' the rest of the PDL program expression, namely α_2, so the corresponding path in a model is s, α_1, s', α_2, s'. Note that this way s' is reachable by a path $\alpha_1; \alpha_2$ which corresponds to a completely executed PDL program expression.

reachable from the initial state by $R_{tr(R)}$. (By a model generated by a state s_0 we mean a model where all states are reachable from s_0).

Theorem 3. *Let R be a 3APL program, $\mathcal{S}$ a transition system generated by the operational semantics of R with initial configuration c_0. Let M be a model generated by state $s_0 \sim c_0$. Then a configuration c with an empty plan base is reachable in $\mathcal{S}$ from c_0 iff a state $s \sim c$ is reachable in M from s_0 by $tr(R)$.*

Proof. The theorem has two directions:

1. If $s_0 \sim c_0$ and there is a path in $\mathcal{S}$ from c_0 to c where c has an empty plan base, then there is an s with $s \sim c$ such that $R_{tr(R)}(s_0, s)$.
2. If $s_0 \sim c_0$ and $R_{tr(R)}(s_0, s)$, then there exists a path in $\mathcal{S}$ from c_0 to c such that $s \sim c$.

1. Assume $s_0 \sim c_0$ and there is a path in $\mathcal{S}$ from c_0 to c where c has an empty plan base. We need to show that there is an s with $s \sim c$ such that $R_{tr(R)}(s_0, s)$.

The proof is by induction on the number of labels in the path in $\mathcal{S}$, using the preconditions of the transitions of the operational semantics and conditions on $M_{3APL}(R)$. We show that for every configuration c, the set of transitions possible in c is included in the set of transitions possible in a state $s \sim c$, and moreover the configurations reachable from c are in the relation $\sim$ with the states reachable by the corresponding transitions from s and the resulting path is in $R_{tr(R)}$.

The initial configuration c_0 has an empty plan base, and the only applicable transition is a step by (5, δ_{ri}) to a configuration s_1 with plan π_i. It is easy to see that given that $s_0 \sim c_0$ and condition **C9** on models, there is a corresponding step in M to a state s_1 such that $s_1 \sim c_1$. Note that $tr(R)$ is a union of $\delta_{ri}; f_p(\pi_i)$ with $\delta_{pj}; f_p(\pi_j)$. We have now made a first step in M which matches an initial segment δ_{ri} of $\delta_{ri}; f_p(\pi_i)$. In order for the matching path in M to be in $tr(R)$, it has to contain all the steps in $f_p(\pi_i)$. If the next steps on a path in $\mathcal{S}$ correspond to a 'normal' execution of π_i, that is, they follow the operational semantics rules (1a), (2), (3a), (3b), (4a), (4b), then matching steps exist in M by conditions **C2**, **C3**, **C7**. If the goal of the plan is achieved by α before the plan finishes executing, the step in $\mathcal{S}$ is of the form (1b, α). By **C2**, a transition to an x-state exists, and from the x-state there is a path to itself corresponding to the rest of $f_p(\pi_i)$ steps by condition **C8**. The final state on this path is in the $\sim$ relation to a configuration with an empty plan base in $\mathcal{S}$, so the inductive hypothesis applies. Similarly, if one of the steps in plan π_i in $\mathcal{S}$ blocks, we have one of the steps (6a, $\delta_{pk}(\alpha)$), (6b, $\delta_{pk}(t(\phi))$), (6c, $\delta_{pk}(\bar{\alpha})$) to a configuration c' with plan π'_j. In all of those cases, in M by one of **C4**, **C5**, **C6**, there is a transition to an x-state t, and from t a path to itself by the rest of $f_p(\pi_i)$ by **C8**, and there is also a transition from t to a state s' with $s' \sim c'$ by **C10**. In this case, we start 'tracing' a $\delta_{pj}; f_p(\pi'_j)$ path in M, in exactly the same way as for $\delta_{ri}; f_p(\pi_i)$.

2. Assume $s_0 \sim c_0$ and $R_{tr(R)}(s_0, s)$. We need to show that there exists a path in $\mathcal{S}$ from c_0 to c such that $s \sim c$. Again observe that every path where the begin and end states are in $R_{tr(R)}(s_0, s)$ starts with an δ_{ri} or δ_{pj} transition, and since c_0 has an empty plan base, so does s_0, so there are no $R_{\delta_{pj}}$ edges out of s_0 and the only possibility is an δ_{ri} transition. Then since the belief and goal bases are the same in c_0 and s_0, there

is a matching transition possible from c_0, to a configuration c_1 where the plan is π_i. The next step in M is the first transition in $f_p(\pi_i)$ to some state s'. If s' is not an x state, that is, the transition is enabled by one of **C2** (clause 3), **C3**, or **C7**, then there is a matching step in $\mathcal{S}$ to a matching c' by one of (1a), (2), (3a)–(4b). If s' is an x state by **C2** (clause 4), then there is a matching step in $\mathcal{S}$ to c' by (1b), $s' \sim c'$. If s' is an x state by **C4**, **C5**, or **C6**, then there is a transition from s' to s'' by $R_{\delta_{p_j}}$ and a matching step in $\mathcal{S}$ to c' by (6a), (6b) or (6c), so that $s'' \sim c'$. □

5. Verification of 3APL Programs

Given the above translation of 3APL programs into PDL-3APL, we can verify properties of agent programs, such as 'all executions of a program R result in a state satisfying property ψ', $[tr(R)]\psi$, or 'there is an execution of R which achieves ψ', $\langle tr(R)\rangle\psi$. More precisely, we can show 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. Since we are using PDL, we cannot inspect every state along the path corresponding to an agent's execution. Rather we sample the states where either a PG or a PR rule are about to be applied. Note that these states are uniquely determined both in the operational semantics and in the $\mathbf{M}_{3APL}(R)$ models: in case of PG rules, they have an empty plan base; in case of PR rule, the plan base contains a plan the first action of which is not executable. In other words, we sample states at the end of the agent's 'cycle', which consists of firing a PG or PR rule and executing the corresponding plan to completion or to the point of exceptional termination when some plan action is not executable or the goal of the plan has been achieved half-way through execution.

To verify properties of 3APL programs expressed in PDL-3APL we can use a theorem prover such as MSPASS or PDL-TABLEAU [19, 20].[7] Note that while there are currently no PDL theorem provers which can interpret belief, goal and plan modalities, formulas starting with these modalities can always be encoded as propositional variables with extra axioms.

As an illustration, we show how we can prove properties of the simple agent program introduced above. We shall use the following abbreviations for propositions: a for `attendConference`, p for `paper`, c for `clearance`, d for `deadlinePast`, s for `submitted`, y for `accepted`, f for `fly` and t for `ticket`. In addition, we use the following abbreviations for belief update actions: wP for `writePaper`, rC for `requestClearance`, sP for `submitPaper`, rP for `revisePaper`, bT for `buyPlaneTicket`, fC for `flyToConference`, and ttC for the abstract plan `travelToConference`. The

[7] 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 PDL are readily available.

agent's program consists of 6 belief update actions:

$$\begin{array}{rll} \{-p,-d\} & wP & \{\{p\},\{p,d\}\} \\ \{p,-d,-c\} & rC & \{\{\,\},\{d\},\{c\},\{d,c\}\} \\ \{p,-d,c\} & sP & \{\{\,\},\{y\}\} \\ \{p,-d\} & rP & \{\{\,\},\{d\}\} \\ \{-t\} & bT & \{\{t\}\} \\ \{t\} & fC & \{\{a\}\} \end{array}$$

a single PG rule r_1:

$$r_1 \;=\; a \;\leftarrow\; -p \;\texttt{and}\; -d \mid wP; rC; sP; \texttt{if}\; y \;\texttt{then}\; ttC$$

and four PR rules:

$$\begin{array}{rcl} p_1 & = & rC; \pi_1 \;\leftarrow d \mid \epsilon \\ p_2 & = & sP; \pi_2 \;\leftarrow d \mid \epsilon \\ p_3 & = & sP; \pi_2 \;\leftarrow -d \;\texttt{and}\; -c \mid rP; rC; sP; \pi \\ p_4 & = & ttC; \leftarrow\; f \mid bT; fC \end{array}$$

where $\pi_1 = sP; \pi_2$ and $\pi_2 = \texttt{if}\; y \;\texttt{then}\; ttC$. The translation of the agent's program is then

$$\begin{array}{l} tr(R) = \\ \quad (\;\; (\delta_{r1}; wP; rC; sP; ((t(y); ttC) \cup t(\neg y)))\;\cup \\ \qquad \delta_{p_1} \cup \delta_{p_2} \cup \\ \qquad (\delta_{p3}; rP; rC; sP; ((t(y); ttC) \cup t(\neg y))\;\cup \\ \qquad (\delta_{p4}; bT; fC)\;\;)^+ \end{array}$$

To reason about the execution of the agent program we need instances of axioms for the pre- and postconditions of the agent's actions. For example the following instances of **BA1** defining pre- and postconditions of `submitPaper`:

$$\begin{array}{l} \neg x \wedge P^a(sP; ((t(y); ttC) \cup t(\neg y))) \wedge \neg Bp \wedge \neg Bd \wedge Bc \wedge \neg By \\ \rightarrow \langle sP\rangle(\neg x \wedge P^a((t(y); ttC) \cup t(\neg y)) \wedge \neg By) \end{array}$$

$$\begin{array}{l} \neg x \wedge P^a(sP; ((t(y); t) \cup t(\neg y)) \wedge \neg Bp \wedge \neg Bd \wedge Bc \rightarrow \\ \langle sP\rangle(\neg x \wedge P^a((t(y); ttC) \cup t(\neg y)) \wedge By) \end{array}$$

and the following instance of **BA3**:

$$\begin{array}{l} \neg x \wedge P^a(sP; ((t(y); ttC) \cup t(\neg y))) \wedge Bp \wedge \neg Bd \wedge Bc \rightarrow \\ [sP](\neg x \wedge P^a((t(y); ttC) \cup t(\neg y)) \wedge (\neg By \vee By)) \end{array}$$

If we specify the initial state as $\neg x \wedge P\epsilon \wedge \neg Ba \wedge \neg Bp \wedge \neg Bd \wedge \neg Bc \wedge \neg By \wedge \neg Bt \wedge Bf$ it is possible to prove (using pre- and postcondition axioms for other actions,

omitted for brevity), that, for example, the agent will only attend the conference if its paper is accepted:

$$\neg x \wedge P\epsilon \wedge \neg Bp \wedge \neg Bd \rightarrow \langle [tr(R)] \rangle (Ba \rightarrow By)$$

We can also prove that the agent *may* achieve its goal (intuitively, provided it receives clearance before the deadline, and has the paper accepted):

$$\neg x \wedge P\epsilon \wedge \neg Bp \wedge B-d \rightarrow \langle tr(R) \rangle Ba$$

Clearly, to prove that the agent is *guaranteed* to achieve its goal, we need additional assumptions which can be expressed as extra axioms: e.g., that agent always receives clearance and that papers it submits are always accepted.

6. Discussion

Our approach builds on and extends previous research on proving correctness of programs in APL-like and other BDI-based agent programming languages, e.g., [21, 11]. However much of this work has not considered plan revision. Levesque et al. [21] have described GOLOG, a high level agent programming language in which basic actions are defined by action precondition and successor state axioms. By reasoning in situation calculus, it is possible to prove that a (non-exceptional) execution of a GOLOG program will achieve the agent's goal. Recent work on verification of ConGolog programs [22] expresses ConGolog programs in a logic which extends situation calculus, dynamic logic and temporal logic, and proposes a model-checking algorithm for verifying properties of ConGolog programs. In [11, 12] Alechina et al. gave a complete axiomatisation of a logic for reasoning about a subset of 3APL, but without nondeterministic actions or plan revision rules. In [23] a dynamic logic for reasoning about programs written in Dribble was proposed. Dribble includes 'practical reasoning' rules which allow the revision of agent programs. However no axiomatisation of the logic was given. In [24], an axiomatisation of a logic for reasoning about Dribble programs is given, but based on a temporal logic CTL rather than on dynamic logic.

Recovery from plan execution failure has also been investigated in the agent programming literature. For example, in Jason [5] a plan execution failure generates a goal deletion event that may initiate "clean up" plans prior to attempting another plan for the goal. In CANPLAN2 [15] if the plan for a goal is deemed to have failed or is blocked (not executable), the plan is abandoned and any alternative plans for achieving the goal are tried. In [9] Thangarajah et al. present an integrated approach to aborting tasks and plan failure. Plans have an associated failure condition which triggers a *failure method* to clean up, before the plan is dropped and another plan tried. However while an operational semantics is given for failure recovery in [5, 15, 9], there is no corresponding logical analysis.

In contrast to these approaches, we have focused on a replanning mechanism that allows a failed plan to be either revised to allow execution to continue, or dropped (possibly after performing clean up actions) and another applicable plan tried. We have presented a logic that can be used to specify and verify properties of agent programs

that employ this replanning mechanism. We have provided a complete axiomatisation of the logic and, using simple examples, we have shown how it can be used to specify properties of agent programs under both normal and exceptional executions.

In this paper, we assumed that an agent executes a single plan at a time. In future work we intend to extend the framework to allow multiple plans whose executions can be interleaved along the lines of [12].

Acknowledgments

Natasha Alechina and Brian Logan were supported by EPSRC grant no. EP/E031226.

[1] M. Dastani, M. B. van Riemsdijk, F. Dignum, J.-J. C. Meyer, A programming language for cognitive agents: Goal directed 3APL, in: M. Dastani, J. Dix, A. E. Fallah-Seghrouchni (Eds.), Programming Multi-Agent Systems, First International Workshop, ProMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited papers, Vol. 3067 of LNCS, Springer, 2004, pp. 111–130. doi:http://www.springerlink.com/content/l7dqkvqh5u941l4b.

[2] M. Dastani, M. B. van Riemsdijk, J.-J. C. Meyer, Programming multi-agent systems in 3APL, in: R. H. Bordini, M. Dastani, J. Dix, A. E. Fallah-Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications, Vol. 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations, Springer, 2005, pp. 39–67.

[3] M. Dastani, 2APL: a practical agent programming language, Autonomous Agents and Multi-Agent Systems 16 (3) (2008) 214–248.

[4] M. Dastani, J.-J. C. Meyer, 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), Vol. 4908 of LNCS, Springer, 2008, pp. 107–123.

[5] R. H. Bordini, J. F. Hübner, R. Vieira, 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, Springer-Verlag, 2005, pp. 3–37.

[6] R. H. Bordini, J. F. Hübner, M. Wooldridge, Programming multi-agent systems in AgentSpeak using Jason, Wiley, 2007.

[7] P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas, JACK intelligent agents - components for intelligent agents in Java, AgentLink Newsletter (2) (1992) 2–5.

[8] D. Morley, K. Myers, The SPARK agent framework, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 714–721. doi:http://dx.doi.org/10.1109/AAMAS.2004.267.

[9] J. Thangarajah, J. Harland, D. Morley, N. Yorke-Smith, Aborting tasks in BDI agents, in: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'07), Honolulu, HI, 2007, pp. 8–15.

[10] J. Thangarajah, J. Harland, D. Morley, N. Yorke-Smith, Suspending and resuming tasks in BDI agents, in: Proceedings of the Seventh International Conference on Autonomous Agents and Multi Agent Systems (AAMAS'08), Estoril, Portugal, 2008, pp. 405–412.

[11] N. Alechina, M. Dastani, B. Logan, J.-J. C. Meyer, A logic of agent programs, in: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI 2007), AAAI Press, 2007, pp. 795–800.

[12] N. Alechina, M. Dastani, B. Logan, J.-J. C. Meyer, 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), AAAI, Sydney, Australia, 2008, pp. 16–26.

[13] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.

[14] A. S. Rao, AgentSpeak(L): BDI agents speak out in a logical computable language, in: Proceedings of Modelling Autonomous Agents in a Multi-Agent World, no. 1038 in LNAI, Springer Verlag, 1996, pp. 42–55.

[15] S. Sardiña, L. Padgham, Goals in the context of BDI plan failure and planning, in: E. H. Durfee, M. Yokoo, M. N. Huhns, O. Shehory (Eds.), Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), ACM, 2007, pp. 1–8.

[16] G. Plotkin, A structural approach to operational semantics, Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, Denmark (1981).

[17] P. Blackburn, M. de Rijke, Y. Venema, Modal Logic, Vol. 53 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2001.

[18] M. J. Fischer, R. E. Ladner, Propositional dynamic logic of regular programs, Journal of Computer and System Sciences 18 (2) (1979) 194–211.

[19] U. Hustadt, R. A. Schmidt, MSPASS: Modal reasoning by translation and first-order resolution., in: Proc. TABLEAUX 2000, Vol. 1847 of LNCS, Springer, 2000, pp. 67–71.

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

[21] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. B. Scherl, GOLOG: A logic programming language for dynamic domains, Journal of Logic Programming 31 (1-3) (1997) 59–83.

[22] J. Claßen, G. Lakemeyer, A logic for non-terminating Golog programs, in: G. Brewka, J. Lang (Eds.), Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08), AAAI, Sydney, Australia, 2008, pp. 589–599.

[23] B. van Riemsdijk, W. van der Hoek, J.-J. C. Meyer, Agent programming in Dribble: from beliefs to goals using plans, in: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), ACM Press, New York, NY, USA, 2003, pp. 393–400. doi:http://doi.acm.org/10.1145/860575.860639.

[24] D. T. Trang, B. Logan, N. Alechina, Verifying Dribble agents, in: M. Baldoni, J. Bentahar, J. Lloyd, M. B. van Riemsdijk (Eds.), Seventh International Workshop on Declarative Agent Languages and Technologies (DALT 2009), Workshop Notes, Budapest Hungary, 2009, pp. 162–177.