Dynamic Logic for Plan Revision in Agent Programming

M. BIRNA VAN RIEMSDIJK, *ICS, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands.*
Email: birna@cs.uu.nl

FRANK S. DE BOER, *ICS, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands and CWI, Amsterdam, The Netherlands and LIACS, Leiden University, The Netherlands.*
Email: frankb@cs.uu.nl

JOHN-JULES Ch. MEYER, *ICS, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands.*
Email: jj@cs.uu.nl

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 by means of plan revision rules. 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. The logic is one for plans that are restricted in a certain way. For this logic, we give a sound and complete axiomatization. Further, we discuss how this logic for restricted 3APL plans can be extended to a logic for non-restricted plans and we discuss some example proofs, using the logic. Finally, we consider the relation between proving properties of 3APL agents and proving properties of procedural programs.

Keywords: Specification and verification of programs, agent programming languages, dynamic logic, operational semantics, plan revision.

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 [39]. 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 use these mental attitudes in order to exhibit the desired flexible problem solving behaviour.

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 [24, 16].

Vol. 16 No. 3,
doi:10.1093/logcom/exi084

In this paper, we are concerned with the verification of agents programmed in (a simplified version of) the cognitive agent programming language *3APL*[1] [17, 37, 6]. This language is based on theoretical research on cognitive notions [3, 5, 23, 27]. In the latest version [6], 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 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 behaviour 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.[2] Now, in order to increase the possible flexibility of agents, 3APL [17] 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 [22, 26, 12, 10, 21]. The idea is that an agent should not blindly execute an adopted plan, but should be able to revise it under certain conditions. As this paper focuses on the plan revision aspect of 3APL, we consider a version of the language with only beliefs and plans, i.e. without goals. For more background on how to incorporate goals in an agent programming language, we refer the reader to [37, 31, 32, 33, 38, 4]. We will use a propositional and otherwise slightly simplified variant of the original 3APL language as defined in [17].

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.

The outline of the paper is as follows. In Section 2, we address related work. After defining (a simplified version of) 3APL and its semantics (Section 3), we propose a dynamic logic for proving properties of 3APL plans in the context of plan revision rules (Section 4). As will become clear, this is actually not a logic for general 3APL plans, but the plans that the logic can deal with are restricted in a certain way. For this logic, we provide a sound and complete axiomatization (Section 5). In Section 6, we discuss how this logic for restricted 3APL plans can be extended to a logic for non-restricted plans and we discuss some example proofs, using the logic. Finally, we consider the relation between proving properties of procedural programs and proving properties of 3APL agents in Section 7. In particular, we compare procedures with 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.[3] 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

[1] 3APL is to be pronounced as 'triple-a-p-l'.

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

[3] Parts of this work have been published in [34].

further research.

2 Related work

This research builds on a body of work done in the area of theoretical computer science on *formal semantics* and *logics* of programming languages [7]. A formal semantics for a programming language is used to formally specify the meaning of the programs written in this language. Specifying the meaning of a programming language using formal semantics is important for a number of reasons. For example, the specification of a formal semantics can be used to identify issues and problems with the language. Defining the semantics forces one to be precise, which might uncover problems overlooked earlier. Also, the semantics can serve as a basis for comparing various languages. Further, and most important in this context, it is a necessary prerequisite if one wants to do formal verification of programs written in some language. One cannot claim to have proven that a program satisfies a certain property, without knowing exactly what this program does.

Semantics of programming languages can be defined in different ways. One kind of semantics is the so-called operational semantics (see Section 3.2 for further explanation), which is the kind of semantics generally used to specify the meaning of 3APL. The operational semantics of plan revision rules, which is important in this paper, 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 [7]. 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 semantics of the sequential composition operator no longer being compositional. We elaborate on this issue in Sections 4 and 7. The issue has also been considered from a semantic perspective in [35, 36].

With respect to verification, there are in general two approaches: model checking [9] and theorem proving [7]. In model checking, a model is built describing the execution of the program, and it is checked whether some temporal property is satisfied by this model. An example of work on model checking in the area of agent programming languages is [2], in which model checking of the agent programming language AgentSpeak is addressed.

In theorem proving, on which we focus in this paper, a program is proven to satisfy a certain property using a logic with deductive system or axiomatization. Various logics can be used for this purpose, such as Hoare logic (see [1] for a survey) and dynamic logic [15], which we use in this paper. In the context of 3APL, a sketch of a dynamic logic to reason about programs written in this language has been given in [37]. 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 [18], a programming logic (without axiomatization) was given for a fragment of 3APL without plan revision rules.

Further, we mention related work in the field of *planning*. In general, planning deals with the problem of how to get from some current state to a desired goal state through a sequence of actions forming the plan. The way this problem is approached in this field, is by *searching* for an appropriate plan using a specification of the available actions and their preconditions and effects [25]. The search space can however become quite large in realistic problems. Part of the planning research thus involves the investigation of more efficient ways in which this search can be performed, and the development of heuristics to guide this search.

While the general objective of planning, i.e. generating a plan with which the agent can achieve its goals, is closely related to the objective of cognitive agent programming, the techniques that

are used in the two fields differ. Planning involves *reasoning* about the effects of actions, while in a programming context it is the *programmer* who defines the available plans, together with the situations in which they might be executed. In fact, the so-called Procedural Reasoning System [11] on which most of today's agent programming languages, including 3APL, are directly or indirectly based, was proposed as an alternative to the traditional planning systems.[4] It was in part motivated by the observation that those systems require search through potentially large search spaces. In contrast with research in the field of planning, research regarding agent programming languages involves the design and investigation of appropriate *programming constructs* for the specification of plans, and in this paper we are concerned with the programming construct of plan revision rules of the 3APL language in particular. Also, the structure of plans generally differs between the two fields: in planning, a plan often consists of a partially ordered set of actions, while in agent programming the structure of plans is generally simpler.

Nevertheless, the general idea of plan revision as incorporated in 3APL is also being investigated in the field of planning. In that context, it is mostly referred to as *plan repair*. The motivation for that work is similar to the motivation of the addition of plan revision capabilities to 3APL agents, i.e. things may go wrong during execution of a plan since the agent executes its plans in some environment, and in that case the agent will have to replan, or to adapt the old plan to the changed circumstances [13, 28]. In theory, modifying an existing plan is (worst-case) no more efficient than a complete replanning [19], but the idea is that in practice, plan repair is often more efficient [28].

While approaches to plan repair mostly involve reasoning about actions, there are also some approaches which use precompiled plans to do plan repair, such as [29, 8]. The latter approaches are somewhat more closely related to plan revision as done in 3APL, since the way in which plans may be repaired is prespecified in both approaches. Nevertheless, these approaches to plan repair are embedded in a general planning framework, and the exact relation with plan revision as used in 3APL is not immediately clear. Investigations along these lines fall outside the scope of this paper, but form an interesting issue for future research.

3 3APL

3.1 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.[5]

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

DEFINITION 3.1 (belief bases)
Assume a propositional language $\mathcal{L}$ with typical formula p and the connectives $\wedge$ and $\neg$ with the

[4] An exception to this distinction between planning and programming is formed by the language ConGolog [12], in which both approaches are combined. ConGolog is based on the situation calculus, and a program written in the language is used to constrain the search space for finding an appropriate plan.

[5] Abstract plans could also be modelled as non-executable basic actions.

usual meaning. Then the set of belief bases Σ with typical element σ is defined to be $\wp(\mathcal{L})$.[6]

DEFINITION 3.2 (plans)
Assume that a set BasicAction with typical element a is given, together with a set AbstractPlan with typical element p.[7] Then the set of plans Plan with typical element π is defined as follows:

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

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 taken to be identical to 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 3.3 (configuration)
Let Σ be the set of belief bases and let Plan be the set of plans. Then $\mathsf{Plan} \times \Sigma$ 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 3.4 (plan revision (PR) rules)
The set of PR rules $\mathcal{R}$ is defined as follows: $\mathcal{R} = \{\pi_h \rightsquigarrow \pi_b \mid \pi_h, \pi_b \in \mathsf{Plan}, \pi_h \neq \epsilon\}$.[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 3.5 (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.

---

[6] $\wp(\mathcal{L})$ denotes the powerset of $\mathcal{L}$.

[7] Note that we use p to denote an element from the propositional language $\mathcal{L}$, as well as an element from AbstractPlan. It will however be indicated explicitly which kind of element is meant.

[8] In [17], 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.

3.2 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 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 [7] for details on this subject).

The operational semantics of a language is usually defined using transition systems [20]. 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 [17]. 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 3.6 (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 3.7 (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 3.8 (rule application)
Let $\mathcal{R}$ be the set of PR rules and let Plan be the set of plans. Let $\rho : \pi_h \rightsquigarrow \pi_b \in \mathcal{R}$ and $\pi, \pi' \in \mathsf{Plan}$. The partial function $apply : (\mathcal{R} \times \mathsf{Plan}) \rightarrow \mathsf{Plan}$ 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 $applicable : (\wp(\mathcal{R}) \times \mathsf{Plan}) \rightarrow \wp(\mathcal{R})$ 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 sequence, 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 3.9 (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 3.10 (function for calculating computation sequences)
Let $x_i \in \{exec, app\}$ for $1 \leq i \leq m$. The function $\mathcal{C}^{\mathcal{A}} : (\mathsf{Plan} \times \Sigma) \rightarrow \wp(\Sigma^+)$ 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 3.11 (operational semantics)
Let $\kappa : \Sigma^+ \rightarrow \Sigma$ be a function yielding the last element of a finite computation sequence, extended to handle sets of computation sequences as follows, where I is some set of indices: $\kappa(\{\delta_i \mid i \in I\}) = \{\kappa(\delta_i) \mid i \in I\}$. The operational semantic function $\mathcal{O}^{\mathcal{A}} : \mathsf{Plan} \rightarrow (\Sigma \rightarrow \wp(\Sigma))$ is defined as follows:

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

We will sometimes omit the superscript $\mathcal{A}$ to functions as defined above, for reasons of presentation.

EXAMPLE 3.12
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 σ_{ab} be the belief base 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 $\mathcal{O}^{\mathcal{A}}(p; a)(\sigma) = \{\sigma_b, \sigma_{ca}\}$.

4 Plan revision 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 [14, 15], 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 [30]), 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 valid: $[\pi_1;\pi_2]\phi \leftrightarrow [\pi_1][\pi_2]\phi$. 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 3.12. 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 were 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.

4.1 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, 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 4.1 (restricted plans)
Let Plan be the language of plans and let $\mathbb{N}^- = \mathbb{N} \cup \{-1\}$. Then, the language Plan_r of restricted plans is defined as $\{\pi\restriction_n \mid \pi \in \mathsf{Plan}, n \in \mathbb{N}^-\}$.

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 4.2 (plan revision dynamic logic (PRDL))
Let $\pi\restriction_n \in \mathsf{Plan}_r$ be a restricted plan and let $\mathcal{A}$ be a 3APL agent (Definition 3.5). 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 $\phi \wedge \phi' \in \mathcal{L}_{\mathsf{PRDL}}$.

4.2 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.

[9] We will use the word 'execution' in two ways. First, 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. Second, 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.

[11] Or with the number -1, it will become clear in the sequel why we need this.

DEFINITION 4.3 (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 $\mathcal{C}_r^{\mathcal{A}} : (\mathsf{Plan}_r \times \Sigma) \rightarrow \wp(\Sigma^+)$ 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 3.10, 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.

DEFINITION 4.4 (operational semantics)
Let κ be as in Definition 3.11. The operational semantic function $\mathcal{O}_r^{\mathcal{A}} : \mathsf{Plan}_r \rightarrow (\Sigma \rightarrow \wp(\Sigma))$ is defined as follows:

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

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

DEFINITION 4.5 (semantics of PRDL)
Let $p \in \mathcal{L}$ 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.

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

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 $\models_{\mathsf{Rule}} \phi$.

5 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 non-restricted 3APL plans. The idea is that this can be done using the axiom system for restricted plans, by relating the semantics of restricted plans to that of non-restricted plans. We will explain and elaborate on this in Section 6.

DEFINITION 5.1 (axiom system ($\mathsf{AS}_{\mathsf{Rule}}$))
Let BasicAction be a set of basic actions, AbstractPlan be a set of abstract plans and $\mathsf{Rule} \subseteq \mathcal{R}$ 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)	$[\epsilon\restriction_n]\phi \leftrightarrow \phi$	with $0 \leq n$
(PRDL4)	$[c;\pi\restriction_n]\phi \leftrightarrow [c\restriction_0][\pi\restriction_n]\phi \wedge \bigwedge_\rho [apply(\rho, c; \pi)\restriction_{n-1}]\phi$	with $0 \leq n$
(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.

We will now explain the PRDL axioms of the system. The other axioms and the rules are standard for propositional dynamic logic (PDL) [14]. 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 3.6 and 3.7). 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 analyse the plan $a; \pi$ by analysing 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 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 $c; \epsilon$.

5.1 Soundness

The axiom system of Definition 5.1 is sound.

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

[13] See Definition 3.8 for the definitions of the functions $apply$ and $applicable$.

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

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

THEOREM 5.2 (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 $\mathsf{AS}_{\mathsf{Rule}}$. In the following, let $\pi \in \mathsf{Plan}$ be an arbitrary plan and let $\phi \in \mathcal{L}_{\mathsf{PRDL}}$ be an arbitrary PRDL formula. Furthermore, $\mathcal{A} = \langle \mathsf{Rule}, \mathcal{T} \rangle$ and $\models_{\langle \mathsf{Rule}, \mathcal{T} \rangle}$ will be abbreviated by $\models_{\mathsf{Rule}}$.

(PRDL1) To prove: $\forall \mathcal{T}, \sigma : \sigma \models_{\mathsf{Rule}} [\pi\restriction_{-1}]\phi$. Let $\sigma \in \Sigma$ be an arbitrary belief base and let $\mathcal{T}$ be an arbitrary belief update function. We have that $\sigma \models_{\mathsf{Rule}} [\pi\restriction_{-1}]\phi \Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(\pi\restriction_{-1})(\sigma) : \sigma' \models_{\mathsf{Rule}} \phi$ by Definition 4.5. Furthermore, $\mathcal{O}_r^{\mathcal{A}}(\pi\restriction_{-1})(\sigma) = \emptyset$ by Definition 4.4, trivially yielding the desired result.

(PRDL2) Let $p \in \mathsf{AbstractPlan}$ be an arbitrary abstract plan. To prove: $\forall \mathcal{T}, \sigma : \sigma \models_{\mathsf{Rule}} [p\restriction_0]\phi$. Let $\sigma \in \Sigma$ be an arbitrary belief base and let $\mathcal{T}$ be an arbitrary belief update function. We have that $\sigma \models_{\mathsf{Rule}} [p\restriction_0]\phi \Leftrightarrow \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(p\restriction_0)(\sigma) : \sigma' \models_{\mathsf{Rule}} \phi$ by Definition 4.5. Furthermore, $\mathcal{O}_r^{\mathcal{A}}(p\restriction_0)(\sigma) = \emptyset$ by Definition 3.6, trivially yielding the desired result.

(PRDL3) To prove: $\forall \mathcal{T}, \sigma : \sigma \models_{\mathsf{Rule}} [\epsilon\restriction_n]\phi \leftrightarrow \phi$ where $n \geq 0$, i.e. $\forall \mathcal{T}, \sigma : (\sigma \models_{\mathsf{Rule}} [\epsilon\restriction_n]\phi \Leftrightarrow \sigma \models_{\mathsf{Rule}} \phi)$. Let $\sigma \in \Sigma$ be an arbitrary belief base and let $\mathcal{T}$ be an arbitrary belief update function. By Definition 4.3, we have that $\mathcal{C}_r^{\mathcal{A}}(\epsilon\restriction_n, \sigma) = \{\sigma\}$ where $n \geq 0$, i.e.

$$\kappa(\mathcal{C}_r^{\mathcal{A}}(\epsilon\restriction_n, \sigma)) = \{\sigma\}. \tag{5.1}$$

By Definitions 4.5 and 4.4 and (5.1), we have the following, yielding the desired result:

$$\begin{array}{lll} \sigma \models_{\mathsf{Rule}} [\epsilon\restriction_n]\phi & \Leftrightarrow & \forall \sigma' \in \mathcal{O}_r^{\mathcal{A}}(\epsilon\restriction_n)(\sigma) : \sigma' \models_{\mathsf{Rule}} \phi \\ & \Leftrightarrow & \forall \sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\epsilon\restriction_n, \sigma)) : \sigma' \models_{\mathsf{Rule}} \phi \\ & \Leftrightarrow & \sigma \models_{\mathsf{Rule}} \phi \end{array}$$

(PRDL4) 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 4.3. Let ρ range over $applicable(\mathsf{Rule}, c; \pi)$. Now, observe the following by Definition 4.3:

$$\kappa(\mathcal{C}_r^{\mathcal{A}}(c; \pi\restriction_n, \sigma)) = \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n, \sigma_1)) \cup \bigcup_\rho \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho, c; \pi)\restriction_{n-1}, \sigma)). \tag{5.2}$$

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, the first term of the right-hand side of (5.2) is empty.

($\Rightarrow$) Assume $\sigma \models_{\mathsf{Rule}} [c;\pi\restriction_n]\phi$, i.e. by Definition 4.5 $\forall\sigma' \in \mathcal{O}_r^{\mathcal{A}}(c;\pi\restriction_n,\sigma) : \sigma' \models_{\mathsf{Rule}} \phi$, i.e. by Definition 4.4:

$$\forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c;\pi\restriction_n,\sigma)) : \sigma' \models_{\mathsf{Rule}} \phi. \tag{5.3}$$

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 4.5 and 4.4.

$$\begin{array}{rcl} \sigma \models_{\mathsf{Rule}} [c\restriction_0][\pi\restriction_n]\phi & \Leftrightarrow & \forall\sigma' \in \mathcal{O}_r^{\mathcal{A}}(c\restriction_0,\sigma) : \sigma' \models_{\mathsf{Rule}} [\pi\restriction_n]\phi \\ & \Leftrightarrow & \forall\sigma' \in \mathcal{O}_r^{\mathcal{A}}(c\restriction_0,\sigma) : \forall\sigma'' \in \mathcal{O}_r^{\mathcal{A}}(\pi\restriction_n,\sigma') : \sigma'' \models_{\mathsf{Rule}} \phi \\ & \Leftrightarrow & \forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c\restriction_0,\sigma)) : \forall\sigma'' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n,\sigma')) : \sigma'' \models_{\mathsf{Rule}} \phi \\ & \Leftrightarrow & \forall\sigma'' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n,\sigma_1)) : \sigma'' \models_{\mathsf{Rule}} \phi \end{array} \tag{5.4}$$

From (5.2), 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 (5.3), we can now conclude the desired result (5.4).

(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 4.5 and 4.4, we have the following:

$$\begin{array}{rcl} \sigma \models_{\mathsf{Rule}} [apply(\rho,c;\pi)\restriction_{n-1}]\phi & \Leftrightarrow & \forall\sigma' \in \mathcal{O}_r^{\mathcal{A}}(apply(\rho,c;\pi)\restriction_{n-1},\sigma) : \sigma' \models_{\mathsf{Rule}} \phi \\ & \Leftrightarrow & \forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho,c;\pi)\restriction_{n-1},\sigma)) : \sigma' \models_{\mathsf{Rule}} \phi \end{array} \tag{5.5}$$

From (5.2), 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 (5.3), we can now conclude the desired result (5.5).

($\Leftarrow$) 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. $\forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(\pi\restriction_n,\sigma_1)) : \sigma' \models_{\mathsf{Rule}} \phi$ (5.4) and $\forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho,c;\pi)\restriction_{n-1},\sigma)) : \sigma' \models_{\mathsf{Rule}} \phi$ (5.5).
To prove: $\sigma \models_{\mathsf{Rule}} [c;\pi\restriction_n]\phi$, i.e. $\forall\sigma' \in \kappa(\mathcal{C}_r^{\mathcal{A}}(c;\pi\restriction_n,\sigma)) : \sigma' \models_{\mathsf{Rule}} \phi$ (5.3). 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 $\kappa(\mathcal{C}_r^{\mathcal{A}}(c;\pi\restriction_n,\sigma)) = \bigcup_\rho \kappa(\mathcal{C}_r^{\mathcal{A}}(apply(\rho,c;\pi)\restriction_{n-1},\sigma))$ (5.2). 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 (5.2). From this and the assumption, we again have the desired result. ∎

5.2 Completeness

In order to prove completeness of the axiom system, we first prove Proposition 5.5, 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. The idea is, that the size of a formula is 0 if all restriction parameters are 0. In order to make the induction technically possible, we however also need to incorporate the length of plans and of the formula into the function $size$. This is explained further after the definition of the function.

DEFINITION 5.3 (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 $l(\epsilon) = 0$. The function $size : \mathcal{L}_{\mathsf{PRDL}} \to \mathbb{N}^3$ is then as defined below.

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

Note that when calculating the plan length of a formula $[\pi\restriction_n]\phi$, i.e. the second element of the tuple $size([\pi\restriction_n]\phi)$, the length of π is added to the length of the plans in ϕ in case $n > 0$. If however $n = 0$ or $n = -1$, the length of π is *not* added to the length of the plans in ϕ and $s(size(\phi))$ is simply returned. This definition of the function $size$ results in the fact that a formula ϕ in which all restriction parameters are 0 (or -1), will satisfy $size(\phi) = \langle 0, 0, l(\phi)\rangle$. Further, this definition gives us that $size([c\restriction_0][\pi\restriction_n]\phi)$ is smaller than $size([c; \pi\restriction_n]\phi)$, which is needed in the proof of Lemma 5.4, which will be used in the proof of Proposition 5.5.

Clause (5.7) of Lemma 5.4 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 5.4
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 hold:

$$size(\phi) < size([\epsilon\restriction_n]\phi) \tag{5.6}$$
$$size([c\restriction_0][\pi\restriction_n]\phi \wedge \bigwedge_{\rho}[apply(\rho, c; \pi)\restriction_{n-1}]\phi) < size([c; \pi\restriction_n]\phi) \tag{5.7}$$
$$size(\phi) < size(\phi \wedge \phi') \tag{5.8}$$
$$size(\phi') < size(\phi \wedge \phi') \tag{5.9}$$

PROOF. First, we prove (5.6). From Definition 5.3, we have

$$size([\epsilon\restriction_n]\phi) = \langle n + f(size(\phi)), s(size(\phi)), l([\epsilon\restriction_n]\phi)\rangle.$$

This is bigger than $size(\phi)$.

Now we prove (5.7). We have the following from Definition 5.3, using that $n > 0$:

$$\begin{array}{lcl}
size([c; \pi\restriction_n]\phi) & = & \langle n + f(size(\phi)), l(c; \pi) + s(size(\phi)), l([c; \pi\restriction_n]\phi)\rangle, \\
size([c\restriction_0][\pi\restriction_n]\phi) & = & \langle n + f(size(\phi)), l(\pi) + s(size(\phi)), l([c\restriction_0][\pi\restriction_n]\phi)\rangle, \\
size([apply(\rho, c; \pi)\restriction_{n-1}]\phi) & = & \langle (n-1) + f(size(\phi)), l(apply(\rho, c; \pi)) \\
 & & +s(size(\phi)), l([apply(\rho, c; \pi)\restriction_{n-1}]\phi)\rangle.
\end{array}$$

Let $F = [c\restriction_0][\pi\restriction_n]\phi$ and $S = [apply(\rho, c; \pi)\restriction_{n-1}]\phi$. Then, $max(size(F), size(S)) = size(F)$ for any PR rule ρ. Thus, $size(F \wedge \bigwedge_\rho S) = \langle n + f(size(\phi)), l(\pi) + s(size(\phi)), l(F \wedge \bigwedge_\rho S)\rangle$, which is smaller than $size([c; \pi\restriction_n]\phi)$, yielding the desired result.

Finally, we prove (5.8) and (5.9). First, we show that $size(\phi) < size(\phi \wedge \phi')$, which we will refer to by R. We thus have to show:

$$\langle f(size(\phi)), s(size(\phi)), l(\phi)\rangle < \langle f(max(size(\phi), size(\phi'))), s(max(size(\phi), size(\phi'))), l(\phi \wedge \phi')\rangle.$$

If $f(size(\phi)) < f(max(size(\phi), size(\phi')))$, we have R. If $f(size(\phi)) = f(max(size(\phi), size(\phi')))$ and $s(size(\phi)) < s(max(size(\phi), size(\phi')))$, we again have R. If $s(size(\phi)) = s(max(size(\phi), size(\phi')))$, we also have R, because $l(\phi) < l(\phi \wedge \phi')$. Covering all cases, this yields the desired result. The same line of reasoning can be applied to show $size(\phi') < size(\phi \wedge \phi')$. ∎

Now we can formulate and prove the following proposition.

PROPOSITION 5.5
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 $size(\phi)$.

- $\phi \equiv p$
  $size(p) = \langle 0, 0, l(p)\rangle$ and let $p_{\mathsf{PDL}} = p$, then $\mathsf{PDL}(p)$.
- $\phi \equiv [\pi\restriction_n]\phi'$
  If $n = -1$, we have that $[\pi\restriction_n]\phi'$ is equivalent with $\top$ (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 $\pi = \epsilon$. By Lemma 5.4, 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 (PRDL3), we also have $\mathsf{PDL}([\epsilon\restriction_n]\phi')$.
  Now let $\pi = c; \pi'$ 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 5.4, we have that $size(R) < size(L)$. Therefore, by induction, we have $\mathsf{PDL}(R)$. As R and L are equivalent by axiom (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 5.4, 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 5.6 (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. If $\pi_1 = \epsilon$, we have $[\pi_2\restriction_0] \leftrightarrow [\epsilon\restriction_0][\pi_2\restriction_0]\phi$ by axiom (PRDL3). Otherwise, let $c_i \in (\mathsf{BasicAction} \cup \mathsf{AbstractPlan})$ for $i \geq 1$, let $\pi_1 = c_1; \ldots; c_n$, with $n \geq 1$. 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), we derive the desired result.[16]

$$\begin{array}{rcl}
[\pi_1; \pi_2\restriction_0]\phi & \leftrightarrow & [c_1; \ldots; c_n; \pi_2\restriction_0]\phi \\
& \leftrightarrow & [c_1\restriction_0][c_2; \ldots; c_n; \pi_2\restriction_0]\phi \\
& \leftrightarrow & \ldots \\
& \leftrightarrow & [c_1\restriction_0][c_2\restriction_0]\ldots[c_n\restriction_0][\pi_2\restriction_0]\phi \\
& \leftrightarrow & [c_1; c_2\restriction_0][c_3\restriction_0]\ldots[c_n\restriction_0][\pi_2\restriction_0]\phi \\
& \leftrightarrow & \ldots \\
& \leftrightarrow & [c_1; \ldots; c_n\restriction_0][\pi_2\restriction_0]\phi \\
& \leftrightarrow & [\pi_1\restriction_0][\pi_2\restriction_0]\phi
\end{array}$$

∎

THEOREM 5.7 (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 5.5 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.

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

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

[16] We use the notation $\phi_1 \leftrightarrow \phi_2 \leftrightarrow \phi_3 \leftrightarrow \ldots$, which should be read as a shorthand for $\phi_1 \leftrightarrow \phi_2$ and $\phi_2 \leftrightarrow \phi_3$ and $\ldots$ This notation will also be used in the sequel.

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).

The action language of PDL is built using basic actions, sequential composition, test, nondeterministic choice and iteration. The action language of PRDL is built using basic actions, abstract plans, empty plans and sequential composition. If we for the moment disregard abstract plans and empty plans, the language PRDL is a subset of the language PDL. If we take the subset of PDL axioms and rules dealing with formulas in this subset, this axiom system should be complete with respect to these formulas.

The action language of full PRDL, however, also contains abstract plans and empty plans. The question is, how these should be axiomatized such that we obtain a complete axiomatization. In order to answer this question, we make the following observation. In a formula ϕ_{PDL}, abstract and empty plans can only occur with a 0 restriction parameter by definition. Further, the semantics of a formula $[p\restriction_0]\phi_{\mathsf{PDL}}$ where p is an abstract plan, is similar to the semantics of the `fail` statement of (an extended version of) PDL. The set of states resulting from "execution" of both statements is empty.[17] The semantics of a formula $[\epsilon\restriction_0]\phi_{\mathsf{PDL}}$ is similar to the semantics of the `skip` statement of PDL. The set of states resulting from the execution of both statements in a state σ is $\{\sigma\}$,[18] i.e. the semantics is the identity relation. The action language of PRDL can thus be considered to be a subset of the action language of PDL, where $p\restriction_0$ and $\epsilon\restriction_0$ correspond respectively to `fail` and `skip`.

Now, `fail` and `skip` are not axiomatized in the basic axiom system of PDL. These statements are however, defined as $\mathbf{0}?$ and $\mathbf{1}?$ respectively and the test statement *is* axiomatized: $[\psi?]\phi \leftrightarrow (\psi \rightarrow \phi)$. We now fill in $\mathbf{0}$ and $\mathbf{1}$ for ψ in this axiom, which gives us the following.

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

The statements `fail` and `skip` are thus implicitly axiomatized through the axiomatization of the test. For our axiom system to be complete for formulas ϕ_{PDL}, it should thus contain 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 latter three are explicitly contained in $\mathsf{AS}_{\mathsf{Rule}}$. The axiom for sequential composition is derivable in the system $\mathsf{AS}_{\mathsf{Rule}}$ for formulas ϕ_{PDL}, by Proposition 5.6. Axiom (PRDL2) for $p\restriction_0$ corresponds with the axiom for `fail`. The axiom for $\epsilon\restriction_0$, corresponding with the axiom for `skip`, is an instantiation of axiom (PRDL3). Axiom (PRDL3), i.e. the more general version of $[\epsilon\restriction_0]\phi \leftrightarrow \phi$, is needed in the proof of Proposition 5.5, which is used elsewhere in this completeness proof. ∎

We conclude with a remark with respect to axiom (PRDL3). In the proof above, we explained that the semantics of $\epsilon\restriction_0$ and `skip` are equivalent. As it turns out (see Proposition 5.8), $[\epsilon\restriction_0]\phi$ is equivalent with $[\epsilon\restriction_n]\phi$, as can be proven from axiom (PRDL3), which is thus also equivalent with `skip`.

PROPOSITION 5.8 (empty plan)
Let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. The following is then derivable in the axiom system $\mathsf{AS}_{\mathsf{Rule}}$.

$$\vdash_{\mathsf{Rule}} [\epsilon\restriction_0]\phi \leftrightarrow [\epsilon\restriction_n]\phi \text{ with } 0 \leq n$$

[17] An abstract plan p cannot be executed directly, it can only be transformed using PR rules. The restriction parameter is, however, 0, so no PR rules may be applied and the set $\mathcal{O}_r^{\mathcal{A}}([p\restriction_0]\phi)(\sigma) = \emptyset$ for all $\mathcal{A}$ and σ.

[18] $\mathcal{C}_r^{\mathcal{A}}([\epsilon\restriction_0]\phi_{\mathsf{PDL}})(\sigma) = \{\sigma\} = \kappa(\mathcal{C}_r^{\mathcal{A}}([\epsilon\restriction_0]\phi_{\mathsf{PDL}})(\sigma)) = \mathcal{O}_r^{\mathcal{A}}([\epsilon\restriction_0]\phi_{\mathsf{PDL}})(\sigma)$.

PROOF.

1.	$[\epsilon\restriction_n][\epsilon\restriction_0]\phi \leftrightarrow [\epsilon\restriction_0]\phi$	(PRDL3)
2.	$[\epsilon\restriction_0]\phi \leftrightarrow \phi$	(PRDL3)
3.	$[\epsilon\restriction_n][\epsilon\restriction_0]\phi \leftrightarrow [\epsilon\restriction_n]\phi$	2, (GEN), (PDL)
4.	$[\epsilon\restriction_0]\phi \leftrightarrow [\epsilon\restriction_n]\phi$	$1, 3$, (PL)

∎

6 Proving properties of non-restricted plans

In Sections 4 and 5 we have presented a logic for restricted plans with sound and complete axiomatization. This means that it should be possible to construct a proof for a formula $[a; b\restriction_3]\phi$ if and only if it is true for a given agent. This might be considered an interesting result, but our ultimate aim is to prove properties of non-restricted 3APL plans.

The semantics of restricted plans is closely related to the semantics of non-restricted plans. Using this relation, we will show how the proof system for restricted plans can be extended to a proof system for non-restricted plans. Then we will discuss the usability of this system, using examples.

6.1 From restricted to non-restricted plans

We first add the following clause to the language $\mathcal{L}_{\mathsf{PRDL}}$ (Definition 4.2),[19] yielding a language that we will call $\mathcal{L}_{\mathsf{PRDL}^+}$: if $\phi \in \mathcal{L}_{\mathsf{PRDL}^+}$ and $\pi \in \mathsf{Plan}$, then $[\pi]\phi \in \mathcal{L}_{\mathsf{PRDL}^+}$. By means of this construct, we can thus specify properties of non-restricted plans. We define the semantics of this construct in terms of the operational semantics of non-restricted plans as follows.

DEFINITION 6.1 (semantics of PRDL^+)
Let $\mathcal{A}$ be a 3APL agent (Definition 3.5). The semantics of formulas not of the form $[\pi]\phi$ with $\phi \in \mathcal{L}_{\mathsf{PRDL}^+}$ is as in Definition 4.5. The semantics of formulas of the form $[\pi]\phi$ is as defined below:

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

This definition thus takes the operational semantics of non-restricted plans to define the semantics of constructs of the form $[\pi]\phi$. In the following proposition, we relate the operational semantics of plans and the operational semantics of restricted plans.

PROPOSITION 6.2

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

PROOF. Immediate from Definitions 4.4, 4.3, 3.11 and 3.10. ∎

From this proposition, we have the following corollary, which shows how the construct $[\pi\restriction_n]\phi$ is related to the construct $[\pi]\phi$.

COROLLARY 6.3

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

[19] Replacing each occurrence of $\mathcal{L}_{\mathsf{PRDL}}$ in this definition by $\mathcal{L}_{\mathsf{PRDL}^+}$.

PROOF. Immediate from Proposition 6.2, Definition 4.5 and Definition 6.1. ∎

From this corollary, we can conclude that we can prove a property of the form $[\pi]\phi$ by proving $\forall n \in \mathbb{N} : \vdash_{\mathsf{Rule}} [\pi\restriction_n]\phi$, using the system for restricted plans. This idea can be captured in a proof rule as follows.

DEFINITION 6.4 (proof rule for non-restricted plans)

$$\frac{[\pi\restriction_n]\phi,\ n \in \mathbb{N}}{[\pi]\phi}$$

This rule should be read as having an infinite number of premisses, i.e. $[\pi\restriction_0]\phi$, $[\pi\restriction_1]\phi$, $[\pi\restriction_2]\phi$, ... (see also [15]). Deriving a formula $[\pi]\phi$ using this infinitary rule thus requires infinitely many premisses to have been previously derived.

The rule is sound by Corollary 6.3. The system $\mathsf{AS}_{\mathsf{Rule}}$ for restricted plans (Definition 5.1) taken together with the rule above, is a complete axiom system for PRDL^+: if $[\pi]\phi$ is true then each of the premisses of the rule is true (Corollary 6.3) and each of these premisses can be proven by completeness of $\mathsf{AS}_{\mathsf{Rule}}$. The notion of a proof in this case is, however, non-standard, as a proof can be infinite. This completeness result is therefore theoretical, and putting the system to use in this way is obviously problematic.

One way to try to deal with this problem is the following. 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, rather than proving $[\pi\restriction_n]\phi$ for each n. 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. In the next section we will illustrate how this could be done, using examples. The examples, however, show that it is not obvious that this kind of induction can be applied in all cases.

6.2 Examples

EXAMPLE 6.5

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 5.6. 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:

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

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

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

EXAMPLE 6.6

We will prove a property of a very simple 3APL agent using axiom (PRDL4) and induction on the number of PR rule applications. Our agent has one PR rule: $\mathsf{Rule} = \{a \rightsquigarrow a; a\}$. Furthermore, assume that $\mathcal{T}$ is defined such that $[a\restriction_0]\phi$. We want to prove the following: $\forall n \in \mathbb{N} : [a\restriction_n]\phi$. In order to prove the desired result by induction on the number of PR rule applications, we thus have to

prove $[a\restriction_0]\phi$ and $\forall n \in \mathbb{N} : [a\restriction_n]\phi \vdash_{\mathsf{Rule}} [a\restriction_{n+1}]\phi$. $[a\restriction_0]\phi$ was given. Let a^i denote a sequence of a of length i, with $a^0 = \epsilon$. The premiss of the second conjunct can be rewritten using axiom (PRDL4) as follows:

$$\begin{array}{rcl}
[a\restriction_n]\phi & \leftrightarrow & [a\restriction_0]\phi \wedge [(a;a)\restriction_{n-1}]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_{n-1}]\phi \wedge [(a;a;a)\restriction_{n-2}]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_{n-1}]\phi \wedge [a\restriction_0][(a;a)\restriction_{n-2}]\phi \wedge [(a;a;a;a)\restriction_{n-3}]\phi \\
& \vdots & \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_{n-1}]\phi \wedge \ldots \wedge [a\restriction_0][(a^n)\restriction_0]\phi \wedge [(a;(a^n))\restriction_0]\phi.
\end{array}$$

So, in order to prove $[a\restriction_{n+1}]\phi$, we may assume — among other things — $[a\restriction_n]\phi$, $[(a;a)\restriction_{n-1}]\phi$, $[(a;a;a)\restriction_{n-2}]\phi$, …, $[(a;(a^n))\restriction_0]\phi$ (last conjunct of each line). Equivalently, we may thus assume the following.[20]

$$\bigwedge_i [(a;(a^i))\restriction_{n-i}]\phi \qquad \text{for } 0 \leq i \leq n. \tag{6.1}$$

The consequent, i.e. $[a\restriction_{n+1}]\phi$, can be rewritten using axiom (PRDL4) as below:

$$\begin{array}{rcl}
[a\restriction_{n+1}]\phi & \leftrightarrow & [a\restriction_0]\phi \wedge [(a;a)\restriction_n]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_n]\phi \wedge [(a;a;a)\restriction_{n-1}]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_n]\phi \wedge [a\restriction_0][(a;a)\restriction_{n-1}]\phi \wedge [(a;a;a;a)\restriction_{n-2}]\phi \\
& \vdots & \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_n]\phi \wedge \ldots \wedge [a\restriction_0][(a;(a^n))\restriction_0]\phi \wedge [(a;a;(a^n))\restriction_0]\phi.
\end{array} \tag{6.2}$$

As $[a\restriction_{n+1}]\phi$ is equivalent to all of the lines on the right-hand side of (6.2), we may prove any of these lines, in order to prove the desired result. As it turns out, it is easiest to prove the last line. The reason is that in this case, the last conjunct has a restriction parameter of 0. We can thus use Proposition 5.6 for sequential composition to prove this conjunct as follows:

1.	$[a\restriction_0]\phi$	assumption
2.	$[(a;a;(a^{n-1}))\restriction_0][a\restriction_0]\phi$	1, (GEN)
3.	$[(a;a;(a^{n-1});a)\restriction_0]\phi$	2, Proposition 5.6
4.	$[(a;a;(a^n))\restriction_0]\phi$	3, definition of a^i

Proving the other part of the last line of (6.2), i.e. $\bigwedge_i [a\restriction_0][(a;(a^i))\restriction_{n-i}]\phi$ for $0 \leq i \leq n$, can be done by applying (GEN) to each of the conjuncts of (6.1), yielding the desired result.

The important thing to note about this example is that the rewriting of formulas like $[a\restriction_n]\phi$ using (PRDL4), terminates. This is because the number of rewrite steps is restricted by n. If we did not have this restriction parameter, we might have the following variant of (PRDL4):

$$[c;\pi]\phi \leftrightarrow [c\restriction_0][\pi]\phi \wedge \bigwedge_\rho [apply(\rho, c;\pi)]\phi.$$[21]

[20] Note that $[a\restriction_0][(a^0)\restriction_n]\phi \leftrightarrow [a\restriction_0][\epsilon\restriction_n]\phi$ and $[a\restriction_0][\epsilon\restriction_n]\phi \leftrightarrow [a\restriction_0]\phi$, using axiom (PRDL3).

[21] We use the 0-restriction parameter here to distinguish between rule application and action execution, i.e. $[c;\pi]\phi$ is true, if and only if $[\pi]\phi$ is true after the execution of c and ϕ is true after the plans resulting from the application of the PR rules of the agent.

An attempt at proving $[a]\phi$ for an agent with the PR rule of Example 6.6 and this 'axiom', would however result in infinite regression:

$$\begin{array}{rcl} [a]\phi & \leftrightarrow & [a\restriction_0]\phi \wedge [a;a]\phi \\ & \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a]\phi \wedge [a;a;a]\phi \\ & \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a]\phi \wedge [a\restriction_0][a;a]\phi \wedge [a;a;a;a]\phi \\ & \vdots & \end{array}$$

In the example above, we have proven the desired result in our axiom system, using the key axiom (PRDL4). Another way to look at an agent with only the PR rule $a \rightsquigarrow a;a$, is by considering the language of plans that is 'generated' by this rule. By doing this, a much simpler proof can be obtained.

EXAMPLE 6.7
We take again the agent of Example 6.6, i.e. an agent with one PR rule $a \rightsquigarrow a;a$, and with $[a\restriction_0]\phi$. We want to prove again $\forall n \in \mathbb{N} : [a\restriction_n]\phi$. Taking into account the PR rule that is given and the initial plan a, one can conclude that the action sequences that can be executed by this agent, are sequences of a of an arbitrary length. Given this, one could instead prove $\forall n \in \mathbb{N}^+ : [a^n\restriction_0]\phi$, where $\mathbb{N}^+$ is the set of positive natural numbers.[22] We prove this by taking an arbitrary n and proving $[a^n\restriction_0]\phi$ for this n.

1.	$[a\restriction_0]\phi$	assumption
2.	$[a\restriction_0][a\restriction_0]\phi$	1, GEN
3.	$[a;a\restriction_0]\phi$	2, proposition 5.6
	$\vdots$	
	$[a^n\restriction_0]\phi$	

Obviously, this proof is much shorter than the proof of Example 6.6. It is, however, obtained through meta-reasoning about the PR rules of the agent. In the desired result $\forall n \in \mathbb{N}^+ : [a^n\restriction_0]\phi$, the restriction parameter is 0. The application of PR rules has thus in effect been eliminated from the expression in the object language.

Meta-reasoning could be done in this simple case: the PR rule actually generates the language of plans that can be represented by the simple regular expression a^*. PR rules in general however do not only generate languages that can be represented by regular expressions. In particular, rules of the form $p \rightsquigarrow \pi$, where p is an abstract plan, can be compared with parameterless recursive procedures (see also Section 7), which can in turn be linked to context-free programs [15, Chapter 9]. Furthermore, PR rules can have the form $\pi_h \rightsquigarrow \pi_b$, where the head is an arbitrary plan. It is thus not obvious that a meta-argument about the plans generated by the agent can be constructed in the general case. Investigations along these lines are however not within the scope of this paper and remain for future research.

In the next example, we will use Proposition 6.9 below, in the proof of which we use the following lemma.

LEMMA 6.8
Let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. The following is then derivable in the axiom system $\mathsf{AS}_{\mathsf{Rule}}$.

$$\vdash_{\mathsf{Rule}} [\pi\restriction_n]\phi \rightarrow [\pi\restriction_0]\phi.$$

[22] The result $\forall n \in \mathbb{N} : [a\restriction_n]\phi$ that we want to prove specifies that always at least one action a is executed: if $n = 0$, the required result is $[a\restriction_0]\phi$, which specifies the execution of a. The result does not require proving $[\epsilon\restriction_n]\phi$, which would be provable if we would assume ϕ to be valid.

PROOF. Let $c_i \in (\mathsf{BasicAction} \cup \mathsf{AbstractPlan})$ for $i \geq 1$ and let $\pi = c_1; \ldots; c_m$, with $m \geq 1$. Through repeated application of axiom (PRDL4), from left to right, then using (PRDL3) to get rid of $[\epsilon\restriction_n]$ and then using Proposition 5.6 for sequential composition with a 0 restriction parameter, we derive the desired result.

$$\begin{array}{lcl}
[\pi\restriction_n]\phi & \leftrightarrow & [c_1; \ldots; c_m\restriction_n]\phi \\
& \rightarrow & [c_1\restriction_0][c_2; \ldots; c_m\restriction_n]\phi \\
& \rightarrow & \ldots \\
& \rightarrow & [c_1\restriction_0][c_2\restriction_0] \ldots [c_m\restriction_0][\epsilon\restriction_n]\phi \\
& \rightarrow & [c_1\restriction_0][c_2\restriction_0] \ldots [c_m\restriction_0]\phi \\
& \rightarrow & [c_1; c_2\restriction_0][c_3\restriction_0] \ldots [c_m\restriction_0]\phi \\
& \rightarrow & \ldots \\
& \rightarrow & [c_1; \ldots; c_m\restriction_0]\phi \\
& \rightarrow & [\pi\restriction_0]\phi
\end{array}$$

∎

In the following proposition, we use some notation that we will first explain. The notation $(\mathsf{PRDL4})_i([\pi\restriction_n]\phi)$, with $0 \leq i \leq n$, denotes the formula that results from rewriting $[\pi\restriction_n]\phi$ using (PRDL4) from left to right, such that all restriction parameters are either 0 or i. Formulas of the form $[\epsilon\restriction_m]\phi$ are replaced by ϕ, using axiom (PRDL3). In this process, (PRDL4) may only be applied to a formula $[\pi\restriction_m]\phi$ if $m > i$.

Take, for example, the agent of Example 6.6 with $a \rightsquigarrow a; a$ as the only PR rule. The formula $(\mathsf{PRDL4})_3([a\restriction_5]\phi)$ then for example, denotes the formula $[a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_0]\phi \wedge [a\restriction_0][a; a\restriction_3]\phi \wedge [a; a; a\restriction_3]\phi$, which can be obtained by rewriting the formula $[a\restriction_5]\phi$ as below.

$$\begin{array}{lcl}
[a\restriction_5]\phi & \leftrightarrow & [a\restriction_0][\epsilon\restriction_5]\phi \wedge [a; a\restriction_4]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_4]\phi \wedge [a; a; a\restriction_3]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_0][\epsilon\restriction_4]\phi \wedge [a\restriction_0][a; a\restriction_3]\phi \wedge [a; a; a\restriction_3]\phi \\
& \leftrightarrow & [a\restriction_0]\phi \wedge [a\restriction_0][a\restriction_0]\phi \wedge [a\restriction_0][a; a\restriction_3]\phi \wedge [a; a; a\restriction_3]\phi
\end{array}$$

The idea is thus, that formulas of the form $[\pi\restriction_m]\phi$ are rewritten until formulas are obtained with i as the restriction parameter. A formula $[\pi\restriction_i]\phi$ may not be rewritten.

Any formula $[\pi\restriction_n]\phi$ can be rewritten into a formula $(\mathsf{PRDL4})_i([\pi\restriction_n]\phi)$ with $0 \leq i \leq n$. An application of (PRDL4) to a formula $[\pi\restriction_m]\phi$ yields two conjuncts (the second of which is again a conjunction). The first conjunct is smaller in plan size than $[\pi\restriction_m]\phi$.[23] Each conjunct of the second conjunct is smaller than $[\pi\restriction_m]\phi$ with respect to the restriction parameter. With each rewrite step, we thus have a decrease either in plan size or in size of the restriction parameter of each resulting conjunct. This can thus continue for each conjunct until either the plan size (minus the plan size of ϕ) is 0 or the non-zero restriction parameters are equal to i.

Another notation that we will use is $to0(\phi)$, denoting the formula that results from replacing all restriction parameters in ϕ by 0.

PROPOSITION 6.9 (restriction parameter)
Let $\mathsf{Rule} \subseteq \mathcal{R}$ be a finite set of PR rules. The following is then derivable in the axiom system $\mathsf{AS}_{\mathsf{Rule}}$:

$$\vdash_{\mathsf{Rule}} [\pi\restriction_n]\phi \rightarrow [\pi\restriction_i]\phi \text{ with } -1 \leq i \leq n$$

[23] The second element of $size(F)$, where F denotes the first conjunct, is smaller than the second element of $size([\pi\restriction_m]\phi$.

PROOF. If $i = -1$, the desired result follows immediately by axiom (PRDL1). We will now prove the result for $i \geq 0$.

1.	$[\pi\restriction_n]\phi \leftrightarrow (\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi)$	(PRDL4)
2.	$[\pi\restriction_i]\phi \leftrightarrow (\mathsf{PRDL4})_0([\pi\restriction_i]\phi)$	(PRDL4)
3.	$(\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi) \rightarrow to0((\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi))$	Lemma 6.8
4.	$to0((\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi)) \leftrightarrow (\mathsf{PRDL4})_0([\pi\restriction_i]\phi)$	syntactic equality
5.	$(\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi) \rightarrow (\mathsf{PRDL4})_0([\pi\restriction_i]\phi)$	$3, 4$
6.	$[\pi\restriction_n]\phi \rightarrow [\pi\restriction_i]\phi$	$1, 2, 5.$

Step 4 is justified, because both $(\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi)$ and $(\mathsf{PRDL4})_0([\pi\restriction_i]\phi)$ result from the same number of applications of (PRDL4) to $[\pi\restriction_n]\phi$ and $[\pi\restriction_i]\phi$ respectively. The latter two formulas are syntactically equal, except for the restriction parameter. The formulas $(\mathsf{PRDL4})_{n-i}([\pi\restriction_n]\phi)$ and $(\mathsf{PRDL4})_0([\pi\restriction_i]\phi)$ are thus also syntactically equal,[24] except for the restriction parameters, which are $n-i$ or 0 in the first case and 0 in the latter. Setting the restriction parameters of the first formula to 0, will thus give us equivalent formulas. ∎

EXAMPLE 6.10

We now consider an agent with two PR rules: $\mathsf{Rule} = \{a \rightsquigarrow a; a,\ a; a; a \rightsquigarrow b\}$ and we assume that $[a\restriction_0]\phi$ and $[b\restriction_0]\phi$. We want to prove $\forall n \in \mathbb{N} : [a\restriction_n]\phi$. Along similar lines of reasoning to those in Example 6.6, i.e. by using axiom (PRDL4) to rewrite $[a\restriction_n]\phi$, we can conclude that we may again use assumption (6.1) from Example 6.6: We have to prove the following, taking the 'last line' of the rewriting of $[a\restriction_{n+1}]\phi$ by (PRDL4).

$$\bigwedge_i [a\restriction_0][(a;(a^i))\restriction_{n-i}]\phi \qquad \text{for } 0 \leq i \leq n \tag{6.3}$$

$$\bigwedge_i [(b;(a^{i-2}))\restriction_{n-i}]\phi \qquad \text{for } 2 \leq i \leq n \tag{6.4}$$

$$[(a;a;(a^n))\restriction_0]\phi \tag{6.5}$$

The formulas (6.3) and (6.5) were proven in the example above, using assumption (6.1). We will prove (6.4) by proving $\bigwedge_i [(a^{i-2})\restriction_{n-i}]\phi$ and using (GEN) to derive the desired formula.

In the proof below, let $3 \leq i \leq n$ and let $0 \leq r \leq n$ in the first line and $0 \leq r \leq n-3$ in the second line.

1.	$\bigwedge_r [(a;(a^r))\restriction_{n-r}]\phi$	assumption (6.1)
2.	$\bigwedge_r [(a;(a^r))\restriction_{n-r-3}]\phi$	1, Proposition 6.9
3.	$\bigwedge_i [(a;(a^{i-3}))\restriction_{n-i}]\phi$	where $r = i-3$
4.	$\bigwedge_i [(a^{i-2})\restriction_{n-i}]\phi$	definition of a^i
5.	$\bigwedge_i [b\restriction_0][(a^{i-2})\restriction_{n-i}]\phi$	4, (GEN)
6.	$\bigwedge_i [b\restriction_0][(a^{i-2})\restriction_{n-i}]\phi \leftrightarrow \bigwedge_i [(b;(a^{i-2}))\restriction_{n-i}]\phi$	(PRDL4)
7.	$\bigwedge_i [(b;(a^{i-2}))\restriction_{n-i}]\phi$	$5, 6,$ (MP).

The above proves $[(b;(a^{i-2}))\restriction_{n-i}]\phi$ for $3 \leq i \leq n$. If $i = 2$, we need to prove $[b\restriction_{n-2}]\phi$. According to axiom (PRDL4), this is equivalent to proving $[b\restriction_0]\phi$.[25] This was given, so we are done.

In Section 6.1, we have presented an infinitary axiom system to prove the properties of non-restricted 3APL plans. As an infinitary axiom system is difficult to use, we have suggested using induction

[24] That is, modulo swapping of conjuncts.

[25] By (PRDL4) we have $[b\restriction_{n-2}]\phi \leftrightarrow [b\restriction_0][\epsilon\restriction_{n-2}]\phi$ and by (PRDL3): $[b\restriction_0][\epsilon\restriction_{n-2}]\phi \leftrightarrow [b\restriction_0]\phi$.

on the number of PR rule applications, i.e. on the restriction parameter, in an expression. Some examples have been worked out to illustrate this approach. As the examples show, it is doable (at least for the example cases) to use induction on the number of PR rule applications. It is however a fairly complicated undertaking. Future research will have to show whether this type of reasoning is amenable to some kind of automation, and what the limits of the approach are.

7 PR rules versus procedures

As stated in the introduction, the operational semantics of (parameterless) procedures is similar to that of PR rules. The operational semantics of a procedure $p \Leftarrow S$ where p is the procedure name and the statement S is the body of the procedure, can be defined by a transition $\langle p; S', \sigma\rangle \rightarrow \langle S; S', \sigma\rangle$, where S' is a statement. If we compare this semantics to the semantics of PR rules of Definition 3.7, we can see that both are so-called body-replacement semantics: if the head of a PR rule or the name of a procedure occur at the head of a statement that is to be executed, the head or the procedure name are replaced by the body of the rule or the procedure respectively.

Because of this similarity, one might think that techniques used for reasoning about procedures can be used to reason about PR rules. This however turns out not to be the case, due to the non-compositional semantics of the sequential composition operator in 3APL (see introduction to Section 4). In this section, we will elaborate on this issue by studying inference rules of Hoare logic for reasoning about procedures (see, for example, [7, 1] for a detailed explanation of Hoare logic). We will also show that reasoning by induction on the number of PR rule applications and reasoning about procedures using Hoare logic inference rules, although very different at first sight, actually do have similarities.

7.1 Reasoning about Procedures

Hoare logic is used for reasoning about programs. Inference rules are defined to derive so-called Hoare triples. A Hoare triple is of the form $\{\phi_1\}\ S\ \{\phi_2\}$ and intuitively means that if ϕ_1 holds, ϕ_2 will always hold after the execution of the statement S.[26] To reason about *non-recursive* procedures, the following inference rule can be defined for a procedure $p \Leftarrow S$ (for simplicity, we assume we only have one procedure) with procedure name p and body S.

$$\frac{\{\phi_1\}\ S\ \{\phi_2\}}{\{\phi_1\}\ p\ \{\phi_2\}}$$

The rule states that if we can prove that ϕ_2 holds after the execution of the body S of the procedure (assuming ϕ_1 holds before execution), we can infer that ϕ_2 holds after the procedure call p.

If the procedure $p \Leftarrow S$ is *recursive*, that is, if p is called in S, the rule above will still be sound, but a system with only this rule for reasoning about procedure calls will not be complete (see also [1]). An attempt at proving $\{\phi_1\}\ p\ \{\phi_2\}$ results in an infinite regression. The following rule [1], which is a variant of so-called Scott's induction rule (see for example [7]), is meant to overcome this difficulty.

DEFINITION 7.1 (Scott's induction rule)

$$\frac{\{\phi_1\}\ p\ \{\phi_2\} \vdash \{\phi_1\}\ S\ \{\phi_2\}}{\{\phi_1\}\ p\ \{\phi_2\}}$$

[26] The Hoare triple $\{\phi_1\}\ S\ \{\phi_2\}$ can be characterized in dynamic logic by the formula $\phi_1 \rightarrow [S]\phi_2$.

The rule states that if we can prove $\{\phi_1\}\ S\ \{\phi_2\}$ from the assumption that $\{\phi_1\}\ p\ \{\phi_2\}$, we can infer $\{\phi_1\}\ p\ \{\phi_2\}$. Using this rule for reasoning about procedure calls, a complete proof system can be obtained [1].[27]

In a proof of a property of a procedural program, the rule above is (often) used in combination with the following rule for sequential composition.

DEFINITION 7.2 (rule for sequential composition)

$$\frac{\{\phi_1\}\ S\ \{\phi_2\} \quad \{\phi_2\}\ S'\ \{\phi_3\}}{\{\phi_1\}\ S; S'\ \{\phi_3\}}$$

Consider for example a procedure $p \Leftarrow p$ and suppose we want to prove $\{\phi_1\}\ p; S\ \{\phi_3\}$ (p is non-terminating, so we should be able to prove this for any ϕ_1 and ϕ_3). We then have to prove $\{\phi_1\}\ p\ \{\phi_2\}$ and $\{\phi_2\}\ S\ \{\phi_3\}$ for some ϕ_2. If we take $\phi_2 = \mathbf{0}$, i.e. falsum, the second conjunct follows immediately. In proving $\{\phi_1\}\ p\ \{\mathbf{0}\}$, which we will refer to as H, we use Scott's induction rule and we thus have to prove H from the assumption H. This is immediate, concluding the proof.

The point of this example is the following. Using Scott's induction rule, we can prove properties of a procedure call p. If we want to prove a property of a statement involving the sequential composition of this procedure call and some other statement S, we can use properties proven about the procedure call (obtained using Scott's induction rule) and compose it with properties proven about S by means of the rule for sequential composition. In particular, this technique can be applied to, for example, a procedure $p \Leftarrow p; S$, where an assumption about p can be used to prove properties of $p; S$. Scott's induction rule for proving properties of procedure calls is thus most useful if used in combination with the rule for sequential composition.

7.1.1 Scott's induction rule for PR rules

A question one might ask, is whether a variant of Scott's induction rule can be used to reason about PR rules. Assuming one PR rule $\pi_h \rightsquigarrow \pi_b$, the following rule could be formulated.

$$\frac{\{\phi_1\}\ \pi_h\ \{\phi_2\} \vdash \{\phi_1\}\ \pi_b\ \{\phi_2\}}{\{\phi_1\}\ \pi_h\ \{\phi_2\}}$$

Assume for the moment that it is possible to use this rule to prove $\{\phi_1\}\ \pi_h\ \{\phi_2\}$ for some PR rule $\pi_h \rightsquigarrow \pi_b$ and properties ϕ_1 and ϕ_2. The question now is, whether the fact that we can prove $\{\phi_1\}\ \pi_h\ \{\phi_2\}$, will do us any good if we want to prove properties of more complex plans such as $\pi_h; \pi$.

Proving properties of $\pi_h; \pi$ based on properties proven of π_h, would have to be done using the rule for sequential composition. This rule is however not sound in the context of PR rules. In general, it is *not* the case that $\mathcal{O}(\pi_1; \pi_2)(\sigma) \subseteq \mathcal{O}(\pi_2)(\mathcal{O}(\pi_1)(\sigma))$ (see also the introduction to Section 4). Let $\Sigma_1 = \mathcal{O}(\pi_1)(\sigma)$ and $\Sigma_2 = \mathcal{O}(\pi_2)(\Sigma_1)$. If ϕ_2 holds in all states in Σ_1 (if ϕ_1 holds in σ), then ϕ_3 will hold in all states in Σ_2 by assumption. Let $\Sigma_3 = \mathcal{O}(\pi_1; \pi_2)(\sigma)$ and let $\sigma' \in \Sigma_3$, but $\sigma' \notin \Sigma_2$. Then we may not conclude that ϕ_3 will hold in σ' and therefore the rule is not sound.

The fact that we can prove $\{\phi_1\}\ \pi_h\ \{\phi_2\}$, will thus not help if we want to prove properties of a plan like $\pi_h; \pi$, because we do not have a rule for sequential composition. In particular, the assumption

[27] Note that this is a proof rule for deriving partial correctness specifications, a Hoare triple $\{\phi_1\}\ p\ \{\phi_2\}$ meaning that *if* p terminates, ϕ_2 will hold after execution of p (provided that p is executed in a state in which ϕ_1 holds). If p does not terminate, anything is derivable for p. The rule cannot be used to prove termination of p.

$\{\phi_1\}\ \pi_h\ \{\phi_2\}$ will not help to prove $\{\phi_1\}\ \pi_b\ \{\phi_2\}$, even if $\pi_b = \pi_h; \pi$. It is thus not clear whether it should be possible in the general case to prove $\{\phi_1\}\ \pi_b\ \{\phi_2\}$ from the assumption $\{\phi_1\}\ \pi_h\ \{\phi_2\}$. Moreover, the rule above is not sound for agents with more than one PR rule. It is then in general *not* the case that $\mathcal{O}(\pi_b)(\sigma) = \mathcal{O}(\pi_h)(\sigma)$, rather $\mathcal{O}(\pi_b)(\sigma) \subseteq \mathcal{O}(\pi_h)(\sigma)$. Therefore, we may not conclude $\{\phi_1\}\ \pi_h\ \{\phi_2\}$ from a proof of $\{\phi_1\}\ \pi_b\ \{\phi_2\}$.

7.2 Induction

In Section 7.1 we argued that, although the operational semantics of PR rules and procedure calls are very similar, we cannot use Scott's induction rule, which is used for reasoning about procedure calls, to reason about PR rules. Our solution to the issue of reasoning about PR rules as presented in this paper, is to do induction on the number of PR rule applications. In this section, we will elaborate on why Scott's induction rule is called an *induction* rule and by doing this, we will see that induction on the number of PR rule applications and induction as used in Scott's induction rule, have strong similarities.

At first sight, it does not look like using Scott's induction rule involves doing induction, because we do not see formulas parameterized with natural numbers n and $n+1$. To see why the rule actually *is* an induction rule, we first rephrase the rule of Definition 7.1 and adopt notation used by De Bakker [7]. Ω is used to denote a non-terminating statement (similar to the `fail` statement mentioned in the proof of Theorem 5.7). The first element of a tuple $\langle \ldots \mid \ldots \rangle$ is used to indicate the procedures, in the presence of which the formula of the second element should hold.

$$\frac{\{\phi_1\}\ \Omega\ \{\phi_2\} \qquad \langle \mid \{\phi_1\}\ p\ \{\phi_2\} \vdash \{\phi_1\}\ S\ \{\phi_2\}\rangle}{\langle p \Leftarrow S \mid \{\phi_1\}\ p\ \{\phi_2\}\rangle} \tag{7.1}$$

The rule above is an instantiation of a more general version of this rule for multiple procedures [7]. The first antecedent is derived from this general rule, but could be omitted in this form: Ω is a non-terminating statement and therefore the triple $\{\phi_1\}\ \Omega\ \{\phi_2\}$ is valid for any ϕ_1, ϕ_2. We will however not eliminate it for the purpose of comparing this rule with reasoning about PR rules.

Now, consider a procedure $p \Leftarrow S$ and let S^n be defined as follows: $S^0 = \Omega$ and $S^{n+1} = S[S^n/p]$, where $S[S^n/p]$ means that every occurrence of p in S is replaced by S^n. If for example $S = p; S'$, then $S^1 = S^0; S' = \Omega; S'$, $S^2 = S^1; S' = (\Omega; S'); S'$, etc.

Using this substitution construction, we can define the meaning $\mathcal{M}$ of a procedure $p \Leftarrow S$ in the following way (see Apt [1]): $\mathcal{M}(p) = \bigcup_{n=0}^{\infty} \mathcal{M}(S^n)$. From this, we can conclude that $\langle p \Leftarrow S \mid \{\phi_1\}\ p\ \{\phi_2\}\rangle$ is true iff $\forall n : \langle p \Leftarrow S^n \mid \{\phi_1\}\ p\ \{\phi_2\}\rangle$ is true [1]. Therefore, the induction rule above is equivalent to the following rule.

$$\frac{\{\phi_1\}\ \Omega\ \{\phi_2\} \qquad \langle \mid \{\phi_1\}\ p\ \{\phi_2\} \vdash \{\phi_1\}\ S\ \{\phi_2\}\rangle}{\forall n : \langle p \Leftarrow S^n \mid \{\phi_1\}\ p\ \{\phi_2\}\rangle} \tag{7.2}$$

The meaning of a procedure call p of a procedure $p \Leftarrow S$ is equivalent with the meaning of S. More generally, the meaning of a statement S' in which a call to procedure $p \Leftarrow S$ occurs, is equivalent to the meaning of the statement $S'[S/p]$, i.e. the statement S' in which all occurrences of p are replaced with S (see [7]). Therefore, we may replace p with S^n in rule (7.2) and we may replace occurrences of p in S with S^n. We have, by definition, that $S[S^n/p] = S^{n+1}$, yielding the following equivalent rule.[28]

$$\frac{\{\phi_1\}\ \Omega\ \{\phi_2\} \qquad \forall n : (\{\phi_1\}\ S^n\ \{\phi_2\} \vdash \{\phi_1\}\ S^{n+1}\ \{\phi_2\})}{\forall n : \{\phi_1\}\ S^n\ \{\phi_2\}} \tag{7.3}$$

[28] We omit the procedure declaration $p \Leftarrow S$, because there are no occurrences of p in either S^n or S^{n+1} by definition.

This rule, which is equivalent to Scott's induction rule, demonstrates clearly why Scott's induction rule is called an *induction* rule. The idea of proving properties of a 3APL agent of the form $\forall n : \vdash_{\mathsf{Rule}} [\pi \restriction_n]\phi$ by induction on n, is that we prove $[\pi \restriction_0]\phi$ and $\forall n : ([\pi \restriction_n]\phi \vdash_{\mathsf{Rule}} [\pi \restriction_{n+1}]\phi)$. The similarity between the two approaches is thus that induction on respectively the number of procedure calls and PR rule applications is done (implicitly or explicitly).

The important difference however is that the statement S in rule (7.3) corresponds with the body of a procedure p in the equivalent rule (7.1). The plan π on the other hand does not correspond with the body of a PR rule, but rather refers to the initial plan of the agent. Related to this is the fact that rule (7.3) or the equivalent rule (7.1) can be used in combination with the rule for sequential composition, as explained in Section 7.1. In the case of using induction to reason about 3APL plans, this is impossible.

Concluding, the general idea of doing induction on the number of PR rule applications is less obscure than one might have thought at first sight, because of the similarity with the standard Scott's induction rule. The way in which induction can be used to prove properties of plans or programs, however differs between the two approaches due to the non-compositional semantics of the sequential composition operator in plans, as a result of the presence of PR rules.

8 Conclusion

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 analysed by structural induction, which means that standard propositional dynamic logic cannot be used to reason about 3APL plans. Instead, we proposed a logic of restricted plans with sound and complete axiomatization. We also showed that this logic can be extended to a logic for non-restricted plans. This however results in an infinitary axiom system. We suggested that a possible way of dealing with the infinitary nature of the axiom system, is reasoning by induction on the restriction parameter. We showed some examples of how this could be done. Finally, we discussed the relation between PR rules and procedures. In particular, we argued that there is a similarity between the use of Scott's induction rule for reasoning about procedures, and the use of induction on the number of PR rules applications for reasoning about PR rules.

Concluding, 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. The resulting infinitary axiom system is nevertheless more of theoretical than practical importance. Future research will have to show whether reasoning by doing induction on the number of PR rule applications is amenable to some kind of automation, working towards an extension of these results to a more practical setting. Another important line for future research is the investigation of the relation between term rewriting systems and PR rules, and between PR rules and formal language theory. We hope that those investigations will lead to the definition of interesting subclasses of PR rules that *can* be analysed by structural induction.

References

[1] K. R. Apt. Ten years of Hoare's logic: A survey - part I. *ACM Transactions of Programming Languages and Systems*, **3**, 431–483, 1981.

[2] 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), pp. 409–416, Melbourne, 2003.

[3] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, MA, 1987.

[4] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal representation for BDI agent systems. In *Programming Multiagent Systems, Second International Workshop (ProMAS'04)*, volume 3346 of *Lecture Notes in Artificial Intelligence*, pp. 44–65. Springer, Berlin, 2005.

[5] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence, 42*, 213–261, 1990.

[6] 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 *Lecture Notes in Artificial Intelligence*, pp. 111–130. Springer, Berlin, 2004.

[7] J. de Bakker. *Mathematical Theory of Program Correctness*. Series in Computer Science. Prentice-Hall International, London, 1980.

[8] B. Drabble, J. Dalton, and A. Tate. Repairing plans on the fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, 1997.

[9] E.M.Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, Cambridge, MA, 2000.

[10] 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 *Lecture Notes in Artificial Intelligence*, pp. 18–49. Springer, Berlin, 2004.

[11] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 677–682, 1987.

[12] G. d. Giacomo, Y. Lespérance, and H. Levesque. *ConGolog*, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, **121**, 109–169, 2000.

[13] K. J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, **45**, 173–228, 1990.

[14] D. Harel. *First-Order Dynamic Logic*. Volume 68 of *Lecture Notes in Computer Science*. Springer, Berlin, 1979.

[15] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press, Cambridge, MA, 2000.

[16] M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gordon-Spears, eds. *Formal Approaches to Agent-Based Systems (Proceedings of FAABS'02)*, Volume 2699 of *Lecture Notes in Artificial Intelligence*, Berlin, 2003. Springer.

[17] 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**, 357–401, 1999.

[18] 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.

[19] B. Nebel and J. Koehler. Plan reuse versus plan generation: a theoretical and empirical analysis. *Artificial Intelligence*, **76**, 427–454, 1995.

[20] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[21] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: a BDI reasoning engine. In *Multi-Agent Programming: Languages, Platforms and Applications*. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, eds, Springer, Berlin, 2005.

[22] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Agents Breaking Away (LNAI 1038)*, W. van der Velde and J. Perram, eds, pp. 42–55. Springer-Verlag, 1996.

[23] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, J. Allen, R. Fikes, and E. Sandewall, eds, pp. 473–484. Morgan Kaufmann, 1991.

[24] 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 *Lecture Notes in Artificial Intelligence*, Berlin, 2001. Springer.

[25] R.E.Fikes and N.J.Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, 189–208, 1971.

[26] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, **60**, 51–92, 1993.

[27] W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. An integrated modal approach to rational agents. In *Foundations of Rational Agency*, Applied Logic Series 14, M. Wooldridge and A. S. Rao, editors, pp. 133–168. Kluwer, Dordrecht, 1998.

[28] R. P. van der Krogt and M. M. de Weerdt. Plan repair as an extension of planning. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS'05)*, pp. 161–170, 2005.

[29] R. P. van der Krogt and M. M. de Weerdt. Plan repair using a plan library. In *Proceedings of the Belgium-Dutch Conference on Artificial Intelligence (BNAIC'05)*, pp. 254–259. BNVKI, 2005.

[30] P. van Emde Boas. The connection between modal logic and algorithmic logics. In *Mathematical foundations of computer science 1978*, volume 64 of *Lecture Notes in Computer Science*, pp. 1–15. Springer, Berlin, 1978.

[31] M. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of declarative goals in agent programming. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Declarative agent languages and technologies II: second international workshop (DALT'04)*, volume 3476 of *Lecture Notes in Artificial Intelligence*, pp. 1–18, 2005.

[32] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Semantics of declarative goals in agent programming. In *Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS'05)*, pp. 133–140, Utrecht, 2005.

[33] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Subgoal semantics in agent programming. In *Progress in Artificial Intelligence: 12th Portuguese Conference on Artificial Intelligence (EPIA'05)* C. Bento, A. Cardoso and G. Dias, eds, pp. 548–559. Volume 3808 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2005.

[34] M. B. van Riemsdijk, F. S. de Boer, and J.-J. Ch. Meyer. Dynamic logic for plan revision in intelligent agents. In J. A. Leite and P. Torroni, editors, *Computational logic in multi-agent systems: fifth international workshop (CLIMA'04)*, Volume 3487 of *Lecture Notes in Artificial Intelligence*, pp. 16–32, 2005.

[35] M. B. van Riemsdijk, J.-J. Ch. Meyer, and F. S. de Boer. Semantics of plan revision in intelligent agents. In *Proceedings of the 10th International Conference on Algebraic Methodology And Software Technology (AMAST04)*, C. Rattray, S. Maharaj, and C. Shankland, editors, Volume 3116 of *Lecture Notes in Computer Science*, pp. 426–442. Springer-Verlag, 2004.

[36] M. B. van Riemsdijk, J.-J. Ch. Meyer and F. S. de Boer. Semantics of Plan Revision in Intelligent Agents. C. Rattray, S. Maharaj and C.Shankland, eds. *Theoretical Computer Science*, **351**, 240–257, 2006. Speical issue of *Algebraic Methodology and Software Technology (AMAST'04)*.

[37] 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 Systesm (AAMAS'03)*, pp.393–400. Melbourne, 2003.

[38] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.

[39] M. Wooldridge. Agent-based software engineering. *IEEE Proceedings Software Engineering*, **144**, 26–37, 1997.

Received 26 April 2005