Quantitative Modelling and Analysis of BDI Agents

Blair Archibald[1], Muffy Calder[1], Michele Sevegnani[1] and Mengwei Xu[1*]

[1]School of Computing Science, University of Glasgow, Glasgow, United Kingdom.

*Corresponding author(s). E-mail(s): mengwei.xu@glasgow.ac.uk;
Contributing authors: blair.archibald@glasgow.ac.uk; muffy.calder@glasgow.ac.uk;
michele.sevegnani@glasgow.ac.uk;

Abstract

Belief-Desire-Intention (BDI) agents are a popular agent architecture. We extend Conceptual Agent Notation (Can)—a BDI programming language with advanced features such as failure recovery and declarative goals—to include probabilistic action outcomes, *e.g.* to reflect failed actuators, and probabilistic policies, *e.g.* for probabilistic plan and intention selection. The extension is encoded in Milner's bigraphs. Through application of our BigraphER tool and the PRISM model checker, the *probability* of success (intention completion) under different probabilistic outcomes and plan/event/intention selection strategies can be investigated and compared. We present a smart manufacturing use case. A significant result is that plan selection has limited effect compared with intention selection. We also see that the impact of action failures can be marginal—even when failure probabilities are large—due to the agent making smarter choices.

Keywords: BDI Agents; Quantitative Analysis; Bigraphs; Probabilistic Modelling; Robotic Software

1 Introduction

A well-studied and popular architecture for developing rational agents is the Belief-Desire-Intention (BDI) paradigm. BDI agents build on a sound theoretical foundation to model an agent where (B)eliefs represent what the agent knows, (D)esires what the agent wants to bring about, and (I)ntentions the desires the agent is currently acting upon. BDI agents have inspired many agent-oriented programming languages including AgentSpeak [1], Can [2], CanPlan [3], 3APL [4], and 2APL [5] along with a collection of mature software toolkits and platforms including JACK [6], Jason [7], and Jadex [8]. BDI agents have been recognised for their efficiency and scalability in areas such as business [9] and healthcare [10].

In BDI languages, desires and intentions are often represented using a plan library. Each plan describes a course of actions which an agent can perform to address an adopted event (often representing a task from the external environment) given some beliefs hold, while the set of intentions are the plans currently being executed. Typically BDI languages: (1) assume that action outcomes (*i.e.* the effects on *external* environment) are deterministic, (2) remain agnostic *internally* to the choice of an applicable plan to address an adopted event, (3) remain agnostic *internally* to the choice of a pending event to adopt from the external environment, (4) remain agnostic *internally* to the order that intentions are progressed. These assumptions facilitate the formal verification of agent behaviour through a non-deterministic underlying transition system (depicted in Fig. 3) in

work such as [11, 12]), where plan, event, and intention selection denotes branching choices and actions have a single outcome. As such, most verification approaches are limited to analysing qualitative properties, querying whether an intention completes or not.

Though useful to have qualitative assurance, unfortunately, this often does not adequately represent agent behaviours in realistic setting such as cyber-physical robotics systems [13]. For example, the outcome of an action may be probabilistic due to imprecise actuation, *e.g.* the robot tries to open a door, but might fail. Plans, event, and intentions are not created equal and likely have different (domain-specific) characteristics such as preference and urgency, which may require different certain selection strategies (*e.g.* ordered, fixed schedules, or sampled from a probabilistic distribution). As a result, there is a growing need for formal techniques that can provide support for automated analysis of quantitative properties such as "what is the probability of eventually completing an intention?" and "what is the worst-case probability of eventually completing an intention over all possible selection strategies?".

To illustrate the problem, we use a robot packaging task in a smart manufacturing scenario as an example (detailed quantitative analysis is given in Section 4). The overall goal is to pack products automatically for shipping. The robot insulates products with suitable wrapping bags, to prevent temperature rise and consequent spoilage, and then transfers the wrapped products to a storage location. There are two types of wrapping bags: premium and standard. The standard wrapping is preferred as the cheaper option, however it may not be effective if the product temperature is already too high, and/or the packaging can occasionally break, which results in damaged product (*i.e.* a negative action outcome with some probability). Before wrapping the products, the robot also has to decide which product to handle first (as there may be multiple products waiting), meaning handling a product before it spoils requires a notion of urgency. While it is important to prioritise the more urgent products, it is also sensible to progress less urgent ones from time-to-time, before they also become urgent and spoiled. So we need to model and *quantify* agent behaviour when there are a range of choices, inherent uncertainty, and characteristics of preference and urgency. For example, we may wish to know the probability the robot can complete packaging under different schedules, negative outcomes, and decisions.

In the BDI community, probabilistic action outcomes are usually implicit—requiring the agent to *sense* failures and *revise* the beliefs (*i.e.* to enable new plans)—and are often disregarded when modelling. Although most agent language semantics specify non-deterministic *plan selection*, *e.g.* in [2], it is typical in practice for plans to be ordered—either statically [7] or at run-time [14]—to enforce deterministic branching. While desirable to exploit the highest priority plan, it may be worthwhile exploring other plans every now and then to avoid being stuck in a local maximum. Similarly, *event/intention selection* are also not implemented in a non-deterministic fashion either, but in a fixed schedule including Round-Robin (executing a step of each intention in turn) or First-In-First-Out. Interestingly, customised selection implementations change the semantics of agent languages implicitly, and is often a point where implementations and semantics diverge.

We argue that the highest ordering (*i.e.* local maximum) and fixed schedules (*e.g.* Round-Robin) are not always the best approach to plan, event, and intention selection and suggest agents should support probabilistic selection strategies together with the need to evaluate the undesired outcomes of actions. We present a formal approach (in contrast with the informal customisation of implementations mentioned above) to specify, model, and quantitatively analyse BDI agents with probabilistic action outcomes and plan, event, and intention selections drawn from a probability distribution. Quantitative verification, *e.g.* asking the probability some intention completes, aids the design of agents by enabling plan, event, and intention selection strategies to be explored and compared, and mitigates the risk of negative action outcomes by providing much-needed quantitative assurance.

1.1 Approach

We have chosen to work with Can [2] as it captures the essence of BDI concepts without describing implementation details such as data structures. As a superset of AgentSpeak [1], Can includes *declarative goals*, *concurrency*, and *failure*

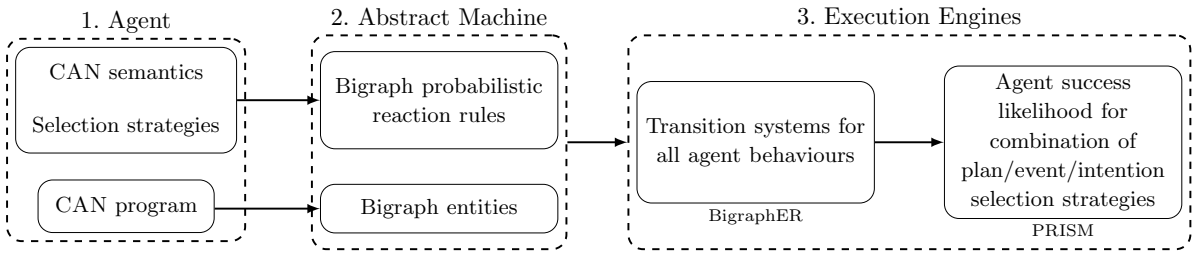

Figure 1: Analysis of intention success for probabilistic CAN programs with different selection strategies.

recovery. Here, we extend the operational semantics to a probabilistic setting. Although we focus on CAN, its features are similar to other BDI languages and our approach would apply equally well to them.

Our approach is depicted in Fig. 1. On the left, we have the inputs: (above) CAN semantics and selection strategies and (below) the agent program. In the middle, we have the abstract machine: (above) the CAN semantics are encoded by probabilistic bigraph reaction rules and (below) the agent program is encoded by bigraph entities. On the right we have the execution engines BigraphER [15] and PRISM [16]. We use BigraphER to generate a transition system (a DTMC—Discrete Time Markov Chain) of all possible agent behaviours, for each given combination of selection strategies and initial states. We express successful (or failure) completion of intentions as a Probabilistic Computation Tree Logic (PCTL) [17] formula (*e.g.eventually* the intention(s) complete successfully). The transition system and formula are the inputs to the PRISM model checker, which returns a likelihood. Put more simply, the user simply "runs" their PCTL formula and agent model with different plan/intention/event selection strategies, as required.

We employ probabilistic bigraphs [18] as the intermediate language, building on our previous work on (non-probabilistic) bigraphs as an executable semantics for (non-probabilistic) CAN [19]. We choose bigraphs, over any other formalism, for several reasons. First, its entity and type system allow a natural encoding of beliefs, desires, intentions, and plans as parallel regions. Second, its matching and rewriting nature closely mirrors CAN operational semantics, allowing us flexibility to trial different underlying semantics by changing a few bigraph rules. Third, the priority and conditional rule features provided in BigraphER support straightforward expression of selection strategies (*e.g.* ordered and fixed schedules). Fourth, there is an intuitive diagrammatic representation. The overall result is a user-friendly, direct and smooth translation that supports both probabilistic modelling and predicate-labelled transition systems that can be exported to model checkers like PRISM.

Parts of this study and preliminary results were presented in [20]. We make the following additional research contributions:

- a probabilistic extension of the full structural operational semantics of CAN;
- an extended executable semantics of CAN based on probabilistic bigraphs;
- a presentation of how different selection strategies are encoded in bigraphs;
- an extended evaluation and analysis use case, comparing various plan, event, and intention selection under probabilistic action outcomes, *e.g.* ordered and Round-Robin;
- a reflection on insights gained from creating a probabilistic extension of CAN, and the practical value of probabilistic agents for *e.g.* agent-designers.

The paper is organised as follows. In Section 2 we provide a brief overview of BDI agents and bigraphs. In Section 3 we propose the probabilistic extension of CAN semantics. In Section 4 we evaluate our approach on a smart manufacturing example. In Section 5, we reflect on the generality and limits of our approach. We discuss related work in Section 6, future work in Section 7, and conclude in Section 8.

2 Background

2.1 BDI Agents

A BDI agent has an explicit representation of beliefs, desires, and intentions. The beliefs correspond to what the agent believes about the environment, while the desires are a set of *external* events that the agent can respond to. To respond to those events, the agent selects a plan (given its beliefs) from the pre-defined plan library and commits to the selected plan by turning it into a new intention.

2.1.1 BDI Syntax

The CAN language formalises a classical BDI agent consisting of a belief base $\mathcal{B}$ and a plan library Π. The belief base $\mathcal{B}$ is a set of formulas encoding the current beliefs and has belief operators for entailment (*i.e.* $\mathcal{B} \models \varphi$), and belief atom addition (resp. deletion) $\mathcal{B} \cup \{b\}$ (resp. $\mathcal{B} \setminus \{b\}$). In general, any logic is allowed providing entailment is supported for a belief base. A propositional logic with natural number comparisons is used in this work. A plan library Π is a collection of plans of the form $e : \varphi \leftarrow P$ with e the triggering event, φ the context condition, and P the plan-body. The triggering event e specifies why the plan is triggered, while the context condition φ determines *when* the plan-body P is able to handle the event. Events can be either be external (*i.e.* from the environment in which the agent is operating) or internal (*i.e.* sub-goals that the agent itself tries to accomplish). A (partially executed) plan-body P for a selected plan $e : \varphi \leftarrow P$ is the *intention* that is addressing e. The language used in the plan-body is defined by the following grammar:

$$\begin{aligned} P ::= {} & \mathit{nil} \mid +b \mid -b \mid \mathit{act} \mid ?\varphi \mid e \mid P_1;\ P_2 \mid P_1 \triangleright P_2 \mid \\ & P_1 \parallel P_2 \mid e : (\mid \varphi_1 : P_1, \cdots, \varphi_n : P_n \mid) \mid \\ & \mathit{goal}(\varphi_s, P, \varphi_f) \end{aligned}$$

where nil is an empty program, $+b$ and $-b$ belief addition and deletion, act a primitive action, $?\varphi$ a test for φ in the belief base, and e is a sub-event (*i.e.* internal event). Actions act take the form $\mathit{act} = \varphi \leftarrow \langle \phi^-, \phi^+ \rangle$, where φ is the pre-condition, and ϕ^- and ϕ^+ are the deletion and addition sets (resp.) of belief atoms, *i.e.* a belief base $\mathcal{B}$ is revised to be $(\mathcal{B} \setminus \phi^-) \cup \phi^+$ when the action executes. We also denote the set of actions in the plan library as Λ. To execute a sub-event, a plan (corresponding to that event) is selected and the plan-body added in place of the event. In this way we allow plans to be nested (similar to sub-routine calls in other languages). In addition, there are composite programs $P_1;\ P_2$ for sequence, $P_1 \triangleright P_2$ that executes P_2 in the case that P_1 fails, and $P_1 \parallel P_2$ for interleaved concurrency. A set of relevant plans (those that respond to the same event) is denoted by $e : (\mid \psi_1 : P_1, \cdots, \psi_n : P_n \mid)$. Finally, a declarative goal program $\mathit{goal}(\varphi_s, P, \varphi_f)$ expresses that the declarative goal φ_s should be achieved through program P, failing if φ_f becomes true, and retrying as long as neither φ_s nor φ_f is true (see in [3] for details).

2.1.2 BDI Semantics

CAN semantics is specified by two types of transitions. The first type, denoted $\Rightarrow$, specifies *agent-level* evolution over $\langle E^e, \mathcal{B}, \Gamma \rangle$, detailing how to execute a complete agent where E^e is the set of pending external events to address (the desires), $\mathcal{B}$ the belief base, and Γ a set of partially executed plan-bodies (intentions). The second, denoted $\rightarrow$, specifies *intention-level* evolution on configurations $\langle \mathcal{B}, P \rangle$ where $\mathcal{B}$ is the belief base, and P the plan-body currently being executed.

The agent-level semantics are given in Fig. 2a. Rule A_{event} handles external events, that originate from the environment, by adopting them as intentions. Rule A_{step} selects an intention from the intention base, and evolves a single step w.r.t. the intention-level transition, while A_{update} discards any intentions that cannot make progress (either because they have already succeeded, or failed).

Fig. 2b gives intention-level rules for evolving any single intention. For example, the rule *act* handles the execution of an action, when the pre-condition ψ is met, resulting in a belief state update. Rule *event* replaces an event with the set of relevant plans, while rule *select* chooses an applicable plan from a set of relevant plans while retaining un-selected plans as backups. With these backup plans, rules for failure recovery $\triangleright_;$, $\triangleright_\top$, and $\triangleright_\bot$ enable new plans to be selected if the current plan fails (*e.g.* due to environment changes). Rules $;$ and $;_\top$ allow executing plan-bodies in sequence, while rules $\parallel_1$, $\parallel_2$, and $\parallel_\top$

$$A_{event}\ \frac{e \in E^e}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow \langle E^e \setminus \{e\}, \mathcal{B}, \Gamma \cup \{e\}\rangle} \qquad A_{step}\ \frac{P \in \Gamma \quad \langle \mathcal{B}, P\rangle \to \langle \mathcal{B}', P'\rangle}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow \langle E^e, \mathcal{B}', (\Gamma \setminus \{P\}) \cup \{P'\}\rangle}$$

$$A_{update}\ \frac{P \in \Gamma \quad \langle \mathcal{B}, P\rangle \nrightarrow}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow \langle E^e, \mathcal{B}, \Gamma \setminus \{P\}\rangle}$$

(a) Agent-level CAN semantics.

$$act\ \frac{act : \psi \leftarrow \langle \phi^-, \phi^+\rangle \quad \mathcal{B} \vDash \psi}{\langle \mathcal{B}, act\rangle \to \langle (\mathcal{B} \setminus \phi^- \cup \phi^+), nil\rangle} \qquad event\ \frac{\Delta = \{\varphi : P \mid (e' = \varphi \leftarrow P) \in \Pi \wedge e' = e\}}{\langle \mathcal{B}, e\rangle \to \langle \mathcal{B}, e : (|\ \Delta\ |)\rangle}$$

$$select\ \frac{\varphi : P \in \Delta \quad \mathcal{B} \vDash \varphi}{\langle \mathcal{B}, e : (|\ \Delta\ |)\rangle \to \langle \mathcal{B}, P \rhd e : (|\ \Delta \setminus \{\varphi : P\}\ |)\rangle} \qquad \rhd;\ \frac{\langle \mathcal{B}, P_1\rangle \to \langle \mathcal{B}', P_1'\rangle}{\langle \mathcal{B}, P_1 \rhd P_2\rangle \to \langle \mathcal{B}', P_1' \rhd P_2\rangle}$$

$$\rhd_\top\ \frac{}{\langle \mathcal{B}, nil \rhd P_2\rangle \to \langle \mathcal{B}', nil\rangle} \qquad \rhd_\bot\ \frac{P_1 \neq nil \quad \langle \mathcal{B}, P_1\rangle \nrightarrow \quad \langle \mathcal{B}, P_2\rangle \to \langle \mathcal{B}', P_2'\rangle}{\langle \mathcal{B}, P_1 \rhd P_2\rangle \to \langle \mathcal{B}', P_2'\rangle} \qquad ;\ \frac{\langle \mathcal{B}, P\rangle \to \langle \mathcal{B}', P'\rangle}{\langle \mathcal{B}, P_1;\ P_2\rangle \to \langle \mathcal{B}', P_1';\ P_2\rangle}$$

$$\|_1\ \frac{\langle \mathcal{B}, P_1\rangle \to \langle \mathcal{B}', P_1'\rangle}{\langle \mathcal{B}, P_1 \| P_2\rangle \to \langle \mathcal{B}', P_1' \| P_2\rangle} \qquad \|_2\ \frac{\langle \mathcal{B}, P_2\rangle \to \langle \mathcal{B}', P_2'\rangle}{\langle \mathcal{B}, P_1 \| P_2\rangle \to \langle \mathcal{B}', P_1 \| P_2'\rangle} \qquad \|_\top\ \frac{}{\langle \mathcal{B}, nil \| nil\rangle \to \langle \mathcal{B}, nil\rangle}$$

$$G_s\ \frac{\mathcal{B} \vDash \varphi_s}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f)\rangle \to \langle \mathcal{B}, nil\rangle} \qquad G_f\ \frac{\mathcal{B} \vDash \varphi_f}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f)\rangle \to \langle \mathcal{B}, ?false\rangle}$$

$$G_{init}\ \frac{P \neq P_1 \rhd P_2 \quad \mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f}{\langle \mathcal{B}, goal(\varphi_s, P, \varphi_f)\rangle \to \langle \mathcal{B}, goal(\varphi_s, P \rhd P, \varphi_f)\rangle} \qquad G_;\ \frac{\mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f \quad \langle \mathcal{B}, P_1\rangle \to \langle \mathcal{B}', P_1'\rangle}{\langle \mathcal{B}, goal(\varphi_s, P_1 \rhd P_2, \varphi_f)\rangle \to \langle \mathcal{B}', goal(\varphi_s, P_1' \rhd P_2, \varphi_f)\rangle}$$

$$G_\rhd\ \frac{\mathcal{B} \nvDash \varphi_s \quad \mathcal{B} \nvDash \varphi_f \quad \langle \mathcal{B}, P_1\rangle \nrightarrow}{\langle \mathcal{B}, goal(\varphi_s, P_1 \rhd P_2, \varphi_f)\rangle \to \langle \mathcal{B}, goal(\varphi_s, P_2 \rhd P_2, \varphi_f)\rangle}$$

(b) Intention-level CAN semantics.

Figure 2: CAN semantics from [3].

specify how to execute (interleaved) concurrent programs. Rules G_s and G_f deal with declarative goals when either the success condition φ_s or the failure condition φ_f become true. Rule G_{init} initialises persistence by setting the program in the declarative goal to be $P \rhd P$, *i.e.* if P fails try P again. This ensures P runs indefinitely unless either the success condition φ_s or failure condition φ_f holds. Rule $G_;$ takes care of performing a single step on an already initialised program. Finally, the derivation rule $G_\rhd$ re-starts the original program if the current (partially-executed) program has finished or got blocked (when neither φ_s nor φ_f becomes true).

2.1.3 Agent Example

For illustration, we give a classic example [19]: arranging a conference trip. The agent program is shown in Listing 1, and commentary follows.

An agent desires to arrange a conference trip, denoted by an external event `e_conference_travelling` (line 6). We assume there are only two ways to travel to the conference. The first is by car, given by the plan in line 2, which expresses that if the agent believes it owns a car (*i.e.* `own_car`) and the venue is in the driving distance (*i.e.* `driving_distance`), it can start the car (`start_car`) and drive (`driving`) all the way to the venue. To specify the actions, for example, the action `start_car` (line 8) expresses that if the car is functional (*i.e.* `car_functional`) and after executing it, the belief of the engine being on (*i.e.* `engine_on`) will be added while deleting nothing from the belief base.

The second way to travel is by air, given by the plans in lines 3 to 4. This plan expresses that if the budget allows and there is a flight, the agent can book the ticket first, then post internally a sub-event to actually travelling by plane, and go to the venue after landing. To address the sub-event `e_get_on_board`, we have plan in line 4, which expresses that if the agent believes the flight has been booked, it can go to the airport and fly by plane.

3 Probabilistic CAN Semantics

The semantics of CAN are specified by two types of transitions. The first is the agent-level transition $\Rightarrow$ in Fig. 2a that specifies how to execute a complete agent. The second is the intention-level

Listing 1: Agent design of an conference trip arrangement.

```
// Plans
e_conference_travelling : own_car ∧ driving_distance ← start_car; driving
e_conference_travelling : budget_allowed ∧ flight_available
e_get_on\board : flight_booked ← go_to_airport; flying
// External events
e_conference_travelling
// Action descriptions
start_car = car_functional ← ⟨φ⁻ = ∅, φ⁺ = {engine_on}⟩
driving = engine_on ← ⟨φ⁻ = ∅, φ⁺ = {at_venue}⟩
book_flight = true ← ⟨φ⁻ = ∅, φ⁺ = {flight_booked}⟩
go_to_venue = flight_landed ← ⟨φ⁻ = ∅, φ⁺ = {at_venue}⟩
flying = flight_landed ← ⟨φ⁻ = {flight_booked,at_airport}, φ⁺ = {flight_landed}⟩
```

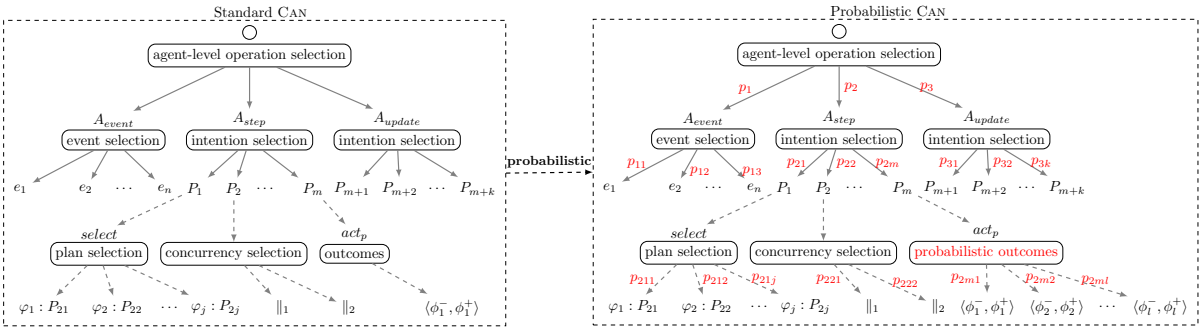

Figure 3: Non-deterministic (standard CAN) and new probabilistic transitions highlighting plan, event, and intention selection, and action execution. Solid lines are agent-level transitions, while dashed lines are intention-level.

transition $\rightarrow$ in Fig. 2b that specifies how to evolve a given single intention.

CAN semantics feature non-deterministic transitions, *e.g.* for plan selection. To allow for probabilistic selection and action outcomes, we must extend this to support probabilistic transitions. Figure 3 provides a high level comparison of the standard non-deterministic, and our new probabilistic semantics for CAN.

Choices appear throughout both the agent and intention level semantics. An agent with multiple external events to respond and a set of intentions to pursue is faced with three operations to choose from, namely agent-level operation selection. The agent can incorporate any pending external events specified by semantic rule A_{event}, it can select an intention and execute a step (according to the intention-level semantics) using A_{step}, or it can manage the intention set by removing an unprogressable intentions using A_{update}. In the original CAN semantics, these are chosen non-deterministically, that is, there is no way to prioritise completing existing intentions over handling new events.

Once an agent-level operation is chosen, there are further decisions to make. For example, which pending external event (there may be multiple) should be adopted? Similarly, both A_{step} and A_{update} must select one intention from a set of intentions (*i.e.* intention selection). These choices are also made non-deterministically, again meaning we cannot prioritise specific events/intentions.

After choosing to step an intention (A_{step}), progressing this intention may imply (visualised as dashed lines in Fig. 3) selecting an applicable plan, progressing concurrent program, and executing an action. Again, plan selection is made non-deterministic using rule *select*, the order of concurrent program progress is non-deterministic using $\|_1$ or $\|_2$, and action has only one single outcome by *act*.

Previous work [19] formally modelled and analysed non-probabilistic CAN (*i.e.* the left side of Fig. 3). We extend this to define a probabilistic semantics for CAN, and show how this allows quantitative analysis. The right side of Fig. 3 shows our probabilistic extension of CAN semantics.

To move from non-deterministic transitions to probabilistic transitions, we employ probabilistic transitions $\mathcal{C} \rightarrow_p \mathcal{C}'$ (*i.e.* move from $\mathcal{C}$ to $\mathcal{C}'$ with probability p) [21]. To extend the non-deterministic transition, the key is to assign probability to each selection choice. In next sections we detail why and how we extend both agent-level and intention-level transitions from CAN, and how suitable distributions can be constructed to support quantitative analysis.

Notation

We use μ, η to refer to probability distributions over a set A. We write $\mu = [x \mapsto p_1, y \mapsto p_2]$ to denote the probability distribution, over $\{x, y\}$ where, for example, x is sampled with probability p_1, and access the probability of an element using function notation, *e.g.* $\mu(x) = p_1$. For a distribution we require $\sum_{p \in \mu} = 1$. Only probabilities for non-zero elements are given, such that for $[x \mapsto 1, y \mapsto 0]$ we instead write $[x \mapsto 1]$. We use $Dist(A)$ to refer to the *set* of discrete probability distributions over A, *i.e.* a set with probability distributions as elements.

We denote the set of all possible belief atoms, external events and intentions—for a *specific program*—as $\overline{\mathcal{B}}$, $\overline{E^e}$, and $\overline{\Gamma}$ respectively. At each agent step, the belief base (resp. events, intentions) is given as $\mathcal{B} \subseteq \overline{\mathcal{B}}$.

3.1 Probabilistic Agent-Level Semantics

The agent-level semantics of CAN characterise the evolution of an agent which has multiple external events to respond and is currently pursuing a set of intentions While the agent-level semantics allow the agent to respond to new events even while already dealing with other events, only one agent-level operation can be performed at each step. Such non-deterministic choice of agent-level operation is implicit in CAN semantics. Here we formalise and express it as a function as follows where we denote the set of agent-level rules as $\mathcal{A} = \{A_{event}, A_{step}, A_{update}\}$:

$$\mathcal{S}_{ao} : 2^{\overline{E^e}} \times 2^{\overline{\mathcal{B}}} \times 2^{\overline{\Gamma}} \rightarrow \mathcal{A} \cup \{\bot\}$$

which returns a choice of agent-level operation given any agent-level configuration $\langle E^e, \mathcal{B}, \Gamma \rangle$ and $\bot$ stands for no applicable rules available.

In practice, a common approach of selecting an agent-level operation is often done in deterministic fashion such as incorporating an external event if any before selecting any intention to execute a step if possible. However, we may need to choose an agent-level operation from a distribution. For example, it may be better for the agent to *mainly* incorporate external events as intentions at the early operation stage and to *mainly* progress existing intentions at later stage. To allow this we sample agent-level operations based on a probability distribution, *i.e.* with the following selection function:

$$\mathcal{S}^p_{ao} : 2^{\overline{E^e}} \times 2^{\overline{\mathcal{B}}} \times 2^{\overline{\Gamma}} \rightarrow Dist(\mathcal{A} \cup \{\bot\})$$

The probability of $\bot$ of any distribution $\mu \in Dist(\mathcal{A} \cup \{\bot\})$ is either $\mu(\bot) = 0$ (agent-level operation(s) available.) or $\mu(\bot) = 1$ (no agent-level operation(s) available). Using $\mathcal{S}^p_{ao}$, we will define probabilistic rules for actual execution of agent-level operations in the next sections.

We will next detail how an agent decides which event or intention should be selected when a given agent-level operation is selected (according to the distribution from selection function $\mathcal{S}^p_{ao}$). The details of how these functions are implemented are given later on in Section 4.4.

3.1.1 Probabilistic Event Adoption

BDI agents operate by continuously handling external events that represent tasks originating from the external environment.

To respond to these events, an agent selects an external event ($e \in E^e$) and adopts it in the intention set ($\Gamma \cup \{e\}$), using rule A_{event}

$$A_{event} \frac{e \in E^e}{\langle E^e, \mathcal{B}, \Gamma \rangle \Rightarrow \langle E^e \setminus \{e\}, \mathcal{B}, \Gamma \cup \{e\} \rangle}$$

There may be multiple pending external events, due to different requests from the environment, and it is not clear which event should be selected:

the rule above picking any waiting event. In practice, we want more control over the event that is selected as different events may be more or less urgent. Many agent *implementations*, going against the semantics, choose events using an event selection function $\mathcal{S}_e$ that is customised to account for priorities and is formalised in the following form:

$$\mathcal{S}_e : 2^{\overline{\mathcal{B}}} \times 2^{\overline{E^e}} \to \overline{E^e} \cup \{\bot\}$$

Given a belief base and a set of external events it returns an event or $\bot$, *i.e.* no requested event present. In other words, the agent always takes an event if one exists the agent. We also note that the belief base is needed to provide relevant information (*e.g.* priority) for the agent to make more informed event selection decisions.

To allow non-strict orderings we sample events based on a probability distribution, *i.e.* with the probabilistic event selection function:

$$\mathcal{S}^p_e : 2^{\overline{\mathcal{B}}} \times 2^{\overline{E^e}} \to Dist(\overline{E^e} \cup \{\bot\})$$

Using $\mathcal{S}^p_e$ and $\mathcal{S}^p_{ao}$ (which defines probability of selecting rule A_{event}), we can define a probabilistic A_{event} rule:

$$A^p_{event}\ \frac{\mathcal{S}^p_{ao}(E^e, \mathcal{B}, \Gamma) = \eta \qquad \eta(A_{event}) = p_1 \qquad \mathcal{S}^p_e(\mathcal{B}, E^e) = \mu_1 \qquad e \in E^e \qquad \mu_1(e) = p'_1}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow_{p_1 \cdot p'_1} \langle E^e \setminus \{e\}, \mathcal{B}, \Gamma \cup \{e\}\rangle}$$

The rule A^p_{event} says that if the probability of performing event selection at this step is p_1 and the probability of selecting a pending external event is p'_1, then the probability of selecting this event at this step is $p_1 \cdot p'_1$. When no external event is available (*i.e.* $\mathcal{S}^p_e(\mathcal{B}, E^e) = \mu_1$ and $\mu_1(\bot) = 0$), A^p_{event} is not applicable.

3.1.2 Probabilistic Intention Progression

Every time an agent adopts an external event a new intention is created. As agents should adopt events to stay reactive if one exists, we end up with a *set* of intentions competing for the agent's attention. As Can agents are single-threaded, at most one intention can be executed each agent step in an interleaved manner. If an agent decides to work on intentions (rather than adopt new events), the agent must make a choice: out of the set of progressable intentions, which should be progressed? In standard Can this is captured by

$$A_{step}\ \frac{P \in \Gamma \qquad \langle \mathcal{B}, P\rangle \to \langle \mathcal{B}', P'\rangle}{\langle E^e, \mathcal{B}, \Gamma\rangle \Rightarrow \langle E^e, \mathcal{B}', (\Gamma \setminus \{P\}) \cup \{P'\}\rangle}$$

This rule non-deterministically progresses any intention (that can be progressed) with respect to the intention level rules in Fig. 2b.

The precondition states that P must be an intention but does not control which. If we want more control, we can add an intention selection function as follows:

$$\mathcal{S}_i = 2^{\overline{\mathcal{B}}} \times 2^{\overline{\Gamma}} \to \overline{\Gamma} \cup \{\bot\}$$

As before, this function returns a fixed intention, or $\bot$ if no intention is present in the intention set (*i.e.* $\Gamma = \emptyset$). We note that the function $\mathcal{S}_i$, by definition, includes the set of all possible stages of each intention as every step of an intention is itself a different intention. To efficiently construct such an function, however, we often treat different stages of an intention as the same intention. In fact, we decide to link each intention with the related external event as intentions ultimately address external events to construct this function (detailed in Section 4). As such, regardless of how an intention evolves, it is treated as the same intention. Similar to event selection, the key component of beliefs in the function domain is to provide domain-specific information of intentions to aid customised selection.

Due to the need to choose intentions from a distribution, we provide the following function to allow intention selection from a distribution:

$$\mathcal{S}^p_i = 2^{\overline{\mathcal{B}}} \times 2^{\overline{\Gamma}} \to Dist(\overline{\Gamma} \cup \{\bot\})$$

The agent-level transitions of A_{step} depends on the intention-level transitions and we need to account for this in the transition probabilities. To have a probabilistic agent-level rule A_{step}, we assume, for a chosen progressable intention $P \in \Gamma$, $\langle \mathcal{B}, P\rangle \to_{p'} \langle \mathcal{B}', P'\rangle$ holds, for example, if a plan selection for the given intention P is required based on $select^P$. The detailed probabilistic intention-level semantics will be given Section 3.2.

The probabilistic rule for intention selection is

$$A^p_{step}\ \frac{\begin{array}{c}\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \qquad \eta(A_{step})=p_2\\ P\in\Gamma \quad \mathcal{S}^p_i(\mathcal{B},\Gamma)=\mu_2 \quad \mu_2(P)=p_2'\\ \langle\mathcal{B},P\rangle \rightarrow_{p_2''} \langle\mathcal{B}',P'\rangle\end{array}}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_{p_2\cdot p_2'\cdot p_2''} \langle E^e,\mathcal{B}',\Gamma'\rangle}$$

where $\Gamma' = (\Gamma \setminus \{P\}) \cup \{P'\}$. Rule A^p_{step} says that if the probability of performing intention selection at this step is p_2, the probability of selecting intention P is p_2', and the probability of progressing it to P_2'', then the probability of selecting and progressing intention P to P' at this step is $p_2 \cdot p_2' \cdot p_2''$.

3.1.3 Probabilistic Intention Update

The final agent-level rule A_{update} drops *any* unprogressable intention from the intention set.

$$A_{update}\ \frac{P\in\Gamma \qquad \langle\mathcal{B},P\rangle \nrightarrow}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow \langle E^e,\mathcal{B},\Gamma\setminus\{P\}\rangle}$$

Same as rule A_{step}, rule A_{update} also requires the same distribution $\mathcal{S}^p_i$ to allow intention selection. Unlike A_{step}, however, A_{update} depends on the intention-level transitions which has a default probability 1, namely $\langle\mathcal{B},P\rangle \nrightarrow_1$. To have a probabilistic agent-level rule A_{update}, we present the following probabilistic rule for intention selection.

$$A^p_{update}\ \frac{\begin{array}{cc}\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta & \eta(A_{update})=p_3\\ \mathcal{S}^p_i(\mathcal{B},\Gamma)=\mu_3 & P\in\Gamma\\ \mu(P)=p_3' & \langle\mathcal{B},P\rangle \nrightarrow_1\end{array}}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_{p_3\cdot p_3'} \langle E^e,\mathcal{B},\Gamma\setminus\{P\}\rangle}$$

The new rule A^p_{update} says that if the probability of updating intention selection at this step is p_3, the probability of selecting an unprogressable intention P is p_3', then the probability of selecting and removing it from the intention set at this step is $p_3 \cdot p_3'$.

Finally, when there is no agent-level operation available, we provide a default idle rule that transitions the agent to itself. This is required as DTMCs have must always have an outgoing edge probabilities that sum to 1. This allows for verification using probabilistic model checking tools (in Section 4.5). The self-transition rule is

$$A^p_{idle}\ \frac{\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \qquad \eta(\bot)=1}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_1 \langle E^e,\mathcal{B},\Gamma\rangle}$$

3.2 Probabilistic Intention-level Semantics

Figure 2b gives rules for evolving any single intention and each rule is either defined in either deterministic (*e.g.* rule *act*) or non-deterministic nature (*e.g.* rule *select*). Though most of deterministic rules are indeed expected and appropriate such as progressing a sequence of programs one by one, the rule for action execution may have uncertain outcomes (*i.e.* the effects on external environment), which may need probabilistic treatment. Similarly, we naturally extend non-deterministic rules such as plan selection *select* to probabilistic setting based on agent-specific information such as preference.

3.2.1 Probabilistic Action Outcomes

Agents execute actions that both interact with an external environment (*e.g.* pick up an object), and in-turn revise the internal belief base (*e.g.* the agent believes it holds the object). Recall that action execution is specified in CAN as follows:

$$act\ \frac{act:\varphi \leftarrow \langle\phi^-,\phi^+\rangle \qquad \mathcal{B}\vDash\varphi}{\langle\mathcal{B},act\rangle \rightarrow \langle(\mathcal{B}\setminus\phi^-\cup\phi^+),nil\rangle}$$

This states that an action applies only if the precondition φ holds, and the outcome is to update the belief base by adding and removing the belief atoms specified by ϕ^- and ϕ^+, respectively. Therefore, the action outcome is implicitly made deterministically by function

$$\mathcal{S}_a : \Lambda \rightarrow 2^{\overline{\mathcal{B}}} \times 2^{\overline{\mathcal{B}}}$$

Given an action, it returns a product of set of added and deleted atoms

In practice, we know the outcomes of an action are uncertain (*e.g.* due to actuator malfunctions). For example, an agent may execute an action to pick up an object but fail to do so because a robotic arm fails. In this case, updating the beliefs that an object is held can lead to misalignment between the true environment and the

agent's representation of it. To allow uncertain action outcomes, we can sample outcomes based on a probability distribution, *i.e.* with the following action outcome function:

$$\mathcal{S}_a^p : \Lambda \rightarrow Dist(2^{\overline{\mathcal{B}}} \times 2^{\overline{\mathcal{B}}})$$

A probabilistic action execution is defined by

$$act^p \frac{\mathcal{S}_a^p(act) = \mu \quad \mu(\phi^-, \phi^+) = p \quad \mathcal{B} \vDash \varphi}{\langle \mathcal{B}, act \rangle \rightarrow_p \langle (\mathcal{B} \setminus \phi^- \cup \phi^+), nil \rangle}$$

Importantly we do not expect programming language *implementations* based on these semantics to draw action outcomes probabilistically. Instead it is used solely for modelling, which allows us to capture environmental effects in a semantics where they are usually overlooked or ignored.

3.2.2 Probabilistic Plan Selection

BDI agents employ a user-provided plan library to respond to events. Each plan has i) a triggering event defining what event the plan can respond to, ii) a pre-condition defining what beliefs must hold for the plan to apply, and iii) a plan-body defining what steps should be taken to execute the plan. To address a pending event originating from the external environment, the agent retrieves a set of *relevant* plans, *i.e.* those with a matching triggering event, as specified by CAN rule

$$event \frac{\Delta = \{\varphi : P \mid (e' = \varphi \leftarrow P) \in \Pi \wedge e' = e\}}{\langle \mathcal{B}, e \rangle \rightarrow \langle \mathcal{B}, e : (\mid \Delta \mid) \rangle}$$

Given a set of relevant plans, the agent then selects an *applicable* plan (one where the precondition is true):

$$select \frac{\varphi : P \in \Delta \quad \mathcal{B} \models \varphi}{\langle \mathcal{B}, e : (\mid \Delta \mid) \rangle \rightarrow \langle \mathcal{B}, P \rhd e : (\mid \Delta' \mid) \rangle}$$

where $\Delta' = \Delta \setminus \{\varphi : P\}$. If there are no applicable plans a separate rule such as rule $\rhd_\perp$ in Fig. 2b propagates the failure.

Notice that the preceding select rule does not specify which plan should be selected in case of multiple applicable plans, *i.e.* it is non-deterministic. However, in many implementations, the choice is often made deterministically by a plan selection function of the following form:

$$\mathcal{S}_p : 2^{\overline{\mathcal{B}}} \times 2^{\Pi} \rightarrow \Pi \cup \{\perp\}$$

Given a belief base and a set of plans it returns an applicable plan or no applicable plan ($\perp$).

While a common heuristic is to select the plan with the highest order based on some characteristics (*e.g.* preference), it may not lead to globally optimal behaviours due to action side-effects. We argue that it should be possible to *prioritise* plan choice based on plan characteristics, but not assume a totally fixed ordering in order to allow exploration of non-highest order plans that might have better properties. This is akin to discrepancy search techniques [22] to go against the heuristic, and is particularly useful for declarative goals (*e.g.* rules G_{init} and $G_\rhd$ in Fig. 2b) to avoid always repeating the same plan.

To support non-strict orderings, we can sample the choice of applicable plans based on a probability distribution, *i.e.* with the following plan selection function:

$$\mathcal{S}_p^p : 2^{\overline{\mathcal{B}}} \times 2^{\Pi} \rightarrow Dist(\Pi \cup \{\perp\})$$

Using $\mathcal{S}_p^p$, a probabilistic *select* rule is defined by

$$select^p \frac{\begin{array}{c}\mathcal{S}_p^p(\mathcal{B}, \Delta) = \mu \quad \mu(\perp) = 0 \\ \varphi : P \in \Delta \quad \mu(\varphi : P) = p\end{array}}{\langle \mathcal{B}, e : (\mid \Delta \mid) \rangle \rightarrow_p \langle \mathcal{B}, P \rhd e : (\mid \Delta' \mid) \rangle}$$

where $\Delta' = \Delta \setminus \{\varphi : P\}$ and μ is the probability distribution returned from $\mathcal{S}_p^p$ such that any non-relevant and non-applicable plans are assigned the probability 0.

Trialling different distributions is possible by changing $\mathcal{S}_p^p$ which could, for example, be extracted from historical data. With our approach, it allows quantifying *exact* probabilistic effects of different $\mathcal{S}_p^p$ choices.

3.2.3 Probabilistic Concurrency

CAN also supports the execution of concurrent programs. To execute a concurrent plan-body

program, the agent can execute either part non-deterministically given in following two cases:

$$\|_1 \frac{\langle \mathcal{B}, P_1 \rangle \to \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, P_1 \| P_2 \rangle \to \langle \mathcal{B}', P_1' \| P_2 \rangle}$$

$$\|_2 \frac{\langle \mathcal{B}, P_2 \rangle \to \langle \mathcal{B}', P_2' \rangle}{\langle \mathcal{B}, P_1 \| P_2 \rangle \to \langle \mathcal{B}', P_1 \| P_2' \rangle}$$

To choose the part of concurrent programs to progress, we may also require flexibility to choose from a distribution by a selection function for a concurrent program $P_1 \| P_2$ as follows:

$$\mathcal{S}_c^p : 2^{\overline{\mathcal{B}}} \times P \to Dist(P \cup \{\bot\})$$

where $P \subseteq \Gamma$ is all possible plan-body programs, $Dist(P \cup \{\bot\}))$ is the set of discrete probability distributions over all possible plan-body programs (or a delta distribution to $\bot$ if no part of a concurrent program can be progressed).

We note that both rules $\|_1$ and $\|_2$ imply that the evolution of a concurrent program depends on another intention-level transition of either part of concurrent program and we need to account for this in the transition probabilities. To have a probabilistic intention-level transition for a concurrent program, we assume, for a chosen progressable intention $P_i \in P$, $\langle \mathcal{B}, P_i \rangle \to_{p_i'} \langle \mathcal{B}', P_i' \rangle$ holds where $i \in \{1, 2\}$, for example, if a plan selection for the given intention P_i is required based on $select^P$. Using $\mathcal{S}_c^p$ we can define a probabilistic extension for rules $\|_1$ and $\|_2$:

$$\|_1^p \frac{\mathcal{S}_c^p(\mathcal{B}, P_1 \| P_2) = \mu \qquad \mu(\bot) = 0 \qquad \mu(P_1) = p_1 \qquad \langle \mathcal{B}, P_1 \rangle \to_{p_1'} \langle \mathcal{B}', P_1' \rangle}{\langle \mathcal{B}, P_1 \| P_2 \rangle \to_{p_1 \cdot p_1'} \langle \mathcal{B}, P_1' \| P_2 \rangle}$$

$$\|_2^p \frac{\mathcal{S}_c^p(\mathcal{B}, P_1 \| P_2) = \mu \qquad \mu(\bot) = 0 \qquad \mu(P_2) = p_2 \qquad \langle \mathcal{B}, P_2 \rangle \to_{p_2'} \langle \mathcal{B}', P_2' \rangle}{\langle \mathcal{B}, P_1 \| P_2 \rangle \to_{p_2 \cdot p_2'} \langle \mathcal{B}, P_1 \| P_2' \rangle}$$

where μ is the probability distribution returned from $\mathcal{S}_c^p$ such that where $\sum_i p_i \cdot p_i' = 1$, $p_i, p_i' \in [0, 1]$, and $i \in \{1, 2\}$.

Finally, we reiterate that when there is no applicable plan available to select or neither of two concurrent programs is progressable, a separate rule of failure recovery ($\rhd_\bot$ in Fig. 2b) will propagate the failure to continue the transition. The rest of intention-level rules in Fig. 2b are automatically extended with a probability 1. The full rule set for the probabilistic extension of the CAN semantics is in Fig. 15.

3.3 Constructing Selection Functions

The probabilistic CAN semantic rules either transition with probability 1 or through probability distributions. These probability distributions are abstract, and the rules themselves do not specify how to construct them in practice. In this section, we present and extended syntax for CAN programs that allows agent programmers to define the specific distributions to be used.

3.3.1 Situation Value Functions

We introduce additional *syntax* for relevant agent programs (*e.g.* plans) that, through a process of normalisation, determines the correct probabilistic distributions. Following [14], we annotate programs using *situation value description* functions $\theta : 2^{\mathcal{B}} \to \mathbb{R}_{\geq 0}$. Intuitively these map the current situation, as described by the *current* beliefs, to a real valued number. We allow users to define θ functions as folds/aggregation functions as follows:

$$\langle d_0, \{(\varphi_1, d_1), \cdots, (\varphi_n, d_n)\}, f \rangle$$

where d_0 is a default value and values d_i are aggregated using function f (*e.g.* sum) whenever $\mathcal{B} \models \varphi_i$ holds.

In general, situation value functions are *dynamic* in that they respond based on the current set of beliefs representing the current situation the agent is in. A special case are static values, *e.g.* $\theta = \langle 0, \{(true, d_i)\}, + \rangle$ that do not depend on the current beliefs of the agent. For ease of notation, we allow users to denote these simply as the value, *e.g.* $\theta = d_i$, rather than giving the full function.

Situation value functions can then be attached to the following agent programs to determine the correct probabilistic selection function.

- **Plans** $e : \varphi \leftarrow P[\theta]$
- **Concurrency** $P_1[\theta_1] \parallel P_2[\theta_2]$
- **Events and Intentions** $e[\theta]$

Intuitively, plans with a higher (current) θ value should be selected more often. Concurrency annotations determine which branch should be preferred, while the event annotations determine

which event should be adopted first (given a set of possible events). Because intentions ultimately address external events, we measure the situation value of an intention by considering the value of its related external event, which suffices for our smart manufacturing example in Section 4.

Finally, to construct the probabilistic selection function for agent-level operations, we add three keywords A_{event}, A_{step}, and A_{update} (mirroring the agent-level rule names) that allow annotations with a situation value function, *i.e.* $A_{event}[\theta_1], A_{step}[\theta_2]$, and $A_{update}[\theta_3]$. The syntax is shown as part of agent configurations in lines 20 to 22 in Listing 2. Each situation value function describes the relative weight to selecting each agent-level rule. As usual, the resulting probabilities to select each agent-level rule are then determined through normalisation.

3.3.2 Selection Functions

We now describe how the selection functions are constructed given the situation value functions. We give the mapping for plan selection as an example, and the others follow similarly. We define

$$app(\mathcal{B}, \Delta) = \{P \in \Delta \mid P = \varphi : Q,\ \mathcal{B} \models \varphi\}$$

as a filter that chooses the applicable plans given a specific belief set $\mathcal{B}$ and set of plans Δ. Here we use Q to indicate a plan-body. The plan selection function is then defined as

$$\mathcal{S}^p_p(\mathcal{B}, \Delta) = \begin{cases} \mu & \text{if } app(\mathcal{B}, \Delta) \neq \emptyset \\ [\bot \mapsto 1] & \text{otherwise} \end{cases}$$

$$\mu = \left[P_1 \mapsto \frac{\theta_1(\mathcal{B})}{N} \ldots, P_n \mapsto \frac{\theta_n(\mathcal{B})}{N}, \bot \mapsto 0\right]$$

with $P_{i \in \{1,\ldots,n\}} \in app(\mathcal{B}, \Delta)$ and $N = \sum_{i=1}^{n} \theta_i(\mathcal{B})$. That is, for a non-empty set of applicable plans we normalise the situation values into the range $0 \leq p \leq 1$ allowing them to be used as a probability distribution. If there are no applicable plans, or the plan set is empty, we select $\bot$ with probability 1, allowing different Can rules (*e.g.* failure recovery $\rhd_\bot$ in Fig. 2b) to apply.

3.3.3 Action Outcomes

Action outcomes are statically defined based on estimates of environmental effects at design time. We attach the static situation value functions, *i.e.* values, to each effect using the follwing syntax:

$$act = \varphi \leftarrow [\langle \phi_1^-, \phi_1^+ \rangle [\theta_1], \ldots, \langle \phi_n^-, \phi_n^+ \rangle [\theta_n]]$$

As before, the specific probabilities are determined through normalisation.

Finally, we note that assigning static values to action outcomes has been considered extensively in the planning literature and has led to, *e.g.* probabilistic planning domain definition languages (PPDDL) [23], that consider multiple outcomes with associated probabilities (*e.g.* estimated from historical data).

4 Evaluation

We demonstrate, using a smart manufacturing example and existing probabilistic model checking tools, how to quantitatively analyse BDI agent programs. Specifically, we evaluate our probabilistic plan/event/intention selection against common strategies such as always selecting the most preferred plan. The results are promising, with the intention completion probability using probabilistic distributions being 97% higher than some strictly ordered plan and intention selection strategies.

We build on previous quantitative analysis, for the same example [20], by extending the experiments to include new agent-level operation selection strategies.

The models are freely available in BigraphER format online[1]. For quantitative analysis we use PRISM to check properties (through bigraph patterns) by importing the labelled DTMC produced by BigraphER. While we only give details of a single case study, users of the executable semantics can employ BigraphER to "run" models with different settings, *e.g.* external events, plan libraries, customised situation value functions.

4.1 Smart Manufacturing Example

We consider a robotic packaging scenario, *extended* from [24], where a robot packs products and moves them to a storage area. Products have specific temperatures and must be packed in a suitable wrapping bag to prevent decay. If the

[1] https://bitbucket.org/uog-bigraph/sosym-sefm21-special-issues/src/master/

product stays on the production line too long, the temperature increases and it is spoiled and lost. Given multiple waiting products (*i.e.* events to trigger the operation) on the production line, the robot must *choose* which to handle first (event selection). Once chosen, the robot must then decide which wrapping to use: either premium or standard (plan selection). Premium wrapping is expensive but always stops product decay and never breaks. On the other hand, standard wrapping is cheap, only works if the product temperature remains low, and has a risk of breaking (a negative action outcome). The (partially-executed) plan-bodies for the selected plans become intentions that handle the products. Among all current intentions, the agent also needs to decide which intention to progress further, for example, moving to the storage once finishing wrapping (intention selection)

Complexity arises from the following factors: (1) losses avoided depend on *when* a product is packed, (2) *when* a product is packed determines which wrappings are applicable; earlier packing means cheaper bags, (3) cheaper wrappings introduce uncertainty as they may break. A formal model of the agent system allows us to quantitatively reason about the robot's behaviours under this uncertainty and use these results as evidence, *e.g.* for regulatory certification. Furthermore, it can help improve the design of the robot, *e.g.* using a standard wrapping as often possible but within tolerable failure threshold.

4.2 Agent Design

We consider a simplified scenario with two products that are initially present on the production line, *i.e.* there are no dynamic events. The agent program is given in Listing 2 and we assume beliefs are in a propositional logic with numerical comparisons.

Products awaiting processing are captured by external events shown in lines 9 and 10, *e.g.* `e_product1` with its situation value function θ_{13} (explained below). The agent responds to the events using a declarative goal on line 2 that states it wants to achieve the state `success1` (*i.e.* wrapped and moved) through addressing the (internal) event `e_process_product1`; failing if `failure1` (*i.e.* dropped or decayed) ever becomes true. Two plans (in lines 3 and 4), which represent the different wrappings, can handle the event `e_process_product1` each with different situation value functions. Event `e_product2` is handled in a similar way (in line 5–7).

The description of actions are given from lines 12 to 19. There is a probabilistic outcome for the `move_product_standard1` action in line 13, such that it has a 10% chance of causing `failure1` by dropping the product accidentally, else it succeeds (adding `success1` to the beliefs), whereas `move_product_premium1` action always succeeds in line 15. In Section 4.5 we investigate how varying action success probability effects the overall outcomes in the dedicated section.

To construct probabilistic distributions, we encode the (discrete) temporal information for progress and deadline. Progress determines how far (in terms of agent steps) an agent is through an intention, while deadline determines how many steps we can make before the product spoils. Mirroring implementations, we update timings in the background, without executing an explicit action. In this case, the progress increases whenever a specific intention is stepped, whereas deadline decreases after a step of *any* intention. That is, we use *agent time* (*i.e.* agent steps which is implicit in the semantics) rather than real-time (as this requires a secondary clock) to remain agnostic to the actual time required for each agent step which can be difficult to anticipate at design stage due to the delay or variation in real process deployed on the hardware.

Figure 4a gives the specifications for quantitative reasoning. A short commentary is as follows. $de_1 = 10$ and $de_2 = 14$ are the initial deadlines of two external events: `e_product1` and `e_product2`. The precondition $\varphi_{11} = de_1 \geq 3$ indicates whether de_1 is greater than or equal to 3. The situation value function $\theta_{11} = \langle 1, \{\varphi_{11}, 1\}, sum \rangle$ indicates that if φ_{11} holds, then $\theta_{11}(\varphi_{11}) = 1 + 1 = 2$. The situation value description θ_{13} for the external event `e_product1` is defined as a function $(de_1 + pr_1)^{-3}$. Intuitively, if $de_1 + pr_1$ is smaller relative to other products, then it has been progressed less and the deadline is approaching, so it is more urgent. Finally, we have the situation value functions for the agent-level operations so that we ensure the highest weighting for the rule A_{event} when $pr_1 + pr_2$ is small (*i.e.* relatively low

Listing (2) Agent design of a smart manufacturing example.

```
// Plans
e_product1 : true ← goal(success1,e_process_product1,failure1)
e_process_product1 : φ11 ← wrap_standard1; move_product_standard1 [θ11]
e_process_product1 : φ12 ← wrap_premium1; move_product_premium1 [θ12]
e_product2 : true ← goal(success2,e_process_product2,failure2)
e_process_product2 : φ21 ← wrap_standard2; move_product_standard2 [θ21]
e_process_product2 : φ22 ← wrap_premium2; move_product_premium2 [θ22]
// External events
e_product1 [θ13]
e_product2 [θ23]
// Action descriptions
wrap_standard1 = true ← [⟨φ1− = ∅, φ1+ = {product_1_packed}⟩ ↦ 1]
move_product_standard1 = product_1_packed ← [⟨φ2− = ∅, φ2+ = {success1}⟩ ↦ 0.9,
wrap_premium1 = true ← [⟨φ1− = ∅, φ1+ = {product_1_packed}⟩ ↦ 1]
move_product_premium1 = product_1_packed ← [⟨φ2− = ∅, φ2+ = {success1}⟩ ↦ 1]
wrap_standard2 = true ← [⟨φ3− = ∅, φ3+ = {product_2_packed}⟩ ↦ 1]
move_product_standard2 = product_2_packed ←
    [⟨φ4− = ∅, φ4+ = {success2}⟩ ↦ 0.9, ⟨φ4− = ∅, φ4+ = {failure2}⟩ ↦ 0.1]
wrap_premium2 = true ← [⟨φ3− = ∅, φ3+ = {product_2_packed}⟩ ↦ 1]
move_product_premium2 = product_2_packed ← [⟨φ4− = ∅, φ4+ = {success1}⟩ ↦ 1]
Aevent [θ31]
Astep [θ32]
Aupdate [θ33]
```

Initial Deadlines	Pre-conditions	Situation Value Descriptions
$de_1 = 10$	$\varphi_{x1} = de_x \geq 3$	$\theta_{x1} = \{1, \{\varphi_{x1}, 1\}, sum\}$
$de_2 = 14$	$\varphi_{x2} = de_x \geq 0$	$\theta_{x2} = \{1, \{\varphi_{x3}, 1\}, sum\}$
	$\varphi_{x3} = 3 \geq de_x \geq 0$	$\theta_{x3} = (de_x + pr_x)^{-3}$
		$\theta_{3y} = (4 - y) \cdot (pr_1 + pr_2)^y$

(a) Situation value functions where $x \in \{1, 2\}$ and $y \in \{1, 2, 3\}$. de_i is the deadline for product i.

Figure 4: Agent design employing the syntax of Section 2.1 with the situation value functions.

overall progress). When $pr_1 + pr_2$ gets bigger, the power of y in $(4 - y) \cdot (pr_1 + pr_2)^y$ ensures the rule A_{step} and A_{update} to have a higher weight than A_{event}, and A_{update} higher than A_{step} as well. Importantly, all of deadline values and the choice of situation value descriptions are made by the agent designer, *i.e.* $(de_1 + pr_1)^{-3}$ was their choice. Our approach enables the analysis of alternative functions quantitatively, before deploying the agent.

4.3 Selection Strategies

We experiment with multiple selection strategies used by the agent (in Listing 2), including: agent-level operation selection, event/intention selection, and plan selection, which are standard in work *e.g.* [7]. A summary is given in Table 1, and we are particularly interested in selection strategies that use dynamic distributions based on domain-specific information (excluding uniform random selection strategies). A short commentary for each selection mechanism is given next.

4.3.1 Agent-level Operation Selection Strategies

At each agent step an agent can either: incorporate any pending external events through CAN rule A_{event}, select an intention and execute a step through CAN rule A_{step} (according to intention-level semantics), or remove unprogressable intentions from intention set using rule A_{update}.

Here, agent-level operation selection strategies control which agent-level operation will be applied, *e.g.* select a new event or progress an existing intention. In our smart manufacturing

Table 1: Selection strategies.

Agent-level Operation Selection Strategies (AOSS)	Event and Intention Selection Strategies (EISS)	Plan Selection Strategies (PSS)
SIP: Select In Priority **ProD**: Progress Distribution	**SMU**: Select Most Urgent **FIFO**: First-In-First-Out **RR**: Round-Robin **UD**: Urgency Distribution **CUD**: Conditioned Urgency Distribution **OCUD**: Optimised Conditioned Urgency Distribution	**SMP**: Select Most Preferred **PreD**: Preference Distribution

example, this affects *when* a waiting product is initially handled, and *how long* it takes to pack a product. We use two different selection strategies: the **SIP** (Select In Priority) strategy selects agent-level operations in a priority order: pending events are adopted first, then intentions are progressed, finally when there are no events/intentions it finally removes them from the intention set. The **ProD** (Progress Distribution) strategy instead selects an agent-level operation from a dynamic probability distribution based on the current progress of the agent. Initially we bias towards adopting events (to give the agent work to do), and, as the agent progresses, we increase the probability of progressing intentions instead (to finish tasks before becoming overwhelmed). Finally, we garbage collect unprogressable intentions near the end of a run (when there is less work to do). The functions to compute this distribution are in lines 20–22 in Listing 2.

4.3.2 Event/Intention and Plan Selection Strategies

For event and intention selection, the **SMU** (Select Most Urgent) strategy always selects the intention closest to the deadline. **FIFO** (First-In-First-Out) and **RR** (Round-Robin) are fixed orders where the former always selects the intention which arrives first and the latter selects each intention in turn. The **UD** (Urgency Distribution) strategy selects an intention by sampling from a distribution where situation value function is given by $(de+pr)^{-3}$. Unlike the **UD**, the **CUD** (Conditioned Urgency Distribution) only deems an intention urgent if the product is not packed or spoiled. As such, it will not select an intention in which the product is packed when there is another intention whose product is not packed. Finally, **OCUD** (Optimised Conditioned Urgency Distribution) selects an intention similarly to **CUD** but the situation value description is revised to be $|de + pr - steps\_expected|^{-3}$, which accounts for the steps remaining to pack a product (to avoid spoilage).

For plan selection, **SMP** always selects the highest weighted plan, while **PreD** selects a plan by sampling distribution based on preference.

4.4 Encoding in Bigraphs

In this section we show how we encode agent design, probabilistic agent semantics, different strategies, and logical predicates in bigraphs. We begin with a brief introduction to bigraphs.

4.4.1 Bigraphs

Bigraphs are a universal graph-based modelling formalism introduced by Milner [25], with conditional, priority, parameterised, and probabilistic extensions [18, 26]. They have an algebraic and diagrammatic form, we employ mainly the latter here.

An example bigraph is in Fig. 5a. It consists of a set of entities, *e.g.* A, B, drawn as (coloured) shapes[2]. Entities can be related through *nesting* (to arbitrary depth), *e.g.* the B entities inside A. Entities can also be related through hyperlinks (permitting any-to-any links rather than just one-to-one as is usual), such as the green link between the B and C entities. Entities have a *fixed* number

[2]We often use the shape to denote the entity type to reduce the need for excessive labelling.

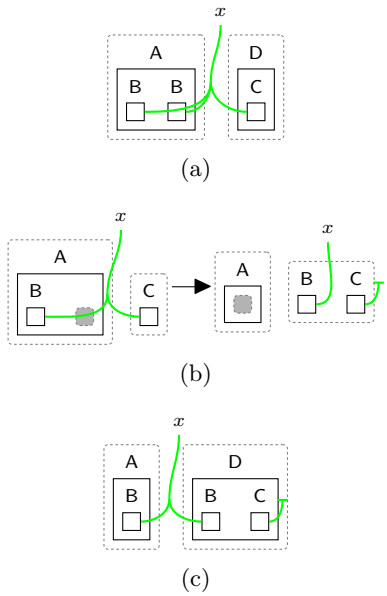

Figure 5: (a) example bigraph, (b) reaction rule, and (c) result after applying (b) to (a).

of links, called the arity, although a link can be disconnected as shown by the C entity in Fig. 5c. The name x means this link is *open* and can connect to other (unspecified) parts of the system. Likewise, the filled grey rectangles denote that other (unspecified) entities can exist here. Dashed unfilled rectangles are *regions* that represent parallel parts of the system: that is, these two regions can, but do not have to, share a single parent in some larger system model.

A bigraph represents a system at a single point in time. To allow models to evolve over time we can specify *reaction rules* of the form $L \longrightarrow R$, where L and R are bigraphs. Intuitively, a bigraph B evolves to B' by matching and rewriting an occurrence of L in B with R. Such a reaction is indicated with $B \longrightarrow\!\!\!\!\triangleright\, B'$. Given an initial bigraph and set of reaction rules, we can derive a transition system capturing all possible behaviours.

An example reaction rule is in Fig. 5b, which models the disconnection of B and C and also removes the nesting of B in A. The filled grey rectangles are called *sites* and represent parts of the model, below some entity, that have been abstracted away. That is, it allows matching on an A with *multiple* children. Without the site, the rule would only match when A had a single B child. Similarly, the use of the open name x means that the B can be connected not just to the C but also elsewhere, in this case the other B. As B remains connected to x the link remains connected in the result (likewise if it had been C connected to x then it would remain connected in the result). Reaction rules can affect both linking and placement, as shown here with the B entity also moving next to C.

Priority rewriting [26] permits an ordering on rules, defined by specifying classes of rules and an ordering between the classes. A reaction of lower priority can be applied only when no reaction of higher priority is applicable. Probabilistic bigraphs [18] permit rules be weighted, *e.g.* $\mathtt{t}_1 = L_1 \xrightarrow{2} R_1$ and $\mathtt{t}_2 = L_2 \xrightarrow{1} R_2$, such that if both (and only) $\mathtt{t}_1$ and $\mathtt{t}_2$ are applicable then $\mathtt{t}_1$ is twice as likely to apply as $\mathtt{t}_2$. We allow rule priorities, where a reaction of lower priority can be applied only if no reaction of higher priority is applicable. We write $\{\mathtt{r1}\} < \{\mathtt{r2}\}$ to denote when sets of rules have higher priority.

The encoding of probabilistic Can in probabilistic bigraphs follows directly from the encoding for the non-probabilistic versions [19]. For example, the encoding of agent design remains the same while there is a syntax change from `-->` to `-[1]->` in BigaphER for any deterministic intention-level semantics rule that is assigned with default probability 1. Additional rules are required, and some rules must be updated, to support different selection strategies and we describe these changes in the coming section. Importantly, the different strategies define a *family* of related models rather than a single model with different strategy selection. This means the same rule might appear differently, *e.g.* with different parameters, depending on the specific strategies we are implementing.

Notation

We use fonts to distinguish between Can semantics rules, *e.g.* A_{event}, bigraph reaction rules, *e.g.* `choose_a_event`, and bigraph entities, *e.g.* Aevent.

4.4.2 Encoding Agent-level Operation Selection Strategies

To encode the selection of agent-level operation, we add the following new reaction rules:

$\{$`choose_a_update`, `choose_a_step`, `choose_a_event`$\}$

These reaction rules determine the next agent-level operation *e.g.* if `choose_a_event` (illustrated in Fig. 6) is applied, then the agent-level rule A_{event} is applied at the next step.

Priorities can be used to implement selection strategies such as **SIP**, which selects agent operations in a priority order. That is, we can assign rule priorities:

$$\{\texttt{choose\_a\_update}\} < \{\texttt{choose\_a\_step}\} < \{\texttt{choose\_a\_event}\}$$

To encode the **ProD** strategy, which selects an agent-level operation rule from a dynamic distribution, we employ parameterised reactions that define a *family* of rules. For example, reaction $\texttt{r}(k)$ generates a set of rules $\texttt{r}(k_1), \texttt{r}(k_2), \ldots$ for all values of k. We then define the **ProD** strategy by:

$$\{\texttt{choose\_a\_event}(pr_1, pr_2), \texttt{choose\_a\_step}(pr_1, pr_2), \texttt{choose\_a\_update}(pr_1, pr_2)\}$$

where pr_1 and pr_2 denote the steps being applied to events `e_product1` and `e_product2`, respectively. Recall in Section 4.3.1, we have the situation value of selecting each agent-level rule (using the same syntax as lines 20 - 22 in Listing 2):

$$A_{event}[\theta_1],\ A_{step}[\theta_2],\ A_{update}[\theta_3]$$

where the situation value function θ_i is $(4-i)\cdot(pr_1+pr_2)^i$ where m_i is a positive number, $i \in \{1,2,3\}$. Therefore, reaction rule `choose_a_event` has weight $3\cdot(pr_1+pr_2)$, `choose_a_step` $2\cdot(pr_1+pr_2)^2$, and `choose_a_update` $(pr_1+pr_2)^3$. As an example, reaction rule $\texttt{choose\_a\_event}(pr_1, pr_2)$ is illustrated in Fig. 8. This is a modification of Fig. 6 including the progress step of two products.

The weights are normalised (automatically by BigraphER) based on which reaction rules are applicable to a given (bigraphical encoding of) agent state. For example, if $pr_1 = 0$ (`e_product1` is not adopted in the intention set) and $pr_2 = 1$ (`e_product2` is adopted in the intention set and ready for being progressed), both `choose_a_event` and `choose_a_event` are applicable. Then we have the weight of rule `choose_a_event` is 3 and `choose_a_step` is 2. After normalisation, the final probability of selecting corresponding CAN rule A_{event} is 0.6, while the probability of selecting the corresponding A_{step} is 0.4. Such a distribution indicates that at this early stage, the agent is more likely to adopt any pending event over progressing the existing intention. Importantly, these reaction rules are all in the same priority class (in the bigraph models) meaning any *could* be applied at each step, and the probabilities indicate relative likelihoods.

4.4.3 Encoding Event/Intention and Plan Selection Strategies

To encode the event/intention selection strategies, we modify the reaction rules for corresponding to agent-level rules of A_{event}, A_{step}, and A_{update} from our previous work [19].

The first event/intention selection strategy is to always select the most urgent (**SMU**). To encode this, we can continue to employ the parameterised reaction. Unlike using parameters for obtaining the weight in **ProD** previously, parameter is used to control whether the first event or its related intention should be selected. For example, the parameterised reaction rule $\texttt{a\_event}(1)$ only select the `e_product1`, *i.e.* parameter 1 corresponds to the product identifier number. Since we know, in Listing 2, that the deadline of `e_product1` is nearer (hence more urgent) than `e_product2` We can have the following priority to implement the **SMU** strategy:

$$\{\texttt{a\_event}(2), \texttt{a\_step}(2), \texttt{a\_update}(2)\} < \{\texttt{a\_event}(1), \texttt{a\_step}(1), \texttt{a\_update}(1)\}$$

Similarly, we can have the following priority reactions for First-In-First-Out (**FIFO**) event/intention selection strategy under the assumption that `e_product2` arrives earlier than `e_product1` (otherwise it will be same as **SMU**):

$$\{\texttt{a\_event}(1), \texttt{a\_step}(1), \texttt{a\_update}(1)\} < \{\texttt{a\_event}(2), \texttt{a\_step}(2), \texttt{a\_update}(2)\}$$

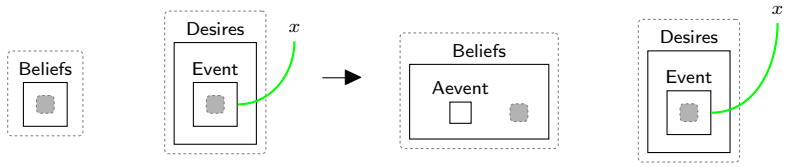

Figure 6: Reaction rule `choose_a_event`.

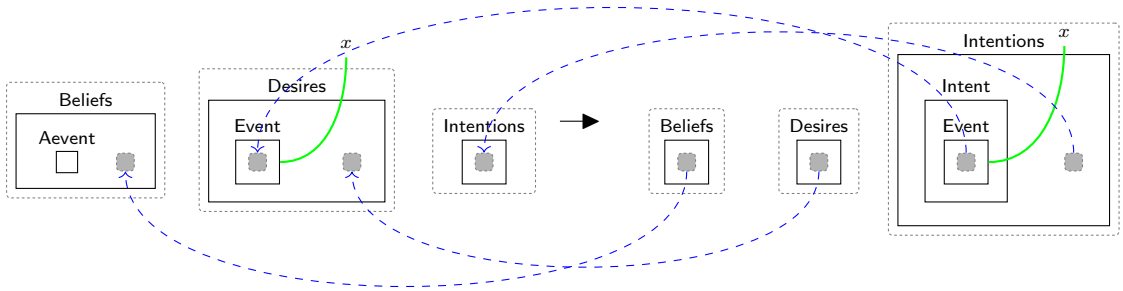

Figure 7: Reaction rule `a_event` adds an event to the intention set if it matches token `Aevent`. The dashed arrows (called the instantiation map in bigraphs) forces the site in the right hand side to be the copy of the site on the left.

To encode the Round-Robin (**RR**) strategy, we also use some auxiliary token to control the order of selection. For instance, we have `a_step` to select an intention which is linked with an entity Pointer (a token showing that now it should be progressed). In our example with two intentions (corresponding to two products), the token Pointer will be moved to one intention after the other one is processed one step (through rule `a_step`). Illustrated in Fig. 9, the execution of an intention linked through x_1 with an event labelled with Pointer will result in moving the Pointer to another event (linked through x_2) and vice versa.

To encode urgency distribution (**UD**), we have the following set of parameterised reactions for agent-level rules:

$$\{\texttt{a\_event}(pr, de),\ \texttt{a\_step}(pr, de),\ \texttt{a\_update}(pr, de)\}$$

where pr and de denote how many steps have been progressed and are left for an event or its related intention. Each rule has the weight of $1/(pr+de)^3$ (according to Section 4.3.2).

Conditioned urgency distribution (**CUD**) is same as **UD**, but only deems an intention urgent if the product is not packed or spoiled. To encode it, we employ conditional bigraphs [26] that allow *application conditions* to specify contextual requirements within the bigraphical system. As such, we only need to add the contextual requirement to reaction rule $\texttt{a\_step}(pr, de)$ (illustrated in Fig. 10).

The optimised conditioned urgency distribution (**OCUD**) is the same as (**CUD**) but with weight $|de + pr - steps\_expected|^{-3}$, where *step_expected* accounts for the number of steps that is expected to pack a product (to avoid spoilage). This value can be obtained through a simulation on one single product in BigraphER.

Finally, the similar priority (resp. parameter) approach can be used to encode the plan selection strategy **SMP** which selects the highest weighted plan and **ProD** which selects a plan by sampling distribution based on preference.

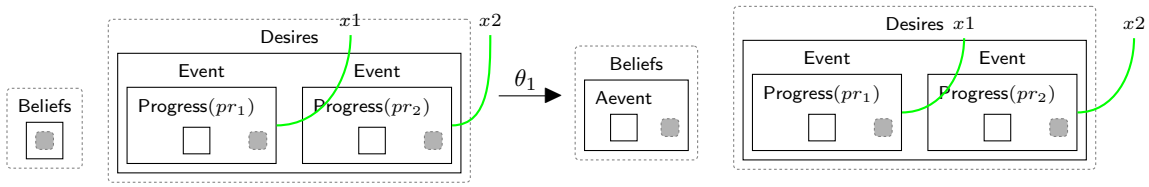

Figure 8: Reaction rule `choose_a_event`(pr_1, pr_2) where we use parameterised entities, *e.g.* Progress(pr_1), to represent the steps of an event which has been progressed and $\theta_1 = 3 \cdot (pr_1 + pr_2)$.

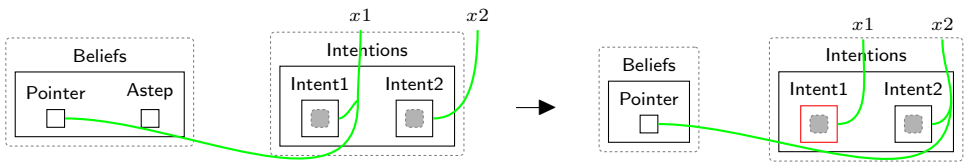

Figure 9: Reaction rule `a_step` encodes Round-Robin intention selection via a token entity Pointer that moves after each step. Entity Intent, highlighted in red, denotes that this intention will be progressed through intention-level rules.

4.4.4 Encoding Probabilistic Action Outcomes

To encode probabilistic outcomes, we extend the representation of action with a set of outcomes. Each outcome is pre-assigned with a parameterised bigraph entity EffWeight(n). Figure 11 shows the bigraph representation of action `move_product_standard1` from line 13 in Listing 2. To execute an action with probabilistic outcomes, we can encode intention-level rule act^p (in Section 3.2.1) as a parameterised reaction illustrated in Fig. 12 with transition weight n based on the given EffWeight(n) . This is then normalised to a probability by BigraphER.

4.4.5 Intention Success and Failure

CAN does not indicate whether an intention has completed successfully or with a failure. This is, A_{update} (Fig. 2a) removes a completed intention from the intention base, regardless if it had completed successfully (was the *nil* program) or if it could not make any further progress (failed). Following previous work [19] (which encodes standard CAN semantics), we overcome this limitation in two ways: 1. we add identifiers to each intention using the event name that generated the intention (not possible in CAN semantics); 2. We encode the CAN rule A_{update} as two different bigraph reaction rules: one handling a successfully completed intention, and the other a failed intention. This allows the rules to add additional entities to track intention state (either Success or Failure).

To add the labels to output DTMC states, we use two bigraph *patterns*, for success and failure, as shown in in Fig. 13. Once states are labelled, can use these in an *eventually* PCTL formulae for PRISM. This gives a general approach to reason about each intention individually, or a combination of intentions (through conjugation). For example, we use the formula $\mathcal{P}_{=?}\mathbf{F}[S(1) \wedge S(2)]$ in our packing use-case, which computes the probability that both products are processed successfully.

4.5 Analysis

Table 2 gives the probability of processing the products successfully or with a failure, under different agent-level operation selection, and event/intention selection strategies, with **SMP** chosen for plan selection. We use the shorthand (X1,Y2) to stand for $\mathcal{P}_{=?}\mathbf{F}[X(1) \wedge Y(2)]$ where X and Y are drawn from $\{S, F\}$ denoting success or failure.

We see the necessity for good event/intention selection, with the first three combinations *never* successfully processing both products, *i.e.* (S1, S2). Using **UD**, it starts to have limited success

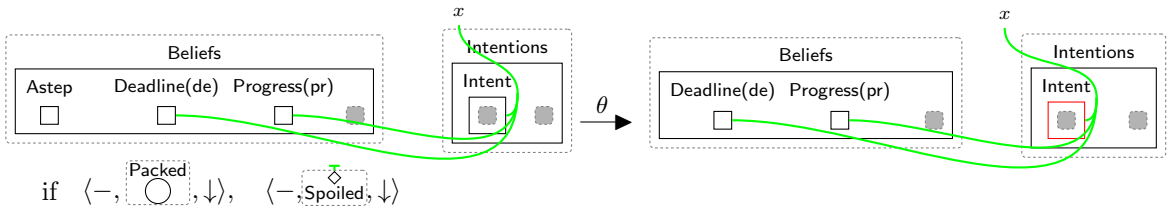

Figure 10: Conditional reaction rule a_step(pr, de) for **CUD** strategy with $\theta = 1/(pr+de)^3$. The symbol − indicates a negative condition *i.e.* that the bigraph of the condition should *not* appear/be matched. Packed and Spoiled are bigraphs representing the unwanted product statues. The ↓ means we do not want these states to appear in (any of) the sites. The red highlighted Intent on the right hand side means this intention will be progressed by further rules.

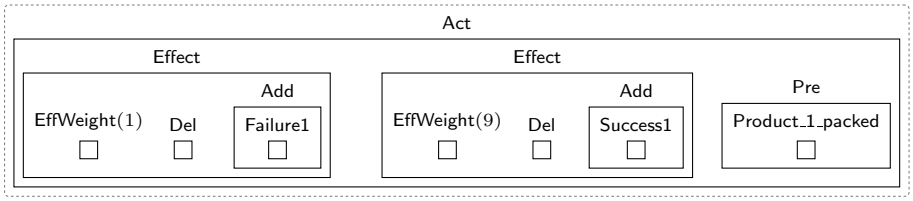

Figure 11: Bigraph representation of actions (Act) with a set of outcomes (Effect) each one containing a parameterised entity (EffWeight(n)) indicating its weight.

($p = 0.06$ for **SIP** and $p = 0.05$ for **ProD**). With **UD**, the chance of succeeding with product 1 increases to more than 50% whereas the failure of product 2 is nearly 72%. This indicates the weighting function is skewed toward product 1 at the detriment of product 2, leading to the improved **CUD** strategy. This is a key advantage of our approach: discovering potential pitfalls and trialling new strategies without changing the underlying agent programs and semantics. Similar reasoning, that now product 2 was succeeding more often under strategy **OCUD** being trialled with extremely good success rates, *e.g.* $p = 0.97$. We should never expect the probability of (S1, S2) = 1 due to the action outcome uncertainty (*e.g.* the wrapping bag breaks).

We also see a better performance of **SIP** strategy for agent-level operation selection. For example, the probability of successfully processing both products with the last three event/intention strategies under **ProD** is consistently lower (though by margin) than those under **SIP**. In general, any success rate improvement, even marginally, should be used as it can result in great savings—particularly in large scale processes, *e.g.* an expected two-product successful behaviour tending to occur 97% of the time instead of 90%. It shows that it is better to get the event adoption done in the beginning of the agent operation, updating unprogressable intention in the end, and the usual intention progress in the middle. In particular, **ProD** has a significantly detrimental effect when having **RR** for plan selection, with $p = 0.8$ for (S1, S2). The reason is that under **ProD** the agent may still continue to progress the same event just after it has been adopted, or remove an unprogressable intention just after progressing it. Both of the situations can lead to the other product being left there too long and becoming spoiled before it needs to be packed. Interestingly, agent-level operation selection seems to make no difference for event/intention selection strategies (*i.e.* **SMU** and **FIFO**).

Table 3 gives the probability of processing the products either successfully or with a failure, under different event/intention selection strategies and plan selection strategies, but with **SIP** for agent-level operation selection listed in Table 1. In this example, we find that plan selection has limited effect compared to event/intention selection, but nevertheless positive effects, which is key to this application. This itself is a valuable insight,

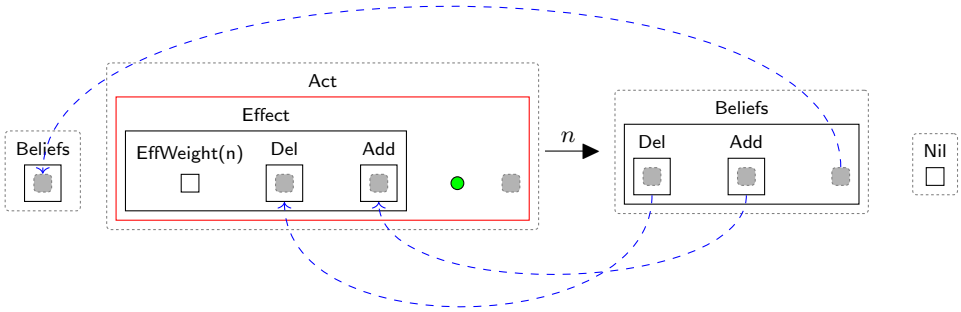

Figure 12: Bigraph rule for $action^p$ executing an action with an effect having the parameterised entity (EffWeight(n)). The dashed arrows (called the instantiation map in bigraphs) forces the site in the right hand side to be the copy of the site on the left. The green circle stands the bigraph of the applicability of pre-condition of this action and the red highlighted entity Act implies that this action is to be executed.

Table 2: Probability of (product 1, product 2) completing successfully/with failure for different agent-level operation selection and event/intention selection strategies using the **SMP** (Select Most Prefered) plan selection. Remaining abbreviations are in Table 1

		EISS					
		SMU		**FIFO**		**RR**	
AOSS	**SIP**	(S1,S2) 0	(S1,F2) 0.9	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0
		(F1,S2) 0	(F1, F2) 0.1	(F1,S2) 0.9	(F1,F2) 0.1	(F1,S2) 1	(F1,F2) 0
	ProD	(S1,S2) 0	(S1,F2) 0.9	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0.1
		(F1,S2) 0	(F1,F2) 0.1	(F1,S2) 0.9	(F1,F2) 0.1	(F1,S2) 0.1	(F1,F2) 0.8
		UD		**CUD**		**OCUD**	
AOSS	**SIP**	(S1,S2) 0.06	(S1,F2) 0.45	(S1,S2) 0.51	(S1,F2) 0	(S1,S2) 0.97	(S1,F2) 0
		(F1,S2) 0.22	(F1,F2) 0.27	(F1,S2) 0.48	(F1,F2) 0.01	(F1,S2) 0.03	(F1,F2) 0
	ProD	(S1,S2) 0.05	(S1,F2) 0.47	(S1,S2) 0.48	(S1,F2) 0.04	(S1,S2) 0.9	(S1,F2) 0.05
		(F1,S2) 0.17	(F1,F2) 0.31	(F1,S2) 0.46	(F1,F2) 0.02	(F1,S2) 0.04	(F1,F2) 0.01

and given the complexity of agent behaviours, determining this expected probability precisely, without such a model, would be difficult. In particular, we can see that when the event/intention selection is in a one-by-one manner, selecting plans from a dynamic distribution is much more useful, *e.g.* an expected two-product failure behaviour tending to occur 7% of the time instead of 10%.

The effects of different action outcomes are shown in Figure 14 where the probability of standard wrapping failing is increased from 10% to

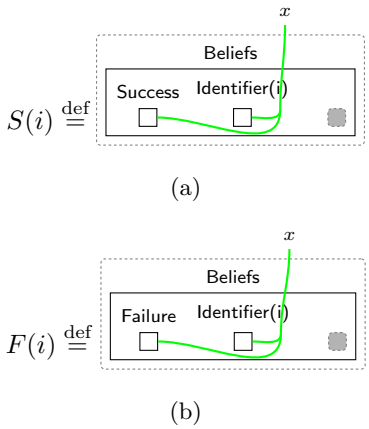

Figure 13: Bigraph patterns for checking intention success and failure. (a) $S(i)$: product i completes successfully. (b) $F(i)$: product i completes with failure.

90% for three strategy pairs: (**SIP**, **SMU**, **PreD**), (**SIP**, **RR**, **PreD**), and (**SIP**, **OCUD**, **PreD**)

We can see that negative action outcomes have a much larger effect on strictly ordered intention selection (**SMU**), *e.g.* the probability of (S1,F2) decreases from over 90% to below 40%. Meanwhile, (**SIP**, **OCUD**, **PreD**) is more robust to action outcome changes. For example, the probability of (S1, S2) in it has a minor decrease of no more than 20%. This is due to increased interleaving of these two intentions, rendering the standard wrapping inapplicable more often. In particular, in (**SIP**, **RR**, **PreD**), the strategy of round-robin renders the standard wrapping inapplicable at all times when handling product 2. As such, the probability changes of standard wrapping failing has no impact to the final result in this case.

5 Discussion

We reflect on the insights gained by constructing a probabilistic extension of CAN language, including the process of building bigraph models. We detail our first-hand experience of the value and limits of the bigraph approach applied to agent languages and their policies, which is not included in our previous work *e.g.* [20].

By building on an existing encoding of CAN in bigraphs [19], much of the probabilistic extension required limited effort. For example, most (deterministic) CAN semantics rules are modified to be probabilistic rules with a probability 1. For bigraphs, this requires the change of a reaction rule from $L \longrightarrow R$ to $L \xrightarrow{1} R$. The design of each agent (*e.g.* plan library) remains unchanged. This is because we focus on probabilistic *behaviours* rather than the probabilistic *knowledge* (*e.g.* probabilistic belief bases [27] that we detail in Section 6). Although it is promising to analyse an agent with both probabilistic behaviours and knowledge, it remains a challenging task in the context of verification given the computational complexity of uncertainty theories and their revision strategies. A feasible starting point of this integration can be an executable encoding of a belief base of a BDI agent in any form of these uncertainties theory together with its revision strategies in bigraphs before the final step of verification analysis, which we leave it as future work.

A characteristic of CAN is that the transition rules can be given incrementally, *i.e.* a modular operational semantics. In this case, the modularity in CAN separates how to evolve an intention (*i.e.* the intention-level semantics) from how to evolve the whole agent (*i.e.* the agent-level semantics). This approach has its merits, for example, we can easily extend or modify one side of the semantics (*e.g.* the agent-level) without altering the other one. As such, it allows us to separate concerns (decreasing errors) efficiently in the modelling plan/event/intention selection strategies. In particular, such a modular operational semantics turned out to be beneficial when analysing various combination of these plan/event/intention selection strategies under different action outcomes (seen in Section 4.5).

When modelling different plan/event/intention selection strategies, we found that BigraphER provides expressive and highly flexible features to construct these. For example, the priority classes of BigraphER naturally supports ordered selection strategies (*e.g.* First-In-First-Out) while conditional rules allow fixed schedules (*e.g.* Round-Robin) through auxiliary token entities. These strategies are often domain-independent (regardless of agent designs) as they do not require the details of specific intention to make a decision.

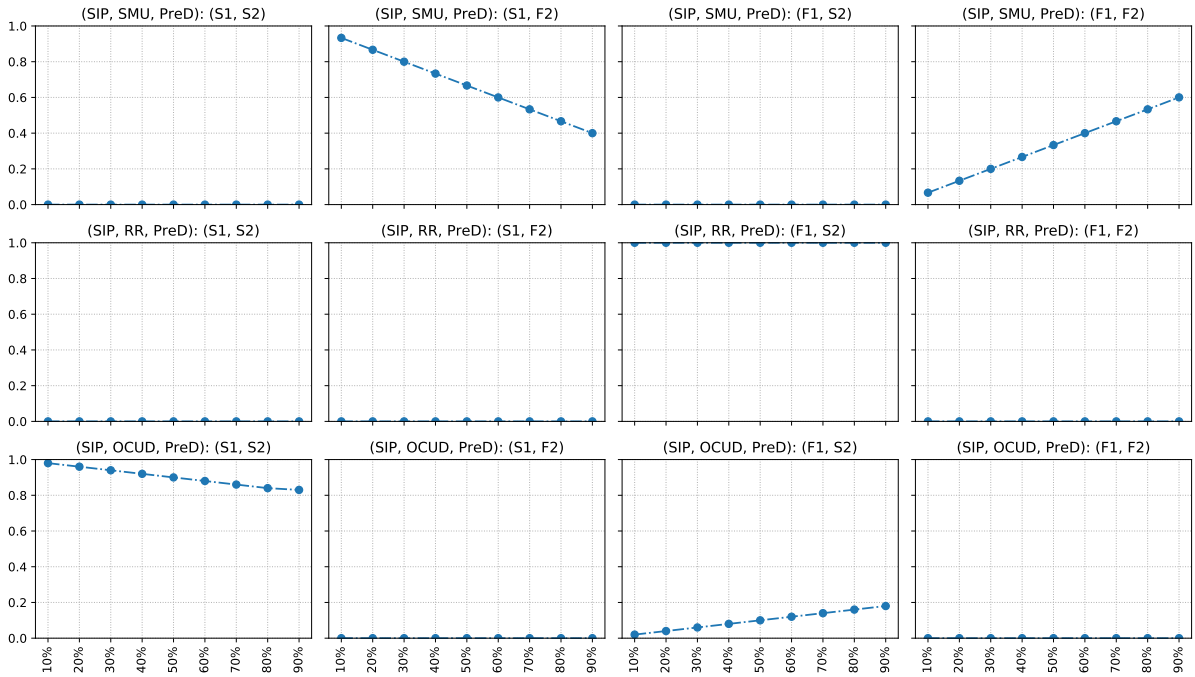

Figure 14: Probability of reaching the final state (product 1, product 2) with increasing failure probability in (**SIP**, **SMU**, **PreD**), (**SIP**, **RR**, **PreD**), and (**SIP**, **OCUD**, **PreD**).

The parameterised reaction rules in BigraphER also play a key role in probabilistic selection (especially from dynamic distribution based on run-time domain-specific information). These probabilistic selection strategies are likely to be domain-dependent as they require the domain-specific knowledge, *e.g.* deadlines. In practice, we often needed a set of new rules to capture some domain specific information updates. For example, the temporal information of progress and deadline in smart manufacturing case are maintained as separately: to increase the progress of an event whenever either such an event or its related intention is stepped, whereas to decrease the deadline of all events after an application of any agent-level rule. Mirroring implementations, we update timings in the background through applying instantaneous reaction rules [15] on the bigraphs. To write these new instantaneous reaction rules, it is often sufficient to have some form of discrete step update/manipulation rules. As these instantaneous rules do not show up in the resulting transition system, it does not affect our analysis on the agent behaviours, which is the focus of our work. We also note that these temporal timings are treated as agent steps which suffices in our case (rather than real times). However, a domain-specific mapping of real time of each agent step on different agent programs can be specified.

Our modelling approach comes with some limitations. Firstly, our approach does not naturally support mixed strategies, *e.g.* starting with an ordered strategy before swapping to probabilistic distribution. Modelling mixed strategies could be possible by providing conditioned selection strategies. In this approach, certain strategies are only applicable if related conditions are held in the environment. To model this in bigraphs, we could constrain strategies using pre-defined ranges of agent-operation parameter steps, *e.g.* use strategy x when deadline is less than 5. Secondly, there is a growing amount of work employing external advanced decision-making tools to solve selection problems (which we detail Section 6). Our approach cannot be compared directly with complex external decision-making techniques as advanced decision-making tools are often black-box techniques that we can not easily derive a discrete formal model for. For example, [28] formalises the intention selection problem in BDI

Table 3: Probability of product 1, product 2 for the properties, *e.g.* (S1, S2) with different event/intention selection strategies and plan selection strategies, and **SIP** (Select In Priority) for agent-level operation selection given in Table 1.

		EISS					
		SMU		**FIFO**		**RR**	
PSS	**SMP**	(S1,S2) 0	(S1,F2) 0.9	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0
		(F1,S2) 0	(F1, F2) 0.1	(F1,S2) 0.9	(F1,F2) 0.1	(F1,S2) 1	(F1,F2) 0
	PreD	(S1,S2) 0	(S1,F2) 0.93	(S1,S2) 0	(S1,F2) 0	(S1,S2) 0	(S1,F2) 0
		(F1,S2) 0	(F1,F2) 0.07	(F1,S2) 0.93	(F1,F2) 0.07	(F1,S2) 1	(F1,F2) 0
		UD		**CUD**		**OCUD**	
PSS	**SMP**	(S1,S2) 0.06	(S1,F2) 0.45	(S1,S2) 0.51	(S1,F2) 0	(S1,S2) 0.97	(S1,F2) 0
		(F1,S2) 0.22	(F1,F2) 0.27	(F1,S2) 0.48	(F1,F2) 0.01	(F1,S2) 0.03	(F1,F2) 0
	PreD	(S1,S2) 0.06	(S1,F2) 0.45	(S1,S2) 0.51	(S1,F2) 0	(S1,S2) 0.98	(S1,F2) 0
		(F1,S2) 0.22	(F1,F2) 0.27	(F1,S2) 0.48	(F1,F2) 0.01	(F1,S2) 0.02	(F1,F2) 0

agents as a planning problem in the PDDL description language [29]. A possible way to compare with selection strategies offered by this advanced decision-making tools is to employ models at runtime by taking model updates as inputs from external decision-making tools. Finally, we do not model costs/rewards, *e.g.* the price of the wrapping bags in Section 4.2. Utilising costs/rewards, we could perform multi-objective optimisation *e.g.* achieving different success rates and robustness to action outcomes while keeping the overall cost low.

Besides ensuring the agent model is correct, it is important to make sure the model deals with the right issue and helps ask the right questions. Our computational models can help agent programmers understand the behaviours of the agent they design before they are even employed. In other words, these allow agent programmers to do virtual "what if?" experiments—even changing the rules of how this detail operates—before we try things out for real. For example, the agent designer can use our models to understand the potential consequences of choices of different selection strategies quantitatively, which selection strategy has a dominant effect regard to the task completion in the given scenario, and how to parameterise the best probabilistic distribution, all of which our approach supports and have demonstrated in Section 4. As such, the designer can have the quantitative assurance on the behaviours of the agents which they programmed.

6 Related Work

Verifying BDI agents through model checking and theorem proving has been well explored (seen in survey [30]). For example, the work [31] (resp. [32]) applies the Java PathFinder model-checker (resps. Isabelle/HOL proof assistant) to verify BDI programs in a non-deterministic fashion. Recent work also started considering probabilistic verification of BDI agents. The work [33] uses a two-stage verification methods that first *generates* a model through program model checking (of a system implementation), and then converts this model to PRISM input format for analysis. However, unlike our focus on probabilistic extensions of the BDI semantics itself, the BDI agent used in [33] does not contain any probabilistic aspects. Instead, the *environment* where the agent executes enables the

probabilistic reasoning. Similarly, the work of [34] facilitates probabilistic verification of BDI agents by encoding them in PRISM. In this case, instead of generating the model based on an implementation, they implement a significantly simplified version of AgentSpeak directly in PRISM. The simplifications deviate from realistic BDI agents, *e.g.* enabling truly-concurrent intentions (and no intention selection) and treating plan selection as non-deterministic. Our approach faithfully modelled the *full* CAN semantics with various selection strategies development support while still providing PRISM verification capabilities. One of the closest work to us is perhaps the work [35] which introduces probabilistic state transitions in BDI agents. Same as us, they are motivated to capture the situation such as "if an agent at state s_1 executes an action, then it transfers to state s_2 with probability 0.7, or transfers to state s_3 with probability 0.3", which is difficult to reason in standard BDI agents. Unlike our work on the level of a BDI programming language, however, their work is to propose a modal logic system and proofs of properties are obtained from the resulting deduction system. Notably, a main disadvantage of their work is that the description with the probability is restricted to the transition between current time and the next time due to its next-time-like temporal operator to construct a proof system base on the tableau method.

Besides BDI agents, quantitative verification techniques have also applied to other types of agent systems. For example, the work of [36] considers uncertain communication channels between systems of interacting agents. For verification the multi-agent system is transformed to finite state Markov chains for establishing quantitative temporal properties of the system. Similar to our evaluation of plan/event/intention selection strategies, the work of [37] provides a quantitative assessment for a decentralised control policies in multi-vehicle scenarios. Specifically they study conflict resolution policies to ensure that a policy never causes collisions under some mild assumptions on the initial conditions. For general agent-based verification (which is beyond the scope of this work), we refer to [38] for the interested readers.

Works studying plan and intention selection strategies have also been well investigated separately within the BDI community. In fact, most BDI platforms provide some forms of hooks that allow the agent developers to control which plan is adopted. For example, the plan selection function in [7] is a user-defined function to customise plan/intention selection for a particular application domain. Meanwhile, various plan selection strategies such as precedence-based selection (*e.g.* preference) is also studied in [39] to select more preferred plans (according to some domain-specific plan characteristics).

Unlike plan selection to choose the "best" means to achieve an event, the intention selection which decides about which intention is the best to execute next often comes as how to manage interleaving. Therefore, it is possible that the interleaving of steps in different intentions may result in undesired outcomes such as overlooked product left to be spoiled in our smart manufacturing scenario. To manage intention interleaving, researchers tend to employ external tools to help the agent to pursue multiple intentions in parallel. For example, the work of [24] compiles agent programs to TÆMS (Task Analysis, Environment Modelling, and Simulation) framework to represent the coordination aspects of problems such as "enables" and "hinders" relations between tasks. A Design-To-Criteria scheduler is then used for intention selection to determine the full set of decisions that the agent needs to perform. The work [40] applies the Single-Player Monte Carlo Tree Search [41] to selects which intention to progress at the current step. The work [28] showed that many of the intention selection issues can be modelled in planning domain definition language (PDDL) [42] (the de-facto standard planning language) and resolved through suitable planners such as a modern highly efficient (online) planner [43]. In fact, an increasingly popular topic in the BDI community is intention progression [44], *e.g.* the Intention Progression Contest[3].

However, the goal of these plan and intention selection studies above in BDI community is to help the agent to make better decisions, by modifying or replacing entirely the original BDI reasoning, either through some extra booking of domain-specific information or through other advanced decision-making techniques. On

[3] https://sites.google.com/site/intentionprogression/home

the contrary, the focus of our work (arguably complementary to them at large) is to provide an automated quantitative analysis of BDI agents under different common selection strategies (though excluding the strategies provided by the external tool-based approach).

Existing work provides "what-if" analysis capability for BDI agents through simulation. For example, [45, 46] proposes an evacuation model using BDI agents and other network-oriented modelling approaches (*e.g.* [47]). This model simulates crowd behaviour to evaluate the effects of changing psychological and socio-cultural factor parameters. While useful, simulations or experiments only examine a subset of all the possible behaviours of the given system. If the resulting system is to be used in safety-critical areas, the above approaches *guarantee* little about actual system behaviour. Instead, we reason about systems through formal reasoning and verification, analysing *all* the possible behaviours of the system against pre-defined requirements.

There are many existing work on providing probabilistic capacity to BDI agents for various reasons. For example, the work [27] addressed the uncertainty in the belief base of a BDI agent *e.g.* due to sensor noise (80% the agent believes a true and 20% the agent believes $\neg a$ true). To achieve so, they modelled the beliefs of an agent as a set of epistemic states and each state can use a distinct underlying uncertainty theory (*e.g.* probability and possibilities probabilities) with its own belief revision strategy. Similarly, it is possible to use Bayesian Networks to represent probabilistic knowledge in BDI agents [48]. Contrary to our approach in which probability comes to model the transition of agent behaviours for a quantitative behaviour analysis, their focus is to provide a quantitative approach of representing the knowledge (*e.g.* probabilistic beliefs) of the agent for, at best, standard agent qualitative testing. In particular, we note that a plan selection strategy has been proposed a BDI agent under probabilistic beliefs in [49]. However, such a plan selection strategy is abstract *i.e.* no actual implementation and, importantly, its feasibility and computational cost (*e.g.* tractability) remains unclear, in particular in context of practical formal verification. The work [50] presents an implementation of the appraisal process of emotions using an add-on probabilistic reasoning (specifically Bayesian networks) in BDI agents. According to appraisal theory [51], the appraisal depends of one's goals and values, which can be represented as BDI agents' events and beliefs, and is calculated by Bayesian networks to estimate *e.g.* the undesirability (a value) of being in a smashed state (representing the emotion of fear) for a robot.

7 Future Work

Once we accept probabilistic reasoning *inside* an agent, it quickly becomes apparent we could consider an *external* uncertain environment. Currently, only some aspects of an uncertain environment are addressed, *i.e.* interactions between agent and environment can be probabilistic. For example, the agent tries to open the door but may fail to open it. However, the environment may change itself due to *e.g.* natural phenomena for example, 30% chance it will rain tomorrow. We may need to assess whether the agent behaves as required in all possible environmental changes. The difficulty is to obtain a realistic environment abstraction that can be integrated with existing BDI semantics whilst avoiding state explosion due to branching in both environment changes and agent reasoning. We have previously considered self-dynamic environments (without probabilistic distribution) for BDI agents [52]. One way forward is to extend this with probabilities and integrate it with our new probabilistic semantics.

As discussed, BDI agents have several key decisions to make when operating: which event to handle first (event selection), and which intention to progress next (intention selection). Given the number of decisions faced by an agent, we may want to synthesise a strategy to determine ahead-of-time the decisions an agent should make *e.g.* to avoid the worst-case execution. Though we cannot replicate some advanced decision-making techniques *e.g.* from the planning community, formal verification does offer some strategy synthesis capabilities. For example, model checkers can give a trace of evolution that makes some reachability-related properties hold (*e.g.* some goals are achieved). To allow this, instead of using pre-defined selection strategies (fixed, round-robin, probabilistic choice), we can keep the non-determinism explicit and ask a model checker for a good strategy. This is our current ongoing work.

In principle, our current framework can support reasoning about multi-agents naively: each agent as a thread. Though feasible, we expect the state space will very quickly increase and be practically infeasible. A way forward is to enforce a schedule in which an agent can progress. Before we consider multi-agent settings, a fundamental question is "what are the interesting properties of agent behaviours in a multi-agent setting that we can obtain from analysing only a single agent?" If these properties are related to competition between these agents, then a game-theoretical approach (*e.g.* in [53]) might be more suitable than a full multi-agent setting.

BDI agents draw heavily from logic programming, *e.g.* Prolog [54], and feature similar syntax and semantics. Given the probabilistic extensions to logic languages, *e.g.* Prolog [55], a comparative study would allow research ideas to flow between the probabilistic BDI agent and probabilistic logic programming domains. We leave this investigation as future work.

8 Conclusions

A quantitative evaluation and comparison framework can aid design-time specification of agents by allowing us to reason about agents that exhibit probabilistic behaviours from uncertain or failed actuators, and probabilistic decision policies.

We have extended the Can language—that formalises the behaviour of a classical BDI agents including advanced features such as failure recovery and declarative goals—to a probabilistic setting, allowing both probabilistic action outcomes and probabilistic selections, *e.g.* of plans. The extended semantics is executable through an encoding to probabilistic bigraphs, which enables quantitative analysis using BigraphER and the probabilistic model checker PRISM. Importantly, this approach allows examination of the potential consequences of different selection strategies.

Through a smart manufacturing example we have shown that it is possible to reason about different combinations of selection strategies, and that probabilistic selection strategies can reduce the impact of undesirable outcomes, compared with ordered or fixed strategies. In this example, we found that plan selection has limited effect compared to intention selection, which is a valuable insight. In particular, due to the agent making smarter intention selection choices, the impact of action outcomes can be marginal—even when the failure probabilities are large.

Acknowledgements

This work is supported by the Engineering and Physical Sciences Research Council, under PETRAS SRF grants MAGIC and FARM (EP/S035362/1) and S4: Science of Sensor Systems Software (EP/N007565/1).

References

[1] Rao, A.S.: AgentSpeak (L): BDI agents speak out in a logical computable language. In: European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pp. 42–55 (1996). Springer

[2] Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and procedural goals in intelligent agent systems. In: the 8th International Conference on Principles of Knowledge Representation and Reasoning (2002). Morgan Kaufman

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

[4] Hindriks, K.V., Boer, F.S.D., Hoek, W.V.d., Meyer, J.-J.C.: Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems **2**(4), 357–401 (1999)

[5] Dastani, M.: 2APL: a practical agent programming language. Autonomous agents and multi-agent systems **16**(3), 214–248 (2008)

[6] Winikoff, M.: JACK intelligent agents: an industrial strength platform. In: Multi-Agent Programming, vol. 15, pp. 175–193 (2005)

[7] Bordini, R.H., HüJomi, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason (2007)

[8] Pokahr, A., Braubach, L., Jander, K.: The

Jadex project: programming model. Multiagent Systems and Applications, 21–53 (2013)

[9] Benfield, S.S., Hendrickson, J., Galanti, D.: Making a strong business case for multiagent technology. In: the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 10–15 (2006). ACM

[10] Braubach, L., Pokahr, A., Lamersdorf, W.: Negotiation-based patient scheduling in hospitals. In: Advanced Intelligent Computational Technologies and Decision Support Systems, pp. 107–121 (2014)

[11] Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multiagent Systems **12**(2), 239–256 (2006)

[12] Dennis, L.A., Fisher, M., Lincoln, N.K., Lisitsa, A., Veres, S.M.: Practical verification of decision-making in agent-based autonomous systems. Automated Software Engineering **23**(3), 305–359 (2016)

[13] Chen, H.: Applications of cyber-physical system: a literature review. Journal of Industrial Integration and Management **2**(03), 1750012 (2017)

[14] Padgham, L., Singh, D.: Situational preferences for BDI plans. In: the 2013 International Conference on Autonomous Agents and Multi-agent Systems, pp. 1013–1020 (2013)

[15] Sevegnani, M., Calder, M.: BigraphER: rewriting and analysis engine for bigraphs. In: Proceedings of International Conference on Computer Aided Verification, pp. 494–501 (2016). Springer

[16] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: the 23rd International Conference on Computer Aided Verification, vol. 6806, pp. 585–591 (2011)

[17] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal aspects of computing **6**(5), 512–535 (1994)

[18] Archibald, B., Calder, M., Sevegnani, M.: Probablistic bigraphs. Formal Aspects of Computing **34** (2022)

[19] Archibald, B., Calder, M., Sevegnani, M., Xu, M.: Modelling and verifying BDI agents with bigraphs. Science of Computer Programming **215**, 102760 (2022)

[20] Archibald, B., Calder, M., Sevegnani, M., Xu, M.: Probabilistic BDI Agents: Actions, Plans, and Intentions. In: Proceedings of Software Engineering and Formal Methods, pp. 262–281 (2021)

[21] Di Pierro, A., Wiklicky, H.: An operational semantics for probabilistic concurrent constraint programming. In: the 1998 International Conference on Computer Languages, pp. 174–183 (1998). IEEE

[22] Prosser, P., Unsworth, C.: Limited discrepancy search revisited. Journal of Experimental Algorithmics (JEA) **16**, 1–6 (2011)

[23] Younes, H.L., Littman, M.L.: PPDDL1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-162 **2**, 99 (2004)

[24] Bordini, R.H., Bazzan, A.L.C., Jannone, R.D.O., Basso, D.M., Vicari, R.M., Lesser, V.R.: AgentSpeak (XL) efficient intention selection in BDI agents via decision-theoretic task scheduling. In: the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3, pp. 1294–1302 (2002)

[25] Milner, R.: The space and motion of communicating agents,. Cambridge University Press (2009)

[26] Archibald, B., Muffy, C., Sevegnani, M.: Conditional bigraphs. In: International Conference on Graph Transformation, pp. 3–19 (2020). Springer

[27] Bauters, K., McAreavey, K., Liu, W., Hong, J., Godo, L., Sierra, C.: Managing different

sources of uncertainty in a BDI framework in a principled way with tractable fragments. Journal of Artificial Intelligence Research **58**, 731–775 (2017)

[28] Xu, M., McAreavey, K., Bauters, K., Liu, W.: Intention interleaving via classical replanning. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), pp. 85–92 (2019). IEEE

[29] McDermott, D.: the AIPS-98 planning competition committee. PDDL – The Planning Domain Definition Language (1998)

[30] Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal specification and verification of autonomous robotic systems: A survey. ACM Computing Surveys (CSUR) **52**(5), 1–41 (2019)

[31] Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: Model checking agent programming languages. Automated software engineering **19**(1), 5–63 (2012)

[32] Jensen, A.B.: Machine-checked verification of cognitive agents. In: Proceedings of the 14th International Conference on Agents and Artificial Intelligence, pp. 245–256 (2022)

[33] Dennis, L.A., Fisher, M., Webster, M.: Two-stage agent program verification. Journal of Logic and Computation **28**(3), 499–523 (2018)

[34] Izzo, P., Qu, H., Veres, S.M.: A stochastically verifiable autonomous control architecture with reasoning. In: IEEE Conference on Decision and Control, pp. 4985–4991 (2016)

[35] NIDE, N., Takata, S., Fujita, M.: Bdi logic with probabilistic transition and fixed-point operator. Proc. of CLIMA'09, 71–86 (2009)

[36] Dekhtyar, M.I., Dikovsky, A.J., Valiev, M.K.: Temporal verification of probabilistic multiagent systems, 256–265 (2008)

[37] Pallottino, L., Scordio, V.G., Frazzoli, E., Bicchi, A.: Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems. In: IEEE International Conference on Robotics and Automation, pp. 2448–2453 (2006)

[38] Bakar, N.A., Selamat, A.: Agent systems verification: systematic literature review and mapping. Applied Intelligence **48**(5), 1251–1274 (2018)

[39] Visser, S., Thangarajah, J., Harland, J.: Reasoning about preferences in intelligent agent systems. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)

[40] Yao, Y., Logan, B.: Action-level intention selection for BDI agents. In: Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, pp. 1227–1236 (2016)

[41] Schadd, M.P., Winands, M.H., Tak, M.J., Uiterwijk, J.W.: Single-player monte-carlo tree search for samegame. Knowledge-Based Systems **34**, 3–11 (2012)

[42] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL-the planning domain definition language. Technical report (1998)

[43] Keller, T., Eyerich, P.: Prost: Probabilistic planning based on UCT. In: Twenty-Second International Conference on Automated Planning and Scheduling (2012)

[44] Logan, B., Thangarajah, J., Yorke-Smith, N.: Progressing intention progression: A call for a goal-plan tree contest. In: AAMAS, pp. 768–772 (2017)

[45] Van der Wal, C.N., Formolo, D., Robinson, M.A., Minkov, M., Bosse, T.: Simulating crowd evacuation with socio-cultural, cognitive, and emotional elements. In: Transactions on Computational Collective Intelligence XXVII, pp. 139–177 (2017). Springer

[46] Van der Wal, C.N., Formolo, D., Robinson, M.A., Gwynne, S.: Examining evacuee response to emergency communications with agent-based simulations. Sustainability

13(9), 4623 (2021)

[47] Treur, J.: Network-oriented Modeling, 1st edn. Springer, Switzerland (2016)

[48] Kieling, G.L., Vicari, R.M.: Insertion of probabilistic knowledge into BDI agents construction modeled in bayesian networks. In: 2011 International Conference on Complex, Intelligent, and Software Intensive Systems, pp. 115–122 (2011). IEEE

[49] Ma, J., Liu, W., Hong, J., Godo, L., Sierra, C.: Plan selection for probabilistic BDI agents. In: 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, pp. 83–90 (2014). IEEE

[50] Gluz, J.C., Jaques, P.A.: A probabilistic implementation of emotional BDI agents. In: ICAART (1), pp. 121–129 (2014)

[51] Moors, A., Ellsworth, P.C., Scherer, K.R., Frijda, N.H.: Appraisal theories of emotion: State of the art and future development. Emotion Review **5**(2), 119–124 (2013)

[52] Archibald, B., Calder, M., Sevegnani, M., Xu, M.: Verifying BDI agents in dynamic environments. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, pp. 136–141 (2022)

[53] Abate, A., Gutierrez, J., Hammond, L., Harrenstein, P., Kwiatkowska, M., Najib, M., Perelli, G., Steeples, T., Wooldridge, M.: Rational verification: game-theoretic verification of multi-agent systems. Applied Intelligence **51**(9), 6569–6584 (2021)

[54] Nugues, P.M.: An Introduction to Prolog. Springer, USA (2006)

[55] Dries, A., Kimmig, A., Meert, W., Renkens, J., Van den Broeck, G., Vlasselaer, J., De Raedt, L.: Problog2: Probabilistic logic programming. In: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part III 15, pp. 312–315 (2015). Springer

Appendix

We provide the full set of probabilistic rules for both agent (Fig. 15a) and intention-level semantics (Fig. 15b) for Can in Fig. 15.

$$A^p_{event}\ \frac{\begin{array}{c}\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \quad \eta(A_{event})=p_1\\ \mathcal{S}^p_e(\mathcal{B},E^e)=\mu_1 \quad \mu_1(\bot)=0\\ e\in E^e \quad \mu_1(e)=p_1'\end{array}}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_{p_1\cdot p_1'} \langle E^e\setminus\{e\},\mathcal{B},\Gamma\cup\{e\}\rangle} \qquad A^p_{step}\ \frac{\begin{array}{c}\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \quad \eta(A_{step})=p_2 \quad \mathcal{S}^p_i(\mathcal{B},\Gamma)=\mu_2\\ \mu_2(\bot)=0 \quad P\in\Gamma \quad \mu_2(P)=p_2' \quad \langle\mathcal{B},P\rangle \rightarrow_{p_2''} \langle\mathcal{B}',P'\rangle\end{array}}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_{p_2\cdot p_2'\cdot p_2''} \langle E^e,\mathcal{B}',(\Gamma\setminus\{P\})\cup\{P'\}\rangle}$$

$$A^p_{update}\ \frac{\begin{array}{c}\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \quad \eta(A_{update})=p_3\\ \mathcal{S}^p_i(\mathcal{B},\Gamma)=\mu_3 \quad \mu_3(\bot)=0 \quad P\in\Gamma\\ \mu(P)=p_3' \quad \langle\mathcal{B},P\rangle \nrightarrow_1\end{array}}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_{p_3\cdot p_3'} \langle E^e,\mathcal{B},\Gamma\setminus\{P\}\rangle} \qquad A^p_{idle}\ \frac{\mathcal{S}^p_{ao}(E^e,\mathcal{B},\Gamma)=\eta \quad \eta(\bot)=1}{\langle E^e,\mathcal{B},\Gamma\rangle \Rightarrow_1 \langle E^e,\mathcal{B},\Gamma\rangle}$$

(a) Probabilistic agent-level CAN semantics.

$$act^p\ \frac{\mathcal{S}^p_a(act)=\mu \quad \mu(\phi^-,\phi^+)=p \quad \mathcal{B}\vDash\varphi}{\langle\mathcal{B},act\rangle \rightarrow_p \langle(\mathcal{B}\setminus\phi^-\cup\phi^+),nil\rangle} \qquad event^p\ \frac{\Delta=\{\varphi:P \mid (e'=\varphi\leftarrow P)\in\Pi\wedge e'=e\}}{\langle\mathcal{B},e\rangle \rightarrow_1 \langle\mathcal{B},e:(\mid\Delta\mid)\rangle}$$

$$select^p\ \frac{\begin{array}{c}\mathcal{S}^p_p(\mathcal{B},\Delta)=\mu \quad \mu(\bot)=0\\ \varphi:P\in\Delta \quad \mu(\varphi:P)=p\end{array}}{\langle\mathcal{B},e:(\mid\Delta\mid)\rangle \rightarrow_p \langle\mathcal{B},P\rhd e:(\mid\Delta\setminus\{\varphi:P\}\mid)\rangle} \qquad \rhd^p_{;}\ \frac{\langle\mathcal{B},P_1\rangle \rightarrow_p \langle\mathcal{B}',P_1'\rangle}{\langle\mathcal{B},P_1\rhd P_2\rangle \rightarrow_p \langle\mathcal{B}',P_1'\rhd P_2\rangle}$$

$$\rhd^p_{\top}\ \frac{}{\langle\mathcal{B},nil\rhd P_2\rangle \rightarrow_1 \langle\mathcal{B}',nil\rangle} \qquad \rhd^p_{\bot}\ \frac{P_1\neq nil \quad \langle\mathcal{B},P_1\rangle \nrightarrow_1 \quad \langle\mathcal{B},P_2\rangle \rightarrow_p \langle\mathcal{B}',P_2'\rangle}{\langle\mathcal{B},P_1\rhd P_2\rangle \rightarrow_p \langle\mathcal{B}',P_2'\rangle}$$

$$;^p_{\top}\ \frac{\langle\mathcal{B},P_1\rangle \rightarrow_p \langle\mathcal{B}',P_1'\rangle}{\langle\mathcal{B},nil;\ P\rangle \rightarrow_p \langle\mathcal{B}',P'\rangle} \qquad ;^p\ \frac{\langle\mathcal{B},P_1\rangle \rightarrow_p \langle\mathcal{B}',P_1'\rangle}{\langle\mathcal{B},P_1;\ P_2\rangle \rightarrow_p \langle\mathcal{B}',P_1';\ P_2\rangle}$$

$$\|^p_1\ \frac{\begin{array}{c}\mathcal{S}^p_c(\mathcal{B},P_1\|P_2)=\mu \quad \mu(\bot)=0\\ \mu(P_1)=p_1 \quad \langle\mathcal{B},P_1\rangle \rightarrow_{p_1'} \langle\mathcal{B}',P_1'\rangle\end{array}}{\langle\mathcal{B},P_1\|P_2\rangle \rightarrow_{p_2\cdot p_2'} \langle\mathcal{B},P_1'\|P_2\rangle} \qquad \|^p_2\ \frac{\begin{array}{c}\mathcal{S}^p_c(\mathcal{B},P_1\|P_2)=\mu \quad \mu(\bot)=0\\ \mu(P_2)=p_2 \quad \langle\mathcal{B},P_2\rangle \rightarrow_{p_2'} \langle\mathcal{B}',P_2'\rangle\end{array}}{\langle\mathcal{B},P_1\|P_2\rangle \rightarrow_{p_2\cdot p_2'} \langle\mathcal{B},P_1\|P_2'\rangle} \qquad \|^p_{\top}\ \frac{}{\langle\mathcal{B},nil\|nil\rangle \rightarrow_1 \langle\mathcal{B},nil\rangle}$$

$$G^p_s\ \frac{\mathcal{B}\models\varphi_s}{\langle\mathcal{B},goal(\varphi_s,P,\varphi_f)\rangle \rightarrow_1 \langle\mathcal{B},nil\rangle} \qquad G^p_f\ \frac{\mathcal{B}\models\varphi_f}{\langle\mathcal{B},goal(\varphi_s,P,\varphi_f)\rangle \rightarrow_1 \langle\mathcal{B},?false\rangle}$$

$$G^p_{init}\ \frac{P\neq P_1\rhd P_2 \quad \mathcal{B}\not\models\varphi_s \quad \mathcal{B}\not\models\varphi_f}{\langle\mathcal{B},goal(\varphi_s,P,\varphi_f)\rangle \rightarrow_1 \langle\mathcal{B},goal(\varphi_s,P\rhd P,\varphi_f)\rangle} \qquad G^p_{;}\ \frac{\mathcal{B}\not\models\varphi_s \quad \mathcal{B}\not\models\varphi_f \quad \langle\mathcal{B},P_1\rangle \rightarrow_p \langle\mathcal{B}',P_1'\rangle}{\langle\mathcal{B},goal(\varphi_s,P_1\rhd P_2,\varphi_f)\rangle \rightarrow_p \langle\mathcal{B}',goal(\varphi_s,P_1'\rhd P_2,\varphi_f)\rangle}$$

$$G^p_{\rhd}\ \frac{\mathcal{B}\not\models\varphi_s \quad \mathcal{B}\not\models\varphi_f \quad \langle\mathcal{B},P_1\rangle \nrightarrow_1}{\langle\mathcal{B},goal(\varphi_s,P_1\rhd P_2,\varphi_f)\rangle \rightarrow_p \langle\mathcal{B},goal(\varphi_s,P_2\rhd P_2,\varphi_f)\rangle}$$

(b) Probabilistic intention-level CAN semantics.

Figure 15: Probabilistic extension of CAN semantics from [3].