A Compositional Semantics of Plan Revision in Intelligent Agents

M. Birna van Riemsdijk John-Jules Ch. Meyer

Utrecht University, Department of Information and Computing Sciences
P.O. Box 80.089, 3508 TB Utrecht
The Netherlands
{birna, jj}@cs.uu.nl

Abstract. This paper revolves around the so-called plan revision rules of the agent programming language 3APL. These rules can be viewed as a generalization of procedures. This generalization however results in the semantics of programs of the 3APL language no longer being compositional. This gives rise to problems when trying to define a proof system for the language. In this paper we define a restricted version of plan revision rules which extends procedures, but which does have a compositional semantics, as we will formally show.

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 [1]. Autonomy means that an agent encapsulates its state and makes decisions about what to do based on this state, without the direct intervention of humans or others. Agents are situated in some environment which can change during the execution of the agent. This requires *flexible* problem solving behavior, i.e., the agent should be able to respond adequately to changes in its environment. Programming flexible computing entities is not a trivial task. Consider for example a standard procedural language. The assumption in these languages is that the environment does not change while some procedure is executing. If problems do occur during the execution of a procedure, the program might throw an exception and terminate (see also [2]). This works well for many applications, but we need something more if change is the norm and not the exception.

A philosophical view that is well recognized in the AI literature is that rational behavior can be explained in terms of the concepts of *beliefs*, *goals* and *plans*. [3,4,5]. This view has been taken up within the AI community in the sense that it might be possible to *program* flexible, autonomous agents *using* these concepts. 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. When programming in terms of these concepts, beliefs can be compared to the program state,

plans can be compared to statements, i.e., plans constitute the procedural part of the agent, and goals can be viewed as the (desired) postconditions of executing the statement or plan. Through executing a plan, the world and therefore the beliefs reflecting the world will change and this execution should have the desired result, i.e., achievement of goals.

This view has been adopted by the designers of the agent programming language *3APL*[1] [6,7], which is a well-known language in the agent programming community. The dynamic parts of a 3APL agent thus consist of a set of beliefs, a plan[2] and a set of goals. A plan can consist of sequences of so-called basic actions, which change the beliefs[3] if executed. To provide for the possibility of programming flexible behavior, so-called *plan revision* rules were added to the language. These rules can be compared to procedures in the sense that they have a head, which is comparable with the procedure name, and a body, which is a plan in the case of 3APL and a statement in the case of procedural languages.

The operational meaning of plan revision rules is similar to that of procedures: if the procedure name or head is encountered in a statement or plan, this name or head is replaced by the body of the procedure or rule, respectively (see [8] for the operational semantics of procedure calls). The difference however is that the head in a plan revision rule can be *any* plan (or statement) and not just a procedure name. In procedural languages it is furthermore usually assumed that procedure names are distinct. In 3APL however, it is possible that multiple rules are applicable at the same time. This provides for very general and flexible plan revision capabilities, which is a distinguishing feature of 3APL compared to other agent programming languages [9,10,11].

As argued, we consider these general plan revision capabilities to be an essential part of agenthood. The introduction of these capabilities now gives rise to interesting issues concerning the *semantics of plan execution*, which we will be concerned with in this paper.

The main issue which arises with the introduction of plan revision rules, is the issue of *compositionality* of semantics of plans. For standard procedural languages [8, Chapter 5], the semantics of statements is compositional, i.e., the semantics of a composed statement can be defined in terms of the semantics of the parts of which it is composed. The semantics of plans however, which can be viewed as the statements of 3APL, is *not* compositional. The reason for this lies in the presence of plan revision rules, which we will elaborate on in section 3.2.

The fact that the semantics of plans is not compositional, gives rise to problems when trying to reason about 3APL programs. A proof system for a programming language will typically contain rules by means of which properties of the entire program can be proven by proving properties of the parts of which the program is composed. Since the semantics of 3APL plans is not compositional,

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

[2] In the original version this was a set of plans.

[3] A change in the environment is a possible "side effect" of the execution of a basic action.

this is problematic in the case of 3APL. One way of trying to approach this problem is by defining a specialized logic for 3APL which tries to circumvent the issue, as was done in [12,13]. The resulting logic is however non-standard and can be difficult to use, which will be explained in more detail in section 3.3.

The approach we take in this paper, is to try to *restrict* the allowed plan revision rules, such that the semantics of plans becomes compositional in some sense. It is not immediately obvious what kind of restriction would yield the desired result. In this paper, we propose such a restriction and prove that the semantics of plans in that case is compositional.

The outline of the paper is as follows. In section 2, we present the syntax and semantics of a simplified version of 3APL. It is important to note that we use a simplified version, in order to be able to focus on the issue of compositionality of plans. In particular, we do not include a model of the environment in the semantics, since this is not necessary for investigating the compositionality issue. In section 3 we elaborate on the issue of compositionality and explain why the semantics of full 3APL is not compositional. In section 4 we present our proposal for a restricted version of plan revision rules, and prove that the semantics of plans is compositional, given this restriction on plan revision rules. This paper aims to be a first step towards a compositional proof system for 3APL. Investigating automated theorem proving and providing accompanying tool support for this is left for future research.

2 3APL

2.1 Syntax

Below, we define belief bases and plans. A belief base is a set of propositional formulas. A plan is a sequence of basic actions. Basic actions can be executed, resulting in a change to the beliefs of the agent.

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

Definition 11 *(belief bases)* Assume a propositional language $\mathcal{L}$ with typical formula p and the connectives $\wedge$ and $\neg$ with the usual meaning. Then the set of belief bases Σ with typical element σ is defined to be $\wp(\mathcal{L})$.[4]

Definition 12 *(plans)* Assume that a set BasicAction with typical element a is given. The set of plans Plan with typical element π is then defined as follows.

$$\pi ::= a \mid \pi_1; \pi_2$$

We use ϵ to denote the empty plan and identify $\epsilon; \pi$ and $\pi; \epsilon$ with π.

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.

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

Definition 13 *(plan revision rules)* The set of plan revision rules $\mathcal{R}$ is defined as follows: $\mathcal{R} = \{\pi_h \rightsquigarrow \pi_b \mid \pi_h, \pi_b \in \mathsf{Plan}, \pi_h \neq \epsilon\}$.[5]

Take for example a plan $a; b$ where a and b are basic actions, and a plan revision 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 plan revision rule yielding a new plan c, which can in turn be executed.

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 14 *(3APL agent)* A 3APL agent $\mathcal{A}$ is a tuple $\langle \sigma_0, \pi_0, \mathsf{PR}, \mathcal{T} \rangle$ where $\sigma_0 \in \Sigma$, $\pi_0 \in \mathsf{Plan}$, $\mathsf{PR} \subseteq \mathcal{R}$ is a finite set of plan revision rules and $\mathcal{T} : (\mathsf{BasicAction} \times \Sigma) \rightarrow \Sigma$ is a partial function, expressing how belief bases are updated through basic action execution.

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 15 *(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. If $\langle \sigma_0, \pi_0, \mathsf{PR}, \mathcal{T} \rangle$ is an agent, then $\langle \pi_0, \sigma_0 \rangle$ is the initial configuration of the agent.

2.2 Semantics

The semantics of a programming language can be defined as a function taking a statement and a state, and yielding the set of states resulting from executing the 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 [8] for details on this subject).

The operational semantics of a language is usually defined using transition systems [14]. A transition system for a programming language consists of a set of axioms and derivation rules for deriving transitions for this language. A transition is a transformation of one configuration into another and it corresponds to a single computation step. Let $\mathcal{A}$ be a 3APL agent with a set of plan revision rules PR, belief update function $\mathcal{T}$, and let $\mathsf{BasicAction}$ be its set of basic actions. Below, we give the transition system $\mathsf{Trans}_{\mathcal{A}}$ for our simplified 3APL language,

[5] In [6], plan revision rules were defined to have a guard, i.e., rules were of the form $\pi_h \mid \phi \rightsquigarrow \pi_b$, where ϕ is a condition on the belief base. 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.

which is based on the system given in [6]. 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 16 *(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 17 *(rule application)* Let $\rho : \pi_h \rightsquigarrow \pi_b \in \mathsf{PR}$.

$$\langle \pi_h \bullet \pi, \sigma \rangle \rightarrow_{apply} \langle \pi_b \bullet \pi, \sigma \rangle$$

Using the transition system, individual transitions can be derived for a 3APL agent. These transitions can be put in sequel, yielding transition sequences, which are typically denoted by θ. 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 18 *(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 19 *(function for calculating computation sequences)* Let $x_i \in \{exec, apply\}$ for $1 \leq i \leq m$. The function $\mathcal{C}^{\mathcal{A}} : (\mathsf{Plan} \times \Sigma) \rightarrow \wp(\Sigma^+)$ is then as defined below.

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

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 110 *(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 in the sequel omit the superscript $\mathcal{A}$ to functions as defined above, for reasons of presentation.

3 3APL and Non-Compositionality

Before we go into discussing why the semantics of 3APL plans is not compositional, we consider compositionality of standard procedural languages.

3.1 Compositionality of Procedural Languages

The semantics of standard procedural languages such as described in [8, Chapter 5] are compositional. Informally, a semantics for a programming language is compositional if the semantics of a composed program can be defined in terms of the semantics of the parts of which it is composed. To be more specific, the meaning of a composed program $S_1; S_2$ should be definable in terms of the meaning of S_1 and S_2, for the semantics to be compositional.

A semantics can be defined directly in a compositional way, in which case the semantics is often termed a denotational semantics [8]. Alternatively, a semantics can be *defined* in a *non*-compositional way, such as an operational semantics defined using computation sequences, while it still *satisfies a compositionality property*. In this paper, we focus on the latter case. It turns out that the operational semantics for a procedural language such as discussed in [8, Chapter 5] satisfies such a compositionality property, while the operational semantics of 3APL of definition 110 does not. All results and definitions with respect to procedural languages which we refer to in section 3, can be found in [8, Chapter 5].

An operational semantics of a procedural language can be defined analogously to the operational semantics of 3APL of definition 110, where plans are statements and belief bases are states. Both operational semantics are defined in a non-compositional way, since they do not use the structure of the plan or statement to define its semantics. Nevertheless, the operational semantics of a procedural language does satisfy a compositionality property, i.e., the following holds: $\mathcal{O}(S_1; S_2)(\sigma) = \mathcal{O}(S_2)(\mathcal{O}(S_1)(\sigma))$, where S_1 and S_2 are statements. This property specifies that the set of states possibly resulting from the execution of a composed statement $S_1; S_2$ in σ is equal to the set of states resulting from the execution of S_2 in all states resulting from the execution of S_1 in σ.

3.2 Non-Compositionality of 3APL

While the presented compositionality property is termed "natural" in [8, Chapter 5], it is *not* satisfied by the operational semantics of 3APL, i.e., it is not the case that $\mathcal{O}(\pi_1;\pi_2)(\sigma) = \mathcal{O}(\pi_2)(\mathcal{O}(\pi_1)(\sigma))$ always holds. The reason for this lies in the presence of plan revision rules. Take for example an agent with one plan revision rule $a; b \rightsquigarrow c$. Let σ_{ab} and σ_c be the belief bases resulting from the execution of actions a followed by b, and c in σ, respectively. We then have that $\mathcal{O}(a;b)(\sigma) = \{\sigma_{ab}, \sigma_c\}$, i.e., the agent can either execute the actions a and b one after the other, or it can apply the plan revision rule and then execute c.

If the semantics of 3APL plans would have been compositional, we would also have that $\mathcal{O}(b)(\mathcal{O}(a)(\sigma)) = \{\sigma_{ab}, \sigma_c\}$. This is however not the case, since $\mathcal{O}(b)(\mathcal{O}(a)(\sigma)) = \{\sigma_{ab}\}$.[6] This stems from the fact that if one "breaks" the composed plan $a; b$ in two, one can no longer apply the plan revision rule $a; b \rightsquigarrow c$, because this rule can only be applied if the composed plan $a; b$ is considered. The set of belief bases $\mathcal{O}(a)(\sigma)$ only contains those resulting from the execution of a. The action b is then executed on those belief bases, yielding $\mathcal{O}(b)(\mathcal{O}(a)(\sigma))$. The result thus does not contain σ_c.

3.3 Reasoning about 3APL

This non-compositionality property of 3APL plans gives rise to problems when trying to define a proof system for reasoning about 3APL plans. In standard procedural languages, the following proof rule is part of any Hoare logic for such a language [8], where p, p' and q are assertions.

$$\frac{\{p\}\ S_1\ \{p'\} \qquad \{p'\}\ S_2\ \{q\}}{\{p\}\ S_1; S_2\ \{q\}} \tag{3.1}$$

This rule specifies that one can reason about a composed program by proving properties of the parts of which it is composed. The soundness of this rule depends on the fact that $\mathcal{O}(S_1; S_2)(\sigma) = \mathcal{O}(S_2)(\mathcal{O}(S_1)(\sigma))$. Because this property does not hold for 3APL plans, a similar rule for 3APL would not be sound (see also the discussion in [13]). Nevertheless, one would still want to reason about composed 3APL plans.

In [13],[7] we have presented a specialized dynamic logic for this purpose. In that paper, we define a logic for reasoning about 3APL plans in which we can restrict the number of plan revision rule applications allowed to occur during the execution of the plan. Based on this logic, we define a logic for reasoning about 3APL plans in general. The resulting complete proof system however contains an infinitary proof rule, i.e., a rule with an infinite number of premises. In some cases, induction can be used to prove the premises of this rule. These induction

[6] Note that $\mathcal{O}(b)(\mathcal{O}(a)(\sigma)) \subseteq \mathcal{O}(a;b)(\sigma)$.

[7] Parts of [13] were published in [12].

proofs are however quite involved, and it is not yet clear whether these can somehow be automated, etc.

Another possible approach for reasoning about 3APL plans has been suggested in [15,16]. In that paper, we define a denotational (i.e., compositional) semantics for a 3APL *meta*-language. This meta-language is relatively standard, as it is essentially a non-deterministic language with a `while` construct. It was suggested that it might by possible to reason about this meta-language, rather than about 3APL plans directly.

While the two discussed papers aim at reasoning about full 3APL, we take a different approach in this paper. Here, we investigate whether we can somehow *restrict* plan revision rules, such that the semantics of plans becomes compositional (in some sense). The idea is that given such a compositional semantics, it will be possible to come up with a more standard and easy to use proof system for 3APL.

4 Compositional 3APL

One obvious candidate for a restricted version of plan revision rules is the restriction to rules with an atomic head, i.e., to rules of the form $a \rightsquigarrow \pi$. These rules are very similar to procedures, apart from the fact that an action a could either be transformed using a plan revision rule, *or* executed directly. In contrast with actions, procedure variables cannot be executed, i.e., they can only be replaced by the body of a procedure. It is easy to see that a semantics for 3APL with only these plan revision rules would be compositional.

However, this kind of plan revision rules would capture very little of the general plan revision capabilities of the non-restricted rules. The challenge is thus to find a less restrictive kind of plan revision rules, which would still satisfy the desired compositionality property. Finding such a restricted kind of plan revision rules is non-trivial. We discuss the line of reasoning by which it can be obtained in section 4.1. In section 4.2, we present and explain the theorem that expresses that the proposed restriction on plan revision rules indeed establishes (some form of) compositionality. Finally, in section 5, we briefly address the issue of reasoning about 3APL with restricted plan revision rules, and point to directions for future research regarding this issue.

4.1 Restricted Plan Revision Rules

The restriction to plan revision rules that we propose is given in definition 111 below, and can be understood by trying to get to the essence of the compositionality problem arising from non-restricted plan revision rules.

First, we have to observe that the general kind of compositionality as specified in section 3.1 for procedural languages is in general not obtainable for 3APL, if the set of plan revision rules contains a rule with a non-atomic head. The property specifies that the semantics of a composed plan (or program)

should be definable in terms of the parts of which it is composed. The property however does not specify how a composed plan should be broken down into parts. That is, for a plan to be compositional in the general sense, compositionality should hold, no matter how the plan is decomposed. Consider for example the plan $a; b; c$. It should then be the case that $\mathcal{O}(a; b; c)(\sigma) = \mathcal{O}(c)(\mathcal{O}(a; b)(\sigma)) = \mathcal{O}(b; c)(\mathcal{O}(a)(\sigma))$, i.e., the compositionality property should hold, no matter whether the plan is decomposed into $a; b$ and c, or a and $b; c$.

If a set of plan revision rules however contains a rule with a non-atomic head, it is always possible to come up with a plan (and belief base and belief update function) for which this property does not hold. This plan should contain the head of the plan revision rule. If the decomposition of the plan is then chosen such that it "breaks" this occurrence of the head of the rule in the plan, the compositionality property in general does not hold for this decomposition. This is because the plan revision rule can in that case not be applied when calculating the result of the operational semantic function. Consider for example the plan revision rule $a; b \rightsquigarrow c$ and the plan $a; b; c$. If the plan is decomposed into a and $b; c$, the rule cannot be applied and thus $\mathcal{O}(a; b; c)(\sigma) = \mathcal{O}(b; c)(\mathcal{O}(a)(\sigma))$ does not always hold.

The question is now which kind of compositionality *can* be obtained for 3APL. We have established that being allowed to decompose a composed plan into arbitrary parts for a definition of compositionality gives rise to problems in the case of 3APL. That is, the standard definition of compositionality will always be problematic if we want to consider plan revision rules with a non-atomic head. Since we want our restriction on plan revision rules to allow at least some form of non-atomicity (because otherwise we would essentially be considering procedures), we have to come up with another definition of compositionality if we want to make any progress.

The idea that we propose is essentially to take the operational meaning of a plan as the basis for a compositionality property. When executing a plan π, either the first action of π is executed, or an applicable plan revision rule is applied. In the first case, π has to be of the form $a; \pi_r$[8], and in the latter case of the form $\pi_h; \pi'_r$, given an applicable plan revision rule of the form $\pi_h \rightsquigarrow \pi_b$. Taking this into account, we are, broadly speaking, looking for a restriction to plan revision rules which allows us to decompose π into a and π_r, or π_h and π'_r. To be more specific, it should be possible to execute a and then consider π_r separately, or to apply the specified plan revision rule and then consider the body of the rule π_b and the rest of the plan, i.e., π'_r, separately. That is, we are after something like the following compositionality property:[9]

$$\mathcal{O}(\pi)(\sigma) = \mathcal{O}(\pi_r)(\mathcal{O}(a)(\sigma)) \cup \mathcal{O}(\pi'_r)(\mathcal{O}(\pi_b)(\sigma)). \tag{4.1}$$

In order to come up with a restriction on plan revision rules that gives us such a property, we have to understand why this property does not always hold in

[8] The subscript r here indicates that π_r is the *rest* of the plan π.

[9] The property that will be proven in section 4.2 differs slightly, as it takes into account the existence of multiple applicable plan revision rules.

the presence of non-restricted plan revision rules. Essentially, what this property specifies is that we can separate the semantics of certain prefixes of the plan π (i.e., a and π_h), from the semantics of the rest of π.

A case in which this is *not* possible, is the following. Consider a plan of the form $\pi_h; \pi'_h; \pi$, and plan revision rules of the form $\pi_h \rightsquigarrow \pi_b$ and $\pi_b; \pi'_h \rightsquigarrow \pi'_b$. We can apply the first rule to this plan, yielding $\pi_b; \pi'_h; \pi$. If the semantics of the plan would be compositional in the sense of (4.1), it should now be possible to consider the semantics of $\pi'_h; \pi$, i.e., the "rest" of the plan, separately. Given the second plan revision rule however, this is not possible: if we separate $\pi_b; \pi'_h; \pi$ into π_b and $\pi'_h; \pi$, we can no longer apply the second plan revision rule, whereas we *can* apply the rule if the plan is considered in its composed form. The semantics of the plan $\pi_h; \pi'_h; \pi$ is thus not compositional, given the two plan revision rules.

This argument is similar to the explanation of why the general notion of compositionality does not hold for 3APL. Contrary to the general case however, we can in the case of compositionality as defined in (4.1), specify a restriction to plan revision rules that prevents this problem from occurring. The restriction will thus allow us to consider the semantics of π'_r (see (4.1)) separately from the semantics of π_b, thereby establishing compositionality property (4.1).

As explained, if there is a plan revision rule of the form $\pi_h \rightsquigarrow \pi_b$, a plan revision rule with a head of the form $\pi_b; \pi'_h$ is problematic. A restriction one could thus consider, is the restriction that if there is a rule of the form $\pi_h \rightsquigarrow \pi_b$, there should not also be a rule of the form $\pi_b; \pi'_h \rightsquigarrow \pi'_b$, i.e., the body of a rule cannot be equal to the prefix of the head of another rule. This restriction however does not do the trick completely. The reason has to do with the fact that actions from a plan of the form $\pi_b; \pi'_h$ can be executed.

Consider for example a plan $a_1; a_2; b_1; b_2$ and plan revision rules $a_1; a_2 \rightsquigarrow c_1; c_2$ and $c_2; b_1 \rightsquigarrow c_3$. The head of the second rule does not have the form $c_1; c_2; \pi$, i.e., the body of the first rule is not equal to the prefix of the head of another rule. Therefore, according to the suggested restriction, this rule is allowed. We can apply the first rule to the plan, yielding $c_1; c_2; b_1; b_2$. If the compositionality property holds, we should now be able to consider the semantics of $b_1; b_2$ separately. Suppose the action c_1 is executed, resulting in the plan $c_2; b_1; b_2$. Considering the second plan revision rule, we observe that this rule is applicable to this plan. This is however only the case if we consider this plan in its composed form. If we separate the semantics of $b_1; b_2$ as specified by the compositionality property (4.1), we cannot apply the rule. Given the plan $a_1; a_2; b_1; b_2$ and the two plan revision rules, the compositionality property thus does not hold.

The solution to this problem is to adapt the suggested restriction which considers the body of a rule in relation with the prefix of the head of another rule, to a restriction which consider the *suffix* of the body of a rule in relation with the prefix of the head of another rule. The restriction should thus specify that the suffix of the body of a rule cannot be equal to the prefix of the head of another rule. Under that restriction, the second rule of the example discussed above would not be allowed, and the compositionality property (4.1) would hold.

This restriction on plan revision rules is specified formally below. The fact that under this restriction, the property (4.1) (or a slight variation thereof) holds, is formally shown in section 4.2.

Definition 111 *(restricted plan revision rules)* Let PR be a set of plan revision rules. Let suff be a function taking a plan and yielding all its suffixes, and let pref be a function taking a plan and yielding all its strict prefixes.[10] We say that PR is *restricted* iff the following holds:

$$\forall \rho \in \mathsf{PR} : (\rho : \pi_h \rightsquigarrow \pi_b) : \neg \exists \rho' \in \mathsf{PR} : (\rho' : \pi'_h \rightsquigarrow \pi'_b) : \big(\mathsf{suff}(\pi_b) \cap \mathsf{pref}(\pi'_h)\big) \neq \emptyset.$$

The fact that we define pref as yielding *strict* prefixes allows the suffix of the body of a plan revision rule to be exactly equal to the head of another rule. This does not violate the compositionality property, and it results in restricted plan revision rules being a superset of rules with an atomic head. Otherwise, a rule $b \rightsquigarrow c$, for example, would not be allowed if there is also a rule $a \rightsquigarrow a; b$, since b, i.e., the suffix of the latter rule, would then by definition be equal to the prefix of the head of the first rule.

4.2 Compositionality Theorem

The theorem expressing the compositionality property that holds for plans under a restricted set of plan revision rules, is given below. It is similar to property (4.1) specified in section 4.1, except that we take into account the existence of multiple applicable plan revision rules. A plan π can thus be decomposed into a and π_r (where π is of the form $a; \pi$), or into π_h^ρ and π_r^ρ (where π is of the form $\pi_h^\rho; \pi_r^\rho$) for any applicable plan revision rule ρ of the form $\pi_h^\rho \rightsquigarrow \pi_b^\rho$.

Theorem 11 *(compositionality of semantics of plans)* Let $\mathcal{A}$ be an agent with a restricted set of plan revision rules PR. Let ρ range over the set of rules from PR that are applicable to the plan π, and let π be of the form $\pi_h^\rho; \pi_r^\rho$ for an applicable rule ρ of the form $\pi_h^\rho \rightsquigarrow \pi_b^\rho$. Further, let a be the first action of π, i.e., let π be of the form $a; \pi_r$. We then have for all $\pi \neq \epsilon$ and σ:

$$\mathcal{O}(\pi)(\sigma) \quad = \quad \mathcal{O}(\pi_r)(\mathcal{O}(a)(\sigma)) \ \cup \ \bigcup_\rho \mathcal{O}(\pi_r^\rho)(\mathcal{O}(\pi_b^\rho)(\sigma)).$$

In order to prove this theorem, we use lemma 11 below. This lemma, broadly speaking, specifies that for a plan of the form $\pi_h; \pi$, the following is the case: after application of a plan revision rule of the form $\pi_h \rightsquigarrow \pi_b$, yielding the plan $\pi_b; \pi$, it will always be the case that π_b is executed entirely, before π is executed. Because of this, the semantics of π_b and of π can be considered separately, which is the core of our compositionality theorem.

[10] The plan a is for example a strict prefix of $a; b$, but the plan $a; b$ is not.

Lemma 11 Let $\mathcal{A}$ be an agent with a restricted set of plan revision rules PR, and let $\pi_h \rightsquigarrow \pi_b \in \mathsf{PR}$. We then have that any transition sequence $\langle \pi_b; \pi, \sigma \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma' \rangle$ has the form[11]

$$\langle \pi_b; \pi, \sigma \rangle \rightarrow \ldots \rightarrow \langle \pi, \sigma'' \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma' \rangle$$

such that each configuration in the first part of the sequence, i.e., in $\langle \pi_b; \pi, \sigma \rangle \rightarrow \ldots \rightarrow \langle \pi, \sigma'' \rangle = \theta$, has the form $\langle \pi_i; \pi, \sigma_i \rangle$. That is, π is always the suffix of the plan of the agent in each configuration of θ.

In the proof of this lemma, we use the notion of a plan π' *being suffix* in π with respect to some set of plan revision rules. A plan π' is suffix in π, if π is the suffix of π', i.e., if π' is of the form $\pi_{pre}; \pi$. Further, π_{pre} should be a concatenation of suffixes of the bodies of the relevant set of plan revision rules.

Definition 112 *(suffix in π)* Let PR be a set of plan revision rules. Let suf_i with $1 \leq i \leq n$ denote plans that are equal to the suffix of the body of a rule in PR, i.e., for each suf_i there is a rule in PR of the form $\pi_h \rightsquigarrow \pi_r; suf_i$. We say that *a plan π' is suffix in π* with respect to PR, iff π' is of the form $suf_1; \ldots; suf_n; \pi$, and the length of $suf_1; \ldots; suf_n$ is greater than 0.

The idea is that, given a plan of the form $\pi_b; \pi$ which is suffix in π by definition[12], this property is preserved until the plan is of the form π. If this is the case, we have that π is always the (strict) suffix of the plan of each configuration, until the plan equals π. We thus use the preservation of this property to prove lemma 11 (see below).

We need the fact that the part of the plan occurring before π is a sequence of suffixes, in order to prove that π is preserved as the suffix of the plan.[13] The reason is, that if this is the case, we know by the fact that our plan revision rules are restricted, that there cannot occur a rule application which transforms π, thereby violating our requirement that π remains the suffix of the plan of the agent, until the plan becomes equal to π. If a plan is of the form $suf_1; \ldots; suf_n; \pi$, where each suf_i denotes a plan that is equal to the suffix of the body of a plan revision rule, we know that any plan revision rule will only modify a prefix of suf_1, because the plan revision rules are restricted. There cannot be a rule with a head of the form $suf_1; \pi_h$, because this would violate the requirement of restricted plan revision rules.

Proof of lemma 11: Let $\mathcal{A}$ be an agent with a restricted set of plan revision rules PR. Let $\langle \pi_1, \sigma \rangle \rightarrow \langle \pi_2, \sigma' \rangle$ be a transition of $\mathcal{A}$. First, we show that if π_1 is suffix in π (with respect to PR), it has to be the case that π_2 is suffix in π, or that $\pi_2 = \pi$.

---

[11] In this lemma we omit the labels of transitions, for reasons of presentation.

[12] That is, if π_b is the body of a plan revision rule.

[13] Note that we use the term suffix to refer to suffixes of the plans of the bodies of plan revision rules, and to refer to the suffix of the plan in a configuration.

Assume that π_1 is suffix in π, i.e., let $\pi_1 = suf_1; \ldots; suf_n; \pi$. If $\pi = \epsilon$, the result is immediate. Otherwise, the proof is as follows. A transition from $\langle \pi_1, \sigma \rangle$ results either from the execution of an action, or from the application of an applicable rule.

Let $suf_1 = a; suf_1'$. If action a is executed, π_2 is of the form $suf_1'; \ldots; suf_n; \pi$. If $suf_1', \ldots, suf_n$ are ϵ, we have that $\pi_2 = \pi$. Otherwise, we have that π_2 is suffix in π.

Let $\rho : \pi_h \rightsquigarrow \pi_b$ be a rule from PR that is applicable to π_1. Then it must be the case that π_1 is of the form $\pi_h; \pi_r$. By the fact that PR is restricted, we have that there is not a rule ρ' of the form $suf_1; \pi' \rightsquigarrow \pi_b'$, i.e., such that suf_1, which is the suffix of the body of a rule, is the prefix of the head of ρ'. Given that ρ is applicable to π_1, it must thus be the case that π_h is a prefix of suf_1, i.e., that suf_1 is of the form $\pi_h; \pi''$. Applying ρ to π_1 thus yields a plan of the form $\pi_b; \pi''; suf_2; \ldots; suf_n; \pi$. Since both π_b and π'' are suffixes of the bodies of rules in PR, we have that π_2 is suffix in π.

We have to show that any transition sequence θ of the form $\langle \pi_b; \pi, \sigma \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma' \rangle$ has a prefix θ' such that π is always a suffix of the plan in each configuration of θ'. Let π_2 be the plan of the second configuration of θ. We have that $\pi_b; \pi$ is suffix in π. Therefore, it must be the case that π_2 is also suffix in π, or that $\pi_2 = \pi$. In the latter case, we have the desired result. In the former case, we have that π is a suffix of π_2, in which case the first two configuration may form a prefix of θ'. Let π_3 be the plan of the third configuration of θ. If π_2 is suffix in π, it has to be the case that π_3 is suffix in π, or that $\pi_3 = \pi$. In the latter case, we are done. In the former case, the first three configurations may form a prefix of θ'. This line of reasoning can be continued. Since θ is a finite sequence, it has to be the case that at some point a configuration of the form $\langle \pi, \sigma'' \rangle$ is reached. This yields the desired result. □

Proof of theorem 11: We have to show the following:

$$\sigma' \in \mathcal{O}(\pi)(\sigma) \quad \Leftrightarrow \quad \sigma' \in \mathcal{O}(\pi_r)(\mathcal{O}(a)(\sigma)) \ \cup \ \bigcup_{\rho} \mathcal{O}(\pi_r^{\rho})(\mathcal{O}(\pi_b^{\rho})(\sigma)).$$

($\Leftarrow$) Follows in a straightforward way from the definitions.
($\Rightarrow$) Let n be the number of plan revision rules applicable to π, where $\pi_h^{\rho_i}$ and $\pi_b^{\rho_i}$ respectively denote the head and body of rule ρ_i. We then have to show:

$$\begin{aligned} \sigma' \in \mathcal{O}(\pi)(\sigma) \Rightarrow \sigma' &\in \mathcal{O}(\pi_r^{\rho_1})(\mathcal{O}(\pi_b^{\rho_1})(\sigma)) \text{ or} \\ &\vdots \\ \sigma' &\in \mathcal{O}(\pi_r^{\rho_n})(\mathcal{O}(\pi_b^{\rho_n})(\sigma)) \text{ or} \\ \sigma' &\in \mathcal{O}(\pi_r)(\mathcal{O}(a)(\sigma)). \end{aligned}$$

If $\sigma' \in \mathcal{O}(\pi)(\sigma)$, then there is a transition sequence of the form

$$\langle \pi, \sigma \rangle \rightarrow_x \ldots \rightarrow_x \langle \epsilon, \sigma' \rangle$$

i.e., if $\pi_h^\rho \rightsquigarrow \pi_b^\rho$ is an arbitrary rule ρ that is applicable to π, where $\pi = \pi_h^\rho; \pi_r^\rho$, there are transition sequences of the form

$$\langle \pi_h^\rho; \pi_r^\rho, \sigma \rangle \rightarrow_{apply} \langle \pi_b^\rho; \pi_r^\rho, \sigma \rangle \rightarrow_x \ldots \rightarrow_x \langle \epsilon, \sigma' \rangle \tag{4.2}$$

or, if $\pi = a; \pi_r$, of the form

$$\langle a; \pi_r, \sigma \rangle \rightarrow_{exec} \langle \pi_r, \sigma'' \rangle \rightarrow_x \ldots \rightarrow_x \langle \epsilon, \sigma' \rangle. \tag{4.3}$$

In case σ' has resulted from a transition sequence of form (4.2), we prove

$$\sigma' \in \mathcal{O}(\pi_r^\rho)(\mathcal{O}(\pi_b^\rho)(\sigma)). \tag{4.4}$$

In case σ' has resulted from a transition sequence of form (4.3), we prove

$$\sigma' \in \mathcal{O}(\pi_r)(\mathcal{O}(a)(\sigma)). \tag{4.5}$$

Assume σ' has resulted from a transition sequence of form (4.2). We then have to prove (4.4), i.e., we have to prove that there is a belief base $\sigma'' \in \mathcal{O}(\pi_b^\rho)(\sigma)$, such that $\sigma' \in \mathcal{O}(\pi_r^\rho)(\sigma'')$. That is, we have to prove that there are transition sequences of the form $\langle \pi_b^\rho, \sigma \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma'' \rangle$, and of the form $\langle \pi_r^\rho, \sigma'' \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma' \rangle$.

By definitions 16 and 17, we have that if $\langle \pi_1; \pi_2, \sigma \rangle \rightarrow \langle \pi_1'; \pi_2, \sigma' \rangle$ is a transition for arbitrary plans π_1 and π_2, then $\langle \pi_1, \sigma \rangle \rightarrow \langle \pi_1', \sigma' \rangle$ is also a transition. By lemma 11, we have that there is a prefix of (4.2) of the form $\langle \pi_h^\rho; \pi_r^\rho, \sigma \rangle \rightarrow \langle \pi_b^\rho; \pi_r^\rho, \sigma \rangle \rightarrow \ldots \rightarrow \langle \pi_r^\rho, \sigma'' \rangle$, such that the plan of each configuration in this sequence is of the form $\pi_i; \pi$. From this we can conclude the desired result, i.e., that there are transition sequences of the form $\langle \pi_b^\rho, \sigma \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma'' \rangle$, and of the form $\langle \pi_r^\rho, \sigma'' \rangle \rightarrow \ldots \rightarrow \langle \epsilon, \sigma' \rangle$.

Assume σ' has resulted from a transition sequence of form (4.3). Then proving (4.5) is analogous to proving (4.4), except that we do not need lemma 11. □

5 Conclusion and Future Work

As argued, an important reason for defining a variant of 3APL with a compositional semantics, is that it is more likely that it will be possible to come up with a more standard and easy to use proof system for such a language. A natural starting point for such an effort is the definition of a proof rule for sequential composition, analogous to rule (3.1), as specified below (we use the notation of theorem 11).

$$\frac{\{p\}\ a\ \{p'\} \quad \{p'\}\ \pi_r\ \{q\} \qquad \bigwedge_\rho \big(\{p\}\ \pi_b^\rho\ \{p'\} \text{ and } \{p'\}\ \pi_r^\rho\ \{q\}\big)}{\{p\}\ \pi\ \{q\}} \tag{5.1}$$

The soundness proof of this rule is analogous to the soundness proof of rule (3.1) [8, Chapter 2], but using theorem 11 instead of $\mathcal{O}(S_1; S_2)(\sigma) = \mathcal{O}(S_2)(\mathcal{O}(S_1)(\sigma))$. A complete proof system for compositional 3APL would however also need an induction rule. We conjecture that it will be possible to define an analogue of Scott's induction rule [8, Chapter 5] which is used for proving properties of recursive procedures, for reasoning about plans in the context of plan revision rules. Investigating this is however left for future research.

References

1. Wooldridge, M.: Agent-based software engineering. IEEE Proceedings Software Engineering **144**(1) (1997) 26–37
2. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of the Art. In Ciancarini, P., Wooldridge, M., eds.: First Int. Workshop on Agent-Oriented Software Engineering. Volume 1957. Springer-Verlag, Berlin (2001) 1–28
3. Bratman, M.E.: Intention, plans, and practical reason. Harvard University Press, Massachusetts (1987)
4. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann (1991) 473–484
5. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence **42** (1990) 213–261
6. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.J.Ch.: Agent programming in 3APL. Int. J. of Autonomous Agents and Multi-Agent Systems **2**(4) (1999) 357–401
7. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.J.Ch.: A programming language for cognitive agents: goal directed 3APL. In: Programming multiagent systems, first international workshop (ProMAS'03). Volume 3067 of LNAI. Springer, Berlin (2004) 111–130
8. de Bakker, J.: Mathematical Theory of Program Correctness. Series in Computer Science. Prentice-Hall International, London (1980)
9. Giacomo, G.d., Lespérance, Y., Levesque, H.: *ConGolog*, a Concurrent Programming Language Based on the Situation Calculus. Artificial Intelligence **121**(1-2) (2000) 109–169
10. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In van der Velde, W., Perram, J., eds.: Agents Breaking Away (LNAI 1038), Springer-Verlag (1996) 42–55
11. Shoham, Y.: Agent-oriented programming. Artificial Intelligence **60** (1993) 51–92
12. van Riemsdijk, M.B., de Boer, F.S., Meyer, J.J.Ch.: Dynamic logic for plan revision in intelligent agents. In Leite, J.A., Torroni, P., eds.: Computational logic in multiagent systems: fifth international workshop (CLIMA'04). Volume 3487 of LNAI. (2005) 16–32
13. van Riemsdijk, M.B., de Boer, F.S., Meyer, J.J.Ch.: Dynamic logic for plan revision in intelligent agents. Technical Report UU-CS-2005-013, Utrecht University, Institute of Information and Computing Sciences (2005) To appear in Journal of Logic and Computation.
14. Plotkin, G.D.: A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus (1981)
15. van Riemsdijk, M.B., Meyer, J.J.Ch., de Boer, F.S.: Semantics of plan revision in intelligent agents. In Rattray, C., Maharaj, S., Shankland, C., eds.: Proceedings of the 10th International Conference on Algebraic Methodology And Software Technology (AMAST04). Volume 3116 of LNCS., Springer-Verlag (2004) 426–442
16. van Riemsdijk, M.B., Meyer, J.J.Ch., de Boer, F.S.: Semantics of plan revision in intelligent agents. Theoretical Computer Science **351**(2) (2006) 240–257 Special issue of Algebraic Methodology and Software Technology (AMAST'04).