International Journal of Approximate Reasoning 165 (2024) 109101

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

journal homepage: www.elsevier.com/locate/ijar

An answer set programming-based implementation of epistemic probabilistic event calculus

Fabio Aurelio D'Asaro [a,*], Antonis Bikakis [b], Luke Dickens [b], Rob Miller [b]

[a] Department of Human Sciences, University of Verona, Verona, Italy
[b] Department of Information Studies, University College London, London, UK

ARTICLE INFO

Keywords:
Answer set programming (ASP)
Epistemic reasoning
Probabilistic reasoning
Event calculus
Knowledge representation
Artificial intelligence

ABSTRACT

We describe a general procedure for translating Epistemic Probabilistic Event Calculus (EPEC) action language domains into Answer Set Programs (ASP), and show how the Python-driven features of the ASP solver Clingo can be used to provide efficient computation in this probabilistic setting. EPEC supports probabilistic, epistemic reasoning in domains containing narratives that include both an agent's own action executions and environmentally triggered events. Some of the agent's actions may be belief-conditioned, and some may be imperfect sensing actions that alter the strengths of previously held beliefs. We show that our ASP implementation can be used to provide query answers that fully correspond to EPEC's own declarative, Bayesian-inspired semantics.

1. Introduction

In [15] we described EPEC (Epistemic Probabilistic Event Calculus), an action language that supports probabilistic, epistemic reasoning about narratives of action occurrences and environmentally triggered events, and in particular facilitates reasoning about future belief-conditioned actions and their consequences in domains that include both perfect and imperfect sensing actions. The semantics of EPEC is fully declarative and reflects a Bayesian view of probabilistic knowledge as justified degrees of belief.[1] In this paper we show how answer set programming (ASP) may be used to compute EPEC entailment, and show by example that this computation can be made tractable for reasonably small but non-trivial domains. More precisely, we show how practical implementation issues, arising from the necessity of calculating with real number probability values, can be addressed using the state-of-the-art ASP engine Clingo (see e.g. [18]) and in particular its interface with the interpreted language Python. In [15] we discussed how models of domains expressed in two other frameworks for probabilistic reasoning about actions, e.g., see [3] and [2], can be transformed into EPEC domains, and so the implementation described here indirectly offers a computational mechanism for these frameworks as well.

We use the following example to illustrate some of the features of EPEC domains and how the various EPEC proposition types are translated into ASP:

* Corresponding author.
E-mail addresses: fabioaurelio.dasaro@univr.it (F.A. D'Asaro), a.bikakis@ucl.ac.uk (A. Bikakis), l.dickens@ucl.ac.uk (L. Dickens), r.s.miller@ucl.ac.uk (R. Miller).

[1] For this reason some readers may find it useful to think of EPEC as supporting *doxastic* rather than *epistemic* reasoning. The reader's preference between these two terms may depend on whether he or she regards probabilistic knowledge, involving statistically justified numerical degrees of belief, as "true" knowledge. See https://www.sciencedirect.com/topics/mathematics/epistemic-probability for some example perspectives about this.

https://doi.org/10.1016/j.ijar.2023.109101
Received 23 March 2023; Received in revised form 5 December 2023; Accepted 5 December 2023

Available online 14 December 2023

Example 1.1. "A crime has almost certainly been committed on the island of Epécos. Earlier in the day, Detective Smart at the mainland police station received a telephone call from Ms Prigg. Ms Prigg reported that, as she arrived at the only hotel on the otherwise uninhabited island, she had felt someone touching her shoulder bag from behind. She had turned and caught a glimpse of the person, and was confident she would recognise him again. After this event there was no money in her bag, and she was almost certain that she had money in the bag previously. From experience, Det. Smart initially judged that there was 90% probability that money had been stolen, and initially reasoned that if this were the case then one of the other 962 guests and staff in the hotel must have taken the money, each with equal likelihood. However, later, on arriving at the island, she found a partial fingerprint on the bag that matched with those on record of the only hotel occupant with a criminal record (for drink-driving), a Mr Biggs. It is known that there is a 0.1% false positive rate and 5% false negative rate with these kinds of matches, and that fingerprints are generally left during this type of crime 70% of the time. At this point, Det. Smart calculates that there is more than a 33% probability that Biggs is the thief, so she decides to conduct an identity parade including Biggs back on the mainland. Identity parades, in these general circumstances, correctly identify suspects 85% of the time and incorrectly identify non-suspects 10% of the time. She reasons that if Ms Prigg identifies Biggs in the identity parade, the subsequent combination of evidence will give a more than 80% likelihood that Mr Biggs is the thief. Under police rules this will enable Det. Smart to charge him with stealing. Otherwise, she will have to leave the probable crime unsolved." □

Like the medical example in [15], this example contains many of the general features that motivated the development of EPEC. It includes a *narrative* of past and future events, some of which (e.g. the theft) are probable rather than certain, and only some of which are actions performed by the agent (Det. Smart). Some of the *causal information* about actions is probabilistic (e.g. thieving causes fingerprints with probability 0.7). The actions of dusting for fingerprints and conducting an identity parade are both *imperfect sensing actions* that alter the detective's probabilistic knowledge. And the detective's tentative plan to charge Biggs is *conditioned on a future belief state* resulting from the outcome of the identity parade. EPEC's ability to model this last type of feature (belief conditioned actions) is particularly important for epistemic reasoning about dynamic domains, and to our knowledge is not facilitated by any other probabilistic framework for epistemic reasoning about actions. In particular, epistemic planning that utilises sensing actions also requires belief conditioned actions, since there is little point in an agent adjusting its beliefs through sensing if it is not able to subsequently adjust its behaviour in the light of its new belief state. Finally, note that the specific probability values employed in the example – e.g. the initial 90% probability of theft, and the 0.1% false positive rate of fingerprint matches – must be supplied by the EPEC user and are assumed to be externally justified. Furthermore, there is no implicit simplifying independence assumption between EPEC fluents and actions – if present this has to be made explicit in the domain description. This is in contrast to frameworks such as ProbLog [16] that employ the distribution semantics [30].

2. Overview of EPEC

In this section we recap the syntax and semantics of EPEC, as presented in [15]. Definitions 2.1 to 2.44 are taken directly and unchanged from [15].

2.1. Syntax

In order to describe the various types of proposition that EPEC domain descriptions contain, we first define their key components. An EPEC domain language has sorts $\mathcal{F}$ for *fluents*, $\mathcal{A}_e$ for *environmental actions*, $\mathcal{A}_a$ for *agent actions*, $\mathcal{I}$ for *instants* (timepoints), and $\mathcal{V}$ *values*. $\mathcal{F}$, $\mathcal{A}_e$, $\mathcal{A}_a$ and $\mathcal{V}$ are finite and $\mathcal{I}$ is a non-empty, totally ordered, possibly infinite (discrete or continuous) set of *instants* with a least element $\bar{0}$. Agent actions are actions under the control of the agent being modelled, such as a patient choosing to take antibiotics or a detective dusting for fingerprints. Environmental actions are events that happen around that agent, such as a patient contracting an infection or a crime being committed in a detective's jurisdiction. *Literals* such as $F = V$ (for $F \in \mathcal{F}$) and $A = V$ (for $A \in \mathcal{A}$) assign a value V from $\mathcal{V}$ to elements of $\mathcal{F}$, $\mathcal{A}_e$ and $\mathcal{A}_a$, with the constraint that actions must be Boolean-valued. For $Z \in \mathcal{F} \cup \mathcal{A}_e \cup \mathcal{A}_a$, $Z = true$ (resp. $Z = false$) can be shortened to Z (resp. $\neg Z$). *i-Literals* of the form $[L]@I$ associate a literal L with an instant I. Literals (resp. i-literals) can be combined in *formulas* (resp. *i-formulas*) using the standard propositional connectives. Sets of literals that mention each fluent and action exactly once are called *states* and sets that mention each fluent but no actions are called *fluent states*. Subsets of (fluent) states are called *partial (fluent) states*. Finally, an *outcome* is a pair of the form $(\tilde{X}, P^+)$ where $\tilde{X}$ is a partial fluent state and $P^+ \in (0, 1]$ is a non-zero probability. To give the formal details, here are the definitions relating to an EPEC domain language, taken from [15]:

Definition 2.1 *(Domain Language).* An EPEC *domain language* is a tuple $\langle \mathcal{F}, \mathcal{A}, \mathcal{A}_e, \mathcal{A}_a, \mathcal{V}, vals, \mathcal{I}, \leq, \bar{0} \rangle$, where $\mathcal{F}$ is a finite non-empty set of *fluents*, $\mathcal{A}$ is a finite set of *actions*, $\mathcal{A}_e$ is a finite set of *environmental actions*, $\mathcal{A}_a$ is a finite set of *agent actions*, $\mathcal{A} = \mathcal{A}_e \cup \mathcal{A}_a$ and $\mathcal{A}_e \cap \mathcal{A}_a = \emptyset$, $\mathcal{V}$ is a finite non-empty set of *values* such that $\{false, true\} \subseteq \mathcal{V}$, *vals* is a function mapping elements in $\mathcal{F} \cup \mathcal{A}$ to tuples of elements (without repetitions) from $\mathcal{V}$ (i.e. *vals* specifies the values that each fluent and action can take), and $\mathcal{I}$ is a non-empty set of *instants* (i.e. time-points) with a minimum element $\bar{0}$ w.r.t. a total ordering $\leq$ over $\mathcal{I}$. For every $A \in \mathcal{A}$, $vals(A) = \langle false, true \rangle$ and for any $X \in \mathcal{F} \cup \mathcal{A}$ the expression $V \in vals(X)$ means that if $vals(X) = \langle V_1, \ldots, V_n \rangle$ then $V = V_i$ for some $1 \leq i \leq n$. □

Definition 2.2 *(Fluent and Action Literals, i-Literals).* A *fluent literal* is an expression of the form $F = V$ for some $F \in \mathcal{F}$ and $V \in vals(F)$. A fluent is *boolean* if $vals(F) = \langle false, true \rangle$. An *action literal* is either $A = false$ or $A = true$ for some $A \in \mathcal{A}$. Where no ambiguity can

arise, $Z = \mathit{true}$ and $Z = \mathit{false}$ are sometimes abbreviated to Z and $\neg Z$ respectively for a fluent or action Z. A *literal* is either a fluent literal or an action literal, and an *i-literal* is an expression of the form $[L]@I$ for some literal L and some $I \in \mathcal{I}$. □

Definition 2.3 *(Formulas, Fluent Formulas, i-Formulas).* The set of *formulas*, denoted by Θ, is the closure of the set of literals under $\wedge$, $\vee$, $\neg$ and $\rightarrow$. A formula θ is said to be a *fluent formula* if it contains no action literals. The set of *i-formulas*, denoted by Φ, is the closure of the set of i-literals under $\wedge$, $\vee$, $\neg$ and $\rightarrow$. The shorthand $[\theta]@I$ stands for the i-formula formed from the formula θ and the instant I by replacing all literals L occurring in θ by $[L]@I$ (e.g. $[F = V \rightarrow F' = V']@3$ is shorthand for $[F = V]@3 \rightarrow [F' = V']@3$). The symbol $\top$ stands for an arbitrary tautological formula. □

Definition 2.4 *(State, Partial State, Fluent State).* A *state* S is a set of literals, exactly one for each $F \in \mathcal{F}$ and $A \in \mathcal{A}$. A *partial state* is a subset $X \subseteq S$ of a state S. The subset of a partial state X containing exactly the fluent literals in X is a *partial fluent state*, and is denoted by $X \restriction \mathcal{F}$. For S a state, $S \restriction \mathcal{F}$ is called a *fluent state*. The subset of X containing exactly the action literals in X is denoted by $X \restriction \mathcal{A}$. The set of all states is denoted by $\mathcal{S}$, and the set of all partial states is denoted by $\mathcal{X}$. Finally, the sets $\{S \restriction \mathcal{F} \mid S \in \mathcal{S}\}$ and $\{X \restriction \mathcal{F} \mid X \in \mathcal{X}\}$ are denoted by $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{X}}$ respectively. □

Definition 2.5 *(Outcome, Projection Functions).* An *outcome* is a pair of the form $(\tilde{X}, P^+)$ where $\tilde{X}$ is a partial fluent state and $P^+ \in (0, 1]$ (i.e. P^+ is a non-zero probability). The *projection functions* χ and π are such that for any outcome $O = (\tilde{X}, P^+)$, $\chi(O) = \tilde{X}$ and $\pi(O) = P^+$. The set of all outcomes $\tilde{\mathcal{X}} \times (0, 1]$ is denoted by $\mathcal{O}$. □

Definition 2.6 *(Weight of a Set of Outcomes).* Given a finite set of outcomes $B = \{O_1, O_2, \ldots, O_m\}$ the *weight of* B is defined as

$$\pi(B) = \sum_{i=1}^{m} \pi(O_i).$$

(i.e. the sum of the probabilities of its elements.) □

To understand the intended meaning of an outcome, consider an action that, when legally executed in a state, has an outcome of the form $(\{F_1 = V_1, ..., F_n = V_n\}, P^+)$ (expressed within a *c-proposition* – see below). The intuitive meaning of this is that there is a probability P^+ that the values of fluents $F_1, \ldots, F_n$ will be changed to $V_1, \ldots, V_n$, respectively. For any literals not mentioned in the outcome, their truth values will persist.

We model the hotel theft scenario[2] of Example 1.1 with fluents *MoneyInBag*, *BiggsPrints*, *BiggsIsThief* and *OtherIsThief* (all Boolean), environmental actions *BiggsSteals* and *OtherSteals*, and agent actions *DustForPrints*, *Charge* and *DoIdParade*. As examples of EPEC's syntax described above, *BiggsSteals* and $\neg$*MoneyInBag* are literals, $(\{\mathit{BiggsPrints}, \neg \mathit{MoneyInBag}\}, 0.7)$ is an outcome, and $([\neg \mathit{MoneyInBag}]@1 \wedge [\mathit{Charge}]@2)$ is an i-formula.

The probabilistic effects of actions are described in EPEC with *c-propositions* ('c' for 'causes'). To reflect the 70% chance that Biggs will leave his fingerprints if he steals from the bag, our example domain includes two of these:

$$\mathit{BiggsSteals} \wedge \neg \mathit{OtherSteals} \ \textbf{causes-one-of} \ \{(\{\neg \mathit{MoneyInBag}\}, 0.3), (\{\mathit{BiggsPrints}, \neg \mathit{MoneyInBag}\}, 0.7)\} \tag{HT1}$$

$$\mathit{OtherSteals} \wedge \neg \mathit{BiggsSteals} \ \textbf{causes-one-of} \ \{(\{\neg \mathit{MoneyInBag}\}, 1)\} \tag{HT2}$$

The general form of a c-proposition is "θ **causes-one-of** $\{O_1, \ldots, O_m\}$", where each O_j is an outcome, no two outcomes contain the same partial fluent state, the probabilities of the outcomes sum to 1, and the formula θ (which may also capture preconditions) classically entails at least one positive action literal. Note that outcomes are considered to be mutually exclusive, i.e. per action occurrence, one and only one outcome can come into effect. Here is the definition of a c-proposition taken from [15]:

Definition 2.7 *(c-proposition).* A *c-proposition* has the form

$$\theta \ \textbf{causes-one-of} \ \{O_1, O_2, \ldots, O_m\} \tag{1}$$

where, for $i = 1, \ldots, m$, $O_i \in \mathcal{O}$, $\chi(O_i) \neq \chi(O_j)$ when $i \neq j$, θ is a formula such that $\theta \models (A = \mathit{true})$ for at least one $A \in \mathcal{A}$, and $\pi(\{O_1, \ldots, O_m\}) = 1$. For a c-proposition $\underline{\mathbf{c}}$ of the form (1), the formula $\mathit{body}(\underline{\mathbf{c}}) = \theta$ and the set $\mathit{head}(\underline{\mathbf{c}}) = \{O_1, \ldots, O_m\}$ are the *body* and *head* of $\underline{\mathbf{c}}$, respectively. Outcome O_i is often omitted from $\mathit{head}(\underline{\mathbf{c}})$ if $\chi(O_i) = \emptyset$ (leaving $\pi(O_i)$ implicit since $\pi(\{O_1, \ldots, O_m\}) = 1$). □

In our framework, certain agent actions alter the agent's knowledge state instead of (or in addition to) directly impacting the environment. We refer to these as *sensing actions*. Their effect on the agent's knowledge is expressed using *s-propositions* ('s' for 'senses'). To illustrate, in our running example the epistemic effects of our agent's sensing actions *DustForPrints* and *DoIdParade* are modelled with the following s-propositions:

[2] The full EPEC model of this example is available to load and view on the EPEC web interface at https://www.ucl.ac.uk/infostudies/epec/translator.html.

DustForPrints **senses** *BiggsPrints* **with-accuracies** $\begin{pmatrix} 0.999 & 0.001 \\ 0.05 & 0.95 \end{pmatrix}$ (HT3)

DoIdParade **senses** *BiggsIsThief* **with-accuracies** $\begin{pmatrix} 0.9 & 0.1 \\ 0.15 & 0.85 \end{pmatrix}$ (HT4)

The matrices in (HT3) and (HT4) reflect the extent to which the sensing is imperfect. The leading diagonals give the probabilities of correctly sensing the values *false* and *true*, and the remaining values give the probabilities of false positives (top row) and false negatives. The general form of s-propositions is "θ **senses** X **with-accuracies** $\mathbf{M}$", where formula θ classically entails at least one positive action literal, $\mathbf{M}$ is an $m \times m$ matrix of probabilities whose rows add to 1, and where $X \in \mathcal{F} \cup \mathcal{A}_e$ takes m possible values in $vals(X)$. For non-Boolean fluents, the row and column order of $\mathbf{M}$ is specified by a *v-proposition*, the general form of which is "X **takes-values** $\langle V_1, \ldots, V_m \rangle$". (When no v-proposition is given, a fluent is taken to be Boolean with value order $\langle false, true \rangle$). Here are the definitions of v- and s-propositions from [15]:

Definition 2.8 *(v-proposition).* A *v-proposition* has the form

$$F \textbf{ takes-values } \langle V_1, \ldots, V_m \rangle \tag{2}$$

where $m \geq 1$ and $\langle V_1, \ldots, V_m \rangle = vals(F)$. □

Definition 2.9 *(s-proposition).* Let $X \in \mathcal{F} \cup \mathcal{A}_e$ and $vals(X) = \langle V_1, \ldots, V_m \rangle$. An *s-proposition* has the form

$$\theta \textbf{ senses } X \textbf{ with-accuracies } \mathbf{M} \tag{3}$$

where $\theta \models (A = true)$ for some $A \in \mathcal{A}_a$, and $\mathbf{M}$ is an $m \times m$ matrix with all elements in $[0, 1]$. For an s-proposition $\underline{\mathbf{s}}$ of the form (3), θ is called the *body of* $\underline{\mathbf{s}}$, or *body*($\underline{\mathbf{s}}$), and X is called the *object of* $\underline{\mathbf{s}}$, or *object*($\underline{\mathbf{s}}$). The pair (θ, X) is called the *signature of* $\underline{\mathbf{s}}$ and denoted by *sign*($\underline{\mathbf{s}}$). The element $\mathbf{M}_{i,j}$ in $\mathbf{M}$ represents the probability that, given that V_i is the actual value of X when θ occurs, the value V_j is sensed. Hence $\mathbf{M}$ is subject to the condition that each row adds to 1:

$$\forall 1 \leq i \leq m, \quad \sum_{j=1}^{m} \mathbf{M}_{i,j} = 1 \tag{4}$$

The s-proposition "θ **senses** X **with-accuracies** $\mathbf{I}_M$" (where $\mathbf{I}_M$ is the $m \times m$ identity matrix, representing perfect sensing) is sometimes abbreviated to "θ **senses** X". □

In every EPEC domain the agent's initial beliefs are modelled with a unique *i-proposition* ('i' for 'initial', not to be confused with 'i' in 'i-formulas' which stands for 'instant'). Recall that our detective is 90% sure that money has actually been stolen, in which case any one of the 962 occupants other than Ms Prigg are, a priori, equally likely to have stolen the money. In the case of the hotel theft scenario, the following i-proposition captures these initial assumptions:

initially-one-of { (HT5)

$(\{MoneyInBag, \neg BiggsPrints, \neg BiggsIsThief, OtherIsThief\}, 0.9 \times \frac{961}{962})$,

$(\{\neg MoneyInBag, \neg BiggsPrints, \neg BiggsIsThief, \neg OtherIsThief\}, 0.1)$,

$(\{MoneyInBag, \neg BiggsPrints, BiggsIsThief, \neg OtherIsThief\}, 0.9 \times \frac{1}{962})$ }

The general form of an i-proposition is "**initially-one-of** $\{O_1, O_2, \ldots, O_n\}$", where each outcome O_j describes a unique (complete) fluent state and (as for c-propositions) the probabilities of the outcomes sum to 1. The formal definition from [15] is:

Definition 2.10 *(i-proposition).* An *i-proposition* has the form

$$\textbf{initially-one-of } \{O_1, O_2, \ldots, O_m\} \tag{5}$$

where, for $i = 1, \ldots, m$, $O_i \in \mathcal{O}$, $\pi(\{O_1, \ldots, O_m\}) = 1$, $\chi(O_i) \in \tilde{\mathcal{S}}$, and $\chi(O_i) \neq \chi(O_j)$ when $i \neq j$ (i.e. $\chi(O_i)$ and $\chi(O_j)$ are mutually exclusive fluent states). □

Action occurrences (or events) in EPEC domains are of two kinds. Those in the environment and not under the control of the agent are represented by *o-propositions* ('o' for 'occurs'), and those executed by the agent by *p-propositions* ('p' for 'performed'). In the hotel example, the probable theft and dusting for fingerprints have already occurred, whereas the identity parade and charging of Biggs are (provisionally) planned for the future, so for simplicity we choose the set of instants $\mathcal{I} = \{-2, -1, 0, 1, 2, 3\}$, where 0 is the time at which the story is being narrated. Note that this instant 0 is different from the minimum instant (the "$\bar{0}$" in Definition 2.1) of the domain, which in this case is -2. The o- and p-propositions for this domain are:

BiggsSteals **occurs-at** -2 **if-holds** *BiggsIsThief* (HT6)

OtherSteals **occurs-at** -2 **if-holds** *OtherIsThief* (HT7)

DustForPrints **performed-at** -1 (HT8)

DoIdParade **performed-at** 1 **if-believes** $(BiggsIsThief, (0.33, 1])$ (HT9)

Charge **performed-at** 2 **if-believes** $(BiggsIsThief, (0.8, 1])$ (HT10)

As can be seen from (HT6)–(HT10), o-propositions are (optionally) conditioned on what actually holds at the instant in question, whereas p-propositions are optionally conditional on the strength of a belief falling within some specified range. Although not illustrated in this example, both p-propositions and o-propositions can also model probabilistic occurrences. The general form of a p-proposition is "A **performed-at** I **with-prob** P^+ **if-believes** $(\theta, \bar{P})$", and that of an o-proposition is "A' **occurs-at** I **with-prob** P^+ **if-holds** θ", for some $A \in \mathcal{A}_a$, $A' \in \mathcal{A}_e$, $I \in \mathcal{I}$, $P^+ \in (0, 1]$, formula θ and interval $\bar{P}$ with endpoints in $[0, 1]$. The formal definitions from [15] are:

Definition 2.11 *(o-proposition).* An *o-proposition* has the form

$$A \textbf{ occurs-at } I \textbf{ with-prob } P^+ \textbf{ if-holds } \theta \tag{6}$$

for some action $A \in \mathcal{A}_e$, instant I, $P^+ \in (0, 1]$ and fluent formula θ. For an o-proposition $\underline{\mathbf{o}}$ of the form (6), θ is called the *body of* $\underline{\mathbf{o}}$ or *body*($\underline{\mathbf{o}}$), and $\underline{\mathbf{o}}$ is said to *have instant* I and *principal action* A. If $P^+ = 1$ then the " **with-prob** P^+" part of the proposition may be omitted, and if θ is $\top$ then the " **if-holds** θ" part may be omitted. □

Definition 2.12 *(p-proposition).* A *p-proposition* has the form

$$A \textbf{ performed-at } I \textbf{ with-prob } P^+ \textbf{ if-believes } (\theta, \bar{P}) \tag{7}$$

for some action $A \in \mathcal{A}_a$, instant I, $P^+ \in (0, 1]$, fluent formula θ, and (open, half-open or closed) interval $\bar{P}$ with endpoints in $[0, 1]$ (i.e. a probability range). For a p-proposition $\underline{\mathbf{p}}$ of the form (7), $(\theta, \bar{P})$ is called the *body of* $\underline{\mathbf{p}}$ or *body*($\underline{\mathbf{p}}$), and $\underline{\mathbf{p}}$ is said to *have instant* I and *principal action* A. If $P^+ = 1$ then the " **with-prob** P^+" part of the proposition may be omitted, and if θ is $\top$ and $1 \in \bar{P}$ then the " **if-believes** $(\theta, \bar{P})$" part may be omitted. □

An EPEC domain description is a finite set of c-, s-, v-, i-, o- and p-propositions containing exactly one i-proposition. We will denote the domain description (HT1)–(HT10) by $\mathcal{D}_h$. The following definitions, both from [15], impose additional syntactic constraints to ensure that no two c-propositions are activated simultaneously, and that the probabilities included in different o- and p-propositions referring to the same action and instant are not contradictory:

Definition 2.13 *(Compatibility of Formulas).* Given a partial state X and a formula θ, we sometimes write $X \vDash \theta$ to indicate that $\bigwedge_{L \in X} L \vDash \theta$. Two formulas θ_1 and θ_2 are *compatible* if there is a state S such that $S \vDash \theta_1 \wedge \theta_2$, and *incompatible* otherwise. □

Definition 2.14 *(Domain Description).* A *domain description* is a finite set $\mathcal{D}$ of v-propositions, c-propositions, p-propositions, o-propositions, i-propositions and s-propositions such that:

(i) $\mathcal{D}$ contains exactly one v-proposition for each $F \in \mathcal{F}$ [see Definition 2.8],
(ii) $\mathcal{D}$ contains exactly one i-proposition [see Definition 2.10],
(iii) for any two distinct c-propositions in $\mathcal{D}$ with bodies θ_1 and θ_2 [see Definition 2.7], θ_1 and θ_2 are incompatible [see Definition 2.13],
(iv) for any given $A \in \mathcal{A}_e$ and $I \in \mathcal{I}$, if $\mathcal{D}$ contains a pair of o-propositions "A **occurs-at** I **with-prob** P_1^+ **if-holds** θ_1" and "A **occurs-at** I **with-prob** P_2^+ **if-holds** θ_2", then θ_1 and θ_2 are incompatible [see Definitions 2.13, 2.11],
(v) for any given $A \in \mathcal{A}_a$ and $I \in \mathcal{I}$, if $\mathcal{D}$ contains a pair of p-propositions "A **performed-at** I **with-prob** P_1^+ **if-believes** $(\theta_1, \bar{P}_1)$" and "A **performed-at** I **with-prob** P_2^+ **if-believes** $(\theta_2, \bar{P}_2)$" then $\theta_1 = \theta_2$ and $\bar{P}_1 \cap \bar{P}_2 = \emptyset$ [see Definition 2.12],
(vi) no two s-propositions in $\mathcal{D}$ have the same signature [see Definition 2.9]. □

In the remainder of the paper, since by Definition 2.14 each s-proposition within a domain description $\mathcal{D}$ has a unique signature (θ, X), we will sometimes refer to its accuracy matrix as $\mathbf{M}_D(\theta, X)$.

As an EPEC-based agent moves through time in the domain described by the domain description, it executes a particular subset of the p-propositions that is compatible both with its current beliefs and with the probabilities of performance (P^+ in the p-proposition template above), and receives particular sensory inputs in accordance with s-propositions activated via those p-propositions.

$\mathcal{D}_h$ |⊨ **at** 3 **believes** [*BiggsIsThief*]@−2 **with-probs** (E1)
{ (⟨ {((*DustForPrints*, *BiggsPrints*), *false*)}@−1 ⟩, 0.9984, 0.0003),
(⟨ {((*DustForPrints*, *BiggsPrints*), *true*)}@−1,
{((*DoIdParade*, *BiggsIsThief*), *false*)}@1 ⟩, 0.001, 0.0908),
(⟨ {((*DustForPrints*, *BiggsPrints*), *true*)}@−1,
{((*DoIdParade*, *BiggsIsThief*), *true*)}@1, {*Charge*}@2 ⟩,
0.0006, 0.8359) }

Fig. 1. An unconditional EPEC entailment.

($\mathcal{D}_h$ | $\mathcal{R}_h$) |⊨ **at** 3 **believes** [*BiggsIsThief*]@−2 **with-probs** (E2)
{ (⟨ {((*DustForPrints*, *BiggsPrints*), *true*)}@−1,
{((*DoIdParade*, *BiggsIsThief*), *false*)}@1 ⟩, 0.619, 0.0908),
(⟨ {((*DustForPrints*, *BiggsPrints*), *true*)}@−1,
{((*DoIdParade*, *BiggsIsThief*), *true*)}@1, {*Charge*}@2 ⟩,
0.381, 0.8359) }

Fig. 2. A conditional EPEC entailment.

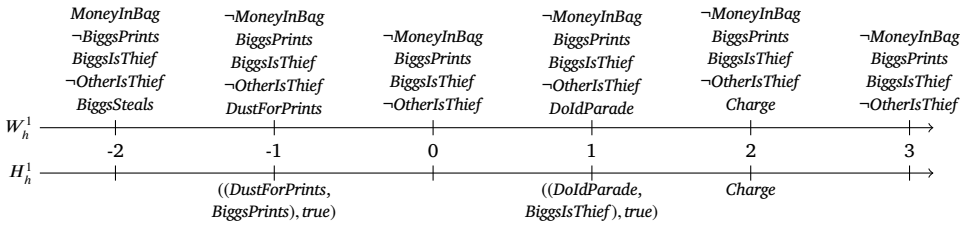

Fig. 3. A well-behaved h-world for $\mathcal{D}_h$.

2.2. An informal overview of EPEC's semantics

To aid the reader's intuition, in this section we informally summarise the semantics of EPEC, before giving the semantics in full technical detail in Section 2.3.

EPEC's semantics defines an entailment relation |⊨ between an EPEC domain description and a class of *b-propositions*. A b-proposition declares the agent's degree of belief in an i-formula at a specific instant with respect to each of the possible sensing and acting histories that the agent might have up to that instant, along with the probability of actually experiencing each possible history.

For example, $\mathcal{D}_h$ (which describes the hotel theft scenario but without the knowledge that dusting for prints did in fact give a positive result for Biggs' prints) gives the entailment (E1) in Fig. 1. This contains three possible sensing and acting histories. Entailment (E1) says that, initially and in the absence of test results, Det. Smart is aware that (i) there is a 99.84% chance that the result will be negative and that if this turns out to be the case she will (at future instant 3) have only a 0.03% belief that Biggs is the thief. (ii) There is a 0.1% chance that the print dusting result will be positive but the identity parade will be negative, in which case she will have a 9.08% belief that Biggs is the thief. Finally (iii) there is a 0.06% chance that both the print dusting and the identity parade will be positive in which case she will have an 83.59% belief that Biggs is the thief.

At the time of narration of the hotel theft scenario, Det. Smart has already received a positive print result for Biggs, and this is represented in the *activity report* $\mathcal{R}_h$. An activity report is a complete record of the agent's experience, up to the agent's current position in time. In the example scenario, Det. Smart's activity report consists of the following two *r-propositions*:

report *DustForPrints* **performed-at** −1 (AR1)

report *BiggsPrints* **sensed-with** {*DustForPrints*} **at** −1 (AR2)

where the two r-propositions (AR1) and (AR2) state respectively that dusting for prints and the associated sensing of Mr. Bigg's fingerprints occur at instant −1.

The activity report $\mathcal{R}_h$ = {(AR1), (AR2)} is not part of the domain description $\mathcal{D}_h$, but can be used to appropriately adapt entailments from it. The entailment (E1) can be *conditioned* on $\mathcal{R}_h$, removing the first (initially 99.84% probable) history that is incompatible with this evidence. This re-normalises the probabilities of the remaining histories appropriately, giving rise to the conditional entailment (E2) in Fig. 2. (E2) says, for example, that in the light of the positive fingerprint result for Biggs (recorded in (AR2) ∈ $\mathcal{R}_h$), Det. Smart knows that there is now a 38.1% chance that at instant 3 she will have an 83.59% belief that Biggs is the thief.

The semantic structure that underlies entailments from an EPEC domain description $\mathcal{D}$ is a probability distribution M_D over *h-worlds*. An h-world is a description of a possible state of the world, with M_D being a probability distribution over such states. It pairs together a possible sensing and acting experiential sequence of the agent (a "history") with a possible unfolding of the agent's actual

environment (a "world"). An h-world is *well-behaved* if these two components are compatible according to a number of intuitive criteria, and only well-behaved h-worlds are assigned a non-zero probability, calculated by appropriate arithmetic combinations of the probabilities appearing in the individual propositions of $\mathcal{D}$. For example, the pair (W_h^1, H_h^1) illustrated in Fig. 3 is one of the twelve well-behaved h-worlds for $\mathcal{D}_h$. (W_h^1, H_h^1) is well-behaved because the fluent values in W_h^1 persist appropriately along the timeline unless a change is caused by an action occurrence, the changes between instants -2 and -1 are justified by (HT1) and (HT6), the action occurrences in W_h^1 are consistent with (HT6)–(HT10), and those performed by the agent are identical in both W_h^1 and H_h^1. Moreover, and although this cannot be ascertained by consideration of the single h-world (W_h^1, H_h^1) in isolation, the agent's strengths of belief in *BiggsIsThief* at instants 1 and 2 are such that the belief-conditioned action performances specified in (HT9) and (HT10) respectively are both triggered. These strengths of belief are calculated as *conditional probabilities* – the probabilities that the agent is in an h-world where *BiggsIsThief* = *true* given that the agent is in an h-world whose experiential history is identical to H_h^1 up to the instant of action performance in question.

The overall probability assigned to an h-world by $M_{\mathcal{D}}$ is the product of the probabilities of each change of state along its timeline given the previous state. To help formulate these products, EPEC's semantics also employs the notion of an *h-trace*. An h-trace is similar to an h-world but contains additional information giving the causal justification for the particular changes in fluent values along its timeline. This additional information is in the form of a series of outcome choices, one from each of the c-propositions activated along the timeline. An h-world may therefore be associated with more than one h-trace, but each h-trace gives rise to a unique h-world, and hence the probability assigned to an h-world is the sum of the probabilities of its associated h-traces. Since h-worlds can be built from them, h-traces are the elementary building blocks of EPEC, and they are therefore at the core of the implementation described in Section 3 below.

For each well-formed EPEC domain, there is a unique set of h-traces that results in a consistent set of well-behaved h-worlds, each adhering to the criteria described above (for proof of this see [15]). This set of h-traces is defined using the fixed-point notion of an *epistemic reduct*. The epistemic reduct is a unique simplified version of the domain description in which a subset of the p-propositions has been 'reduced' into equivalent o-propositions, and the remainder of the p-propositions have been discarded. In this epistemic reduct, the newly introduced o-propositions are exactly those for which the belief conditions are satisfied in the original p-propositions from which they were formed. From a computational perspective, the epistemic reduct can be regarded as a 'generate then test' procedure which generates then tests a series of *candidate reducts*, constructed from an increasing number of candidate h-traces, until the epistemic reduct is identified. This procedure is reflected in the ASP code in Section 3.2.2 below.

Once the h-traces that form the epistemic reduct have been identified (within a unique single answer set), the answers to b-proposition queries such as those in Fig. 2 can be straightforwardly generated (via post-processing of the answer set) by grouping the h-traces with respect to their sensing histories and, for each group, calculating a normalised sum of the probabilities of those h-traces in which the belief of the b-proposition holds.

2.3. Formal details of EPEC's semantics

We now give the full formal details of EPEC's semantics as first presented in [15]. Definitions 2.15 to 2.44 that follow are unchanged from [15] although they are numbered differently.

2.3.1. Probability functions

The semantics of EPEC domain descriptions is expressed through *probability functions*. Thus, we begin by defining what we mean by a probability function in this context. Such a function assigns probabilities to a set of propositional formulas consistently. Our definition is a slight generalization of Paris's [28, Chapter 2, p.10]. Unlike Paris's definition, which necessitates the entailment relation to be in accordance with all interpretations of the language (i.e., classical propositional entailment), ours only requires it to align with a specific non-empty set of interpretations.

Definition 2.15 *(Probability Function, Conditional Probability for Formulas).* Let $\models$ be the entailment relation defined in the standard way with respect to some non-empty set of interpretations of a propositional language L, and let $\mathbf{F}_{set}$ be a set of propositional formulas of L closed under the propositional operators. A *probability function over* $\mathbf{F}_{set}$ *w.r.t.* $\models$ is a function $p : \mathbf{F}_{set} \mapsto [0, 1]$ such that for all $\varphi, \psi \in \mathbf{F}_{set}$:

1. if $\models \varphi$, then $p(\varphi) = 1$, and
2. if $\models \neg(\psi \wedge \phi)$ then $p(\varphi \vee \psi) = p(\varphi) + p(\psi)$.

For $p(\psi) \neq 0$, the associated *conditional probability* of φ given ψ is defined as

$$p(\varphi \mid \psi) = \frac{p(\varphi \wedge \psi)}{p(\psi)} \quad \square \tag{8}$$

2.3.2. Formal semantics of non-epistemic domains

We next give a semantics to *non-epistemic* domain descriptions, i.e. those that contain no information about sensing actions or belief-conditioned action occurrences:

Definition 2.16 *(Ne-domain Description).* An *ne-domain description* (*non-epistemic domain description*) is a domain description that contains no s- or p-propositions. □

In the remainer of Section 2.3, $\mathcal{L}$, $\mathcal{D}$ and $\mathcal{N}$ signify an arbitrary domain language, domain description and ne-domain description respectively.

For ne-domain descriptions, the key semantic structure is a *world*. Worlds are effectively timelines labelled with all fluent values and action occurrences/non-occurrences at each instant, i.e. mappings from instants to states:

Definition 2.17 *(World).* A *world* is a function $W : \mathcal{I} \to S$. The set of all worlds is denoted by $\mathcal{W}$. □

Definitions 2.18 to 2.25 which follow allow us to distinguish between worlds that are compatible with the ne-domain description in question – we call these *well-behaved* worlds – and those that are not. We start with a notion of *satisfaction* of an i-formula:

Definition 2.18 *(Satisfaction of an i-formula, Logical Consequence for i-formulas).* Given a world W and a literal L, W *satisfies an i-formula* $[L]@I$, written $W \mid\!\models [L]@I$, iff $L \in W(I)$. Otherwise, $W \not\mid\!\models [L]@I$. The definition of $\mid\!\models$ is recursively extended for arbitrary i-formulas as follows: if φ and ψ are i-formulas, $W \mid\!\models \varphi \wedge \psi$ iff $W \mid\!\models \varphi$ and $W \mid\!\models \psi$, and $W \mid\!\models \neg\varphi$ iff $W \not\mid\!\models \varphi$. The i-formulas $\varphi \vee \psi$ and $\varphi \to \psi$ are interpreted as shorthand for $\neg(\neg\varphi \wedge \neg\psi)$ and $\neg(\varphi \wedge \neg\psi)$ respectively. Given a (possibly empty) set Δ of i-formulas, $W \mid\!\models \Delta$ iff $W \mid\!\models \psi$ for all $\psi \in \Delta$. Given an i-formula φ and a set Δ of i-formulas $\Delta \mid\!\models \varphi$ if for all $W \in \mathcal{W}$ such that $W \mid\!\models \Delta$, $W \mid\!\models \varphi$ also holds. For two i-formulas φ and ψ, $\psi \mid\!\models \varphi$ is shorthand for $\{\psi\} \mid\!\models \varphi$, and $\mid\!\models \varphi$ is shorthand for $\emptyset \mid\!\models \varphi$. □

The first criterion that a world must satisfy to be well-behaved with respect to an ne-domain description $\mathcal{N}$ is that the action occurrences it identifies along its timeline exactly match those represented as o-propositions in $\mathcal{N}$:

Definition 2.19 *(Closed World Assumption for Actions).* A world W satisfies the *closed world assumption for actions* (*CWAA*) *w.r.t. an ne-domain description* $\mathcal{N}$ if for all $A \in \mathcal{A}$ and $I \in \mathcal{I}$:

(i) if $W \mid\!\models [A]@I$ then there exists some $P^+ \in (0, 1]$ and fluent formula θ such that $W \mid\!\models [\theta]@I$ and "A **occurs-at** I **with-prob** P^+ **if-holds** θ" is in $\mathcal{N}$, and
(ii) for any θ such that $W \mid\!\models [\theta]@I$ and the o-proposition "A **occurs-at** I **with-prob** 1 **if-holds** θ" is in $\mathcal{N}$, $W \mid\!\models [A]@I$. □

The second criterion that a world must satisfy to be well-behaved with respect to an ne-domain description $\mathcal{N}$ is that it is compatible with one of the alternative initial conditions identified in the domain description's i-proposition. (Note that the next few definitions apply to an arbitrary domain description $\mathcal{D}$, and thus also to any ne-domain description $\mathcal{N}$.)

Definition 2.20 *(Initial Choice and Initial Consistency).* Let $\mathcal{D}$ be a domain description with (unique) i-proposition "**initially-one-of** $\{O_1, O_2, \ldots, O_m\}$". Each $O_1, O_2, \ldots, O_m$ is called an *initial choice of* $\mathcal{D}$. A world W is said to *satisfy the initial condition of* $\mathcal{D}$ if there exists an initial choice O_i of $\mathcal{D}$ such that $W(\bar{0}) \restriction \mathcal{F} = \chi(O_i)$ [See Definitions 2.1, 2.4, 2.5]. In this case it is said that W *and* O_i *are initially consistent with each other w.r.t.* $\mathcal{D}$. □

The last criterion that a world must satisfy to be well-behaved is that the changes in fluent values along its time-line are explainable in terms of the domain description's c-propositions. To formalise this, we need to be able to identify which c-propositions are activated at which time-points in that world:

Definition 2.21 *(Cause Occurrence).* Let θ be the body of a c-proposition $\underline{\mathbf{c}}$ in a domain description $\mathcal{D}$ and $I \in \mathcal{I}$. If $W \mid\!\models [\theta]@I$ then it is said that *a cause occurs at instant* I *in* W *w.r.t.* $\mathcal{D}$, and that $\underline{\mathbf{c}}$ *is activated at* I *in* W *w.r.t.* $\mathcal{D}$. The set $occ_D(W)$ is the set $\{I \in \mathcal{I} \mid \text{a cause occurs at } I \text{ in } W\}$. The function $cprop_D$ with domain $\{(W, I) \mid W \in \mathcal{W}, I \in occ_D(W)\}$ is defined as $cprop_D(W, I) = \underline{\mathbf{c}}$ where $\underline{\mathbf{c}}$ is the (unique) c-proposition activated at I in W. □

We next define an *effect choice* for a given world. This selects one particular outcome from each c-proposition at each point that it is activated:

Definition 2.22 *(Effect Choice).* Let W be a world and $\mathcal{D}$ a domain description. An *effect choice for* W *w.r.t.* $\mathcal{D}$ is a function $ec : occ_D(W) \to \mathcal{O}$ such that for all instants $I \in occ_D(W)$, $ec(I) \in head(cprop_D(W, I))$. □

To describe the expected effect of one of a c-proposition's outcomes on the state in which it is activated, we define the notion of a *fluent state update*:

Definition 2.23 *(Fluent State Update).* Given a fluent state $\tilde{S}$ and a partial fluent state $\tilde{X}$, the *update of* $\tilde{S}$ *w.r.t.* $\tilde{X}$, written $\tilde{S} \oplus \tilde{X}$, is the fluent state $(\tilde{S} \ominus \tilde{X}) \cup \tilde{X}$, where $\tilde{S} \ominus \tilde{X}$ is the partial fluent state formed by removing all fluent literals from $\tilde{S}$ of the form $F = V$ for some F and V' such that $F = V \in \tilde{X}$. The operator $\oplus$ is left-associative, so e.g. $\tilde{S} \oplus \tilde{X} \oplus \tilde{X}'$ is understood as $((\tilde{S} \oplus \tilde{X}) \oplus \tilde{X}')$. □

We can now state our final criterion for a world to be well-behaved – it must satisfy the *justified change condition*. Definition 2.24 below states that changes in fluents' values along a world's time-line occur only immediately after instants where a c-proposition is activated, and that all the fluents' new values must appear in a single outcome of that c-proposition. The definition thus encapsulates a solution to the frame problem, as it ensures that fluents not explicitly affected by the activation of a c-proposition have a default persistence. Like all EPEC's definitions, Definition 2.24 is stated in a manner that allows it to be applied to continuous as well as discrete time-lines.

Definition 2.24 *(Justified Change).* A world W satisfies the *justified change condition w.r.t.* $\mathcal{D}$ if and only if there exists an effect choice ec w.r.t. $\mathcal{D}$ such that for all instants I and I' with $I < I'$, ec maps the instants in $occ_{\mathcal{D}}(W) \cap [I, I') = \{I_1, \ldots, I_n\}$ to $O_1, O_2, \ldots, O_n$ respectively, where $I_1, \ldots, I_n$ are ordered w.r.t. $\leq$, and

$$W(I')\restriction_{\mathcal{F}} = (W(I)\restriction_{\mathcal{F}}) \oplus \chi(O_1) \oplus \chi(O_2) \oplus \cdots \oplus \chi(O_n) \tag{9}$$

If a world W satisfies the justified change condition for some effect choice ec, W and ec are said to be *consistent* with each other *w.r.t.* $\mathcal{D}$. □

Definition 2.25 summarises the conditions for a world to be well-behaved:

Definition 2.25 *(Well-behaved World).* A world is *well-behaved w.r.t. an ne-domain description* $\mathcal{N}$ if it satisfies the closed world assumption for actions w.r.t. $\mathcal{N}$ [see Definition 2.19], the initial condition of $\mathcal{N}$ [see Definition 2.20] and the justified change condition w.r.t. $\mathcal{N}$ [see Definition 2.24]. The set of well-behaved worlds w.r.t. $\mathcal{N}$ is denoted by $\mathcal{W}_{\mathcal{N}}$. □

We can use the notion of a well-behaved world to adapt the notion of entailment (symbolised by $|\!\models$) given in Definition 2.18 to a specific ne-domain $\mathcal{N}$. *$\mathcal{N}$-entailment* is symbolised by $|\!\models_{\mathcal{N}}$ in the following definition:

Definition 2.26 *($\mathcal{N}$-entailment).* Given an i-formula φ, a set Δ of i-formulas and an ne-domain description $\mathcal{N}$, Δ *$\mathcal{N}$-entails* φ, written $\Delta \,|\!\models_{\mathcal{N}} \varphi$, if for all well-behaved worlds $W \in \mathcal{W}_{\mathcal{N}}$ such that $W \,|\!\models \Delta$, $W \,|\!\models \varphi$ also holds [see Definition 2.18]. For two i-formulas ψ and φ, $\psi \,|\!\models_{\mathcal{N}} \varphi$ is shorthand for $\{\psi\} \,|\!\models_{\mathcal{N}} \varphi$, and $|\!\models_{\mathcal{N}} \varphi$ is shorthand for $\emptyset \,|\!\models_{\mathcal{N}} \varphi$. □

To complete the semantics of an ne-domain description, it remains to define a probability distribution over the set of well-behaved worlds that properly accounts for the various probabilities embedded in the propositions. We begin with the notion of a *trace* of a well-behaved world, which is an initial choice coupled with an effect choice that can account for the values of fluents along the world's time-line. We can straightforwardly evaluate the probability of a given trace, conditional on the action occurrences in the associated world actually taking place, by taking the product of the probabilities of the individual outcomes selected by the trace:

Definition 2.27 *(Trace).* Let W be a well-behaved world w.r.t. $\mathcal{N}$. A *trace of W w.r.t.* $\mathcal{N}$ is a pair $\langle ic, ec \rangle$ where ic is an initial choice [see Definition 2.20] consistent with W and ec is an effect choice [see Definition 2.22] consistent with W. In this case $\langle ic, ec \rangle$ is also a *trace of* $\mathcal{N}$. The set *traces*$(W, \mathcal{N})$ is the set of all traces of W w.r.t. $\mathcal{N}$. For an arbitrary trace $tr = \langle ic, ec \rangle$ of $\mathcal{N}$, we identify the (unique) corresponding well-behaved world as W_{tr}, and define the *evaluation of* tr, written $\epsilon(tr)$, as:

$$\epsilon(tr) = \pi(ic) \quad . \prod_{I \in occ_{\mathcal{D}}(W_{tr})} \pi(ec(I)) \tag{10}$$

(Where $\pi : \mathcal{O} \mapsto (0, 1]$ is as defined in Definition 2.5.) □

To calculate the probability that the exact sequence of action occurrences/non-occurrences entailed by a particular well-behaved world W actually takes place, we can take the product of the probabilities/complement probabilities indicated in the corresponding o-propositions. We call this product the *narrative evaluation* of $\mathcal{N}$ w.r.t. W:

Definition 2.28 *(Narrative Evaluation).* Given an o-proposition $\underline{\mathbf{o}}$ of the form "A **occurs-at** I **with-prob** P^+ **if-holds** θ" and a world W, the *narative evaluation of* $\underline{\mathbf{o}}$ *w.r.t.* W is defined as

$$\epsilon(\underline{\mathbf{o}}, W) = \begin{cases} 1 & \text{if } W \not|\!\models [\theta]@I \\ P^+ & \text{if } W \,|\!\models [\theta]@I \text{ and } W \,|\!\models [A]@I \\ 1 - P^+ & \text{if } W \,|\!\models [\theta]@I \text{ and } W \,|\!\models [\neg A]@I \end{cases} \tag{11}$$

For an ne-domain description $\mathcal{N}$ the above definition is extended to:

$$\epsilon(\mathcal{N}, W) = \prod_{\underline{\mathbf{o}} \in \mathcal{N}} \epsilon(\underline{\mathbf{o}}, W). \tag{12}$$

If $\mathcal{N}$ contains no o-propositions then $\epsilon(\mathcal{N}, W) = 1$. □

We now define our probability distribution, the *ne-model-function*, over the well-behaved worlds of an ne-domain description $\mathcal{N}$, symbolised as $M^{ne}_{\mathcal{N}}$, and prove that it is indeed a probability distribution in Proposition 2.1 that follows. The probability of a well-behaved world W is the product of the probability of its narrative component (action occurrences) being true and the probability of the particular chosen causal effects of those actions having taken place.

Definition 2.29 *(Ne-model-function).* The *ne-model-function* of an ne-domain description $\mathcal{N}$ is the function $M^{ne}_{\mathcal{N}} : \mathcal{W}_{\mathcal{N}} \mapsto [0,1]$ defined for each well-behaved world W as:

$$M^{ne}_{\mathcal{N}}(W) = \epsilon(\mathcal{N}, W) \cdot \sum_{tr \in traces(W,\mathcal{N})} \epsilon(tr) \tag{13}$$

The ne-model-function $M^{ne}_{\mathcal{N}}$ is extended to a function $M^{ne}_{\mathcal{N}} : \Phi \mapsto [0,1]$ over i-formulas in the following way:

$$M^{ne}_{\mathcal{N}}(\varphi) = \sum_{W \mid\!\models \varphi} M^{ne}_{\mathcal{N}}(W) \tag{14}$$

and if ψ is such that $M^{ne}_{\mathcal{N}}(\psi) \neq 0$, then the function $M^{ne}_{\mathcal{N}}(\cdot \mid \psi) : \Phi \mapsto [0,1]$ is defined as:

$$M^{ne}_{\mathcal{N}}(\varphi \mid \psi) = \frac{M^{ne}_{\mathcal{N}}(\varphi \wedge \psi)}{M^{ne}_{\mathcal{N}}(\psi)} \quad \square \tag{15}$$

The following propositions, whose proofs can be found in [15], show that ne-models are in fact probability distributions over the set of worlds and the set of formulas:

Proposition 2.1. *Given an ne-domain description $\mathcal{N}$ [see Definition 2.16], the ne-model $M^{ne}_{\mathcal{N}} : \mathcal{W}_{\mathcal{N}} \mapsto [0,1]$ [see Definition 2.29] is a probability distribution over $\mathcal{W}_{\mathcal{N}}$.* $\square$

Proposition 2.2. *Given an ne-domain description $\mathcal{N}$, the ne-model $M^{ne}_{\mathcal{N}} : \Phi \mapsto [0,1]$ [see Definition 2.29] is a probability function over Φ w.r.t. $|\!\models_{\mathcal{N}}$ [see Definitions 2.15, 2.3, 2.26].* $\square$

Specific evaluations of $M^{ne}_{\mathcal{N}}(\varphi)$ and $M^{ne}_{\mathcal{N}}(\varphi \mid \psi)$ using equations (14) and (15) of Definition 2.29 can be written in action-language-style syntax using *h-propositions*. We write "$\mathcal{N} \mid\!\models \varphi$ **holds-with-prob** P^+" to mean that $M^{ne}_{\mathcal{N}}(\varphi) = P^+$, and we write $M^{ne}_{\mathcal{N}}(\varphi \mid \psi) = P^+$ as "$(\mathcal{N} \mid \psi) \mid\!\models \varphi$ **holds-with-prob** P^+".

2.3.3. Semantics of epistemic domains

We complete our specification of the semantics of EPEC by extending the definitions to cover full domain descriptions that may include p-propositions (describing belief-conditioned agent action occurrences) and s-propositions (describing the effects of sensing actions).

We represent the result of a sensing action as follows:

Definition 2.30 *(Sensing Outcome).* A *sensing outcome* is a pair of the form $((\theta, X), V)$ for some signature (θ, X) of an s-proposition [see Definition 2.9] and some value $V \in vals(X)$. $\square$

Sensing outcomes and performances of agent actions represent the information that potentially becomes available to the agent as time progresses. This information and is collected together in a *sensing and acting history*. Each sensing and acting history records a possible sequence of experiences and behaviour (inputs and outputs) that the agent might have as it progresses through the time-line. Note that our definitions reflect our assumption that the agent is aware of its own actions:

Definition 2.31 *(Sensing and Acting History, Indistinguishable Histories).* A *sensing and acting history* (or *history*) is a function H from $\mathcal{I}$ to the power set of sensing outcomes and agent actions. The set of all histories is denoted by $\mathcal{H}$. For a given instant I, two histories H and H' are *indistinguishable up to (and excluding) I* if $H(I') = H'(I')$ for all $I' < I$. For a given history H and instant I, the equivalence class of histories indistinguishable from H up to I is denoted by $[H]_{<I}$. $\square$

Equivalence classes of the form $[H]_{<I}$ represent agent experiences which are identical up to I, and will be used later to evaluate whether the agent will execute belief-conditioned actions at the instant I.

Whereas for ne-domain descriptions the key semantic structure is a world (see Definition 2.17), in the case of epistemic domains the key semantic structure is a world paired with a history. We call such a pairing an *h-world*:

Definition 2.32 *(h-world).* An *h-world* is a pair (W, H) for a world $W \in \mathcal{W}$ and a sensing history $H \in \mathcal{H}$. [See Definitions 2.17 and 2.31]. $\square$

EPEC's semantics identifies the set of h-worlds that contain a history and world pair that are compatible both with each other and with the domain description in question – these are called *well-behaved h-worlds* – and defines a probability distribution over this set. Some complexity arises because at all points in the narrative of any particular h-world where the agent makes decisions about performing belief-conditioned actions, the strengths of those beliefs (represented as probabilities) have to be evaluated by consideration of all h-worlds and their respective probabilities. However, as the strengths of those evaluated beliefs influence whether the belief-conditioned actions fire, they thus influence the probability distribution over h-worlds on which the belief was evaluated.

The next three definitions allow us to assign a probability to a history, and to an equivalence class of histories (indistinguishable up to a given instant), in both cases conditioned on a particular world. First, we define a *sensing occurrence* in a particular world, and the associated mapping $socc_{\mathcal{D}}$. For a given h-world, $socc_{\mathcal{D}}$ indicates which s-propositions (if any) are activated (occur) at each instant, and what both the true and the sensed values of the associated fluents are for these activations:

Definition 2.33 *(Sensing Occurrence).* Let $\mathcal{D}$ be a domain description, $\underline{\mathbf{s}}$ be an s-proposition in $\mathcal{D}$ with body θ, and $I \in \mathcal{I}$. If $W \mid\!\models [\theta]@I$ then it is said that *a sensing action occurs at instant I in W w.r.t. $\mathcal{D}$*, and that $\underline{\mathbf{s}}$ *is activated at I in W w.r.t. $\mathcal{D}$*. Let (W, H) be an h-world. For any instant $I \in \mathcal{I}$, $socc_{\mathcal{D}}((W, H), I)$ is defined as

$$\{((\theta, X), V, V') \mid \underline{\mathbf{s}} \text{ is an s-proposition in } \mathcal{D},\ sign(\underline{\mathbf{s}}) = (\theta, X),$$
$$W \mid\!\models [\theta \wedge X = V]@I,\ ((\theta, X), V') \in H(I)\} \quad \square$$

We can now define the following well-behavedness property for h-worlds. For an h-world to be well-behaved, it must satisfy the *closed world assumption for sensing and acting* (*CWSA*). This ensures that the history gives exactly one sensed value for each of the s-propositions activated in the world (condition (i)), no sensed values where no s-proposition is activated (condition (ii)), and also that the history properly describes the agent's awareness of its own actions in the world (condition (iii)):

Definition 2.34 *(CWSA).* Given a domain description $\mathcal{D}$, an h-world (W, H) satisfies the *closed world assumption for sensing and acting (CWSA) w.r.t. $\mathcal{D}$* if for each $I \in \mathcal{I}$:

(i) for each s-proposition $\underline{\mathbf{s}}$ activated at I in W, $(sign(\underline{\mathbf{s}}), V) \in H(I)$ for exactly one value V,
(ii) for each $((\theta, X), V') \in H(I)$ there is an s-proposition in $\mathcal{D}$ with signature (θ, X) activated at I in W, and
(iii) for each $A \in \mathcal{A}_a$, $A \in H(I)$ if and only if $W \mid\!\models [A = true]@I$. $\square$

We next address the issue of assigning an overall probability to an entire h-world (W, H). We start by assigning a probability to H conditioned on W, which we call an *evaluation* of H w.r.t. W. Definition 2.35 below says that the evaluation of H given W is the product of the accuracy matrix entries for each of the sensing results contained in H. To aid reading of this definition, recall (from Definition 2.9) that $\mathbf{M}(\theta, X)$ refers to the accuracy matrix in the s-proposition with unique signature (θ, X) (where θ is the condition of the s-proposition and X is the fluent being sensed). The element $\mathbf{M}(\theta, X)_{i,j}$ in $\mathbf{M}(\theta, X)$ represents the probability that, given that V_i is the actual value of X when θ occurs, the value V_j is sensed. Recall also (from Definition 2.33) that $socc_{\mathcal{D}}((W, H), I)$ gives a sensed value of a fluent according to H together with the actual value according to W, which in turn identify this specific element in $\mathbf{M}(\theta, X)$.

Definition 2.35 *(History Evaluation).* Let $\mathcal{D}$ be a domain-description, (W, H) an h-world, and $\mathcal{J}_H^W = \{I \mid I \in \mathcal{I}, socc_{\mathcal{D}}((W, H), I) \neq \emptyset\}$ (i.e. $\mathcal{J}_H^W$ is the set of instants at which some sensing occurs). For all $I \in \mathcal{J}_H^W$ let

$$\mathbf{Mprod}_H^W(I) = \prod_{((\theta,X),V_i,V_j) \in socc_{\mathcal{D}}((W,H),I)} \mathbf{M}(\theta, X)_{i,j}$$

where for each expression in this product V_i and V_j are the ith and jth elements of $vals(X)$ respectively. The *evaluation of H given W w.r.t. $\mathcal{D}$* is defined as:

$$\epsilon_{\mathcal{D}}(H \mid W) = \begin{cases} 0 & \text{if } (W, H) \text{ does not satisfy the CWSA} \\ 1 & \text{if } (W, H) \text{ satisfies CWSA and } \mathcal{J}_H^W = \emptyset \\ \prod_{I \in \mathcal{J}_H^W} \mathbf{Mprod}_H^W(I) & \text{otherwise} \end{cases}$$

For a class $[H]_{<I}$ of indistinguishable sensing histories up to I, $\epsilon_{\mathcal{D}}([H]_{<I} \mid W)$ denotes the sum:

$$\sum_{H' \in [H]_{<I}} \epsilon_{\mathcal{D}}(H' \mid W) \quad \square$$

Note that $\epsilon_{\mathcal{D}}(H \mid W)$ is equal to 1 when (W, H) satisfies the CWSA and there are no sensing occurrences, because there is no uncertainty that is coming from sensing noise. When there are sensing occurrences, the first (inner) product in the definition allows for concurrent (and potentially conflicting) sensing at any particular instant, whereas the second (outer) product allows for

different instants at which sensing occurs along the time-line. The use of products reflects the fact that we consider sensing acts to be probabilistically independent (no sensing act interferes with any other, even when they are concurrent).

Two key ideas underly the remaining definitions for EPEC's semantics. The first is the intuition that, from the point of view of an agent residing in world W after experiencing all the events in a particular history H, the domain description $\mathcal{D}$ can effectively be "reduced" to a corresponding ne-domain description. This is because the belief-conditioned decisions embedded in the p-propositions will have already been made, making those p-propositions no different in retrospect from o-propositions with either a *true* or a *false* condition, and the relevant probabilities from matrix entries in the s-propositions can be accounted for by the overall probability of the history H given the world W. This idea motivates the notion of a *reduct* in Definitions 2.38 and 2.39 below. The second idea is that, using the product rule $p(X,Y) = p(Y|X)p(X)$ for random variables X and Y from probability theory (notation here as in [11]), we can express a probability distribution over h-worlds in terms of a marginal probability $p(W)$ for each world and a conditional probability $p(H\,|\,W)$ for each history given a particular world. We already have two such candidate probability distributions, $M^{ne}_{\mathcal{N}}$ and $\epsilon_{\mathcal{D}}$, from Definitions 2.29 and 2.35 respectively. Definition 2.37 below utilises these to provide a general joint probability distribution called the *pre-model-function*, written $\tilde{M}^{\mathcal{N}}_{\mathcal{D}}$, with respect to any domain description $\mathcal{D}$ and ne-domain description $\mathcal{N}$. However, the subsequent focus will then be on ne-domain descriptions $\mathcal{N}$ that are reducts of $\mathcal{D}$ in the sense described informally above.

In order to re-express p-propositions as o-propositions in a reduct ne-domain description, we need to re-classify agent actions as environmental actions in the underlying domain language:

Definition 2.36 *(Ne-transform of a Domain Language).* The *ne-transform* of a domain language $\mathcal{L} = \langle \mathcal{F}, \mathcal{A}, \mathcal{A}_e, \mathcal{A}_a, \mathcal{V}, vals, \mathcal{I}, \leq, \bar{0}\rangle$, written $\mathcal{L}^{ne}$, is the domain language $\langle \mathcal{F}, \mathcal{A}, \mathcal{A}_e \cup \mathcal{A}_a, \emptyset, \mathcal{V}, vals, \mathcal{I}, \leq, \bar{0}\rangle$. (i.e. Agent actions are re-classified as environmental actions.) □

We now give the definition of a pre-model function $\tilde{M}^{\mathcal{N}}_{\mathcal{D}}$. Intuitively, this is the probability distribution over h-worlds induced by $\mathcal{D}$ and $\mathcal{N}$ which describes the likelihood for a given world W and history H of the agent receiving the sensory input in H and evaluating it according to the s-propositions in $\mathcal{D}$, but acting and causing change according to W and evaluating this according to $\mathcal{N}$. As indicated above, for generality this abstract definition is for an arbitrary $\mathcal{D}$ and $\mathcal{N}$. But, for a given $\mathcal{D}$, our objective in subsequent definitions is to identify the unique $\mathcal{N}$ (the "reduct") that mirrors the rational behaviour of the agent according to its full specification in $\mathcal{D}$. This provides a stepping-stone to enable the full epistemic semantics of EPEC to be couched in terms of its non-epistemic part. Note that including o-propositions in $\mathcal{N}$ that mirror some of the p-propositions in $\mathcal{D}$ is the key to identifying this unique $\mathcal{N}$.

Definition 2.37 *(Pre-model-function).* Given a domain description $\mathcal{D}$ written in language $\mathcal{L}$ and an ne-domain description $\mathcal{N}$ written in $\mathcal{L}^{ne}$, the *pre-model-function of $\mathcal{D}$ w.r.t. $\mathcal{N}$* is the function $\tilde{M}^{\mathcal{N}}_{\mathcal{D}} : \mathcal{W} \times \mathcal{H} \mapsto [0,1]$ (with $\mathcal{H}$ defined in $\mathcal{L}$) defined as:

$$\tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, H = \begin{cases} \epsilon_{\mathcal{D}}(H\,|\,W) \cdot M^{ne}_{\mathcal{N}}(W) & \text{if } W \in \mathcal{W}_{\mathcal{N}} \\ 0 & \text{otherwise} \end{cases}$$

The domain of $\tilde{M}^{\mathcal{N}}_{\mathcal{D}}$ is extended to equivalence classes of histories and to i-formulas in the usual way:

$$\tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, [H]_{<I} = \sum_{H' \in [H]_{<I}} \tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, H'$$

$$\tilde{M}^{\mathcal{N}}_{\mathcal{D}} \varphi, H = \sum_{W|\models\varphi} \tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, H \quad \text{and} \quad \tilde{M}^{\mathcal{N}}_{\mathcal{D}} \varphi, [H]_{<I} = \sum_{W|\models\varphi} \tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, [H]_{<I} \quad \square$$

The following proposition, proved in [15], establishes that $\tilde{M}^{\mathcal{N}}_{\mathcal{D}}$ is indeed a probability distribution over the set of h-worlds.

Proposition 2.3. *Let $\mathcal{D}$ and $\mathcal{N}$ be domain and ne-domain descriptions in languages $\mathcal{L}$ and $\mathcal{L}^{ne}$ respectively. The pre-model-function $\tilde{M}^{\mathcal{N}}_{\mathcal{D}}$ of $\mathcal{D}$ w.r.t. $\mathcal{N}$ is a probability distribution over $\mathcal{W} \times \mathcal{H}$, i.e. $\sum_{(W,H)\in\mathcal{W}\times\mathcal{H}} \tilde{M}^{\mathcal{N}}_{\mathcal{D}} W, H = 1$.* □

The next definition takes us a step closer to formalising the notion of a reduct of a domain description (discussed informally on page 12) by specifying how to either eliminate a p-proposition or convert it to an o-proposition with respect to a probability distribution over worlds:

Definition 2.38 *(Covers, Reduct of a p-proposition).* Let $\underline{\mathbf{p}}$ be the p-proposition "*A* **performed-at** I **with-prob** P^+ **if-believes** $(\theta, \bar{P})$" in the language $\mathcal{L}$, $\underline{\mathbf{o}}$ be the o-proposition "*A* **occurs-at** I **with-prob** P^+" in the language $\mathcal{L}^{ne}$, and d be a probability distribution over worlds. Then d *covers* $\underline{\mathbf{p}}$ if $\sum_{W|\models[\theta]@I} d(W) \in \bar{P}$. In this case $\underline{\mathbf{p}}$ *reduces to* the singleton set $\{\underline{\mathbf{o}}\}$ and $\{\underline{\mathbf{o}}\}$ *is the reduct of* $\underline{\mathbf{p}}$ w.r.t. d. Otherwise $\underline{\mathbf{p}}$ *reduces to* $\emptyset$ and $\emptyset$ *is the reduct of* $\underline{\mathbf{p}}$ w.r.t. d. □

Definition 2.39 below states that the reduct of a domain description $\mathcal{D}$ w.r.t. a history H is $\mathcal{D}$ itself but with all s-propositions removed, and each p-proposition replaced with its reduct. For each p-proposition, the probability distribution used in its reduction

is conditional on the probability of the history H up to the instant of the p-proposition (expressed as an equivalence class of indistinguishable histories). This is because it is the agent's sensing and acting history up to that instant that influences its belief in the condition of the p-proposition. Definition 2.39 is stated in general terms for any probability distribution d over the set of h-worlds, but in subsequent definitions the probability distribution used in the reduction will be the pre-model-function $\tilde{M}_{\mathcal{D}}^{\mathcal{N}}$ [see Definition 2.37]. (Note that if d gives a zero probability for a particular equivalence class of indistinguishable histories then any associated conditional probability is undefined, so, for mathematical completeness, in this case we use the unconditional marginal probability d over worlds to reduce the p-proposition, although histories with zero probability do not contribute to EPEC's semantics in subsequent definitions.)

Definition 2.39 *(Reduct of a Domain Description).* Let $\mathcal{D}$ be domain description, d be a probability distribution over $\mathcal{W} \times \mathcal{H}$ and H be a history. Then *the reduct of $\mathcal{D}$ w.r.t. d and H* is the ne-domain description obtained by first removing all the s-propositions and p-propositions from $\mathcal{D}$, and then for each removed p-proposition $\underline{\mathbf{p}}$ with instant I, conjoining the ne-domain description with the reduct of $\underline{\mathbf{p}}$ w.r.t. the probability distribution $d(\cdot \mid [H]_{<I})$ over $\mathcal{W}$, defined as follows:

$$d(\cdot \mid [H]_{<I}) = \begin{cases} \dfrac{\sum_{H' \in [H]_{<I}} d(\cdot, H')}{\sum_{H' \in [H]_{<I}} d(H')} & \text{if } \sum_{H' \in [H]_{<I}} d(H') \neq 0 \\ d(\cdot) & \text{if } \sum_{H' \in [H]_{<I}} d(H') = 0 \end{cases} \quad \square$$

Definition 2.40 and Proposition 2.4 below show that for a given history H there is at most one "reasonable" reduct of a domain description $\mathcal{D}$, which we denote $\mathcal{R}_H^{\mathcal{D}}$. Definition 2.40 also implicitly gives a theoretical procedure for finding $\mathcal{R}_H^{\mathcal{D}}$, as follows: (i) Arbitrarily generate an ne-domain description $\mathcal{N}$ from $\mathcal{D}$ by deleting all the s-propositions, replacing some of the p-propositions by equivalent but conditionless o-propositions, and deleting the rest of the p-propositions (at most 2^n such $\mathcal{N}$s may be generated in this way, where n is the number of p-propositions in $\mathcal{D}$). (ii) Form the pre-model-function $\tilde{M}_{\mathcal{D}}^{\mathcal{N}}$ with the arbitrarily chosen $\mathcal{N}$. (iii) Reduce $\mathcal{D}$ w.r.t. $\tilde{M}_{\mathcal{D}}^{\mathcal{N}}$ and H. (iv) If step (iii) results in $\mathcal{N}$ itself, then $\mathcal{N}$ is in fact $\mathcal{R}_H^{\mathcal{D}}$. If it does not, go back to step (i).

Definition 2.40 *(Reduct Set).* Let $\mathcal{D}$ be a domain description and H a history. Then the *reduct set of $\mathcal{D}$ w.r.t. H*, denoted $R(\mathcal{D}, H)$, is the set of ne-domain descriptions such that $\mathcal{N} \in R(\mathcal{D}, H)$ if and only if the reduct of $\mathcal{D}$ w.r.t. the pre-model-function $\tilde{M}_{\mathcal{D}}^{\mathcal{N}}$ and history H is $\mathcal{N}$ itself. $\square$

Proposition 2.4. *Let $\mathcal{D}$ be a domain description and H a history. Then the reduct set $R(\mathcal{D}, H)$ of $\mathcal{D}$ w.r.t. H contains at most one element. If $R(\mathcal{D}, H) \neq \emptyset$ this unique element is denoted $\mathcal{R}_H^{\mathcal{D}}$.* $\square$

The intuition behind Proposition 2.4, proved in [15], is that for a completely specified sensor input stream (as defined by H) the agent's belief state is uniquely and fully determined at every instant, and therefore, since the agent must exactly follow the prescription of what to perform provided by the p-propositions in $\mathcal{D}$, this will result in a unique sequence of agent action performances (re-expressed as additional o-propositions in the reduct). The p-propositions in $\mathcal{D}$ not re-instated as o-propositions in $\mathcal{R}_H^{\mathcal{D}}$ are exactly those whose belief-preconditions are not met by the agent's belief state w.r.t. H.

We can now formally define when an h-world is well-behaved, giving a second criterion by applying Definition 2.25 of a well-behaved world w.r.t. an ne-domain description using $\mathcal{R}_H^{\mathcal{D}}$. (Recall that the first criterion is that the h-world satisfy the CWSA as discussed on page 11):

Definition 2.41 *(Well-behaved h-world).* Let $\mathcal{D}$ be a domain description. The h-world (W, H) is *well-behaved w.r.t. $\mathcal{D}$* if (i) it satisfies the CWSA, and (ii) $R(\mathcal{D}, H)$ is non-empty and W is well-behaved w.r.t. $\mathcal{R}_H^{\mathcal{D}}$. $\square$

We can now state the main overarching definition for the semantics of EPEC. Definition 2.42 defines the *model-function* mapping from h-worlds to $[0, 1]$ (analogous to the ne-model function for ne-domains – see Definition 2.29), and Proposition 2.5 immediately after confirms that the model-function is a probability distribution over h-worlds:

Definition 2.42 *(Model Function).* The *model-function of a domain description $\mathcal{D}$* is the function $M_{\mathcal{D}} : \mathcal{W} \times \mathcal{H} \mapsto [0, 1]$ defined as follows:

$$M_{\mathcal{D}}(W, H) = \begin{cases} \tilde{M}_{\mathcal{D}}^{\mathcal{R}_H^{\mathcal{D}}} W, H & \text{if } (W, H) \text{ is well-behaved w.r.t. } \mathcal{D} \\ 0 & \text{otherwise} \end{cases} \quad \square \tag{16}$$

The following proposition, proved in [15], confirms that $M_{\mathcal{D}}$ is indeed a probability distribution:

Proposition 2.5. *Let $\mathcal{D}$ be a domain description. Then the model-function $M_{\mathcal{D}}$ of $\mathcal{D}$ is a probability distribution over $\mathcal{W} \times \mathcal{H}$, i.e. $\sum_{(W,H) \in \mathcal{W} \times \mathcal{H}} M_{\mathcal{D}}(W, H) = 1$.* $\square$

Since M_D is a joint probability distribution over worlds and histories, we can refer to marginal probabilities $M_D(W)$, $M_D(H)$, $M_D([H]_{<I})$, etc. and conditional probabilities $M_D(W \mid H)$, $M_D(H \mid W)$, $M_D(W \mid [H]_{<I})$ etc. using the standard definitions. Like $M^{ne}_{\mathcal{N}}$, M_D can also be straightforwardly extended to provide probabilities for i-formulas: $M_D(\varphi, H) = \sum_{W \mid\!\models \varphi} M_D(W, H)$.

We can now formally define this entailment, starting with the definition of the general form of a b-proposition:

Definition 2.43 *(b-proposition).* A *b-proposition* has the form

at I **believes** φ **with-probs** $\{([H_1], B_1, P_1), \ldots, ([H_m], B_m, P_m)\}$

for some instant I, i-formula φ, and real numbers $B_1, \ldots, B_m \in (0, 1]$ and $P_1, \ldots, P_m \in [0, 1]$ such that $\sum_{i=1}^{m} B_i = 1$. Each $[H_i]$ is an equivalence class $[H_i]_{<I}$ of histories represented as "$\langle H_i(I_1)@I_1, \ldots, H_i(I_n)@I_n\rangle$" for instants $I_1, \ldots, I_n$ such that, for $1 \leq j \leq n$, $I_j < I$ and $H_i(I_j) \neq \emptyset$, and with repetitions of actions entailed by a sensing outcome in $H_i(I_j)$ removed. □

We use the model-function M_D to ascertain when a b-proposition is entailed by a domain description D:

Definition 2.44 *(Entailment for Domain Descriptions).* The b-proposition

at I **believes** φ **with-probs** $\{([H_1], B_1, P_1), \ldots, ([H_m], B_m, P_m)\}$

is entailed by the domain description D iff, for $1 \leq i \leq m$, $M_D(\varphi \mid [H_i]_{<I}) = P_i$ and $M_D([H_i]_{<I}) = B_i$. □

Intuitively, if a domain description D entails a b-proposition

at I **believes** φ **with-probs** $\{([H_1], B_1, P_1), \ldots, ([H_m], B_m, P_m)\}$

this means that at instant I the agent will believe that the i-formula φ holds with one of the probabilities $P_1, \ldots, P_m$ depending on the input/output from its sensors/effectors, which is "recorded" in $[H_i]$ and has an associated probability B_i of actually being experienced/executed.

3. EPEC domains as ASP programs

In this section, we describe the details of the translation from domain descriptions of the form described in Section 2 to ASP. The domain dependent part of the translation described in Section 3.1 below, which has been automated within the publicly available EPEC web interface (See Section 5), is combined with the domain independent ASP implementation of EPEC's semantics described in Section 3.2 (publicly available on GitHub[3]) in order to compute a unique answer set. As explained above, this answer set contains exactly the set of h-traces that define the epistemic reduct of the domain, and is thus also a representation of all of the well-behaved h-worlds of the domain description under consideration. A key module of the domain independent implementation, described in Section 3.2.2, provides the Python sub-routines that compute numerical probabilities. These numerical values are used to attach probabilities to the h-traces in each candidate reduct that is generated, so that it can be tested to ascertain if it is the unique epistemic reduct. As mentioned in Section 2.2, once the unique answer set has been computed there is enough information within it to answer *queries* in the form of b-proposition templates such as

at 3 **believes** [*BiggsIsThief*]@−2 **with-probs** { ? }

Various issues and practicalities as regards the computation are discussed in Section 4.

3.1. A translation procedure from EPEC domain descriptions to ASP

We describe the translation of each type of EPEC proposition in turn. First, the translation includes a declaration of two numerical constants, `initialinstant` and `maxinstant`, to indicate the minimum and maximum elements of the set $\mathcal{I} = \{I_{initial}, I_{initial} + 1, \ldots, I_{max}\}$ of instants for the domain under consideration. In the hotel example, we have that $\mathcal{I} = \{-2, -1, 0, 1, 2, 3\}$, therefore:

```
#const initialinstant=-2.
#const maxinstant=3.
```

V-propositions of the form 'F **takes-values** $\langle V_1, \ldots, V_n\rangle$' are translated to ASP facts of the form:

```
fluent(f).
possVal(f,v1).
...
possVal(f,vn).
```

[3] See https://gitlab.com/fdasaro/epec-vanilla.

so that, for example, '*BiggsIsThief* **takes-values** $\langle false, true \rangle$' translates to:

```
fluent(biggsIsThief).
possVal(biggsIsThief,false).
possVal(biggsIsThief,true).
```

Note that, since constants in ASP are represented as strings that start with a lowercase letter, we adopt the convention that EPEC constants are always translated to match this format. For example, *DustForPrints* translates to '`dustForPrints`'.

Each action A is declared in the translation as:

```
action(a).
```

and each agent action B is additionally declared as:

```
agentAction(b).
```

For example, the action *BiggsSteals* is declared as '`action(biggsSteals)`', and *DustForPrints* is declared both as '`agentAction(dustForPrints)`' and '`action(dustForPrints)`'.

The formulas (θ's) contained in c- and s-propositions and the (partial) fluent states in i- and c-propositions are represented in the ASP translation using an expandable collection of *unique identifiers* `conj0`, `disj0`, `neg0`, `conj1`, ... etc. as arguments in the binary predicates `inConj`, `inDisj` and `inNeg`. For the purposes of this representation, (partial) fluent states are regarded as conjunctions, so that a partial fluent state

$$\tilde{X}_k = \{F_1 = V_1, \ldots, F_n = V_n\}$$

translates to:

```
inConj((f1,v1),conjk).
...
inConj((fn,vn),conjk).
```

where the k in '`conj`k' is some integer not used in any other unique identifier, so that `conj`k uniquely identifies $\tilde{X}_k$. For instance, the fluent state

$$\tilde{X}_0 = \{MoneyInBag, \neg BiggsPrints, \neg BiggsIsThief, OtherIsThief\}$$

may translate to:

```
inConj((moneyInBag,true),conj0).
inConj((biggsPrints,false),conj0).
inConj((biggsIsThief,false),conj0).
inConj((otherIsThief,true),conj0).
```

More generally, unique identifiers are used for representing arbitrary formulas composed with the connectives $\wedge$, $\vee$ and $\neg$ by an appropriate collection of `inConj`, `inDisj` and `inNeg` facts. For example, $\neg(F_1{=}V_1 \vee F_2{=}V_2) \wedge F_3{=}V_3$ might be assigned the unique identifier `conj28` via the following five facts:

```
inDisj((f1,v1),disj23).
inDisj((f2,v2),disj23).
inNeg(disj23,neg25).
inConj(neg25,conj28).
inConj((f3,v3),conj28).
```

An i-proposition of form '**initially-one-of** $\{(\tilde{X}_1, P_1^+), \ldots, (\tilde{X}_n, P_n^+)\}$' translates to:

```
initialCondition((conjx1,p1)).
...
initialCondition((conjxn,pn)).
```

where `conj`x_1, ..., `conj`x_n are unique identifiers associated with the fluent states $\tilde{X}_1$, ..., $\tilde{X}_n$ and $p_1 \ldots p_n$ are ASP string encodings of the probabilities P_1^+, ..., P_n^+. For instance, (HT5) translates to:

```
initialCondition((conj0,"0.8991")).
initialCondition((conj1,"0.1")).
initialCondition((conj2,"0.0009")).
```

where `conj0`, `conj1` and `conj2` are the unique identifiers of fluent states {*MoneyInBag*, ¬*BiggsPrints*, ¬*BiggsIsThief*, *OtherIsThief*}, {¬*MoneyInBag*, ¬*BiggsPrints*, ¬*BiggsIsThief*, ¬*OtherIsThief*} and {*MoneyInBag*, ¬*BiggsPrints*, *BiggsIsThief*, ¬*OtherIsThief*} respectively.

A c-proposition of form 'θ **causes-one-of** $\{(\tilde{X}_1, P_1^+), \ldots, (\tilde{X}_n, P_n^+)\}$' translates to:

```
causedOutcome(TR,(conjx_1,p_1)),I) :- holds(TR,(id_θ,I)).
...
causedOutcome(TR,(conjx_n,p_n)),I) :- holds(TR,(id_θ,I)).
```

where id_θ is the unique identifier of θ and, as before, `conj`x_1, …, `conj`x_n identify $\tilde{X}_1$, …, and $\tilde{X}_n$, and $p_1 \ldots p_n$ encode probabilities P_1^+, …, P_n^+. For instance, (HT1) translates to:

```
causedOutcome(TR,(conj4,"0.3"),I) :- holds(TR,(conj3,I)).
causedOutcome(TR,(conj5,"0.7"),I) :- holds(TR,(conj3,I)).
```

where `conj3` uniquely identifies formula $\theta = biggsSteals \wedge \neg OtherSteals$, and `conj4` and `conj5` uniquely identify partial fluent states {¬*MoneyInBag*} and {*BiggsPrints*, ¬*MoneyInBag*} respectively.

An s-proposition of the form

$$\theta \textbf{ senses } F \textbf{ with-accuracies } \begin{pmatrix} P_{1,1}, \ldots, P_{1,n} \\ \ldots \\ P_{n,1}, \ldots, P_{n,n} \end{pmatrix}$$

translates to:

```
sprop(id_θ).
objectOf(id_θ,f).
```

together with, for all $i, j = 1, \ldots, n$ such that $P_{i,j} > 0$:

```
possiblySensed(TR,((id_θ,v_i),p_ij),I) :-
     holds(TR,(id_θ,I)),
     holds(TR,((f,v_j),I)).
```

where id_θ is the unique identifier of θ, p_{ij} is a string encoding of $P_{i,j}$ and v_i and v_j are the i-th and j-th values listed in F's v-proposition. For example, (HT3) translates to:

```
sprop(conj8).
objectOf(conj8,biggsPrints).

possiblySensed(TR,((conj8,false),"0.999"),I) :-
    holds(TR,(conj8,I)),
    holds(TR,((biggsPrints,false),I)).

possiblySensed(TR,((conj8,true),"0.001"),I) :-
    holds(TR,(conj8,I)),
    holds(TR,((biggsPrints,false),I)).

possiblySensed(TR,((conj8,false),"0.05"),I) :-
    holds(TR,(conj8,I)),
    holds(TR,((biggsPrints,true),I)).

possiblySensed(TR,((conj8,true),"0.95"),I) :-
    holds(TR,(conj8,I)),
    holds(TR,((biggsPrints,true),I)).
```

where `conj8` uniquely identifies both the formula $\theta = DustForPrints$ and the s-proposition (HT3), serving as a unique name for each.

An o-proposition of the form 'A **occurs-at** I **with-prob** P^+ **if-holds** θ' translates to:

```
occurs(TR,A,I,p):- holds(TR,(id_θ,I)).
```

where id_θ is the unique identifier of θ and p is the string encoding of the probability P^+. For instance, (HT6) translates to:

```
occurs(TR,biggsSteals,-2,"1"):- holds(TR,(conj10,-2)).
```

where `conj10` uniquely identifies the formula $\theta = BiggsIsThief$.

A p-proposition 'A **performed-at** I **with-prob** P^+ **if-believes** $(\theta, \bar{P})$' translates to:

```
performed(A,I,p,id_θ,int_P̄).
```

where id_θ is the unique identifier of θ, p is the string encoding of the probability P^+, and $int_{\bar{P}}$ translates the interval $\bar{P}$ to its ASP form, so that, for example, $(0.5, 0.6]$ translates to `"(","0.5","0.6","]"`. For instance, (HT9) translates to

```
performed(doIdParade,1,"1",conj12,"(","0.33","1","]").
```

where `conj12` uniquely identifies formula $\theta = BiggsIsThief$.

3.2. Implementation of EPEC semantics

The domain independent component of the ASP translation of EPEC, the full listing of which is publicly available at https://gitlab.com/fdasaro/epec-vanilla, consists of two sub-modules that deal with the logical and the probabilistic aspects of EPEC's semantics. In the following sections we describe the key features of each of these sub-modules in turn.

3.2.1. The logical sub-module

EPEC's model-theoretic semantics, as summarised in Section 2.2, imposes intuitive constraints on well behaved h-worlds and their associated h-traces, such as the *default persistence* of fluents and what we have termed in EPEC's semantics (see Definition 2.19) the *Closed World Assumption for Actions* (CWAA), i.e. that only actions explicitly referred to by o- and p-propositions can occur along the time line. The rules in the logical sub-module `epec.lp` ensure that each h-trace generated by the program adheres to these constraints.

We begin with some rules to establish the basic building blocks of the semantics. The sort $\mathcal{I}$ of instants is defined by the rule

```
instant(initialinstant..maxinstant).
```

Actions can only take value *true* or *false*:

```
possVal(A,true) :- action(A).
possVal(A,false) :- action(A).
```

and a characteristic predicate is defined for the set $\mathcal{F} \cup \mathcal{A}_a \cup \mathcal{A}_e$:

```
fluentOrAction(X) :- fluent(X).
fluentOrAction(X) :- action(X).
```

Characteristic predicates are also included for literals and i-literals, that is, elements of $\mathcal{F} \cup \mathcal{A}_a \cup \mathcal{A}_e$ coupled with a specific value (with an instant also attached in the case of i-literals):

```
literal((X, V)) :-
    fluentOrAction(X),
    possVal(X, V).

iliteral((L, I)) :-
    literal(L),
    instant(I).
```

At the top level of control, a Python script (the file `epec` in the GitLab repository) iterates by incrementing and feeding an integer parameter `ntraces` into `epec.lp`, repeating this until a value of `ntraces` is found that results in a unique answer set. Within each of these iterations, the rules of `epec.lp` treat `ntraces` as an integer constant indicating the total number of h-traces. Distinct individual h-traces are identified via the numerical argument of the unary predicate `trace`:

```
trace(1..ntraces).
```

Given an h-trace `TR` and an instant `I`, the `holds` predicate maps fluents and actions to the value they take at `I`. For each h-trace `TR`, its characteristic 'holds' predicate is initialised using an ASP choice rule:

```
1{holds(TR, ((X, V), I)) : iliteral(((X, V), I))}1 :-
    trace(TR),
    fluentOrAction(X),
    instant(I).
```

In order for our translation procedure to work, the `holds` predicate must also be able to handle formulas that contain conjunctions (encoded via `inConj`), disjunctions (encoded via `inDisj`), and negations (encoded via `inNegated`). This is the goal of the following code:

```
  holds(TR, (Theta,I)) :-
   trace(TR),
   instant(I),
```

```
    inDisj(Theta_dash, Theta),
    holds(TR, (Theta_dash,I)).

  holds(TR, (Theta,I)) :-
    trace(TR),
    instant(I),
    inNegated(Theta_dash, Theta),
    not holds(TR, (Theta_dash,I)).

  someConjunctDoesNotHoldIn(TR, (Theta,I)) :-
    trace(TR),
    instant(I),
    inConj(Theta_dash, Theta),
    not holds(TR, (Theta_dash,I)).

  holds(TR, (Theta,I)) :-
    trace(TR),
    instant(I),
    inConj(Theta_dash, Theta),
    not someConjunctDoesNotHoldIn(TR, (Theta,I)).
```

The intuition behind the first rule above is that if `Theta` is a disjunction such that `Theta_dash` is one of the disjuncts, and `Theta_dash` holds, then the entire disjunction `Theta` holds. The second rule states that if `Theta` is the negation of some formula `Theta_dash` that does not hold, then `Theta` holds. The third rule checks for the existence of any conjunct `Theta_dash` that does not hold in `Theta`, and is used within the fourth rule for conjunction, which asserts that a conjunction `Theta` holds if and only if there is conjunct `Theta_dash` within it, and no conjunct within it that does not hold.

We now introduce the logical machinery that is needed to filter out h-traces that do not adhere to the commonsense law of intertia for fluents, i.e. h-traces where fluents change value without a matching cause.

Recall that the unique i-proposition in a domain consists of a series of *outcomes* with probabilities attached. Each outcome encodes an initial fluent state and the corresponding probability that this state initially holds. For each h-trace `TR`, the following choice rule picks exactly one outcome `O` from the unique i-proposition in the domain, and assigns it to `TR`:

```
1{initialChoice(TR,O) : initialCondition(O)}1 :-
    trace(TR).
```

Auxiliary predicates `inOcc` and `sOcc` identify instants `I` in each h-trace `TR` where a cause or sensing occurs respectively:

```
inOcc(TR,I) :-
    trace(TR),
    instant(I),
    causedOutcome(TR, O, I).

sOcc(TR,S,I) :-
    trace(TR),
    instant(I),
    sprop(S),
    possiblySensed(TR, ((S,W),Q), I).
```

The program uses `inOcc` to impose that exactly one partial state from the unique c-proposition occurring at some instant is the chosen effect (the second argument of `effectChoice`) in each h-trace:

```
1{effectChoice(TR, O, I) : causedOutcome(TR, O, I)}1 :-
    trace(TR),
    inOcc(TR,I).
```

and similarly uses `sOcc` to impose that, in each h-trace, exactly one value is sensed each time a sensing action occurs (i.e. each time an s-proposition is triggered):

```
1{sensedValue(TR, ((S,V),P), I ) :
                  possiblySensed(TR, ((S,V),P), I)}1 :-
   trace(TR),
   sOcc(TR, S, I).
```

The next two clauses define the auxiliary predicates `possiblyPerformed` and `definitelyPerformed`, representing respectively the set of action-instant pairs such that `A` might occur at `I` (i.e. with a probability greater than 0) and the subset of action-instant pairs such that `A` definitely occurs at `I` (i.e. with probability 1):

```
possiblyOccurs(TR,A,I) :-
    trace(TR),
    occurs(TR,A,I,P).

definitelyOccurs(TR,A,I) :-
    trace(TR),
    occurs(TR,A,I,"1").
```

The following constraints implement the *Closed World Assumption for Actions* (CWAA - see Defintion 2.19), by stating that, at any given instant in a given h-trace, actions definitely occurring (i.e. with probability 1) cannot hold false, and that if it is impossible that an action occurs then the action cannot hold true:

```
:- action(A),
   instant(I),
   trace(TR),
   holds(TR, ((A, false),I) ),
   definitelyOccurs(TR,A,I).

:- action(A),
   instant(I),
   trace(TR),
   holds(TR, ((A, true),I) ),
   not possiblyOccurs(TR,A,I).
```

The next constraint excludes h-traces in which the fluent state holding initially does not match with the initial outcome chosen:

```
:- trace(TR),
   initialChoice(TR,(S, P)),
   literal(L),
   inConj(L, S),
   not holds(TR,(L, initialinstant)).
```

The next three constraints implement three aspects of the Commonsense Law of Inertia (see for example [15]), namely *justified change*, *minimal change* and *default persistence* of fluents:

```
:- trace(TR),
   not holds(TR, ((F, V), I+1) ),
   instant(I),
   I<maxinstant,
   fluent(F),
   effectChoice(TR, (X, P), I),
   inConj( (F, V), X ).

:- trace(TR),
   holds(TR, ((F, V), I+1) ),
   instant(I),
   I<maxinstant,
   fluent(F),
   effectChoice(TR, (X, P), I),
   not inConj( (F, V), X ),
   not holds(TR, ((F, V),I) ).

:- trace(TR),
   instant(I),
   I<maxinstant,
   fluent(F),
   not inOcc(TR,I),
   holds(TR, ((F, V), I) ),
   not holds(TR, ((F, V), I+1) ).
```

The following choice rule populates the *candidate reduct* being generated within the potential answer set, by choosing a subset of action occurrences from amongst those that are possibly performed by the agent:

```
0{occurs(TR, A, I, P)}1 :-
   trace(TR),
   action(A),
   instant(I),
   performed(A, I, P, Precondition, LeftBracket,
                                    X, Y, RightBracket).
```

A number of predicates are also needed to implement the concept of distinguishable and indistinguishable sensing and acting histories up to some instant I. As the name suggests, these represent the agent's sensory experiences and actions, as opposed to what actually holds, within the environment:

```
distinguishableSprops(S1,S2) :-
   sprop(S1), sprop(S2), objectOf(S1,F),
   not objectOf(S2,F).

distinguishableSprops(S1,S2) :-
   sprop(S1), sprop(S2), inConj((X,V),S1),
   not inConj((X,V),S2), agentAction(X).

indistinguishableSprops(S1, S2) :-
   sprop(S1), sprop(S2),
   not distinguishableSprops(S1,S2).

sensingDistinguishableUpTo(TR1, TR2, I) :-
   instant(I), sprop(S1), sprop(S2),
   trace(TR1), trace(TR2),
   indistinguishableSprops(S1, S2),
   sensedValue(TR1, ((S1,V1),P), I),
   sensedValue(TR2, ((S2,V2),Q), I),
   V1 != V2.

sensingDistinguishableUpTo(TR1, TR2, I) :-
   instant(I),
   trace(TR1), trace(TR2),
   sensingDistinguishableUpTo(TR1, TR2, I-1).

sensingIndistinguishableUpTo(TR1, TR2, initialinstant-1) :-
   trace(TR1), trace(TR2).

sensingIndistinguishableUpTo(TR1, TR2, I) :-
   instant(I),
   trace(TR1), trace(TR2),
   not sensingDistinguishableUpTo(TR1, TR2, I).
```

The first two rules define what it means for two s-propositions to be distinguishable, i.e., when they refer to the sensing of different objects, or when they have distinct preconditions[4]). The subsequent rule then defines the concept of indistinguishability between two s-propositions, which is simply the negation of them being distinguishable. The `sensingDistinguishableUpTo` rules determine up to which instant two traces are distinguishable based on the sensed values of their s-propositions. The first rule checks if there is a sensed value at a given instant `I` in both traces such that indistinguishable s-propositions detect different values. The second rule is a recursive rule that states that if two traces are distinguishable up to an instant `I-1`, they are also distinguishable up to instant `I`. Lastly, the `sensingIndistinguishableUpTo` rules define when two traces are indistinguishable up to a certain instant. The first rule is the base case, stating that all (empty) traces are initially indistinguishable. The second is a general rule that states two traces are indistinguishable up to an instant `I` if they are not distinguishable up to that same instant.

Finally, similarly to [24] the domain-independent module defines a number of rules and constraints that ensure the existence of a sufficient number of h-traces. The program includes rules that ensure that each trace has a unique initial condition, and that each causal occurrence (a change in the state of the system caused by an action) does take place in some trace. The program also includes similar rules for sensing outcomes. Conversely, the program also defines predicates to make sure that the traces being considered are all distinct. Both components, which we do not detail here, are necessary to ensure that all possible behaviours of the system are being adequately represented.

[4] Note that, for simplicity, in the implementation we only deal with the less general case where the precondition of an s-proposition is a conjunction.

3.2.2. The probabilistic sub-module

A separate module, `epec_prob.lp`, takes care of the probabilistic calculations and implements the fixpoint semantics of EPEC. This module gets grounded and solved as soon as the main module `epec.lp` produces a model. This is to minimize the computational complexity of the task (more on this in Section 4). Since Clingo does not natively support numerical operations, we use Clingo's Python API in this module to perform arithmetic calculations. Readers who are unfamiliar with this topic may find it useful to refer to the current Python API manual for Clingo for an introduction (version 5.6 at the time of writing, at https://potassco.org/clingo/python-api/5.6). The primary functionality we utilize in the code below is the @-syntax, which allows one to write Python functions that are invoked by clingo during grounding. For example, the Python code

```
def div(a,b):
    x = float(a.string)
    y = float(b.string)
    return clingo.String(str(x/y))
```

can be applied to any pair of numbers encoded as strings in `a` and `b` within Clingo. This routine casts them to the `float` type, divides them, and then converts the result back to the appropriate Clingo `String` type. The behaviour of a Clingo program containing a fact of the form `a(@div("3.4","0.8"))` would be to ground this fact into `a("4.25")`. In the remainder of this section, we will use other similar Python functions, namely: `@sub` (for subtraction), `@log` (for base 10 logarithm), `@exp` (for base-10 exponentiation), and `@in` (to check whether a given number falls within a specified interval, e.g. `@in("1.2","[","1","2",")")` evaluates to `true` because $1.2 \in [1,2)$). We also utilize Clingo's native `#sum` aggregate to sum probabilities.

In the probabilistic sub-module, we first introduce a predicate to state the probabilities of a given o-proposition resulting or not resulting in an occurrence of the action to which it refers within a particular trace at a particular instant:

```
evalOProp(TR,A,I,P) :-
   trace(TR), action(A), instant(I),
   occurs(TR,A,I,P), holds(TR, ((A,true),I) ).

evalOProp(TR,A,I,@sub("1",P)) :-
   trace(TR), action(A), instant(I),
   occurs(TR,A,I,P), holds(TR, ((A,false),I) ).
```

In other words, the probability of action A holding at I in trace TR as a consequence of the o-proposition "A **occurs-at** I **with-prob** P^+" is P^+, whereas the probability of A not holding is $1 - P^+$.

The following clause of this module calculates the probability of each h-trace:

```
evalTraceNarrative(TR,@exp(P)) :-
   P = #sum{ @log(X),-1,O : initialChoice(TR,(O,X));
             @log(X),I,O  : effectChoice(TR,(O,X),I);
             @log(X),I,O  : sensedValue(TR,(O,X),I);
             @log(X),I,A  : evalOProp(TR,A,I,X) },
             trace(TR).
```

where the python functions `@log` and `@exp` calculate the (base e) logarithm and exponential of the given probabilities respectively. This is a trick to exploit clingo's in-built `#sum` aggregate to multiply the probabilities encoded as log probabilities in the predicates `initialChoice`, `effectChoice`, `sensedValue` and `evalOProp`. The trick works as follows: the base-e logarithm is calculated for all probabilities `X` in all predicates, then they get summed via the `#sum` predicate, and finally they get exponentiated back to produce the evaluation of a specific trace – the output of `evalTraceNarrative` is consistent with the Pre-model-function in Definition 2.37 since summing log-probabilities and then exponentiating them is equivalent to multiplying probabilities.

The fixpoint semantics, as formalized in Definition 2.40, is implemented in two constraints. The first of these ensures that, within the epistemic reduct, no p-propositions (represented via the predicate `performed`) whose belief conditions fall outside their probabilistic range have been 'reduced' to corresponding o-propositions (represented via the `occurs` predicate):

```
:- trace(TR), action(A), instant(I), occurs(TR, A, I, P),
  performed(A, I, P, Precondition, LeftBracket,
                                      X, Y, RightBracket),
  B = #sum{ E,OTHER_TR :
            sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
            holds(OTHER_TR, (Precondition,I) ),
            evalTraceNarrative(OTHER_TR,E) },
  C = #sum{ F,OTHER_TR :
            sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
            evalTraceNarrative(OTHER_TR,F) },
  D = @div(B,C),
  not @in(D,LeftBracket,@exp(X),@exp(Y),RightBracket).
```

The second constraint ensures that each p-proposition whose belief condition is within its probabilistic range has been reduced to a corresponding o-proposition:

```
:- trace(TR),action(A),instant(I), not occurs(TR, A, I, P),
   performed(A, I, P, Precondition, LeftBracket,
                                      X, Y, RightBracket),
   B = #sum{ E,OTHER_TR :
             sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
             holds(OTHER_TR, (Precondition,I) ),
             evalTraceNarrative(OTHER_TR,E) },
   C = #sum{ F,OTHER_TR :
             sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
             evalTraceNarrative(OTHER_TR,F) },
   D = @div(B,C),
   @in(D, LeftBracket,@exp(X),@exp(Y),RightBracket).
```

In these constraints the python function `@in` implements the set theoretic relation $\in$, so that, for example, `@in(0.2,"(",0.1,0.4,"]")` means $0.2 \in (0.1, 0.4]$ and evaluates to *true*. The bodies of these constraints can be read as follows: sum the probabilities of h-traces indistinguishable up to `I` and satisfying the formula `Precondition`. Then, divide by the total probability of all h-traces indistinguishable from the current trace. Finally, without loss of generality let `LeftBracket` be the open square bracket "[" and `RightBracket` be the closed square bracket "]". Then, the last row in the constraints checks whether the precondition of the corresponding p-proposition is (resp. is not) satisfied. This is consistent with the fixpoint construction in Definition 2.40.

3.3. Extracting b-propositions from answer sets

In this section we describe how b-propositions entailed by an EPEC domain description D can be derived from the unique answer set Z_{Π_D} computed from D's ASP translation Π_D, where Π_D contains the domain dependent and domain independent code described in Sections 3.1 and 3.2. Recall from Definition 2.43 that the general form of a b-proposition is

at I **believes** φ **with-probs** $\{([H_1], B_1, P_1), \ldots, ([H_m], B_m, P_m)\}$

First, we define a correspondence between i-formulas and appropriate subsets of Z_{Π_D}. We say that an i-formula ϕ *holds* for trace `id` in answer set Z if and only if `holds(id,(`id_θ`,`I`))` belongs to the answer set Z. Similarly, an equivalence class of acting and sensing histories $[H_i] = \langle H_i(I_1)@I_1, \ldots, H_n(I_n)@I_n\rangle$ *holds* for trace `id` in answer set Z if and only if for all $1 \leq j \leq n$, if $H_i(I_j)@I_j$ is the set $\{O_1, \ldots, O_m, A_1, \ldots, A_p\}$ where $O_1 = ((\theta_1, X_1), V_1), \ldots, O_m = ((\theta_m, X_m), V_m)$ are sensing outcomes and $A_1, \ldots, A_p$ are agent actions, then the literals

`sensedValue(id,((id`$_{\theta_1}$`,`V_1`),`P_1`),`I_j`)`
`...`
`sensedValue(id,((id`$_{\theta_m}$`,`V_m`),`P_m`),`I_j`)`
`isActuallyPerformed(id,`A_1`,`I_j`)`
`...`
`isActuallyPerformed(id,`A_p`,`I_j`)`

belong to Z.

For example, the i-formula $[BiggsIsThief = true]@1 \wedge [BiggsPrints = true]@0$ holds for trace `12` in the answer set Z_{D_h} shown in part in Fig. 4 because it contains both the literal `holds(12,((biggsIsThief,true),1))` and the literal `holds(12,((biggsPrints,true),1))`. Also the sensing history $[H] = \langle DustForPrints@-1, ((DustForPrints, BiggsPrints), true)@-1\rangle$ holds for trace `12` in Z_{D_h} since both `isActuallyPerformed(12,dustForPrints,-1)` and `sensedValue(12,((conj8,true),"0.95"),-1)` belong to it (recall from Section 2.1 that `conj8` is the unique id assigned to the relevant s-proposition).

Using the above terminology, we define that an answer set Z *derives* a b-proposition of the general form above if and only if for all $1 \leq i \leq m$, (i) B_i is the sum of all `B`'s such that `evalTraceNarrative(id,B)` is in Z for some `id` and $[H_i]$ holds for trace `id` in Z, and (ii) P_i is the sum, divided by B_i, of all `P`'s such that `evalTraceNarrative(id,P)` is in Z for some `id`, and $[H_i]$ and ϕ both hold for trace `id` in Z.

3.4. Correctness of the translation

In this section we state and prove the correctness of the translation of epistemic domains into ASP. We do this by showing that b-propositions can be derived from the unique answer set computed from the translation, as described in Section 3.3, if and only if they are entailed from the domain description, as described in Section 2.3. The proof relies on a technique known as *splitting* [23], which we now briefly summarize.

A set U of ground atoms is a *splitting set* for a ground program Π if, for each rule in Π, if U contains some atom in the head of the rule, then it also contains all the atoms occurring in that rule. For instance, if $\Pi' = \{a \leftarrow not\ b, b \leftarrow c, c\}$ then $\{a, b, c\}$, $\emptyset$, $\{b, c\}$ and $\{c\}$ are splitting sets for Π', whereas $\{a, b\}$, $\{a\}$ and $\{b\}$ are not. A splitting set U splits an answer set program Π into a bottom

```
holds(12,((moneyInBag,true),-2)) holds(12,((biggsPrints,false),-2)) holds(12,((biggsIsThief,true),-2))
holds(12,((otherIsThief,false),-2)) holds(12,((biggsSteals,true),-2)) holds(12,((otherSteals,false),-2))
holds(12,((dustForPrints,false),-2)) holds(12,((charge,false),-2)) holds(12,((do12Parade,false),-2))
holds(12,((moneyInBag,false),-1)) holds(12,((biggsPrints,true),-1)) holds(12,((biggsIsThief,true),-1))
holds(12,((otherIsThief,false),-1)) holds(12,((dustForPrints,true),-1)) holds(12,((biggsSteals,false),-1))
holds(12,((otherSteals,false),-1)) holds(12,((charge,false),-1)) holds(12,((do12Parade,false),-1))
holds(12,((moneyInBag,false),0)) holds(12,((biggsPrints,true),0)) holds(12,((biggsIsThief,true),0))
holds(12,((otherIsThief,false),0)) holds(12,((biggsSteals,false),0)) holds(12,((otherSteals,false),0))
holds(12,((dustForPrints,false),0)) holds(12,((charge,false),0)) holds(12,((do12Parade,false),0))
holds(12,((moneyInBag,false),1)) holds(12,((biggsPrints,true),1)) holds(12,((biggsIsThief,true),1))
holds(12,((otherIsThief,false),1)) holds(12,((do12Parade,true),1)) holds(12,((biggsSteals,false),1))
holds(12,((otherSteals,false),1)) holds(12,((dustForPrints,false),1)) holds(12,((charge,false),1))
holds(12,((moneyInBag,false),2)) holds(12,((biggsPrints,true),2)) holds(12,((biggsIsThief,true),2))
holds(12,((otherIsThief,false),2)) holds(12,((charge,true),2)) holds(12,((biggsSteals,false),2))
holds(12,((otherSteals,false),2)) holds(12,((dustForPrints,false),2)) holds(12,((do12Parade,false),2))
holds(12,((moneyInBag,false),3)) holds(12,((biggsPrints,true),3)) holds(12,((biggsIsThief,true),3))
holds(12,((otherIsThief,false),3)) holds(12,((biggsSteals,false),3)) holds(12,((otherSteals,false),3))
holds(12,((dustForPrints,false),3)) holds(12,((charge,false),3)) holds(12,((do12Parade,false),3))
sensedValue(12,((conj8,true),"0.95"),-1) sensedValue(12,((conj9,true),"0.85"),1)
evalTraceNarrative(12,"0.0005087250317770103") isActuallyPerformed(12,charge,2) isActuallyPerformed(12,do12Parade,1)
isActuallyPerformed(12,dustForPrints,-1)
```

Fig. 4. This subset of answer set Z_{D_h} of the domain D_h described in Example 1.1 corresponds to the h-trace, assigned id 12, in Fig. 3. These parts of answers sets are used to derive b-propositions of interest. Parts in bold are those relevant for the example in the text.

program $bot_U(\Pi)$ and a top program $top_U(\Pi) = \Pi \setminus bot_U(\Pi)$. The program $bot_U(\Pi)$ contains all the rules of Π with a member of U as their head. With the program Π' defined as above, $U' = \{c\}$ splits Π' into $bot_{U'}(\Pi') = \{c\}$ and $top_{U'}(\Pi') = \{a \leftarrow not\ b, b \leftarrow c\}$.

The *splitting set theorem* states that the answer sets of Π are exactly those that can be expressed as $X \cup Y$ for some X that is an answer set of $bot_U(\Pi)$ and some Y that is an answer set of $\epsilon_U(top_U(\Pi), X)$. The set $\epsilon_U(top_U(\Pi), X)$ denotes the *partial evaluation of the program* $top_U(\Pi)$ *w.r.t.* U defined as follows: a rule r is in $\epsilon_U(top_U(\Pi), X)$ if and only if there exists a rule $r' \in top_U(\Pi)$ such that (i) all positive literals that are both in the body of r' and in U are also in X, (ii) there is no negative literal in the body of r' whose corresponding atom is in both U and X, and (iii) the rule r is obtained from r' by removing all literals from the body of r' whose corresponding atom is in U. If we consider Π' and U' again and let X' be the only answer set $\{c\}$ of $bot_{U'}(\Pi') = \{c\}$, $\Pi'' = \epsilon_{U'}(top_{U'}(\Pi'), X') = \{a \leftarrow not\ b, b\}$ and notice that now we can split Π'' itself. If we let $U'' = \{b\}$, then $bot_{U''}(\Pi'') = \{b\}$ and $\Pi''' = \epsilon_{U''}(top_{U''}(\Pi''), X'') = \emptyset$ for the only answer set $X'' = \{b\}$ of $bot_{U''}(\Pi'')$. The answer sets of the original program Π' can now be obtained as $X' \cup X'' \cup X'''$, where $X' = \{c\}$ is the answer set of $bot_{U'}(\Pi')$, $X'' = \{b\}$ is the answer set of $bot_{U''}(\Pi'') = \{c\}$ and $X''' = \emptyset$ is the answer set of Π'''. Then, the program Π' has only one answer set $\{b, c\}$.

Conventionally, if the considered language includes predicate symbols, splitting a program Π with respect to some predicates $p_1, \ldots, p_n$ means splitting it with respect to the set U of all groundings of $p_1, \ldots, p_n$, and so in this context $U = \{p_1, \ldots, p_n\}$ signifies this set of all groundings.

Proposition 3.1 *(Correctness). Let $\mathcal{D}$ be an EPEC domain description, let Π_D be its translation to ASP as described in Section 3.1, with the addition of the domain independent rules as described in Section 3.2, and let Z_D be an answer set of Π_D. Then the b-proposition*

$$\textbf{\textit{at } I \textit{ believes } } \varphi \textbf{\textit{ with-probs }} \{([H_1], B_1, P_1), \ldots, ([H_m], B_m, P_m)\}$$

can be derived from Z_D if and only if it is entailed from $\mathcal{D}$.[5]

Proof. The proof consists of performing subsequent splits on the ASP translation (both the domain-independent and domain-dependent part) and then showing that predicates match the formal definitions in Section 2.3. In particular, we rely on the proof of correctness in [14,13] for non-epistemic domains, and then show how the same strategy can be extended to epistemic domains. We present the proof in three parts: (i) to aid intuition, we first show how the proof works for a simple predicate of the non-epistemic part of the domain, (ii) we then show that the handling of s-propositions is correct with respect to Definition 2.33. Finally, (iii) we show how the constraints in the program implement the fixpoint construction.

Let $\mathcal{D}$ be a domain description, and let Π_D be its translation to ASP as in the statement of the proposition.

For part (i) of the proof, consider the predicate `action`. This predicate occurs only in ground facts resulting from the translation. Therefore, if we let $U = \{\texttt{action}\}$ one can split the program Π_D in two parts $bot_U(\Pi_D)$, which contains only `action` ground facts, and $top_U(\Pi_D)$ which contains the rest of the program. Therefore, an answer set of Π_D will contain all the ground instances of `action`, union an answer set of $top_U(\Pi_D)$. Note that, by virtue of the translation, for all actions $A \in \mathcal{A}$, $bot_U(\Pi_D)$ contains a corresponding ground fact `action(a)` for some constant `a` that is in a one-to-one correspondence with action A, and therefore the predicate `action` correctly represents the action sort $\mathcal{A}$, i.e. $A \in \mathcal{A} \Leftrightarrow \texttt{action(a)} \in bot_U(\Pi_D)$. Now focus on the predicate `possval`. The partial evaluation of $top_U(\Pi_D)$ with respect to the unique answer set of $bot_U(\Pi_D)$ contains rules of the form

```
possVal(a,true) :- action(a).
```

[5] Note that this proposition assumes that Python correctly implements the numerical and set-theoretical functions `@log`, `@exp`, `@div`, `@sub` and `@in`.

Predicate	Relevant Definitions
ne-Domain Description	Proved correct in [14,13]

`agentAction/1`
`sprop/1`
`objectOf/2`
`possiblySensed/3`
`performed/9`
	Definition 2.1 (Domain Language),
	Definition 2.9 (s-proposition),
	Definition 2.12 (p-proposition)

`sOcc/3`
`sensedValue/4`
`distinguishableSprops/2`
`occurs/4`
	Definition 2.33 (Sensing Occurrence),
	Definition 2.34 (CWSA),
	Definition 2.38 (Reduct of a p-proposition)

`indistinguishableSprops/2`	
`sensingDistinguishableUpTo/3` `evalTraceNarrative/2`	Definition 2.35 (History Evaluation), Definition 2.37 (Pre-model-function)
`sensingIndistinguishableUpTo/3`	Definition 2.31 (Indistinguishable Histories)
Constraints in Section 3.2.2	Definition 2.41 (Well-behaved h-world)

Fig. 5. This table, when read from top to bottom, provides an overview of the proof of correctness, as it shows the series of splits that can be applied to the program Π_D to derive an answer set consistent with the formal machinery of EPEC. Each row corresponds to a series of predicates that, at every step, are considered to be in the bottom program against which the remainder of the program is evaluated. The definitions on the right are those with which the predicates on the left must be shown to be consistent. The proof relies on the correctness of the ne-Domain Descriptions, as reported in the first row. Finally, the fixpoint construction is verified by the constraints in Section 3.2.2, which are evaluated at the end.

```
possVal(a,false) :- action(a).
```

for all constants `a`. However, if we partially evaluate this program with respect to the set $U = \{$`action`$\}$ (i.e., $\epsilon_U(top_U(\Pi_D), Z)$, where Z is the unique answer set of the bottom program) we get

```
possVal(a1,true).
possVal(a1,false).
...
possVal(an,true).
possVal(an,false).
```

for all constants `a1`, …, `an` that are translations of actions in the sort $\mathcal{A}$. This is consistent with Definition 2.1 which imposes that $vals(A) = \langle false, true \rangle$ for all actions $A \in \mathcal{A}$. Therefore, `possVal` correctly represents *vals* for actions, in the sense that for all actions $A \in \mathcal{A}$,

$$vals(A) = \langle false, true \rangle \Leftrightarrow \texttt{possVal(a,false)} \text{ and}$$

$$\texttt{possVal(a,true)} \in \epsilon_U(top_U(\Pi_D), Z)$$

This process can be repeated recursively by appropriately splitting the partially evaluated top program and checking, at every step, that the rules in the translation correctly match the definitions in EPEC's semantics.

The correctness of the translation, for ne-domain descriptions only, was proved in [14,13].[6] Moving on to part (ii) of the proof, we now show how that result can be extended to full epistemic domains.

Among the predicates that were not proven correct in [14,13] we focus on `possiblySensed` and `sOcc`.

Recall that an s-proposition $\underline{\mathbf{s}}$ of the form

$$\theta \textbf{ senses } F \textbf{ with-accuracies } \begin{pmatrix} P_{1,1}, \ldots, P_{1,n} \\ \ldots \\ P_{n,1}, \ldots, P_{n,n} \end{pmatrix}$$

is translated to a series of `possiblySensed` rules of the form

```
possiblySensed(TR,((id_θ,v_i),p_ij),I) :-
    holds(TR,(id_θ,I)),
    holds(TR,((f,v_j),I)).
```

where id_θ is a constant standing for the precondition θ, v_i stands for the value of F being sensed, and p_{ij} stands for the probability of sensing V_i when V_j is the actual value of F. Then, since `holds` was proved in [14,13] to be a correct representation of what holds in a given trace at a given instant, partially evaluating the predicate `possiblySensed` with respect to `holds` reduces the above to a series of ground facts of the form

[6] Note that, in [13], ne-domain descriptions were referred to as PEC+ domain descriptions.

`possiblySensed(tr,((`id_θ`,`v_i`),`p_{ij}`),i).`

whenever the preconditions of **s** are satisfied in a trace represented by `tr` at some instant represented by `i`. We now split the program on the `possiblySensed` predicate, so that we are able to partially evaluate the top program with a focus on `sOcc`. Again, we start by recalling its definition:

```
sOcc(TR,S,I) :-
  trace(TR),
  instant(I),
  sprop(S),
  possiblySensed(TR, ((S,W),Q), I).
```

and, again, partially evaluating this will produce a series of ground facts of the form

```
sOcc(tr,s,i).
```

for correct representations of trace `tr`, s-proposition `s` and instant `i` whenever the preconditions of **s** are satisfied in trace `tr` at `i`. This correctly matches the definition of a sensing action occurring at some instant in an h-trace in Definition 2.33, i.e.:

> **s** is activated at I in world W w.r.t. $D \Leftrightarrow$
>
> `sOcc(tr,s,i)` $\in \epsilon_{U'}(top_{U'}(\Pi'_D), Z')$

for `tr` an h-trace of W, $U' = \{$`sOcc`$\}$, and Π'_D, Z' the partially evaluated program and an answer set resulting from the previous splits, respectively.

All the other predicates (e.g. `evalTraceNarrative`, `sensedValue`, `sensedValue`, etc.) can be similarly verified to match their semantical counterparts with consecutive splits. A map of the whole splitting process, with the definitions predicates can be verified against, is shown in Fig. 5.

Moving on to part (iii), we now show that the constraints given at the end of Section 3.2.2 correctly implement the fixpoint construction in EPEC's semantics. The effect of these constraints is that of eliminating answer sets that satisfy their bodies. We analyze the following constraint:

```
:- trace(TR), action(A), instant(I), occurs(TR, A, I, P),
  performed(A, I, P, Precondition, LeftBracket,
                                      X, Y, RightBracket),
  B = #sum{ E,OTHER_TR :
            sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
            holds(OTHER_TR, (Precondition,I) ),
            evalTraceNarrative(OTHER_TR,E) },
  C = #sum{ F,OTHER_TR :
            sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
            evalTraceNarrative(OTHER_TR,F) },
  D = @div(B,C),
  not @in(D,LeftBracket,@exp(X),@exp(Y),RightBracket).
```

having shown in parts (i) and (ii) that all the predicates appearing in its body, that is `trace/1`, `action/1`, `instant/1`, `occurs/4`, `performed/9`, `holds/2`, `evalTraceNarrative/2` and `sensingIndistinguishableUpTo/3` are correct with respect to their semantical counterparts. Considering that the predicate `evalTraceNarrative` outputs probabilities in variables `E` and `F`, assuming that the `#sum` aggregate sums them, and substituting the set- and number-theoretical operators with usual ones, we interpret the body of the constraint as follows: "(W, H) is an h-trace [encoded as `TR` satisfying `trace(TR)`], $A \in \mathcal{A}$ [`action(A)`], $I \in \mathcal{I}$ [`instant(I)`], A **occurs-at** I **with-prob** P in $\mathcal{L}^{ne}$ is the reduct of p-proposition A **performed-at** I **with-prob** P^+ **if-believes** $(\phi, [X, Y])$ [`occurs(TR, A, I, P)` and `performed(A, I, P, Precondition, "[", X, Y, "]"` considered together, where we let ϕ be the formula identified by `Precondition`], $B = \tilde{M}_D^{\mathcal{N}}(\phi, [H]_{<I})$, $C = \sum_{W \in \mathcal{W}} \tilde{M}_D^{\mathcal{N}}(W, [H]_{<I})$ and $D = \frac{B}{C}$ [`D = @div(B,C)`]." Therefore (see Definition 2.37), this constraint is checking whether p-proposition (A **performed-at** I **with-prob** P^+ **if-believes** $(\phi, [X, Y])$) was transformed into the corresponding o-proposition in the reduct $\mathcal{N}$ encoded by the answer set Z being considered. If this fails to satisfy the condition $D \in [X, Y]$ [`not @in(@exp(D),LeftBracket,@exp(X),@exp(Y),RightBracket)`], then Z is not an answer set of the program Π_D.

Similar reasoning can be applied to the other constraint:

```
:- trace(TR), action(A), instant(I), not occurs(TR, A, I, P),
   performed(A, I, P, Precondition, "[", X, Y, "]"),
   B = #sum{ E,OTHER_TR :
             sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
             holds(OTHER_TR, (Precondition,I) ),
             evalTraceNarrative(OTHER_TR,E) },
   C = #sum{ F,OTHER_TR :
             sensingIndistinguishableUpTo(TR,OTHER_TR,I-1),
```

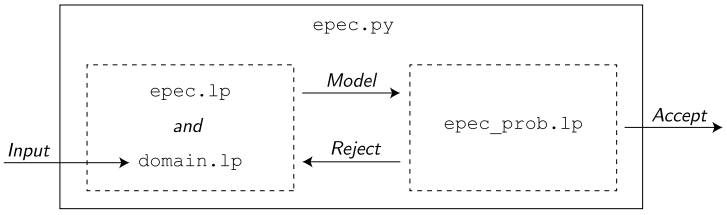

Fig. 6. A simple diagram of the EPEC modules. A python wrapper `epec.py` grounds the domain dependent part `domain.lp` together with the "logical" part of the domain independent program, `epec.lp`. The solver then produces one stable model at a time. Each stable model is passed to the "probabilistic" part, `epec_prob.lp`, to generate the probabilities needed to evaluate the fixed point condition. This process will either result in a model returned as an answer set, or cause the python wrapper to cycle back to the next model of `epec.lp` + `domain.lp`.

```
            evalTraceNarrative(OTHER_TR,F) },
D = @div(B,C),
@in(D,"[",@exp(X),@exp(Y),"]",).
```

that deals with the complementary unwanted case of a p-proposition not being reduced, but with the corresponding belief falling in the appropriate interval. These constraints filter out h-traces that are not consistent with any well behaved h-world w.r.t. the domain description $\mathcal{D}$ (see Definition 2.41). Given that all predicates are correct with respect to their corresponding counterparts, it follows directly that the derivation of b-propositions as described in Section 3.3 is also correct, since we have guaranteed that only well-behaved h-worlds give their contribution to the sums that allow one to calculate B_i's and P_i's in the b-proposition. □

4. Implementation

In this section we describe how the two sub-modules of the domain independent part of EPEC interact with each other and with the domain dependent description. An overview of the architecture is shown in Fig. 6.

Dealing with probabilities within Clingo is potentially problematic, since encoding probabilities as strings can result in a huge grounding cost for building every expressible fractional probability. To avoid this, our architecture is such that only probabilities of interest are instantiated and used in the full grounding process.

Recall that the domain independent program is split into two modules, `epec.lp` and `epec_prob.lp`. Similarly to the implementation in [24], the first module, described in Section 3.2.1, implements logical reasoning in the domain and returns a set of h-traces that are logically consistent with the domain description without performing any probabilistic operation on them. The second module, described in Section 3.2.2, then implements another domain independent answer set program for probabilistic reasoning which uses a simple Python library to deal with log probabilities (see Fig. 6). Although in theory the two modules could be merged in a single file this would result in extreme inefficiency. This is because the fixpoint condition would have to be checked against several nonsensical groundings, for example, ones that contain h-worlds that do not satisfy persistence. In particular, the separation of the EPEC implementation into logical and probabilistic sub-modules is necessary because the probabilistic sub-module handles probabilities using Python, rather than directly in `clingo`. This means that most ASP-specific optimizations are not available for the probabilistic sub-module, which can potentially lead to inefficiency in the computation.

The optimisation resulting from this architecture gives rise to a practical computational tool for accurate probabilistic, epistemic reasoning about reasonably sized domains such as Example 1.1 and the medical example from [15], which would be hard for a human to evaluate in an unsupported way. On a standard laptop (an Apple M1 MacBook Pro with 16 GB of memory), generating the answer set for these two domains takes approximately 1.31 and 1.63 seconds respectively. The third example hard-wired into our EPEC web interface (see Section 5) is an abstract example illustrating each type of EPEC proposition in its most general form. It uses one environmental action, one agent action, one Boolean-valued and one three-valued fluent, an i-proposition with two initial states, a c-proposition, two s-propositions, one o-proposition and one p-proposition. Both the o- and the p-proposition have a non-unary probability associated with their occurrence. This makes it a complex domain for non-supported human reasoning. The answer set for this domain takes approximately 1.12 seconds to generate in Clingo. Computation remains feasible for modest increases in the complexity of this abstract domain (for example by increasing the number of possible values of the non-Boolean fluent, or adding an extra p-proposition or o-proposition). See the remarks in Section 6 below for alternative approximate and/or restricted computation techniques we envisage for significantly larger domains.

5. Web interface

A web-based GUI for EPEC is available at https://www.ucl.ac.uk/infostudies/epec/translator.html. This provides several utilities. (i) It provides a specialised editor allowing users to define and display new domains, with associated syntax and arithmetical checks on the user's input. (ii) It contains some pre-defined domains (such as Example 1.1 and the medical example from [15]) that can be loaded into the editor and then modified and/or experimented with. (iii) It provides automatic translation of domains loaded into

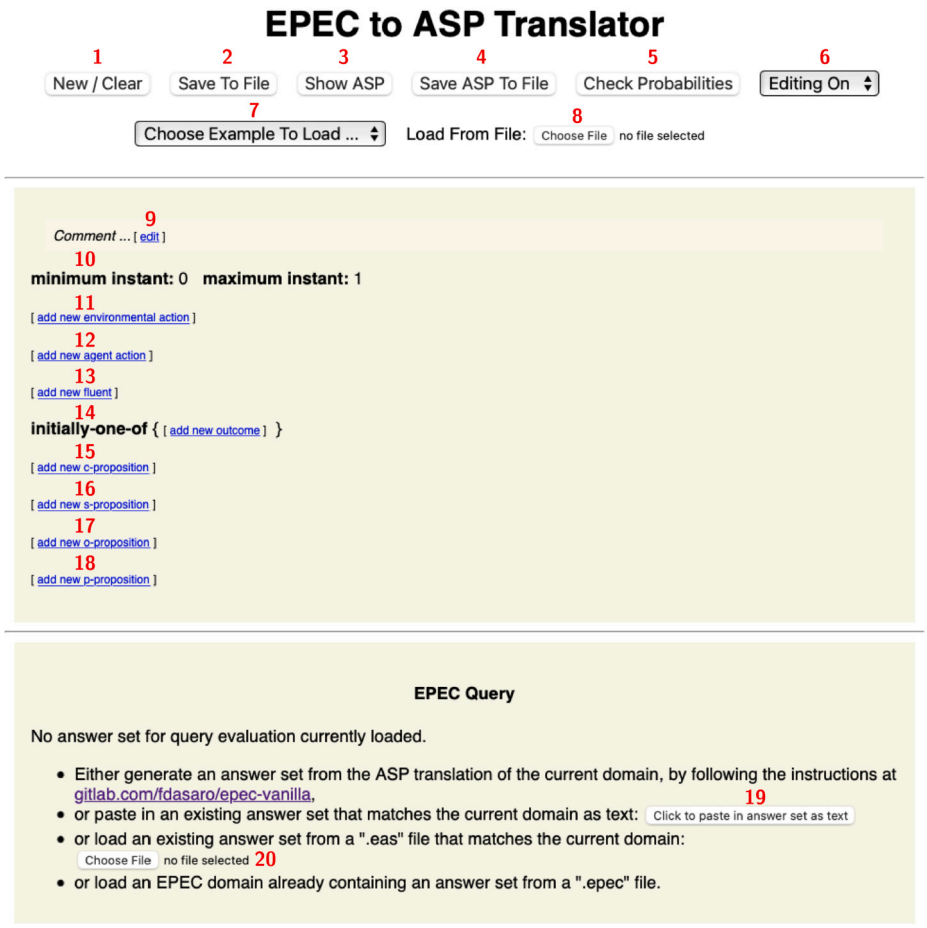

Fig. 7. The web interface for EPEC. Buttons and links allow the user to: (1) create a new domain description, (2) save the domain description to a file, (3) display the ASP translation of the domain, (4) save the ASP translation to a file, (5) check that probabilities are consistent, (6) switch between editing/non-editing modes, (7) load pre-defined examples, (8) load a domain description from the local machine, (9) enter a comment for the domain description, (10) define the minimum and maximum instants, (11) add new environmental actions, (12) add new agent actions, (13) add new fluents, (14) set the unique i-proposition in the domain, (15) add new c-propositions, (16) add new s-propositions, (17) add new o-propositions, and (18) add new p-propositions. Once the user has obtained the answer set there are two ways to load it: (19) paste it in via a text box, or (20) upload a file.

the editor into ASP, as described in Section 3.1. (iv) It provides a facility for the answer set associated with the currently loaded domain (generated offline by running the publicly available EPEC ASP package in Clingo) to be uploaded into the interface. (The pre-defined domains already include their associated answer sets.) (v) Once an answer set has been uploaded, it provides a query interface able to generate and display b-propositions (see Figs. 1 and 2) using an in-built Prolog and JavaScript based post-processing procedure.

Fig. 7 shows the web interface when first accessed. At the top of the web page are buttons and menus for creating new domains, clearing the current domain, loading an in-built example, and saving and loading domains and their ASP translations to and from files. The editing links in the area below allow the user to specify the sorts $\mathcal{F}$, $\mathcal{A}_e$, $\mathcal{A}_a$, and $\mathcal{V}$, as well as the i-propositions, c-propositions, s-propositions, o-propositions and p-propositions in the domain. Once editing is completed, the web interface can automatically check that the probabilities are consistent before translating the domain into an ASP. The resulting ASP code can then be saved and, as mentioned above, be run on the user's local machine to generate an answer set.

At the bottom of the web page is the EPEC Query facility. Fig. 8 shows this once an answer set has been loaded. The user is now able to enter a query in the form of the first 'at' and 'believes' components of a b-proposition. Pressing the 'query' button will then cause the interface to generate the remainder of the b-proposition, as shown in Fig. 9. The generated b-proposition can then be 'filtered' inline with a given activity report, using the middle button below the displayed b-proposition.

The web interface is written in JavaScript, using a JavaScript library called TauProlog (http://tau-prolog.org/) to handle queries and facilitate the display of the resulting b-propositions. Our future development plans include adding a facility for online generation of answer sets (via server-side processing), as an alternative to the user generating them with a local copy of Clingo and the EPEC system.

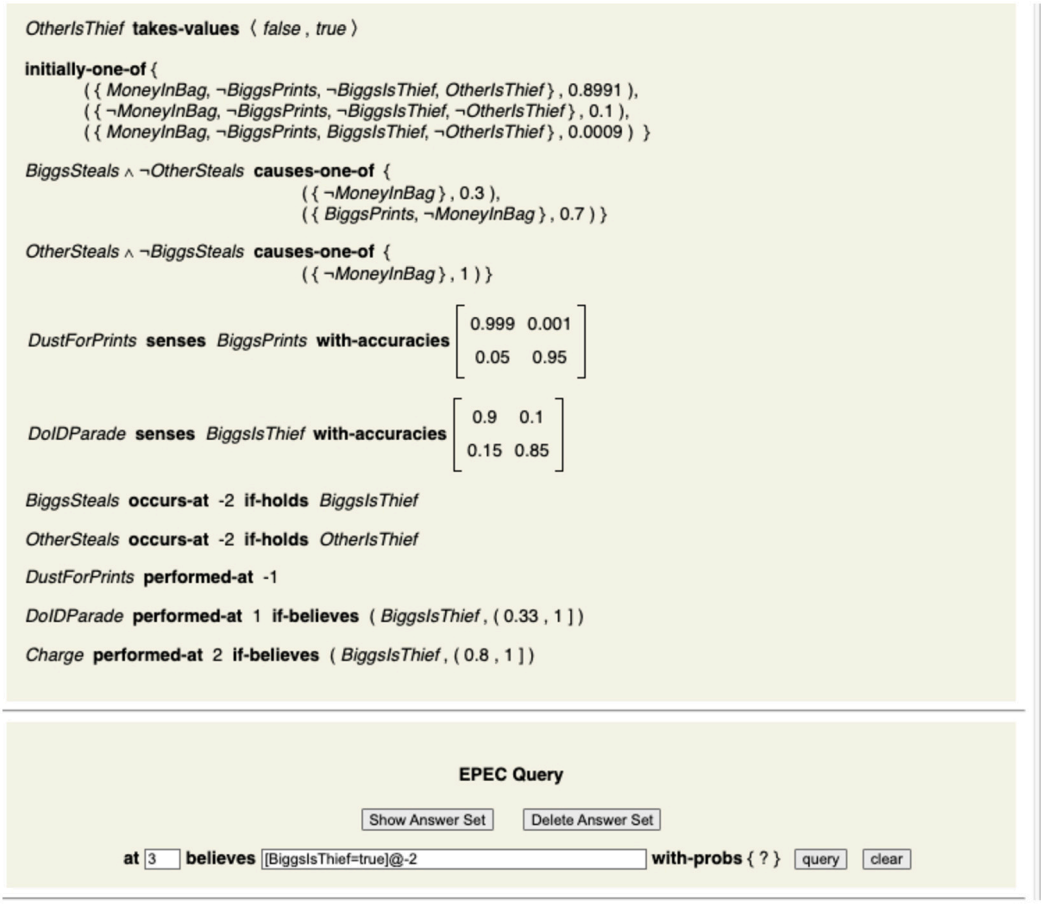

Fig. 8. The EPEC web interface query box.

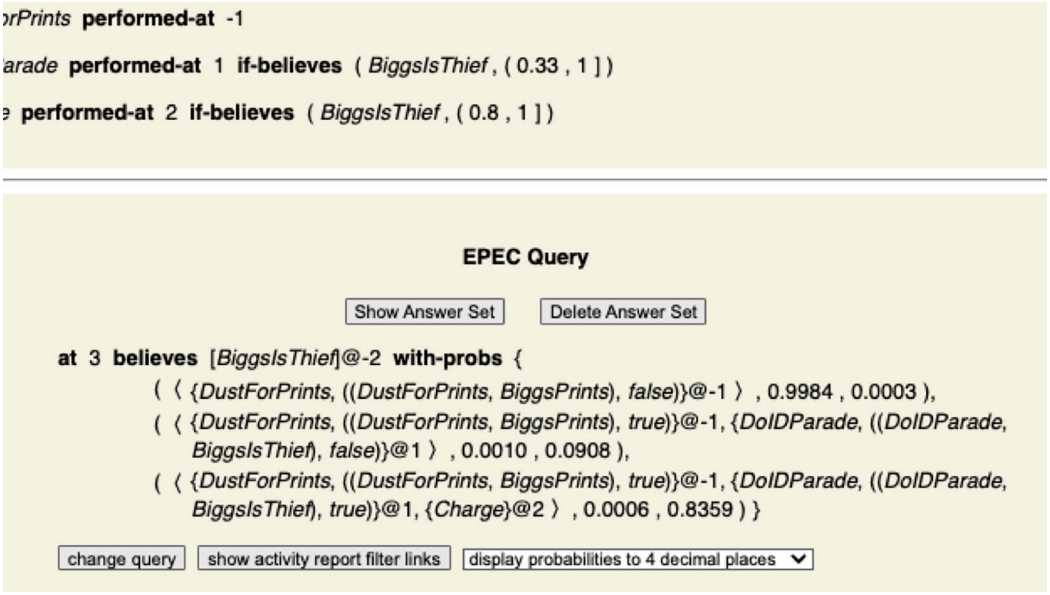

Fig. 9. A b-proposition displayed in the EPEC web interface.

Table 1
A 'feature chart' of frameworks for Reasoning About Actions and some of the domain features they support. The double checkmark (✓✓) indicates that EPEC can model actions whose epistemic precondition incorporates strengths of belief. The triple checkmark (✓✓✓) indicates that BHL can deal seamlessly with discrete and continuous probability distributions. The checkmark with star (✓*) indicates that Modular-E is both 'modular' and elaboration tolerant and argues the inseparability of these properties.

	Modular-E	EFEC	BHL	PSC	PAL	Language E+	MLN-EC	Prob-EC	PEC	EPEC
Classical logic formalism		✓	✓	✓						
Action language	✓				✓	✓			✓	✓
Supports narratives	✓	✓			✓		✓	✓	✓	✓
Functional fluents		✓	✓	✓					✓	✓
Elaboration tolerant	✓*	✓			✓		✓	✓		
Supports ramifications	✓				✓	✓				
Supports qualifications	✓					✓				
Probabilistic			✓	✓	✓	✓	✓	✓	✓	✓
Epistemic		✓	✓	✓		✓				✓
Concurrent actions	✓	✓					✓	✓	✓	✓
Triggered actions		✓							✓	✓
Imperfect sensing			✓	✓						✓
Reasoning about knowledge of past, present and future		✓								✓
Belief conditioned actions		✓				✓				✓✓
Supports continuous probability distributions			✓✓✓	✓						
Supports only knowing			✓							
Mixes probabilities and non-determinism			✓	✓		✓				

6. Conclusions and related work

6.1. Related work

There are several other languages for Reasoning About Actions that also support some form of uncertain reasoning. We compare these, together with EPEC, in Table 1 in terms of the domain features they support. Although our focus is on probabilistic reasoning, Table 1 also lists two non-probabilistic languages that have influenced the development of EPEC, namely Modular-$\mathcal{E}$ [21] and EFEC [24], since they include features such as the ability to reason about knowledge of past, present and future, and support for resolving conflicts between concurrent actions.

BHL [2] is a cornerstone of early work integrating probabilistic degrees of belief with logical aparatus for reasoning about actions, and has served as an inspiration for similar approaches (e.g. [26]). Although it does not support narrative reasoning and belief-conditioned actions, it has been extended and enriched with several other features not provided for by EPEC. For example, [5,9] add support for continuous (as well as mixed) probability distributions, and [8] applies BHL and these extensions to the problem of localisation, i.e. to the case where an agent moves in a (multi-dimensional) world and can sense its position. The problem of extending BHL with a modality known as 'only knowing', which allows for a precise specification of what is and what is not known within a logical theory of actions, has also been tackled in [10]. BHL was also implemented in the form of a regression system [6] and a golog-like programming model [7].

The Probabilistic Situation Calculus [26] ('PSC' in Table 1) is similar to BHL in that it is based on Reiter's Situation Calculus, and supports continuous probability distributions and observations. Its semantics is given in terms of randomly reactive automata and implemented in Mathematica.

PAL is an action-language representation for Markov Decision Processes. Although it is ontologically close to the Situation Calculus, it allows for a limited class of non-probabilistic narratives. Additionally, it incorporates a degree of elaboration tolerance – the end-user can define additional random variables alongside an existing theory provided these variables are probabilistically independent. PAL does not support sensing or epistemic reasoning, and so in these respects the planning-oriented action language $\mathcal{E}+$ [20] is an advance. $\mathcal{E}+$ supports both sensing and belief-conditioned actions, and the authors of [20] provide algorithms for the efficient computation of plans, which makes it an advanced tool for epistemic planning. However, sensing actions are always assumed to be perfect and so the association of a confusion matrix with its semantics is not supported.

The Event Calculus has not been enhanced with any form of probabilistic reasoning until more recently. To our knowledge, the first attempts to do so are MLN-EC [32] and the closely related Prob-EC [31]. These two languages give a probabilistic semantics to the Event Calculus, respectively using Markov Logic Networks [29] and ProbLog [16], a probabilistic variant of Prolog. Both

are based on a discrete-time reworking of the Event Calculus and applied to the task of event recognition, where a set of complex activities must be detected when a set of time-stamped short-term activities is received as input. They provide separate support for causal rules (MLN-EC) and probabilistic events (Prob-EC), which in EPEC are integrated together. These two frameworks have opened the way for other probabilistic extensions of the Event Calculus. For example, [27] extends Prob-EC to deal with uncertain event observations, uncertain effects, and uncertain composite event definitions. Recent implementations extend MLN-EC and Prob-EC with the possibility of learning Event Calculus Theories [22] and the ability to deal with the probability of events occurring in time-intervals rather than at specific instants [1]. These works have recently developed into demonstrably very efficient tools for composite event recognition such as [25].

The work in the current paper builds upon PEC ("Probabilisatic Event Calculus"), an action language we introduced in [14] for reasoning about action and change, and which we implemented in Clingo. PEC is notable for its support of triggered and concurrent actions, but cannot handle sensing or belief-conditioned actions. EPEC, as described here, was first introduced in [15] to overcome these limitations. We implemented a specific segment of EPEC in [17] to enable efficient and exact runtime reasoning. However, that implementation, which we call PEC-RUNTIME, has several restrictions, particularly as regards reasoning about the past and future. The implementation described in the current paper addresses these limitations, offering a practical and correct reasoning tool applicable across all EPEC domains.

The use of different underlying languages such as [32] and ProbLog [16] to implement MLN-EC and Prob-EC highlights the potential of leveraging various logical languages to implement the declarative semantics of EPEC. Among the alternatives, P-log [4] represents an early effort to merge the features of logic programming with probability theory, with the goal of representing both knowledge and uncertainty. Its design effectively bridges Probability and Logic Programming through Answer Sets. Another significant language is Plingo [19]. Plingo enhances Clingo by introducing native probabilistic capabilities. While it can be mapped onto LP^{MLN}, it can also serve as a front end for both P-log and ProbLog. Additionally, PASP [12] has the ability to amalgamate rules, facts, and independent probabilistic facts. PASP adopts a credal semantics, where each consistent program aligns with a credal set. This methodology offers a sturdy framework for calculating probabilities based on a given program, further expanding the realm of probabilistic answer set programming. Since all these languages are rooted in ASP, it is possible that EPEC could be seamlessly transitioned to them without compromising correctness, and potentially enhancing performance.

6.2. Summary and conclusions

We have presented an ASP implementation of EPEC, a state-of-the-art reasoning system for epistemic and probabilistic reasoning. The implementation is accompanied by a web GUI that facilitates users wanting to model their own domains. The semantics of EPEC is based on a probabilistic fixpoint construction that can be ported naturally to ASP, and is reflected by the implementation's internal architecture. This handles the computation using a two-step procedure, where the first step takes care of the purely qualitative aspect of the domain, filtering out logically inconsistent answer set choices, and the second step implements the fixpoint checks, using the probabilistic information built into the propositions that comprise the domain representation.

To the best of our knowledge, our implementation is the first Event Calculus implementation dealing with temporal, causal narratives having both an epistemic and a probabilistic aspect. The web GUI facilitates its use by non-experts and students, making this implementation a useful tool e.g. for educational purposes, or for supporting decisions involving medical tests and treatments.

In future, we intend to implement efficient approximate query answering for large domains by (i) extending our non-epistemic probabilistic implementation (available at https://github.com/dasaro/pec-anglican) based on Markov chain Monte Carlo (MCMC) methods so that it additionally deals with epistemic domains, (ii) exploiting advanced constructs in Clingo to find maximal probability answer sets, and (iii) building on ideas from [17], which transformed PEC into PEC-RUNTIME – an efficient architecture for runtime reasoning by limiting its inferential capabilities – and extending these optimizations to EPEC. We also aim to explore the intepretability of our model by allowing natural language input, translating its output inferences to natural language, and evaluating the result with humans involved in user studies. Our implementation could also be used as a shared ontology to collect and share data from, e.g., medical domains, where narratives involving multiple imprecise epistemic actions (e.g., testing for diseases, readings of MRI scans, etc.) and probabilistic interventions (e.g., taking medications, undergoing operations whose risks are quantifiabile, etc.) are omnipresent. Such uses of our tool could prove useful to increase the awareness and understanding of patients, e.g., by using it to annotate hard-to-understand medical records with explanations about actual probabilities of having contracted a disease given a positive test result, or risk levels of operations given a patient's history, or the probability of experiencing particular side-effects of medicines.

CRediT authorship contribution statement

Fabio Aurelio D'Asaro: Conceptualization, Software, Writing – original draft, Writing – review & editing, Methodology, Project administration. **Antonis Bikakis:** Conceptualization, Writing – original draft, Writing – review & editing. **Luke Dickens:** Conceptualization, Writing – original draft, Writing – review & editing. **Rob Miller:** Conceptualization, Project administration, Software, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The author Fabio Aurelio D'Asaro acknowledges the funding and support of PON "Ricerca e Innovazione" 2014-2020 (PON R&I FSE-REACT EU), Azione IV.6 "Contratti di ricerca su tematiche Green" in the context of the project titled "Il miglioramento dell'algorithmic fairness in ambito assicurativo: Un'analisi concettuale a partire da uno studio di caso".

References

[1] Alexander Artikis, Evangelos Makris, Georgios Paliouras, A probabilistic interval-based event calculus for activity recognition, Ann. Math. Artif. Intell. 89 (1) (2021) 29–52.
[2] Fahiem Bacchus, Joseph Y. Halpern, Hector J. Levesque, Reasoning about noisy sensors and effectors in the situation calculus, Artif. Intell. 111 (1–2) (1999) 171–208, https://doi.org/10.1016/S0004-3702(99)00031-4.
[3] Chitta Baral, Nam Tran, Le-Chi Tuan, Reasoning about actions in a probabilistic setting, in: AAAI/IAAI, 2002, pp. 507–512.
[4] Chitta Baral, Michael Gelfond, Nelson Rushton, Probabilistic reasoning with answer sets, Theory Pract. Log. Program. 9 (1) (2009) 57–144.
[5] Vaishak Belle, Hector Levesque, Reasoning about continuous uncertainty in the situation calculus, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, AAAI Press, ISBN 978-1-57735-633-2, 2013, pp. 732–738, http://dl.acm.org/citation.cfm?id=2540128.2540235.
[6] Vaishak Belle, Hector Levesque, Prego: an action language for belief-based cognitive robotics in continuous domains, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 28, 2014.
[7] Vaishak Belle, Hector Levesque Allegro, Belief-based programming in stochastic dynamical domains, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
[8] Vaishak Belle, Hector J. Levesque, A logical theory of localization, Stud. Log. 104 (4) (2016) 741–772.
[9] Vaishak Belle, Hector J. Levesque, Reasoning about discrete and continuous noisy sensors and effectors in dynamical systems, Artif. Intell. 262 (2018) 189–221.
[10] Vaishak Belle, Gerhard Lakemeyer, Hector J. Levesque, A first-order logic of probability and only knowing in unbounded domains, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016, pp. 893–899, http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12271.
[11] Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag, Berlin, Heidelberg, ISBN 0387310738, 2006.
[12] Fabio Gagliardi Cozman, Denis Deratani Mauá, The joy of probabilistic answer set programming: semantics, complexity, expressivity, inference, Int. J. Approx. Reason. 125 (2020) 218–239, https://doi.org/10.1016/j.ijar.2020.07.004.
[13] Fabio Aurelio D'Asaro, Probabilistic epistemic reasoning about actions, PhD thesis, UCL, University College London, 2019.
[14] Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, Rob Miller, Foundations for a probabilistic event calculus, in: LPNMR, 2017, pp. 57–63.
[15] Fabio Aurelio D'Asaro, Antonis Bikakis, Luke Dickens, Rob Miller, Probabilistic reasoning about epistemic action narratives, Artif. Intell. 287 (2020) 103–352, https://doi.org/10.1016/j.artint.2020.103352, ISSN 0004-3702, http://www.sciencedirect.com/science/article/pii/S0004370219300906.
[16] Luc De Raedt, Angelika Kimmig, Hannu Toivonen Problog, A probabilistic prolog and its application in link discovery, in: IJCAI, vol. 7, 2007, pp. 2462–2467.
[17] Fabio Aurelio D'Asaro, Luca Raggioli, Salim Malek, Marco Grazioso, Silvia Rossi, An application of a runtime epistemic probabilistic event calculus to decision-making in e-health systems, Theory Pract. Log. Program. (2022) 1–24.
[18] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Multi-shot ASP solving with clingo, CoRR, arXiv:1705.09811 [abs], 2017.
[19] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Rühling, Torsten Schaub, Plingo: a system for probabilistic reasoning in clingo based on LP^{MLN}, in: International Joint Conference on Rules and Reasoning, Springer, 2022, pp. 54–62.
[20] Luca Iocchi, Thomas Lukasiewicz, Daniele Nardi, Riccardo Rosati, Reasoning about actions with sensing under qualitative and probabilistic uncertainty, ACM Trans. Comput. Log. 10 (1) (2009).
[21] Antonis Kakas, Loizos Michael, Rob Miller, $\mathcal{M}$odular-$\mathcal{E}$ and the role of elaboration tolerance in solving the qualification problem, Artif. Intell. 175 (1) (2011) 49–78, https://doi.org/10.1016/j.artint.2010.04.008, ISSN 0004-3702.
[22] Nikos Katzouris, Georgios Paliouras, Alexander Artikis, Online learning probabilistic event calculus theories in answer set programming, Theory Pract. Log. Program. (2021) 1–25, https://doi.org/10.1017/S1471068421000107.
[23] Vladimir Lifschitz, Hudson Turner, Splitting a logic program, in: ICLP, vol. 94, 1994, pp. 23–37.
[24] Jiefei Ma, Rob Miller, Leora Morgenstern, Theodore Patkos, An epistemic event calculus for ASP-based reasoning about knowledge of the past, present and future, in: Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR-19, 2013.
[25] Periklis Mantenoglou, Alexander Artikis, Georgios Paliouras, Online event recognition over noisy data streams, Int. J. Approx. Reason. (2023) 108993.
[26] Paulo Mateus, Antonio Pacheco, Javier Pinto, Amilcar Sernadas, Cristina Sernadas, Probabilistic situation calculus, Ann. Math. Artif. Intell. 32 (1/4) (January 2001) 393–431, ISSN 1012-2443.
[27] Kevin McAreavey, Kim Bauters, Weiru Liu, Jun Hong, The event calculus in probabilistic logic programming with annotated disjunctions, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS '17, Richland, SC, in: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 105–113, http://dl.acm.org/citation.cfm?id=3091125.3091146.
[28] Jeff Paris, The Uncertain Reasoner's Companion: a Mathematical Perspective, vol. 39, Cambridge University Press, 2006.
[29] Matthew Richardson, Pedro Domingos, Markov logic networks, Mach. Learn. 62 (1) (Feb 2006) 107–136, https://doi.org/10.1007/s10994-006-5833-1, ISSN 1573-0565.
[30] Taisuke Sato, A statistical learning method for logic programs with distribution semantics, in: Logic Programming: The 12th International Conference, MIT Press, 1995, pp. 715–729.
[31] Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, Georgios Paliouras, A probabilistic logic programming event calculus, Theory Pract. Log. Program. 15 (2015) 213–245, https://doi.org/10.1017/S1471068413000690, ISSN 1475-3081, http://journals.cambridge.org/article_S1471068413000690.
[32] Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, George A. Vouros, Probabilistic event calculus for event recognition, ACM Trans. Comput. Log. 16 (2) (2015) 11.