Plan Generation and Plan Execution in Agent Programming

M. Birna van Riemsdijk and Mehdi Dastani

Institute of Information and Computing Sciences
Utrecht University
The Netherlands
{birna, mehdi}@cs.uu.nl

Abstract. *This paper presents two approaches for generating and executing the plans of cognitive agents. They can be used to define the semantics of programming languages for cognitive agents. The first approach generates plans before executing them while the second approach interleaves the generation and execution of plans. Both approaches are presented formally and their relation is investigated.*

1 Introduction

Various programming languages have been proposed to implement cognitive agents [14,2,8,6,9,12,5,7,11]. These languages provide data structures to represent the agent's mental attitudes such as beliefs, goals and plans. Beliefs describe the state of the world the agent is in, goals describe the state the agent wants to reach and plans are the means to achieve these goals.

Most of these programming languages can be viewed as inspired in some way by the Procedural Reasoning System (PRS) [6]. This system was proposed as an alternative to the traditional planning systems [13], in which plans to get from a certain state to a goal state are constructed by reasoning about the results of primitive actions. PRS and most of today's cognitive agent programming languages, by contrast, use a library of pre-specified plans.[1] The goals for the achievement of which these plans can be selected, are part of the plan specification. Further, plans might not consist of primitive actions only, but they can also contain subgoals. If a subgoal is encountered during the execution of a plan, a plan for achieving this subgoal should be selected from the plan library, after which it can be executed. An agent can for example have the plan to take the bus into town, to achieve the subgoal of having bought a birthday cake, and then to eat the cake.[2] This subgoal of buying a birthday cake will have to be fulfilled by selecting and executing in turn an appropriate plan of for example which shops to go to, paying for the cake, etc., before the agent can execute

[1] The language ConGolog [7], in which the agent reasons about the result of the execution of its actions, is an exception.

[2] Assuming that both taking the bus into town and eating cake are primitive actions that can be executed directly.

the action of eating the cake. Plans containing subgoals are called *partial* plans, while plans containing only primitive actions are called *total*.

An important advantage of PRS and similar systems over traditional planning systems is that they do not require search through potentially large search spaces. A disadvantage of PRS-like systems has to do with the fact that most of these systems allow for multiple plans to be executed concurrently, i.e., the agent may pursue multiple goals simultaneously. These plans can conflict, as they, for example, can require the same resources. In PRS-like systems, in which plans for subgoals are selected during execution of the plan, it is difficult to predict whether plans will conflict. If a plan containing subgoals is selected, it is not yet known how the subgoals of this plan will be achieved. It is therefore difficult to assess whether this plan will conflict with other plans of the agent.

One way to approach this issue, is to use a representation of plans that contains information that can be used to detect possible conflicts among plans, as proposed by Thangarajah et al. [16,15]. Once these conflicts are detected, plans can be scheduled in such a way that conflicts do not occur during execution of the plans.

In this paper, we take a slightly different approach. That is, in order to be able to compare an approach in which information about conflicting plans is taken into account with an approach of *plan execution* in the PRS style, we take an operational approach to the former, which we call *plan generation*. The idea of plan generation is to use pre-specified partial plans to generate total plans offline, i.e., before the plans are executed. Since conflicts among plans generally depend on the primitive actions within the plans, the generation of total plans provides for the possibility to check whether plans are conflicting. We assume that a specification of conflicts among plans is given, e.g., in a way comparable with the work of Thangarajah et al.

In order to compare plan generation with plan execution, we first introduce a framework for plan generation (Section 2). This framework defines how non-conflicting sets of plans can be generated on the basis of a plan library (i.e., rules for selecting plans to achieve (sub)goals), a set of top-level goals, and a set of initial partial plans. These definitions are inspired by default logic. In default logic, various so-called extensions, which consist of consistent sets of first-order formulas, can be derived on the basis of possibly conflicting default rules, and an initial set of facts. The fact that default rules might conflict, gives rise to the possibility of deriving multiple extensions on the basis of a single default theory. We adapt the notion of extension as used in default logic, to the context of conflicting plans. An extension then consists of a set of non-conflicting plans. The idea of adapting the notion of extension as used in default logic to the context of plans, is inspired by the BOID framework [2]. It was however not worked out in detail in the cited paper.

The language we use as an example of a PRS style framework, is a simplified version of the cognitive agent programming language 3APL [8,3], and is presented in Section 3. We assume that a specification of conflicts among plans is given. Ways of specifying conflicts have been investigated in the literature (see,

e.g., [16]), and further research along these lines is beyond the scope of this paper. We show in Section 4 that, for any total plan in an extension of a so-called plan generation agent, there is a corresponding initial plan in the execution setting, which has the same semantics. If one would assume that in an offline plan generation context, a single extension is chosen for execution, one could say that the behavior of a plan generation agent is "included" in the behavior of a plan execution agent. This is intuitive, since the incorporation of a notion of conflict among plans *restricts* the set of plans which can be executed concurrently.

2 Plan Generation

In this section, we present a framework for plan generation that is based on [2]. In that paper, a non-standard approach to planning is taken, in which rules are used to specify which plan can be adopted for a certain goal. This is in contrast with planning from first principles, in which action specifications are taken as the basis, and a sequence of actions is sought that realizes a certain goal state according to the action specifications, given an initial situation. In [2] and in the current paper, it is the job of the agent *programmer* to specify which (composed) plan (or plan recipe) is appropriate for which goal.

Throughout this paper, we assume a language of propositional logic $\mathcal{L}$ with negation and conjunction, with typical element ϕ. The symbol $\models$ will be used to denote the standard entailment relation for $\mathcal{L}$.

Below, we define the language of plans. A plan is a sequence of basic actions and $achieve(\phi)$ statements, the latter representing that the goal ϕ is to be achieved. In correspondence with the semantics of 3APL, basic actions change an agents beliefs when executed. This will be defined formally in Section 3. One could add a test statement and non-deterministic choice, but we leave these out for reasons of simplicity. A total plan is a plan containing only basic actions.

Definition 1 *(plans)* Let BasicAction with typical element a be a set of basic actions and let $\phi \in \mathcal{L}$. The set of plans Plan with typical element π is then defined as follows.

$$\pi ::= a \mid achieve(\phi) \mid \pi_1; \pi_2$$

The set of total plans TotalPlan is the subset of Plan containing no $achieve(\phi)$ statements. We use ϵ to denote the empty plan and identify $\epsilon; \pi$ and $\pi; \epsilon$ with π.

Before we define the notion of an agent, we define the rules that represent which plan can be adopted to achieve a certain goal. These plan generation rules have a propositional formula as the head, representing the goal, and a plan as the body. In principle, plan generation rules can be extended to include a belief condition in the head, indicating that the plan in the body can be adopted if the agent has a certain goal *and* a certain belief. The belief condition could then be viewed as the precondition of the plan. For reasons of simplicity, we however define rules as having only a condition on goals.

Definition 2 *(plan generation rule)* The set of plan generation rules $\mathcal{R}_{\mathsf{PG}}$ is defined as follows: $\mathcal{R}_{\mathsf{PG}} = \{\phi \Rightarrow \pi \mid \phi \in \mathcal{L}, \pi \in \mathsf{Plan}\}$.

A plan generation agent is a tuple consisting of a belief base, a goal base, a plan base and a rule base. The belief base and goal base are consistent. The rule base consists of a set of plan generation rules and may not contain multiple rules for the same goal. This prevents that multiple plans for the same goal can be adopted, which could be considered undesirable. The plans base contains the initial set of plans of the agent.

Definition 3 *(plan generation agent)* A plan generation agent[3], typically denoted by $\mathcal{A}$, is a tuple $\langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ where $\sigma \subseteq \mathcal{L}$ is the belief base, $\gamma \subseteq \mathcal{L}$ is the goal base, $\Pi \subseteq \mathsf{Plan}$ is the plan base and $\mathsf{PG} \subseteq \mathcal{R}_{\mathsf{PG}}$ is a set of rules. Further, $\sigma \not\models \bot$ and $\gamma \not\models \bot$ and all sets σ, γ, Π and PG are finite. Finally, PG does not contain multiple rules with an equivalent head, i.e., if $\phi \Rightarrow \pi \in \mathsf{PG}$, there is not a rule $\phi' \Rightarrow \pi' \in \mathsf{PG}$ such that $\phi \equiv \phi'$.

When generating plans, we want to take into account conflicts, for example with respect to resources, that may arise among plans. For this, we assume a notion of coherency of plans. A plan π being coherent with a set of plans Π will be denoted by $\mathit{coherent}(\pi, \Pi)$. We assume that once a (partial) plan is incoherent with a set of plans, this plan cannot become coherent again by refining the plan, i.e., by replacing a subgoal with a more concrete plan.

We are now in a position to define how a coherent set of plans is generated on the basis of an agent $\langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$. A natural way in which to define this plan generation process, is an approach inspired by default logic. In default logic, consistent sets of formulas or *extensions* are generated on the basis of a possibly conflicting set of default rules, and a set of formulas representing factual world knowledge. Here, we generate sets of coherent plans on the basis of an initial set of plans Π, a goal base γ, and a set of plan generation rules PG.

The idea is that we take the plan base Π of the agent, which may contain partial plans, as the starting point. These partial plans in Π are refined by means of applying plan generation rules from PG. If $\pi_1; \mathit{achieve}(\phi); \pi_2$ is a plan in Π and $\phi \Rightarrow \pi$ is a rule in PG, then this rule can be applied, yielding the plan $\pi_1; \pi; \pi_2$. This process can continue, until total plans are obtained. Further, a plan generation rule $\phi \Rightarrow \pi$ can be applied if ϕ follows from the goal base γ. In that case, a new plan π is added to the existing set of plans, which can in turn be refined through rule applications.

The plans that are generated in this way should however be mutually coherent. A plan can thus only be added to the existing set of plans through refinement or plan addition, if this plan is coherent with already existing ones. Different choices of which plan to refine or to add may thus have different outcomes in terms of the resulting set of coherent plans: the addition of a plan may prevent the addition of other plans that are incoherent with this plan.

[3] In this section we take the term "agent" to mean "plan generation agent".

Differing from [2], we define the notion of an extension in the context of plans through the notion of a *process*. This is based on the concept of a process as used in [1] to define extensions in the context of default logic. A process is a sequence of sets of plans, such that each consecutive set is obtained from the previous by applying a plan generation rule. A process can formally be defined in terms of a transition system which is a set of transition rules that indicate the transitions between consecutive sets of plans.

Given a set of plans E_i and an agent $\langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$, a rule $\phi \Rightarrow \pi \in \mathsf{PG}$ can be applied if ϕ follows from γ. The plan π is then added to E_i, that is, if $\pi \notin E_i$ and $\mathit{coherent}(\pi, E_i)$. This rule can also be applied if there is a plan of the form $\pi_1; \mathit{achieve}(\phi); \pi_2$ in E_i.[4] In that case, the plan $\pi_1; \pi; \pi_2$ is added to E_i, again only if the plan is not already in E_i and it is coherent with E_i. One could also remove the original plan $\pi_1; \mathit{achieve}(\phi); \pi_2$ from E_i, but addition of the refined plan is more in line with the definition of processes and extensions in default logic. It would be more useful if a plan of the form $\pi_1; \mathit{achieve}(\phi); \pi_2$ could be refined by a rule $\phi' \Rightarrow \pi$ if $\phi \equiv \phi'$, but we omit this extra clause to simplify our definitions. The first element of a process of an agent is the plan base Π of the agent.

Definition 4 *(process)* Let $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ be an agent. A sequence of sets $E_0, \ldots, E_n$ with $E_i \subseteq \mathsf{Plan}$ is a process of $\mathcal{A}$ iff $E_0 = \Pi$ and it holds for all E_i with $0 \leq i \leq n-1$ that $E_i \rightarrow E_{i+1}$ is a transition that can be derived in the transition system below. Let $\phi \Rightarrow \pi \in \mathsf{PG}$ be a plan generation rule. The transition rule for *plan addition* is then defined as follows:

$$\frac{\gamma \models \phi \qquad \pi \notin E \qquad \mathit{coherent}(\pi, E)}{E \rightarrow E'}$$

where $E' = E \cup \{\pi\}$. The transition rule for *plan refinement* is defined as follows:

$$\frac{\pi_1; \mathit{achieve}(\phi); \pi_2 \in E \qquad \pi_1; \pi; \pi_2 \notin E \qquad \mathit{coherent}(\pi_1; \pi; \pi_2, E)}{E \rightarrow E'}$$

where $\pi_1, \pi_2 \in \mathsf{Plan}$ and $E' = E \cup \{\pi_1; \pi; \pi_2\}$.

We assume that the plan generation rules of an agent are such that no infinite processes can be constructed on the basis of the corresponding transition system.

The notion of an extension is defined in terms of the notion of a closed process. A process is closed iff no rules are applicable to the last element of the process. This is formalized in the definitions below. Note that not all processes are closed. A closed process can be viewed as a process that has terminated, i.e., there are no transitions possible from the last element in the process. It is however the case that we assume that any process can become a closed process.

[4] Note that, for example, a plan $\mathit{achieve}(\phi)$ is also of this form, as π_1 and π_2 can be the empty plan ϵ (see Definition 1).

Definition 5 *(applicability)* A plan generation rule $\phi \Rightarrow \pi$ is applicable to a set $E \subseteq \mathsf{Plan}$ iff a transition $E \rightarrow E'$ can be derived in the transition system above on the basis of this rule.

Definition 6 *(closed process)* A process $E_0, \ldots, E_n$ of an agent $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ is closed iff there is not a plan generation rule $\delta \in \mathsf{PG}$ such that δ is applicable to E_n.

Definition 7 *(extension)* A set $E \subseteq \mathsf{Plan}$ is an extension of $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ iff there is a closed process $E_0, \ldots, E_n$ of $\mathcal{A}$ such that $E = E_n$.

The execution of a plan generation agent is as follows. An extension of the agent is generated. This extension is a coherent set of partial and total plans. The total plans can then be executed according to the semantics of execution of basic actions as will be provided in Section 3.

3 Plan Execution

In this section, we present a variant of the agent programming language 3APL, which suits our purpose of comparing the language with the plan generation framework of the previous section. An important component of 3APL agents that we need in this paper, is the so-called plan revision rules which have a plan as the head and as the body. During execution of a plan, a plan revision rule can be used to replace a prefix of the plan, which is identical to the head of the rule, by the plan in the body. If the agent for example executes a plan $a; b; c$ and has a plan revision rule $a; b \Rightarrow d$, it can apply this rule, yielding the plan $d; c$.

Here we do not need the general plan revision rules that can have a composed plan as the head. We only need rules with statements of the form $achieve(\phi)$ as the head and a plan as the body.

Definition 8 *(plan revision rule)* The set of plan revision rules $\mathcal{R}_{\mathsf{PR}}$ is defined as follows: $\mathcal{R}_{\mathsf{PR}} = \{achieve(\phi) \Rightarrow \pi \mid \phi \in \mathcal{L}, \pi \in \mathsf{Plan}\}$.

An agent in this context is similar to the plan generation agent of Definition 3, with a rule base consisting of a set of plan revision rules. The rule base may not contain multiple rules for the same $achieve(\phi)$ statement. We also introduce a function $\mathcal{T}$ that takes a belief base σ and a basic action a and yields the belief base resulting from executing a in σ. This function is needed in order to define the semantics of plan execution. We use $\Sigma = \wp(\mathcal{L})$ to denote the set of belief bases.

Definition 9 *(plan execution agent)* Let $\mathcal{T} : (\mathsf{BasicAction} \times \Sigma) \rightarrow \Sigma$ be a function specifying the belief update resulting from the execution of basic actions. A plan execution agent, typically denoted by $\mathcal{A}'$, is a tuple $\langle \sigma, \gamma, \Pi, \mathsf{PR}, \mathcal{T} \rangle$, where $\sigma \subseteq \mathcal{L}$ is the belief base, $\gamma \subseteq \mathcal{L}$ is the goal base, $\Pi \subseteq \mathsf{Plan}$ is the plan base and $\mathsf{PR} \subseteq \mathcal{R}_{\mathsf{PR}}$ is a set of plan revision rules. Further, $\sigma \not\models \bot$ and $\gamma \not\models \bot$ and

all sets σ, γ, Π and PR are finite. The rule base PR does not contain multiple rules with an equivalent head, i.e., if $achieve(\phi) \Rightarrow \pi \in \mathsf{PR}$, there is not a rule $achieve(\phi') \Rightarrow \pi' \in \mathsf{PR}$ such that $\phi \equiv \phi'$.

We can now move on to defining the semantics of plan execution. As it will become clear, we only need the semantics of individual plans for the relation between plan generation and plan execution that we will establish in Section 4. The semantics of executing a plan base containing a set of plans can be defined by interleaving the semantics of individual plans (see [8]).

The semantics of a programming language can be defined as a function taking a statement (plan) and a state (beliefbase), 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. There are various ways of defining a semantic function and in this paper we are concerned with the so-called *operational* semantics [4].

The operational semantics of a language is usually defined using transition systems [10]. 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. A configuration is here a tuple $\langle \pi, \sigma \rangle$, consisting of a plan π and a belief base σ. Below, we give the transition system $\mathsf{Trans}_{\mathcal{A}'}$ that defines the semantics of plan execution. 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 10 *($\mathsf{Trans}_{\mathcal{A}'}$)* Let $\mathcal{A}'$ be a plan execution agent with a set of plan revision rules PR and a belief update function $\mathcal{T}$. The transition system $\mathsf{Trans}_{\mathcal{A}'}$, consisting of a transition rule for action execution and one for rule application, is defined as follows. Let $a \in \mathsf{BasicAction}$.

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

Let $achieve(\phi) \Rightarrow \pi \in \mathsf{PR}$.

$$\langle achieve(\phi); \pi', \sigma \rangle \rightarrow_{apply} \langle \pi; \pi', \sigma \rangle$$

Note that the goal base is not used in this semantics. Based on this transition system, we define the operational semantic function below. This function takes an initial plan and belief base. It yields the belief base resulting from executing the plan on the initial belief base, as specified through the transition system.

Definition 11 *(operational semantics)* Let $x_i \in \{exec, apply\}$ for $1 \leq i \leq n$. The operational semantic function $\mathcal{O}^{\mathcal{A}'} : \mathsf{Plan} \to (\Sigma \to \Sigma)$ is a partial function that is defined as follows.

$$\mathcal{O}^{\mathcal{A}'}(\pi)(\sigma) = \begin{cases} \sigma_n & \text{if } \langle \pi, \sigma \rangle \to_{x_1} \ldots \to_{x_n} \langle \epsilon, \sigma_n \rangle \text{ is a finite sequence of} \\ & \text{transitions in } \mathsf{Trans}_{\mathcal{A}'} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The result of executing a plan is a single belief base, as plan execution as defined in this paper is deterministic: in any configuration, there is only one possible next configuration (or none). See for example [17] for a specification of the semantics of plan execution in case of non-determinism.

4 Relation between Plan Generation and Plan Execution

In this section, we will investigate how these two are related. In order to do this, we first define a function f, which transforms plan generation rules into plan revision rules of a similar form.

Definition 12 *(plan generation rules to plan revision rules)* The function $f : \wp(\mathcal{R}_{\mathsf{PG}}) \to \wp(\mathcal{R}_{\mathsf{PR}})$, transforming plan generation rules into plan revision rules, is defined as follows: $f(\mathsf{PG}) = \{achieve(\phi) \Rightarrow \pi \mid \phi \Rightarrow \pi \in \mathsf{PG}\}$.

The theorem we prove, relates the operational semantics of the total plans of an extension of a plan generation agent, to the plans in the initial plan base of a corresponding plan execution agent. It says that for any total plan α in the extension, there is a plan π in the plan base of the plan execution agent, such that the operational semantics of α and π are equivalent. The plan α is a plan from the plan generation agent and we have not defined an operational semantics in this context. We however take for the operational semantics of α the operational semantics for plans as defined in the context of plan execution agents. Note though that, for the semantics of α, only the *exec* transition of the transition system on which the operational semantics is based, is relevant.[5]

The intuition as to why this relation would hold, is the following. The generation of a total plan α from a partial plan π under a set of plan generation rules PG, corresponds with the execution of π, under a set of plan revision rules $f(\mathsf{PG})$. The plan revision rules applied during execution of π have a plan generation counterpart that is applied during generation of α. Further, the basic actions that are executed during the execution of π, are precisely the basic actions of α (in the same order). Because of this, the operational semantics of α and π are equivalent, as the execution of basic actions completely determines the changes to the initial belief base, and therefore the belief base at the end of the execution.

[5] We could have defined a new transition system for total plans, only containing the *exec* transition of the system of Definition 10, and a corresponding operational semantics. This is straightforward, so we omit this.

If $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ is a plan generation agent, the rule base of the corresponding plan execution agent $\mathcal{A}'$ should thus be $f(\mathsf{PG})$. For the belief base and goal base of $\mathcal{A}'$, we take σ and γ, respectively. As for the plan base of $\mathcal{A}'$, we cannot just take Π, for the following reason. A total plan α in an extension of $\mathcal{A}$ can be generated either from a partial plan π that was already in Π, or from a plan π that has been added by applying a plan generation rule to the goal base (through a plan addition transition in the process). If the latter is the case, we have to make sure that π is in the plan base of $\mathcal{A}'$, as this is the plan of which the semantics is equivalent with α. We thus define that the plan base of $\mathcal{A}'$ is $\Pi \cup \{\pi \mid \mathit{achieve}(\phi) \Rightarrow \pi \in f(\mathsf{PG}), \gamma \models \phi\}$. We now have the following theorem.

Theorem 1 Let $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ be an agent and let E be an extension of $\mathcal{A}$. Let $\mathcal{A}' = \langle \sigma, \gamma, \Pi', f(\mathsf{PG}), \mathcal{T} \rangle$ where $\Pi' = \Pi \cup \{\pi \mid \mathit{achieve}(\phi) \Rightarrow \pi \in f(\mathsf{PG}), \gamma \models \phi\}$ and let $\alpha \in \mathsf{TotalPlan}$. We then have the following.

$$\forall \alpha \in E : \exists \pi \in \Pi' : \mathcal{O}^{\mathcal{A}'}(\alpha)(\sigma) = \mathcal{O}^{\mathcal{A}'}(\pi)(\sigma)$$

In order to prove this theorem, we need a number of auxiliary definitions and lemmas. The first is the notion of an extended process. The idea is, that we want to derive from a given process p and a given total plan α in the extension corresponding with p, those steps in p that lead from some initial partial plan π to α. For this, we give each plan in the plan base of the agent a unique number. Then, we associate with each step in the process the number of the plan that is being refined. If a plan is added through a plan addition transition, we give this new plan a unique number and associate this number with the transition step.

The elements of the sets of an extended process are thus pairs from $\mathsf{Plan} \times \mathbb{N}$. A pair $(\pi, i) \in (\mathsf{Plan} \times \mathbb{N})$ will be denoted by π^i. We use the notion of a natural number i being fresh in E to indicate uniqueness of i in E: i is fresh in E if there is not a plan π^i in E.[6] Further, a rule $\phi \Rightarrow \pi$ can only be applied to refine a plan $\pi_1; \mathit{achieve}(\phi); \pi_2$, if $\mathit{achieve}(\phi)$ is the leftmost *achieve* statement of the plan, i.e., if π_1 is a total plan. This corresponds more closely with the application of plan revision rules in plan execution, as during execution always the first (or leftmost) *achieve* statement of a plan is rewritten.

Definition 13 *(extended process)* Let $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ be a plan generation agent and let $I(\Pi)$ be Π where each plan in Π is assigned a unique natural number. A sequence of sets, alternated with natural numbers, $E_0, i_1, E_1, \ldots, i_n, E_n$ with $E_i \subseteq \mathsf{Plan}$ and $i_j \in \mathbb{N}$ with $1 \leq j \leq n$ is an extended process of $\mathcal{A}$ iff $E_0 = I(\Pi)$ and it holds for all triples E_k, i, E_{k+1} in this sequence that $E_k \rightarrow_i E_{k+1}$ is a transition that can be derived in the transition system below.

Let $\phi \Rightarrow \pi \in \mathsf{PG}$ be a plan generation rule. The transition rule for *plan addition* is then defined as follows:

$$\frac{\gamma \models \phi \qquad \pi \notin E \qquad \mathit{coherent}(\pi, E)}{E \rightarrow_i E'}$$

[6] We refer to the pairs π^i as plans and we will from now on take the set Plan as including both ordinary plans π and pairs π^i.

where $E' = E \cup \{\pi^i\}$ with i fresh in E. The transition rule for *plan refinement* is defined as follows:

$$\frac{(\alpha_1; achieve(\phi); \pi_2)^i \in E \qquad (\alpha_1; \pi; \pi_2)^i \notin E \qquad coherent(\alpha_1; \pi; \pi_2, E)}{E \rightarrow_i E'}$$

where $\alpha_1 \in \mathsf{TotalPlan}$, $\pi_2 \in \mathsf{Plan}$ and $E' = E \cup \{(\alpha_1; \pi; \pi_2)^i\}$.

The notion of a closed process (Definition 6) as defined for processes in Definition 4, is applied analogously to extended processes.

We will prove theorem 1 using the notion of an extended process. Theorem 1 is however defined in terms of an extension, which is defined in terms of ordinary processes, rather than extended processes. We thus have to show that extended processes and processes are equivalent in some sense. We show that for any closed process there is a closed extended process that has the same final set of plans, with respect to the total plans in this set. We only provide a brief sketch of the proof.

Lemma 1 *(process equivalence)* Let $\mathcal{A}$ be a plan generation agent and let $t : \wp(\mathsf{Plan}) \rightarrow \wp(\mathsf{TotalPlan})$ be a function yielding the total plans of a set of plans. The following then holds: there is a closed process $E_0, \ldots, E_n$ of $\mathcal{A}$, iff there is a closed extended process $E'_0, i_1, E'_1, \ldots, i_n, E'_n$ of $\mathcal{A}$ such that $t(E_n) = t(E'_n)$ (modulo superscripts of plans).

Sketch of proof: ($\Leftarrow$) If a transition $E \rightarrow_i E'$ can be derived in the transition system of Definition 13, then a transition $E \rightarrow E'$ can be derived in the system of Definition 4 (modulo superscripts). ($\Rightarrow$) This is proven by viewing the plan generation rules as the production rules of a grammar and the total plans that can be generated by these rules as the language of this grammar. The formulas ϕ and the statements $achieve(\phi)$ are considered the non-terminals of the grammar and the set of basic actions $\mathsf{BasicAction}$ the terminals. The plans of the first element of an (extended) process can be viewed as the start symbols of the grammar, together with those plans that are added through the transition rule for plan addition.

It is the case that for any derivation of a string (or total plan) in the grammar, an equivalent leftmost derivation, in which at each derivation step the leftmost non-terminal is rewritten, can be constructed. Derivations in an extended process correspond with leftmost derivations, from which the desired result can be concluded. □

Given a closed extended process p with E_n as its final element, and a total plan $\alpha^i \in E_n$, we are interested in those steps of p that lead to the derivation of α^i. In other words, we are interested in those steps that are labelled with i. For this, we define the notion of an i-process of an extended process. This consists of a sequence of pairs of sets of plans, where each pair corresponds with a derivation step that is labelled with i, in the original extended process.

Given the i-process p_i of an extended process p, we define the notion of the i-derivation of p_i. The i-derivation of p_i is the sequence of singleton sets of plans,[7] that is yielded by subtracting for each pair (E, E') occurring in p_i, the set E from the set E'. An i-derivation is thus a sequence $\pi^i_1, \pi^i_2, \ldots, \pi^i_m$,[8] in which each plan is labelled with i. The sequence can be viewed as the derivation of the plan π^i_m from the initial plan π^i_1, as each step from π^i_j to π^i_{j+1} in this sequence corresponds with the application of a plan generation rule to π^i_j, yielding π^i_{j+1}.

Definition 14 *(i-derivation)* Let $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ be a plan generation agent and let $p = E_0, i_1, E_1, \ldots, i_n, E_n$ be a closed extended process of $\mathcal{A}$. The i-process p_i of p is then defined as a sequence of pairs $(E'_0, E'_1), \ldots, (E'_{m-1}, E'_m)$ such that the following holds: (E, E') occurs in p_i iff E, i, E' occurs in p and for any two consecutive pairs $(E_j, E_{j+1}), (E_{j+2}, E_{j+3})$ occurring in p_i it should hold that $E_{j+1} \subseteq E_{j+2}$.

Let $p_i = (E_0, E_1), \ldots, (E_{m-1}, E_m)$ be the i-process of a closed extended process p. The i-derivation of p_i is then defined as follows: $(E_1 \setminus E_0), \ldots, (E_m \setminus E_{m-1})$.

We want to associate the semantics of a total plan α in some extension of a plan generation agent, with the semantics of a corresponding plan π in the initial plan base of a plan execution agent. We do this by showing that the basic actions executed during the execution of π, correspond exactly with the basic actions of α. For this, we define a variant of the transition system of Definition 10, in which the configurations are extended with a third element. This element, which is a total plan, represents the basic actions that have been executed so far in the execution. Further, we define the execution of a sequence of basic actions in one transition step. This is convenient when proving lemma 2.

Definition 15 *($\mathsf{Trans}'_{\mathcal{A}'}$)* Let $\mathcal{A}'$ be a plan execution agent with a set of plan revision rules PR and a belief update function $\mathcal{T}$. The transition system $\mathsf{Trans}'_{\mathcal{A}'}$, consisting of a transition rule for action execution and one for rule application, is defined as follows.

Let $\alpha \in \mathsf{TotalPlan}$ be a sequence of basic actions and let $\mathcal{T}' : (\mathsf{TotalPlan} \times \Sigma) \rightarrow \Sigma$ be the lifting of $\mathcal{T}$ to sequences of actions, i.e., $\mathcal{T}'(a;\alpha)(\sigma) = \mathcal{T}'(\alpha)(\mathcal{T}(a)(\sigma))$. Further, let $\alpha' \in \mathsf{TotalPlan}$ be a sequence of basic actions, representing the actions that have already been executed.

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

Let $achieve(\phi) \Rightarrow \pi \in \mathsf{PR}$.

$$\langle achieve(\phi); \pi', \sigma, \alpha \rangle \rightarrow_{apply} \langle \pi; \pi', \sigma, \alpha \rangle$$

[7] It is a sequence of *singleton* sets, as each pair in an i-process corresponds with a derivation step in the original process. In a derivation step from E to E', exactly one plan is added to E.

[8] We omit curly brackets.

It is easy to see that an operational semantics $\mathcal{O}'$ can be defined[9] on the basis of this transition system that is equivalent with the operational semantics of Definition 11, i.e., such that $\mathcal{O}'(\pi)(\sigma) = \mathcal{O}(\pi)(\sigma)$ for any plan π and belief base σ. The initial configuration of any transition sequence in $\mathsf{Trans}'_{\mathcal{A}'}$ should be of the form $\langle \pi, \sigma, \epsilon \rangle$, as the third element represents the sequence of actions that have been executed, which are none in the initial configuration.

In the proof of lemma 2, we use the notion of a maximum prefix of a plan.

Definition 16 *(maximum prefix)* Let $\alpha \in \mathsf{TotalPlan}$ and let $\pi \in \mathsf{Plan}$. We then say that α is a maximum prefix of π iff $\alpha = \pi$ or $\pi = \alpha; achieve(\phi); \pi'$. Note that π' can be ϵ.

Lemma 2 says the following. Let α^i be a total plan in a closed extended process of a plan generation agent, and let π^i_1 be the first plan of the i-derivation of α^i. It is then the case that the actions executed during the execution of π_1 (given an appropriate set of plan revision rules), are exactly the actions of α (in the same order).

Lemma 2 Let $\mathcal{A} = \langle \sigma, \gamma, \Pi, \mathsf{PG} \rangle$ be a plan generation agent and let $p = E_0, i_1, E_1, \ldots, i_n, E_n$ be a closed extended process of $\mathcal{A}$. Let $\alpha^i \in E_n$ where $\alpha \in \mathsf{TotalPlan}$. Further, let $\pi^i_1, \ldots, \alpha^i$ be the i-derivation of the i-process p_i of p. Let $\mathcal{A}' = \langle \sigma, \gamma, \Pi', f(\mathsf{PG}), \mathcal{T} \rangle$ be a plan execution agent where $\Pi' = \Pi \cup \{\pi \mid achieve(\phi) \Rightarrow \pi \in f(\mathsf{PG}), \gamma \models \phi\}$. Further, let $\mathcal{T}'(\alpha)(\sigma)$ be defined and let $x_i \in \{exec, apply\}$ for $1 \leq i \leq m-1$. The following then holds.

A transition sequence of the form

$$\langle \pi_1, \sigma, \epsilon \rangle \rightarrow_{x_1} \ldots \rightarrow_{x_{m-1}} \langle \epsilon, \sigma_m, \alpha \rangle \quad \text{can be derived in } \mathsf{Trans}'_{\mathcal{A}'}. \quad (4.1)$$

Sketch of proof: We say that a plan π^i corresponds with a configuration $\langle \pi', \sigma, \alpha \rangle$ iff $\pi = \alpha; \pi'$. Let π^i_k and π^i_{k+1} be two consecutive plans in the i-derivation of p_i, where π^i_k is of the form $\alpha_2; achieve(\phi_2); \pi_2$ and π^i_{k+1} is of the form $\alpha_2; \pi; \pi_2$. This corresponds with the application of plan generation rule $\phi_2 \Rightarrow \pi$. Let π be of the form $\alpha_3; achieve(\phi_3); \pi_3$. We then have that the following transition sequence can be derived in $\mathsf{Trans}'_{\mathcal{A}'}$.

$$\langle achieve(\phi_2); \pi_2, \sigma, \alpha_2 \rangle \rightarrow_{apply} \langle \alpha_3; achieve(\phi_3); \pi_3; \pi_2, \sigma, \alpha_2 \rangle \rightarrow_{exec} \langle achieve(\phi_3); \pi_3; \pi_2, \sigma', \alpha_2; \alpha_3 \rangle \quad (4.2)$$

This pair of transitions is correspondence and maximum prefix preserving. If π_1 (transition sequence (4.1)) is of the form $\alpha_1; achieve(\phi_1); \pi$, we can derive a transition in which α_1 is executed. This yields a configuration of the form

[9] We omit superscript $\mathcal{A}'$.

$\langle achieve(\phi_1); \pi, \sigma', \alpha_1 \rangle$, which corresponds with π_1^i and for which it holds that α_1 is a maximum prefix of π_1. From this configuration, a sequence of *apply* and *exec* transitions can be derived, given that we have (4.2) for every pair π_k^i and π_{k+1}^i occurring in the i-derivation. From the fact that this sequence of transitions is correspondence and maximum prefix preserving, we can conclude that the final configuration $\langle \pi_m, \sigma_m, \alpha_m \rangle$ of the sequence must be of the form $\langle \epsilon, \sigma_m, \alpha \rangle$ (observe that α_i is the final plan of the i-derivation, which should correspond with $\langle \pi_m, \sigma_m, \alpha_m \rangle$). □

We are now in a position to prove theorem 1.

Proof of theorem 1 (sketch): We do not repeat the premisses of the theorem. Let $\alpha \in E$ be a total plan in E. By lemma 1, we then have that there is a closed extended process with a final set E_n such that $\alpha^i \in E_n$ for some natural number i. Let $\pi_1^i, \ldots, \alpha^i$ be the corresponding i-derivation. The plan π_1 was either added in the process through a plan addition transition, or it was already in Π. From this we can conclude that $\pi_1 \in \Pi'$.

If $\mathcal{T}'(\alpha)(\sigma)$ is defined, we have by lemma 2 that a transition sequence of the form $\langle \pi_1, \sigma, \epsilon \rangle \rightarrow_{x_1} \ldots \rightarrow_{x_{m-1}} \langle \epsilon, \sigma_m, \alpha \rangle$ can be derived in $\mathsf{Trans}'_{\mathcal{A}'}$. We thus have $\mathcal{O}^{\mathcal{A}}(\pi_1)(\sigma) = \sigma_m$. From the fact that only action executions may change the belief base, and the fact that α are the actions executed over the transition sequence, we can then conclude that $\mathcal{O}^{\mathcal{A}}(\alpha)(\sigma) = \sigma_m$. A similar line of reasoning can be followed if $\mathcal{T}'(\alpha)(\sigma)$ is not defined. □

5 Conclusion and Future Research

In this paper, we presented two formal approaches for generating and executing the plans of cognitive agents and discussed their characteristics. We explained how these approaches can be used to define the semantics of programming languages for cognitive agents in terms of operational semantics. The relation between these approaches is investigated and formally established as a theorem. The presented theorem shows that the behavior of plan generation agents is "included" in the behavior of plan execution agents.

However, for reasons simplicity, many simplifying assumptions have been introduced which make the presented approaches too limited to be applied to real cognitive agent programming languages. Future research will thus concern extending the results to more elaborate versions of the presented agent programming frameworks. Also, the characteristics of special cases will have to be investigated such as the case where there is only one extension of a plan generation agent. Finally, the notion of coherence between plans is not explored and left for future research.

References

1. G. Antoniou. *Nonmonotonic Reasoning.* Artificial Intelligence. The MIT Press, Cambridge, Massachusetts, 1997.

2. M. Dastani and L. van der Torre. Programming BOID-Plan agents: deliberating about conflicts among defeasible mental attitudes and plans. In *Proceedings of the Third Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04)*, pages 706–713, New York, USA, 2004.
3. M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A programming language for cognitive agents: goal directed 3APL. In *Programming multiagent systems, first international workshop (ProMAS'03)*, volume 3067 of *LNAI*, pages 111–130. Springer, Berlin, 2004.
4. J. de Bakker. *Mathematical Theory of Program Correctness*. Series in Computer Science. Prentice-Hall International, London, 1980.
5. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *ATAL '97: Proceedings of the 4th International Workshop on Intelligent Agents IV, Agent Theories, Architectures, and Languages*, pages 155–176, London, UK, 1998. Springer-Verlag.
6. M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.
7. G. d. Giacomo, Y. Lespérance, and H. Levesque. *ConGolog*, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.
8. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Int. J. of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.
9. F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, 1992.
10. G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
11. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: a BDI reasoning engine. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.
12. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.
13. 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.
14. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
15. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.
16. J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Lyon, France, 2002.
17. 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 *LNAI*, pages 16–32, 2005.