An Operational Semantics for a Fragment of PRS

Lavindra de Silva[1], **Felipe Meneguzzi**[2] and **Brian Logan**[3]
[1] Institute for Advanced Manufacturing, University of Nottingham, Nottingham, UK
[2] Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
[3] School of Computer Science, University of Nottingham, Nottingham, UK
lavindra.desilva@nottingham.ac.uk, felipe.meneguzzi@pucrs.br, brian.logan@nottingham.ac.uk

Abstract

The Procedural Reasoning System (PRS) is arguably the first implementation of the Belief–Desire–Intention (BDI) approach to agent programming. PRS remains extremely influential, directly or indirectly inspiring the development of subsequent BDI agent programming languages. However, perhaps surprisingly given its centrality in the BDI paradigm, PRS lacks a formal operational semantics, making it difficult to determine its expressive power relative to other agent programming languages. This paper takes a first step towards closing this gap, by giving a formal semantics for a significant fragment of PRS. We prove key properties of the semantics relating to PRS-specific programming constructs, and show that even the fragment of PRS we consider is strictly more expressive than the plan constructs found in typical BDI languages.

1 Introduction

The Procedural Reasoning System (PRS) [Georgeff and Ingrand, 1989; Georgeff and Lansky, 1987; 1986] is generally recognised as one of the first implementations of the Belief–Desire–Intention (BDI) [Bratman, 1987] model of agency and practical reasoning. PRS has been extremely influential, and is still widely used [Ghallab *et al.*, 2016], particularly in robotics, e.g., [Ingrand *et al.*, 1996; Alami *et al.*, 1998; Foughali *et al.*, 2016; Niemueller *et al.*, 2017; Lemaignan *et al.*, 2017]. For example, PRS-based systems secured first and second places in recent RoboCup and ICAPS logistics competitions. PRS has also influenced the design of many subsequent BDI-based agent programming languages, e.g., [Rao, 1996; Huber, 1999; Busetta *et al.*, 1999; Winikoff *et al.*, 2002; Morley and Myers, 2004; Sardina *et al.*, 2006; Sardina and Padgham, 2011], though most of these languages implement only a subset of the programming language features supported by PRS. Surprisingly, given its centrality in the BDI paradigm, PRS lacks a formal operational semantics, which makes it difficult to determine its expressive power relative to other BDI agent programming languages, or to verify the correctness of PRS programs. For example, there is a widespread "folk belief" in the agent programming community that the plan graphs used by PRS are more expressive than most if not all other BDI agent formalisations, yet no proof of this intuition exists.

In this paper, we give a formal semantics for a significant fragment of PRS. We focus on language features specific to PRS, and in particular, its graph-based representation for plans and its programming constructs for maintaining a condition (i.e., maintenance goals). We make three main contributions. First, we develop a formalisation for the syntax of PRS as a directed bipartite graph (Section 2). Second, we provide an operational semantics that accounts for graph-based plans and adopting, suspending, resuming, and aborting (nested) maintenance goals (Section 3). Third, we prove key properties of the semantics of these PRS-specific programming constructs, and show that PRS plan-graphs are strictly more expressive than the plan rules found in typical BDI languages (Section 4). In Section 5, we briefly discuss related work and conclude.

2 PRS Syntax

We briefly recall the syntax and deliberation cycle of PRS, as defined in [Ingrand, 1991]. In the interests of brevity, we omit some features of the language, including meta-level reasoning, 'true concurrency', semaphores, and features such as 'if' and 'else' statements expressible in terms of the fragment we define. Plans in PRS consist of graphs. For compactness, and to allow a precise specification of the semantics, we adopt a plan-rule notation similar to that used in other BDI-based agent programming languages to specify the triggering and context conditions of plans, and specify plan bodies using a textual representation of bipartite graphs. We assume a first-order language with a vocabulary consisting of mutually disjoint and infinite sets of variable, constant, predicate, node, event-goal, and action symbols.

A PRS agent is defined by a belief base $\mathcal{B}$, an action-library Λ, and a plan-library Π. A *belief base* is a set of ground atoms. An *action-library* is a set of action-rules specifying the actions available to the agent. An *action* is of the form $act(\vec{t})$, where act is an n-ary action symbol denoting an evaluable function that may change the agent's environment, and $\vec{t} = t_1, \ldots, t_n$ are (possibly ground) terms. *Action-rules* are similar to STRIPS operators, and are of the form $act(\vec{v}){:}\psi \leftarrow \Phi^+; \Phi^-$, where $\vec{v} = v_1, \ldots, v_n$ are variables; ψ is a formula specifying the *precondition* of the action; and the *add-list* Φ^+ and *delete-list* Φ^- are sets of atoms that specify the effects of executing the action. A *plan-library* Π consists of a set of *plan-rules* of the form $e(\vec{t}){:}\varphi\ ;\psi \ \leftarrow G$, where e is an n-ary event-goal symbol, $\vec{t} = t_1, \ldots, t_n$ are terms, φ and ψ are formulas, and G is a plan-body graph. The rule states that, if the *context condition* ψ holds, G is a 'standard operating procedure' for achieving the *event-goal* $e(\vec{t})$ or the *goal-condition* φ.

Plan-body graphs are built from the following set of user programs: actions; *belief addition* $+b$ adds the atom b to $\mathcal{B}$; *belief removal* $-b$ removes the atom b from $\mathcal{B}$; *test* $?\phi$, where ϕ is a formula, tests whether ϕ holds in $\mathcal{B}$; *event-goal program* or *goal-condition program* $!\mathbf{ev}$, where $\mathbf{ev} \in \{e, \phi\}$, specifies that **ev** needs to be achieved; *wait* $\mathrm{WT}(\phi)$ waits until formula ϕ holds in $\mathcal{B}$; *passive preserve* $\mathrm{PR}_p(!\mathbf{ev}, \phi)$ specifies that $!\mathbf{ev}$ needs to be achieved while monitoring ϕ and aborted if **ev** fails or ϕ does not hold; and *active preserve* $\mathrm{PR}_a(!\mathbf{ev}, \phi)$, which is similar to the former except that if condition ϕ does not hold, the plan-body graph for $!\mathbf{ev}$ is suspended and the re-achievement of ϕ is attempted by posting the goal $!\phi$. If the goal succeeds and ϕ is re-achieved then the active preserve is resumed, and otherwise it is aborted (if the goal fails) or re-suspended (if the goal succeeds but ϕ is still not achieved). Since user program $!\mathbf{ev}$ may generate subgoals, wait and preserve programs may also become 'nested' within one another, giving rise to potentially complex interactions. We use $\mathrm{PR}_x(!\mathbf{ev}, \phi)$ to denote either $\mathrm{PR}_a(!\mathbf{ev}, \phi)$ or $\mathrm{PR}_p(!\mathbf{ev}, \phi)$. Formally, a *user program* P_u is a formula in the language defined by the grammar

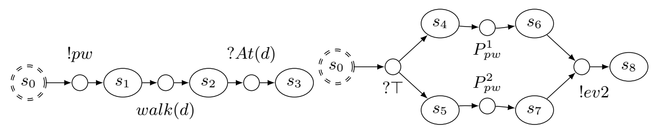
$$P_u \quad ::= \quad act \mid ?\phi \mid +b \mid -b \mid !\mathbf{ev} \mid \mathrm{WT}(\phi) \mid \mathrm{PR}_x(!\mathbf{ev}, \phi).$$

A plan-body graph is a directed bipartite graph representing a partially ordered set of user programs, where the set of nodes (i.e., node symbols) is split into state nodes, and transition nodes initially labelled with the user programs.

Definition 1. *Let $\boldsymbol{P}_u$ be the set of all user programs. A plan-body graph G is a tuple $\langle S, T, E_{in}, E_{out}, L_0, N, s_0\rangle$, where: (i) S is a set of state nodes and $s_0 \in S$ the initial node; (ii) T is a disjoint set of transition nodes; (iii) $E_{in} \subseteq S \times T$ is a set of input edges; (iv) $E_{out} \subseteq T \times S$ is a set of output edges; (v) function $L_0 : T \mapsto \boldsymbol{P}_u$ represents the user programs labelling transition nodes; and (vi) $N \subseteq S \cup T$ is a set of current nodes.*

We stipulate that for any node $n \in S \cup T$ there exists a sequence $s_1 \cdot t_1 \cdot \ldots \cdot s_{k-1} \cdot t_{k-1} \cdot s_k$, such that $s_0 = s_1$, node n is in the sequence, and for each $i \in [1, k-1] : (s_i, t_i) \in E_{in}$ and $(t_i, s_{i+1}) \in E_{out}$.

The PRS deliberation cycle consists of three main steps: processing environment updates, e.g., new event-goals or belief updates; instantiating a plan-body graph to achieve an event-goal or goal condition, or executing a single step in an instantiated plan-body graph; and notifying plan-body graphs when conditions they are monitoring are established or violated. Execution of a plan-body graph starts in the initial node (s_0), and progresses to one or more state/transition nodes (N). Transition nodes initially represent user programs (L_0) and evolve following the semantics in Section 3 until they reach the 'empty program' and terminate. The plan-body graph *finishes* executing (successfully) if no current state nodes N of G lead anywhere. Formally, the plan-body graph is *initial* if $N = \{s_0\}$, and *finished*, denoted by $\mathit{fin}(G)$, if $N \subseteq S$, and for all $s \in N$ and $t \in T$: $(s, t) \notin E_{in}$. All plan-body graphs occurring in a plan-library are initial.



(a) G_{walk} (b) G_{pw}

Figure 1: Plan-body graphs G_{walk} and G_{pw}. G_{pw} occurs in a plan-rule for the event-goal program $!pw$ (prepare to walk) in G_{walk}.

Example 1. *Consider an agent that has a goal to travel from its current location to a destination d [Sardina and Padgham, 2011]. The goal can be achieved in various ways depending on the distance to the destination, represented by the following plan-rules (each omitted goal-condition φ is $\top$):*

$$travel(d) : At(x) \wedge WalkDist(x, d) \leftarrow G_{walk},$$
$$travel(d) : At(x) \wedge \exists y(InCity(x, y) \wedge InCity(d, y)) \leftarrow G_{city},$$
$$travel(d) : At(x) \wedge \neg\exists y(InCity(x, y) \wedge InCity(d, y)) \leftarrow G_{far}.$$

The first plan-rule refers to the plan-body graph G_{walk} shown in Fig. 1.

3 PRS Semantics

Our semantics for PRS follows the approach adopted by the CAN [Winikoff *et al.*, 2002] agent programming language, as defined in [Sardina and Padgham, 2011], where a *transition relation* on *agent configurations* is defined in terms of a set of derivation rules [Plotkin, 1981]. An *agent configuration* is a tuple $[\Pi, \Lambda, \mathcal{B}, \mathcal{A}, \Gamma]$ where Π is a plan-library, Λ is an action-library, $\mathcal{B}$ is a belief base, $\mathcal{A}$ is a sequence of executed actions, and Γ is a set of *intentions* (as Π and Λ do not change during execution, we omit them from derivation rules). Agent configurations represent the execution state of a PRS agent, and intentions are the current states of *full programs* being pursued in order to achieve top-level event-goals. As in CAN, full programs extend the syntax of user programs to represent the current evolution of a user program, and may contain information used to decide the next transition, e.g. the plan-body graphs yet to be tried in achieving an event-goal.

Formally, a *full program* (or simply *program*) is a formula in the language defined by the grammar $P ::=$

$$\eta \mid act \mid ?\phi \mid +b \mid -b \mid !\mathbf{ev} \mid \mathrm{WT}_x(\phi) \mid \mathrm{PR}_x(P, \phi) \mid$$
$$\mathbf{ev} : (\!|\{\psi_1 : G_1, \ldots, \psi_n : G_n\}|\!) \mid P \rightsquigarrow P' \mid G \triangleright P \mid \eta \triangleright P$$

where η, ('nil') indicates that there is nothing left to execute; $\mathrm{WT}_x(\phi)$ denotes either $\mathrm{WT}(\phi)$ or $\overline{\mathrm{WT}}(\phi)$, where $\overline{\mathrm{WT}}(\phi)$ indicates that program $\mathrm{WT}(\phi)$ has been *adopted* (i.e., its execution has started); $\mathbf{ev} : (\!|\psi_1 : G_1, \ldots, \psi_n : G_n|\!)$ represents the set of relevant plan-rules for achieving the event-goal or goal-condition **ev**; $P \rightsquigarrow P'$ represents a *suspended* active preserve program P' whose associated condition is being re-achieved by a *recovery* program P; $G \triangleright P$, where $P = \mathbf{ev} : (\!|\psi_1 : G_1, \ldots, \psi_n : G_n|\!)$, represents the default deliberation mechanism for 'goal commitment': achieve **ev** using an applicable plan-body graph G, but if that fails, try an

$$\frac{}{[\mathcal{B}, \mathcal{A}, +b] \rightarrow [\mathcal{B} \cup \{b\}, \mathcal{A}, \eta]}\ add \qquad \frac{}{[\mathcal{B}, \mathcal{A}, -b] \rightarrow [\mathcal{B} \setminus \{b\}, \mathcal{A}, \eta]}\ del \qquad \frac{act' : \psi \leftarrow \Phi^+; \Phi^- \in \Lambda \quad act'\theta = act \quad \mathcal{B} \models \psi\theta}{[\mathcal{B}, \mathcal{A}, act] \rightarrow [(\mathcal{B} \setminus \Phi^-\theta) \cup \Phi^+\theta, \mathcal{A} \cdot act, \eta]}\ A$$

$$\frac{[\mathcal{B}, \mathcal{A}, G] \rightarrow [\mathcal{B}', \mathcal{A}', G']}{[\mathcal{B}, \mathcal{A}, G \rhd P] \rightarrow [\mathcal{B}', \mathcal{A}', G' \rhd P]}\ \rhd_{stp} \qquad \frac{}{[\mathcal{B}, \mathcal{A}, \eta \rhd P] \rightarrow [\mathcal{B}, \mathcal{A}, \eta]}\ \rhd_{end} \qquad \frac{\neg fin(G) \quad [\mathcal{B}, \mathcal{A}, G] \not\rightarrow \quad [\mathcal{B}, \mathcal{A}, P] \rightarrow [\mathcal{B}', \mathcal{A}', P']}{[\mathcal{B}, \mathcal{A}, G \rhd P] \rightarrow [\mathcal{B}', \mathcal{A}', P']}\ \rhd_f$$

Figure 2: PRS derivation rules for actions, belief updates, and goal commitment.

alternative applicable graph from those appearing in P; and $\eta \rhd P$ indicates that a plan-body graph for **ev** has succeeded. Note that full programs are more general than those yielded by our semantics.

We first give derivation rules for configurations of the form $[\Pi, \Lambda, \mathcal{B}, \mathcal{A}, P]$, where P is a full program, i.e., for a single intention. We give the derivation rules for actions, belief operations, goal and plan adoption, and goal commitment in Section 3.1; for advancing plan-body graphs in Section 3.2; for PRS-specific wait and preserve programs in Section 3.3; and for agents with configurations of the form $[\Pi, \Lambda, \mathcal{B}, \mathcal{A}, \Gamma]$, i.e., multiple intentions, in Section 3.4.

3.1 Semantics for Actions, Belief Updates, Goal and Plan Adoption, and Goal Commitment

Fig. 2 shows the derivation rules for actions and belief operations. Rules add and del simply update $\mathcal{B}$, replacing programs $+b$ and $-b$ with η ($\mathcal{A}$ remains unchanged). The semantics for actions is given by rule A. The antecedent checks whether there is a relevant action-rule for act (i.e., one whose 'head' act' matches act under substitution θ), and whether the action is applicable (i.e., its precondition holds in $\mathcal{B}$); the conclusion of the derivation rule applies the action's add- and delete-lists to $\mathcal{B}$ and appends the action to $\mathcal{A}$.

Rule Ev_1 *adopts* the event-goal program $!e$ by creating the set Δ of plan-rules relevant for the event, i.e, the rules in Π with event-goals matching e via a most general unifier mgu.

$$\frac{\Delta = \{\psi\theta : G\theta \mid e' : \varphi\, ; \psi \leftarrow G \in \Pi,\ \theta = \mathsf{mgu}(e, e')\} \neq \emptyset}{[\mathcal{B}, \mathcal{A}, !e] \rightarrow [\mathcal{B}, \mathcal{A}, e : (\!|\Delta|\!)]}\ Ev_1$$

Example 2. *Processing an event-goal program of the form* $!travel(Uni)$ *using rule* Ev_1 *yields the following (full) program, encoding all* relevant *options for this event:*

$$travel(Uni) : (\!|\psi_1 : G_{walk}, \psi_2 : G_{city}, \psi_3 : G_{far}|\!)$$

with $\psi_1 = At(x) \wedge WalkDist(x, Uni)$;
$\psi_2 = At(x) \wedge \exists y(InCity(x, y) \wedge InCity(Uni, y))$; *and*
$\psi_3 = At(x) \wedge \neg\exists y(InCity(x, y) \wedge InCity(Uni, y))$.

Similarly, rule Ev_2 adopts the goal-condition program $!\phi$ by creating the set Δ of plan-rules in Π that can achieve ϕ.

$$\frac{\Delta = \{\psi\theta : G\theta \mid e : \varphi\, ; \psi \leftarrow G \in \Pi,\ \varphi\theta \models \phi\} \neq \emptyset}{[\mathcal{B}, \mathcal{A}, !\phi] \rightarrow [\mathcal{B}, \mathcal{A}, \phi : (\!|\Delta|\!)]}\ Ev_2$$

Rule Sel selects an applicable plan-rule for event-goal or goal-condition **ev** from the set of relevant rules, and schedules the associated plan-body graph for execution.

$$\frac{\psi : G \in \Delta \quad \mathcal{B} \models \psi\theta}{[\mathcal{B}, \mathcal{A}, \mathbf{ev} : (\!|\Delta|\!)] \rightarrow [\mathcal{B}, \mathcal{A}, G\theta \rhd \mathbf{ev} : (\!|\Delta \setminus \{\psi : G\}|\!)]}\ Sel$$

Rules for goal commitment are shown in Fig. 2. Rule $\rhd_{stp}$ executes a single step in a program $G \rhd P$ if the plan-body graph G has neither failed nor finished (see Section 3.2). Rule $\rhd_{end}$ discards the alternative program P in a program $\eta \rhd P$ (where η here represents a completed graph). Finally, rule $\rhd_f$ schedules the alternative $P = \mathbf{ev} : (\!|\Delta|\!)$ for execution and executes a single step in it, provided G has failed and P has not, i.e., an applicable plan-rule exists for **ev**.

3.2 Semantics for Plan-Body Graphs

We extend the definition of a plan-body graph in Definition 1 to a *full plan-body graph*, representing the current 'state' in the evolution of an initial plan-body graph. A *full plan-body graph* is of the form $G = \langle S, T, E_{in}, E_{out}, L_0, L_c, N, s_0\rangle$ where $L_c : T \mapsto \mathbf{P}$ is a function that maps each transition node $t \in T$ to a full program $P \in \mathbf{P}$, which represents the current form of the possibly evolved (initial) user program $L_0(t)$. We use the following auxiliary definitions: $bef(t) = \{s \mid (s, t) \in E_{in}\}$ are the input state nodes of t; $aft(t) = \{s \mid (t, s) \in E_{out}\}$ are its output state nodes; and the *update* to function L_c with a (new) program P for t is

$$\textsc{upd}(L_c, t, P) = \big(L_c \setminus \{\langle t, L_c(t)\rangle\}\big) \cup \{\langle t, P\rangle\}.^{1}$$

The first rule states that a transition node t, in the case where it is not initially associated with a test condition, becomes active if it is not already active but all of its input states are. Becoming active includes (re-)initialising t to correspond to its user program. This is done in case t is part of a cycle.

$$\frac{t \in T \quad t \notin N \quad bef(t) \subseteq N \quad L_0(t) \neq ?\phi}{[\mathcal{B}, \mathcal{A}, G] \rightarrow [\mathcal{B}, \mathcal{A}, G']}\ G^P_{start}$$

where $G' = \langle S, T, E_{in}, E_{out}, L_0, L'_c, N', s_0\rangle$;
$L'_c = \textsc{upd}(L_c, t, L_0(t))$; $N' = \big(N \setminus bef(t)\big) \cup \{t\}$.

Once a transition node t is active, it can perform a single execution step in its associated (current) program $L_c(t)$.

$$\frac{t \in N \quad [\mathcal{B}, \mathcal{A}, L_c(t)] \rightarrow [\mathcal{B}', \mathcal{A}', P]}{[\mathcal{B}, \mathcal{A}, G] \rightarrow [\mathcal{B}', \mathcal{A}', G']}\ G^P_{stp}$$

$G' = \langle S, T, E_{in}, E_{out}, L_0, L'_c, N, s_0\rangle$; $L'_c = \textsc{upd}(L_c, t, P)$.

If a transition node's program has finished execution, the node becomes inactive and its outgoing nodes become active.

$$\frac{t \in N \quad L_c(t) = \eta}{[\mathcal{B}, \mathcal{A}, G] \rightarrow [\mathcal{B}, \mathcal{A}, G']}\ G^P_{end}$$

$G' = \langle S, T, E_{in}, E_{out}, L_0, L_c, N', s_0\rangle$; $N' = (N \setminus \{t\}) \cup aft(t)$.

[1] We treat the functions L_0 and L_c as relations, i.e., as sets of ordered pairs of the form $\langle t, P\rangle$.

Example 3. *Suppose that the agent believes it is currently at home, which is walking distance to the university. In this case, the Sel rule transforms the set of relevant options represented by the program in Example 2 into the program $G_{walk} \rhd travel(Uni) : (\!|\psi_2 : G_{city}, \psi_3 : G_{far}|\!)$, i.e., the agent selects the G_{walk} plan-body graph while keeping the other graphs as backup alternatives, represented by the right-hand side of the $\rhd$ operator. When the agent starts executing the G_{walk} graph, the rule G^P_{start} removes s_0 from N and adds the transition node associated with subgoal $!pw$. This is then executed using rule G^P_{stp}, whose antecedent uses rule Ev_1 to resolve the subgoal.*

If a transition node is initially associated with a test condition, then the node becomes active only if the condition holds in the current belief base. The chosen transition node also becomes inactive at the same execution step, as once the condition is tested there is nothing left to execute. Under this semantics, PRS allows choices in execution within the graph: either a non-deterministic choice when multiple transition nodes, with non 'mutex' test programs, exit the same state node, or deterministic choices induced by such tests.

$$\frac{t \in T \quad \mathit{bef}(t) \subseteq N \quad L_0(t) = ?\phi \quad \mathcal{B} \models \phi}{[\mathcal{B}, \mathcal{A}, G] \to [\mathcal{B}, \mathcal{A}, G']}\ G^{?\phi}_{stp}$$

$$G' = \langle S, T, E_{in}, E_{out}, L_0, L'_c, N', s_0\rangle;$$
$$L'_c = \textsc{upd}(L_c, t, \eta); \quad N' = \big(N \setminus \mathit{bef}(t)\big) \cup \mathit{aft}(t).$$

The case where $\mathcal{B} \not\models \phi$ holds represents failure, i.e., the inability to execute a step in t.

Finally, if a plan-body graph has finished (Section 2), rule G_{end} replaces it with program η.

$$\frac{\mathit{fin}(G)}{[\mathcal{B}, \mathcal{A}, G] \to [\mathcal{B}, \mathcal{A}, \eta]}\ G_{end}$$

Example 4. *Consider the evolution $G_{pw} \rhd pw : (\!|\Delta_{pw}|\!)$ of subgoal $!pw$. Achieving the former using the graph in Figure 1 involves the parallel execution of the programs P^1_{pw} and P^2_{pw}. This 'split' is represented by the outgoing edges from the transition node associated with the $?\top$ user program, and results in state nodes s_4 and s_5 being added to N, using rule $G^{?\phi}_{stp}$. Note that for the transition node associated with $!ev2$ to become active, both P^1_{pw} and P^2_{pw} must complete execution, and transition to s_6 and s_7, respectively.*

3.3 Semantics for Wait and Preserve Programs

We now give a semantics for wait and preserve programs of the form $\textsc{wt}(\phi)$, $\textsc{pr}_p(P, \phi)$, and $\textsc{pr}_a(P, \phi)$, and suspended active preserve programs of the form $P_1 \rightsquigarrow \textsc{pr}_a(P_2, \phi)$, where P_1 is the recovery program. In all cases, we assume that condition ϕ is ground when the program is adopted.

Rule W_{adopt} *adopts* a wait program, i.e., changes its form to indicate that condition ϕ is now being monitored. Rule W specifies that the wait for ϕ should continue if ϕ does not hold in the belief base. Finally, rule W_{end} specifies that the wait should end if ϕ does hold. In all cases, $C = [\mathcal{B}, \mathcal{A}, \overline{\textsc{wt}}(\phi)]$.

$$\frac{}{[\mathcal{B}, \mathcal{A}, \textsc{wt}(\phi)] \to C}\ W_{adopt} \quad \frac{\mathcal{B} \not\models \phi}{C \to C}\ W \quad \frac{\mathcal{B} \models \phi}{C \to [\mathcal{B}, \mathcal{A}, \eta]}\ W_{end}$$

The first set of derivation rules for preserve programs apply to both passive and active preserves. Rule Pr_{adopt} specifies the adoption of a preserve program, i.e., the adoption of its event-goal or goal-condition program $!\mathbf{ev}$. Rule Pr_{stp} executes a single step in a preserve program if ϕ is not violated, and Pr_{succ} removes a completed preserve program.

$$\frac{[\mathcal{B}, \mathcal{A}, !\mathbf{ev}] \to [\mathcal{B}, \mathcal{A}, \mathbf{ev} : (\!|\Delta|\!)]}{[\mathcal{B}, \mathcal{A}, \textsc{pr}_x(!\mathbf{ev}, \phi)] \to [\mathcal{B}, \mathcal{A}, \textsc{pr}_x(\mathbf{ev} : (\!|\Delta|\!), \phi)]}\ Pr_{adopt}$$

$$\frac{P \neq !\mathbf{ev} \quad \mathcal{B} \models \phi \quad [\mathcal{B}, \mathcal{A}, P] \to [\mathcal{B}', \mathcal{A}', P']}{[\mathcal{B}, \mathcal{A}, \textsc{pr}_x(P, \phi)] \to [\mathcal{B}', \mathcal{A}', \textsc{pr}_x(P', \phi)]}\ Pr_{stp}$$

$$\frac{}{[\mathcal{B}, \mathcal{A}, \textsc{pr}_x(\eta, \phi)] \to [\mathcal{B}, \mathcal{A}, \eta]}\ Pr_{succ}$$

Rule Pr_{fail} specifies that the passive preserve fails if ϕ is violated or P is blocked.

$$\frac{P \notin \{\eta, !\mathbf{ev}\} \quad (\mathcal{B} \not\models \phi \quad \vee \quad [\mathcal{B}, \mathcal{A}, P] \not\to)}{[\mathcal{B}, \mathcal{A}, \textsc{pr}_p(P, \phi)] \to [\mathcal{B}, \mathcal{A}, ?\mathit{false}]}\ Pr_{fail}$$

Rules APr_{fail} to APr^{stp2}_{sus} operationalise adopted active and suspended preserve programs. We define three special multisets.[2] Given an expression $\mathcal{E}$ that is either a program or plan-body graph $G = \langle S, T, E_{in}, E_{out}, L_0, L_c, N, s_0\rangle$, we define a 'path' of 'nested' (adopted) preserve and wait programs as any element in the multiset $\mathcal{T}(\mathcal{E})$, defined as $\mathcal{T}(\mathcal{E}) =$

$$\begin{cases} \{\mathcal{E} \cdot \tau \mid \tau \in \mathcal{T}(\hat{P})\} & \text{if } \mathcal{E} = P \rightsquigarrow \textsc{pr}_a(\hat{P}, \phi) \wedge P \neq G \rhd P_1; \\ \{\mathcal{E} \cdot \tau \mid \tau \in \mathcal{T}(\hat{P})\} & \\ \uplus\, \mathcal{T}(G) & \text{if } \mathcal{E} = (G \rhd P) \rightsquigarrow \textsc{pr}_a(\hat{P}, \phi); \\ \{\mathcal{E} \cdot \tau \mid \tau \in \mathcal{T}(G)\} & \text{if } \mathcal{E} = \textsc{pr}_x(G \rhd P, \phi); \\ \{\mathcal{E}\} & \text{if } \mathcal{E} \in \big\{\overline{\textsc{wt}}(\phi),\ \textsc{pr}_x(\mathbf{ev} : (\!|\Delta|\!), \phi)\big\}; \\ \mathcal{T}(G) & \text{if } \mathcal{E} = G \rhd P; \\ \mathcal{T}(L_c(t_1)) & \\ \uplus \ldots \uplus\, \mathcal{T}(L_c(t_n)) & \text{if } \mathcal{E} = G \text{ and } N \cap T = \{t_1, \ldots, t_n\}; \\ \emptyset & \text{otherwise.} \end{cases}$$

When $\mathcal{E} = G$ we take the multiset union of the sequences corresponding to transition nodes in G that are executed in parallel. The first element in such a sequence is a 'most abstract' preserve or wait program occurring in $\mathcal{E}$, and the last element is a 'most deeply' nested preserve or wait program occurring in $\mathcal{E}$. We use $\mathcal{S}_\tau(\mathcal{E})$ and $\mathcal{P}_\tau(\mathcal{E})$ to denote the multisets of all the elements in all the sequences in $\mathcal{T}(\mathcal{E})$ that are, and are not, of the form $P \rightsquigarrow P'$, respectively; i.e., any element in $\mathcal{S}_\tau(\mathcal{E})$ is a suspended (adopted) active preserve program, and any element in $\mathcal{P}_\tau(\mathcal{E})$ is an adopted wait, passive preserve, or active preserve program that is not suspended. We use $\mathcal{T}(\mathcal{E})$ to check whether a wait/preserve program in some 'path' in $\mathcal{E}$ may be 'pruned' by a more abstract preserve program in the path, and we use $\mathcal{S}_\tau(\mathcal{E})$ and $\mathcal{P}_\tau(\mathcal{E})$ to count the number of suspended and unsuspended programs occurring in $\mathcal{E}$.

Example 5. *Suppose P^1_{pw} and P^2_{pw} have evolved to, respectively, adopted programs $\overline{\textsc{wt}}(\phi)$ and $\textsc{pr}_p(P, \phi')$, with $P = e : (\!|\Delta|\!)$ (i.e., one parallel branch waits for a condition ϕ while the other tries to achieve an event-goal program $!e$ while preserving ϕ'). Then, $\mathcal{T}(G_{walk}) = \{\overline{\textsc{wt}}(\phi), \textsc{pr}_p(P, \phi')\}$. Moreover, if P evolves to $P' = G_e \rhd e : (\!|\Delta'|\!)$, where G_e mentions an adopted program, e.g., $\overline{\textsc{wt}}(\phi'')$, then $\mathcal{T}(G_{walk}) = \{\overline{\textsc{wt}}(\phi), \textsc{pr}_p(P', \phi') \cdot \overline{\textsc{wt}}(\phi'')\}$.*

[2] Adapted from [Sardina and Padgham, 2011].

$$\frac{\mathbf{C}_1 \xRightarrow{\text{INT}} \mathbf{C}_2 \quad \mathbf{C}_2 \xRightarrow{\text{EVENT}} \mathbf{C}_3 \quad \mathbf{C}_3 \xRightarrow{\text{COND}*} \mathbf{C}_4 \quad \mathbf{C}_4 \overset{\text{COND}}{\not\Longrightarrow}}{\mathbf{C}_1 \xRightarrow{\text{PRS}} \mathbf{C}_4} A_{prs} \qquad \frac{P \in \Gamma \quad [\mathcal{B}, \mathcal{A}, P] \to [\mathcal{B}', \mathcal{A}', P']}{\mathbf{C} \xRightarrow{\text{INT}} [\mathcal{B}', \mathcal{A}', (\Gamma \setminus \{P\}) \cup \{P'\}]} A_{int}$$

$$\frac{P \in \Gamma \quad [\mathcal{B}, \mathcal{A}, P] \not\to}{\mathbf{C} \xRightarrow{\text{INT}} [\mathcal{B}, \mathcal{A}, \Gamma \setminus \{P\}]} A_{rem} \quad \frac{e_1, \ldots, e_n \text{ are external event-goals}}{\mathbf{C} \xRightarrow{\text{EVENT}} [\mathcal{B}, \mathcal{A}, \Gamma \cup \Gamma']} A_{ev} \quad \frac{P \in \Gamma \quad [\mathcal{B}, \mathcal{A}, P] \to [\mathcal{B}, \mathcal{A}, P'] \quad P' \prec P}{\mathbf{C} \xRightarrow{\text{COND}} [\mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\}]} A_{cond}$$

Figure 3: Agent-level derivation rules; each $\mathbf{C}_i$ is an agent configuration, $\mathbf{C} = [\mathcal{B}, \mathcal{A}, \Gamma]$, and $\Gamma' = \{!e_1, \ldots, !e_n\}$.

Rule APr_{fail} specifies that the adopted active preserve program $\text{PR}_a(P, \phi)$ fails if P fails and the monitored condition ϕ is not violated. If ϕ is violated, APr_{sus} suspends the preserve program and attempts to re-establish ϕ using the recovery (goal-condition) program $!\phi$. Rules APr^{fail}_{sus} and APr_{unsus} specify, respectively, that a suspended preserve program fails if the recovery program fails, and is resumed if the recovery program completes.

$$\frac{P \notin \{\eta, !\mathbf{ev}\} \quad \mathcal{B} \models \phi \quad [\mathcal{B}, \mathcal{A}, P] \not\to}{[\mathcal{B}, \mathcal{A}, \text{PR}_a(P, \phi)] \to [\mathcal{B}, \mathcal{A}, ?\mathit{false}]} APr_{fail}$$

$$\frac{P \notin \{\eta, !\mathbf{ev}\} \quad \mathcal{B} \not\models \phi}{[\mathcal{B}, \mathcal{A}, \text{PR}_a(P, \phi)] \to [\mathcal{B}, \mathcal{A}, !\phi \rightsquigarrow \text{PR}_a(P, \phi)]} APr_{sus}$$

$$\frac{P_1 \neq \eta \quad [\mathcal{B}, \mathcal{A}, P_1] \not\to}{[\mathcal{B}, \mathcal{A}, P_1 \rightsquigarrow P_2] \to [\mathcal{B}, \mathcal{A}, ?\mathit{false}]} APr^{fail}_{sus}$$

$$\frac{}{[\mathcal{B}, \mathcal{A}, \eta \rightsquigarrow \text{PR}_a(P, \phi)] \to [\mathcal{B}, \mathcal{A}, \text{PR}_a(P, \phi)]} APr_{unsus}$$

Finally, rules APr^{stp1}_{sus} and APr^{stp2}_{sus} define the execution of recovery programs and suspended (active preserve) programs. Rule APr^{stp2}_{sus} executes a single 'cleanup' or 'notification' step in the suspended program. Such an execution step amounts to a program P_1 evolving to a program P_2 that has fewer suspended programs, or fewer unsuspended preserve or wait programs, e.g. due to a failed passive preserve that had a 'nested' wait program. The relation $P \prec P'$ is defined for programs P and P' as: $P \prec P' \stackrel{\text{def}}{=} |\mathcal{P}_\tau(P)| < |\mathcal{P}_\tau(P')| \vee |\mathcal{S}_\tau(P)| < |\mathcal{S}_\tau(P')|$.

$$\frac{[\mathcal{B}, \mathcal{A}, P_1] \to [\mathcal{B}', \mathcal{A}', P_1']}{[\mathcal{B}, \mathcal{A}, P_1 \rightsquigarrow P_2] \to [\mathcal{B}', \mathcal{A}', P_1' \rightsquigarrow P_2]} APr^{stp1}_{sus}$$

$$\frac{[\mathcal{B}, \mathcal{A}, P_2] \to [\mathcal{B}, \mathcal{A}, P_2'] \quad P_2' \prec P_2}{[\mathcal{B}, \mathcal{A}, P_1 \rightsquigarrow \text{PR}_a(P_2, \phi)] \to [\mathcal{B}, \mathcal{A}, P_1 \rightsquigarrow \text{PR}_a(P_2', \phi)]} APr^{stp2}_{sus}$$

Note that rule APr^{stp2}_{sus} implies that if an active preserve is suspended, all of the (possibly adopted) 'nested' programs occurring in P_2 are 'implicitly suspended', i.e., they can only perform 'cleanup' or 'notification' steps, so we are guaranteed to terminate.

Proposition 1. *Any sequence of configurations* $[\mathcal{B}_1, \mathcal{A}_1, P_1] \cdot \ldots \cdot [\mathcal{B}_n, \mathcal{A}_n, P_n]$ *is finite if for all* $i \in [1, n-1]$, *we have that* $P_{i+1} \prec P_i$ *and* $[\mathcal{B}_i, \mathcal{A}_i, P_i] \to [\mathcal{B}_{i+1}, \mathcal{A}_{i+1}, P_{i+1}]$.

Proof. Since each such execution step from P_i to P_{i+1} must yield fewer wait programs, preserve programs, and/or programs of the form $P \rightsquigarrow \text{PR}_a(P', \phi)$, it is sufficient to consider whether a 'switch' of the latter to resume $\text{PR}_a(P', \phi)$ can lead to it (possibly with an 'evolved' P') being suspended again, and whether this can continue indefinitely.

For $\text{PR}_a(P', \phi)$ to resume, we must have $P = \eta$, and then if ϕ still does not hold, the program will indeed evolve to $!\phi \rightsquigarrow \text{PR}_a(P', \phi)$. Moreover, recovery program $!\phi$ does have at least one relevant plan-rule, as it was once able to evolve to η (recall that ϕ has been ground from the moment its associated active preserve program was adopted). However, the only possible execution step on $!\phi$ cannot reduce the number of aforementioned programs. □

We use the notation $C \dashrightarrow C'$ to denote that there exists a non-empty sequence of configurations from C to C'.

3.4 Agent-Level ('Top-Level') Semantics

We now give the derivation rules for the top-level execution of an agent program. Transitions between agent configurations are defined by the derivation rules in Fig. 3; an expression $\mathbf{C} \xRightarrow{\text{t}} \mathbf{C}'$ denotes a transition of type $t \in \{\text{PRS}, \text{EVENT}, \text{COND}, \text{INT}\}$.

Rule A_{prs} is the top-level rule, and represents the PRS deliberation cycle. A single PRS type execution step comprises three things: progressing an intention by one step, or removing a completed intention (i.e., $P = \eta$) or a failed one (using rules A_{int} or A_{rem}, respectively); processing newly observed event-goals (using rule A_{ev}), i.e., creating an intention for each new event-goal that is observed from the (external) environment; and finally, performing all the necessary 'notification' and 'cleanup' steps on wait, preserve, suspended, and recovery programs (using rule A_{cond}) to leave the agent in a 'sound' or 'stable' configuration. More specifically, A_{cond} takes a single step in an intention if the step will yield an intention with fewer suspended programs, or fewer unsuspended (adopted) preserve or wait programs.

4 Properties of the Semantics

We now prove key properties of the semantics and show the greater expressivity of our PRS fragment compared to CAN. In what follows, we use $[\mathcal{B}, \mathcal{A}, P] \to$ as an abbreviation for $\exists \mathcal{B}', \mathcal{A}', P' : [\mathcal{B}, \mathcal{A}, P] \to [\mathcal{B}', \mathcal{A}', P']$, and $\mathbf{C}_1$, $\mathbf{C}_2$ are agent configurations of the form $[\mathcal{B}_i, \mathcal{A}_i, \Gamma_i]$ such that $\mathbf{C}_1$ is *sound* and $\mathbf{C}_1 \xRightarrow{\text{PRS}} \mathbf{C}_2$. A configuration $\mathbf{C} = [\Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma]$ is <u>*sound*</u> iff *(i)* for all $\overline{\text{WT}}(\phi) \in \mathcal{P}_\tau(\Gamma)$, $\mathcal{B} \not\models \phi$; *(ii)* for all $P \rightsquigarrow P' \in \mathcal{S}_\tau(\Gamma)$, $[\mathcal{B}, \mathcal{A}, P] \to$; and *(iii)* for all $\text{PR}_x(P, \phi) \in \mathcal{P}_\tau(\Gamma)$, $\mathcal{B} \models \phi$ and $[\mathcal{B}, \mathcal{A}, P] \to$.

Theorem 1 states that all configurations resulting from applying rule A_{prs} on a sound configuration are sound.

Theorem 1. *Let* $\mathbf{C}_1$ *and* $\mathbf{C}_2$ *be as above. Then,* $\mathbf{C}_2$ *is sound.*

Proof Sketch. Observe from the antecedent of derivation rule A_{prs} that only one step of type INT is performed on $\mathbf{C}_1$, followed by one of type EVENT, and zero or more of type COND. Assume the theorem does not hold because there is a passive preserve $\text{PR}_p(P,\phi) \in \mathcal{P}_\tau(\Gamma_2)$, for some P, such that $\mathcal{B}_2 \not\models \phi$ or $[\mathcal{B}_2, \mathcal{A}_2, P] \not\rightarrow$. Then, since $\text{PR}_p(P,\phi)$ is an adopted program appearing in some intention $P_I \in \Gamma_2$, either rule Pr_{fail} or Pr_{succ} can be applied to configuration $[\mathcal{B}_2, \mathcal{A}_2, \text{PR}_p(P,\phi)]$ to yield an intention P'_I. Since $P'_I \prec P_I$, rule A_{cond} will be applied (possibly multiple times) to P_I until $\text{PR}_p(P',\phi) \notin \mathcal{P}_\tau(\Gamma_2)$ for all P', which contradicts our assumption. Assume instead that the theorem does not hold because there is a program $P \rightsquigarrow P' \in \mathcal{S}_\tau(\Gamma_2)$ such that $[\mathcal{B}_2, \mathcal{A}_2, P] \not\rightarrow$. Then, either rule APr^{fail}_{sus} or APr_{unsus} will be applied to configuration $[\mathcal{B}_2, \mathcal{A}_2, P \rightsquigarrow P']$ or its 'evolution', again resulting in a contradiction. The remaining cases are proved similarly. □

The next theorem states that an adopted wait program is only removed in a PRS step if either its condition becomes satisfied, or the program is *pruned*, i.e., it is a descendant of an adopted preserve or a suspended preserve that is discarded. Given a program P and 'path' $\tau \in \mathcal{T}(P)$, we denote the program at index $n > 0$ as $\tau[n]$ (where $\tau[1]$ is τ's most abstract program). A program P is <u>*pruned*</u> between $\mathbf{C}_1$ and $\mathbf{C}_2$ iff for any $\tau \in \mathcal{T}(\Gamma_1)$ with $\mathcal{T}(\Gamma) = \biguplus_{P' \in \Gamma} \mathcal{T}(P')$ and $\tau[k] = P$ for some $k > 0$, there exists a $0 < j < k$ such that:[3]

1. $\tau[j] = \text{PR}_p(P_j, \phi_j)$ and $\mathcal{B}_2 \not\models \phi_j$;
2. $\tau[j] = \text{PR}_a(P_j, \phi_j)$, $\mathcal{B}_2 \not\models \phi_j$, and $[\mathcal{B}_2, \mathcal{A}_2, !\phi_j] \not\rightarrow$; or
3. $\tau[j] = P_1 \rightsquigarrow P_2$ and either
   $[\mathcal{B}_2, \mathcal{A}_2, P_1] \dashrightarrow [\mathcal{B}_2, \mathcal{A}_2, P'_1] \not\rightarrow$, or
   $[\mathcal{B}_1, \mathcal{A}_1, P_1] \rightarrow [\mathcal{B}_2, \mathcal{A}_2, P'_1] \dashrightarrow [\mathcal{B}_2, \mathcal{A}_2, P''_1 \neq \eta] \not\rightarrow$.

Theorem 2. *Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be as before. For each $\overline{\text{WT}}(\phi) \in \mathcal{P}_\tau(\Gamma_1)$ such that $\overline{\text{WT}}(\phi) \notin \mathcal{P}_\tau(\Gamma_2)$, we have that $\mathcal{B}_2 \models \phi$, or $\overline{\text{WT}}(\phi)$ is pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$.*

Proof Sketch. Let $\overline{\text{WT}}(\phi) \in \mathcal{P}_\tau(\Gamma_1)$ be a program such that $\overline{\text{WT}}(\phi) \notin \mathcal{P}_\tau(\Gamma_2)$. Let t be a transition node currently labelled with $\overline{\text{WT}}(\phi)$, where $T = \{t, \ldots\}$ and $N = \{t, \ldots\}$ are the transition nodes and current nodes in a ('partially executed') plan-body graph G. We consider the case where if $\mathcal{B}_2 \not\models \phi$, then $\overline{\text{WT}}(\phi)$ must have been pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$, because no other derivation rules can 'remove' $\overline{\text{WT}}(\phi)$. First, rule G_{end} (which, if applicable, 'removes' G) requires $fin(G)$ to hold, which cannot be the case as $t \in N$; for the same reason, rule G^P_{start} cannot 'reset' $\overline{\text{WT}}(\phi)$. Second, the antecedent of rule $\triangleright_f$ requires that $[\mathcal{B}, \mathcal{A}, G] \not\rightarrow$ (for some $\mathcal{A}$ and $\mathcal{B} \in \{\mathcal{B}_1, \mathcal{B}_2\}$), which cannot hold as we can take a step on $\overline{\text{WT}}(\phi)$ via rule W. Similarly, if P is an intention in which G occurs, the antecedent of agent-level rule A_{rem} cannot hold (i.e., $[\mathcal{B}_1, \mathcal{A}, P] \not\rightarrow$ cannot hold for any $\mathcal{A}$). Then, it follows that $\overline{\text{WT}}(\phi)$ is pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$. □

Theorem 3 states that an adopted preserve program is only 'removed' if: its condition becomes violated; its associated program becomes blocked; or the preserve program is pruned.

[3]The definitions of $\mathcal{P}_\tau(\mathcal{E})$ and $\mathcal{S}_\tau(\mathcal{E})$, for some expression $\mathcal{E}$, can be generalised similarly.

Theorem 3. *If $\mathbf{C}_1$ and $\mathbf{C}_2$ are as before, and $\text{PR}_x(P,\phi) \in \mathcal{P}_\tau(\Gamma_1)$ is a program s.t. $\text{PR}_x(P_2,\phi) \notin \mathcal{P}_\tau(\Gamma_2)$ for any P_2:*

1. *$\mathcal{B}_2 \not\models \phi$ or $[\mathcal{B}_2, \mathcal{A}_2, P] \dashrightarrow [\mathcal{B}_2, \mathcal{A}_2, P'] \not\rightarrow$;*
2. *$[\mathcal{B}_1, \mathcal{A}_1, P] \rightarrow [\mathcal{B}_2, \mathcal{A}_2, P'] \dashrightarrow [\mathcal{B}_3, \mathcal{A}_3, P''] \not\rightarrow$; or*
3. *$\text{PR}_x(P,\phi)$ is pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$.*

Proof Sketch. We show that if two of the conditions above do not hold, then the third will. For example, if *(2)* and *(3)* do not hold, then $\text{PR}_x(P,\phi)$ must have been 'removed' by rule Pr_{fail} or suspended by APr_{sus}, whose antecedents entail *(1)*. Similarly, if *(1)* and *(2)* do not hold, then $\text{PR}_x(P,\phi)$ must have been pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$ for the reasons given in the proof of Theorem 2. □

Theorem 4 states that a suspended preserve program is only 'removed' during a PRS step if: the recovery program completes and the preserve program's condition is re-established; the recovery program becomes blocked; or both are pruned.

Theorem 4. *If $\mathbf{C}_1$ and $\mathbf{C}_2$ are as before, and $P_1 \rightsquigarrow \text{PR}_a(P_2,\phi) \in \mathcal{S}_\tau(\Gamma_1)$ is a program s.t. $P \rightsquigarrow \text{PR}_a(P',\phi) \notin \mathcal{S}_\tau(\Gamma_2)$ for any P and P':*

1. *$[\mathcal{B}_1, \mathcal{A}_1, P_1] \rightarrow [\mathcal{B}_1, \mathcal{A}_1, \eta]$ and $\mathcal{B}_1 \models \phi$;*
2. *$[\mathcal{B}_2, \mathcal{A}_2, P_1] \dashrightarrow [\mathcal{B}_2, \mathcal{A}_2, P'_1] \not\rightarrow$;*
3. *$[\mathcal{B}_1, \mathcal{A}_1, P_1] \rightarrow [\mathcal{B}_2, \mathcal{A}_2, P'_1] \dashrightarrow [\mathcal{B}_3, \mathcal{A}_3, P''_1 \neq \eta] \not\rightarrow$;*
4. *or $P_1 \rightsquigarrow \text{PR}_a(P_2,\phi)$ is pruned between $\mathbf{C}_1$ and $\mathbf{C}_2$.*

The proof of Theorem 4 is similar to that of Theorem 3.

Theorems 3 and 4 considered the case where *all* occurrences of preserve programs associated with the same condition ϕ, e.g. in multiple intentions in Γ_1, are removed in a PRS step. There is also the case where only *some* of them are removed in such a step. It is not difficult to develop theorems for this case, though we would need additional formal machinery. Similarly, we can show that active preserve programs are suspended and resumed for the right reasons. For example, with a minor extension to Theorem 3, we can show that if $\text{PR}_a(P,\phi)$ becomes suspended (i.e., $!\phi \rightsquigarrow \text{PR}_a(P,\phi) \in \mathcal{S}_\tau(\Gamma_2)$), then $\mathcal{B}_2 \not\models \phi$ and $[\mathcal{B}_2, \mathcal{A}_2, !\phi] \rightarrow$ hold.

In the remainder of this section, we characterise the relative expressivity of our PRS fragment and the CAN formalism as defined in [Sardina and Padgham, 2011]. A CAN plan-rule is of the form $e : \psi \leftarrow P$, where e is an event-goal, ψ is a context condition, and P is a <u>*plan-body*</u>, i.e., a formula in the language defined by the grammar $P ::=$

$$act \mid ?\phi \mid +b \mid -b \mid !e \mid P_1; P_2 \mid P_1 \parallel P_2 \mid \mathsf{Goal}(\phi_s, P', \phi_f),$$

where $P_1; P_2$ is sequential composition, $P_1 \parallel P_2$ is parallel composition, and $\mathsf{Goal}(\phi_s, P', \phi_f)$ is a declarative goal specifying that formula ϕ_s (the goal) should be achieved using program P', failing if ϕ_f becomes true. The remaining programs are defined as for PRS user programs.

To state our results, we need the notions of *execution traces* and *solutions*. We define these only for PRS as the definitions for CAN are analogous.

Definition 2. *An <u>execution trace</u> of an agent configuration $\mathbf{C} = [\Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma]$ is a finite sequence of agent configurations $\mathbf{C}_1 \cdot \ldots \cdot \mathbf{C}_n$ such that $\mathbf{C} = \mathbf{C}_1$ and $\mathbf{C}_i \overset{\text{PRS}}{\Longrightarrow} \mathbf{C}_{i+1}$ for all*

$i \in [1, n-1]$*; the* *solution* *in* $\boldsymbol{C}_1 \cdot \ldots \cdot \boldsymbol{C}_n$ *is the sequence* $\mathcal{A}$ *of actions such that* $\mathcal{A}_n = \mathcal{A}_1 \cdot \mathcal{A}$.

With $\Lambda, \Pi, \mathcal{B}$, and Γ as above, $\text{SOL}(\Lambda, \Pi, \mathcal{B}, \Gamma)$ denotes the set of solutions, i.e., the set of sequences of actions performed in the execution traces of configuration $[\Lambda, \Pi, \mathcal{B}, \epsilon, \Gamma]$, where ϵ denotes the empty string. Theorem 5 states that a CAN plan-library Π_c^- not mentioning $\mathsf{Goal}(\phi_s, P, \phi_f)$ programs (as there is no corresponding program in PRS) can be translated into an equivalent PRS plan-library.

Theorem 5. *If* Π_c^- *is a CAN library and* Λ *an action-library, there exists a PRS library* Π_p *s.t. for any event-goal* $!e$ *and beliefs* $\mathcal{B}$: $\text{SOL}(\Lambda, \Pi_c^-, \mathcal{B}, \{!e\}) = \text{SOL}(\Lambda, \Pi_p, \mathcal{B}, \{!e\})$.

Proof Sketch. Given a CAN plan-rule $e : \psi \leftarrow P \in \Pi_c^-$, the first step is to obtain the corresponding PRS plan-rule $e : \top; \psi \leftarrow G$. We define three functions: $g(P,n)_{in}$, $g(P,n)_{out}$, and $g(P,n)_L$, where $n \in \mathbb{N}$. When $n = 1$, these functions represent respectively the elements E_{in}, E_{out}, and L_0 in G. If $P = act$, we define $g(P,n)_{in} = \{(n \cdot s, n \cdot t)\}$; $g(P,n)_{out} = \{(n \cdot t, n \cdot s')\}$; and $g(P,n)_L = \{\langle n \cdot t, P\rangle\}$, where s, s' and t are unique symbols (and thus, $n \cdot t$, for example, is a string). We also define $g(P,n)_{start} = n \cdot s$ and $g(P,n)_{end} = n \cdot s'$. Intuitively, n uniquely identifies the PRS plan-body 'subgraph' corresponding to P. If $P = P_1; P_2$ is the sequential composition of CAN programs P_1 and P_2,

$$\begin{aligned}
&g(P,n)_{out} = \\
&\quad g(P_1, n\cdot 1)_{out} \cup g(P_2, n \cdot 2)_{out} \cup \{(n \cdot t'', n \cdot s''), \\
&\quad (n \cdot t, g(P_1, n \cdot 1)_{start}), \ (n \cdot t', g(P_2, n \cdot 2)_{start})\}; \\
&g(P,n)_{in} = \\
&\quad g(P_1, n \cdot 1)_{in} \cup g(P_2, n \cdot 2)_{in} \cup \{(n \cdot s, n \cdot t), \\
&\quad \big(g(P_1, n \cdot 1)_{end}, n \cdot t'\big), \ \big(g(P_2, n \cdot 2)_{end}, n \cdot t''\big)\}; \\
&g(P,n)_L = g(P_1, n \cdot 1)_L \cup g(P_2, n \cdot 2)_L \cup \\
&\quad \{\langle n \cdot t, ?\top\rangle, \langle n \cdot t', ?\top\rangle, \langle n \cdot t'', ?\top\rangle\},
\end{aligned}$$

where transition node $n \cdot t'$ 'connects' the subgraphs corresponding to P_1 and P_2. As before, we define $g(P,n)_{start} = n \cdot s$ and $g(P,n)_{end} = n \cdot s''$, and s, s'', t, t' and t'' are unique symbols. We similarly define the subgraph corresponding to a CAN parallel composition $P_1 || P_2$, test program, etc.

We then show by induction on the structures of P and G above that their traces yield the same solutions. For example, the derivation rules for CAN's sequential composition can be simulated by repeatedly applying the G^P_{start}, G^P_{stp} and G^P_{end} rules of PRS, and vice versa. $\square$

Theorem 6 states that the converse does not hold: even if we ignore programs that have no counterparts in CAN, PRS plan-libraries cannot be 'directly mapped' to CAN libraries. This result follows from a similar one in [de Silva, 2017] which showed that the 'plan-body' representation used in HTN planning allows a more fine-grained interleaving of steps than do CAN plan-bodies: a CAN plan-body must specify steps in a 'series-parallel' manner, whereas HTN 'plan-bodies' (and PRS plan-body graphs) do not.

In what follows, Π_p^- denotes PRS plan-libraries that do not mention goal-condition, wait, nor preserve programs, and $\Pi_c \in \text{CAN}(\Pi_p^-)$ denotes a *directly* mapped CAN library, i.e., one obtained from Π_p^- by ignoring the goal-condition φ in each plan-rule, and replacing each graph G appearing in Π_p^- with a CAN plan-body P such that the multisets of (user) programs occurring in G and P are the same.

Theorem 6. *There exists a PRS library* Π_p^-*, an action-library* Λ*, and event-goal* $!e$*, s.t. for any CAN library* $\Pi_c \in \text{CAN}(\Pi_p^-)$ *and beliefs* $\mathcal{B}$: $\text{SOL}(\Lambda, \Pi_p^-, \mathcal{B}, \{!e\}) \neq \text{SOL}(\Lambda, \Pi_c, \mathcal{B}, \{!e\})$.

Proof Sketch. We translate an example HTN 'plan-body' provided in [de Silva, 2017] into a PRS plan-body graph. First, we create the PRS plan-rule $e_{top} : \top; \top \leftarrow G$, in particular by taking the following input and output edges for G:

$$\begin{aligned}
E_{in} = \{&(s_1,t_1),(s_2,t_2),(s_3,t_3),(s_4,t_4),\\
&(s_5,t_4),(s_6,t_5),(s_7,t_6),(s_8,t_6)\}; \text{ and}\\
E_{out} = \{&(t_1,s_2),(t_1,s_3),(t_2,s_4),(t_3,s_5),\\
&(t_3,s_6),(t_4,s_7),(t_5,s_8),(t_6,s_9)\}.
\end{aligned}$$

Second, we set the initial program $L_0(t_i)$ for each $t_i \in \{t_1, \ldots, t_6\}$ to a different event-goal e_{t_i}, each of which is associated with a single plan-body graph representing the sequence of unique actions $a^1_{t_i} \cdot a^2_{t_i}$. Finally, we show that the following sequence of actions:

$$a^1_{t_1} \cdot a^2_{t_1} \cdot a^1_{t_2} \cdot a^1_{t_3} \cdot a^2_{t_3} \cdot a^1_{t_5} \cdot a^2_{t_2} \cdot a^1_{t_4} \cdot a^2_{t_4} \cdot a^2_{t_5} \cdot a^1_{t_6} \cdot a^2_{t_6}$$

cannot be produced by any CAN trace of program $!e_{top}$, relative to any CAN plan-library that is directly mapped from the above set of PRS plan-rules; e.g., the CAN body $e_{t_1}; (e_{t_2} \parallel (e_{t_3}; e_{t_5})); e_{t_4}; e_{t_6}$ does not allow $a^2_{t_5}$ to follow $a^2_{t_4}$. $\square$

5 Discussion

We proposed an operational semantics for a significant fragment of the OpenPRS variant of PRS that accounts for (possibly nested) graph-based plan-bodies; language features such as active preserves; and for adopting, suspending, resuming, and aborting such programs. We showed that our semantics is sound, correctly accounts for the key interactions between (nested) wait and preserve programs, and that plan-body graphs do not have a direct translation to plan-bodies in a typical BDI-based agent programming language (CAN).

Our work is closely related to that of Dastani et al. [2011] on the semantics of resuming, suspending, and aborting maintenance goals. They define complex goal types that include achievement of goal formulas, maintenance goals and complex temporal goals. All of these can be encoded as plan-body graphs in PRS through a straightforward mapping. Van Riemsdijk et al. [2009] develop a modal logic for goal representation and reasoning mechanisms for non-temporal goals. While the goal types they consider could probably be represented as PRS plan-body graphs, encoding the associated mechanisms for reasoning about goal conflicts is less straightforward. Thangarajah et al. [2011; 2014] define an operational semantics for various types of goals, most of which can be implemented using PRS plan-body graphs and programs. However, their maintenance goals include a construct to proactively maintain a goal condition by anticipating whether it will fail. This specific capability of predicting conditions in the future is lacking in our semantics.

Future work includes investigating whether an arbitrary PRS plan-rule can be simulated by a *set* of CAN (or AgentSpeak) plan-rules, and extending our semantics to add more PRS features, e.g. meta-level reasoning and plan steps that can overlap in execution.

Acknowledgements

Lavindra is grateful to Félix Ingrand for useful discussions about OpenPRS while Lavindra was a researcher on the GOAC/MARAE projects (2009-2011) at LAAS-CNRS. Felipe thanks CNPq for partial financial support under its PQ fellowship, grant number 305969/2016-1.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *The International Journal of Robotics Research (IJRR)*, 17(4):315–337, 1998.

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

[Busetta *et al.*, 1999] P Busetta, R Rönnquist, A Hodgson, and A Lucas. Jack intelligent agents - components for intelligent agents in Java. Technical report, Agent Oriented Software, Australia, January 1999.

[Dastani *et al.*, 2011] M Dastani, M B van Riemsdijk, and M Winikoff. Rich goal types in agent programming. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 405–412, 2011.

[de Silva, 2017] L de Silva. BDI Agent Reasoning with Guidance from HTN Recipes. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 759–767, 2017.

[Foughali *et al.*, 2016] M Foughali, B Berthomieu, S Dal Zilio, F Ingrand, and A Mallet. Model checking real-time properties on the functional layer of autonomous robots. In *Proc. of Formal Methods and Software Engineering (ICFEM)*, pages 383–399, 2016.

[Georgeff and Ingrand, 1989] M P Georgeff and F F Ingrand. Decision-Making in an Embedded Reasoning System. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 972–978, 1989.

[Georgeff and Lansky, 1986] M P Georgeff and A L Lansky. A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle. Technical Report 375, Artificial Intelligence Center, SRI International, Menlo Park, California, 1986.

[Georgeff and Lansky, 1987] M P Georgeff and A L Lansky. Reactive Reasoning and Planning. In *Proc. of the American Association for Artificial Intelligence (AAAI)*, pages 677–682, 1987.

[Ghallab *et al.*, 2016] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, New York, NY, USA, 1st edition, 2016.

[Huber, 1999] M J Huber. JAM: a BDI-theoretic mobile agent architecture. In *Proc. of the Annual Conference on Autonomous Agents*, pages 236–243, 1999.

[Ingrand *et al.*, 1996] F F Ingrand, R Chatila, R Alami, and F Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, pages 43–49, 1996.

[Ingrand, 1991] F Ingrand. *OPRS Development Environment Version 1.1b7*, 1991. Last updated January 2014. https://git.openrobots.org/projects/openprs.

[Lemaignan *et al.*, 2017] S Lemaignan, M Warnier, E. Akin Sisbot, A Clodic, and R Alami. Artificial cognition for social human-robot interaction: An implementation. *Artificial Intelligence (AIJ)*, 247:45 – 69, 2017. Special Issue on AI and Robotics.

[Morley and Myers, 2004] D Morley and K Myers. The SPARK agent framework. In *Proc. of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 714–721, 2004.

[Niemueller *et al.*, 2017] T Niemueller, F Zwilling, G Lakemeyer, M Löbach, S Reuter, S Jeschke, and A Ferrein. *Cyber-Physical System Intelligence*, pages 447–472. Springer International Publishing, 2017.

[Plotkin, 1981] G D Plotkin. A Structural Approach to Operational Semantics. Technical report, University of Aarhus, 1981.

[Rao, 1996] A S Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In Walter Van de Velde and John W Perram, editors, *Proc. of the European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 42–55. Springer-Verlag, 1996.

[Sardina and Padgham, 2011] S Sardina and L Padgham. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011.

[Sardina *et al.*, 2006] S Sardina, L de Silva, and L Padgham. Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In *Proc. of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1001–1008, 2006.

[Thangarajah *et al.*, 2011] J Thangarajah, J Harland, D Morley, and N Yorke-Smith. Operational Behaviour for Executing, Suspending and Aborting Goals in BDI Agent Systems. In *Proc. of the International Workshop on Declarative Agent Languages and Technologies (DALT)*, pages 1–21. Springer, 2011.

[Thangarajah *et al.*, 2014] J Thangarajah, J Harland, D Morley, and N Yorke-Smith. An Operational Semantics for the Goal Life-Cycle in BDI Agents. *Autonomous Agents and Multi-Agent Systems*, 28(4):682–719, 2014.

[van Riemsdijk *et al.*, 2009] M. B van Riemsdijk, M Dastani, and J-J Ch. Meyer. Goals in conflict: Semantic foundations of goals in agent programming. *Autonomous Agents and Multi-Agent Systems*, 18(3):471–500, 2009.

[Winikoff *et al.*, 2002] M Winikoff, L Padgham, J Harland, and J Thangarajah. Declarative & Procedural Goals in Intelligent Agent Systems. In *Proc. of the International Conference on Principles and Knowledge Representation and Reasoning (KR)*, pages 470–481, 2002.