Dynamic Logic for Plan Revision in Intelligent Agents

M. Birna van Riemsdijk[1] Frank S. de Boer[1,2,3] John-Jules Ch. Meyer[1]

[1] ICS, Utrecht University, The Netherlands
[2] CWI, Amsterdam, The Netherlands
[3] LIACS, Leiden University, The Netherlands

Abstract. In this paper, we present a dynamic logic for a propositional version of the agent programming language 3APL. A 3APL agent has beliefs and a plan. The execution of a plan changes an agent's beliefs. Plans can be revised during execution. Due to these plan revision capabilities of 3APL agents, plans cannot be analyzed by structural induction as in for example standard propositional dynamic logic. We propose a dynamic logic that is tailored to handle the plan revision aspect of 3APL. For this logic, we give a sound and complete axiomatization.

1 Introduction

An agent is commonly seen as an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [24]. Programming these flexible computing entities is not a trivial task. An important line of research in this area, is research on *cognitive* agents. These are agents endowed with high-level mental attitudes such as beliefs, desires, goals, plans, intentions, norms and obligations. Intelligent cognitive agents should be able to reason with these mental attitudes in order to exhibit the desired flexible problem solving behavior.

The very concept of (cognitive) agents is thus a complex one. It is imperative that programmed agents be amenable to precise and formal specification and verification, at least for some critical applications. This is recognized by (potential) appliers of agent technology such as NASA, which organizes specialized workshops on the subject of formal specification and verification of agents [17,11].

In this paper, we are concerned with the verification of agents programmed in (a simplified version of) the cognitive agent programming language *3APL*[4] [12,23,4]. This language is based on theoretical research on cognitive notions [2,3,16,19]. In the latest version [4], a 3APL agent has a set of beliefs, a plan and a set of goals. The idea is, that an agent tries to fulfill its goals by selecting appropriate plans, depending on its beliefs about the world. Beliefs should thus represent the world or environment of the agent; the goals represent the state of

[4] 3APL is to be pronounced as "triple-a-p-l".

the world the agent wants to realize and plans are the means to achieve these goals.

As explained, cognitive agent programming languages are designed to program flexible behavior using high-level mental attitudes. In the various languages, these attitudes are handled in different ways. An important aspect of 3APL is the way in which plans are dealt with. A plan in 3APL can be executed, resulting in a change of the beliefs of the agent[5]. Now, in order to increase the possible flexibility of agents, 3APL [12] was endowed with a mechanism with which the programmer can program agents that can *revise* their plans during execution of the agent. This is a distinguishing feature of 3APL compared to other agent programming languages and architectures [15,18,7,6]. The idea is, that an agent should not blindly execute an adopted plan, but it should be able to revise it under certain conditions. As this paper focusses on the plan revision aspect of 3APL, we consider a version of the language with only beliefs and plans, i.e., without goals. We will use a propositional and otherwise slightly simplified variant of the original 3APL language as defined in [12].

In 3APL, the plan revision capabilities can be programmed through plan revision rules. These rules consist of a head and a body, both representing a plan. A plan is basically a sequence of so-called basic actions. These actions can be executed. The idea is, informally, that an agent can apply a rule if it has a plan corresponding to the head of this rule, resulting in the replacement of this plan by the plan in the body of the rule. The introduction of these capabilities now gives rise to interesting issues concerning the characteristics of plan execution, as will become clear in the sequel. This has implications for reasoning about the result of plan execution and therefore for the formal verification of 3APL agents, which we are concerned with in this paper.

To be more specific, after defining (a simplified version of) 3APL and its semantics (section 2), we propose a dynamic logic for proving properties of 3APL plans in the context of plan revision rules (section 3). For this logic, we provide a sound and complete axiomatization (section 4).

As for related work, verification of agents programmed in an agent programming language has for example been addressed in [1]. This paper addresses model checking of the agent programming language AgentSpeak. A sketch of a dynamic logic to reason about 3APL agents has been given in [23]. This logic however is designed to reason about a 3APL interpreter or deliberation language, whereas in this paper we take a different viewpoint and reason about plans. In [13], a programming logic (without axiomatization) was given for a fragment of 3APL without plan revision rules. Further, the operational semantics of plan revision rules is similar to that of procedures in procedural programming. In fact, plan revision rules can be viewed as an extension of procedures. Logics and semantics for procedural languages are for example studied in De Bakker [5]. Although the operational semantics of procedures and plan revision rules are similar, techniques for reasoning about procedures cannot be used for plan revision rules. This is due to the fact that the introduction of these rules results in the seman-

[5] A change in the environment is a possible "side effect" of the execution of a plan.

tics of the sequential composition operator no longer being compositional (see section 3). This issue has also been considered from a semantic perspective in [22,21]. In [10], a framework for planning in dynamic environments is presented in a logic programming setting. The approach is based on hierarchical task network planning. The motivation for that work is similar to the motivation for the introduction of plan revision rules.

To the best of our knowledge, this is the first attempt to design a logic and deductive system for plan revision rules or similar language constructs. Considering the semantic difficulties that arise with the introduction of this type of construct, it is not a priori obvious that it would be possible at all to design a deductive system to reason about these constructs. The main aim of this work was thus to investigate whether it is possible to define such a system and in this way also to get a better theoretical understanding of the construct of plan revision rules. Whether the system presented in this paper is also practically useful to verify 3APL agents, remains to be seen and will be subject to further research.

2 3APL

2.1 Syntax

Below, we define belief bases and plans. A belief base is a set of propositional formulas. A plan is a sequence of basic actions and abstract plans. Basic actions can be executed, resulting in a change to the beliefs of the agent. An abstract plan can, in contrast with basic actions, not be executed directly in the sense that it updates the belief base of an agent. Abstract plans serve as an abstraction mechanism like procedures in procedural programming. If a plan consists of an abstract plan, this abstract plan could be transformed into basic actions through the application of plan revision rules, which will be introduced below[6].

In the sequel, a language defined by inclusion shall be the smallest language containing the specified elements.

Definition 1. *(belief bases)* Assume a propositional language $\mathcal{L}$ with typical formula q and the connectives $\wedge$ and $\neg$ with the usual meaning. Then the set of belief bases Σ with typical element σ is defined to be $\wp(\mathcal{L})$.[7]

Definition 2. *(plans)* Assume that a set $\mathsf{BasicAction}$ with typical element a is given, together with a set $\mathsf{AbstractPlan}$ with typical element p. Then the set of plans Π with typical element π is defined as follows:

- $\mathsf{BasicAction} \cup \mathsf{AbstractPlan} \subseteq \Pi$,
- if $c \in (\mathsf{BasicAction} \cup \mathsf{AbstractPlan})$ and $\pi \in \Pi$ then $c\,;\pi \in \Pi$.

[6] Abstract plans could also be modelled as non-executable basic actions.
[7] $\wp(\mathcal{L})$ denotes the powerset of $\mathcal{L}$.

Basic actions and abstract plans are called atomic plans and are typically denoted by c. For technical convenience, plans are defined to have a list structure, which means strictly speaking, that we can only use the sequential composition operator to concatenate an atomic plan and a plan, rather than concatenating two arbitrary plans. In the following, we will however also use the sequential composition operator to concatenate arbitrary plans π_1 and π_2 yielding $\pi_1; \pi_2$. The operator should in this case be read as a function taking two plans that have a list structure and yielding a new plan that also has this structure. The plan π_1 will thus be the prefix of the resulting plan.

We use ϵ to denote the empty plan, which is an empty list. The concatenation of a plan π and the empty list is equal to π, i.e., $\epsilon; \pi$ and $\pi; \epsilon$ are identified with π.

A plan and a belief base can together constitute a so-called configuration. During computation or execution of the agent, the elements in a configuration can change.

Definition 3. *(configuration)* Let Σ be the set of belief bases and let Π be the set of plans. Then $\Pi \times \Sigma$ is the set of configurations of a 3APL agent.

Plan revision rules consist of a head π_h and a body π_b. Informally, an agent that has a plan π_h, can replace this plan by π_b when applying a plan revision rule of this form.

Definition 4. *(plan revision (PR) rules)* The set of PR rules $\mathcal{R}$ is defined as follows: $\mathcal{R} = \{\pi_h \rightsquigarrow \pi_b \mid \pi_h, \pi_b \in \Pi, \pi_h \neq \epsilon\}$.[8]

Take for example a plan $a; b$ where a and b are basic actions, and a PR rule $a; b \rightsquigarrow c$. The agent can then either execute the actions a and b one after the other, or it can apply the PR rule yielding a new plan c, which can in turn be executed. A plan p consisting of an abstract plan cannot be executed, but can only be transformed using a procedure-like PR rule such as $p \rightsquigarrow a$.

Below, we provide the definition of a 3APL agent. The function $\mathcal{T}$, taking a basic action and a belief base and yielding a new belief base, is used to define how belief bases are updated when a basic action is executed.

Definition 5. *(3APL agent)* A 3APL agent $\mathcal{A}$ is a tuple $\langle \mathsf{Rule}, \mathcal{T} \rangle$ where $\mathsf{Rule} \subseteq \mathcal{R}$ is a finite set of PR rules and $\mathcal{T} : (\mathsf{BasicAction} \times \sigma) \rightarrow \sigma$ is a partial function, expressing how belief bases are updated through basic action execution.

2.2 Semantics

The semantics of a programming language can be defined as a function taking a statement and a state, and yielding the set of states resulting from executing the

[8] In [12], PR rules were defined to have a guard, i.e., rules were of the form $\pi_h \mid \phi \rightsquigarrow \pi_b$. For a rule to be applicable, the guard should then hold. For technical convenience and because we want to focus on the plan revision aspect of these rules, we however leave out the guard in this paper. The results could be extended for rules with a guard.

initial statement in the initial state. In this way, a statement can be viewed as a transformation function on states. In 3APL, plans can be seen as statements and belief bases as states on which these plans operate. There are various ways of defining a semantic function and in this paper we are concerned with the so-called *operational* semantics (see for example De Bakker [5] for details on this subject).

The operational semantics of a language is usually defined using transition systems [14]. A transition system for a programming language consists of a set of axioms and derivation rules for deriving transitions for this language. A transition is a transformation of one configuration into another and it corresponds to a single computation step. Let $\mathcal{A} = \langle \mathsf{Rule}, \mathcal{T} \rangle$ be a 3APL agent and let $\mathsf{BasicAction}$ be a set of basic actions. Below, we give the transition system $\mathsf{Trans}_{\mathcal{A}}$ for our simplified 3APL language, which is based on the system given in [12]. This transition system is specific to agent $\mathcal{A}$.

There are two kinds of transitions, i.e., transitions describing the execution of basic actions and those describing the application of a plan revision rule. The transitions are labelled to denote the kind of transition. A basic action at the head of a plan can be executed in a configuration if the function $\mathcal{T}$ is defined for this action and the belief base in the configuration. The execution results in a change of belief base as specified through $\mathcal{T}$ and the action is removed from the plan.

Definition 6. *(action execution)* Let $a \in \mathsf{BasicAction}$.

$$\frac{\mathcal{T}(a, \sigma) = \sigma'}{\langle a; \pi, \sigma \rangle \rightarrow_{exec} \langle \pi, \sigma' \rangle}$$

A plan revision rule can be applied in a configuration if the head of the rule is equal to a prefix of the plan in the configuration. The application of the rule results in the revision of the plan, such that the prefix equal to the head of the rule is replaced by the plan in the body of the rule. A rule $a; b \rightsquigarrow c$ can for example be applied to the plan $a; b; c$, yielding the plan $c; c$. The belief base is not changed through plan revision.

Definition 7. *(rule application)* Let $\rho : \pi_h \rightsquigarrow \pi_b \in \mathsf{Rule}$.

$$\langle \pi_h; \pi, \sigma \rangle \rightarrow_{app} \langle \pi_b; \pi, \sigma \rangle$$

In the sequel, it will be useful to have a function taking a PR rule and a plan, and yielding the plan resulting from the application of the rule to this given plan. Based on this function, we also define a function taking a set of PR rules and a plan and yielding the set of rules applicable to this plan.

Definition 8. *(rule application)* Let $\mathcal{R}$ be the set of PR rules and let Π be the set of plans. Let $\rho : \pi_h \rightsquigarrow \pi_b \in \mathcal{R}$ and $\pi, \pi' \in \Pi$. The partial function $apply : (\mathcal{R} \times \Pi) \rightarrow \Pi$ is then defined as follows.

$$apply(\rho)(\pi) = \begin{cases} \pi_b; \pi' & \text{if } \pi = \pi_h; \pi', \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The function *applicable* : $(\wp(\mathcal{R}) \times \Pi) \rightarrow \wp(\mathcal{R})$ yielding the set of rules applicable to a certain plan, is then as follows: $applicable(\mathsf{Rule}, \pi) = \{\rho \in \mathsf{Rule} \mid apply(\rho)(\pi) \text{ is defined}\}$.

Using the transition system, individual transitions can be derived for a 3APL agent. These transitions can be put in sequel, yielding transition sequences. From a transition sequence, one can obtain a *computation sequence* by removing the plan component of all configurations occurring in the transition sequence. In the following definitions, we formally define computation sequences and we specify the function yielding these sequences, given an initial configuration.

Definition 9. *(computation sequences)* The set Σ^+ of finite computation sequences is defined as $\{\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n \mid \sigma_i \in \Sigma, 1 \leq i \leq n, n \in \mathbb{N}\}$.

Definition 10. *(function for calculating computation sequences)* Let $x_i \in \{exec, app\}$ for $1 \leq i \leq m$. The function $\mathcal{C}^{\mathcal{A}} : (\Pi \times \Sigma) \rightarrow \wp(\Sigma^+)$ is then as defined below.

$$\mathcal{C}^{\mathcal{A}}(\pi, \sigma) = \{\sigma, \ldots, \sigma_m \in \Sigma^+ \mid \theta = \langle \pi, \sigma \rangle \rightarrow_{x_1} \ldots \rightarrow_{x_m} \langle \epsilon, \sigma_m \rangle \text{ is a finite sequence of transitions in } \mathsf{Trans}_{\mathcal{A}}\}.$$

Note that we only take into account successfully terminating transition sequences, i.e., those sequences ending in a configuration with an empty plan. Using the function defined above, we can now define the operational semantics of 3APL.

Definition 11. *(operational semantics)* Let $\kappa : \wp(\Sigma^+) \rightarrow \wp(\Sigma)$ be a function yielding the last elements of a set of finite computation sequences, which is defined as follows: $\kappa(\Delta) = \{\sigma_n \mid \sigma_1, \ldots, \sigma_n \in \Delta\}$. The operational semantic function $\mathcal{O}^{\mathcal{A}} : \Pi \rightarrow (\Sigma \rightarrow \wp(\Sigma))$ is defined as follows:

$$\mathcal{O}^{\mathcal{A}}(\pi)(\sigma) = \kappa(\mathcal{C}^{\mathcal{A}}(\pi, \sigma)).$$

We will sometimes omit the superscript $\mathcal{A}$ from functions as defined above, for reasons of presentation. The example below is used to explain the definition of the operational semantics.

Example 1. Let $\mathcal{A}$ be an agent with PR rules $\{p; a \rightsquigarrow b, p \rightsquigarrow c\}$, where p is an abstract plan and a, b, c are basic actions. Let σ_a be the belief base resulting from the execution of a in σ, i.e., $\mathcal{T}(a, \sigma) = \sigma_a$, let be σ_{ab} the belief resulting from executing first a and then b in σ, etc.

Then $\mathcal{C}^{\mathcal{A}}(p; a)(\sigma) = \{(\sigma, \sigma, \sigma_b), (\sigma, \sigma, \sigma_c, \sigma_{ca})\}$, which is based on the transition sequences $\langle p; a, \sigma \rangle \rightarrow_{app} \langle b, \sigma \rangle \rightarrow_{exec} \langle \epsilon, \sigma_b \rangle$ and $\langle p; a, \sigma \rangle \rightarrow_{app} \langle c; a, \sigma \rangle \rightarrow_{exec} \langle a, \sigma_c \rangle \rightarrow_{exec} \langle \epsilon, \sigma_{ca} \rangle$. We thus have that $\mathcal{O}^{\mathcal{A}}(p; a)(\sigma) = \{\sigma_b, \sigma_{ca}\}$.

3 Dynamic Logic

In programming language research, an important area is the specification and verification of programs. Program logics are designed to facilitate this process. One such logic is dynamic logic [8,9], with which we are concerned in this paper. In dynamic logic, programs are explicit syntactic constructs in the logic. To be able to discuss the effect of the execution of a program π on the truth of a formula ϕ, the modal construct $[\pi]\phi$ is used. This construct intuitively states that in all states in which π halts, the formula ϕ holds.

Programs in general are constructed from atomic programs and composition operators. An example of a composition operator is the sequential composition operator (;), where the program $\pi_1; \pi_2$ intuitively means that π_1 is executed first, followed by the execution of π_2. The semantics of such a compound program can in general be determined by the semantics of the parts of which it is composed. This compositionality property allows analysis by structural induction (see also [20]), i.e., analysis of a compound statement by analysis of its parts. Analysis of the sequential composition operator by structural induction can in dynamic logic be expressed by the following formula, which is usually a validity: $[\pi_1; \pi_2]\phi \leftrightarrow [\pi_1][\pi_2]\phi$. For 3APL plans on the contrary, this formula does not always hold. This is due to the presence of PR rules.

We will informally explain this using the 3APL agent of example 1. As explained, the operational semantics of this agent, given initial plan $p; a$ and initial state σ, is as follows: $\mathcal{O}(p; a)(\sigma) = \{\sigma_b, \sigma_{ca}\}$. Now compare the result of first "executing"[9] p in σ and then executing a in the resulting belief base, i.e., compare the set $\mathcal{O}(a)(\mathcal{O}(p)(\sigma))$. In this case, there is only one successfully terminating transition sequence and it ends in σ_{ca}, i.e., $\mathcal{O}(a)(\mathcal{O}(p)(\sigma)) = \{\sigma_{ca}\}$. Now, if it would be the case that $\sigma_{ca} \models \phi$ but $\sigma_b \not\models \phi$, the formula $[p; a]\phi \leftrightarrow [p][a]\phi$ would not hold[10].

Analysis of plans by structural induction in this way thus does not work for 3APL. In order to be able to prove correctness properties of 3APL programs however, one can perhaps imagine that it is important to have *some* kind of induction. As we will show in the sequel, the kind of induction that can be used to reason about 3APL programs, is induction on the *number of PR rule applications in a transition sequence*. We will introduce a dynamic logic for 3APL based on this idea.

3.1 Syntax

In order to be able to do induction on the number of PR rule applications in a transition sequence, we introduce so-called *restricted plans*. These are plans,

[9] We will use the word "execution" in two ways. Firstly, as in this context, we will use it to denote the execution of an arbitrary plan in the sense of going through several transition of type *exec* or *app*, starting in a configuration with this plan and resulting in some final configurations. Secondly, we will use it to refer to the execution of a basic action in the sense of going through a transition of type *exec*.

[10] In particular, the implication would not hold from right to left.

annotated with a natural number[11]. Informally, if the restriction parameter of a plan is n, the number of rule applications during execution of this plan cannot exceed n.

Definition 12. *(restricted plans)* Let Π be the language of plans and let $\mathbb{N}^- = \mathbb{N} \cup \{-1\}$. Then, the language Π_r of restricted plans is defined as $\{\pi\restriction_n \mid \pi \in \Pi, n \in \mathbb{N}^-\}$.

Below, we define the language of dynamic logic in which properties of 3APL agents can be expressed. In the logic, one can express properties of restricted plans. As will become clear in the sequel, one can prove properties of the plan of a 3APL agent by proving properties of restricted plans.

Definition 13. *(plan revision dynamic logic (*PRDL*))* Let $\pi\restriction_n \in \Pi_r$ be a restricted plan. Then the language of dynamic logic $\mathcal{L}_{\mathsf{PRDL}}$ with typical element ϕ is defined as follows:

- $\mathcal{L} \subseteq \mathcal{L}_{\mathsf{PRDL}}$,
- if $\phi \in \mathcal{L}_{\mathsf{PRDL}}$, then $[\pi\restriction_n]\phi \in \mathcal{L}_{\mathsf{PRDL}}$,
- if $\phi, \phi' \in \mathcal{L}_{\mathsf{PRDL}}$, then $\neg\phi \in \mathcal{L}_{\mathsf{PRDL}}$ and $\phi \wedge \phi' \in \mathcal{L}_{\mathsf{PRDL}}$.

3.2 Semantics

In order to define the semantics of PRDL, we first define the semantics of restricted plans. As for ordinary plans, we also define an operational semantics for restricted plans. We do this by defining a function for calculating computation sequences, given an initial restricted plan and a belief base.

Definition 14. *(function for calculating computation sequences)* Let $x_i \in \{exec, app\}$ for $1 \leq i \leq m$. Let $N_{app}(\theta)$ be a function yielding the number of transitions of the form $s_i \rightarrow_{app} s_{i+1}$ in the sequence of transitions θ. The function $\mathcal{C}_r^{\mathcal{A}} : (\Pi_r \times \Sigma) \rightarrow \wp(\Sigma^+)$ is then as defined below.

$$\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma) = \{\sigma, \ldots, \sigma_m \in \Sigma^+ \mid \theta = \langle \pi, \sigma\rangle \rightarrow_{x_1} \ldots \rightarrow_{x_m} \langle \epsilon, \sigma_m\rangle \text{ is a finite sequence of transitions in } \mathsf{Trans}_{\mathcal{A}} \text{ where } 0 \leq N_{app}(\theta) \leq n\}$$

As one can see in the definition above, the computation sequences $\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma)$ are based on transition sequences starting in configuration $\langle \pi, \sigma\rangle$. The number of rule applications in these transition sequences should be between 0 and n, in contrast with the function $\mathcal{C}^{\mathcal{A}}$ of definition 10, in which there is no restriction on this number.

Based on the function $\mathcal{C}_r^{\mathcal{A}}$, we define the operational semantics of restricted plans by taking the last elements of the computation sequences yielded by $\mathcal{C}_r^{\mathcal{A}}$. The set of belief bases is empty if the restriction parameter is equal to -1.

[11] Or with the number -1. The number -1 is introduced for technical convenience and it will become clear in the sequel why we need this.

Definition 15. *(operational semantics)* Let κ be as in definition 11. The operational semantic function $\mathcal{O}_r^{\mathcal{A}} : \Pi_r \to (\Sigma \to \wp(\Sigma))$ is defined as follows:

$$\mathcal{O}_r^{\mathcal{A}}(\pi\restriction_n)(\sigma) = \begin{cases} \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma)) & \text{if } n \geq 0, \\ \emptyset & \text{if } n = -1. \end{cases}$$

In the following proposition, we relate the operational semantics of plans and the operational semantics of restricted plans.

Proposition 1.

$$\bigcup_{n \in \mathbb{N}} \mathcal{O}_r(\pi\restriction_n)(\sigma) = \mathcal{O}(\pi)(\sigma)$$

Proof. Immediate from definitions 15, 14, 11 and 10.

Using the operational semantics of restricted plans, we can now define the semantics of the dynamic logic.

Definition 16. *(semantics of* PRDL*)* Let $q \in \mathcal{L}$ be a propositional formula, let $\phi, \phi' \in \mathcal{L}_{\mathsf{PRDL}}$ and let $\models_{\mathcal{L}}$ be the entailment relation defined for $\mathcal{L}$ as usual. The semantics $\models_{\mathcal{A}}$ of $\mathcal{L}_{\mathsf{PRDL}}$ is then as defined below.

$$\begin{aligned} &\sigma \models_{\mathcal{A}} q && \Leftrightarrow \sigma \models_{\mathcal{L}} q \\ &\sigma \models_{\mathcal{A}} [\pi\restriction_n]\phi && \Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(\pi\restriction_n)(\sigma) : \sigma' \models_{\mathcal{A}} \phi \\ &\sigma \models_{\mathcal{A}} \neg\phi && \Leftrightarrow \sigma \not\models_{\mathcal{A}} \phi \\ &\sigma \models_{\mathcal{A}} \phi \wedge \phi' && \Leftrightarrow \sigma \models_{\mathcal{A}} \phi \text{ and } \sigma \models_{\mathcal{A}} \phi' \end{aligned}$$

As $\mathcal{O}_r^{\mathcal{A}}$ is defined in terms of agent $\mathcal{A}$, so is the semantics of $\mathcal{L}_{\mathsf{PRDL}}$. We use the subscript $\mathcal{A}$ to indicate this. Let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. If $\forall \mathcal{T}, \sigma : \sigma \models_{\langle \mathsf{Rule}, \mathcal{T} \rangle} \phi$, we write $\models_{\mathsf{Rule}} \phi$.

In the dynamic logic PRDL, one can express properties of restricted plans, rather than of ordinary 3APL plans. The operational semantics of ordinary plans $\mathcal{O}$ and of restricted plans $\mathcal{O}_r$ are however related (proposition 1). As the semantics of the construct $[\pi\restriction_n]\sigma$ is defined in terms of $\mathcal{O}_r$, we can use this construct to specify properties of 3APL plans, as shown by the following corollary.

Corollary 1.

$$\forall n \in \mathbb{N} : \sigma \models_{\mathcal{A}} [\pi\restriction_n]\phi \Leftrightarrow \forall \sigma' \in \mathcal{O}^{\mathcal{A}}(\pi)(\sigma) : \sigma' \models_{\mathcal{A}} \phi$$

Proof. Immediate from proposition 1 and definition 16.

4 The Axiom System

In order to prove properties of restricted plans, we propose a deductive system for PRDL in this section. Rather than proving properties of restricted plans, the aim is however to prove properties of 3APL plans. We thus want to prove properties of the form $\forall n \in \mathbb{N} : [\pi\restriction_n]\phi$, as these are directly related to 3APL by corollary 1. The idea now is, that these properties can be proven by induction on n. We will explain this in more detail after introducing the axiom system for restricted plans.

Definition 17. *(axiom system ($\mathsf{AS}_{\mathsf{Rule}}$))* Let BasicAction be a set of basic actions, AbstractPlan be a set of abstract plans and Rule $\subseteq \mathcal{R}$ be a finite set of PR rules. Let $a \in$ BasicAction, let $p \in$ AbstractPlan, let $c \in$ (BasicAction $\cup$ AbstractPlan) and let ρ range over $applicable(\mathsf{Rule}, c; \pi)$. The following are then the axioms of the system $\mathsf{AS}_{\mathsf{Rule}}$.

(PRDL1)	$[\pi\restriction_{-1}]\phi$	
(PRDL2)	$[p\restriction_0]\phi$	
(PRDL3)	$[\epsilon\restriction_n]\phi \leftrightarrow \phi$	if $0 \leq n$
(PRDL4)	$[c;\pi\restriction_n]\phi \leftrightarrow [c\restriction_0][\pi\restriction_n]\phi \wedge \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi$	if $0 \leq n$
(PL)	axioms for propositional logic	
(PDL)	$[\pi\restriction_n](\phi \rightarrow \phi') \rightarrow ([\pi\restriction_n]\phi \rightarrow [\pi\restriction_n]\phi')$	

The following are the rules of the system $\mathsf{AS}_{\mathsf{Rule}}$.

(GEN)
$$\frac{\phi}{[\pi\restriction_n]\phi}$$

(MP)
$$\frac{\phi_1,\ \phi_1 \rightarrow \phi_2}{\phi_2}$$

As the axiom system is relative to a given set of PR rules Rule, we will use the notation $\vdash_{\mathsf{Rule}} \phi$ to specify that ϕ is derivable in the system $\mathsf{AS}_{\mathsf{Rule}}$ above.

The idea is that properties of the form $\forall n \in \mathbb{N} : \vdash_{\mathsf{Rule}} [\pi\restriction_n]\phi$ can be proven by induction on n as follows. If we can prove $[\pi\restriction_0]\phi$ and $\forall n \in \mathbb{N} : ([\pi\restriction_n]\phi \vdash_{\mathsf{Rule}} [\pi\restriction_{n+1}]\phi)$, we can conclude the desired property. These premises should be proven using the axiom system above. Consider for example an agent with a PR rule $a \rightsquigarrow a; a$ and assume that $\mathcal{T}$ is defined such that $[a\restriction_0]\phi$. One can then prove $\forall n : [a\restriction_n]\phi$ by proving $[a\restriction_n]\phi \vdash_{\mathsf{Rule}} [a\restriction_{n+1}]\phi$, for arbitrary n.

We will now explain the PRDL axioms of the system. The other axioms and the rules are standard for propositional dynamic logic (PDL) [8]. We start by explaining the most interesting axiom: (PRDL4). We first observe that there are two types of transitions that can be derived for a 3APL agent: action execution and rule application (see definitions 6 and 7). Consider a configuration $\langle a; \pi, \sigma\rangle$ where a is a basic action. Then during computation, possible next configurations are $\langle \pi, \sigma'\rangle$[12] (action execution) and $\langle apply(\rho, a; \pi), \sigma\rangle$ (rule application) where ρ ranges over the applicable rules, i.e., $applicable(\mathsf{Rule}, a; \pi)$[13]. We can thus analyze the plan $a; \pi$ by analyzing π after the execution of a, and the plans resulting from applying a rule, i.e., $apply(\rho, a; \pi)$[14]. The execution of an action can be represented by the number 0 as restriction parameter, yielding the first term of

[12] assuming that $\mathcal{T}(a, \sigma) = \sigma'$

[13] See definition 8 for the definitions of the functions *apply* and *applicable*.

[14] Note that one could say we analyze a plan $a; \pi$ partly by structural induction, as it is partly analyzed in terms of a and π.

the right-hand side of (PRDL4): $[a\restriction_0][\pi\restriction_n]\phi$[15]. The second term is a conjunction of $[apply(\rho, c; \pi)\restriction_{n-1}]\phi$ over all applicable rules ρ. The restriction parameter is $n-1$ as we have "used" one of our n permitted rule applications. The first three axioms represent basic properties of restricted plans. (PRDL1) can be used to eliminate the second term on the right-hand side of axiom (PRDL4), if the left-hand side is $[c; \pi\restriction_0]\phi$. (PRDL2) can be used to eliminate the first term on the right-hand side of (PRDL4), if c is an abstract plan. As abstract plans can only be transformed through rule application, there will be no resulting states if the restriction parameter of the abstract plan is 0, i.e., if no rule applications are allowed. (PRDL3) states that if ϕ is to hold after execution of the empty plan, it should hold "now". It can be used to derive properties of an atomic plan c, by using axiom (PRDL4) with the plan $c; \epsilon$.

Example 2. Let $\mathcal{A}$ be an agent with one PR rule, i.e., $\mathsf{Rule} = \{a; b \rightsquigarrow c\}$ and let $\mathcal{T}$ be such that $[a\restriction_0]\phi$, $[b\restriction_0]\phi$ and $[c\restriction_0]\phi$. We now want to prove that $\forall n : [a; b\restriction_n]\phi$. We have $[a; b\restriction_0]\phi$ by using that this is equivalent to $[a\restriction_0][b\restriction_0]\phi$ by proposition 3 (section 4.1). The latter formula can be derived by applying (GEN) to $[b\restriction_0]\phi$. We prove $\forall n \in \mathbb{N} : ([a; b\restriction_n]\phi \vdash_{\mathsf{Rule}} [a; b\restriction_{n+1}]\phi)$ by taking an arbitrary n and proving that $[a; b\restriction_n]\phi \vdash_{\mathsf{Rule}} [a; b\restriction_{n+1}]\phi$. Using (PRDL4) and (PRDL3), we have the following equivalences. In order to apply (PRDL4) to the conjunct $[c\restriction_{n-1}]\phi$, n has to be greater than 0. This is however not a problem, as the result was proven separately for $n = 0$.

$$\begin{aligned}
[a; b\restriction_n]\phi &\leftrightarrow [a\restriction_0][b\restriction_n]\phi \quad\;\; \wedge [c\restriction_{n-1}]\phi \\
&\leftrightarrow [a\restriction_0][b\restriction_0][\epsilon\restriction_n]\phi \wedge [c\restriction_0][\epsilon\restriction_{n-1}]\phi \\
&\leftrightarrow [a\restriction_0][b\restriction_0]\phi \quad\;\; \wedge [c\restriction_0]\phi
\end{aligned}$$

Similarly, we have the following equivalences for $[a; b\restriction_{n+1}]\phi$, yielding the desired result.

$$\begin{aligned}
[a; b\restriction_{n+1}]\phi &\leftrightarrow [a\restriction_0][b\restriction_{n+1}]\phi \quad\;\; \wedge [c\restriction_n]\phi \\
&\leftrightarrow [a\restriction_0][b\restriction_0][\epsilon\restriction_{n+1}]\phi \wedge [c\restriction_0][\epsilon\restriction_n]\phi \\
&\leftrightarrow [a\restriction_0][b\restriction_0]\phi \quad\;\; \wedge [c\restriction_0]\phi
\end{aligned}$$

4.1 Soundness and Completeness

The axiom system of definition 17 is sound.

Theorem 1. *(soundness)* Let $\phi \in \mathcal{L}_{\mathsf{PRDL}}$. Let $\mathsf{Rule} \subseteq \mathcal{R}$ be an arbitrary finite set of PR rules. Then the axiom system $\mathsf{AS}_{\mathsf{Rule}}$ is sound, i.e.:

$$\vdash_{\mathsf{Rule}} \phi \;\Rightarrow \models_{\mathsf{Rule}} \phi.$$

Proof. We prove soundness of the PRDL axioms of the system $\mathsf{AS}_{\mathsf{Rule}}$.
(PRDL1) The proof is through observing that $\mathcal{O}_r(\pi\restriction_{-1})(\sigma) = \emptyset$ by definition 15.
(PRDL2) The proof is analogous to the proof of axiom (PRDL1), with p for π

[15] In our explanation, we consider the case where c is a basic action, but the axiom holds also for abstract plans.

and 0 for -1 and using definition 6 to derive that $\mathcal{O}_r^{\mathcal{A}}(p\restriction_0)(\sigma) = \emptyset$.
(PRDL3) The proof is through observing that $\kappa(\mathcal{C}_r(\epsilon\restriction_n, \sigma)) = \{\sigma\}$ by definition 14.
(PRDL4) Let $\pi \in \Pi$ be an arbitrary plan and $\phi \in \mathcal{L}_{\mathsf{PRDL}}$ be an arbitrary PRDL formula. To prove: $\forall \mathcal{T}, \sigma : \sigma \models_{\langle \mathsf{Rule}, \mathcal{T} \rangle} [c; \pi\restriction_n]\phi \leftrightarrow [c\restriction_0][\pi\restriction_n]\phi \wedge \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi$, i.e.:

$$\forall \mathcal{T}, \sigma : \sigma \models_{\langle \mathsf{Rule}, \mathcal{T} \rangle} [c; \pi\restriction_n]\phi \Leftrightarrow \forall \mathcal{T}, \sigma : \sigma \models_{\langle \mathsf{Rule}, \mathcal{T} \rangle} [c\restriction_0][\pi\restriction_n]\phi \text{ and } \forall \mathcal{T}, \sigma : \sigma \models_{\langle \mathsf{Rule}, \mathcal{T} \rangle} \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi.$$

Let $\sigma \in \Sigma$ be an arbitrary belief base and let $\mathcal{T}$ be an arbitrary belief update function. Assume $c \in \mathsf{BasicAction}$ and furthermore assume that $\langle c; \pi, \sigma\rangle \rightarrow_{execute} \langle \pi, \sigma_1\rangle$ is a transition in $\mathsf{Trans}_{\mathcal{A}}$, i.e., $\kappa(\mathcal{C}_r^{\mathcal{A}}(c\restriction_0, \sigma)) = \{\sigma_1\}$ by definition 14. Let ρ range over $applicable(\mathsf{Rule}, c; \pi)$. Now, observe the following by definition 14:

$$\kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma)) = \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma_1)) \cup \bigcup_\rho \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma)). \quad (1)$$

If $c \in \mathsf{AbstractPlan}$ or if a transition of the form $\langle c; \pi, \sigma\rangle \rightarrow_{execute} \langle \pi, \sigma_1\rangle$ is not derivable, the first term of the right-hand side of (1) is empty.

($\Rightarrow$) Assume $\sigma \models_{\mathsf{Rule}} [c; \pi\restriction_n]\phi$, i.e., by definition 16 $\forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma) : \sigma' \models_{\mathsf{Rule}} \phi$, i.e., by definition 15:

$$\forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma)) : \sigma' \models_{\mathsf{Rule}} \phi. \quad (2)$$

To prove: (A) $\sigma \models_{\mathsf{Rule}} [c\restriction_0][\pi\restriction_n]\phi$ and (B) $\sigma \models_{\mathsf{Rule}} \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi$.
(A) If $c \in \mathsf{AbstractPlan}$ or if a transition of the form $\langle c; \pi, \sigma\rangle \rightarrow_{execute} \langle \pi, \sigma_1\rangle$ is not derivable, the desired result follows immediately from axiom (PRDL2) or an analogous proposition for non executable basic actions. If $c \in \mathsf{BasicAction}$, we have the following from definitions 16 and 15.

$$\begin{aligned} \sigma \models_{\mathsf{Rule}} [c\restriction_0][\pi\restriction_n]\phi &\Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(c\restriction_0, \sigma) : \sigma' \models_{\mathsf{Rule}} [\pi\restriction_n]\phi \\ &\Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(c\restriction_0, \sigma) : \forall \sigma'' \in \mathcal{O}_r^{\mathcal{A}}(\pi\restriction_n, \sigma') : \sigma'' \models_{\mathsf{Rule}} \phi \\ &\Leftrightarrow \forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c\restriction_0, \sigma)) : \forall \sigma'' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma')) : \sigma'' \models_{\mathsf{Rule}} \phi \\ &\Leftrightarrow \forall \sigma'' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma_1)) : \sigma'' \models_{\mathsf{Rule}} \phi \end{aligned} \quad (3)$$

From 1, we have that $\kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma_1)) \subseteq \kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma))$. From this and assumption (2), we can now conclude the desired result (3).

(B) Let $c \in (\mathsf{BasicAction} \cup \mathsf{AbstractPlan})$ and let $\rho \in applicable(\mathsf{Rule}, c; \pi)$. Then we want to prove $\sigma \models_{\mathsf{Rule}} [apply(\rho, c; \pi)\restriction_{n-1}]\phi$. From definitions 16 and 15, we have the following.

$$\begin{aligned} \sigma \models_{\mathsf{Rule}} [apply(\rho, c; \pi)\restriction_{n-1}]\phi &\Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma) : \sigma' \models_{\mathsf{Rule}} \phi \\ &\Leftrightarrow \forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma)) : \sigma' \models_{\mathsf{Rule}} \phi \end{aligned} \quad (4)$$

From 1, we have that $\kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma)) \subseteq \kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma))$. From this and assumption (2), we can now conclude the desired result (4).

($\Leftarrow$) Assume $\sigma \models_{\mathsf{Rule}} [c\restriction_0][\pi\restriction_n]\phi$ and $\sigma \models_{\mathsf{Rule}} \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi$, i.e., $\forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma_1)) : \sigma' \models_{\mathsf{Rule}} \phi$ (3) and $\forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma)) : \sigma' \models_{\mathsf{Rule}} \phi$ (4).
To prove: $\sigma \models_{\mathsf{Rule}} [c; \pi\restriction_n]\phi$, i.e., $\forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma)) : \sigma' \models_{\mathsf{Rule}} \phi$ (2). If $c \in \mathsf{AbstractPlan}$ or if a transition of the form $\langle c; \pi, \sigma\rangle \rightarrow_{execute} \langle \pi, \sigma_1\rangle$ is not derivable, we have that $\kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma)) = \bigcup_\rho \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma))$ (1). From this and the assumption, we have the desired result.

If $c \in \mathsf{BasicAction}$ and a transition of the form $\langle c; \pi, \sigma\rangle \rightarrow_{execute} \langle \pi, \sigma_1\rangle$ is derivable, we have (1). From this and the assumption, we again have the desired result.

In order to prove completeness of the axiom system, we first prove proposition 2, which says that any formula from $\mathcal{L}_{\mathsf{PRDL}}$ can be rewritten into an equivalent formula where all restriction parameters are 0. This proposition is proven by induction on the size of formulas. The size of a formula is defined by means of the function $size : \mathcal{L}_{\mathsf{PRDL}} \rightarrow \mathbb{N}^3$. This function takes a formula from $\mathcal{L}_{\mathsf{PRDL}}$ and yields a triple $\langle x, y, z\rangle$, where x roughly corresponds to the sum of the restriction parameters occurring in the formula, y roughly corresponds to the sum of the length of plans in the formula and z is the length of the formula.

Definition 18. *(size)* Let the following be a lexicographic ordering on tuples $\langle x, y, z\rangle \in \mathbb{N}^3$:

$$\langle x_1, y_1, z_1\rangle < \langle x_2, y_2, z_2\rangle \text{ iff } x_1 < x_2 \text{ or } (x_1 = x_2 \text{ and } y_1 < y_2) \text{ or } (x_1 = x_2 \text{ and } y_1 = y_2 \text{ and } z_1 < z_2).$$

Let max be a function yielding the maximum of two tuples from $\mathbb{N}^3$ and let f and s respectively be functions yielding the first and second element of a tuple. Let l be a function yielding the number of symbols of a syntactic entity and let $q \in \mathcal{L}$. The function $size : \mathcal{L}_{\mathsf{PRDL}} \rightarrow \mathbb{N}^3$ is then as defined below.

$$\begin{aligned}
size(q) &= \langle 0, 0, l(q)\rangle \\
size([\pi\restriction_n]\phi) &= \begin{cases} \langle n + f(size(\phi)), l(\pi) + s(size(\phi)), l([\pi\restriction_n]\phi)\rangle & \text{if } n > 0 \\ \langle f(size(\phi)), s(size(\phi)), l([\pi\restriction_n]\phi)\rangle & \text{otherwise} \end{cases} \\
size(\neg\phi) &= \langle f(size(\phi)), s(size(\phi)), l(\neg\phi)\rangle \\
size(\phi \wedge \phi') &= \langle f(max(size(\phi), size(\phi'))), s(max(size(\phi), size(\phi'))), l(\phi \wedge \phi')\rangle
\end{aligned}$$

In the proof of proposition 2, we use the following lemma. The first clause specifies that the right-hand side of axiom (PRDL4) is smaller than the left-hand side. This axiom will usually be used by applying it from left to right to prove a formula such as $[\pi\restriction_n]\phi$. Intuitively, the fact that the formula will get "smaller" as specified through the function $size$, suggests convergence of the deduction process.

Lemma 1. Let $\phi \in \mathcal{L}_{\mathsf{PRDL}}$, let $c \in (\mathsf{BasicAction} \cup \mathsf{AbstractPlan})$, let ρ range over $applicable(\mathsf{Rule}, c; \pi)$ and let $n > 0$. The following then holds:

1. $size([c\restriction_0][\pi\restriction_n]\phi \wedge \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi) < size([c; \pi\restriction_n]\phi)$,
2. $size(\phi) < size(\phi \wedge \phi')$ and $size(\phi') < size(\phi \wedge \phi')$.

Proof. The proof is simply by applying definition 18.

Proposition 2. Any formula $\phi \in \mathcal{L}_{\mathsf{PRDL}}$ can be rewritten into an equivalent formula ϕ_{PDL} where all restriction parameters are 0, i.e.:

$$\forall \phi \in \mathcal{L}_{\mathsf{PRDL}} : \exists \phi_{\mathsf{PDL}} \in \mathcal{L}_{\mathsf{PRDL}} : size(\phi_{\mathsf{PDL}}) = \langle 0, 0, l(\phi_{\mathsf{PDL}})\rangle \text{ and } \vdash_{\mathsf{Rule}} \phi \leftrightarrow \phi_{\mathsf{PDL}}.$$

Proof. The fact that a formula ϕ has the property that it can be rewritten as specified in the proposition, will be denoted by $\mathsf{PDL}(\phi)$ for reasons that will become clear in the sequel. The proof is by induction on $size(\phi)$.

- $\phi \equiv q$
  $size(q) = \langle 0, 0, l(q)\rangle$ and let $q_{\mathsf{PDL}} = q$, then $\mathsf{PDL}(q)$.
- $\phi \equiv [\pi\restriction_n]\phi'$
  If $n = -1$, we have that $[\pi\restriction_n]\phi'$ is equivalent with $\top$ ($\mathsf{PRDL1}$). As $\mathsf{PDL}(\top)$, we also have $\mathsf{PDL}([\pi\restriction_n]\phi')$ in this case.
  Let $n = 0$. We then have that $size([\pi\restriction_n]\phi') = \langle f(size(\phi')), s(size(\phi')), l([\pi\restriction_n]\phi')\rangle$ is greater than $size(\phi') = \langle f(size(\phi')), s(size(\phi')), l(\phi')\rangle$. By induction, we then have $\mathsf{PDL}(\phi')$, i.e., ϕ' can be rewritten into an equivalent formula ϕ'_{PDL}, such that $size(\phi'_{\mathsf{PDL}}) = \langle 0, 0, l(\phi'_{\mathsf{PDL}})\rangle$. As $size([\pi\restriction_n]\phi'_{\mathsf{PDL}}) = \langle 0, 0, l([\pi\restriction_n]\phi'_{\mathsf{PDL}})\rangle$, we have $\mathsf{PDL}([\pi\restriction_n]\phi'_{\mathsf{PDL}})$ and therefore $\mathsf{PDL}([\pi\restriction_n]\phi')$.
  Let $n > 0$. Let $\pi \equiv \epsilon$. By lemma 1, we have $size(\phi') < size([\epsilon\restriction_n]\phi')$. Therefore, by induction, $\mathsf{PDL}(\phi')$. As $[\epsilon\restriction_n]\phi'$ is equivalent with ϕ' by axiom ($\mathsf{PRDL3}$), we also have $\mathsf{PDL}([\epsilon\restriction_n]\phi')$. Now let $\pi \equiv c; \pi'$ and let $L = [c; \pi'\restriction_n]\phi'$ and $R = [c\restriction_0][\pi'\restriction_n]\phi' \wedge \bigwedge_\rho [apply(\rho, c; \pi')\restriction_{n-1}]\phi'$. By lemma 1, we have that $size(R) < size(L)$. Therefore, by induction, we have $\mathsf{PDL}(R)$. As R and L are equivalent by axiom ($\mathsf{PRDL4}$), we also have $\mathsf{PDL}(L)$, yielding the desired result.
- $\phi \equiv \neg\phi'$
  We have that $size(\neg\phi') = \langle f(size(\phi')), s(size(\phi')), l(\neg\phi')\rangle$, which is greater than $size(\phi')$. By induction, we thus have $\mathsf{PDL}(\phi')$ and $size(\phi'_{\mathsf{PDL}}) = \langle 0, 0, l(\phi'_{\mathsf{PDL}})\rangle$. Then, $size(\neg\phi'_{\mathsf{PDL}}) = \langle 0, 0, l(\neg\phi'_{\mathsf{PDL}})\rangle$ and thus $\mathsf{PDL}(\neg\phi'_{\mathsf{PDL}})$ and therefore $\mathsf{PDL}(\neg\phi')$.
- $\phi \equiv \phi' \wedge \phi''$
  By lemma 1, we have $size(\phi') < size(\phi' \wedge \phi'')$ and $size(\phi'') < size(\phi' \wedge \phi'')$. Therefore, by induction, $\mathsf{PDL}(\phi')$ and $\mathsf{PDL}(\phi'')$ and therefore $size(\phi'_{\mathsf{PDL}}) = \langle 0, 0, l(\phi'_{\mathsf{PDL}})\rangle$ and $size(\phi''_{\mathsf{PDL}}) = \langle 0, 0, l(\phi''_{\mathsf{PDL}})\rangle$. Then, $size(\phi'_{\mathsf{PDL}} \wedge \phi''_{\mathsf{PDL}}) = \langle 0, 0, l(\phi'_{\mathsf{PDL}} \wedge \phi''_{\mathsf{PDL}})\rangle$ and therefore $size((\phi' \wedge \phi'')_{\mathsf{PDL}}) = \langle 0, 0, l((\phi' \wedge \phi'')_{\mathsf{PDL}})\rangle$ and we can conclude $\mathsf{PDL}((\phi' \wedge \phi'')_{\mathsf{PDL}})$ and thus $\mathsf{PDL}(\phi' \wedge \phi'')$.

Although structural induction is not possible for plans in general, it *is* possible if we only consider action execution, i.e., if the restriction parameter is 0. This is specified in the following proposition, from which we can conclude that a formula ϕ with $size(\phi) = \langle 0, 0, l(\phi)\rangle$ satisfies all standard PDL properties.

Proposition 3. *(sequential composition)* Let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. The following is then derivable in the axiom system $\mathsf{AS}_{\mathsf{Rule}}$.

$$\vdash_{\mathsf{Rule}} [\pi_1; \pi_2{\restriction_0}]\phi \leftrightarrow [\pi_1{\restriction_0}][\pi_2{\restriction_0}]\phi$$

Proof. The proof is through repeated application of axiom (PRDL4), first from left to right and then from right to left (also using axiom (PRDL1) to eliminate the rule application part of the axiom).

Theorem 2. *(completeness)* Let $\phi \in \mathcal{L}_{\mathsf{PRDL}}$ and let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. Then the axiom system $\mathsf{AS}_{\mathsf{Rule}}$ is complete, i.e.:

$$\models_{\mathsf{Rule}} \phi \;\Rightarrow \vdash_{\mathsf{Rule}} \phi.$$

Proof. Let $\phi \in \mathcal{L}_{\mathsf{PRDL}}$. By proposition 2 we have that a formula ϕ_{PDL} exists such that $\vdash_{\mathsf{Rule}} \phi \leftrightarrow \phi_{\mathsf{PDL}}$ and $size(\phi_{\mathsf{PDL}}) = \langle 0, 0, l(\phi_{\mathsf{PDL}})\rangle$ and therefore by soundness of $\mathsf{AS}_{\mathsf{Rule}}$ also $\models_{\mathsf{Rule}} \phi \leftrightarrow \phi_{\mathsf{PDL}}$. Let ϕ_{PDL} be a formula with these properties.

$$\begin{array}{ll} \models_{\mathsf{Rule}} \phi \Leftrightarrow \models_{\mathsf{Rule}} \phi_{\mathsf{PDL}} & (\models_{\mathsf{Rule}} \phi \leftrightarrow \phi_{\mathsf{PDL}}) \\ \Rightarrow \vdash_{\mathsf{Rule}} \phi_{\mathsf{PDL}} & (\text{completeness of } \mathsf{PDL}) \\ \Leftrightarrow \vdash_{\mathsf{Rule}} \phi & (\vdash_{\mathsf{Rule}} \phi \leftrightarrow \phi_{\mathsf{PDL}}) \end{array}$$

The second step in this proof needs some justification. The general idea is, that all PDL axioms and rules are applicable to a formula ϕ_{PDL} and moreover, these axioms and rules are contained in our axiom system $\mathsf{AS}_{\mathsf{Rule}}$. As PDL is complete, we have $\models_{\mathsf{Rule}} \phi_{\mathsf{PDL}} \Rightarrow \vdash_{\mathsf{Rule}} \phi_{\mathsf{PDL}}$. There are however some subtleties to be considered, as our action language is not exactly the same as the action language of PDL, nor is it a subset (at first sight).

In particular, the action language of PDL does not contain abstract plans or the empty action ϵ. These are axiomatized in the system $\mathsf{AS}_{\mathsf{Rule}}$ and the question is, how these axioms relate to the axiom system for PDL. It turns out, that the semantics of $p{\restriction_0}$ and $\epsilon{\restriction_0}$ (or $\epsilon{\restriction_n}$, for that matter) correspond respectively to the special PDL actions `fail` (no resulting states if executed) and `skip` (the identity relation). These actions are respectively defined as $\mathbf{0}?$ and $\mathbf{1}?$. Filling in these actions in the axiom for test ($[\psi?]\phi \leftrightarrow (\psi \rightarrow \phi)$), we get the following, corresponding exactly with the axioms (PRDL2) and (PRDL3).

$$\begin{array}{lllll} [\mathbf{0}?]\phi \leftrightarrow (\mathbf{0} \rightarrow \phi) & \Leftrightarrow & [\mathbf{0}?]\phi & \Leftrightarrow & [\mathtt{fail}]\phi \\ [\mathbf{1}?]\phi \leftrightarrow (\mathbf{1} \rightarrow \phi) & \Leftrightarrow & [\mathbf{1}?]\phi \leftrightarrow \phi & \Leftrightarrow & [\mathtt{skip}]\phi \leftrightarrow \phi \end{array}$$

Our axiom system is complete for formulas ϕ_{PDL}, because it contains the PDL axioms and rules that are applicable to these formulas, that is, the axiom for sequential composition, the axioms for `fail` and `skip` as stated above, the axiom for distribution of box over implication and the rules (MP) and (GEN). The axiom for sequential composition is not explicitly contained in $\mathsf{AS}_{\mathsf{Rule}}$, but is derivable for formulas ϕ_{PDL} by proposition 3. Axiom (PRDL3), i.e., the more general version of $[\epsilon{\restriction_0}]\phi \leftrightarrow \phi$, is needed in the proof of proposition 2, which is used elsewhere in this completeness proof.

5 Conclusion and Future Research

In this paper, we presented a dynamic logic for reasoning about 3APL agents, tailored to handle the plan revision aspect of the language. As we argued, 3APL plans cannot be analyzed by structural induction. Instead, we proposed a logic of restricted plans, which should be used to prove properties of 3APL plans by doing induction on the restriction parameter.

Being able to do structural induction is usually considered an essential property of programs in order to reason about them. As 3APL plans lack this property, it is not at all obvious that it should be possible to reason about them, especially using a clean logic with sound and complete axiomatization. The fact that we succeeded in providing such a logic, thus at least demonstrates this possibility.

We did some preliminary experiments in actually using the logic to prove properties of certain 3APL agents. More research is however needed to establish the practical usefulness of the logic to prove properties of 3APL agents and the possibility to do for example automated theorem proving. In this light, incorporation of interaction with an environment in the semantics is also an important issue for future research.

References

1. R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking AgentSpeak. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 409–416, Melbourne, 2003.
2. M. E. Bratman. *Intention, plans, and practical reason*. Harvard University Press, Massachusetts, 1987.
3. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
4. M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A programming language for cognitive agents: goal directed 3APL. In *Programming multiagent systems, first international workshop (ProMAS'03)*, volume 3067 of *LNAI*, pages 111–130. Springer, Berlin, 2004.
5. J. de Bakker. *Mathematical Theory of Program Correctness*. Series in Computer Science. Prentice-Hall International, London, 1980.
6. R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, and S. Dance. Implementing industrial multi-agent systems using JACK™. In *Proceedings of the first international workshop on programming multiagent systems (ProMAS'03)*, volume 3067 of *LNAI*, pages 18–49. Springer, Berlin, 2004.
7. G. d. Giacomo, Y. Lespérance, and H. Levesque. *ConGolog*, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
8. D. Harel. *First-Order Dynamic Logic*. Lectures Notes in Computer Science 68. Springer, Berlin, 1979.
9. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, Cambridge, Massachusetts and London, England, 2000.

10. H. Hayashi, K. Cho, and A. Ohsuga. A new HTN planning framework for agents in dynamic environments. In *Proceedings of the fourth international workshop on computational logic in multi-agent systems (CLIMA'03)*, pages 108–133, 2003.
11. M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gordon-Spears, editors. *Formal Approaches to Agent-Based Systems (Proceedings of FAABS'02)*, volume 2699 of *LNAI*, Berlin, 2003. Springer.
12. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Int. J. of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
13. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A programming logic for part of the agent language 3APL. In *Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS'00)*, 2000.
14. G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
15. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.
16. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann, 1991.
17. J. Rash, C. Rouff, W. Truszkowski, D. Gordon, and M. Hinchey, editors. *Formal Approaches to Agent-Based Systems (Proceedings of FAABS'01)*, volume 1871 of *LNAI*, Berlin, 2001. Springer.
18. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
19. W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. An integrated modal approach to rational agents. In M. Wooldridge and A. S. Rao, editors, *Foundations of Rational Agency*, Applied Logic Series 14, pages 133–168. Kluwer, Dordrecht, 1998.
20. P. van Emde Boas. The connection between modal logic and algorithmic logics. In *Mathematical foundations of computer science 1978*, volume 64 of *LNCS*, pages 1–15. Springer, Berlin, 1978.
21. M. B. van Riemsdijk, J.-J. Ch. Meyer, and F. S. de Boer. Semantics of plan revision in intelligent agents. Technical report, Utrecht University, Institute of Information and Computing Sciences, 2003. UU-CS-2004-002.
22. M. B. van Riemsdijk, J.-J. Ch. Meyer, and F. S. de Boer. Semantics of plan revision in intelligent agents. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Proceedings of the 10th International Conference on Algebraic Methodology And Software Technology (AMAST04)*, volume 3116 of *LNCS*, pages 426–442. Springer-Verlag, 2004.
23. M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. 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)*, pages 393–400, Melbourne, 2003.
24. M. Wooldridge. Agent-based software engineering. *IEEE Proceedings Software Engineering*, 144(1):26–37, 1997.