Towards a Rational Agent Programming Language with Prioritized Goals

Shakil M. Khan and Yves Lespérance

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

Abstract. Most current approaches to agent programming languages with declarative goals only consider one goal at a time; while planning, they ignore other concurrent intentions of the agent, and as a consequence, the output of planning may not be consistent with the other intentions of the agent. In this paper, we develop a logic-based agent programming language with 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. Our framework is based on a rich theory of agency that models knowledge and prioritized goals, deals with temporally extended goals, formalizes goal dynamics, and handles subgoals. We show how agents programmed in our language satisfy some key rationality requirements.

1 Introduction

Recently, there has been a fair amount of work on establishing a link between agent logics and agent programming frameworks by incorporating declarative goals in agent programming languages/frameworks [3, 18, 14, 13, 17, 4]. 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. These declarative goals play an essential role for monitoring goal achievement and performing recovery when a plan has failed by decoupling plan failure/success from goal failure/success. Since these goals capture the reason for executing plans, its not hard to see that they are also necessary to perform rational deliberation and action, and to react in a rational way to changes in goals that result from communication.

Unfortunately, most of these frameworks suffer from various problems. For instance, some of these frameworks do not provide a formal semantics for declarative goals. Moreover, most do not formalize temporally extended goals and are restricted to achievement goals (e.g. [14]). Furthermore, most assume that all the goals of the agent are equally important (e.g. [3, 18, 13, 17]). The dynamics of these goals are usually specified using some sort of syntactic manipulation determined by the 'operational semantics' of the language. Generally, there is no requirement for an intended plan to be consistent with the intended declarative goals (e.g. in all of the frameworks mentioned above). Thus the execution of such an intended plan can render other contemporary intentions impossible to bring about.

One of the reasons for these deficiencies in these agent programming languages (APL) is the fact that it is quite challenging to formalize an APL that is sufficiently expressive and that can be implemented efficiently. The lack of proper formalizations of goals and their dynamics in the agent theory literature also contributes to this. Most existing APLs with declarative goals (APLwDG, henceforth) follow a similar pattern:

they start with an agent theory that has very little expressiveness, and specify an APL based on this theory. The inherited limited expressiveness of these APLs in turns contributes to the aforementioned deficiencies.

In this paper, we take a different approach : we start with an expressive agent theory, and provide a specification for an APL on top of this theory. In our agent theory, an agent can have multiple goals at different priority levels. We support the specification of general temporally extended goals, not just achievement goals, and specify how these goals evolve when actions/events occur and the agent's knowledge changes or when the agent adopts or drops a goal. We also handle subgoals and their dynamics and ensure that a subgoal is dropped when its supergoal becomes impossible or is dropped. Based on this rich theory, we then define a Simple Rational APL that deals with prioritized goals (SR-APL, henceforth), combining elements from belief-desire-intention APLs such as [11] and from the situation calculus-based ConGolog APL [2]. While doing this, we address some of the aforementioned problems of current APLwDG. In particular, our APL is grounded on a formal theory of goal change. We ensure that our agents' adopted declarative goals and procedural plans are consistent with each other. Thus, we investigate how an APL for a rational agent, i.e. one that conforms to a sound theory of rational agency, should ideally work. We then show that agents programmed in our language satisfy some key rationality requirements. In future work, we will try to restrict the expressiveness of this framework to improve its efficiency/tractability.

The paper is organized as follows: in the next section, we outline our base framework. In Section 3, we present our model of prioritized goals, give our formalization of goal dynamics, and discuss how subgoals change as a result of changes to their parent goals. In Section 4, we specify the semantics of SR-APL. In Section 5, we use a blocks world example to discuss how SR-APL compares to existing APLwDGs. In Section 6, we show that in the absence of external interference, a SR-APL agent behaves in ways that satisfy some key rationality principles. Then, we conclude by discussing previous work on APLwDGs, summarizing our results, and pointing to possible future work.

2 Preliminaries

Our base framework for modeling goal change is the situation calculus as formalized in [9, 12]. 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. Relations (and functions) whose truth values vary from situation to situation, are called relational (functional, resp.) fluents, and are denoted by predicate (function, resp.) 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.

We use a theory D that includes the following set of axioms: (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 [12], (3) initial state axioms describing what is true initially including the mental states of the agents, (4) unique name axioms for actions, and (5) domain-independent foundational axioms describing the structure of situations [9].

Following [15], 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: $\mathrm{Know}(\Phi, s) \overset{\mathrm{def}}{=} \forall s'.\ K(s', s) \supset \Phi(s')$. 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 [15] and some *informing* communicative actions [8]. As shown in [15], 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 [6]. A path is essentially an infinite sequence 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. We have a predicate $\mathrm{OnPath}(p, s)$, meaning that the situation s is on path p. Also, $\mathrm{Starts}(p, s)$ means that s is the starting situation of path p. A path p starts with s iff s is the earliest situation on p. See [6] for an axiomatization of these.

We will use $\Phi(s)$ to denote state formulae in the context of knowledge (and $\phi(p)$ for path formulae in that of goals) each of which has a free situation variable s (path variable p, resp.) in it. Where the intended meaning is clear, we sometimes suppress the situation variable (path variable) from Φ (ϕ, resp.). Note that, by incorporating infinite paths in our framework, we can evaluate goals over these and handle arbitrary temporally extended goals; thus, unlike some other situation calculus based accounts where goal formulae are evaluated w.r.t. finite paths (e.g. [16]), we can handle for example unbounded maintenance goals.

We next define some useful constructs. A state formula Φ *eventually holds* over the path p if Φ holds in some situation that is on p, i.e.: $\Diamond\Phi(p) \overset{\mathrm{def}}{=} \exists s'.\ \mathrm{OnPath}(p, s') \wedge \Phi(s')$. Other Temporal Logic operators can be defined similarly, e.g. always Φ: $\Box\Phi(p)$. Secondly, we define when a path p' is a suffix of another path p w.r.t. a situation s:

$$\mathrm{Suffix}(p', p, s) \overset{\mathrm{def}}{=} \mathrm{OnPath}(p, s) \wedge \mathrm{Starts}(p', s) \wedge \forall s'.\ s' \geq s \supset \mathrm{OnPath}(p, s') \equiv \mathrm{OnPath}(p', s').$$

Finally, $\mathrm{SameHist}(s_1, s_2)$ means that the situations s_1 and s_2 share the same history of action, but perhaps starting from different initial situations.

3 Our Formalization of Goals

In [6], we proposed a logical framework for modeling prioritized goal change. In that framework, an agent can have multiple goals/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. Our agents maximize their "utility". 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.

Prioritized Goals As in [6], we specify each *prioritized goal* or *p-goal* by its own 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 reflect the actions that have been performed so far. A smaller n represents higher priority, and the highest priority level is 0. Thus here we assume that the set of p-goals are totally ordered according to priority. We say that an agent has the p-goal that ϕ at level n in situation s iff ϕ holds over all paths that are G-accessible at n in s. To be able to refer to all the p-goals of the agent at some given priority level, we also define *only p-goals*. An agent has the only p-goal that ϕ at level n in situation s iff ϕ is a p-goal at n in s, and any path over which ϕ holds is G-accessible at n in s.

We allow the agent to have infinitely many goals. However in many cases, the modeler will want to specify a finite set of initial p-goals. When a finite number of prioritized goals is assumed, we use the function $NPGoals(s)$ to represent the number n of prioritized goals that the agent has in situation s. $NPGoals(s) = n$ holds iff n is the highest priority level such that for all $n' \geq n$, the agent has the trivial only p-goal at n' that the history of actions in s has happened.

We use a version of the blocks world for a running example, where each block can have one of four possible colors: blue, yellow, green, and red. We start with all blocks on the table. We assume that we only have a stacking action $stack(b, b')$ that can be used to move a block from the table onto another block. b can be stacked on b' in situation s if b and b' refer to two different blocks that are both clear in s, and if b is on the table in s. Since there are no unstacking actions, the agent cannot use a block to build 2 different towers at different times.

Assume that our blocks world domain consists of four blocks, $B_B, B_Y, B_G,$ and B_R, one of each color. Initially the agent knows the *color of* these blocks, and knows that all the blocks are *on the table* and are *clear*. Now assume that the agent has only two p-goals; at the highest priority level, she has the p-goal to eventually construct a 2 blocks tower that has a green block on top and a non-yellow block underneath, i.e. $\Diamond\text{TowerHP}^{G}_{\overline{Y}}$, where $\text{TowerHP}^{G}_{\overline{Y}} = \exists b, b'.\ \text{OnTable}(b') \wedge \text{On}(b, b') \wedge \neg\text{Yellow}(b') \wedge \text{Green}(b)$. At level 1 she has a similar p-goal as in level 0, but this time with a blue block on top and a non-red block underneath, i.e. $\Diamond\text{TowerLP}^{B}_{\overline{R}}$. 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*. We assume that our domain theory for the blocks world example D_{BW} includes the following:

$$\begin{aligned}
&\text{(a) Init}(s) \supset ((G(p, 0, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \Diamond\text{TowerHP}^{G}_{\overline{Y}}) \\
&\qquad\qquad \wedge (G(p, 1, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \Diamond\text{TowerLP}^{B}_{\overline{R}})), \\
&\text{(b) } \forall n, p, s.\ \text{Init}(s) \wedge n \geq 2 \supset (G(p, n, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s')).
\end{aligned}$$

(a) specifies the p-goals of the agent in the initial situations; (b) makes $G(p, n, s)$ true for every path p that starts with an initial situation for $n \geq 2$. Thus at levels $n \geq 2$, the agent has the trivial p-goal that she be in an initial situation.

An agent's c-goals or intentions 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. We say that an agent has the *realistic p-goal* that ϕ at level n in situation s (i.e. $\text{RPGoal}(\phi, n, s)$) iff ϕ holds over all G_R-accessible paths at n in s.

Using realistic p-goals, we next define *chosen goals* or *c-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 we want to compute the agent's c-goals such that it is the maximal consistent set of higher priority realistic p-goals. We do this iteratively starting with a set that contains the highest priority realistic p-goal. At each iteration we compute the intersection of this set with the next highest priority p-goal. If the intersection is not empty, we thus obtain a new chosen set of p-goals at level i. 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-goals 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, we repeat this until we reach $i = n$.

We say that an agent has a c-goal at some level n that ϕ (i.e. CGoal(ϕ, n, s)) if ϕ holds over all paths that are in the prioritized intersection of the set of G_R-accessible paths up to level n. We define c-goals in terms of c-goals at level n: the agent has the c-goal that ϕ (i.e. CGoal(ϕ, s)) if for any level n, ϕ is a c-goal at n. We can show that initially our agent has the p-goals/c-goals that $\Diamond\text{TowerHP}^G_Y$ and $\Diamond\text{TowerLP}^B_R$, i.e.: $D_{BW} \models \forall s.\ \text{Init}(s) \supset \text{CGoal}(\Diamond\text{TowerHP}^G_Y \wedge \Diamond\text{TowerLP}^B_R, s)$.

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. It can be shown that these constraints then continue to hold after any sequence of actions since they are preserved by the successor state axioms for K and G. See [5] 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. We introduce two 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 w.r.t. a supergoal ϕ, $adopt(\psi, \phi)$. When adopting a subgoal relative to a supergoal, the subgoal is automatically adopted at a priority level just below that of the parent goal.

We specify the dynamics of p-goals 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, we progress all p-goals to reflect the fact that this action has occurred. Secondly, to handle adoption of a p-goal ϕ at level m, we add a new proposition containing the p-goal to the agent's goal hierarchy at m. To be precise, in addition to progressing all p-goals at all levels, we insert a new level containing the p-goal that ϕ at m and push all current levels greater or equal to m one level down in the hierarchy. Finally, to handle the dropping of a p-goal ϕ, we replace the propositions that imply the dropped goal in the agent's goal hierarchy by the trivial proposition that the history of actions in the current situation has occurred, and thus the agent no longer has the p-goal that ϕ. See [6] 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 w.r.t. 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, we drop all the p-goals at all G-accessibility levels that imply ϕ including $\psi \wedge \phi$. Also, this means that dropping a subgoal does not necessarily drop the supergoal. 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. We also assume that 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. We say that ψ is a subgoal of ϕ in situation s (i.e. SubGoal(ψ, ϕ, s)) iff there exists 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 [7, 5] for details of our formalization of subgoals.

Prioritized Goals for Committed Agents Our formalization of prioritized goal dynamics in [6] 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 [6] displays an idealized form of rationality. 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 it serves 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. Our framework in [6] can be viewed as a theory of intention where the filter override mechanism is always triggered.

Note that, in this 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 or was dropped, but due to the fact that ψ has lower priority than and is inconsistent with the newly activated goal ϕ (see [6] 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" its 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 the framework in [6] that eliminates the filter override mechanism altogether so that agents' p-goals are dropped as soon as they become inactive. We can do this with the following simple changes to the framework in [6]: (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 this modified "committed agent" framework, an agent's p-goals are much more dynamic than before. On the other hand, her c-goals are now much more persistent than before, and are simply the consequential closure of her p-goals.

4 Agent Programming with Prioritized Goals

Our proposed agent programming language SR-APL combines elements from BDI APLs such as the AgentSpeak APL [11] and from the ConGolog APL [2]. 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, which is defined on top of the situation calculus. 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 $\mathrm{Trans}(\sigma, s, \sigma', s')$ to denote that program σ when executed starting in situation s can reach situation s' in one elementary step with the program σ' remaining to be executed, and $\mathrm{Final}(\sigma, s)$ to mean that the program σ may legally terminate in situation s. Finally, in the following $\mathrm{Do}(\sigma, s, s')$ means that program σ when executed starting in situation s can legally terminate in situation s'. See [2] for details of how these constructs are defined.

Components of SR-APL First of all, we have a theory specifying actions that can be done, the initial knowledge and goals of the agent, and their dynamics, as discussed in the previous section. Moreover, we also have a plan-base 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 chosen 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 [2] and includes the empty program nil, primitive actions a, waiting for a condition $\Phi?$, sequences $(\sigma_1; \sigma_2)$, and the special action for subgoal adoption, $adopt(\Diamond\Phi, \sigma)$; here $\Diamond\Phi$ is a subgoal in a plan that includes this adopt action, and σ is what remains of the plan after this adopt action has been performed. 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.

Returning to our blocks world example, we can use the plan-base Π below, which has only two rules:

1. $\Diamond\mathrm{TowerHP}^G_{\bar{Y}} : [\mathrm{OnTable}(b) \wedge \mathrm{OnTable}(b') \wedge b \neq b' \wedge \mathrm{Clear}(b) \wedge \mathrm{Clear}(b')$
   $\wedge\, \mathrm{ColorOf}(b) \neq \mathrm{Yellow} \wedge \mathrm{ColorOf}(b') = \mathrm{Green}] \leftarrow stack(b', b),$
2. $\Diamond\mathrm{TowerLP}^B_{\bar{R}} : [\mathrm{OnTable}(b) \wedge \mathrm{OnTable}(b') \wedge b \neq b' \wedge \mathrm{Clear}(b) \wedge \mathrm{Clear}(b')$
   $\wedge\, \mathrm{ColorOf}(b) \neq \mathrm{Red} \wedge \mathrm{ColorOf}(b') = \mathrm{Blue}] \leftarrow stack(b', b).$

This says that if the agent has the goal to have a green and non-yellow tower and knows about a non-yellow block b and a green 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.

Semantics of SR-APL Before giving the formal semantics of SR-APL, let us go over some useful notations. We will use various standard list operations, e.g. nil (representing the empty list), 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), Remove (for removing a given item from a list), Replace (for replacing a given item with another item in a list), etc.

A SR-APL agent can work on multiple goals at the same time, i.e. can interleave hierarchical decomposition of goals by committing to subgoals/plans, and execution of plans for multiple goals. Thus at any time, an agent might be committed to several plans that she will execute in an interleaved fashion. One way of specifying an agent's commitment to execute a plan σ next is to say that she has the intention that $\mathrm{Starts}(s) \wedge \exists s'.\ \mathrm{OnPath}(s') \wedge \mathrm{Do}(\sigma, s, s')$, 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 interleaving of actions in plans, since Do requires that the agent execute σ before executing any other actions/plans.

A better alternative is to represent the procedural goal as $\mathrm{Starts}(s) \wedge \exists s'.\ \mathrm{OnPath}(s') \wedge \mathrm{DoAtleast}(\sigma, s, s')$, which says that the agent has the intention to execute at least the program σ next, and possibly more. DoAtleast(σ, s, s') holds if there is an execution of program σ, possibly interleaved with other actions by the agent herself, starting in situation s and terminating in s', which we define as:[1]

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

However, a new problem with this approach is that it allows the agent to procrastinate in the execution of the intended plans. For instance, suppose that the agent has the goal at 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 DoAtleast, 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. The "other actions" at level n_1 (n_2, resp.) are meant to be actions from the plan σ_2 (σ_1, resp.). However, nothing requires that the additional actions that the agent might execute are indeed from σ_1/σ_2, and thus this allows her to perform actions that are unnecessary as long as they do not perturb the execution of σ_1/σ_2.

To avoid this, we include an additional component, a *procedural intention-base Γ*, to the declarative specification of SR-APL. Γ is a list of plans that the agent has adopted and 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: $\Gamma^{\|} \overset{\text{def}}{=}$ **if** $(\Gamma = \text{nil})$ **then** nil **else** 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 $\langle nil, S_0 \rangle$.

The semantics of SR-APL are defined by transition rules that specify how an agent may evolve from one configuration to another. We have a two-tier transition system. Object/program level transition rules specify how a program written in our plan language may evolve. On top of this, we use meta/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 [2] limited to our plan language; see [2] for details. We use $\langle \sigma, s \rangle \rightarrow \langle \sigma', s' \rangle$ as an abbreviation for Trans(σ, s, σ', s') in the ConGolog definition.

Agent-Level Transition Rules These transition rules are given in Table 1 and are similar to those of a typical BDI APL.[2] First of all, we have a rule A$_{sel}$ for selecting and adopting a plan using the rule-base 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 previously unhandled realistic p-goal at some level 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 DoAtleast$(\sigma\theta)$ w.r.t. $\Diamond\Phi$ next, then she can adopt the plan $\sigma\theta$, adding DoAtleast$(\sigma\theta)$ as a subgoal of $\Diamond\Phi$ to her goals,

[1] We will use this construct to specify the procedural goals of an agent agt. Note that, while our theory accommodates 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.

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

and adding $\sigma\theta$ to Γ (here Handled(ϕ, s) is defined as $\exists\psi$. SubGoal(ψ, ϕ, s)).[3]

A$_{sel}$

$$\frac{\begin{array}{c}\text{Member}(\Diamond\Phi : \Psi \leftarrow \sigma, \Pi), \quad D \models \text{RPGoal}(\Diamond\Phi, n, s), \\ D \models \neg\text{Handled}(\Diamond\Phi, s) \wedge \text{Know}(\Psi', s), \quad \text{mgu}(\Psi, \Psi') = \theta, \\ D \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adopt(\text{DoAtleast}(\sigma\theta), \Diamond\Phi), now, s'), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\sigma\theta, \Gamma), do(adopt(\text{DoAtleast}(\sigma\theta), \Diamond\Phi), s)\rangle}$$

A$_{step}$

$$\frac{\begin{array}{c}\text{Member}(\sigma, \Gamma), \quad D \models \text{RPGoal}(\text{DoAtleast}(\sigma), n, s), \\ D \models \langle \sigma, s\rangle \rightarrow \langle \sigma', do(a, s)\rangle \wedge \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s)\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Replace}(\sigma, \sigma', \Gamma), do(a, s)\rangle}$$

A$_{exo}$

$$\frac{D \models \text{Exo}(a) \wedge \text{Poss}(a, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \Gamma, do(a, s)\rangle}$$

A$_{clean}$

$$\frac{\text{Member}(\sigma, \Gamma), \quad D \models \neg\exists n.\ \text{RPGoal}(\text{DoAtleast}(\sigma), n, s)}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Remove}(\sigma, \Gamma), s\rangle}$$

A$_{repair}$

$$\frac{\begin{array}{c}D \models \neg\exists s'.\ \langle \Gamma^{\|}, s\rangle \rightarrow \langle \Gamma', s'\rangle, \quad D \models \neg\text{Final}(\Gamma^{\|}, s), \\ \text{For all } \sigma \text{ s.t. Member}(\sigma, \Gamma) \text{ we have:} \\ D \models \text{RPGoal}(\text{DoAtleast}(\sigma), s) \wedge \text{Handled}(\text{DoAtleast}(\sigma), s), \\ D \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adopt(\text{Do}(\vec{a}), NPGoals(s)), now, s'), s), \\ D \models \text{Agent}(\vec{a}) = agt \wedge \text{Do}(\vec{a}, s, s') \wedge \langle \Gamma^{\|}, s'\rangle \rightarrow \langle \Gamma', s''\rangle\end{array}}{\langle \Gamma, s\rangle \Rightarrow \langle \text{Cons}(\vec{a}, \Gamma), do(adopt(\text{Do}(\vec{a}), NPGoals(s)), s)\rangle}$$

Table 1. Agent Transition Rules

In our framework, it can be shown that if an agent does not have the chosen goal in s not to adopt a subgoal ψ w.r.t. a supergoal ϕ, then she does not have the chosen goal that $\neg\psi$ next in s, i.e.:

Theorem 1. $D \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(adopt(\psi, \phi), now, s'), s) \supset$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\psi, \phi), s')) \wedge \psi(p'), s).$$

Theorem 1 and condition (e) above imply that the agent does not have the chosen goal not to execute $\sigma\theta$ concurrently with $\Gamma^{\|}$ and possibly other actions next, i.e.: (i). $\neg\text{CGoal}(\neg\exists s', s''.\ \text{Do}(adopt(\text{DoAtleast}(\sigma\theta), \Diamond\Phi), now, s') \wedge \text{DoAtleast}(\sigma\theta \parallel \Gamma^{\|}, s', s''), s)$. 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 p-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. $D \models \exists n.\ \text{RPGoal}(\phi, n, s) \wedge$

$$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\psi, \phi), s')) \wedge \psi(p'), s)$$
$$\supset \text{CGoal}(\psi, do(adopt(\psi, \phi), s)).$$

From (b), (i), and Theorem 2, we have that: (ii). $\text{CGoal}(\exists s'.\ \text{DoAtleast}(\sigma\theta \parallel \Gamma^{\|}, now, s'), do(adopt(\text{DoAtleast}(\sigma\theta), \Diamond\Phi), s))$. (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 adopt action has

[3] Note that, following [17], we allow the agent to adopt plans for parts of her goals.

happened. Note that this notion of consistency is a weak one, since it does not guarantee that there is an execution of the program $(\sigma\theta \parallel \Gamma^{\parallel})$ after the adopt action happens, but rather ensures that the program DoAtleast$(\sigma\theta \parallel \Gamma^{\parallel})$ 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) DoAtleast(σ) is a realistic p-goal at some level 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 can be updated to reflect this.

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 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 $\neg\mathrm{CGoal}(\neg\exists s'.\ \mathrm{Do}(a; (\sigma' \parallel \Gamma^{\parallel}), now, s'), s)$. 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 block's world domain who has a goal to eventually build a 3 blocks tower, i.e. $\Diamond$3Tower, where 3Tower $= \exists b, b', b''.\ \mathrm{OnTable}(b) \wedge \mathrm{On}(b', b) \wedge \mathrm{On}(b'', b')$. Also, in addition to the above rules, her rule-base Π includes the following rule:

$$\Diamond\text{3Tower} : [\mathrm{OnTable}(b) \wedge \mathrm{OnTable}(b') \wedge \mathrm{OnTable}(b'') \wedge b \neq b' \wedge \mathrm{Clear}(b) \wedge \mathrm{Clear}(b')$$
$$\wedge\ \mathrm{Clear}(b'') \wedge \mathrm{ColorOf}(b) \neq \mathrm{Yellow} \wedge \mathrm{ColorOf}(b') = \mathrm{Green} \wedge \mathrm{ColorOf}(b'') = \mathrm{Yellow}] \leftarrow \sigma_1,$$
$$\text{where } \sigma_1 = adopt(\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}, \mathrm{DoAtleast}(\sigma_2)); \sigma_2,$$
$$\text{and } \sigma_2 = \mathrm{TowerHP}^{G}_{\bar{Y}}?; stack(b'', b').$$

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 $adopt(\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}, \mathrm{DoAtleast}(\sigma_2))$ action in her plan σ_1 in Γ, the agent adopts $\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}$ as a subgoal w.r.t. the parent goal of executing the remaining program of σ_1 (i.e. of the progression of σ_1, which in this case is σ_2), possibly along with other actions, i.e. w.r.t DoAtleast(σ_2). Then in the second step, she uses the A_{sel} rule to select and adopt a plan for the subgoal $\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}$. We assume that the subgoal $\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}$ 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 $\Diamond\mathrm{TowerHP}^{G}_{\bar{Y}}$. This can be specified by the programmer by having the supergoal σ_2 start with the test/wait action TowerHP^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 DoAtleast construct) might not have a complete execution as it might get blocked when it reaches TowerHP^G_Y?. Moreover, before a plan has been adopted for handling $\Diamond$TowerHP^G_Y, the execution of $\Gamma^{\|}$ might get blocked due to the fact that σ_2 is a member of Γ. In other words, $\Gamma^{\|}$ will have a complete execution only when all the subgoals in Γ have been fully expanded. Thus, in general for any procedural plan-base Γ in which there is a program σ s.t. σ allows subgoal expansion of some goal $\Diamond\Phi$, $\Gamma^{\|}$ might not have a complete program level execution in a given configuration, since there might be no currently adopted plan in Γ that can be used to achieve Φ. To deal with this, we use a weak consistency check that does not perform full lookahead over $\Gamma^{\|}$. However, our semantics ensures that any action a performed by the agent must be consistent with DoAtleast$(\Gamma^{\|})$, since A$_{step}$ requires that doing a must be consistent with all her DoAtleast procedural goals in her goal hierarchy, i.e. that $\neg$CGoal$(\neg\exists s'.$ Do$(a, now, s'), s)$. Note that, since we do not perform full lookahead, it is possible that the agent might get stuck, e.g. if she adopts a plan that includes the subgoal $\Diamond\Phi$, but there is no applicable plan rule in her plan-base Π that achieves Φ. This is a common problem in many BDI APLs, and can be solved using the usual techniques for plan failure handling [18]. In future work, we will examine how to avoid this by incorporating full lookahead in SR-APL.

Thirdly, we have a rule A$_{exo}$ to accommodate exogenous actions, i.e. actions occurring in the agent's environment that are not under her control. It states that when an executable exogenous action a occurs in s, the agent must update her goals by progressing the situation component of her configuration to $do(a, s)$.

Fourthly, the A$_{clean}$ rule synchronizes the procedural goal-base Γ and its declarative counter-part (i.e. the hierarchy of goals/plans specified by D). This might be required when the occurrence of an exogenous action forces the agent to drop a realistic p-goal/plan by making it impossible to achieve/execute or inconsistent with her higher priority realistic p-goals. Recall that such an impossible/inconsistent plan is automatically dropped from the agent's set of p-goals (by the SSA for G). We use A$_{clean}$ to clean up the procedural goal-base Γ by dropping such programs from Γ that are no longer intended. 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.

Finally, we have a rule A$_{repair}$ that allows the agent to repair her plans in case she gets stuck, i.e. when for all programs σ in Γ, the agent has the realistic p-goal that DoAtleast(σ) at some level n (and thus all of these DoAtleast(σ) are still individually executable and collectively consistent), but together they are not concurrently executable without any 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$_{repair}$ 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 $NPGoals(s)$).

As mentioned above, the agent could get stuck due to the occurrence of an exogenous action e, for instance when e makes the preconditions of some plan σ in $\mathit{\Gamma}$ false; note that, DoAtleast(σ) might still be executable after the occurrence of e, e.g. if there is an action r (encoded by the DoAtleast construct) that can be used to restore the preconditions of σ.

Moreover this could also happen when exogenous actions are absent, since we only perform a partial lookahead when executing actions via A_{step}. Consider the following example: assume that we have only 3 actions in this domain, $a, b,$ and r. All of these actions are initially possible. The execution of b makes the precondition of a false. On the other hand, the execution of r restores the precondition of a. Suppose that an agent has two adopted plans, DoAtleast(a) at level 0 and DoAtleast(b) at level 1. Thus $\mathit{\Gamma}$ is of the form $[a, b]$. Its easy to see that $b; a$ is not a valid execution of $\mathit{\Gamma}^{\|}$, since the execution of b makes the preconditions of a false. But $b; r; a$ is indeed a valid execution of $(\text{DoAtleast}(a) \wedge \text{DoAtleast}(b))$. Since we only do partial consistency checking, our semantics allows the agent to perform b as the first action.[4] In other words, 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, i.e. with $(\text{DoAtleast}(a) \wedge \text{DoAtleast}(b))$, both of which hold. After the execution of b, the agent will get stuck, as there is no action in the progression of $\mathit{\Gamma}$ that she can perform. To deal with this, we include a repair rule that makes the agent plan for and commit to a sequence of actions that can be used to repair $\mathit{\Gamma}$, which for our example is r. Note that, we could have avoided 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 $\mathit{\Gamma}$. 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 [18]. Finally, note that 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, we believe that there must be a repair plan available to the agent 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 DoAtleast(a) and DoAtleast(b) as her realistic p-goals, $\mathit{\Gamma} = [a, b]$, and if she has the c-goal not to execute an action from $\mathit{\Gamma}^{\|}$ (i.e. $\text{CGoal}(\neg\exists s'.\ \langle \mathit{\Gamma}^{\|}, now\rangle \rightarrow \langle \mathit{\Gamma}', s'\rangle, s)$), then it must be the case that she does not have the c-goal not to execute $\mathit{\Gamma}^{\|}$ along with other actions (e.g. r), i.e. $\neg\text{CGoal}(\neg\exists s'.\ \text{DoAtleast}(a\|b, now, s'), s)$. Otherwise, one of DoAtleast(a) or DoAtleast(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 $\mathit{\Gamma}$. 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 DoAtleast 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.

[4] 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.

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.

Now that we have specified the semantics of SR-APL, let us define some useful notions of program execution in SR-APL. A *labeled execution trace* $\mathcal{T}$ is a (possibly infinite) sequence of configurations $\langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \langle \Gamma_1, s_1 \rangle \overset{l_1}{\Rightarrow} \langle \Gamma_2, s_2 \rangle \overset{l_2}{\Rightarrow} \langle \Gamma_3, s_3 \rangle \overset{l_3}{\Rightarrow} \cdots$, 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$_{repair}$, and in the absence of exogenous actions, l_i can be one of A$_{sel}$, A$_{step}$, or A$_{repair}$. We sometimes suppress these labels.

A *complete trace* $\mathcal{T}$ is a finite labeled execution trace $\langle \Gamma_0, s_0 \rangle \overset{l_0}{\Rightarrow} \cdots \overset{l_{n-1}}{\Rightarrow} \langle \Gamma_n, s_n \rangle$, s.t. $\langle \Gamma_n, s_n \rangle$ does not have an agent level transition, i.e. $\langle \Gamma_n, s_n \rangle \not\Rightarrow$, and the concurrent execution of the programs in Γ_n is final in s_n, i.e. $D \models \text{Final}(\Gamma_n^{\|}, s_n)$.

5 An Example

In this section, we use our blocks world example to discuss how our proposed APL compares to existing APLwDGs. Consider a typical BDI APLwDG. In this APL, there is a procedural goal-base Γ that is a list of plans that the agent is committed to execute. In addition, there is a declarative goal-base that is a list of declarative achievement goals that the agent is committed to bring about. Unlike in SR-APL, the rules in this APL just select plans for the agent's goals and eventually execute them in an attempt to achieve her goals (it should be possible to specify such an APL as a variant of SR-APL with modified semantic rules). We claim that such an APL is not always sound and rational. Now, note that one way of building a blue non-red (and a green non-yellow) tower is to construct a blue-green (a green-red, resp.) tower. While these two plans are individually consistent, they are inconsistent with each-other, since the agent has only one green block. Thus a rational agent should not consider adopting the plan of building a blue-green tower. However, we claim that the following would be a legal trace of a blocks world domain in such an APL (here, we assume that SR-APL and this APL share the same rule names and configuration elements):

$$\langle nil, s_0 \rangle \overset{A_{sel}}{\Rightarrow} \langle [stack(B_B, B_G)], s_1 \rangle \overset{A_{sel}}{\Rightarrow} \langle [stack(B_G, B_R), stack(B_B, B_G)], s_2 \rangle\rangle \overset{A_{step}}{\Rightarrow}$$
$$\langle [stack(B_G, B_R), nil], do(stack(B_B, B_G), s_2) \rangle.$$

where, $s_1 = do(adopt(\text{DoAtleast}(stack(B_B, B_G)), \Diamond\text{TowerLP}^B_{\bar{R}}), s_0)$, and $s_2 = do(adopt(\text{DoAtleast}(stack(B_G, B_R)), \Diamond\text{TowerHP}^G_{\bar{Y}}), s_1)$. The above trace ends in a configuration where the agent is stuck and cannot complete successfully. Thus, in this framework, not only the agent is 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 the higher priority goal $\Diamond\text{TowerHP}^G_{\bar{Y}}$ 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. We believe that non-reversible actions are common in real world domains. But, even if we assume that the effects of actions can always be reversed, the agent might have a deadline and thus reusing the same block to build two towers could render some of her goals impossible. Note that, while one could argue that such irrational behavior could 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), we think that this puts an excessive burden on the agent programmer. Ideally, such reasoning about goals should be delegated to the agent. It should be noted that, to the best of our knowledge, all existing APLwDGs suffer from this problem.

In contrast, it can be shown that our SR-APL agent for this blocks world domain will not adopt such inconsistent plans 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 will only consider the case of where exogenous actions are absent. To model this, we use the following axiom, which we call $NoExo$: $\forall a.\ \neg exo(a)$. Thus:

Proposition 1 *(a). There exists a complete trace $\mathcal{T}$ for our blocks world program. (b). For all complete traces $\mathcal{T} = \langle \mathit{\Gamma}_0, s_0\rangle \Rightarrow \langle \mathit{\Gamma}_1, s_1\rangle \Rightarrow \cdots \Rightarrow \langle \mathit{\Gamma}_n, s_n\rangle$, we have:*

$$D_{BW} \cup NoExo \models \text{TowerHP}^{G}_{Y}(s_n) \wedge \text{TowerLP}^{B}_{R}(s_n).$$

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

In this section, we prove some properties that are preserved by an SR-APL agent, and that together ensure the rationality of the agent. Once again, we will 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 the context of external interferences.

To this end, we first show that in each situation, for all domains D that are part of a SR-APL agent, the knowledge and goals as specified by D must be consistent:[5]

Theorem 3 (Consistency of Knowledge and CGoals).

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

We next show that in the absence of exogenous actions, for each configuration of a SR-APL agent trace, the procedural component $\mathit{\Gamma}$ is consistent with the declarative and procedural goals in D:

Theorem 4 (Consistency of $\mathit{\Gamma}$ and $D \cup NoExo$). *If $\mathcal{T} = \langle \mathit{\Gamma}_0, s_0\rangle \Rightarrow \langle \mathit{\Gamma}_1, s_1\rangle \Rightarrow \cdots \Rightarrow \langle \mathit{\Gamma}_n, s_n\rangle$ is a trace of a SR-APL agent, then for all i s.t. $0 < i \leq n$, and for all σ, if $\text{Member}(\sigma, \mathit{\Gamma}_i)$, then we have: $D \cup NoExo \models \text{CGoal}(\text{DoAtleast}(\sigma), s_i)$.*

Thus for any configuration $\langle \mathit{\Gamma}_i, s_i\rangle$, the agent intends to execute each program σ of $\mathit{\Gamma}_i$ in s_i, possibly interleaved by other actions. It follows that the agent intends to execute the programs in $\mathit{\Gamma}_i$ concurrently starting in s_i, possibly with other actions, i.e. $D \cup NoExo \models \text{CGoal}(\exists s'.\ \text{DoAtleast}(\mathit{\Gamma}_i^{\|}, now, s'), s_i)$.

We next show that our agent evolves in a rational way, i.e. in the absence of exogenous actions, the actions produced by a SR-APL agent are consistent with her knowledge and goals in the theory D:

Theorem 5 (Rationality of Actions in a Trace). *If $\mathcal{T} = \langle \mathit{\Gamma}_0, s_0\rangle \overset{l_0}{\Rightarrow} \langle \mathit{\Gamma}_1, s_1\rangle \overset{l_1}{\Rightarrow} \cdots \overset{l_{n-1}}{\Rightarrow} \langle \mathit{\Gamma}_n, s_n\rangle$ is a trace of a SR-APL agent, then for all i s.t. $0 < i \leq n$ and $s_i = do(a, s_{i-1})$, we have:*

[5] Note that, this independently follows from the underlying agent theory. Also, see [6] for a list of other desirable properties of our formalization of prioritized goals.

(a). $D \cup NoExo \models \neg\text{CGoal}(\neg\exists s'.\ \text{Do}(a, now, s'), s_{i-1})$.

(b). If $l_{i-1} = A_{step}$ then $D \cup NoExo \models \text{CGoal}(\exists s'.\ \text{DoAtleast}(a, now, s'), s_{i-1})$.

(c). $D \cup NoExo \models \forall\phi, \psi, n.\ a = adopt(\psi, \phi) \vee a = adopt(\psi, n) \supset$
$\neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(a, s')) \wedge \psi(p'), s_{i-1})$.

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 a 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 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. Note that, Theorem 5(a) is a general property and applies for any type of transitions. 5(c) applies to actions performed via A$_{sel}$ and A$_{repair}$; moreover, it also applies to $adopt$ actions performed via A$_{step}$. Finally, 5(b) is a stronger property than 5(c) for A$_{step}$ and applies to any actions performed via this rule.

7 Related Work, Discussion, and Future Work

There has been much work on agent programming languages with declarative goals where the dynamics of goals and intentions and the dependencies between goals and subgoals are modeled (e.g. [18] and the references therein). However, most of these only handle achievement goals and do not provide a semantic formalization of goal dynamics. Since these do not support temporally extended goals, they often need to accommodate inconsistent desire-bases to allow the agent to achieve conflicting goals at different time points; e.g., in [17], the authors formalized two semantics for representing such conflicting goals/desires using propositional and default logic. Unlike us however, they do not handle goals with different priorities. In [14], the authors present a situation calculus based APL where the agent executes a program while maximizing the achievement of a (possibly inconsistent) set of prioritized goals. However, they do not formalize goal dynamics and goals are not modeled as mental attitudes. In [4], the authors present an extension of the GOAL APL [3] to incorporate temporally extended goals. In this account, the agent's beliefs consist of a (complete) valuation of atomic propositions and a set of LTL formulae that encode the action theory and that includes action precondition axioms and successor state axioms. They assume that the agent's beliefs Σ are a subset of her goals Γ. While they specify progression of beliefs semantically using the aforementioned action theory, the progression of goal formulae χ s.t. $\chi \in \Gamma \backslash \Sigma$ is defined by using a syntactic transformation of LTL formulae. In contrast, we specify goal dynamics using a semantic approach. As in the GOAL APL, this language also specifies (primitive) action selection rules that based on some mental state conditions, allow the agent to concurrently execute a set of primitive actions in each state. Thus it does not allow complex plans or hierarchical decomposition of goals to be directly specified. To the best of our knowledge, none of these APLs maintains consistency between chosen declarative goals and adopted procedural goals.

In this paper, we first presented a variant of our theory of prioritized goal dynamics [6] that is suitable for agents that commit strongly to their intentions. Then based on this theory, we developed a specification of an APL that handles prioritized goals and maintains the consistency of adopted declarative and procedural goals. We showed that an agent specified in this language satisfies some strong rationality properties.

Our proposed APL was designed in the spirit of BDI APLs; agents in SR-APL rely on a user-specified plan library, and interleave plan selection, hierarchical plan decomposition, and plan execution. Note also that our agents can achieve a goal even if such user-defined plans are not available. Indeed the A_{repair} 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-base Π.

In this paper, we have focused on developing an expressive agent programming framework that yields a rational/robust agent without worrying about tractability. Thus our framework is more of a specification of an ideal APL rather than a practical APL. In the future, we would like to investigate restricted versions of SR-APL that improve its efficiency/tractability. 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 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 Univ. Press, MA, 1987.
2. 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.
3. 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*, *LNAI v. 1986*, pp. 228–243, Springer, 2000.
4. K. V. Hindriks, W. van der Hoek, and M. B. van Riemsdijk. Agent Programming with Temporally Extended Goals. In *AAMAS-09*, pp. 137–144, 2009.
5. S. M. Khan. *Rational Agents : Prioritized Goals, Goal Dynamics, and Agent Programming Languages with Declarative Goals (in preparation)*. PhD thesis, York Univ., Canada, 2010.
6. S. M. Khan and Y. Lespérance. A Logical Framework for Prioritized Goal Change. To appear in *AAMAS-10*, 2010.
7. S. M. Khan and Y. Lespérance. Prioritized Goals and Subgoals in a Logical Account of Goal Change – A Preliminary Report. In *DALT-09* , *LNAI v. 5948*, pp. 119–136. Springer, 2010.
8. S. M. Khan and Y. Lespérance. ECASL: A Model of Rational Agency for Communicating Agents. In *AAMAS-05*, pp. 762-769, Utrecht, The Netherlands, 2005.
9. 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.
10. F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *J. of ACM*, 46(3):325–361, 1999.
11. A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Agents Breaking Away*, *LNAI v. 1038*, pp. 42–55. Springer-Verlag, 1996.
12. R. Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
13. S. Sardina, L. de Silva, and L. Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *AAMAS-06*, pp. 1001–1008, Hakodate, Japan, 2006.
14. S. Sardina and S. Shapiro. Rational Action in Agent Programs with Prioritized Goals. In *AAMAS-03*, pp. 417–424, Melbourne, Australia, 2003.
15. R. Scherl and H. Levesque. Knowledge, Action, and the Frame Problem. *Artificial Intelligence*, 144(1–2), 2003.
16. S. Shapiro and G. Brewka. Dynamic Interactions Between Goals and Beliefs. In *IJCAI-07*, pp. 2625–2630, India, 2007.
17. 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.
18. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and Procedural Goals in Intelligent Agent Systems. In *KR&R-02*, pp. 470–481, Toulouse, France, 2002.