Probabilistic Agent Programs

JÜRGEN DIX
The University of Manchester
MIRCO NANNI
University of Pisa
and
V.S. SUBRAHMANIAN
University of Maryland

Agents are small programs that autonomously take actions based on changes in their environment or "state." Over the last few years, there has been an increasing number of efforts to build agents that can interact and/or collaborate with other agents. In one of these efforts Eiter et al. [1999] have shown how agents may be built on top of legacy code. However, their framework assumes that agent states are completely determined, and there is no uncertainty in an agent's state. Thus, their framework allows an agent developer to specify how his agents will react when the agent is 100% sure about what is true/false in the world state. In this paper, we propose the concept of a *probabilistic agent program* and show how, given an arbitrary program written in any imperative language, we may build a declarative "probabilistic" agent program on top of it which supports decision making in the presence of uncertainty. We provide two alternative semantics for probabilistic agent programs. We show that the second semantics, though more epistemically appealing, is more complex to compute. We provide sound and complete algorithms to compute the semantics of *positive* agent programs.

Categories and Subject Descriptors: I.2.12 [**Artificial Intelligence**]: Distributed AI—*Intelligent Agents*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Uncertainty*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation—*Predicate and Modal Logic*; D.2.12 [**Software Engineering**]: Interoperability; H.2.4 [**Database Management**]: Heterogenous Databases

General Terms: Theory

Additional Key Words and Phrases: Probabilistic Reasoning, Uncertainty, Logic Programming, Multi-Agent Reasoning

This work was carried out when the first author was visiting the University of Maryland from January-October 1999.

The third author gratefully acknowledges support by the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, DAAG-55-97-10047, and DAAH04-96-1-0398, by the Army Research Laboratory under contract number DAAL01-97-K0135, and by an NSF Young Investigator award IRI-93-57756.

An earlier version of this paper appeared as Technical Report CS-TR-4054 in September 1999 at the University of Maryland, College Park.

Authors' addresses: Jürgen Dix, The University of Manchester, Dept. of Computer Science, Oxford Road, Manchester, M13 9PL, UK, Mirco Nanni, University of Pisa, Dept. of CS, I-56125 Pisa, Italy, V.S. Subrahmanian, University of Maryland, Dept. of CS, College Park, MD 20752, USA.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2000 ACM 1529-3785/00/1000-0207 $5.00

1. INTRODUCTION

Over the last few years, there has been increasing interest in the area of software agents. Such agents provide a wide variety of services including identification of interesting newspaper articles, software robots that perform tasks (and plan) on a user's behalf, content–based routers, agen–based telecommunication applications, and solutions to logistics problems. *IMPACT*[1] is a multinational project whose aim is to define a formal theory of software agents, implement (appropriate fragments of) the theory efficiently, and develop an appropriate suite of applications on top of this implementation. An *IMPACT* agent manages a set of data types/structures (including a message box) through a set of application program interface (API) function calls. The state of the agent at a given point in time is a set of objects belonging to these data types. Each agent has a set of integrity constraints that its state must always satisfy. When an agent's state changes (due to external events such as receipt of a message), the agent tries to modify its state so that the integrity constraints are satisfied. To do this, it has a suite of actions, and an *agent program* that specifies the operating principles (what is permitted, what is forbidden, what is obligatory, etc., and under what conditions?). Eiter et al. [1999; Eiter and Subrahmanian [1999] provide a detailed study of the semantics and complexity of such agents. Eiter et al. [2000] contains compile-time and run-time algorithms, while Arisha et al. [1999] focuses on system architecture.

Past work on *IMPACT* assumes that all agents reason with a complete and certain view of the world. However, in many real-world applications, agents have only a partial, uncertain view of what is true in the world. Though an agent may need to reason about uncertainty for many reasons, in this paper we will assume that the main cause of uncertainty in an agent is due to its state being uncertain. For example, when an image processing agent is asked to identify an enemy vehicle, it might return the fact that vehicle v_1 is a T-72 tank (with 60–70% probability) and a T-80 tank (with 20–45% probability). However, this raises several problems, the first of which is that as an action can only be executed if its precondition is true in the current state, if the agent does not know what the state is, then it cannot determine which of its actions are executable, and which are not. Second, even if an action is known to be executable, the state that results may not be precisely determinable. One consequence of all this is that the semantics of agent programs change significantly when such uncertainties arise.

The main contributions (and organization) of this paper may now be summed up as follows.

(1) In Section 2, we present a brief overview of agents (without any uncertainty involved) as described in Eiter et al. [1999].

(2) Then, in Section 3, we define the concept of a probabilistic code call, which is the basic syntactic construct through which uncertainty in abstract data types manifests itself.

(3) In Section 4, we define the syntax of probabilistic agent programs. Specifically, we show that probabilistic agent programs allow an agent developer to specify the permissions, obligations, forbidden actions, etc. associated with an agent,

[1] See `http://www.cs.umd.edu/projects/impact/`.

as depending not only on the probabilities that certain conditions hold in the agent's state, but also on the developer's assumptions about the relationship between these conditions (e.g., the probability that a conjunction holds in a given state depends not only on the probabilities of the conjuncts involved, but also on the dependencies if any between the conjuncts).

(4) In Section 5, we develop three formal semantics for probabilistic agent programs which extend each other as well as the semantics for (ordinary, nonprobabilistic) agent programs defined by Eiter et al. [1999]. We also provide results relating these diverse semantics.

(5) Then, in Section 6, we develop a sound and complete algorithm to compute the semantics defined when only positive agent programs are considered. We also show that the classical agent programs of Eiter et al. [1999] are a special case of our probabilistic programs.

(6) In Section 7, we provide an alternative, Kripke–style semantics for agent programs. In contrast to the previous "family" of semantics which assume that an agent's precondition must be true with 100% probability for the agent to execute it, this semantics also allows an agent to execute it when it is not sure (with 100% probability) that the action's precondition is true. We extend all three semantics of probabilistics agent programs defined earlier in Section 5 to handle these intuitions. Unfortunately, as we show in this section, this desire for a "more sophisticated" semantics comes at a high computational price. Fortunately, under some restrictions, the semantics is polynomial. The application developer may judge which semantics best meets his application needs.

As this paper attempts to include a wide variety of probabilistic reasoning methods (which themselves are very complex) into an agent decision–making environment based on deontic logic of actions (based on accessing heterogeneous arbitrary data structures), it is natural that the notation in this paper is involved. We have included, in Appendix A, a "Notation Table" which serves as a quick notational reference for the reader.

2. PRELIMINARIES

In *IMPACT*, each agent $\mathfrak{a}$ is built on top of a body of software code (built in any programming language) that supports a well–defined application programmer interface (either part of the code itself, or developed to augment the code). Hence, associated with each agent $\mathfrak{a}$ is a body of software code defined as follows.

Definition 2.1 *(Software Code).* We may characterize the code on top of which an agent is built as a triple $(\mathcal{T}_\mathfrak{a}, \mathcal{F}_\mathfrak{a}, \mathcal{C}_\mathfrak{a})$ where

(1) $\mathcal{T}_\mathfrak{a}$ is the set of all data types managed by $\mathfrak{a}$,

(2) $\mathcal{F}_\mathfrak{a}$ is a set of predefined functions which makes access to the data objects managed by the agent available to
external processes, and

(3) $\mathcal{C}_\mathfrak{a}$ is a set of type composition operations. A type composition operator is a partial n-ary function c which takes types $\tau_1, \ldots, \tau_n$ as input, and yields a type $c(\tau_1, \ldots, \tau_n)$ as output. As c is a partial function, c may only be defined for

certain arguments $\tau_1, \ldots, \tau_n$, i.e., c is not necessarily applicable on arbitrary types.

When $\mathfrak{a}$ is clear from context, we will often drop the subscript $\mathfrak{a}$. Intuitively, $\mathcal{T}_\mathfrak{a}$ is the set of all data types managed by $\mathfrak{a}$. $\mathcal{F}_\mathfrak{a}$ is the set of all function calls supported by the application programmer interface (*API*) of the agent's legacy code. $\mathcal{C}_\mathfrak{a}$ is the set of ways of creating new data types from existing data types. This characterization of a piece of software code is widely used (cf. the *Object Data Management Group*'s *ODMG* standard Cattell et al. [1997] and the *CORBA* framework [Siegal 1996]). Each agent also has a message box having a well–defined set of associated code calls that can be invoked by external programs.

Example 2.2 *(Surveillance Example).* Consider a surveillance application where there are hundreds of (identical) surveillance agents, and a geographic agent. The data types associated with the surveillance and geographic agent include the standard `int,bool,real,string,file` data types, plus those shown below:

Surveillance Agent

```
image:record of
  imageid:file;
  day:date;
  time:int;
  location:string
imagedb: setof image;
```

Geographic Agent

```
map:↑ quadtree;
quadtree:record of
  place:string;
  xcoord:int;
  ycoord:int;
  pop:int
  nw,ne,sw,se:↑ quadtree
```

A third agent may well merge information from these two agents, tracking a sequence of surveillance events.

The surv agent may support a function surv : *identify*() which takes as input an image and returns as output the set of all identified vehicles in it. It may also support a function called surv : *turret*() that takes as input, a vehicle id, and returns as output, the type of gun-turret it has. Likewise, the geo agent may support a function geo : *getplnode*() which takes as input a map and the name of a place and returns the set of all nodes with that name as the place-field, a function geo : *getxynode*() which takes as input a map and the coordinates of a place and returns the set of all nodes with that coordinate as the node, a function called geo : *range*() that takes as input a map, an x, y coordinate pair, and a distance r and returns as output, the set of all nodes in the map (quadtree) that are within r units of location (x, y).

Throughout this paper, we will expand on this simple example and use it to illustrate and motivate the various definitions in the paper.

Definition 2.3 *(State of an Agent).* The state of an agent $\mathfrak{a}$ at any given point t in time, denoted $\mathcal{O}_\mathfrak{a}(t)$, consists of the set of all instantiated data objects of types contained in $\mathcal{T}_\mathfrak{a}$.

An agent's state may change because it took an action, or because it received a message. Throughout this paper we will assume that except for appending messages to an agent $\mathfrak{a}$'s mailbox, another agent $\mathfrak{b}$ cannot directly change $\mathfrak{a}$'s state. However, it might do so indirectly by shipping the other agent a message requesting a change.

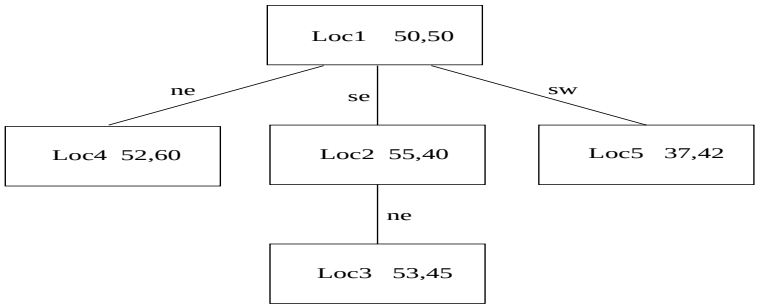

Fig. 1. Example quadtree for surveillance application.

Example 2.4. For example, the state of the geographic agent may consist of two quadtrees (one of which, map1, is shown in Figure 1), and the type `map` may contain two objects, *map*1, and *map*2, pointing to these two quadtrees, respectively. (The figure does not show population values explicitly. Assume the population values are 20,000 for loc1, 28,000 for loc2, 15,000 for loc3, 40,000 for loc4, and 8000 for loc5.)

Queries and/or conditions may be evaluated w.r.t. an agent state using the notion of a code call atom and a code call condition defined below.

Definition 2.5 *(Code Call/Code Call Atom).* If $\mathtt{a}$ is an agent, f is a function defined in $\mathtt{a}$'s underlying software code, and $(d_1, \ldots, d_n)$ is a tuple of arguments of the input type of f, then $\mathtt{a} : f(d_1, \ldots, d_n)$ is called a *code call.*

If `cc` is a code call, and `X` is either a variable symbol or an object of the output type of `cc`, then **in**(`X`, `cc`) and **not_in**(`X`, `cc`) are called *code call atoms.*

For instance, in the surveillance example, $\mathtt{geo} : \mathit{getplnode}(\mathit{map}1, \mathit{loc}1)$ returns the set containing just the single node referring to Loc1 in Figure 1. Likewise, the code call $\mathtt{geo} : \mathit{range}(\mathit{map}1, 55, 50, 11)$ returns the set containing the nodes labeled loc1 and loc2.

Definition 2.6 *(Code Call Condition).* A *code call condition* χ is defined as follows:

(1) Every code call atom is a code call condition.

(2) If $\mathtt{s}, \mathtt{t}$ are either variables or objects, then $\mathtt{s} = \mathtt{t}$ is a code call condition.

(3) If $\mathtt{s}, \mathtt{t}$ are either integers/real valued objects, or are variables over the integers/reals, then $\mathtt{s} < \mathtt{t}, \mathtt{s} > \mathtt{t}, \mathtt{s} \geq \mathtt{t}, \mathtt{s} \leq \mathtt{t}$ are code call conditions.

(4) If χ_1, χ_2 are code call conditions, then $\chi_1 \,\&\, \chi_2$ is a code call condition.

A code call condition satisfying any of the first three criteria above is an *atomic* code call condition.

An example code call condition is shown below.

Example 2.7. $\mathbf{in}(\mathtt{X}, \mathtt{geo} : \mathit{range}(\mathtt{map1}, 55, 50, 11)) \,\&\, \mathtt{X.pop} > 25,000$ is a code call condition that is satisfied by only one node in map1, viz. the loc2 node.

Each agent has an associated set of **integrity constraints**—only states that satisfy these constraints are considered to be *valid* or *legal* states. An integrity constraint is an implication whose consequent is a code call atom, and whose antecedent is a code call condition. Appendix A contains a detailed definition.

Each agent has an action-base describing various actions that the agent is capable of executing. Actions change the state of the agent and perhaps the state of other agents' `msgboxes`. An action has five components: *(i)* a name $\alpha(\mathtt{X}_1, \ldots, \mathtt{X}_n)$ where $n \geq 0$ and $\mathtt{X}_1, \ldots, \mathtt{X}_n$ are variables, *(ii)* a *precondition* which is a code call condition, *(iii)* an add list which is a set of code call atoms, *(iv)* a delete list which is a set of code call atoms, and *(v)* an action *method* (in the strict sense of object–oriented programming) which is a possibly imperative body of code that implements the action. An *instance* of an action is obtained by applying a substitution to components *(i)–(iv)* of an action which causes all these components to become *ground* (i.e., variable free). As usual, an action instance can be executed when the appropriate instance of the precondition is true in the current agent state, and the new state that results is just like the current state except that the ground atoms in the add list instance become true, while the ground atoms in the delete list instance become false. For more detailed definitions, the reader is referred to Eiter et al. [1999].

For instance, the `geo` agent may have an *insert* action that adds a node to the map. Likewise, the `surv` agent may also have an *insert* action which inserts a new image into the image database. Both these agents also have an action that sends a message.

Each agent has an associated "notion of concurrency," **conc**, which takes a set of actions and an agent state as input, and produces as output, a single action that reflects the combination of all the input actions. Eiter et al. [1999] provides examples of three different notions of concurrency. We will sometimes abuse notation and write $\mathbf{conc}(S, \mathcal{O})$ to denote the new state obtained by concurrently executing the actions in S in state $\mathcal{O}$.

Each agent has an associated set of *action constraints* that define the circumstances under which certain actions may be concurrently executed. As at any given point t in time, many sets of actions may be concurrently executable, each agent has an *Agent Program* that determines what actions the agent can take, what actions the agent cannot take, and what actions the agent must take. Agent programs are defined in terms of status atoms defined below.

Definition 2.8 *(Status Atom/Status Set).* If $\alpha(\vec{t})$ is an action, and Op belongs to $\{\mathbf{P}, \mathbf{F}, \mathbf{W}, \mathbf{Do}, \mathbf{O}\}$, then $Op\alpha(\vec{t})$ is called a *status atom.* If A is a status atom, then $A, \neg A$ are called *status literals.* A *status set* is a finite set of ground status atoms.

Intuitively, $\mathbf{P}\alpha$ means α is permitted. $\mathbf{F}\alpha$ means α is forbidden. $\mathbf{Do}\,\alpha$ means α is actually done. $\mathbf{O}\alpha$ means α is obliged, and $\mathbf{W}\alpha$ means that the obligation to perform α is waived.

Definition 2.9 *(Agent Program).* An agent program $\mathcal{P}$ is a finite set of rules of the form

$$A \leftarrow \chi \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$$

where χ is a code call condition and $L_1, \ldots, L_n$ are status literals.

Various semantics of agent programs are well described in Eiter et al. [1999] as well as in Subrahmanian et al. [2000]—due to space reasons, we recapitulate them in the appendix. They are not needed in the sequel, as all newly introduced semantics will be discussed at length. But understanding the framework of ordinary agent programs obviously helps to get a better picture of the extensions to be described in this paper.

3. PROBABILISTIC CODE CALLS

Consider a code call of the form $\mathsf{d}:f(args)$. This code call returns a set of objects. If an object o is returned by such a code call, then this means that o is *definitely* in the result of evaluating $\mathsf{d}:f(args)$. However, there are many cases, particularly in applications involving reasoning about knowledge, where a code call may need to return an "uncertain" answer. In our our surveillance example, $\mathsf{surv}:identify(image1)$ tries to identify all objects in a given image—however, it is well–known that image identification is an uncertain task. Some objects may be identified with 100% certainty, while in other cases, it may only be possible to say it is either a T-72 tank with 40–50% probability, or a T-80 tank with 50-60% probability. There is a long history of image processing algorithms that identify objects in images with different degrees of uncertainty—these include the well–known *PicHunter* System at *NEC* [Cox et al. 1996], the eigenface algorithm for recognizing faces in images at *MIT* [Moghaddam and Pentland 1998; Vasconcelos and Lippman 1998], and numerous other probabilistic approaches [Lei et al. 1995; Devroye et al. 1996; Vailaya et al. 1999]. Similar image processing algorithms for vehicle surveillance applications that return probabilistic identifications are also readily available (e.g., see Friedman and Russell [1997] and Huang and Russell [1998]).

Definition 3.1 *(Random Variable of Type τ).* A *random variable* of type τ is a finite set $\mathbf{RV}$ of objects of type τ, together with a probability distribution $\wp$ that assigns real numbers in the unit interval $[0,1]$ to members of $\mathbf{RV}$ such that $\Sigma_{o\in\mathbf{RV}}\wp(o)\leq 1$.

It is important to note that in classical probability theory [Ross 1997], random variables satisfy the stronger requirement that $\Sigma_{o\in\mathbf{RV}}\wp(o)=1$. However, in many real–life situations, a probability distribution may have missing pieces, which explains why we have chosen a weaker definition. However, the classical probability case when $\Sigma_{o\in\mathbf{RV}}\wp(o)=1$ is an instance of our more general definition.

Definition 3.2 *(Probabilistic Code Call $\mathsf{a}:_{\mathbf{RV}} f(d_1,\ldots,d_n)$).* Suppose the code call $\mathsf{a}:f(d_1,\ldots,d_n)$ has output type τ. The *probabilistic code call* associated with $\mathsf{a}:f(d_1,\ldots,d_n)$, denoted $\mathsf{a}:_{\mathbf{RV}} f(d_1,\ldots,d_n)$, returns a set of random variables of type τ when executed.

The following example illustrates the use of probabilistic code calls.

Example 3.3. Consider the code call $\mathsf{surv}:identify(image1)$. This code call may return the following two random variables.

$$\langle\{t72,t80\},\{\langle t72,0.5\rangle,\langle t80,0.4\rangle\}\rangle \text{ and } \langle\{t60,t84\},\{\langle t60,0.3\rangle,\langle t84,0.7\rangle\}\rangle$$

This says that the image processing algorithm has identified two objects in image1. The first object is either a T-72 or a T-80 tank with 50% and 40% probability,

respectively, while the second object is either a T-60 or a T-84 tank with 30% and 70% probability respectively.

Probabilistic code calls and code call conditions look exactly like ordinary code calls and code call conditions—however, as a probabilistic code call returns a set of *random variables*, probabilistic code call atoms are true or false with some probability.

Example 3.4. Consider the probabilistic code call condition

$$\mathbf{in}(\mathtt{X}, \mathrm{surv}:_{\mathbf{RV}} \mathit{identify}(\mathtt{image1})) \,\&\, \mathbf{in}(a1, \mathrm{surv}:_{\mathbf{RV}} \mathit{turret}(\mathtt{X})).$$

This code call condition attempts to find all vehicles in "image1" with a gun turret of type a1. Let us suppose that the first code call returns just one random variable specifying that image1 contains one vehicle which is either a T-72 (probability 50%) or a T-80 tank (probability 40%). When this random variable (X) is passed to the second code call, it returns one random variable with two values—a1 with probability 30% and a2 with probability 65%. What is the probability that the code call condition above is satisfied by a particular assignment to X?

The answer to this question depends very much upon the knowledge we have (if any) about the dependencies between the identification of a tank as a T-72 or a T-80, and the type of gun turret on these. For instance, if we know that all T-72's have a2 type turrets, then the probability of the conjunct being true when `X` is a T-72 tank is 0. On the other hand, it may be that the turret identification and the vehicle identification are independent for T-80s—hence, when `X` is set to T-80, the probability of the conjunct being true is $0.4 \times 0.3 = 0.12$.

Unfortunately, this is not the only problem. Other problems also arise, as shown in the following example.

Example 3.5. Suppose we consider a code call cc returning the following two random variables.

$$\begin{aligned} \mathbf{RV}_1 &= \langle \{a, b\}, \wp_1 \rangle \\ \mathbf{RV}_2 &= \langle \{b, c\}, \wp_2 \rangle \end{aligned}$$

Suppose $\wp_1(a) = 0.9, \wp_1(b) = 0.1, \wp_2(b) = 0.8, \wp_2(c) = 0.1$. What is the probability that b is in the result of the code call cc?

Answering this question is problematic. The reason is that we are told that there are at most two objects returned by cc. One of these objects is either a or b, and the other is either b or c. This leads to four possibilities, depending on which of these is true. The situation is further complicated because, in some cases, knowing that the first object is b may preclude the second object from being b—this would occur, for instance, if cc examines photographs each containing two different people and provides identifications for each. a, b, and c may be potential id's of such people returned by the image processing program. In such cases, the same person can never be pictured with himself or herself.

Of course, in other cases, there may be no reason to believe that knowing the value of one of two objects tells us anything about the value of the second object. For example if we replace people with colored cubes (with a denoting amber cubes, b black, and c cyan), there is no reason to believe that two identical black cubes cannot be pictured next to each other.

One could argue, however, that the above reasoning is incorrect because if two objects are completely identical, then they must be the same. This means that if we have two distinct black cubes, then these two black cubes must be *distinguishable* from one another via some property such as their location in the photo, or their *Id*s. This is Leibniz's well–known *extensionality principle*. Hence, we will require the results of a probabilistic code call to be *coherent* in the following sense.

Definition 3.6 *(Coherent Probabilistic Code Call).* A probabilistic code call is *coherent if, by definition,* for all distinct $\langle X_1, \wp_1\rangle, \langle X_2, \wp_2\rangle$, $X_1 \cap X_2 = \emptyset$.

Throughout this paper, only coherent probabilistic code calls are considered. Thus, the expression "probabilistic code call" assumes coherence.

Now we are ready to extend the notion of agent state, to that of *probabilistic agent state*:

Definition 3.7 *(Probabilistic State of an Agent).* The probabilistic state of an agent $\mathtt{a}$ at any given point t in time, denoted $\mathcal{O}^p(t)$, consists of the set of all instantiated data objects and random variables of types contained in $\mathcal{T}_{\mathtt{a}}$.

Definition 3.8 *(Satisfying a Code Call Atom).* Suppose $\mathtt{a}:_{\mathbf{RV}} f(d_1, \ldots, d_n)$ is a ground probabilistic code call and o is an object of the output type of this code call w.r.t. probabilistic agent state $\mathcal{O}^p$. Suppose $[\ell, u]$ is a closed, nonempty subinterval of the unit interval $[0, 1]$.

—$o \models^{[\ell,u]}_{\mathcal{O}^p} \mathbf{in}(\mathtt{X}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$ if there is a $(Y, \wp)$ in the answer returned by evaluating $\mathtt{a}:_{\mathbf{RV}} f(d_1, \ldots, d_n)$ w.r.t. $\mathcal{O}^p$ such that $o \in Y$ and $\ell \leq \wp(o) \leq u$.

—$o \models^{[\ell,u]}_{\mathcal{O}^p} \mathbf{not_in}(\mathtt{X}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$
if for all random variables $(Y, \wp)$ returned by evaluating $\mathtt{a}:_{\mathbf{RV}} f(d_1, \ldots, d_n)$ w.r.t. $\mathcal{O}^p$, either $o \notin Y$ or $\wp(o) \notin [\ell, u]$.

Probabilistic code call conditions are defined in exactly the same way as code call conditions. However, extending the above definition of "satisfaction" to probabilistic code call conditions is highly problematic because (as shown in Examples 3.4 and 3.5), the probability that a conjunction is true depends not only on the probabilities of the individual conjuncts, but also on the dependencies between the events denoted by these conjuncts. The notion of a *probabilistic conjunction strategy* defined below captures these different ways of computing probabilities via an abstract definition.

Definition 3.9 *(Probabilistic Conjunction Strategy $\otimes$).* A *probabilistic conjunction strategy* is a mapping $\otimes$ which maps a pair of probability intervals to a single probability interval satisfying the following axioms:

(1) **Bottomline:** $[L_1, U_1] \otimes [L_2, U_2] \leq [\min(L_1, L_2), \min(U_1, U_2)]$ where $[x, y] \leq [x', y']$ if $x \leq x'$ and $y \leq y'$.
(2) **Ignorance:** $[L_1, U_1] \otimes [L_2, U_2] \subseteq [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$.
(3) **Identity:** When $(e_1 \wedge e_2)$ is consistent and $[L_2, U_2] = [1, 1]$, $[L_1, U_1] \otimes [L_2, U_2] = [L_1, U_1]$.
(4) **Annihilator:** $[L_1, U_1] \otimes [0, 0] = [0, 0]$.
(5) **Commutativity:** $[L_1, U_1] \otimes [L_2, U_2] = [L_2, U_2] \otimes [L_1, U_1]$.

(6) **Associativity:** $([L_1, U_1] \otimes [L_2, U_2]) \otimes [L_3, U_3] = [L_1, U_1] \otimes ([L_2, U_2] \otimes [L_3, U_3])$.

(7) **Monotonicity:** $[L_1, U_1] \otimes [L_2, U_2] \leq [L_1, U_1] \otimes [L_3, U_3]$ if $[L_2, U_2] \leq [L_3, U_3]$.

Intuitively, in the above definition, $[L_1, U_1], [L_2, U_2]$ are intervals in which the probability of events e_1, e_2 are known to lie, and $[L_1, U_1] \otimes [L_2, U_2]$ returns a probability range for the cooccurrence of both these events. The Bottomline axiom says that the probability of the conjunct is smaller than the probabilities of the individual events. When we know nothing about the relationship between the events e_1, e_2, Boole [1854] has shown that the probability of the conjunction must lie in the interval $[\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$. This is what is stated in the Ignorance axiom. The identity and annihilator axioms specify what happens when one of the events is deterministic (i.e., not probabilistic). The axioms of commutativity and associativity are self–explanatory. The monotonicity axiom says that if we sharpen the probability range of one of the two events, then the probability range of the conjunctive event is also sharpened.

The concept of a conjunction strategy is very general, and has as special cases, the following well–known ways of combining probabilities.

(1) When we do not know the dependencies between e_1, e_2, we may use the conjunction strategy $\otimes_{ig}$ defined as $([L_1, U_1] \otimes_{ig} [L_2, U_2]) \equiv [\max(0, L_1 + L_2 - 1), \min(U_1, U_2)]$.

(2) When e_1, e_2 have maximal overlap, use the positive correlation conjunctive strategy $\otimes_{pc}$ defined as $([L_1, U_1] \otimes_{pc} [L_2, U_2]) \equiv [\min(L_1, L_2), \min(U_1, U_2)]$.

(3) When e_1, e_2 have minimal overlap, use the negative correlation conjunctive strategy $\otimes_{nc}$ defined as $([L_1, U_1] \otimes_{nc} [L_2, U_2]) \equiv [\max(0, L_1 + L_2 - 1), \max(0, U_1 + U_2 - 1)]$.

(4) When the two events occur independently, use the independence conjunction strategy $([L_1, U_1] \otimes_{in} [L_2, U_2]) = [L_1 \cdot L_2, U_1 \cdot U_2]$.

4. PROBABILISTIC AGENT PROGRAMS: SYNTAX

We are now ready to define the syntax of a probabilistic agent program (pap for short). This syntax builds upon the well–studied *annotated logic paradigm* proposed by Subrahmanian [1987], and later studied extensively [Kifer and Subrahmanian 1992; Ng and Subrahmanian 1993b; Ng and Subrahmanian 1993a].

4.1 Annotation Syntax

We assume the existence of an *annotation language* $\mathbf{L}^{ann}$—the constant symbols of $\mathbf{L}^{ann}$ are the real numbers in the unit interval $[0, 1]$. In addition, $\mathbf{L}^{ann}$ contains a finite set of function symbols, each with an associated arity, and a (possibly infinite) set of variable symbols, ranging over the unit interval $[0, 1]$. All function symbols are *preinterpreted* in the sense that associated with each function symbol f of arity k is a fixed function from $[0, 1]^k$ to $[0, 1]$.

Definition 4.1 *(Annotation Item).* We define annotation items inductively as follows:

—Every constant and every variable of $\mathbf{L}^{ann}$ is an annotation item.

—If f is an annotation function of arity n and $\mathsf{ai}_1, \ldots, \mathsf{ai}_n$ are annotation items, then the term $f(\mathsf{ai}_1, \ldots, \mathsf{ai}_n)$ is an annotation item.

An annotation item is *ground* if no annotation variables occur in it.

For instance, $0, 0.9, (\mathtt{V}+0.9)^2, ((\mathtt{V}+0.9)^2)^2$ are all annotation items if $\mathtt{V}$ is a variable in $\mathbf{L}^{ann}$ and "+", "^{2}" are annotation functions: "+" of arity 2, "^{2}" of arity 1.

Definition 4.2 *(Annotation* $[\mathsf{ai}_1, \mathsf{ai}_2]$*).* If $\mathsf{ai}_1, \mathsf{ai}_2$ are annotation items, then the term $[\mathsf{ai}_1, \mathsf{ai}_2]$ is an *annotation.* If $\mathsf{ai}_1, \mathsf{ai}_2$ are both ground, then $[\mathsf{ai}_1, \mathsf{ai}_2]$ is a *ground annotation.*

For instance, $[0, 0.4], [0.7, 0.9], [0.1, \frac{\mathtt{V}}{2}], [\frac{\mathtt{V}}{4}, \frac{\mathtt{V}}{2}]$ are all annotations. The annotation $[0.1, \frac{\mathtt{V}}{2}]$ denotes an interval only when a value in $[0, 1]$ is assigned to the variable $\mathtt{V}$.

Definition 4.3 *(Annotated Code Call Condition* $\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle$*).* If χ is a probabilistic code call condition, $\otimes$ is a conjunction strategy, and $[\mathsf{ai}_1, \mathsf{ai}_2]$ is an annotation, then $\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle$ is an *annotated code call condition.* $\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle$ is *ground* if there are no variables in either χ or in $[\mathsf{ai}_1, \mathsf{ai}_2]$.

Intuitively, the ground annotated code call condition $\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle$ says that the probability of χ being true (under conjunction strategy $\otimes$) lies in the interval $[\mathsf{ai}_1, \mathsf{ai}_2]$. For example, when $\mathtt{X}$ is ground,

$$\mathbf{in}(\mathtt{X}, \mathsf{surv} :_{\mathbf{RV}} \mathit{identify}(\mathtt{image1})) \,\&\, \mathbf{in}(\mathtt{a1}, \mathsf{surv} :_{\mathbf{RV}} \mathit{turret}(\mathtt{X})) : \langle[0.3, 0.5], \otimes_{ig}\rangle$$

is true *if and only if* the probability that $\mathtt{X}$ is identified by the surv agent and that the turret is identified as being of type a1 lies between 30 and 50% assuming that nothing is known about the dependencies between turret identifications and identifications of objects by surv.

We are now ready to define the concept of a probabilistic agent program.

Definition 4.4 *(Probabilistic Agent Programs* $\mathcal{PP}$*).* Suppose Γ is an annotated code call condition, and $A, L_1, \ldots, L_n$ are status atoms. Then

$$A \leftarrow \Gamma \,\&\, L_1 \,\&\, \ldots \,\&\, L_n \tag{1}$$

is a *probabilistic action rule.* For such a rule r, we use $B^+_{as}(r)$ to denote the positive status atoms in $\{L_1, \ldots, L_n\}$, and $B^-_{as}(r)$ to denote the set of negative status literals in $\{L_1, \ldots, L_n\}$.

A *probabilistic agent program* (pap for short) is a finite set of probabilistic action rules.

It is important to note in the above definition that in a probabilistic action rule, status atoms are *not* annotated—uncertainty is present only in the state, and on the basis of this uncertainty, the agent must determine what it is obliged to do, forbidden from doing, etc.

A simple example of a probabilistic agent program is given below.

Example 4.5 *(Probabilistic Agent Program).* Consider an intelligent sensor agent that is performing surveillance tasks. The following rules specify a small pap that such an agent might use.

$$
\begin{aligned}
\mathbf{Do}\, send_warn(\mathtt{X}) \leftarrow\ & \mathbf{in}(\mathtt{F}, \mathtt{surv} : \mathit{file}(\mathtt{imagedb})) \,\&\\
& \mathbf{in}(\mathtt{X}, \mathtt{surv} :_{\mathbf{RV}} \mathit{identify}(\mathtt{F})) \,\&\\
& \mathbf{in}(\mathtt{a1}, \mathtt{surv} :_{\mathbf{RV}} \mathit{turret}(\mathtt{X}))) : \langle [0.7, 1.0], \otimes_{ig} \rangle\\
& \neg \mathbf{F} send_warn(\mathtt{X}).\\
\mathbf{F} send_warn(\mathtt{X}) \leftarrow\ & \mathbf{in}(\mathtt{F}, \mathtt{surv} : \mathit{file}(\mathtt{imagedb})) \,\&\\
& \mathbf{in}(\mathtt{X}, \mathtt{surv} :_{\mathbf{RV}} \mathit{identify}(\mathtt{F})) \,\&\\
& \mathbf{in}(\mathtt{L}, \mathtt{geo} :_{\mathbf{RV}} \mathit{getplnode}(\mathtt{X.location})) \,\&\\
& \mathbf{in}(\mathtt{L}, \mathtt{geo} :_{\mathbf{RV}} \mathit{range}(100, 100, 20)).
\end{aligned}
$$

This agent operates according to two very simple rules. The first rule says that it sends a warning whenever it identifies an enemy vehicle as having a gun turret of type a1 with over 70% probability, as long as sending such a warning is not forbidden. The second rule says that sending a warning is forbidden if the enemy vehicle is within 20 units of distance from location (100,100).

5. PROBABILISTIC AGENT PROGRAMS: SEMANTICS

We are now ready to define the semantics of paps. The semantics of paps will be defined via the concept of a *probabilistic status set* (defined below).

Definition 5.1 *(Probabilistic Status Set $\mathcal{PS}$).* A *probabilistic status set* is any set $\mathcal{PS}$ of ground status atoms. For any operator $\mathsf{Op} \in \{\mathbf{P}, \mathbf{Do}, \mathbf{F}, \mathbf{O}, \mathbf{W}\}$, we denote by $\mathsf{Op}(\mathcal{PS})$ the set $\{\alpha \mid \mathsf{Op}(\alpha) \in \mathcal{PS}\}$. Similarly, we use $\neg\mathcal{PS}$ to denote the set $\{\neg A \mid A \in \mathcal{PS}\}$.

It will turn out that given any probabilistic agent program, and an (uncertain) agent state evaluated using probabilistic code calls, the meaning of the pap w.r.t. the state may be defined via a set of probabilistic status sets that have some desirable properties. These properties fall into three broad categories:

(1) the probabilistic status set must be "closed" under the rules in the pap;
(2) the probabilistic status set must be deontically consistent (e.g. it cannot require something to be both permitted and forbidden), and it must not violate the action constraints;
(3) the probabilistic status set must not lead to a new state that violates the integrity constraints associated with the agent.

5.1 Satisfaction of Annotated Formulae

In this section, we define what it means for an agent state to satisfy an annotated code call condition.

Definition 5.2 *(Satisfying an Annotated Code Call Condition).* Suppose $\mathcal{O}^p$ is an agent state, and $\chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$ is a ground annotated code call condition. $\mathcal{O}^p$ is said to *satisfy* $\chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$, denoted $\mathcal{O}^p \models^{[\mathsf{ai}_1, \mathsf{ai}_2]} \chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$ iff

—χ is of the form $o = o$ (where o is an object), or
—χ is of the form $r_1 < r_2$, where r_1, r_2 are real numbers (or integers) such that r_1 is less than r_2, or

—χ is of the form $\mathbf{in}(\mathtt{o}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$ and it is the case that $o \models^{[\mathsf{ai}_1, \mathsf{ai}_2]}_{\mathcal{O}^p} \mathbf{in}(\mathtt{X}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$, or

—χ is of the form $\mathbf{not_in}(\mathtt{o}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$ and it is the case that $o \models^{[\mathsf{ai}_1, \mathsf{ai}_2]}_{\mathcal{O}^p} \mathbf{not_in}(\mathtt{X}, \mathtt{a}:_{\mathbf{RV}} f(\mathtt{d}_1, \ldots, \mathtt{d}_n))$, or

—χ is of the form $\chi_1 \wedge \chi_2$ and $[\ell_1, u_1], [\ell_2, u_2]$ are the tightest intervals such that $\mathcal{O}^p \models^{[\ell_1, u_1]} \chi_1$ and $\mathcal{O}^p \models^{[\ell_2, u_2]} \chi_2$ and $[\mathsf{ai}_1, \mathsf{ai}_2] \supseteq [\ell_1, u_1] \otimes [\ell_2, u_2]$.

$\mathcal{O}^p$ is said to *satisfy* a nonground annotated code call $\chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$ *if, by definition,* $\mathcal{O}^p$ satisfies all ground instances of $\chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$.

5.2 Closure and the Operator App

We may associate with any pap $\mathcal{PP}$, an operator $\mathbf{App}_{\mathcal{PP}, \mathcal{O}^p}(\mathcal{PS})$ which maps probabilistic status sets to probabilistic status sets.

Definition 5.3 *(Operator* $\boldsymbol{App}_{\mathcal{PP}, \mathcal{O}^p}(\mathcal{PS})$*).* Suppose $\mathcal{PP}$ is a probabilistic agent program, $\mathcal{O}^p$ is a probabilistic agent state, and $\mathcal{PS}$ is a probabilistic status set. Then

$$\mathbf{App}_{\mathcal{PP}, \mathcal{O}^p}(\mathcal{PS}) = \{\mathsf{Op}\,\alpha \mid \mathsf{Op}\,\alpha \text{ is the head of a ground instance } r \text{ of a rule in } \mathcal{PP} \text{ satisfying the 4 conditions below}\}$$

(1) $B^+_{as}(r) \subseteq \mathcal{PS}$ and $\{L : \neg L \in B^-_{as}(r)\} \cap \mathcal{PS} = \emptyset$.

(2) For every annotated code call condition $\chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$ in the body of r, it is the case that $\mathcal{O}^p \models^{[\mathsf{ai}_1, \mathsf{ai}_2]} \chi : \langle [\mathsf{ai}_1, \mathsf{ai}_2], \otimes \rangle$.

(3) If $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, then $\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\alpha)$.

(4) For every action status atom of the form $\mathsf{Op}\,\beta$ in $B^+_{as}(r)$ such that $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, $\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\beta)$.

The first part of this definition says that for a rule to fire, the action status atoms in its body must be "true" w.r.t. $\mathcal{PS}$. The second condition says that annotated code call conditions in a rule body must be satisfied in the current object state for the rule to fire. The third part is more tricky. It says that if $\mathbf{O}\alpha$ or $\mathbf{Do}\,\alpha$ or $\mathbf{P}\alpha$ is in the head of a rule, then for the rule to fire, the precondition of the action must be true with 100% probability. The final condition is similar w.r.t. to positive action status atoms in the body. *Thus, for now, we assume that for an agent to perform an action (or even be permitted to perform an action), it must be 100% sure that the action's precondition is true* (later in Section 7.1, we will provide an alternate, more complex semantics that does not require this).

The above operator significantly extends the well–known fixpoint operator by van Emden and Kowalski [1976] which is associated with (positive and general) logic programs and is well described in Lloyd [1987]. However, conditions (3) and (4) of the above definition are not present in the van Emden and Kowalski [1976] operator, and likewise the choice in item (2) to access a legacy data structure is also new (though in the same spirit as van Emden and Kowalski [1976]).

Definition 5.4 *(Closure under Program Rules).* $\mathcal{PS}$ is said to be closed under the rules of the probabilistic program $\mathcal{PP}$ in the probabilistic state $\mathcal{O}^p$ *if, by definition,* $\mathbf{App}_{\mathcal{PP}, \mathcal{O}^p}(\mathcal{PS}) \subseteq \mathcal{PS}$.

5.3 Deontic/Action Consistency/Closure

The concept of deontic/action consistency requires that probabilistic status sets satisfy the agent's action constraints and commonsense axioms about deontic modalities.

Definition 5.5 *(Deontic and Action Consistency).* A probabilistic status set $\mathcal{PS}$ is *deontically consistent* with respect to a probabilistic agent state $\mathcal{O}^p$ *if, by definition,* it satisfies the following rules for any ground action α:

—If $\mathbf{O}\alpha \in \mathcal{PS}$, then $\mathbf{W}\alpha \notin \mathcal{PS}$.
—If $\mathbf{P}\alpha \in \mathcal{PS}$, then $\mathbf{F}\alpha \notin \mathcal{PS}$.
—If $\mathbf{P}\alpha \in \mathcal{PS}$, then $\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\alpha)$.

A probabilistic status set $\mathcal{PS}$ is *action consistent* w.r.t. $\mathcal{O}^p$ *if, by definition,* for every action constraint of the form

$$\{\alpha_1(\vec{\mathtt{X}}_1), \ldots, \alpha_k(\vec{\mathtt{X}}_k)\} \hookleftarrow \chi \tag{2}$$

either $\mathcal{O}^p \not\models^{[1,1]} \chi$ or $\{\alpha_1(\vec{\mathtt{X}}_1), \ldots, \alpha_k(\vec{\mathtt{X}}_k)\} \not\subseteq \mathbf{Do}\,(\mathcal{PS})$.

The following example illustrates the concept of deontic and action consistency.

Example 5.6. Suppose we have a resource allocation agent having two actions—*send_A*() and *send_B*()—each of which sends a unit of the resource respectively to agents A B. To execute either of them, we need to have at least one unit of resource, and to execute them together we need at least two units:

$$\mathit{Pre}(\mathit{send_A}()) = \mathbf{in}(\mathtt{X}, \mathtt{allocator}\colon \mathit{avail_rsc}()) \;\&\; \mathtt{X} > 0.$$
$$\mathit{Pre}(\mathit{send_B}()) = \mathbf{in}(\mathtt{X}, \mathtt{allocator}\colon \mathit{avail_rsc}()) \;\&\; \mathtt{X} > 0.$$
$$\{\mathit{send_to_A}(), \mathit{send_to_B}()\} \hookleftarrow \mathbf{in}(\mathtt{X}, \mathtt{allocator}\colon \mathit{avail_rsc}()) \;\&\; \mathtt{X} < 2$$

Suppose the agent's current state $\mathcal{O}^p$ is one in which *avail_rsc*() returns 1. Then

$$\mathcal{PS} = \{\; \mathbf{P}\mathit{send_to_A}(), \mathbf{Do}\,\mathit{send_to_A}(), \\ \mathbf{Do}\,\mathit{send_to_B}(), \mathbf{O}\mathit{send_to_B}() \;\}$$

is deontically consistent (there are no **W** and **F** atoms at all, and the action preconditions are true), but not action consistent.

The deontic and action closure of a probabilistic status set $\mathcal{PS}$ is defined in exactly the same way (see appendix, Definition **??**) as in the nonprobabilistic case.

5.4 Probabilistic State Consistency

The final requirement of a feasible probabilistic status set ensures that the new state that results after concurrently executing a set of actions is consistent with the integrity constraints. We say that "$\mathcal{O}^p$ *satisfies* the integrity constraint $\psi \;\Rightarrow\; \chi$" *if, by definition,* either $\mathcal{O}^p \not\models^{[1,1]} \psi$ or $\mathcal{O}^p \models^{[1,1]} \chi$.

Definition 5.7 *(Probabilistic State Consistency).* A probabilistic status set $\mathcal{PS}$ is *probabilistically state consistent* w.r.t. $\mathcal{O}^p$ *if, by definition,* the new probabilistic state, $\mathcal{O}^{p\prime} = \mathbf{conc}(\mathbf{Do}\,(\mathcal{PS}), \mathcal{O}^p)$ obtained after concurrently executing all actions of the form $\mathbf{Do}\,\alpha \in \mathcal{PS}$, satisfies all integrity constraints.

The following example illustrates the concept of probabilistic state consistency.

Example 5.8. Suppose we have a vehicle coordination agent that tracks vehicle on a road (line), and makes sure that two vehicles do not collide. Such an agent may have the integrity constraint

$$\mathbf{in}(\mathtt{X}, \mathtt{geo} : \mathit{getposition}(\mathtt{a})) \ \& \ \mathbf{in}(\mathtt{Y}, \mathtt{geo} : \mathit{getposition}(\mathtt{b})) \Rightarrow \mathtt{X} \neq \mathtt{Y}.$$

It may be able to perform an action $move_forward(a)$:

$$\begin{array}{ll} \mathit{Pre}(\mathit{move_forward}(a)) & = \mathbf{in}(\mathtt{X}, \mathtt{geo} : \mathit{getposition}(\mathtt{a})) \\ \mathit{Del}(\mathit{move_forward}(a)) & = \mathbf{in}(\mathtt{X}, \mathtt{geo} : \mathit{getposition}(\mathtt{a})) \\ \mathit{Add}(\mathit{move_forward}(a)) & = \mathbf{in}(\mathtt{Y}, \mathtt{geo} : \mathit{getposition}(\mathtt{a})) \,\&\, \mathtt{Y} = \mathtt{X} + 1 \end{array}$$

In a probabilistic state $\mathcal{O}^p$ where $\mathtt{geo} : \mathit{getposition}(a)$ returns 200, and the code call $\mathtt{geo} : \mathit{getposition}(b)$ returns 201, the status set

$$\mathcal{PS} = \{\mathbf{P}\mathit{move_forward}(a), \mathbf{Do}\ \mathit{move_forward}(a)\}$$

is not state consistent, as executing $\mathbf{Do}\ (\mathcal{PS})$ leads to where both agent a and agent b are in position 201, violating the above integrity constraint.

5.5 Feasible Probabilistic Status Sets

The meaning of a pap (w.r.t. a given state) may be characterized via those probabilistic status sets that satisfy the conditions of closure under program rules, deontic/action consistency and probabilistic state consistency. Such probabilistic status sets are said to be *feasible*.

Definition 5.9 *(Feasible Probabilistic Status Set).* Suppose $\mathcal{PP}$ is an agent program and $\mathcal{O}^p$ is a probabilistic agent state. A probabilistic status set $\mathcal{PS}$ is *feasible* for $\mathcal{PP}$ on $\mathcal{O}^p$ if the following conditions hold:

($\mathcal{PS}$1). $\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) \subseteq \mathcal{PS}$ *(closure under the program rules)*;

($\mathcal{PS}$2). $\mathcal{PS}$ is deontically and action consistent *(deontic/action consistency)*;

($\mathcal{PS}$3). $\mathcal{PS}$ is action closed and deontically closed *(deontic/action closure)*;

($\mathcal{PS}$4). $\mathcal{PS}$ is state consistent *(state consistency)*.

paps may have zero, one, or many feasible status sets, as seen via the following examples.

Example 5.10. Consider the following agent program.

$$\begin{array}{rcl} \mathbf{P}\mathit{send_warn}(t80) & \leftarrow & , \\ \mathbf{F}\mathit{send_warn}(t80) & \leftarrow & . \end{array}$$

In any probabilistic agent state $\mathcal{O}^p$ such that $\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\mathit{send_warn}(t80))$, the above program cannot have any feasible probabilistic status set $\mathcal{PS}$. This is because closure under program rules requires that both $\mathbf{P}\mathit{send_warn}(t80)$ and $\mathbf{F}\mathit{send_warn}(t80)$ are in $\mathcal{PS}$, but this causes $\mathcal{PS}$ to violate deontic consistency.

In contrast, consider the following one-rule program for checking the power level of surveillance equipment.

$$\mathbf{O}\mathit{power_warn}() \ \leftarrow \ \mathbf{in}(\mathtt{X}, \mathtt{surv} : \mathit{powerlevel}()) \ \&\ \mathtt{X} < 2000.$$

Suppose $\mathsf{surv}: powerlevel()$ returns 1000 in some state $\mathcal{O}^p$, and $power_warn()$ has no preconditions. If no integrity and action constraints are present, then this pap has exactly one feasible status set, viz.

$$\{\mathbf{O}power_warn(), \mathbf{Do}\, power_warn(), \mathbf{P}power_warn()\}.$$

Now let us consider a pap which says that one of the two agents a, b must be warned (if it is active). Furthermore, if b is to be warned, its regular (nonemergency) channel must not be on.

$$\begin{aligned}
\mathbf{F}open_ch(b) &\leftarrow \neg\mathbf{F}open_ch(b)\ \&\ \mathbf{Do}\, warn_ag(b).\\
\mathbf{Do}\, warn_ag(a) &\leftarrow \mathbf{in}(\mathtt{a}, \mathsf{surv}: activeagents())\ \&\ \neg\mathbf{Do}\, warn_ag(b).\\
\mathbf{O}warn_ag(b) &\leftarrow \mathbf{in}(\mathtt{b}, \mathsf{surv}: activeagents())\ \&\ \neg\mathbf{Do}\, warn_ag(a).
\end{aligned}$$

We assume the absence of integrity constraints and preconditions for all actions. However, the following action constraint is present:

$$\{warn_ag(a), warn_ag(b)\} \hookleftarrow .$$

If $\mathsf{surv}: activeagents()$ returns $\{a, b\}$ in state $\mathcal{O}^p$, then the above program has several feasible probabilistic status sets:

$$\begin{array}{l}
\{\mathbf{F}open_ch(b), \mathbf{O}warn_ag(b), \mathbf{Do}\, warn_ag(b), \mathbf{P}warn_ag(b)\}\\
\{\mathbf{F}open_ch(b), \mathbf{Do}\, warn_ag(a), \mathbf{P}warn_ag(a)\}\\
\{\mathbf{Do}\, warn_ag(a), \mathbf{P}warn_ag(a)\}
\end{array}$$

Notice that no feasible status set contains both $\mathbf{Do}\, warn_ag(a)$ and $\mathbf{Do}\, warn_ag(b)$.

5.6 Rational Probabilistic Status Sets

As seen from the above examples, feasible status sets may contain action status atoms that are not required for feasibility. Rational probabilistic status sets refine this definition.

Definition 5.11 *(Groundedness; Rational Probabilistic Status Set).* A probabilistic status set $\mathcal{PS}$ is *grounded* if there is no probabilistic status set $\mathcal{PS}' \neq \mathcal{PS}$ such that $\mathcal{PS}' \subseteq \mathcal{PS}$ and such that $\mathcal{PS}'$ satisfies conditions $(\mathcal{PS}1)$–$(\mathcal{PS}3)$ of a feasible probabilistic status set.

$\mathcal{PS}$ is *rational if, by definition,* it is feasible and grounded.

Example 5.12. Consider the last case in Example 5.10. Only two of the listed feasible status sets are rational, viz.

$$\begin{array}{l}
\{\mathbf{F}open_ch(b), \mathbf{O}warn_ag(b), \mathbf{Do}\, warn_ag(b), \mathbf{P}warn_ag(b)\} \text{ and}\\
\{\mathbf{Do}\, warn_ag(a), \mathbf{P}warn_ag(a)\}
\end{array}$$

5.7 Reasonable Probabilistic Status Sets

In rational status sets certain action status atoms may be true even though there is no rule whose head contains (or implies) that action status atom. This can be easily illustrated by the rule $\mathbf{P}\alpha \leftarrow \neg\mathbf{P}\beta$. This program has not only $\{\mathbf{P}\alpha\}$, but also $\{\mathbf{P}\beta\}$ as a rational status set. The concept of a reasonable status set

(which is derived from the well–known stable model semantics of logic programs [Gelfond and Lifschitz 1988]) prevents this.

Definition 5.13 *(Reasonable Probabilistic Status Set).* Suppose $\mathcal{PP}$ is a pap, $\mathcal{O}^p$ is a probabilistic agent state, and $\mathcal{PS}$ is a probabilistic status set.

(1) If $\mathcal{PP}$ is a positive (i.e., $B^-_{as}(r) = \emptyset$ for all $r \in \mathcal{PP}$), then $\mathcal{PS}$ is a *reasonable probabilistic status set* for $\mathcal{PP}$ on $\mathcal{O}^p$, *if, by definition,* $\mathcal{PS}$ is a rational probabilistic status set for $\mathcal{PP}$ on $\mathcal{O}^p$.

(2) Otherwise, the reduct of $\mathcal{PP}$ w.r.t. $\mathcal{PS}$ and $\mathcal{O}^p$, denoted by $red^{\mathcal{PS}}(\mathcal{PP}, \mathcal{O}^p)$, is the program which is obtained from the ground instances of the rules in $\mathcal{PP}$ over $\mathcal{O}^p$ as follows.
   (a) First, remove every rule r such that $B^-_{as}(r) \cap \mathcal{PS} \neq \emptyset$.
   (b) Remove all atoms in $B^-_{as}(r)$ from the remaining rules.
   Then $\mathcal{PS}$ is a *reasonable probabilistic status set* for $\mathcal{PP}$ w.r.t. $\mathcal{O}^p$, if it is a reasonable probabilistic status set of the program $red^{\mathcal{PS}}(\mathcal{PP}, \mathcal{O}^p)$ with respect to $\mathcal{O}^p$.

The following example illustrates the concept of a reasonable status set.

Example 5.14. Consider again the last case in Example 5.10. Only one of the listed feasible status sets is reasonable, viz.

$$\mathcal{PS} = \{\mathbf{Do}\, warn_ag(a), \mathbf{P}\, warn_ag(a)\}$$

To see why this probabilistic status set is feasible, note that the reduct of $\mathcal{PP}$ w.r.t. $\mathcal{PS}$ is

$$\mathbf{Do}\, warn_ag(a) \leftarrow \mathbf{in}(a, \mathrm{surv} : activeagents()),$$

whose (unique) rational status set is obviously $\mathcal{PS}$.

5.8 Semantical Properties

In this section, we prove some properties about the different semantics described above.

Proposition 5.15 (Properties of Feasible Status Sets). *Let $\mathcal{PS}$ be a feasible probabilistic status set. Then,*

(1) If $\mathbf{Do}\,(\alpha) \in \mathcal{PS}$, *then* $\mathcal{O}^p \models^{[1,1]} Pre(\alpha)$.
(2) If $\mathbf{P}\alpha \notin \mathcal{PS}$, *then* $\mathbf{Do}\,(\alpha) \notin \mathcal{PS}$.
(3) If $\mathbf{O}\alpha \in \mathcal{PS}$, *then* $\mathcal{O}^p \models^{[1,1]} Pre(\alpha)$.
(4) If $\mathbf{O}\alpha \in \mathcal{PS}$, *then* $\mathbf{F}\alpha \notin \mathcal{PS}$.

The following theorem says that reasonable status sets are rational.

Theorem 5.16 (Reasonable Status Sets are Rational). *Let $\mathcal{PP}$ be a probabilistic agent program and $\mathcal{O}^p$ a probabilistic agent state. Every reasonable probabilistic status set of $\mathcal{PP}$ on $\mathcal{O}^p$ is a rational probabilistic status set of $\mathcal{PP}$ on $\mathcal{O}^p$.*

Given any pap $\mathcal{PP}$ and probabilistic agent state $\mathcal{O}^p$, we may define an operator that maps probabilistic status sets to probabilistic status sets as follows.

Definition 5.17 *(Deontic and Action Closure).* Let $\mathcal{PS}$ be a probabilistic status set and α an action. Then, we use the notation $\mathbf{D\text{-}Cl}(\mathcal{PS})$ to denote the closure of $\mathcal{PS}$ under the rule $\mathbf{O}\alpha \in \mathcal{PS} \Rightarrow \mathbf{P}\alpha \in \mathcal{PS}$ and $\mathbf{A\text{-}Cl}(\mathcal{PS})$ to denote the closure of $\mathcal{PS}$ under the rules $\mathbf{O}\alpha \in \mathcal{PS} \Rightarrow \mathbf{Do}\,\alpha \in \mathcal{PS}$ and $\mathbf{Do}\,\alpha \in \mathcal{PS} \Rightarrow \mathbf{O}\alpha \in \mathcal{PS}$.

Definition 5.18 *($\boldsymbol{T}_{\mathcal{PP},\mathcal{O}^p}$ Operator).* Suppose $\mathcal{PP}$ is a probabilistic agent program and $\mathcal{O}^p$ is a probabilistic agent state. Then, for any probabilistic status set $\mathcal{PS}$,

$$\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) \cup \mathbf{D\text{-}Cl}(\mathcal{PS}) \cup \mathbf{A\text{-}Cl}(\mathcal{PS}).$$

Note that as $\mathbf{D\text{-}Cl}(\mathcal{PS}) \subseteq \mathbf{A\text{-}Cl}(\mathcal{PS})$, we may equivalently write this as

$$\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) \cup \mathbf{A\text{-}Cl}(\mathcal{PS}).$$

The following property of feasible probabilistic status sets, which is similar to the well–known result in classical logic programming linking prefixpoints of the van Emden and Kowalski [1976] fixpoint operator and Herbrand models of a logic program, is easily established. Here, status sets play a role similar to Herbrand interpretations, and the operator $\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$ is analogous to the fixpoint operator in van Emden and Kowalski [1976].

LEMMA 5.19 ($\mathcal{PS}$ AS PREFIXPOINT OF $\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$). *Let $\mathcal{PP}$ be a probabilistic agent program, $\mathcal{O}^p$ be any probabilistic agent state, and $\mathcal{PS}$ be any probabilistic status set. If $\mathcal{PS}$ satisfies conditions ($\mathcal{PS}1$) and ($\mathcal{PS}3$) of feasibility, then $\mathcal{PS}$ is pre-fixpoint of $\boldsymbol{T}_{\mathcal{PP},\mathcal{O}^p}$, i.e., $\boldsymbol{T}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) \subseteq \mathcal{PS}$.*

The following theorem, which has no obvious analog in classical logic programming, says that in the absence of integrity constraints, a pap has a rational probabilistic status set *if and only if* it has a feasible one.

THEOREM 5.20 (EXISTENCE OF RATIONAL PROBABILISTIC STATUS SETS). *Let $\mathcal{PP}$ be a probabilistic agent program. If $\mathcal{IC} = \emptyset$, then $\mathcal{PP}$ has a rational probabilistic status set if and only if $\mathcal{PP}$ has a feasible probabilistic status set.*

6. COMPUTING PROBABILISTIC STATUS SETS OF POSITIVE PAPS

In this section, we present a sound and complete algorithm to compute the unique reasonable status set of a positive pap. For this purpose, we use a variant of the $\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$ operator introduced earlier.

Definition 6.1 *($\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$ Operator).* Suppose $\mathcal{PP}$ is a probabilistic agent program and $\mathcal{O}^p$ is a probabilistic agent state. Then, for any probabilistic status set $\mathcal{PS}$,

$$\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \mathbf{A\text{-}Cl}(\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})).$$

We may compute the operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ using Algorithm 6.2 below.

Algorithm 6.2 *($\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$ Computation of Positive paps).*
Compute-$\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ ($\mathcal{O}^p$: prob. agent state, $\mathcal{PS}$: prob. status set)

(⋆ the probabilistic agent program $\mathcal{PP}$ is positive; ⋆)
(⋆ **Input:** a probabilistic agent state $\mathcal{O}^p$, and a prob. status set $\mathcal{PS}$ ⋆)
(⋆ **Output:** a deontically and action consistent set $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$ (if existent) ⋆)
(⋆ or *"no consistent set exists"* ⋆)

1. $X := \mathcal{PS}$;
2. **for each** rule $r \in \mathcal{PP}$
3. **for each** ground instance $r\theta$ of r
4. **if** $r\theta = \mathsf{Op}\,\alpha \leftarrow \Gamma \,\&\, L_1 \,\&\, \ldots \,\&\, L_n$ **and**
   $\mathcal{O}^p \models \Gamma$ and $\{L_1, \ldots, L_n\} \subseteq \mathcal{PS}$ **and**
   for every atom $\mathsf{Op}'(\beta) \in \{L_1, \ldots, L_n\} \cup \{\mathsf{Op}(\alpha)\}$
   such that $\mathsf{Op}' \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$: $\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\beta)$
5. **then** $X := X \cup \mathbf{A\text{-}Cl}(\{\mathsf{Op}\,\alpha\})$,
6. **if** X contains ($\mathbf{O}\alpha$ and $\mathbf{W}\alpha$) or ($\mathbf{P}\alpha$ and $\mathbf{F}\alpha$)
7. **then Return** *"no consistent set exists"*.
8. **Return** X.

end.

The behavior of Algorithm 6.2 is illustrated by the following example.

Example 6.3. Consider the following program, saying that whenever a (probably) enemy vehicle Y is detected, a warning message about Y is sent to a friendly source, and the agent perfroming the detection is not allowed to move.

$$\begin{aligned}
\mathbf{O}\mathit{send_warn}(\mathtt{Y}) \;&\leftarrow\; \mathbf{in}(\mathtt{F}, \mathrm{surv} : \mathit{file}(\mathtt{imagedb})) \,\&\\
&\quad\; \mathbf{in}(\mathtt{Y}, \mathrm{surv} :_{\mathbf{RV}} \mathit{identify}(\mathtt{F}))\langle[0.5, 1.0], \otimes_{ig}\rangle \,\&\\
&\quad\; \mathbf{O}\mathit{send_warn}(\mathtt{X}).\\
\mathbf{F}\mathit{move}() \;&\leftarrow\; \mathbf{Do}\,\mathit{send_warn}(\mathtt{X}).\\
\mathbf{O}\mathit{send_warn}(\mathtt{X}) \;&\leftarrow\; \mathbf{in}(\mathtt{F}, \mathrm{surv} : \mathit{file}(\mathtt{imagedb})) \,\&\\
&\quad\; \mathbf{in}(\mathtt{X}, \mathrm{surv} :_{\mathbf{RV}} \mathit{identify}(\mathtt{F})) \,\&\\
&\quad\; \mathbf{in}(\mathtt{X}, \mathrm{surv} : \mathit{enemyvehicles}())) : \langle[0.5, 1.0], \otimes_{ig}\rangle.
\end{aligned}$$

Moreover, assume that in the current (probabilistic) state $\mathcal{O}^p$, $\mathrm{surv} : \mathit{file}(\mathit{imagedb})$ returns $\mathit{image}1$, $\mathrm{surv} : \mathit{identify}(\mathit{image}1)$ returns the random variables

$$\langle\{t80\}, \{\langle t80, 0.6\rangle\}\rangle \quad \text{and} \quad \langle\{t72\}, \{\langle t72, 0.5\rangle\}\rangle,$$

and $\mathrm{surv} : \mathit{enemyvehicles}()$ returns $t80$.

Now we apply Algorithm 6.2 to compute $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}(\emptyset)$: Step 1 sets $X = \emptyset$; step 2 selects the first rule, while step 3 considers all ground instances (of the first rule) whose body's truth is checked in step 4: no instance satisfies it (because $\mathcal{PS}$ is empty), so nothing happens. The same result is obtained when step 2 considers the second rule. Eventually, step 2 considers the third rule, which satisfies the condition of step 4 with its head instantiated to $\mathbf{O}\mathit{send_warn}(t80)$. Step 5 inserts $\mathbf{O}\mathit{send_warn}(t80)$, $\mathbf{Do}\,\mathit{send_warn}(t80)$, $\mathbf{P}\mathit{send_warn}(t80)$ into X. There is no deontic inconsistency, so the check in step 6 fails, and then we jump to step 8, which returns the result:

$$X = \{\ \mathbf{O}\mathit{send_warn}(t80), \mathbf{Do}\,\mathit{send_warn}(t80), \mathbf{P}\mathit{send_warn}(t80)\ \}.$$

The operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ may be iteratively applied as follows.

$$\begin{aligned}
\mathbf{S}^0_{\mathcal{PP},\mathcal{O}^p} &= \emptyset.\\
\mathbf{S}^{i+1}_{\mathcal{PP},\mathcal{O}^p} &= \mathbf{S}_{\mathcal{PP},\mathcal{O}^p}(\mathbf{S}^i_{\mathcal{PP},\mathcal{O}^p}).\\
\mathbf{S}^{\omega}_{\mathcal{PP},\mathcal{O}^p} &= \bigcup_{i=0}^{\infty} \mathbf{S}^i_{\mathcal{PP},\mathcal{O}^p}.
\end{aligned}$$

The following theorem says that for positive paps, operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ is monotonic, continuous, and has a (unique) least fixpoint.

LEMMA 6.4 (MONOTONICITY AND CONTINUITY OF $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$). *Suppose $\mathcal{PP}$ is a positive pap. Then the operator $\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$ is monotone and continuous, i.e.,*

(1) $\mathcal{PS}_1 \subseteq \mathcal{PS}_2 \Rightarrow \boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}_1) \subseteq \boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}_2)$,

(2) $\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\bigcup_{i=0}^{\infty} \mathcal{PS}_0) = \bigcup_{i=0}^{\infty} \boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}_i)$ *for any chain* $\mathcal{PS}_0 \subseteq \mathcal{PS}_1 \subseteq \mathcal{PS}_2 \subseteq \cdots$ *of probabilistic status sets, and*

(3) $\boldsymbol{S}^{\omega}_{\mathcal{PP},\mathcal{O}^p}$ *is a fixpoint of* $\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$. *Moreover, it is the least fixpoint of* $\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$.

The following example shows the computation of $\mathbf{S}^{\omega}_{\mathcal{PP},\mathcal{O}^p}$.

Example 6.5. Consider the program in Example 6.3. Applying the operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ iteratively, we obtain:

$$\begin{aligned}
\mathbf{S}^0_{\mathcal{PP},\mathcal{O}^p} &= \emptyset.\\
\mathbf{S}^1_{\mathcal{PP},\mathcal{O}^p} &= \{\mathbf{O}send_warn(t80), \mathbf{Do}\, send_warn(t80), \mathbf{P}send_warn(t80)\}\\
\mathbf{S}^2_{\mathcal{PP},\mathcal{O}^p} &= \mathbf{S}^1_{\mathcal{PP},\mathcal{O}^p} \cup \{\mathbf{O}send_warn(t72), \mathbf{Do}\, send_warn(t72),\\
&\qquad \mathbf{P}send_warn(t72), \mathbf{F}move()\}\\
\mathbf{S}^3_{\mathcal{PP},\mathcal{O}^p} &= \mathbf{S}^2_{\mathcal{PP},\mathcal{O}^p} = \mathbf{S}^{\omega}_{\mathcal{PP},\mathcal{O}^p}
\end{aligned}$$

The following results tell us that Lemma 5.19, which holds for arbitrary programs, can be strengthened to the case of positive probabilistic programs.

THEOREM 6.6 (RATIONAL PROBABILISTIC STATUS SETS AS LEAST FIXPOINTS). *Suppose $\mathcal{PP}$ is a positive pap, and $\mathcal{O}^p$ is a probabilistic agent state. Then $\mathcal{PS}$ is a rational probabilistic status set if and only if $\mathcal{PS} = \mathbf{lfp}(\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p})$ and $\mathcal{PS}$ is a feasible probabilistic status set. Recall that* **lfp** *stands for least fixpoint.*

COROLLARY 6.7. *Let $\mathcal{PP}$ be a positive probabilistic agent program. Then, on every probabilistic agent state $\mathcal{O}^p$, the rational probabilistic status set of $\mathcal{PP}$ (if one exists) is unique, i.e., if $\mathcal{PS}, \mathcal{PS}'$ are rational probabilistic status sets for $\mathcal{PP}$ on $\mathcal{O}^p$, then $\mathcal{PS} = \mathcal{PS}'$.*

An important corollary of this theorem is that to compute a reasonable feasible status set of a pap, all we need to do is to compute $\mathbf{lfp}(\mathbf{S}_{\mathcal{PP},\mathcal{O}^p})$ and check if $\mathbf{lfp}(\mathbf{S}_{\mathcal{PP},\mathcal{O}^p})$ satisfies the integrity constraints, action constraints, and whether all actions to be performed have preconditions that are true with 100% probability in the current state. This may be done via Algorithm **Compute-lfp** below.

Algorithm 6.8 *(Reas. Prob. Status Set Computation for Positive paps).*
Compute-lfp ($\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$): $\mathcal{O}^p$ **prob. agent state,**
$\mathcal{PP}$**: prob. agent program)**

($\star$ the probabilistic agent program $\mathcal{PP}$ is positive; $\star$)
($\star$ **Input:** a probabilistic agent state $\mathcal{O}^p$, and a pap $\mathcal{PP}$ $\star$)
($\star$ **Output:** a reasonable probabilistic status set $\star$)

1. change := **true**; $X := \emptyset$;
2. **while** change **do**
3. $newX$ = **Compute-S**$_{\mathcal{PP},\mathcal{O}^p}$ **(X)**;
4. **if** $newX$ = *"no consistent set exists"*
5. **then return** no reasonable prob. status set exists.
6. **if** $X \neq newX$ **then** $X := newX$
7. **else** change := **false**.
8. **end while**;
9. **if** X satisfies all the following conditions
10. • **Do** $\alpha \in X \Rightarrow \mathcal{O}^p \models^{[1,1]} Pre(\alpha)$;
11. • The new state obtained by executing **conc**$(\{\alpha \mid \mathbf{Do}\,\alpha \in X\}, \mathcal{O}^p)$
12. satisfies the integrity constraints;
13. • X satisfies the action constraints.
14. **then return** X
15. **else return** no reasonable prob. status set exists.

end.

We note that it is not possible to interleave the check of the integrity constraints with the computation of the status set. This is because $\mathcal{PS}1 \subseteq \mathcal{PS}2$ does not imply that **conc**$(\{\alpha \mid \mathbf{Do}\,\alpha \in \mathcal{PS}1\}, \mathcal{O}^p) \subseteq$ **conc**$(\{\alpha \mid \mathbf{Do}\,\alpha \in \mathcal{PS}2\}, \mathcal{O}^p)$.

THEOREM 6.9 (POLYNOMIAL DATA COMPLEXITY). *Algorithm* ***Compute*-lfp** *has polynomial data-complexity.*

The following example walks the reader through the detailed working of this algorithm on the motivating example pap introduced earlier on in this paper.

Example 6.10. We apply the above algorithm to the program (and agent state) of Example 6.3:

—Step 1 initializes X to $\emptyset$, i.e., to $\mathbf{S}^0_{\mathcal{PP},\mathcal{O}^p}$, while steps 2–8 iteratively apply the procedure which implements the operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$.

—At the first iteration, in step 3 $newX$ becomes $\mathbf{S}^1_{\mathcal{PP},\mathcal{O}^p}$ (shown in Example 6.5); since there are no deontic inconsistencies, the test in step 4 fails; then we skip to step 6 which will assign $X := newX$, and the cycle starts again.

—At the second iteration $newX$ becomes $\mathbf{S}^2_{\mathcal{PP},\mathcal{O}^p}$; as it is still inconsistency-free and different from X, we iterate again.

—The third iteration is the final one, because $\mathbf{S}^3_{\mathcal{PP},\mathcal{O}^p} = \mathbf{S}^2_{\mathcal{PP},\mathcal{O}^p}$, and so we skip to the tests in steps 10–13.

—In our example, all actions have empty (hence trivially true) preconditions. Furthermore, as there are no integrity and action constraints, it follows immediately that X is returned as the (unique) reasonable status set of the program.

6.1 Agent Programs are Probabilistic Agent Programs

In this section, we show that the concept of an agent program is a special case of the framework defined here. Hence, paps generalize the Eiter et al. semantics. Fur-

thermore, algorithm **Compute-lfp** may be used to compute reasonable status sets of positive agent programs. First, we show how agent programs may be captured as paps.

Definition 6.11 *(Reductions).* Let $\mathcal{PP}$ be a probabilistic agent program, $\mathcal{PS}$ a probabilistic status set, and $\mathcal{O}^p$ a probabilistic agent state. Assume further that each random variable contains exactly one object with probability 1. Then we can define the following mappings:

$Red_1(\cdot)$,. which maps every random variable of the form $\langle\{o_{\mathbf{RV}}\}, 1\rangle$ to o:

$$\mathrm{Red}_1(\langle\{o_{\mathbf{RV}}\}, 1\rangle) = o.$$

$Red_2(\cdot)$,. which maps annotated code call conditions to code call conditions by simply removing the annotations and the conjunction strategy:

$$\mathrm{Red}_2(\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle) = \chi.$$

We can easily extend $\mathrm{Red}_2(\cdot)$ to a mapping from arbitrary conjunctions of annotated code calls to conjunctions of code calls.

$Red_3(\cdot)$,. which maps every probabilistic agent program to a nonprobabilistic agent program: it clearly suffices to define $\mathrm{Red}_3(\cdot)$ on probabilistic agent rules. This is done as follows:

$$\mathrm{Red}_3(A \leftarrow \Gamma \,\&\, L_1 \,\&\, \ldots \,\&\, L_n) = A \leftarrow \mathrm{Red}_2(\Gamma) \,\&\, L_1 \,\&\, \ldots \,\&\, L_n.$$

Under the above assumptions, the following theorem holds.

THEOREM 6.12 (SEMANTICS OF AGENT PROGRAMS AS AN INSTANCE OF PAPS). *Suppose all random variables have the form*

$$\langle\{\mathit{object}_{\mathbf{RV}}\}, 1\rangle.$$

Then: ($\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle$ is a ground annotated code call condition, $\mathcal{O}^p$ a probabilistic agent state)

Satisfaction:. the satisfaction relations coincide, i.e.,

$$\mathcal{O}^p \models^{[\mathsf{ai}_1, \mathsf{ai}_2]} \chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle \text{ if and only if } \mathcal{O}^p \models \mathit{Red}_2(\chi : \langle[\mathsf{ai}_1, \mathsf{ai}_2], \otimes\rangle).$$

App-Operators:. the App-Operators coincide, i.e.,

$$\boldsymbol{App}_{Red_3(\mathcal{PP}), \mathcal{O}^p}(\mathcal{PS}) = \boldsymbol{App}_{\mathcal{PP}, \mathcal{O}^p}(\mathcal{PS}),$$

where the operator on the left–hand side is the one introduced in Definition **??**.

Feasibility:. Feasible probabilistic status sets coincide with feasible status sets under our reductions, i.e., $\mathcal{PS}$ is a feasible probabilistic status set w.r.t. $\mathcal{PP}$ if and only if $\mathcal{PS}$ is a feasible status set w.r.t. $Red_3(\mathcal{PP})$.

Rational:. Rational probabilistic status sets coincide with rational status sets under our reductions, i.e., $\mathcal{PS}$ is a rational probabilistic status set w.r.t. $\mathcal{PP}$ if and only if $\mathcal{PS}$ is a rational status set w.r.t. $Red_3(\mathcal{PP})$.

Reasonable:. Reasonable probabilistic status sets coincide with reasonable status sets under our reductions, i.e., $\mathcal{PS}$ is a reasonable probabilistic status set w.r.t. $\mathcal{PP}$ if and only if $\mathcal{PS}$ is a reasonable status set w.r.t. $Red_3(\mathcal{PP})$.

Computation of Status Sets:. The computations of probabilistic status sets given in Algorithms 6.2 and 6.8 for a pap $\mathcal{PP}$ reduce to the computation of status sets for $Red_3(\mathcal{PP})$.

7. PROBABILISTIC AGENT PROGRAMS: KRIPKE–STYLE SEMANTICS

The definition of a feasible status set given in Section 5 makes several simplifying assumptions. First (see Definition 5.3), it assumes that an action can be executed only if its precondition is believed by the agent to be true in the agent state with probability 1. Second (see Definition 5.5), every action that is permitted must also have a precondition that is believed to be true with probability 1. In this section, we propose a Kripke–style[2] semantics for agent programs that removes these conditions.

To do this, we will start by noting that in a probabilistic state $\mathcal{O}^p$, the agent returns a set of random variables for each code call. Every probabilistic state implicitly determines a set of (ordinary) states that are "compatible" with it.

Definition 7.1 *(Evaluation of a Code Call w.r.t. a State).* We use the notation $\mathbf{eval}(\mathfrak{a}:f(d_1,\ldots,d_n),\mathcal{O})$ to denote the result of evaluating the code call $\mathfrak{a}:f(d_1,\ldots,d_n)$ w.r.t. the (ordinary) state $\mathcal{O}$, and $\mathbf{eval}(\mathfrak{a}:_{\mathbf{RV}}f(d_1,\ldots,d_n),\mathcal{O}^p)$ to denote the result of evaluating the probabuilistic code call $\mathfrak{a}:_{\mathbf{RV}}f(d_1,\ldots,d_n)$ w.r.t. the probabilistic state $\mathcal{O}^p$. The first one will return a set of objects, while the second one returns a set of random variables.

Definition 7.2 *(Compatibility w.r.t. a Probabilistic State:* $\mathsf{COS}(\mathcal{O}^p)$*).*
Let $\mathcal{O}^p$ be a probabilistic agent state. An (ordinary) agent state $\mathcal{O}$ is said to be *compatible* with $\mathcal{O}^p$ *if, by definition,* for every ground code call $\mathfrak{a}:f(d_1,\ldots,d_n)$, it is the case that for every object $o \in \mathbf{eval}(\mathfrak{a}:f(d_1,\ldots,d_n),\mathcal{O})$, there exists a random variable $(X,\wp) \in \mathbf{eval}(\mathfrak{a}:f(d_1,\ldots,d_n),\mathcal{O}^p)$ such that $o \in X$ and $\wp(o) > 0$, and there is no other object $o' \in X$ such that $o' \in \mathbf{eval}(\mathfrak{a}:f(d_1,\ldots,d_n),\mathcal{O})$.

We use the notation $\mathsf{COS}(\mathcal{O}^p)$ to denote the set of all ordinary agent states that are compatible with a probabilistic agent state $\mathcal{O}^p$.

The following example illustrates this concept.

Example 7.3. Consider a probabilistic agent state $\mathcal{O}^p$ with only two code calls $\mathfrak{surv}:\mathit{identify}(\mathit{image}1)$ and $\mathfrak{surv}:\mathit{location}(\mathit{image}1)$, which respectively return the random variables

$$\langle\{t80,t72,t70\},\{\langle t80,0.3\rangle,\langle t72,0.7\rangle,\langle t70,0.0\rangle\}\rangle$$

and $\langle\{loc2\},\{\langle loc2,0.8\rangle\}\rangle$. The agent states compatible w.r.t. $\mathcal{O}^p$ are described in the following table:

State	Vehicle	Location
1	none	none
2	t80	none
3	t72	none
4	none	loc2
5	t80	loc2
6	t72	loc2

[2]Our semantics will use multiple worlds. However, it differs from well–known Kripke models in the sense that the accessibility relation between worlds in standard Kripke semantics does not play a significant role for our purposes.

The object "t70" in the first random variable has a null probability, and hence it does not appear in any compatible agent state. In states 1–3, the location is unknown. In states 1 and 4, the vehicle in the image is unknown.

We are now ready to explain why a Kripke–style semantics is worth pursuing in our context. Consider an agent whose current probabilistic agent state is $\mathcal{O}^p$ and suppose $\mathcal{O}_1, \ldots, \mathcal{O}_r$ are all compatible with $\mathcal{O}^p$. Now consider some action α. The fact that $\mathcal{O}^p$ is compatible with $\mathcal{O}_1, \ldots, \mathcal{O}_r$ means that one of these (nonprobabilistic) states $\mathcal{O}_1, \ldots, \mathcal{O}_r$ is the "real" state—uncertainty arises because the agent does not precisely know which of these states is the "real" one. Clearly, action α either is or is not executable in the "real" state, whatever it might be. The agent $\mathfrak{a}$ may want to "attempt to execute" action α. In this case, one of two things occur:

(1) The "real" state is one in which the action's precondition is true (and hence the state is modified through the effects of the action). For instance, w.r.t. the above example, if the action has precondition "$\texttt{Vehicle} = t72 \wedge \texttt{Location} = loc2$" then the actions effects are realized if the "real" state is state 5—otherwise the actions effects are unrealized.

(2) The "real" state is one in which the action's precondition is false, and the action is in fact unexecutable (though the agent tried to execute the unexecutable action!). In this case that state is unchanged. This is what would occur if the "real" state is state 4 in the example mentioned above.

Thus, agents must be able to reason about the effects of their actions even when they are not exactly sure what the world state is. In fact, when viewed from a philosophical point of view, this is what most of us do anyway. We execute actions even when we do not know the state of the world, e.g. we may execute an action such as *phone_spouse* without knowing whether the spouse is at the number we are calling or not. Here there are two possible outcomes—one where we speak to our spouse (assuming he or she is at the target phone) or not (if he or she is not there). Such situations are what we seek to capture in this section.

We now define the notion of a probabilistic Kripke structure.

Definition 7.4 *(Probabilistic Kripke Structure $(\mathcal{S}, \wp)$).* A *probabilistic Kripke structure* is a pair $(\mathcal{S}, \wp)$ where $\mathcal{S}$ is a set of ordinary states, and $\wp : \mathcal{S} \rightarrow [0,1]$ is a mapping such that $\sum_{\mathcal{O} \in \mathcal{S}} \wp(\mathcal{O}) = 1$.

Definition 7.5 *(Compatible Probabilistic Kripke Structure $\mathsf{PKS}(\mathcal{O}^p)$).* Let $\mathcal{O}^p$ be a probabilistic agent state. A probabilistic Kripke structure $(\mathsf{COS}(\mathcal{O}^p), \wp)$ is said to be *compatible* with $\mathcal{O}^p$ *if, by definition,* for every ground code call $\mathfrak{a} : f(d_1, \ldots, d_n)$, for every random variable $(X, \wp') \in \mathbf{eval}(\mathfrak{a} : f(d_1, \ldots, d_n), \mathcal{O}^p)$, and for each object o, it is the case that

$$\sum_{o \in \mathbf{eval}(\mathfrak{a} : f(d_1, \ldots, d_n), \mathcal{O})} \wp(\mathcal{O}) = \begin{cases} \wp'(o) & \text{if } o \in X \\ 0 & \textit{otherwise.} \end{cases}$$

We use the notation $\mathsf{PKS}(\mathcal{O}^p)$ to denote the set of all probabilistic Kripke structures compatible with probabilistic agent state $\mathcal{O}^p$.

By definition, two distinct objects from the same random variable cannot appear in the same compatible state. If such a random variable has a complete probability

distribution, then in any compatible Kripke structure, the sum of the probabilities of the states containing one of its objects is equal to 1. This means that any (compatible) agent state containing no such objects will have a null probability, avoiding the intuitive inconsistency pointed out in Example 7.3.

Example 7.6. Considering the situation in Example 7.3 on the previous page, a probabilistic Kripke structure compatible with $\mathcal{O}^p$ is $\langle \mathsf{COS}(\mathcal{O}^p), \wp \rangle$, with the following probability distribution:

State	Probability
1	0
2	0.1
3	0.1

State	Probability
4	0
5	0.2
6	0.6

The following result says that compatible Kripke structures always exist.

THEOREM 7.7 (EXISTENCE OF COMPATIBLE PROBABIL. KRIPKE STRUCTURE). *Suppose $\mathcal{O}^p$ is a probabilistic agent state. Then there is at least one probabilistic Kripke structure which is compatible with it.*

However, in almost all cases, $\mathsf{PKS}(\mathcal{O}^p)$ contains an infinite number of elements.

THEOREM 7.8 (EXISTENCE OF INFINITELY MANY KRIPKE STRUCTURES). *If a probabilistic state $\mathcal{O}^p$ contains at least two random variables (returned by the same code call or by two distinct ones) containing at least one object with associated probability $0 < p < 1$, then there exist an infinite number of probabilistic Kripke structures compatible with $\mathcal{O}^p$.*

Given a code call condition χ, the *probability* that χ is true in a Kripke structure $\kappa = (\mathsf{COS}(\mathcal{O}^p), \wp) \in \mathsf{PKS}(\mathcal{O}^p)$, denoted by $\wp_\kappa(\chi)$, is easily seen to be given by the following formula:

$$\wp_\kappa(\chi) = \sum_{\mathcal{O} \in \mathsf{COS}(\mathcal{O}^p) \wedge \mathcal{O} \models \chi} \wp(\mathcal{O}).$$

This says that the probability of χ being true in κ is obtained via a two-step process: in the first step, we find all (ordinary) agents states $\mathcal{O} \in \mathsf{COS}(\mathcal{O}^p)$ such that χ is true in $\mathcal{O}$. We then add up the probabilities $\wp(\mathcal{O})$ of all such states to find the probability of χ. *It is important to note that in this process, the conjunction strategy $\otimes$ plays no role.*

Now, each of the Kripke structures compatible with probabilistic state $\mathcal{O}^p$ has its own probability distribution $\wp$ over $\mathsf{COS}(\mathcal{O}^p)$. The probability of χ being true in the set of all Kripke structures $\mathsf{PKS}(\mathcal{O}^p)$ (denoted $\wp_{\mathcal{O}^p}(\chi)$) has to take into account all such distributions; thus it will be a probability interval given by the following formula:

$$\wp_{\mathcal{O}^p}(\chi) = [\min_\kappa \wp_\kappa(\chi), \max_\kappa \wp_\kappa(\chi)].$$

We are now in a position to specify what it means to execute an action in a probabilistic Kripke structure.

Definition 7.9 *(Action Execution).* Suppose $\mathcal{O}^p$ is a probabilistic agent state. A ground action α is said to be *possibly executable in $\mathcal{O}^p$ if, by definition,* there is at

least one probabilistic Kripke structure $(\mathsf{COS}(\mathcal{O}^p), \wp) \in \mathsf{PKS}(\mathcal{O}^p)$, and an ordinary agent state $\mathcal{O}$ in $\mathsf{COS}(\mathcal{O}^p)$ in which the precondition of α is true such that $\wp(\mathcal{O}) > 0$. In this case, $\mathcal{O}$ *witnesses* the executability of α.

We say that α is *executable with a probability interval* $[\mathsf{p}_1, \mathsf{p}_2]$ in $\mathcal{O}^p$ *if, by definition,*

$$[\mathsf{p}_1, \mathsf{p}_2] \;=\; \wp_{\mathcal{O}^p}(\mathit{Pre}(\alpha)).$$

It is again important to understand this definition. $\mathsf{COS}(\mathcal{O}^p)$ contains all possible agent states compatible with $\mathcal{O}^p$—one of these is the "real" state, but we do not know which one. A witness to action α's executability does not say in fact that α is executable in the "real" state—just that there's some nonzero probability that α may be executable in the "real" state. The following example illustrates this definition.

Example 7.10. Let us consider the probabilistic agent state of Examples 7.3 and 7.6 and the following actions:

$$\begin{aligned}
\alpha_1 :\ \mathit{Pre}(\alpha_1) &= \mathbf{in}(\mathtt{t70}, \mathrm{surv} : \mathit{identify}(\mathtt{image1})) \\
\alpha_2 :\ \mathit{Pre}(\alpha_2) &= \mathbf{in}(\mathtt{X}, \mathrm{surv} : \mathit{location}(\mathtt{image1})) \;\&\; \mathtt{X} \neq \mathit{loc2} \\
\alpha_3 :\ \mathit{Pre}(\alpha_3) &= \mathbf{in}(\mathtt{loc2}, \mathrm{surv} : \mathit{location}(\mathtt{image1})) \;\&\; \\
&\quad\ \mathbf{in}(\mathtt{t80}, \mathrm{surv} : \mathit{identify}(\mathtt{image1}))
\end{aligned}$$

As stated before, the object "t70" cannot appear in any compatible agent state, and hence the action α_1 is not possibly executable. On the other hand, the precondition of α_2 requires the existence of an object other than "loc2", which is known to be the only one possibly returned by the corresponding code call, and hence α_2 is not possibly executable either. Eventually, the precondition of α_3 requires the presence of both objects "loc2" and "t80", which is true in the agent state number 5 described in the above examples. As this state has a nonzero probability, α_3 is possibly executable, and the agent state witnesses its executability.

We are now ready to define the new probabilistic Kripke structure that results when an action is executed in it.

Definition 7.11 *(Result of Action (θ,γ)-Execution).* Suppose $\mathcal{O}^p$ is a probabilistic agent state, $\alpha(\vec{\mathtt{X}})$ is an action, and γ is a ground substitution for all variables occurring in the precondition, add, and delete list of $\alpha(\vec{\mathtt{X}})\theta$. Let $(\mathsf{COS}(\mathcal{O}^p), \wp)$ be a probabilistic Kripke structure that contains a witness $\mathcal{O}$ to the possible executability of $\alpha(\vec{\mathtt{X}})\theta$. The *result of executing* action $\alpha(\vec{\mathtt{X}})$ under substitutions θ, γ in probabilistic Kripke structure $(\mathcal{S}, \wp) \in \mathsf{PKS}(\mathcal{O}^p)$ is a new Kripke structure $(\mathcal{S}', \wp')$ defined in the following way:

(1) $\mathcal{S}' = \{\mathrm{map}_{\alpha(\vec{\mathtt{X}}),\theta,\gamma}(\mathcal{O}) \mid \mathcal{O} \in \mathcal{S}\}$ where map is defined as follows:

$$\mathrm{map}_{\alpha(\vec{\mathtt{X}}),\theta,\gamma}(\mathcal{O}) = \begin{cases} \mathrm{apply}(\alpha(\vec{\mathtt{X}}), \theta, \gamma, \mathcal{O}) & \text{if } \mathcal{O} \in W, \\ \mathcal{O} & \text{otherwise.} \end{cases}$$

(2) $\wp'$ is defined as follows:

$$\wp'(\mathcal{O}') = \sum\{\wp(\mathcal{O}) \mid \mathcal{O}' = \mathrm{map}_{\alpha(\vec{\mathtt{X}}),\theta,\gamma}(\mathcal{O})\}.$$

In the above definitions, W is the set of all witnesses in $\mathcal{S}$ to the executability of $\alpha(\vec{\mathtt{X}})\theta\gamma$.

The *result of executing* action $\alpha(\vec{\mathtt{X}})$ under substitutions θ, γ in a set $\mathcal{K}$ of probabilistic Kripke structures is $\{\mathcal{S}' | \mathcal{S}'$ is obtained by executing $\alpha(\vec{\mathtt{X}})$ under substitutions θ, γ on $\mathcal{S} \in \mathcal{K}\}$.

The definition causes the agent states in which α's precondition is true to change, while those in which α's precondition is false stay unchanged. The probability of each final state is the sum of the probabilities of the corresponding (old) states. This is illustrated by the following example.

Example 7.12. Let us consider the compatible probabilistic Kripke structure in Example 7.6, and the action *erase*(X):

Pre:. $\mathbf{in}(\mathtt{X}, \mathrm{surv} : \mathit{identify}(\mathtt{image1}))$

Del:. $\mathbf{in}(\mathtt{X}, \mathrm{surv} : \mathit{identify}(\mathtt{image1}))$

Add:. $\emptyset$

The result of executing action *erase*(X) under substitutions $\{\mathtt{X}/t80\}, \epsilon$, is the probabilistic Kripke structure $\langle \mathcal{S}', \wp' \rangle$, briefly described by the following table:

State	Vehicle	Location	Probability
a	none	none	0.1
b	t72	none	0.1
c	none	loc2	0.2
d	t72	loc2	0.6

i.e., the states 1 and 2 merge together yielding the new state "a" and their probabilities are summed. Similarly, states 4 and 5 yield the new state "c".

The following result states that our definitions are coherent.

PROPOSITION 7.13 (CLOSURE OF PROBABILISTIC KRIPKE STRUCTURES). *The result of (θ, γ)-execution of an action in a probabilistic Kripke structure is also a probabilistic Kripke structure.*

7.1 p-Feasible Status Sets

Probabilistic Feasible Status Sets as defined in Definition 5.9 suffer from the following drawbacks:

(1) they prevent an agent from attempting to execute an action unless its precondition is known for sure to be true (which is exactly the intuitive reading of "$\mathcal{O}^p \models^{[1,1]} \mathit{Pre}(\alpha)$" in Definition 5.9(3),

(2) Likewise, an action constraint has to be checked only if its precondition is certainly true.

(3) Finally, state consistency requires that the execution of the actions does not corrupt the consistency of the original agent state, i.e., it has to lead to an agent state where the integrity constraints are with 100% probability.

In this section, we use the definition of a probabilistic Kripke structure introduced earlier to address these problems.

We first define the concept of p-feasibility with respect to a Kripke structure. p is a probability. p-feasibility weakens the above requirements to only requiring that preconditions are true with probability p (or higher).

Definition 7.14 *(Operator* $\mathbf{p\text{-}App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$*).* Suppose $\mathcal{PP}$ is a probabilistic agent program, $\mathcal{O}^p$ is a probabilistic agent state, and $\mathcal{PS}$ is a probabilistic status set. Then

$$\mathbf{p\text{-}App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \{\mathsf{Op}\,\alpha \mid \mathsf{Op}\,\alpha \text{ is the head of a ground instance of a rule } r \text{ in } \mathcal{PP}\} \text{ and:}$$

(1) $B^+_{as}(r) \subseteq \mathcal{PS}$ and $\{L : \neg L \in B^-_{as}(r)\} \cap \mathcal{PS} = \emptyset$.

(2) For every annotated code call condition $\chi : \langle[\mathsf{a}i_1, \mathsf{a}i_2], \otimes\rangle$ in the body of r, it is the case that the probability of χ w.r.t. $\mathsf{PKS}(\mathcal{O}^p)$ lies in the interval $[\mathsf{a}i_1, \mathsf{a}i_2]$.

(3) If $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, then the preconditions of α are true in $\mathsf{PKS}(\mathcal{O}^p)$ with probability p or greater.

(4) For every action status atom of the form $\mathsf{Op}\,\beta$ in $B^+_{as}(r)$ such that $\mathsf{Op} \in \{\mathbf{P}, \mathbf{O}, \mathbf{Do}\}$, the preconditions of β are true in $\mathsf{PKS}(\mathcal{O}^p)$ with probability p or greater.

Note the rather significant differences between this definition and that of operator $\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$ (Definition 5.3). First, here we consider probability of truth with respect to $\mathsf{PKS}(\mathcal{O}^p)$ (i.e., a set of Kripke structures) rather than with respect to an agent state; second, we consider probability of truth being greater than or equal to p in this case, while in Definition 5.3 we examined the case when it was only 1. This latter definition therefore differs in two respects—one is that the semantic structures w.r.t. which probabilistic truth of a code call condition is changed from an agent state to $\mathsf{PKS}(\mathcal{O}^p)$, and second, truth is w.r.t. a probability rather than being required to be 1.

Definition 7.15 *(Deontic and Action p-Consistency).* A probabilistic status set $\mathcal{PS}$ is *deontically p-consistent* with respect to a probabilistic agent state $\mathcal{O}^p$ *if, by definition,* it satisfies the following rules for any ground action α:

—If $\mathbf{O}\alpha \in \mathcal{PS}$, then $\mathbf{W}\alpha \notin \mathcal{PS}$.

—If $\mathbf{P}\alpha \in \mathcal{PS}$, then $\mathbf{F}\alpha \notin \mathcal{PS}$.

—If $\mathbf{P}\alpha \in \mathcal{PS}$, then the precondition of α is true in $\mathsf{PKS}(\mathcal{O}^p)$ with probability p or greater.

A probabilistic status set $\mathcal{PS}$ is *action p-consistent* with respect to an agent state $\mathcal{O}^p$ *if, by definition,* for every action constraint of the form

$$\{\alpha_1(\vec{\mathtt{X}}_1), \ldots, \alpha_k(\vec{\mathtt{X}}_k)\} \hookleftarrow \chi \tag{3}$$

either χ is not true in $\mathsf{PKS}(\mathcal{O}^p)$ with probability p or greater, or the following holds $\{\alpha_1(\vec{\mathtt{X}}_1), \ldots, \alpha_k(\vec{\mathtt{X}}_k)\} \not\subseteq \mathcal{PS}$.

Generalizing probabilistic state consistency to p-probabilistic state consistency may be done in two ways.

Definition 7.16 *(Probabilistic State p-Consistency).* A probabilistic status set $\mathcal{PS}$ is *weakly probabilistically state p-consistent* w.r.t. state $\mathcal{O}^p$ *if, by definition,*

the new state, $\mathcal{O}^{p'} = \mathbf{conc}(\mathbf{Do}(\mathcal{PS}), \mathcal{O}^p)$ obtained after concurrently executing all actions of the form $\mathbf{Do}\,\alpha \in \mathcal{PS}$ satisfies all integrity constraints with probability greater than or equal to p (i.e., for every integrity constraint $\psi \Rightarrow \chi$ either $\mathsf{PKS}(\mathcal{O}^{p'})$ does not satisfy ψ with probability p or higher, or $\mathsf{PKS}(\mathcal{O}^{p'})$ does satisfy ϕ with probability p or higher).

We say that a probabilistic status set $\mathcal{PS}$ is *strongly probabilistically state p-consistent* w.r.t. state $\mathcal{O}^p$ *if, by definition,* the new state $\mathcal{O}^{p'}$ satisfies the following condition: if $\mathcal{O}^p$ satisfies the integrity constraints with probability $\geq q$ ($q \in [0,1]$) then also $\mathcal{O}^{p'}$ does so.

These definitions induce two types of feasibility for arbitrary probabilities p.

Definition 7.17 *(Weak (resp. Strong) p-Feasibility).* Let $\mathcal{PP}$ be an agent program, and let $\mathcal{O}^p$ be an agent state. Then, a probabilistic status set $\mathcal{PS}$ is a *p-feasible probabilistic status set* for $\mathcal{PP}$ on $\mathcal{O}^p$, if the following conditions hold:

(p-$\mathcal{PS}$ 1). $\mathbf{p\text{-}App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) \subseteq \mathcal{PS}$ (closure under the program rules);

(p-$\mathcal{PS}$ 2). $\mathcal{PS}$ is deontically and action p-consistent (deontic/action p-consistency);

(p-$\mathcal{PS}$ 3). $\mathcal{PS}$ is action closed and deontically closed (deontic/action closure);

(p-$\mathcal{PS}$ 4). $\mathcal{PS}$ is weakly (resp. strongly) state p-consistent (state p-consistency).

Remark 7.18. If S is a p-feasible probabilistic status set for $\mathcal{PP}$ on $\mathcal{O}^p$ and $0 \leq q \leq p$, then S *is not always* q-feasible. Indeed, $[q,1] \supseteq [p,1]$, and then for any formula ϕ $\mathcal{O}^p \models^{[p,1]} \phi$ implies that $\mathcal{O}^p \models^{[q,1]} \phi$ (and analogously for $\not\models^{[p,1]}$ and $\not\models^{[q,1]}$). This means that all preconditions of actions, preconditions of action constraints, and integrity constraints which are verified for p are also verified for q. The problem is that $\mathbf{p\text{-}App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$ is antimonotonic w.r.t. p, as a smaller value for p may allow a larger set of rules to be firable. Then the closure under the program rules is not guaranteed any more.

The following example illustrates this point.

Example 7.19. Consider the following trivial program:

$$\mathbf{Do}\,\alpha \leftarrow \ ,$$

where $Pre(\alpha) = \mathbf{in}(\mathtt{a}, \mathtt{d} : f())$, and $\mathbf{eval}(\mathtt{d} : f(), \mathcal{O}^p) = \{\langle\{a\}, \langle 0.7\rangle\rangle\}$. Suppose $\mathcal{PS} = \emptyset$. Conditions p-$\mathcal{PS}$ 2–4 are true for any value of p. Note that $0.8\text{-}\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \emptyset$ and hence, $\mathcal{PS}$ is 0.8-feasible. In contrast, we see that $0.6\text{-}\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \{\mathbf{Do}\,\alpha\} \not\subseteq \mathcal{PS}$, and hence $\mathcal{PS}$ is not 0.6-feasible.

We can easily see that probabilistic feasibility is a particular case p-feasibility:

PROPOSITION 7.20. *Let $\mathcal{PP}$ be a probabilistic agent program, and let $\mathcal{O}^p$ be a 1-consistent agent state. Then, a probabilistic status set $\mathcal{PS}$ is a feasible probabilistic status set if and only if it is weakly 1-feasible if and only if it is strongly 1-feasible.*

7.2 p-Rational and p-Reasonable Status Sets

The notions of Rational and Reasonable Status Sets can be straightforwardly extended to those of p-rational and p-reasonable status sets.

Definition 7.21 *(p-Rational Status Set).* A probabilistic status set $\mathcal{PS}$ is a *p-rational probabilistic status set*, if $\mathcal{PS}$ is a p-feasible probabilistic status set and there exists no probabilistic status set $\mathcal{PS}' \subset \mathcal{PS}$ satisfying conditions (p-$\mathcal{PS}$1)–(p-$\mathcal{PS}$3) of a p-feasible probabilistic status set.

Obviously, in the case that $\mathcal{IC} = \emptyset$ (i.e., there are no integrity constraints) p-rational status sets are simply inclusion-minimal feasible status sets.

Definition 7.22 *(p-Reasonable Probabilistic Status Set).* Let $\mathcal{PP}$ be a probabilistic agent program; let $\mathcal{O}^p$ be a probabilistic agent state; and let $\mathcal{PS}$ be a probabilistic status set.

(1) If $\mathcal{PP}$ is a positive probabilistic agent program, then $\mathcal{PS}$ is a *p-reasonable probabilistic status set* for $\mathcal{PP}$ on $\mathcal{O}^p$, *if, by definition,* $\mathcal{PS}$ is a p-rational probabilistic status set for $\mathcal{PP}$ on $\mathcal{O}^p$.
(2) Exploiting the definition of $red^{\mathcal{PS}}(\mathcal{PP}, \mathcal{O}^p)$ (see Definition 5.13), $\mathcal{PS}$ is a *p-reasonable probabilistic status set* for $\mathcal{PP}$ w.r.t. $\mathcal{O}^p$, if it is a p-reasonable probabilistic status set of the program $red^{\mathcal{PS}}(\mathcal{PP}, \mathcal{O}^p)$ with respect to $\mathcal{O}^p$.

It is easy to verify that all p-reasonable probabilistic status sets are p-rational probabilistic status sets:

Proposition 7.23 (p-Reasonable Status Sets are p-Rational). *Let $\mathcal{PP}$ be a probabilistic agent program and $\mathcal{O}^p$ a probabilistic agent state. Then, every p-reasonable probabilistic status set of $\mathcal{PP}$ on $\mathcal{O}^p$ is a p-rational probabilistic status set of $\mathcal{PP}$ on $\mathcal{O}^p$.*

Proof. Identical to the proof of Proposition 5.16. □

As in Section 6, we can define a fixpoint operator and build an algorithm on its top to compute p-reasonable status sets for positive programs.

Definition 7.24 *(Operator* p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$*).*

$$\text{p}-\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}) = \mathbf{A\text{-}Cl}(\text{p}-\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})).$$

Operator p$-\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ can be computed by an algorithm identical to Algorithm 6.2, but for step 4, where the entailment $\models^{[1,1]}$ has to be replaced by $\models^{[p,1]}$. p$-\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$ is monotonic and continuous and has a unique least fixpoint.

Lemma 7.25 (Monotonicity and Continuity of p$-\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$). *Suppose $\mathcal{PP}$ is a positive* pap*. Then the operator* p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$ *is monotone and continuous, i.e.,*

(1) $\mathcal{PS}_1 \subseteq \mathcal{PS}_2 \Rightarrow \text{p}-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}_1) \subseteq \text{p}-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS}_2)$.
(2) p$-\boldsymbol{S}^{\omega}_{\mathcal{PP},\mathcal{O}^p}$ *is a fixpoint of* p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$*. Moreover, it is the least fixpoint of* p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$*. (We assume the iterations of* p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$ *are defined in the same way as the iterations of* $\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$*).*

The following result now follows immediately and has a proof similar to that of Theorem 6.6.

Theorem 7.26 (p-Rational Probab. Status Sets as Least Fixpoints). *Let $\mathcal{PP}$ be a positive probabilistic agent program, and let $\mathcal{O}^p$ be a probabilistic agent state. Then, $\mathcal{PS}$ is a p-rational probabilistic status set of $\mathcal{PP}$ on $\mathcal{O}^p$, if and only if $\mathcal{PS} =$* **lfp**(p$-\boldsymbol{S}_{\mathcal{PP},\mathcal{O}^p}$) *and $\mathcal{PS}$ is a p-feasible probabilistic status set.*

Uniqueness of the p-reasonable status set (if it exists) holds too, and then we can compute it by Algorithm 6.8, replacing—as usual—the entailment $\models^{[1,1]}$ with $\models^{[p,1]}$.

Unfortunately, the resulting algorithm is not polynomial because checking conditions in our Kripke–style semantics is a hard problem. In particular, steps (3), (10) and (13) require checking on the actual state, while steps (11)–(12) require the checking of integrity constraints in the state obtained by executing the selected actions. In the next section we present an algorithm for the last (more complex) problem, and then show how the first ones are particular cases of the last one.

7.3 Checking p-Consistency of Integrity Constraints

In this section, we provide an algorithm to check p-consistency of an integrity constraint IC after an action has been executed in probabilistic agent state $\mathcal{O}^p$, leading to a new state $\mathcal{O}^{p\prime}$ assuming that all integrity constraints are true in the original state $\mathcal{O}^p$. This is clearly needed because any agent which uses p-feasible, p-rational, and p-reasonable status sets for its computations must have the ability to perform such integrity checks.

Suppose $O_1, \ldots, O_N$ are all states compatible with $\mathcal{O}^p$ while $O'_1, \ldots, O'_{N'}$ are those compatible with the new state $\mathcal{O}^{p'}$. Let p_i, p'_i denote the probabilities of O_i, O'_i respectively. Then consider the following system of constraints:

minimize $\Sigma_{O'_i \models IC} p'_i$ **such that:**

(K)	$\sum_{i=1}^{N} p_i = 1$
(CK)	$\forall i \in \{1, \ldots, E\} : \sum_{j:Obj_i \in O_j} p_j = p(Obj_i)$
(IC)	$\forall IC \in \mathcal{IC} : 1 \geq \sum_{O_i : O_i \models IC} p_i \geq p$
(K $\rightarrow$ K')	$\forall i \in \{1, \ldots, N'\} : p'_i = \sum_{O_j : O_j \xrightarrow{\alpha} O'_i} p_j$
(IG)	$\forall i \in \{1, \ldots, N\} :$ $\max\{0, \sum_{Obj_k \in O_i} p(Obj_k) + 1 - \lvert O_i \rvert\} \leq p_i \leq \min_{Obj_k \in O_i}\{p(Obj_k)\}$

The objective function captures the probability of IC being true in the new Kripke structure. **(K)** and **(CK)** define any arbitrary compatible Kripke structure over N states w.r.t. $\mathcal{O}^p$ (which cointains E objects). **(IC)** expresses the fact that our actual state has to be p-consistent, while **(K$\rightarrow$ K')** defines the Kripke structure obtained after the execution of action α. Eventually, **(IG)** gives an upper and a lower bound to the probability of worlds (it extends the Boolean expression for conjunction of events of unknown interrelation).

Now, evaluating the truth of a code call condition (which can occur for checking action preconditions, action constraints, or any annotated code call condition) is just a particular case of the previous problem. Indeed, code call conditions have the form of simplified integrity constraints, and their truth probability has to be evaluated in the actual state. Thus it can be computed by the system of constraints presented above, discarding constraints **(K$\rightarrow$ K')** (since there is no state transition) and replacing the objective function with $\sum_{O_i \models \chi} p_i$, χ being the code call condition to be evaluated.

It is easy to see that a straightforward implementation of this algorithm (in both the versions: that for checking integrity constraints and that for checking code call conditions) requires exponential time and space. Indeed, it is composed of

$O(N) = O(2^E)$ variables and $O(N' + E + |IC|) = O(2^{E'} + E + |IC|)$ inequalities. However, under the assumption of having a bounded number (E and E') of objects in our world, the size of the system becomes linear w.r.t. the number of integrity constraints $|IC|$. Under these particular circumstances the system can be solved in polynomial time w.r.t. $|IC|$ by classical linear programming algorithms [Khachian 1979; Karmarkar 1984].

8. RELATED WORK

8.1 Work on Uncertainty

There has been an incredible amount of work on uncertainty in knowledge–based and database systems [Shafer and Pearl 1990]. However, almost all this work assumes that we are reasoning with logic or with Bayesian nets [Koller 1998], and most work proceeds under strong assumptions about the relationships between events (e.g. most Bayesian approaches assume conditional independence between events, while other approaches such as Fagin et al. [1990] and Ng and Subrahmanian [1993b] assume that we have no knowledge of the dependencies between events).

This paper introduces techniques to allow an agent developer to encode different assumptions about the relationships between events, when writing probabilistic agent programs. The idea of conjunction strategies to facilitate this was first introduced in the ProbView system [Lakshmanan et al. 1997] in an attempt to allow users querying probabilistic relational databases to express, in their query, their knowledge of the dependencies between events. Later, Dekhtyar and Subrahmanian [1997] extended the use of conjunction and disjunction strategies to the case of logic programs. In this paper, the idea of conjunction strategies is applied in the context of deontic-logic–based agent programs. We are not aware of any extant work on allowing flexible dependency assumptions in the context of logics and actions.

Research on epistemic logic (e.g., Morgenstern [1988], Kraus and Lehmann [1988] and Moore [1985]) enables reasoning about what is known and is not known at a given time. However, epistemic logics have not been used as a representation in decision making and in automated planning systems, perhaps, because the richness of these languages makes efficient reasoning very difficult. In contrast, our framework has polynomial data complexity.

Halpern and Tuttle [1992] study the semantics of reasoning about distributed systems when uncertainty is present. They develop a logic where a process has knowledge about the probability of events which facilitates decision-making by the process. We, on the other hand, consider probabilistic states, and as argued in [Dix et al. 2000] this also allows us to reason about probabilistic beliefs, i.e., probabilities are assigned to the agents' beliefs about events, rather than to the events themselves. That is, in Halpern's work Halpern and Tuttle [1992], the beliefs of the agent are CERTAIN, but in our framework, the beliefs of the agent may themselves be uncertain (with the phenomenon when they are certain being a special case of our framework).

Poole [1997] presented a framework that allows a natural specification of multi-agent decision problems. It extends logic with a new way to handle and think

about nondeterminism and uncertainty in terms of independent choices made by various agents, including nature and a logic program that gives the consequence of choices. It has general independence assumption. This work is more expressive than ours, but its generality leads to complexity problems and to difficulties in using the framework.

Haddawy [1991] developed a logic that allows to write sentences that describe uncertainty in the state of the world, uncertainty of action effects, combine possibility and chance, distinguish between truth and chance, and express information about probability distributions. He uses model–theoretic semantics and demonstrates how his logic can be used to specification of various reasoning and planning problems. The main purpose of the specification is to prove correctness, and not for programming of agents. Kushmerick et al. [1995] model uncertainty about the true state of the world with a probability distribution over the state space. Actions have uncertain effects, and each of these effects is also modeled with a probability distribution. They seek plans whose probability of success exceeds the threshold. They describe BURIDAN, an implemented algorithm for probabilistic planning. In contrast, we focus on programming agents, rather than on how agents will construct plans. Other researchers extended Kushmerick et al.'s model to increase the efficiency of the planning [Haddawy et al. 1996] or to more realistic domains [Doan 1996]. Thiébaux et al. [1995] developed a framework for anytime generation of plans under incomplete and ambiguous knowledge and actions with alternative and context dependent effects.

In Kaelbling et al. [1998], the authors propose using *partially observable Markov decision processes* (POMDPs) for planning under uncertainty. Similar to BURIDAN they use a probability distributions over states to express uncertainty about the situation of the agent. They also consider the problem of nondeterministic actions and getting feedback from the environment which we mentioned only briefly.

8.2 Work on Agent Programming

Shoham [1993] was perhaps the first to propose an explicit programming language for agents, based on object–oriented concepts, and based on the concept of an agent state. In Shoham's approach, an *"agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments"* [Shoham 1993]. He proposes a language, *Agent-0*, for agent programming, that provides a mechanism to express actions, time, and obligations. *Agent-0* is a simple, yet powerful language.

Closely related to Shoham's work is that of Hindriks et al. [1997] where an agent programming language based on *BDI*-agents is presented (*BDI* stands for *"Belief, Desires, Intentionality"*). They proceed upon the assumptions that an agent language must have the ability for updating beliefs, goals and for practical reasoning; finding means to achieve goals. Hindriks et al. [1997] argue that *"Now, to program an agent is to specify its initial mental state, the semantics of the basic actions the agent can perform, and to write a set of practical reasoning rules"* [Hindriks et al. 1997, p. 211].

Another related effort is that of Singh [1997] who builds upon previous work by Rosenschein [Rosenschein 1985; Rosenschein and Kaelbling 1995]. Rosenschein was perhaps the first to say that agents act according to states, and which actions

they take are determined by rules of the form

> *When P is true of the state of the environment, then the agent should take action A.*

Singh builds upon this, with a specific focus on heterogeneity in agents. Intuitively, in his framework, an agent is viewed as a finite state automaton—as is well–known, finite–state automata can be easily encoded in logic. This makes our framework somewhat more general than Singh's, instead of explicitly encoding automata (hard to do when an agent has hundreds of *ground* actions it can take). More importantly, this paper deals with uncertainty which has not been handled in Singh's framework.

All the above frameworks differ from ours in numerous ways. First, our language builds on top of arbitrary data structures, whereas theirs do not. Second, for us, states are instantiations of data structures managed by the program code associated with agents, while for them, states consist of beliefs, capabilities, choices, and commitments. The latter can be easily represented within our framework [Dix et al. 2000]. Most importantly, neither of the above frameworks deal with uncertainty management at all. Nor do they explicitly deal with integrity constraints or concurrent actions.

9. CONCLUSIONS

Agents are programs that autonomously react to changes in their environment by taking appropriate actions. In Eiter et al. [1999], the authors have proposed a framework within which agents may be built on top of an existing body of legacy code, and/or on top of specialized data structures appropriate for the intended functionality of the agent being built.

However, there are an increasing number of applications where agents are uncertain about what is true in their state (or environment). Such situations occur all the time in image identification programs, in programs that predict future events (such as battlefield events, stock market events, etc.), and in scenarios where an agent a attempts to predict what an agent b will do.

In this paper, we first introduce the concept of a probabilistic code call, which is a mechanism to describe uncertain application program interfaces for arbitrary functions over arbitrary data types. Based on this concept, we define probabilistic agent programs—sets of rules that encode the operating principles of an agent. Such rules encode the *probabilistic* conditions under which an agent is obliged to take some actions, permitted to take some actions, and/or forbidden to take some actions.

We then provide two broad classes of semantics for such "probabilistic agents." In the first class of semantics, actions that are permitted, obligatory or done, must have preconditions that are true with 100% probability in the current agent state. In the second class of semantics (which use probabilistic variants of Kripke structures), the actions that are permitted, obligatory or done, must have preconditions that are true with at least a given probability. This latter class of semantics allows reasoning by cases. We provide complexity arguments showing that though the second family of semantics is perhaps epistemologically more appealing than the first, the second family of semantics is also computationally more complex.

Finally, the paper includes algorithms to compute the semantics of probabilistic agent programs, as long as such programs are negation free.

Future work on probabilistic agent programs will focus on computing the semantics of paps that contain negation. A detailed study of computational complexity is also envisaged. We are also interested in identifying polynomially computable fragments of paps and implementing them on top of the current *IMPACT* implementation. Last, but not least, *IMPACT* has been used in a battlefield monitoring application where there is considerable uncertainty in predicting tactical enemy movements. We hope to build an application of paps addressing this problem, once the implementation of paps is complete.

A. NOTATION TABLE

Notation	Location	Description
$\mathcal{O}_{\mathtt{a}}(t)$ (or $\mathcal{O}_{\mathtt{a}}$ or $\mathcal{O}$)	Page 210	State of agent $\mathtt{a}$ at time t
$\mathtt{a}:f(d_1,\ldots,d_n)$	Page 211	Code call of function f by agent $\mathtt{a}$ with parameters $\mathtt{d_i}$
$\mathbf{in}(\mathtt{X},\mathtt{cc})$	Page 211	Code call atom
$\mathbf{not_in}(\mathtt{X},\mathtt{cc})$	Page 211	Negated code call atom
χ	Page 211	Code call condition
$\mathbf{conc}(S,\mathcal{O})$	Page 212	Concurrent execution of actions S on state $\mathcal{O}$
$\mathbf{P}\alpha,\mathbf{F}\alpha,\mathbf{W}\alpha,\mathbf{Do}\,\alpha,\mathbf{O}\alpha$	Page 212	Status atoms over action α
$\mathtt{a}:_{\mathbf{RV}}f(d_1,\ldots,d_n)$	Page 213	Probabilistic code call of f in $\mathtt{a}$ with parameters $\mathtt{d_i}$
$\mathcal{O}^p(t)$ (or $\mathcal{O}^p$)	Page 215	Probabilistic agent state at time t
$o \models^{[\ell,u]}_{\mathcal{O}^p}$	Page 215	Probabilistic satisfaction by object o in state $\mathcal{O}^p$
$\otimes$	Page 215	Probabilistic conjunction strategy
$\chi:\langle[\mathsf{ai}_1,\mathsf{ai}_2],\otimes\rangle$	Page 217	Annotated code call condition
$B^+_{as}(r)$ and $B^-_{as}(r)$	Page 217	Positive (resp. negative) action status atoms in rule r
$\mathcal{O}\models^{[\mathsf{ai}_1,\mathsf{ai}_2]}\chi:\langle[\mathsf{ai}_1,\mathsf{ai}_2],\otimes\rangle$	Page 218	Satisfaction of annotated code call condition $\chi:\langle[\mathsf{ai}_1,\mathsf{ai}_2],\otimes\rangle$
$\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$	Page 219	Rules execution operator for paps
$red^{\mathcal{PS}}(\mathcal{PP},\mathcal{O}^p)$	Page 223	Reduct of $\mathcal{PP}$ w.r.t. $\mathcal{PS}$ and $\mathcal{O}^p$
$\mathbf{D\text{-}Cl}(\mathcal{PS})$ and $\mathbf{A\text{-}Cl}(\mathcal{PS})$	Page 224	Deontic and action closure of probabilistic status set $\mathcal{PS}$
$\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$	Page 224	Immediate consequence operator for paps
$\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$	Page 224	Variant of $\mathbf{T}_{\mathcal{PP},\mathcal{O}^p}$ for positive paps
$\mathbf{eval}(\mathtt{a}:f(d_1,\ldots,d_n),\mathcal{O})$	Page 229	Evaluation of a code call w.r.t. state $\mathcal{O}$
$\mathsf{COS}(\mathcal{O}^p)$	Page 229	Ordinary states compatible with state $\mathcal{O}^p$
$\mathsf{PKS}(\mathcal{O}^p)$	Page 230	Set of probabilistic Kripke structures compatible with $\mathcal{O}^p$
$\mathbf{p\text{-}App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$	Page 234	Extension of operator $\mathbf{App}_{\mathcal{PP},\mathcal{O}^p}(\mathcal{PS})$
$\mathrm{p}{-}\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$	Page 236	Extension of operator $\mathbf{S}_{\mathcal{PP},\mathcal{O}^p}$

ELECTRONIC APPENDIX

The appendix for this article can be accessed in the ACM Digital Library by visiting the URL site for this article (http://www.acm.org/pubs/citations/journals/tocl/2000-1-2/p207-dix)

REFERENCES

Apt, K. 1990. Logic Programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B. Elsevier Science Publishers B.V. (North-Holland), Chapter 10, 493–574.

Arisha, K., Ozcan, F., Ross, R., Subrahmanian, V. S., Eiter, T., and Kraus, S. 1999. IMPACT: A Platform for Collaborating Agents. *IEEE Intelligent Systems 14*, 64–72.

Boole, G. 1854. *The Laws of Thought.* Macmillan, London.

Cattell, R. G. G., et al., Ed. 1997. *The Object Database Standard: ODMG-93.* Morgan Kaufmann.

Cox, I. J., Miller, M. L., Omohundro, S. M., and Yianilos, P. N. 1996. Pichunter: Bayesian relevance feedback for image retrieval. In *Proceedings Intl. Conf. on Pattern Recognition, Vienna, Austria.*

Dekhtyar, A. and Subrahmanian, V. S. 1997. Hybrid Probabilistic Logic Programs. In *Proceedings of the 14. International Conference on Logic Programing*, L. Naish, Ed. MIT Press, Leuven, Belgium, 391–405.

Devroye, L., Gyorfi, L., and Lugosi, G. 1996. *A Probabilistic Theory of Pattern Recognition.* Springer Verlag.

Dix, J., Subrahmanian, V. S., and Pick, G. 2000. Meta Agent Programs. *Journal of Logic Programming 46,* 1-2, 1–60.

Doan, A. 1996. Modeling Probabilistic Actions for Practical Decision-Theoretic Planning. In *Proceedings of the Third International Concerence on Artificial-Intelligence Planning Systems.* Edinburgh, Scotland, UK.

Eiter, T., Subrahmanian, V., and Pick, G. 1999. Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence 108,* 1-2, 179–255.

Eiter, T. and Subrahmanian, V. S. 1999. Heterogeneous Active Agents, II: Algorithms and Complexity. *Artificial Intelligence 108,* 1-2, 257–307.

Eiter, T., Subrahmanian, V. S., and Rogers, T. J. 2000. Heterogeneous Active Agents, III: Polynomially Implementable Agents. *Artificial Intelligence 117,* 1, 107–167.

Fagin, R., Halpern, J. Y., and Megiddo, N. 1990. A logic for reasoning about probabilities. *Information and Computation 87,* 1/2 (July/August), 78–128.

Friedman, N. and Russell, S. 1997. Image segmentation in video sequences: A probabilistic approach. In *Proceedings 13. Conf. on Uncertainty in Artificial Intelligence.*

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings 5. International Conference and Symposium.* MIT Press, Cambridge, Massachusetts, 1070–1080.

Haddawy, P. 1991. Representing Plans under Uncertainty: A Logic of Time, Chance and Action. Ph.D. thesis, University of Illinois. Technical Report UIUCDCS-R-91-1719.

Haddawy, P., Doan, A., and Goodwin, R. 1996. Efficient Decision-Theoretic Planning: Techniques and Empirical Analysis. In *Proceedings of the Third International Concerence on Artificial-Intelligence Planning Systems.* Edinburgh, Scotland, UK.

Halpern, J. Y. and Tuttle, M. 1992. Knowledge, Probability and Adversaries. Tech. rep., IBM. IBM Research Report.

Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J. J. C. 1997. Formal Semantics of an Abstract Agent Programming Language. In *International Workshop on Agent Theories, Architectures, and Languages.* Providence, RI, 204–218.

Huang, T. and Russell, S. 1998. Object identification: A Bayesian analysis with applications to traffic surveillance. *Artificial Intelligence 103(1-2)*, 77–93.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. 1998. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence 101*, 99–134.

KARMARKAR, N. 1984. A New Polynomial Algorithm for Linear Programming. *Combinatorica 4*, 373–395.

KHACHIAN, L. 1979. A polynomial Algorithm in Linear Programming. *Dokl. Akad. Nauk SSSR 244*, 1093–1096. English Translation in Soviet. Math. Dokl., 20 (1979), 191–194.

KIFER, M. AND SUBRAHMANIAN, V. S. 1992. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming 12,* 4, 335–368.

KOLLER, D. 1998. Structured Probabilistic Models: Bayesian Networks and Beyond. In *Proceedings of the 15. National Conference on Artificial Intelligence, AAAI'98.* AAAI Press/MIT Press, Madison, Wisconsin. `http://robotics.Stanford.EDU/~koller/Invited98`.

KRAUS, S. AND LEHMANN, D. 1988. Knowledge, Belief and Time. *Theoretical Computer Science 58*, 155–174.

KUSHMERICK, N., HANKS, S., AND WELD, D. 1995. An Algorithm for probabilistic planning. *Artificial Intelligence 76,* 1-2, 239–286.

LAKSHMANAN, V. S., LEONE, N., ROSS, R., AND SUBRAHMANIAN, V. S. 1997. ProbView: A Flexible Probabilistic Database System. *ACM Transactions on Database Systems 22,* 3 (September), 419–469.

LEI, Z., KEREN, D., AND COOPER, D. B. 1995. Computationally fast bayesian recognition of complex objects based on mutual algebraic invariants. In *Proceedings IEEE International Conference on Image Processing.*

LLOYD, J. 1984, 1987. *Foundations of Logic Programming.* Springer-Verlag, Berlin, Germany.

MOGHADDAM, B. AND PENTLAND, A. 1998. Beyond eigenfaces: Probabilistic matching for face recognition. In *Proceedings International Conference on Automatic Face and Genstures Recognition, Nara, Japan.*

MOORE, R. 1985. A Formal theory of Knowledge and Action. In *Formal Theories of the Commonsesnse World*, J. Hobbs and R. Moore, Eds. ABLEX publishing, Norwood, N.J.

MORGENSTERN, L. 1988. Foundations of a Logic of Knowledge, Action, and Communication. Ph.D. thesis, New York University.

NG, R. AND SUBRAHMANIAN, V. S. 1993a. A Semantical Framework for Supporting Subjective and Conditional Probabilities in Deductive Databases. *Journal of Automated Reasoning 10,* 2, 191–235.

NG, R. AND SUBRAHMANIAN, V. S. 1993b. Probabilistic Logic Programming. *Information and Computation 101,* 2, 150–201.

POOLE, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence 94,* 1-2, 7–56.

ROSENSCHEIN, S. J. 1985. Formal Theories of Knowledge in AI and Robotics. *New Generation Computing 3,* 4, 345–357.

ROSENSCHEIN, S. J. AND KAELBLING, L. P. 1995. A Situated View of Representation and Control. *Artificial Intelligence 73*, 149–173.

ROSS, S. 1997. *A First Course in Probability.* Prentice-Hall.

SHAFER, G. AND PEARL, J., Eds. 1990. *Readings in uncertain reasoning.* Morgan Kaufmann.

SHOHAM, Y. 1993. Agent Oriented Programming. *Artificial Intelligence 60*, 51–92.

SIEGAL, J. 1996. *CORBA Fundementals and Programming.* John Wiley & Sons, New York.

SINGH, M. P. 1997. A Customizable Coordination Service for Autonomous Agents. In *International Workshop on Agent Theories, Architectures, and Languages.* Providence, RI, 86–99.

SUBRAHMANIAN, V., BONATTI, P., DIX, J., EITER, T., KRAUS, S., ÖZCAN, F., AND ROSS, R. 2000. *Heterogenous Active Agents.* MIT-Press.

SUBRAHMANIAN, V. S. 1987. On the Semantics of Quantitative Logic Programs. In *Proceedings of the 4. IEEE Symposium on Logic Programming.* Computer Society Press, 173–182.

THIÉBAUX, S., HERTZBERG, J., SHOAFF, W., AND SCHNEIDER, M. 1995. A stochastic model of actions and plans for anytime planning under uncertainty. *International Journal for Intelligent Systems 10,* 2.

VAILAYA, A., FIGUEIREDO, M., JAIN, A., AND ZHANG, H. 1999. A bayesian framework for semantic classification of outdoor vacation images. In *Proceedings Conference on Electronic Imaging, San Jose, California.*

VAN EMDEN, M. AND KOWALSKI, R. 1976. The semantics of predicate logic as a programming language. *JACM 23*, 733–742.

VASCONCELOS, N. AND LIPPMAN, A. 1998. Embedded mixture modeling for efficient probabilistic content-based indexing and retrieval. In *Proceedings Multimedia Storage and Archiving Systems III, Boston.*

Received October 1999; revised March 2000; accepted March 2000