Logical Foundations for a Rational BDI Agent Programming Language (Extended Version)*

Shakil M. Khan and Yves Lespérance

Department of Computer Science and Engineering
York University, Toronto, ON, Canada
{skhan, lesperan}@cse.yorku.ca

Abstract. To provide efficiency, current BDI agent programming languages with declarative goals only support a limited form of rationality – they ignore other concurrent intentions of the agent when selecting plans, and as a consequence, the selected plans may be inconsistent with these intentions. In this paper, we develop logical foundations for a rational BDI agent programming framework with prioritized declarative goals that addresses this deficiency. We ensure that the agent's chosen declarative goals and adopted plans are consistent with each other and with the agent's knowledge. We show how agents specified in our language satisfy some key rationality requirements.

1 Introduction

This paper contributes to the foundations of Belief-Desire-Intention agent programming languages/frameworks (BDI APLs), such as PRS [10], AgentSpeak [19], etc. Recently, there has been much work on incorporating *declarative goals* in these APLs [7, 28, 21, 5, 27, 22]. In addition to defining a set of plans that can be executed to try to achieve a goal, these programming languages also incorporate goals as declarative descriptions of the states of the world which are sought. A typical BDI APL with declarative goals (APLwDG) uses a user-specified hierarchical plan library Π containing abstract plans, a procedural goal-base Γ containing a set of plans that the agent is committed to execute, and a declarative goal-base Δ that has goals that the agent is committed to achieve. In response to events in the environment and to goals in Δ, in each cycle the agent interleaves selecting plans from Π, adopting them to Γ, and executing actions in Γ. The execution of some of these actions can in turn trigger the adoption of other declarative goals. This process is repeated until all the goals in Δ are successfully achieved. The role of these declarative goals in an APLwDG is essentially for monitoring goal achievement and performing recovery when a plan has failed by decoupling plan failure/success from that of goal. Since these declarative goals capture the reason for executing plans, they are necessary to perform rational deliberation, and react in a rational way to changes in goals that result from communication, e.g. requests.

While current APLwDGs have evolved over the past few years — e.g. some of them handle restricted forms of temporally extended goals [8] — to keep them tractable

* This paper is an extended version of [16] and is also a revised version of [14].

and practical, they sacrifice some principles of rationality. In particular, while selecting plans to achieve a declarative goal, they ignore other concurrent intentions of the agent. As a consequence, the selected plan may be inconsistent with the agent's other intentions. Thus the execution of such an intended plan can render other contemporary intentions impossible to bring about. Also, these APLwDGs typically rely on syntactic formalizations of declarative goals, subgoals, and their dynamics, whose properties are often not well understood.

Apart from this, there has been work that focuses on maintaining consistency of a set of concurrent intentions. For example, Clement et al. [3, 4] argue that agents should be able to reason about abstract HTN plans and their interactions before they are fully refined. They propose a method for deriving summary information (i.e. external preconditions and effects) of abstract plans and discuss how this information can be used to coordinate the interactions of plans at different levels of abstractions. Thangarajah et al. [26] use such summary information to detect and resolve conflicts between goals at run time. Horty and Pollack [9] propose a decision theoretic approach to compute the utility of adopting new (non-hierarchical) plans, given a set of already adopted plans. While some of these approaches can be integrated in APLs (e.g. [26]), they leave out many aspects of rationality (e.g. they do not say what the agent should do if external interference makes two of her intentions permanently incompatible), and do not deal with declarative goals.

In this paper, we develop a logical framework for a rational BDI APL with prioritized declarative goals called Simple Rational APL (SR-APL, henceforth), that addresses these deficiencies of previous APLwDGs. Our framework combines ideas from the situation calculus-based Golog family of APLs (e.g. [6]), our expressive semantic formalization of prioritized goals, subgoals, and their dynamics [13, 15], and work on BDI APLs. We ensure that the agent's chosen declarative goals and adopted plans are consistent with each other and with the agent's knowledge. In doing this, we must address two fundamental questions about rational agency: (1) *What does it mean for a BDI agent to be committed to concurrently execute a set of plans next while keeping the option of further commitments to other plans open, in a way that does not allow procrastination?* (2) *How to ensure consistency between an agent's adopted declarative goals and adopted plans, given that some of the latter might be abstract, i.e. might be only partially instantiated in the sense that they include subgoals for which the agent has not yet adopted a (concrete) plan?* We show how agents specified in our framework satisfy some key rationality requirements. Our framework tries to bridge the gap between agent theories and practical APLs by providing a model and specification of an idealized BDI agent whose behavior is closer to what a rational agent does. As such, it allows one to understand how compromises made during the development of a practical APLwDG affect the agent's rationality.

The paper is organized as follows: in the next section, we discuss a motivating example. In Sections 3 and 4, we outline our formal BDI framework. In Section 5, we specify the semantics of SR-APL. In Section 6, we show that in the absence of external interference, our agent behaves in ways that satisfy some key rationality principles. Then in Section 7, we summarize our results and discuss possible future work.

2 A Motivating Example

Consider a blocks world domain, where each block is one of four possible colors: blue, yellow, green, and red. There is only a stacking action $stack(b, b')$: b can be stacked on b' in state s if $b \neq b'$, both b and b' are *clear* in s, and b is *on the table* in s. There are no unstacking actions, so the agent cannot use a block to build two different towers at different times. Assume that there are four blocks, B_B, B_Y, B_G, and B_R, one of each color. the agent knows the *color of* these blocks, and knows that initially all the blocks are on the table and are clear. Now assume that the agent has the following two goals: (1) to eventually have a 2 blocks tower that has a green block on top and a non-yellow block underneath, and (2) to have a 2 blocks tower with a blue block on top and a non-red block underneath; thus $\Delta = \{\Diamond\text{Twr}^G_{\bar{Y}}, \Diamond\text{Twr}^B_{\bar{R}}\}$, where $\text{Twr}^{C_1}_{\bar{C_2}} \doteq \exists b, b'.\ \text{OnTbl}(b') \wedge \text{On}(b, b') \wedge \neg\text{C}_2(b') \wedge \text{C}_1(b)$. Suppose our agent's plan library Π has two rules:

$$\begin{aligned}
\Diamond\text{Twr}^G_{\bar{Y}} : [&\text{OnTbl}(b) \wedge \text{OnTbl}(b') \wedge b \neq b' \wedge \text{Clear}(b) \\
&\wedge \text{Clear}(b') \wedge \neg\text{Y}(b) \wedge \text{G}(b')] \leftarrow stack(b', b), \\
\Diamond\text{Twr}^B_{\bar{R}} : [&\text{OnTbl}(b) \wedge \text{OnTbl}(b') \wedge b \neq b' \wedge \text{Clear}(b) \\
&\wedge \text{Clear}(b') \wedge \neg\text{R}(b) \wedge \text{B}(b')] \leftarrow stack(b', b).
\end{aligned}$$

That is, if the agent has the goal to have a green and non-yellow tower and knows about a green block b' and a distinct non-yellow block b that are both clear and are on the table, then she should adopt the plan of stacking b' on b, and similarly for the goal of having a blue and non-red tower.

Now, consider a typical APLwDG, that (without considering the overall consistency of the agent's intentions) simply select plans from Π for the agent's goals in Δ and eventually executes them in an attempt to achieve her goals. We claim that such an APL is not always sound and rational. For instance, according to this plan library, one way of building a green non-yellow (and a blue non-red) tower is to construct a green-blue (a blue-green, respectively) tower. While these two plans are individually consistent, they are inconsistent with each other, since the agent has only one block of each color. Thus a rational agent should not adopt these two plans. However, it can be shown that the following would be a legal trace for our blocks world domain in such an APL:

$$\langle\{\}, \Delta\rangle \Rightarrow \langle\{\sigma_1\}, \Delta\rangle \Rightarrow \langle\{\sigma_1, \sigma_2\}, \Delta\rangle \Rightarrow \langle\{\sigma_2\}, \{\Diamond\text{Twr}^G_{\bar{Y}}\}\rangle.$$

The agent first moves to configuration $\langle\{\sigma_1\}, \Delta\rangle$ by adopting the plan $\sigma_1 = stack(B_B, B_G)$ in response to $\Diamond\text{Twr}^B_{\bar{R}}$, then to $\langle\{\sigma_1, \sigma_2\}, \Delta\rangle$ by adopting $\sigma_2 = stack(B_G, B_B)$ to handle $\Diamond\text{Twr}^G_{\bar{Y}}$, and then to $\langle\{\sigma_2\}, \{\Diamond\text{Twr}^G_{\bar{Y}}\}\rangle$ by executing the intended action σ_1. At this point, the agent is stuck and cannot complete successfully. Thus, in such an APL, not only is the agent allowed to adopt two inconsistent plans, but the execution of one of these plans makes other concurrent goals impossible (e.g. the execution of $stack(B_B, B_G)$ makes $\Diamond\text{Twr}^G_{\bar{Y}}$ impossible to achieve).

The problem arises in part because actions are not reversible in this domain; there is no action for moving a block back to the table or for unstacking it. This is common in real world domains, for instance, most tasks with deadlines or resources, e.g. doing

some errands before noon, a robot delivering mail without running out of battery power, etc. While such irrational behavior could in principle be avoided by using appropriate conditions in the antecedent of the plan-selection rules (e.g. by stating that the agent should only adopt a given plan if she does not have certain other goals), this puts an excessive burden on the agent programmer. Ideally, such reasoning about goals should be delegated to the agent.

3 Preliminaries

Our base framework for modeling goal change is the situation calculus as formalized in [17, 20]. In this framework, a possible state of the domain is represented by a situation. There is a set of initial situations corresponding to the ways the agent believes the domain might be initially, i.e. situations in which no actions have yet occurred. Init(s) means that s is an initial situation. The actual initial state is represented by a special constant S_0. There is a distinguished binary function symbol do where $do(a, s)$ denotes the successor situation to s resulting from performing the action a. Thus the situations can be viewed as a set of trees, where the root of each tree is an initial situation and the arcs represent actions. Relations (and functions) whose truth values vary from situation to situation, are called relational (functional, respectively) fluents, and are denoted by predicate (function, respectively) symbols taking a situation term as their last argument. There is a special predicate Poss(a, s) used to state that action a is executable in situation s. Finally, the function symbol Agent(a) denotes the agent of action a.

We use a theory $\mathcal{D}$ that includes the following set of axioms:[1] (1) action precondition axioms, one per action a characterizing Poss(a, s), (2) successor state axioms (SSA), one per fluent, that succinctly encode both effect and frame axioms and specify exactly when the fluent changes [20], (3) initial state axioms describing what is true initially including the mental states of the agents, (4) axioms identifying the agent of actions, one per action a characterizing Agent(a), (5) unique name axioms for actions, and (6) domain-independent foundational axioms describing the structure of situations [17].

Following [23], we model knowledge using a possible worlds account adapted to the situation calculus. $K(s', s)$ is used to denote that in situation s, the agent thinks that she could be in situation s'. Using K, the knowledge of an agent is defined as: Know$(\Phi, s) \doteq \forall s'.\ K(s', s) \supset \Phi(s')$, i.e. the agent knows Φ in s if Φ holds in all of her K-accessible situations in s. K is constrained to be reflexive, transitive, and Euclidean in the initial situation to capture the fact that agents' knowledge is true, and that agents have positive and negative introspection. The dynamics of knowledge is specified by providing a SSA for K that supports knowledge expansion as a result of sensing actions [23] and some *informing* communicative actions [12]. As shown in [23], the constraints on K continue to hold after any sequence of actions since they are preserved by the SSA for K. We also assume that the agent is aware of all actions.

To support modeling temporally extended goals, we introduced a new sort of *paths* along with an axiomatization for paths in [13]. A path is essentially an infinite sequence

[1] We will be quantifying over formulae, and thus assume $\mathcal{D}$ includes axioms for encoding of formulae as first order terms, as in [25]. We will also be using lists of programs, and assume that $\mathcal{D}$ includes an axiomatization of lists.

of situations, where each situation along the path can be reached by performing some *executable* action in the preceding situation. We use (possibly sub/super-scripted) variables p to denote paths. There is a predicate OnPath(p, s), meaning that the situation s is on path p. Also, Starts(p, s) means that s is the starting situation of path p. A path p starts with s iff s is the earliest situation on p.

We use $\Phi(s), \Psi(s), \cdots$, etc. to denote *state formulae* in the context of knowledge (and $\phi(p), \psi(p), \cdots$, etc. for *path formulae* in that of goals), each of which has a free situation variable s (path variable p, respectively). s (and p) will be bound by the context where the formula $\Phi(s)$ (and $\phi(p)$, respectively) appears. Where the intended meaning is clear, we sometimes suppress the situation variable (path variable) from $\Phi, \Psi, \cdots$, etc. ($\phi, \psi, \cdots$, etc. respectively). Also, we often use now to refer to a placeholder constant that stands for the current situation.

We will use some useful constructs that are defined in [13]. A state formula Φ *eventually holds* over the path p if Φ holds in some situation that is on p, i.e.: $\Diamond\Phi(p) \doteq \exists s'.\ \text{OnPath}(p, s') \wedge \Phi(s')$. Secondly, Suffix$(p', p, s)$ means that path p' is a suffix of another path p w.r.t. a situation s; Suffix(p', p, s) holds iff s is on p, and p' is the subpath of p that starts with s. Finally, SameHist(s_1, s_2) means that the situations s_1 and s_2 share the same history of actions, but perhaps starting from different initial situations.

4 Formalization of Prioritized Goals

In [13], we proposed a logical framework for modeling *prioritized goals* and their dynamics. Our formalization here is based on [13], but modified as specified in the last paragraph of this section. In our framework in [13], an agent can have multiple *goals* or *desires* at different priority levels, possibly inconsistent with each other. We specify how these goals evolve when actions/events occur and the agent's knowledge changes. We define the agent's *chosen goals* or *intentions*, i.e. the goals that the agent is actively pursuing, in terms of this goal hierarchy. In that framework, agents constantly optimize their chosen goals. To this end, we keep all prioritized goals in the goal-base unless they are explicitly dropped. At every step, we compute an optimal set of chosen goals given the hierarchy of prioritized goals, preferring higher priority goals, such that chosen goals are consistent with each other and with the agent's knowledge. Thus at any given time, some goals in the hierarchy are *active*, i.e. chosen, while others are *inactive*. Some of these inactive goals may later become active (e.g. if a higher priority active goal that is currently blocking an inactive goal becomes impossible or is dropped) and trigger the inactivation of other currently active (lower priority) goals.

Goal Semantics As in [13], we specify the agent's prioritized goals or *p-goals* using accessibility relation/fluent G. A path p is G-accessible at priority level n in situation s if all the goals of the agent at level n are satisfied over this path and if it starts with a situation that has the same action history as s. The latter requirement ensures that the agent's G-accessible paths are compatible with the actions that have been performed so far. We say that an agent has the p-goal that ϕ at level n in situation s (i.e. PGoal(ϕ, n, s)) iff ϕ holds over all paths that are G-accessible at n in s. A smaller n represents higher priority, and the highest priority level is 0. Thus as in [13], we assume

that the set of p-goals are totally ordered according to priority. Note that, in this framework one can evaluate goals over infinite paths and thus can handle arbitrary temporally extended goals; hence, unlike some other situation calculus based accounts where goal formulae are evaluated w.r.t. finite paths (e.g. [24]), in this framework one can handle, for example, unbounded maintenance goals.

As in [13], we allow the agent to have infinitely many p-goals. However in many cases, the modeler will want to specify a finite set of initial p-goals. When a finite number of p-goals is assumed, we can use the functional fluent $NPGoals(s)$ to represent the number of prioritized goals that the agent has in situation s. The modeler/programmer will usually provide some specification of the agent's initial p-goals at the various priority levels, using some *initial goal axioms*. For instance, the initial prioritized goals for our blocks world example with domain theory $\mathcal{D}_{BW}$ can be specified as follows:

$$\begin{aligned}
&\text{(a) } \text{Init}(s) \supset ((G(p,0,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \Diamond \text{Twr}^G_Y) \\
&\qquad\qquad \wedge (G(p,1,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s') \wedge \Diamond \text{Twr}^B_R)), \\
&\text{(b) } \forall n,p,s.\ \text{Init}(s) \wedge n \geq 2 \supset (G(p,n,s) \equiv \exists s'.\ \text{Starts}(p,s') \wedge \text{Init}(s')).
\end{aligned}$$

(a) specifies the p-goals of the agent in the initial situations (we assume that the goal $\Diamond \text{Twr}^G_Y$ has higher priority than $\Diamond \text{Twr}^B_R$); (b) makes $G(p,n,s)$ true for every path p that starts with an initial situation for $n \geq 2$. Thus at these levels, the agent has the trivial p-goal that she be in an initial situation.

An agent's chosen goals must be realistic. To filter out the paths that are known to be impossible from G, we define *realistic* p-goal accessible paths: p is G_R-accessible at level n in s if it is G-accessible at n in s and if it starts with a situation that is K-accessible in s. In our framework, an agent has the *realistic p-goal* that ϕ at level n in situation s (i.e. RPGoal(ϕ, n, s)) iff ϕ holds over all G_R-accessible paths at n in s.

We define chosen goals or *c-goals* using realistic p-goals. Note that an agent's realistic p-goals at various priority levels can be viewed as candidates for her c-goals. Given the set of realistic p-goals, in each situation the agent's c-goals are specified to be those that are in the maximal consistent set of higher priority realistic p-goals. We define this iteratively starting with a set that contains the highest priority realistic p-goal accessible paths, i.e. G_R-accessible paths at level 0. At each iteration we obtain the intersection of this set with the set of next highest priority G_R-accessible paths. If the intersection is not empty, a new chosen set of p-goal accessible paths (and p-goals defined by these paths) at level i is obtained. We call a p-goal chosen by this process an *active* p-goal. If on the other hand the intersection is empty, then it must be the case that the p-goal represented by this level is either in conflict with another active higher priority p-goal/a combination of two or more active higher priority p-goals, or is known to be impossible. In that case, that p-goal is ignored (i.e. marked as *inactive*), and the chosen set of p-goal accessible paths at level i is the same as at level $i-1$. To get the prioritized intersection of the set of G_R-accessible paths up to level n, the process is repeated until $i = n$ is reached. $G_\cap(p,n,s)$ is used to denote that in situation s, path p is in the prioritized intersection of G_R-accessible paths up to level n. We say that a path p is $G_\cap$-accessible in situation s, i.e. $G_\cap(p,s)$, if $G_\cap(p,n,s)$ holds for all levels n. Finally, we say that an agent has the c-goal that ϕ in situation s (i.e. CGoal(ϕ, s)) if ϕ holds over all $G_\cap$-accessible paths in

s. We can show that initially our blocks world agent has the p-goals/c-goals that $\Diamond\text{Twr}^G_Y$ and $\Diamond\text{Twr}^B_R$, i.e.: $\mathcal{D}_{BW} \models \forall s.\ \text{Init}(s) \supset \text{CGoal}(\Diamond\text{Twr}^G_Y \wedge \Diamond\text{Twr}^B_R, s)$.

To get positive and negative introspection of goals, we impose two inter-attitudinal constraints on the K and G-accessibility relations in the initial situations. We have shown that these constraints then continue to hold after any sequence of actions since they are preserved by the SSAs for K and G. See [11] for details.

Goal Dynamics An agent's goals change when her knowledge changes as a result of the occurrence of an action (including exogenous events), or when she adopts or drops a goal. There are two special actions, for *adopting a p-goal ϕ at some level n* and *dropping a p-goal ϕ*, $adopt(\phi, n)$ and $drop(\phi)$, and a third action for *adopting a subgoal ψ relative to a supergoal ϕ*, $adoptRT(\psi, \phi)$.

The dynamics of p-goals are specified using a SSA for G as follows (the agent's c-goals are automatically updated when her p-goals change). Firstly, to handle the occurrence of a non-adopt/drop action a, all p-goals are progressed to reflect the fact that this action has occurred. Secondly, to handle adoption of a p-goal ϕ at level m, a new formula containing the p-goal is added to the agent's goal hierarchy at m. To be precise, in addition to progressing all p-goals at all levels, a new level containing the p-goal that ϕ is inserted at m and all current levels with priority greater or equal to m are pushed one level down the hierarchy. Finally, to handle the dropping of a p-goal ϕ, the levels that imply the dropped goal in the agent's goal hierarchy are replaced by the trivial formula that the history of actions in the current situation has occurred, and thus the agent no longer has the p-goal that ϕ. See [13] for details.

Handling Subgoals We also handle subgoal adoption and model the dependencies between goals and the subgoals and plans adopted to achieve them. The latter is important since subgoals and plans adopted to bring about a goal should be dropped when the parent goal becomes impossible, or is dropped. We handle this as follows: adopting a subgoal ψ relative to a parent goal ϕ adds a new p-goal that contains *both this subgoal and this parent goal*, i.e. $\psi \wedge \phi$. This ensures that when the parent goal is dropped, the subgoal is also dropped, since when we drop the parent goal ϕ, all the p-goals at all G-accessibility levels that imply ϕ including $\psi \wedge \phi$ are also dropped. Note that the parent goal ϕ could be a p-goal at multiple levels. We assume that the subgoal ψ is always adopted w.r.t. the *highest priority supergoal level*, i.e. the highest priority level where ϕ holds. Also, the subgoal ψ is always adopted at the level immediately below the supergoal ϕ's level. The reason for doing this is that since ψ is a means to the end ϕ, they should have similar priorities. ψ is said to be a subgoal of ϕ in situation s (i.e. SubGoal(ψ, ϕ, s)) iff there is a G-accessibility level n in s such that ϕ is a p-goal at n while ψ is not, and for all G-accessibility levels in s where ψ is a p-goal, ϕ is also a p-goal. See [15, 11] for details of our formalization of subgoals.

Prioritized Goals for Committed Agents The formalization of prioritized goal dynamics in [13] ensures that the agent always tries to optimize her chosen goals. She will abandon a c-goal ϕ if an opportunity to commit to a higher priority but inconsistent with ϕ goal arises. As such, our account in [13] displays an idealized form of rational-

ity. This is in contrast to Bratman's [1] practical rationality that takes into consideration the resource-boundedness of real world agents. According to Bratman, intentions limit the agent's reasoning as they serve as a *filter for adopting new intentions*. However, the agent is allowed to override this filter in some cases, e.g. when adopting ϕ increases her utility considerably. The framework in [13] can be viewed as a theory of intention where the filter override mechanism is always triggered.

Note that, in that framework, the agent's c-goals are very dynamic. For instance, as mentioned earlier, a currently inactive p-goal ϕ may become active at some later time, e.g. if a higher priority active c-goal that is currently blocking ϕ (as it is inconsistent with ϕ) becomes impossible. This also means that another currently active c-goal ψ may as a result become inactive, not because ψ has become impossible, was achieved, or was dropped, but due to the fact that ψ has lower priority than and is inconsistent with the newly activated goal ϕ (see [13] for a concrete example).

Such very dynamic c-goals/intentions are problematic as a foundation for an APL, as the agent spends a lot of effort in "recomputing" her intentions and plans to achieve them, and her behavior becomes hard to predict for the programmer. To avoid this, here we use a modified version of our formalization in [13] that eliminates the filter override mechanism altogether so that agents' p-goals/desires are dropped as soon as they become inactive. We can do this with the following simple changes: (1) we require that initially the agent knows that her p-goals are all possible and consistent with each other, (2) we don't allow the agent to adopt p-goals that are inconsistent with her current c-goals/intentions, and (3) we modify the SSA for G so that the agent's p-goals are dropped when they become impossible or inconsistent with other higher priority c-goals. In the resulting "committed agent" framework, an agent's p-goals are much more dynamic than in the original framework. On the other hand, her c-goals are now much more persistent, and are simply the consequential closure of her desires, as these must now all be consistent with each other and with the agent's knowledge. The resulting model of goals is somewhat simplistic, but is sufficient in an APL context.

5 Agent Programming with Prioritized Goals

Our proposed framework SR-APL combines elements from BDI APLs such as AgentSpeak [19] and from the ConGolog APL [6], which is defined on top of the situation calculus. In addition, to facilitate monitoring of goal achievement and performing plan failure recovery, we incorporate declarative goals in SR-APL. To specify the operational semantics of plans in SR-APL, we will use a subset of the ConGolog APL. This subset includes programming constructs such as primitive actions a, wait/test actions $\Phi?$, sequence of actions $\delta_1; \delta_2$, nondeterministic choice of arguments $\pi v.\ \delta$, nondeterministic iteration δ^*, and concurrent execution of programs $\delta_1 \| \delta_2$, to mention a few. Also, as in ConGolog, we will use $\text{Trans}(\sigma, s, \sigma', s')$ to say that program σ in situation s can make a single step to reach situation s' with the program σ' remaining, and $\text{Final}(\sigma, s)$ to mean that the program σ may legally terminate in situation s. Finally, $\text{Do}(\sigma, s, s')$ means that there is a terminating execution of program σ that starts in s and ends in s'.

Components of SR-APL First of all, we have a *set of axioms/theory* $\mathcal{D}$ specifying

actions that can be done, the initial knowledge and (both declarative and procedural) goals of the agent, and their dynamics, as discussed in Section 3 and 4. Moreover, we also have a *plan library* Π with rules of the form $\phi : \Psi \leftarrow \sigma$, where ϕ is a goal formula, Ψ is a knowledge formula, and σ is a plan; a rule $\phi : \Psi \leftarrow \sigma$ means that if the agent has the c-goal that ϕ and knows that Ψ, then she should consider adopting the plan that σ. The *plan language* for σ is a simplified version of ConGolog and includes the empty program nil, primitive actions, waiting for a condition, sequences, and the special action for subgoal adoption, $adoptRT(\Diamond\Phi, \sigma)$; here $\Diamond\Phi$ is a subgoal to be adopted and σ is the plan relative to which it is adopted.[2] While our account of goal change is expressive enough to handle arbitrary temporally extended goals, here we focus on achievement goals and procedural goals exclusively. We believe that extending our framework to support maintenance goals should be straightforward, since maintenance goals behave like additional constraints on the agent behavior in contrast to achievement goals for which the agent needs to plan for.

Semantics of SR-APL An SR-APL agent can work on multiple goals at the same time. Thus at any time, an agent might be committed to several plans that she will execute in an interleaved fashion. We use our situation calculus domain theory $\mathcal{D}$ to model *both adopted declarative goals and plans*. Initially $\mathcal{D}$ only contains declarative goals. As specified by the SSA for G, $\mathcal{D}$ is updated by adding plans or other declarative goals to the agent's goal hierarchy when a transition rule (see below) makes the agent perform an $adopt$ or $adoptRT$ action. We ensure that an agent's declarative goals and adopted plans are consistent with each other and with the agent's knowledge. In our semantics, we specify this by ensuring that there is at least one possible course of actions (i.e. a path) known to the agent, and if she were to follow this path, she would end up realizing all of her declarative goals and executing all of her procedural goals.

One way of specifying an agent's commitment to execute a plan σ next in $\mathcal{D}$ is to say that she has the intention that $\text{Starts}(s) \wedge \exists s'.\ \text{OnPath}(s') \wedge \text{Do}(\sigma, s, s')$, i.e. that each of her intention-accessible paths p is such that it starts with some situation s, it has the situation s' on it, and s' can be reached from s by executing σ. However, this does not allow for the interleaved execution of several plans, since Do requires that σ be executed before any other actions/plans.

A better alternative is to represent the procedural goal as $\text{Starts}(s) \wedge \exists s'.\ \text{OnPath}(s') \wedge \text{DoAL}(\sigma, s, s')$, which says that the agent has the intention to execute *at least* the program σ next, and possibly more. $\text{DoAL}(\sigma, s, s')$ holds if there is an execution of program σ, possibly interleaved with other actions by the agent herself, that starts in situation s and ends in s', which we define as:[3]

$$\text{DoAL}(\sigma, s, s') \doteq \text{Do}(\sigma \| (\pi a.\ \text{Agent}(a) = agt?; a)^*, s, s').$$

[2] We use the ConGolog APL here because it has a situation calculus-based semantics that is well specified and compatible with our agent theory. We could have used any APL with these characteristics.

[3] Note that, while our theory supports exogenous actions performed by other agents, we assume that all actions in the plans of agt that specify her behavior must be performed by agt herself.

However, a new problem with this approach is that it allows the agent to *procrastinate* in the execution of the intended plans in $\mathcal{D}$. For instance, suppose that the agent has the p-goal at priority level n_1 to execute the program σ_1 and at level n_2 to execute σ_2 next. Then, according to our definition of DoAL, the agent has the intention at level n_1 to execute σ_1 and at level n_2 to execute σ_2, possibly concurrently with other actions next, since we use DoAL to specify those goals. The "other actions" at level n_1 (n_2, respectively) are meant to be actions from the plan σ_2 (σ_1, respectively). However, nothing requires that the additional actions that the agent might execute are indeed from σ_2(σ_1, respectively), and thus this allows her to perform actions that are unnecessary as long as they do not perturb the execution of σ_1 and σ_2.

To deal with this, we include an additional component, a *procedural intention-base* Γ, to an SR-APL agent. Γ is a list of plans that the agent is currently actively pursuing. To avoid procrastination, we will require that any action that the agent actually performs comes from Γ (as specified in the transition rule A_{step} below). In the following, we will use $\Gamma^{\|}$ to denote the concurrent composition of the programs in Γ:[4]

$$\Gamma^{\|} \doteq \textbf{if } (\Gamma = [\text{nil}]) \textbf{ then } \text{nil} \textbf{ else } \text{First}(\Gamma)\|(\text{Rest}(\Gamma))^{\|}.$$

In SR-APL, a *program configuration* $\langle \sigma, s \rangle$ is a tuple consisting of a program σ and a ground situation s. An *agent configuration* on the other hand is a tuple $\langle \Gamma, s \rangle$ that consists of a list of plans Γ and a ground situation s. The initial agent configuration is $\langle [\text{nil}], S_0 \rangle$. Although strictly speaking an agent configuration includes the knowledge and the goals of the agent, these can be obtained from the (fixed) theory $\mathcal{D}$ and the situation in the configuration.

The semantics of SR-APL are defined by a two-tier transition system. *Program-level transition rules* specify how a program written in our plan language may evolve. On top of this, we use *agent-level transition rules* to specify how an SR-APL agent may evolve. Our program-level transition rules are simply a subset of the ConGolog transition rules. We use $\langle \sigma, s \rangle \rightarrow \langle \sigma', s' \rangle$ as an abbreviation for $\text{Trans}(\sigma, s, \sigma', s')$.

Agent-Level Transition Rules These transition rules are given in Table 1 and are similar to those of a typical BDI APL.[5] First of all, we have a rule A_{sel} for *selecting and adopting a plan* using the plan library Π for some realistic p-goal $\Diamond\Phi$. It states that if: (a) there is a rule in the plan library Π which says that the agent should adopt an instance of the plan σ if she has $\Diamond\Phi$ as her p-goal and knows that some instance of Ψ, (b) $\Diamond\Phi$ is a realistic p-goal with priority n in s for which the agent hasn't yet adopted any subgoal, (c) the agent knows in s that Ψ', (d) θ unifies Ψ and Ψ', and (e) the agent does not intend not to adopt $\text{DoAL}(\sigma\theta)$ w.r.t. $\Diamond\Phi$ next, then she can adopt the plan $\sigma\theta$, adding $\text{DoAL}(\sigma\theta)$ as a subgoal of $\Diamond\Phi$ to her goals in the theory $\mathcal{D}$, and adding $\sigma\theta$ to Γ (here $\text{Handled}(\phi, s)$ is defined as $\exists\psi.\ \text{SubGoal}(\psi, \phi, s)$).

We can show that if an agent does not have the c-goal in s not to adopt a subgoal ψ w.r.t. a supergoal ϕ, then she does not have the c-goal that $\neg\psi$ next in s, i.e.:

[4] We will use various standard list operations, e.g. First (representing the first item of a list), Rest (representing the sublist that contains all but the first item of a list), Cons (for constructing a new list from an item and a list), Member (for checking membership of an item within a list),

Table 1. Agent Transition Rules

(A_{sel})	$\dfrac{\begin{array}{c}\text{Member}(\Diamond\Phi : \Psi \leftarrow \sigma, \Pi),\ \ \mathcal{D} \models \text{RPGoal}(\Diamond\Phi, n, s),\\ \mathcal{D} \models \neg\text{Handled}(\Diamond\Phi, s) \wedge \text{Know}(\Psi', s),\ \ \text{mgu}(\Psi, \Psi') = \theta,\\ \mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adoptRT(\text{DoAL}(\sigma\theta), \Diamond\Phi), now, s'), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\sigma\theta, \Gamma), do(adoptRT(\text{DoAL}(\sigma\theta), \Diamond\Phi), s)\rangle}$
(A_{step})	$\dfrac{\begin{array}{c}\text{Member}(\sigma, \Gamma),\ \ \mathcal{D} \models \text{RPGoal}(\text{DoAL}(\sigma), n, s),\\ \mathcal{D} \models \langle \sigma, s\rangle \rightarrow \langle \sigma', do(a, s)\rangle \wedge \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Replace}(\sigma, \sigma', \Gamma), do(a, s)\rangle}$
(A_{exo})	$\dfrac{\mathcal{D} \models \text{Exo}(a) \wedge \text{Poss}(a, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \Gamma, do(a, s)\rangle}$
(A_{clean})	$\dfrac{\text{Member}(\sigma, \Gamma),\ \ \mathcal{D} \models \neg\exists n.\ \text{RPGoal}(\text{DoAL}(\sigma), n, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Remove}(\sigma, \Gamma), s\rangle}$
(A_{rep})	$\dfrac{\begin{array}{c}\mathcal{D} \models \neg\exists s'.\ \langle \Gamma^{\|}, s\rangle \rightarrow \langle \Gamma', s'\rangle,\ \ \mathcal{D} \models \neg\text{Final}(\Gamma^{\|}, s),\\ \text{For all } \sigma \text{ s.t. Member}(\sigma, \Gamma) \text{ we have:}\\ \mathcal{D} \models \exists n.\ \text{RPGoal}(\text{DoAL}(\sigma), n, s) \wedge \text{Handled}(\text{DoAL}(\sigma), s),\\ \mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adopt(\text{Do}(\vec{a}), NPGoals(s)), now, s'), s),\\ \mathcal{D} \models \text{Agent}(\vec{a}) = agt \wedge \text{Do}(\vec{a}, s, s') \wedge \langle \Gamma^{\|}, s'\rangle \rightarrow \langle \Gamma', s''\rangle\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\vec{a}, \Gamma), do(adopt(\text{Do}(\vec{a}), NPGoals(s)), s)\rangle}$

Theorem 1.

$$\mathcal{D} \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adoptRT(\psi,\phi), now, s'), s) \supset$$
$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adoptRT(\psi,\phi), s')) \wedge \psi(p'), s).$$

Theorem 1 and condition (e) above imply that the agent does not have the c-goal not to execute $\sigma\theta$ concurrently with $\Gamma^{\|}$ and possibly other actions next, i.e.:

$$(i).\ \neg\text{CGoal}(\neg\exists s', s''.\ \text{Do}(adoptRT(\text{DoAL}(\sigma\theta), \Diamond\Phi), now, s')$$
$$\wedge\ \text{DoAL}(\sigma\theta \parallel \Gamma^{\|}, s', s''), s).$$

Moreover, it can be shown that in our framework, an agent acquires the c-goal that ψ after she adopts it as a subgoal of ϕ in s, provided that she has the realistic goal at some level n in s that ϕ, and that she does not have the c-goal in s that $\neg\psi$ next, i.e.:

Theorem 2.

$$\mathcal{D} \models \exists n.\ \text{RPGoal}(\phi, n, s) \wedge$$
$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adoptRT(\psi,\phi), s')) \wedge \psi(p'), s)$$
$$\supset\ \text{CGoal}(\psi, do(adoptRT(\psi,\phi), s)).$$

From (b), (i), and Theorem 2, we have that:

$$(ii).\ \text{CGoal}(\exists s'.\ \text{DoAL}(\sigma\theta \parallel \Gamma^{\|}, now, s'), do(adoptRT(\text{DoAL}(\sigma\theta), \Diamond\Phi), s)).$$

(i) ensures that the adopted subgoal $\sigma\theta$ is consistent with $\Gamma^{\|}$ in the sense that they can be executed concurrently, possibly along with other actions in s. (ii) confirms that $\sigma\theta$ is indeed intended after the $adoptRT$ action has happened.

Note that this notion of consistency is a weak one, since it does not guarantee that there is an execution of the program $(\sigma\theta \parallel \Gamma^{\|})$ after the $adoptRT$ action happens, but rather ensures that the program $\text{DoAL}(\sigma\theta \parallel \Gamma^{\|})$ is executable. In other words, $\sigma\theta$ and the programs in Γ *alone* might not be concurrently executable, and additional actions might be required. We'll come back to this issue later.

Secondly, we have a transition rule A_{step} for single stepping the agent program by *executing an intended action* from Γ. It says that if: (a) a program σ in Γ can make a program-level transition in s by performing a primitive action a with program σ' remaining in $do(a, s)$ afterwards, (b) $\text{DoAL}(\sigma)$ is a realistic p-goal with priority n in s, and (c) the transition is consistent with the agent's goals in the sense that she does not have the c-goal not to execute a in s, then the agent can execute a, and Γ and s can be updated accordingly.

Once again we have a weak consistency requirement in condition (c) above. Ideally, we would have added to (c) that the agent can continue from $do(a, s)$ in the sense

---

Remove (for removing all instances of a given item from a list), Replace (for replacing a given item with another item in a list), etc.

[5] We use $\text{CGoal}(\exists s'.\ \text{DoAL}(\sigma, now, s'), s)$ or simply $\text{CGoal}(\text{DoAL}(\sigma), s)$ as a shorthand for $\text{CGoal}(\exists s'.\ \text{Starts}(now) \wedge \text{OnPath}(s') \wedge \text{DoAL}(\sigma, now, s'), s)$.

that she does not have the c-goal not to execute the remaining program σ' concurrently with the other programs in Γ in $do(a, s)$, i.e. that $\mathcal{D} \models \neg$CGoal$(\neg\exists s'.$ Do$(a; (\sigma' \parallel \Gamma^{\parallel}), now, s'), s)$. However, note that Γ may not be complete in the sense that it may include plans that have actions that trigger the adoption of subgoals, for which the execution of $\Gamma^{\parallel}$ waits; but Γ does not have any adopted plans yet that can achieve these subgoals. Thus $\Gamma^{\parallel}$ by itself might currently have no complete execution, and will only become completely executable when all such subgoals have been fully expanded.

For example, consider a new agent for our blocks world domain who has a goal to eventually build a 3 blocks tower, i.e. $\Diamond$3Tower, where 3Tower $\doteq \exists b, b', b''.$ OnTbl$(b) \wedge$ On$(b', b) \wedge$ On(b'', b'). Also, in addition to the above rules, her plan library Π includes the following rule:

$$\begin{aligned}
&\Diamond\text{3Tower} : [\text{OnTbl}(b) \wedge \text{OnTbl}(b') \wedge \text{OnTbl}(b'') \wedge b \neq b' \wedge \\
&\qquad\qquad \text{Clear}(b) \wedge \text{Clear}(b') \wedge \text{Clear}(b'') \wedge \neg\text{Y}(b) \wedge \text{G}(b') \wedge \text{Y}(b'')] \leftarrow \sigma_1, \\
&\text{where } \sigma_1 = adoptRT(\Diamond\text{Twr}^G_Y, \text{DoAL}(\sigma_2)); \sigma_2, \text{ and } \sigma_2 = \text{Twr}^G_Y?; stack(b'', b').
\end{aligned}$$

This says that, if the agent knows about a non-yellow block b, a distinct green block b', and a yellow block b'' that are all clear and on the table, then her goal of building a 3 blocks tower can be fulfilled by adopting the plan that involves adopting the subgoal to eventually build a green non-yellow tower, waiting for the achievement of this subgoal, and then stacking b'' on b'. Suppose that in response to $\Diamond$3Tower, the agent adopted σ_1 as above as a subgoal of this goal using the A_{sel} rule, and thus σ_1 is added to Γ. In the next few steps, she will step through the adopted plan σ_1, executing one action at a time in an attempt to achieve her goal that $\Diamond$3Tower. Note that, in SR-APL, the hierarchical decomposition of a subgoal, e.g. σ_1 above, is a two step process. In the first step, in response to the execution (via A_{step}) of the $adoptRT(\Diamond\text{Twr}^G_Y, \text{DoAL}(\sigma_2))$ action in her plan σ_1 in Γ, the agent adopts $\Diamond\text{Twr}^G_Y$ as a subgoal of executing the remaining program σ_2, possibly along with other actions, i.e. w.r.t. DoAL(σ_2). Then in the second step, she uses the A_{sel} rule to select and adopt a plan for the subgoal $\Diamond\text{Twr}^G_Y$. We assume that the subgoal $\Diamond\text{Twr}^G_Y$ must always be achieved before the supergoal. To do this, we suspend the execution of the supergoal by waiting for the achievement of the subgoal. This can be specified by the programmer by having the supergoal σ_2 start with the wait action $\text{Twr}^G_Y?$ that waits for the subgoal to complete. But this means that σ_2 (and thus σ_1) by itself, i.e. without the DoAL construct, might not have a complete execution as it might get blocked when it reaches $\text{Twr}^G_Y?$. Moreover, since σ_2 is a member of Γ, $\Gamma^{\parallel}$ will have a complete execution only when all the subgoals in Γ have been fully expanded. Thus to deal with this, we use a weak consistency check that does not perform full lookahead over $\Gamma^{\parallel}$. However, our semantics ensures that any action a performed by the agent must not make the concurrent execution of all the adopted plans of the agent possibly with other actions impossible, i.e. it must be consistent with DoAL$(\Gamma^{\parallel})$, since A_{step} requires that doing a must be consistent with all her DoAL procedural goals (and other concurrent declarative goals) in her goal hierarchy, i.e. that $\mathcal{D} \models \neg$CGoal$(\neg\exists s'.$ Do$(a, now, s'), s)$.

Thirdly, we have a rule A_{exo} for *accommodating exogenous actions*, i.e. actions occurring in the agent's environment that are not under her control. When such an action

a occurs in s, the agent must update her p-goals by progressing the situation component of her configuration to $do(a, s)$.

Fourthly, we use the A_{clean} rule for *dropping adopted plans from the procedural goal-base Γ that are no longer intended in the theory $\mathcal{D}$*. It says that if there is a program σ in Γ, and executing σ possibly along with other actions is no longer a realistic p-goal, then σ should be dropped from Γ. This might be required when the occurrence of an exogenous action forces the agent to drop a plan by making it impossible to execute or inconsistent with her higher priority realistic p-goals. Recall that our theory automatically drops such plans from the agent's goal-hierarchy specified by $\mathcal{D}$.

Finally, we have a rule A_{rep} for *repairing an agent's plans in case she gets stuck*, i.e. when for all programs σ in Γ, the agent has the realistic p-goal that $\mathrm{DoAL}(\sigma)$ at some level n (and thus all of these $\mathrm{DoAL}(\sigma)$ are still individually executable and collectively consistent), but together they are not concurrently executable without some non-σ actions in the sense that $\Gamma^{\|}$ has no program-level transition in s. This could happen as a result of an exogenous action or as a side effect of our weak consistency check, as discussed below. The A_{rep} rule says that if: (a) $\Gamma^{\|}$ does not have a program level transition in s (which ensures that A_{step} can't be applied), (b) $\Gamma^{\|}$ is not considered to be completed in s, (c) every program in Γ is currently a realistic p-goal that has been handled (which ensures that A_{clean} and A_{sel} can't be applied), (d) there is a sequence of actions $\vec{a}$ that the agent does not intend not to execute next, and (e) $\vec{a}$ repairs Γ in the sense that there is a program level transition of $\Gamma^{\|}$ after $\vec{a}$ has been executed in s, then in an attempt to repair Γ, the agent should adopt $\vec{a}$ at the lowest priority level (i.e. at $NPGoals(s)$).

Why do we need this rule? One reason is because the agent could get stuck due to the occurrence of an exogenous action e, e.g. when e makes the preconditions of a plan σ in Γ false; note that, $\mathrm{DoAL}(\sigma)$ might still be executable after the occurrence of e, e.g. if there is an action sequence $\vec{r}$ (encoded by the DoAL construct) that can be used to restore the preconditions of σ.

Another reason repair may be needed is that we use partial lookahead when executing actions via A_{step}. For example, assume a domain with actions a, b, and r, all of which are initially possible. The execution of b makes the preconditions of a false, while that of r restores them. Our agent has two adopted plans, $\mathrm{DoAL}(a)$ and $\mathrm{DoAL}(b)$ in the theory $\mathcal{D}$, and $\Gamma = [a, b]$. Note that $b; a$ is not a valid execution of $\Gamma^{\|}$, since the execution of b invalidates the preconditions of a. But $b; r; a$ is indeed a valid execution of $(\mathrm{DoAL}(a) \wedge \mathrm{DoAL}(b))$. Since we only do partial consistency checking, our semantics allows the agent to perform b as the first action.[6] That is, to execute b using the A_{step} transition rule, we only need to ensure that b has a program-level transition in s and that this transition is consistent with the agent's goals in $\mathcal{D}$, i.e. with $\mathrm{DoAL}(a)$ and $\mathrm{DoAL}(b)$, both of which hold. After the execution of b, the agent will get stuck, as there is no action in the progression of Γ that she can perform. To deal with this, we include the repair rule that makes the agent plan for and commit to a sequence of actions that can be used to repair Γ, which for our example is r. Note that, we could have avoided

[6] Note that this does not mean that A_{step} allows the agent to perform an action that makes one of her goals impossible, e.g. to execute b when such a repair action r is not available.

the need for repairing plans in this case by strengthening the conditions of the A_{step} rule to do full lookahead by expanding all subgoals in Γ. However, this requires modeling the plan selection/goal decomposition process as part of the consistency check, which we leave for future work. We could have also relied on plan failure recovery techniques [28]. Finally, our repair rule does a form of conformant planning; more sophisticated forms of planning such as synthesizing conditional plans that include sensing actions could also be performed.

When the agent has complete information, there must be a repair plan available to the agent (whose actions can be performed by the agent herself) if her goals are consistent. In our framework, since the SSA for G drops all inconsistent goals/plans, the agent's p-goals are always consistent, and thus if complete information is assumed, it is always possible to repair the remaining plans. Consider our previous example: if the agent has $\mathrm{DoAL}(a)$ and $\mathrm{DoAL}(b)$ as her realistic p-goals, $\Gamma = [a, b]$, and if she has the c-goal not to execute an action from $\Gamma^{\|}$ (i.e. $\mathrm{CGoal}(\neg\exists s'.\ \langle \Gamma^{\|}, now\rangle \rightarrow \langle \Gamma', s'\rangle, s)$), then it must be the case that she does not have the c-goal not to execute $\Gamma^{\|}$ along with other actions (e.g. r), i.e. $\neg\mathrm{CGoal}(\neg\exists s'.\ \mathrm{DoAL}(a\|b, now, s'), s)$. Otherwise, one of $\mathrm{DoAL}(a)$ or $\mathrm{DoAL}(b)$ would have been dropped by the SSA for G as an agent's p-goals are always consistent with each other. Thus there must be a plan $\vec{a}$ that can repair Γ. Since the agent has complete information, this plan must work in all her epistemic alternatives (our repair rule does a form of conformant planning). Also, since by definition, the agent of the "other actions" in the DoAL construct is the agent herself, this means that she is also the agent of $\vec{a}$. If on the other hand the agent has only incomplete information, then a repair plan may need to perform sensing actions and branch on the results. We leave this kind of conditional planning for future work.

Also, note that this rule allows the agent to procrastinate in the sense that in addition to the plan that actually repairs Γ, she is allowed to adopt and execute actions that are unnecessary. This could be avoided by constraining the repair plan $\vec{a}$, e.g. by requiring it to be the shortest or the least costly plan etc. We leave this for future work.

In our operational semantics, we want to ensure that the procedural goals in Γ are consistent with those in the theory $\mathcal{D}$ before expansion of a subgoal/execution of an action occurs; so we assume that the A_{clean} rule has higher priority than A_{sel} and A_{step}. We can do this by adding appropriate preconditions to the antecedent of the latter, which we leave out for brevity.

To summarize, in SR-APL we formalize both declarative goals and plans uniformly in the same goal hierarchy specified by $\mathcal{D}$. We maintain the consistency of adopted declarative and procedural goals by ensuring that there is at least one path known to the agent over which all of her adopted declarative goals hold and that includes the concurrent execution of all of her adopted plans, possibly along with other actions. Whenever the agent's goals/plans become inconsistent due to some external interference, the successor state axiom for G in $\mathcal{D}$ will drop some of the adopted goals/plans if necessary, respecting their priority, and consistency of the goal-base is automatically restored. We also have a procedural goal-base Γ containing the adopted plans in $\mathcal{D}$, whose sole purpose is to ensure that the agent does not procrastinate w.r.t. her adopted plans. The set of transition rules of SR-APL allows an SR-APL agent to select, adopt, and execute plans from the plan library and thus serves as SR-APL's practical reasoning component.

While adopting plans and executing actions, we use a weak consistency check, and thus avoid searching over the entire plan-space while ensuring consistency. SR-APL also includes a repair rule that can be used to repair plans if the agent gets stuck (a) as a result of our weak consistency check (and lack of lookahead in plan selection), (b) due to external interferences, or (c) due to the existence of an adopted declarative goal for which there is no plan specified in the plan library.

Let us now define some useful notions of program execution in SR-APL. A *labeled execution trace $\mathcal{T}$ relative to a theory $\mathcal{D}$* is a (possibly infinite) sequence of configurations $\langle \Gamma_0, s_0 \rangle \stackrel{l_0}{\Rightarrow} \langle \Gamma_1, s_1 \rangle \stackrel{l_1}{\Rightarrow} \langle \Gamma_2, s_2 \rangle \stackrel{l_2}{\Rightarrow} \langle \Gamma_3, s_3 \rangle \stackrel{l_3}{\Rightarrow} \cdots$, s.t. $\Gamma_0 = [\text{nil}]$, $s_0 = S_0$ is the actual initial configuration, and for all $\langle \Gamma_i, s_i \rangle$, the agent level transition rule l_i can be used to obtain $\langle \Gamma_{i+1}, s_{i+1} \rangle$. Here l_i is one of A_{sel}, A_{step}, A_{exo}, A_{clean}, and A_{rep}, and in the absence of exogenous actions, l_i can be one of A_{sel}, A_{step}, A_{clean}, and A_{rep}. We sometimes suppress these labels. A *complete trace $\mathcal{T}$ relative to a theory $\mathcal{D}$* is a finite labeled execution trace relative to $\mathcal{D}$, $\langle \Gamma_0, s_0 \rangle \stackrel{l_0}{\Rightarrow} \cdots \stackrel{l_{n-1}}{\Rightarrow} \langle \Gamma_n, s_n \rangle$, s.t. $\langle \Gamma_n, s_n \rangle$ does not have an agent level transition, i.e. $\langle \Gamma_n, s_n \rangle \not\Rightarrow$.

For our blocks world example, we can show that our SR-APL agent for this domain will not adopt inconsistent plans as seen in Section 2 and will in fact achieve all her goals. Note that, when arbitrary exogenous actions can occur, even the best laid plans can fail. Here we only consider the case of where exogenous actions are absent. We model this using the following axiom, which we call $NoExo$: $\forall a.\ \neg exo(a)$. Given this, we can show that:

Proposition 1 (*a*). *There exists a complete trace $\mathcal{T}$ relative to $\mathcal{D}_{BW} \cup \{NoExo\}$ for our blocks world program.* (*b*). *For all such complete traces $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \Rightarrow \langle \Gamma_1, s_1 \rangle \Rightarrow \cdots \Rightarrow \langle \Gamma_n, s_n \rangle$, we have: $\mathcal{D}_{BW} \cup \{NoExo\} \models \text{Final}(\Gamma_n^{\|}, s_n) \wedge \text{Twr}_Y^G(s_n) \wedge \text{Twr}_R^B(s_n)$.* (*c*). *There are no infinite traces relative to $\mathcal{D}_{BW} \cup \{NoExo\}$.*

Thus when exogenous actions cannot occur, any execution of our SR-APL blocks world agent achieves all her goals.

6 Rationality of SR-APL Agents

We next prove some rationality properties that are satisfied by SR-APL agents. We only consider the case when exogenous actions do not occur. We could have considered exogenous actions, but in that case we would have to complicate the framework further, e.g. by assuming a fair environment that gives a chance to the agent to perform actions. Moreover, it is not obvious what rational behavior means in such contexts.

First of all, in each situation, for all domains $\mathcal{D}$ that are part of an SR-APL agent, the knowledge and c-goals/intentions as specified by $\mathcal{D}$ must be consistent:[7]

Theorem 3 (Consistency of Knowledge and CGoals).

$$\mathcal{D} \models \forall s.\ \neg\text{Know}(false, s) \wedge \neg\text{CGoal}(false, s).$$

[7] This follows independently from the underlying agent theory.

We can also show that the procedural goals in Γ and the declarative and procedural goals in the theory $\mathcal{D} \cup \{NoExo\}$ remain consistent. Let's say that *the procedural goals in Γ are consistent with those in the theory $\mathcal{D}$ in situation s in a configuration* $\langle \Gamma, s \rangle$ iff for all σ s.t. Member(σ, Γ), we have $\mathcal{D} \models$ CGoal(DoAL$(\sigma), s)$. Also, define $\mathcal{D}_{\bar{Exo}} \doteq \mathcal{D} \cup \{NoExo\}$. We have that:

Theorem 4 (Consistency of Γ and $\mathcal{D}_{\bar{Exo}}$). *If $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \Rightarrow \langle \Gamma_1, s_1 \rangle \Rightarrow \cdots \Rightarrow \langle \Gamma_n, s_n \rangle$ is a complete trace of an SR-APL agent w.r.t. a theory $\mathcal{D}_{\bar{Exo}}$, then for all i s.t. $0 \leq i < n$, we have:*
(a). If $s_{i+1} = do(a, s_i)$ for some a, then the procedural goals in Γ_i are consistent with those in the theory $\mathcal{D}_{\bar{Exo}}$ in s_i,
(b). If $s_i = s_{i+1}$, then there exists j s.t. $0 < i < j \leq n$ and the goals in Γ_j are consistent with those in the theory $\mathcal{D}_{\bar{Exo}}$ in s_j,
(c). The procedural goals in Γ_n are consistent with those in the theory $\mathcal{D}_{\bar{Exo}}$ in s_n.

(a) and (c) are self-explanatory. (b) shows that whenever there is some procedural goal in Γ_i that is not a goal w.r.t. the theory $\mathcal{D}_{\bar{Exo}}$, the A_{clean} rule will remove it from Γ_i, and eventually consistency is restored.[8] It follows from Theorem 4 that in all configurations $\langle \Gamma, s \rangle$ where the plans in Γ are consistent with those in the theory $\mathcal{D}_{\bar{Exo}}$ in s, the agent intends to execute the programs in Γ concurrently starting in s, possibly with other actions, i.e. $\mathcal{D}_{\bar{Exo}} \models$ CGoal$(\exists s'.$ DoAL$(\Gamma^{\|}, now, s'), s)$.

Finally, our agents evolve in a rational way:

Theorem 5 (Rationality of Actions in a Trace). *If $\mathcal{T} = \langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \langle \Gamma_1, s_1 \rangle \overset{l_1}{\Rightarrow} \cdots \overset{l_{n-1}}{\Rightarrow} \langle \Gamma_n, s_n \rangle$ is a trace of an SR-APL agent relative to a theory $\mathcal{D}_{\bar{Exo}}$, then for all i s.t. $0 < i \leq n$ and $s_i = do(a, s_{i-1})$, we have:*

(a). $\mathcal{D}_{\bar{Exo}} \models \neg$CGoal$(\neg\exists s'.$ Do$(a, now, s'), s_{i-1})$.
(b). *If* $l_{i-1} = A_{step}$ *then there exist* σ, σ' *s.t.* Member(σ, Γ_{i-1}) *and*
$\mathcal{D}_{\bar{Exo}} \models \langle \sigma, s_{i-1} \rangle \rightarrow \langle \sigma', do(a, s_{i-1}) \rangle \wedge$ CGoal$(\exists s'.$ DoAL$(a, now, s'), s_{i-1})$.
(c). $\mathcal{D}_{\bar{Exo}} \models \forall \phi, \psi, n.\ a = adoptRT(\psi, \phi) \vee a = adopt(\psi, n) \supset$
$\neg$CGoal$(\neg\exists s', p'.$ Starts$(s') \wedge$ Suffix$(p', do(a, s')) \wedge \psi(p'), s_{i-1})$.

This states that SR-APL is sound in the sense that any trace produced by the APL semantics is consistent with the agent's chosen goals. To be precise, (a) if an SR-APL agent performs the action a in situation s_{i-1}, then it must be the case that she does not have the intention not to execute a next in s_{i-1}. Moreover, (b) if a is performed via A_{step}, then a, which must have come from the procedural goal-base Γ, is indeed intended in s_{i-1} in the sense that she has the intention to execute a possibly along with some other actions next. Finally, (c) if a is the action of adopting a subgoal ψ w.r.t. a supergoal ϕ or that of adopting a goal ψ at some level n, then the agent does not have the c-goal in s_{i-1} not to bring about ψ next.

[8] Recall that applications of A_{clean} do not change the situation.

7 Discussion and Conclusion

Based on a "committed agent" variant of our rich theory of prioritized goal/subgoal dynamics [13], we have developed a specification of an APL framework that handles prioritized goals and maintains the consistency of adopted declarative and procedural goals. We also showed that an agent specified in this language satisfies some strong rationality properties. While doing this, we addressed some fundamental questions about rational agency. We model an agent's concurrent commitments by incorporating the DoAL construct in her adopted plans, which allows her to be open towards future commitments to plans, using a procedural goal-base Γ to prevent procrastination. We formalized a weak notion of consistency between goals and plans that does not require the agent to expand all adopted goals while checking for consistency.

While SR-APL agents rely on a user-specified plan library, they can achieve a goal even if such plans are not specified. Indeed the A_{rep} rule can be used as a first principles planner for goals that can be achieved using sequential plans. Thus, given a goal $\Diamond\Phi$, all the programmer needs to do to trigger the planner is to add a plan of the form $(\Diamond\Phi : true \leftarrow \Phi?)$ to the plan library Π. Since the program $\Phi?$ is neither executable nor final, it will eventually trigger the A_{rep} rule, which will make the agent adopt a sequence of actions to achieve Φ.

Here, we focused on developing an expressive agent programming framework that yields a rational/robust agent without worrying about tractability. Thus our framework is a specification and model of an ideal APL rather than a practical APL. In the future, we would like to investigate restricted versions of SR-APL that are practical, with an understanding of how they compromise rationality. We think this can be done. For instance if we assume a finite domain, then reasoning with the underlying theory should be decidable. We could adapt techniques from partial order planning such as summary information/causal links to support consistency maintenance. We could also simply find a global linear plan and cache it, using summary information to revise it when necessary. There are some controller synthesis techniques that can deal with temporally extended goals [18, 2].

Also, it would be desirable to study a version where the agent fully expands an abstract plan and checks its executability before adopting it. Finally, while our underlying agent theory supports arbitrary temporally extended goals, in SR-APL we only consider achievement goals. We would like to relax this in the future.

References

1. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA, USA, 1987.
2. D. Calvanese, G. De Giacomo, and M. Y. Vardi. Reasoning about Actions and Planning in LTL Action Theories. In *Proc. KR'02*, pages 593–602, 2002.
3. B. J. Clement and E. H. Durfee. Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. In *Proc. AAAI'99*, pages 495–502, 1999.
4. B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract Reasoning for Planning and Coordination. *J. of Artificial Intelligence Research*, 28:453–515, 2007.

5. M. Dastani. 2APL: A Practical Agent Programming Language. *J. of AAMAS*, 16(3):214–248, 2008.
6. G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121:109–169, 2000.
7. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent Programming with Declarative Goals. In *Intelligent Agents VII : Agent Theories, Architecture, and Languages*, vol. 1986 of LNAI, pages 228–243. Springer-Verlag, 2000.
8. K. V. Hindriks, W. van der Hoek, and M. B. van Riemsdijk. Agent Programming with Temporally Extended Goals. In *Proc. AAMAS'09*, pages 137–144, 2009.
9. J. F. Horty and M. E. Pollack. Evaluating New Options in the Context of Existing Plans. *Artificial Intelligence*, 127:199–220, 2001.
10. 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.
11. S. M. Khan. *Rational Agents : Prioritized Goals, Goal Dynamics, and Agent Programming Languages with Declarative Goals (in preparation)*. Ph.D. thesis, York University, Canada, 2011.
12. S. M. Khan and Y. Lespérance. ECASL: A Model of Rational Agency for Communicating Agents. In *Proc. AAMAS'05*, pages 762–769, 2005.
13. S. M. Khan and Y. Lespérance. A Logical Framework for Prioritized Goal Change. In *Proc. AAMAS'10*, pages 283–290, 2010.
14. S. M. Khan and Y. Lespérance. Towards a Rational Agent Programming Language with Prioritized Goals. In *Working Notes of DALT VIII*, pages 18–33, 2010.
15. S. M. Khan and Y. Lespérance. Prioritized Goals and Subgoals in a Logical Account of Goal Change – A Preliminary Report. In *Proc. DALT VII*, vol. 5948 of LNAI, pages 119–136, Springer-Verlag, 2010.
16. S. M. Khan and Y. Lespérance. SR-APL: A Model for a Programming Language for Rational BDI Agents with Prioritized Goals (Extended Abstract). In *Proc. AAMAS'11*, pages 1251–1252, 2011.
17. H. J. Levesque, F. Pirri, and R. Reiter. Foundations for a Calculus of Situations. *Electronic Transactions of AI (ETAI)*, 2(3–4):159–178, 1998.
18. M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-Deterministic Domains. In *Proc. IJCAI'01*, pages 479–484, 2001.
19. A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Agents Breaking Away*, vol. 1038 of LNAI, pages 42–55. Springer-Verlag, 1996.
20. R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
21. S. Sardiña, L. de Silva, and L. Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proc. AAMAS'06*, pages 1001–1008, 2006.
22. S. Sardiña and L. Padgham. A BDI Agent Programming Language with Failure Recovery, Declarative Goals, and Planning. *J. of AAMAS* (to appear), 2010.
23. R. Scherl and H. J. Levesque. Knowledge, Action, and the Frame Problem. *Artificial Intelligence*, 144(1–2):1–39, 2003.
24. S. Shapiro and G. Brewka. Dynamic Interactions Between Goals and Beliefs. In *Proc. IJCAI'07*, pages 2625–2630, 2007.
25. S. Shapiro, Y. Lespérance, and H. J. Levesque. Goal Change in the Situation Calculus. *J. of Logic and Computation*, 17(5):983–1018, 2007.
26. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference between Goals in Intelligent Agents. In *Proc. IJCAI'03*, pages 721–726, 2003.
27. M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Goals in Conflict: Semantic Foundations of Goals in Agent Programming. *J. of AAMAS*, 18(3):471–500, 2009.

28. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and Procedural Goals in Intelligent Agent Systems. In *Proc. KR'02*, pages 470–481, 2002.