Probabilistic Situation Calculus

P. Mateus, A. Pacheco, J. Pinto, A. Sernadas and C. Sernadas

Abstract

In this article we propose a Probabilistic Situation Calculus logical language to represent and reason with knowledge about dynamic worlds in which actions have uncertain effecs. Uncertain effects are modeled by considering that actions can be subdivided into a deterministic input and a probabilistic reaction. We assume that the probabilities of the reactions have known distributions.

Our logical language is an extension to Situation Calculae in the style proposed by Raymond Reiter. There are three aspects to this work. First, we extend the language in order to accommodate the necessary distinctions (e.g., the separation of actions into inputs and reactions). Second, we develop the notion of Randomly Reactive Automata in order to specify the semantics of our Probabilistic Situation Calculus. Finally, we develop a reasoning system in Mathematica capable of performing temporal projection in the Probabilistic Situation Calculus.

Keywords: Probability Logic, Probabilistic Automata, Situation Calculus, Theories of Action, Mathematica.

1 Introduction and Motivation

In this article we address the problem of representing and reasoning with theories of action in domains in which actions might have non-deterministic probabilistic outcomes. This problem has been addressed in the literature by various researchers, for instance [1, 7]. In previous work, reported in [4], we deal with actions that can have finitely many outcomes, and, therefore, a discrete probability distribution. The assumption that the possible outcomes is finite is very strong and often inadequate. For instance, if one is modeling the sensors and effectors of a robotic system, one normally deals with physical parameters of a continuous type. For example, if the robot instructs one of its legs to move one meter ahead, the result will be that the leg will end up one meter ahead plus some error. This error is usually modeled as having a continuous distribution.

Given that the consideration that finitely many outcomes for a probabilistic action is too stringent, we set out to extend our previous work to consider domains in which the set of outcomes for an action could be discrete, continuous, or even mixed. Our work has been developed in the context of the Situation Calculus [2, 9] with ontological extensions to deal with non-deterministic actions [4]. Some important features of the Situation Calculus [1] logical language (derived from [9]) and its semantics are specified below.

- The Situation Calculus is a language of second order logic intended to specify models of dynamic worlds, in which changes are the result of actions.
- There is an initial situation, called S_0, that is considered to be the starting point in time. All possible world developments start here.
- Situations are objects in the domain of discourse, they denote the development of actions that lead from S_0 to it.
- The state of the world at each situation is described by a set of properties, which are called *fluents*. In some Situation Calculus based languages fluents are represented with predicates that have one situation argument; alternatively, fluents can be objects in the domain of discourse. In the latter case, a special sort for fluents is introduced.
- As proposed by Reiter [9] the frame problem[2] is resolved by replacing the axioms for direct effects and the state constraints with a single set of *successor state axioms*.
- Actions are objects of the domain of discourse. This is an important advantages over modal logic approaches in which actions are represented by a fixed set of modal operators.
- Theories of action are written following the structure of theories proposed by Raymond Reiter [10]. That is, a theory of action contains:
  - A description of the initial situation.
  - A set of axioms describing the direct effects of actions.
  - A set of state constraints, describing the constraints that have to hold true in every world situation.
  - A set of precondition axioms, which are used to describe the necessary properties that must hold for a given action to be executable.

The main task we would like our logical language and reasoning mechanism to support is *probabilistic temporal projection*. In this context, it is the ability to evaluate the probability that a certain logical formula be true after executing a sequence of *inputs* (described below).

[1]Keep in mind that here we refer to Situation Calculus as a language that evolved from the language of McCarthy and Hayes, but that it departs in some important ways. A careful study of the differences goes beyond the scope of this article.

[2]The problem of finding a succinct representation for the non-effects of an action, is usually called *frame problem*.

This task is essential in many applications; for instance, in planning with uncertainty, one should be able to provide estimates for the probability of success for various possible plan choices. Also, one might want to verify that plans provided to a system obey certain rules. For example, safety rules that establish limits to the risks taken by an agent. Important features that we have added to the Situation Calculus to obtain the Probabilistic Situation Calculus can be summarized as follows:

- Based on [4], non-deterministic actions are decomposed into *input* (agent action), and *reaction* (nature's random response). Given an input, the set of outcomes has an associated probability distribution.
- Given an input, or agent action, the set of reactions can be infinite, maybe non-discrete. Given a situation, and an input, the set of reactions generates a set of successor situations with an induced probability distribution.
- In order to specify the semantics for the Probabilistic Situation Calculus with continuous reactions we introduce Randomly Reactive Automata, in which actions are taken to be sequences of input/reaction pairs. A probability space is defined on the situations that result for input/reaction pairs, when performed in a given situation.

The main departure of our version of the Situation Calculus from the non probabilistic Situation Calculus is the way in which action preconditions are specified. Basically, a precondition is now divided into two separate components. First, we have the specification of preconditions for *inputs*. Furthermore, we need to specify a probability distribution for the reactions obtained after an input is processed.

The article is divided in two main sections. First, in section 2 we study the simple case in which the reactions to an input are univariate (unidimensional); secondly, in section 3 we study the more general case in which reactions can be multivariate (multidimensional). The unidimensional case is presented separately to facilitate the exposition.

Both sections have the same overall structure. First, a presentation of the basic concepts. Then, the extensions required for the Situation Calculus. That is, we present the logical language of the Probabilistic Situation Calculus with integrated continuous and discrete distributions. For the presentation of the logical language, we assume standard interpretations for the real numbers, natural numbers, etc. Furthermore, we also assume a fixed standard interpretation for common operators for real numbers and probabilities. The reasoning capabilities required for the standard sorts and operations are also assumed to be provided by an oracle.

Later on, we present an approach for reasoning with theories of the Probabilistic Situation Calculus in which the logical specifications are encoded as rewrite rules of Mathematica [11]. The reasoning mechanism utilized combines symbolic and numeric reasoning. In particular, the rewrite rules are used in order to reason about the actions performed and their results. Analytical and numerical methods might be used to reason with the distributions of the outcomes. Unfortunately, the analytical approach is severely limited because it becomes mathematically intractable when dealing with sequences of more than two or three inputs. On the other hand, numeric methods are computationally infeasible due to the high dimensionality of the integrations reauired. Instead, we use a Monte–Carlo approach which proves to be very effective, yielding good results in few steps.

Finally, we provide a semantic account for the Probabilistic Situation Calculus. This is done by introducing the concept of a Randomly Reactive Automata. The notion of a randomly reactive automaton is essential for our specification of the semantics of the Probabilistic Situation Calculus with continuous and discrete distributions.

After the two main sections, we present our final remarks, conclusions, and discuss briefly the future directions in which this research can be extended.

2 Probabilistic Situation Calculus: Univariate Case

2.1 Basic Concepts

The Probabilistic Situation Calculus is inspired by the original conception of the Situation Calculus of McCarthy and Hayes [2] and by Raymond Reiter's approach to dealing with the frame problem [9]. Reiter's approach to the frame problem is based on the idea that all the effects of actions on a single fluent can be compiled together in a single *successor state axiom*. Thus, for each fluent, we have a successor state axiom that describes necessary and sufficient conditions for each fluent to be true after actions are performed.

The Probabilistic Situation Calculus [3] is a second order sorted many-sorted language. In the Situation Calculus we use in this article, we incorporate three types of sorts. Proper Situation Calculus sorts (e.g., sorts for actions and situations), standard data type sorts (real numbers, natural numbers, etc.), plus domain specific *data* sorts (e.g., positions in space, physical objects, etc.). Furthermore, we assume that the interpretation for the standard data type sorts is fixed and do not axiomatize them.

[3] For simplicity, in the future we refer to the Probabilistic Situation Calculus as Situation Calculus.

2.2 Language

2.2.1 Probabilistic Situation Calculus Actions.

An essential aspect of our approach to modeling nondeterministic and uncertain actions is the decomposition of *primitive actions* into two elements: A *deterministic* component and a *non-deterministic* component[4]. The *deterministic* component is the choice that the agent makes regarding what to do; e.g., the agent may decide to flip a coin. In this article, the deterministic component is called *input*. On the other hand the *non-deterministic* component corresponds to nature's *reaction* to the agent's *input*. As discussed later, the reactions will be taken to be real numbers with some probability distribution. For example, after an agent flips a coin, nature's reaction might be a 1 (with probability 0.5), which we might interpret as *heads*, or it might be a -1 (with probability 0.5), which we might interpret as *tails*, or it might be any other real number (with probability 0).

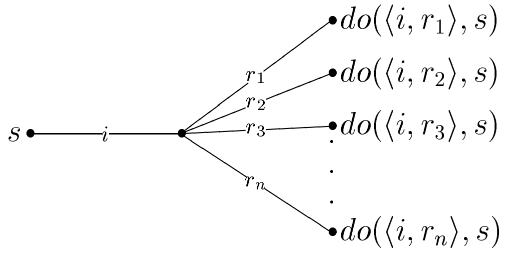

Figure 1: Input and possible reactions

Figure 1 illustrates the fact that to one input i there might be many possible reactions $r_1 \ldots r_n$. The figure shows a situation in which n different reactions might arise to a given input. Each combination of the input i with some reaction r_k gives rise to a different action. Therefore, a composition of i with a reaction r_k leads to an action ir_k. Contrary to what figure 1 suggests, notice that the set of possible reactions[5] to a given input does not need to be finite or even discrete.

As discussed below, the language of the Situation Calculus will be extended with a sort for inputs (reactions are just real numbers) along with suitable operators for constructing actions out of inputs and reactions.

2.2.2 Probabilistic Situation Calculus Sorts

The following are standard sorts in the Situation Calculus tradition, with the possible exception of the sort $\mathcal{F}$ for fluents (in some cases fluents are predicates with one argument of type $\mathcal{S}$).

[4] In [3], these were called "behavior" and "outcome" respectively.

[5] A *possible* reaction is one that has a non-zero probability density of occurring.

- Actions, $\mathcal{A}$; an action corresponds to the standard notion of action in the Situation Calculus. I.e., an instantaneous primitive action.
- Situations, $\mathcal{S}$; at any moment in time, the world is conceived as being in a state that was arrived at by performing a sequence of actions in a *starting situation.* A situation identifies a state and a history of actions from the starting situation.
- Fluents, $\mathcal{F}$; fluents are *dynamic* properties that may change from one situation to another. In the language, fluents are reified. Thus, there are fluent terms that denote fluents which will hold or not hold at any given situation.

We add sorts for *inputs* and *input sequences* to the language:

- Inputs $\mathcal{I}$, which correspond to agent choices, as described above. Given an input, there will be a nature's reaction (chosen from a set of possible reactions).
- Input sequences $\mathcal{I}^*$; correspond to sequences of agent choices (e.g., flip a coin several times in succession).

Since random reactions are simply real numbers, we do not require a special sort in the language to represent them. We assume that the following standard sorts are available. Furthermore, we assume that their interpretation is the standard one, and that this interpretation is fixed.

- Reals, $\mathcal{R}$, interpreted as the reals (the random reactions belong to this sort); natural numbers $\mathcal{N}$, etc.
- Extended reals, $\overline{\mathcal{R}} = \mathcal{R} \cup \{+\infty, -\infty\}$.
- Probabilities, $\mathcal{P}$, corresponding to the real interval $[0, 1]$.

Finally, we have data sorts, $\mathcal{D}$, etc., which correspond to the sorts introduced for any particular application.

2.2.3 Operations

Operations for Reals:
Aside from Probabilistic Situation Calculus specific operations, we need operations to deal with the real numbers, probability distributions, etc. We will assume that we have at our disposal standard data type operations for the $\mathcal{R}$ and $\mathcal{P}$ sorts. Furthermore, we assume that we have an oracle, capable of evaluating expressions involving reals and operations over them. As discussed later, a practical realization of the oracle is the Mathematica software system[11].

As examples of operations for elements and functions of sorts $\mathcal{R}$, $\overline{\mathcal{R}}$ and $\mathcal{P}$, we have:

- $+\infty$ and $-\infty$ of type $\overline{\mathcal{R}}$, denoting themselves (fixed denotation).
- $bernoulli : [0,1] \to [\mathcal{R} \to \mathcal{P}]$. Given a parameter value μ in the range $[0,1]$, the term $bernoulli(\mu)$ denotes the Bernoulli distribution function with expected value μ.
- $exponential : \mathcal{R}^+ \to [\mathcal{R} \to \mathcal{P}]$. Also, given a positive real parameter μ, the term $exponential(\mu)$ denotes the exponential distribution function with expected value μ.
- $sin : \mathcal{R} \to \mathcal{R}$. We assume that an assortment of standard mathematical functions are available, such as the trigonometric functions.
- $lim : [\mathcal{R} \to \mathcal{R}] \times \overline{\mathcal{R}} \times \mathcal{R} \to \mathcal{R}$. This is the limit function. The lim function takes three arguments: First, a one real argument function (call it f, the function for which one wants the limit); second, the value to which the argument of f approaches to; and third, the direction of approach (less than zero is from the left, and from the right otherwise). For example, the expression[6]:

  $$\lim_{y \to x^-} \frac{exp(x) - exp(y)}{x - y} > 0. \tag{1}$$

  should be formally written as:

  $$lim(\lambda y. \frac{exp(x) - exp(y)}{x - y}, x, -1) > 0. \tag{2}$$

  assuming that we accept infix notation for the basic arithmetic operations on the reals. In the rest of the paper, we use notation such as the one used in (1), with the understanding that this expression is syntactic sugar for the corresponding formal notation, such as (2).
- Other operators for real functions. In particular integrals and derivatives. For example, we take the $'$ function:
  $' : [\mathcal{R} \to \mathcal{R}] \to [\mathcal{R} \to \mathcal{R}]$ that takes a function from the reals to the reals and returns another function from the reals to the reals.

Thus, we assume that the oracle at our disposal will be able to evaluate usual functions (exponential, logarithm, etc.) as well as be able to evaluate complex expressions involving limits, derivatives, integrals, etc.

[6] Here $exp(x)$ corresponds to e^x

Probabilistic Situation Calculus Operations

The standard operators of the Situation Calculus:

- S_0: $\mathcal{S}$. In the standard Situation Calculus language, the constant S_0 denotes the *starting* situation. Following Reiter's approach [9], all situations arise from performing sequences of actions starting at S_0.
- *do*: $\mathcal{A} \times \mathcal{S} \to \mathcal{S}$. Give an action term a and a situation term s, the term $do(a, s)$ denotes the situation that results from performing the single primitive action a in s.
- *holds* $\subseteq \mathcal{F} \times \mathcal{S}$. Fluents are objects that denote dynamic world properties. If f is a fluent term, and s a situation term, then the expression $holds(f, s)$ states that the fluent denoted by f holds in the situation denoted by s.
- *Poss* $\subseteq \mathcal{A} \times \mathcal{S}$. As Reiter, we use the *Poss* predicate to signify that some action is possible (the preconditions for its execution hold) in some situation.

The following operators extend the standard operators in order to accommodate the new ontology for non-deterministic actions.

- *iposs* $\subseteq \mathcal{I} \times \mathcal{S}$. Analogous to *Poss*, the *iposs* predicate holds of an input and a situation whenever the input is possible in the corresponding situation.
- *rposs* $\subseteq \mathcal{R} \times \mathcal{I} \times \mathcal{S}$. If r, i and s are real, input and situation terms respectively, then $rposs(r, i, s)$ will hold when the reaction r is possible after performing input i in situation s.
- $\langle \_, \_ \rangle$: $\mathcal{I} \times \mathcal{R} \to \mathcal{A}$. Actions are decomposed into two components (as discussed in section 2.2.1): inputs and reactions. Given an input term i and a reaction term r, the term $\langle i, r \rangle$ denotes the action that results from taking the reaction denoted by r after the input denoted by i.
- $\_{_1}$: $\mathcal{A} \to \mathcal{I}$. Given that actions are decomposed in an input and a reaction, the subscript 1 is used to extract the input of an action. Thus, if a is an action term, then a_1 is the input component of this action[7].
- $\_{_2}$: $\mathcal{A} \to \mathcal{R}$. If a is an action term, then a_2 is the reaction component of this action.
- *cdf*: $\mathcal{I} \times \mathcal{S} \to [\mathcal{R} \to \mathcal{P}]$. Given a situation term s and some input term i, the term $cdf(i, s)$ denotes a *cumulative distribution function* for the reaction to input i in situation s; i.e., a function from the reals (possible reactions) into the interval $[0, 1]$ (probabilities).

[7] Notice that, unless we assume the operators $\_{_1}$ and $\_{_2}$ to be partial, our language forces all actions to have a deterministic and a non-deterministic components. Of course, one can trivially obtain purely deterministic actions if one restricted the possible reactions to a single one.

- *prob*: $\mathcal{F} \times \mathcal{I}^* \times \mathcal{S} \rightarrow \mathcal{P}$. Given a fluent, an input sequence $\vec{i}$, and a situation s, *prob* yields the probability that after performing the input sequence $\vec{i}$ starting in s will lead to a situation in which the fluent holds.

2.3 Axiomatization

In the axiomatization below we assume that all free variables in formulas are universally quantified with maximum scope.

The axiomatization includes a data type theory DTT, which includes axioms for the real numbers (e.g., to establish commutativity of addition), natural numbers, etc.

Also, we need Situation Calculus axioms[8]:

Foundational Axioms :

$$(\forall \varphi)\ [\varphi(S_0) \land (\forall s, a)\ (\varphi(s) \supset \varphi(do(a, s)))] \supset (\forall s)\ \varphi(s), \tag{3}$$
$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \land s_1 = s_2, \tag{4}$$
$$\neg s < S_0, \tag{5}$$
$$s < do(a, s') \equiv Poss(a, s') \land s \leq s'. \tag{6}$$

Axiom (3) is a situation existence axiom, which says that, aside from S_0, no situation exists unless it is reachable from S_0 by performing actions. Axiom (4) is a uniqueness of names axiom for situation, and axioms (5) and (6) define a reachability relation for situations. For more details on this axiomatization, the reader should consult [6][9].

Also, we introduce the predicate *legal* for situations as a shorthand (which will be used in the later encoding of Situation Calculus theories in Mathematica):

$$legal(s) \equiv S_0 \leq s. \tag{7}$$

Aside from the axioms (3)-(6), we need to introduce a new axiom which characterizes the state of situations that are not reachable by performing actions in legal situations:

$$\neg legal(do(a, s)) \supset (\forall f)\ holds(f, s) \equiv holds(f, do(a, s)). \tag{8}$$

This axiom is not present in the theories of action written in Reiter's style. Without this axiom, the theory remains agnostic with regards to the status of fluents in those situations that are reached by performing non-possible actions. Instead, we choose to restrict their value in such a way that they maintain their values (holding or not

[8] The sorts for the variables should be obvious from the context

[9] In [6] the symbol $\sqsubset$ is used instead of $<$, and $\sqsubseteq$ instead of $\leq$.

holding) after illegal actions are performed. This definition has an impact on the way in which the effect axioms and the successor axioms are written. As seen later, we will qualify the effect and successor state axioms with *legal* instead of *Poss*.

Action Structure: Each action is the result of the composition of one input and a random reaction. Thus, we have:

$$(\forall\, a)\ a = \langle a_1, a_2 \rangle \tag{9}$$

Recall that the subindices 1 and 2 are used as functions. Therefore a_1 denotes the input and a_2 denotes the reaction of action a. Also, we need uniqueness of names:

$$\langle i, x \rangle = \langle i', x' \rangle \supset i = i' \wedge x = x'. \tag{10}$$

Since each action is the result of the composition of an input and a reaction, in the domain axiomatization we will have the specification for the action preconditions in terms of *input preconditions*. In order to fix the correspondence between the two, we need to state that:

$$Poss(a, s) \equiv iposs(a_1, s) \wedge rposs(a_2, a_1, s). \tag{11}$$

which can alternatively be written as:

$$Poss(\langle i, x \rangle, s) \equiv iposs(i, s) \wedge rposs(x, i, s). \tag{12}$$

where:

$$rposs(x, i, s) \equiv gdf(i, s)(x) > 0. \tag{13}$$

and

$$gdf(i, s)(x) = cdf(i, s)'(x) = \lim_{y \to x^-} \frac{cdf(i, s)(x) - cdf(i, s)(y)}{x - y}. \tag{14}$$

If the *cdf* is discrete, then the density function would not be properly defined, since the limit in the previous formula would not exist. However, in order to deal with this issue, we can resort to the use of the *Dirac Delta* function, so that the derivative of a step of amplitude D, would be D times the Dirac Delta function.

Cdf Axioms: As stated before, if i and s are an input and a situation, then the $cdf(i, s)$ denotes the cummulative distribution function of the reaction to input i in situation s. In order for *cdf* to be properly defined, we provide the following axioms:

$$\lim_{x \to +\infty} cdf(i, s)(x) = 1, \tag{15}$$

$$\lim_{x \to -\infty} cdf(i, s)(x) = 0, \tag{16}$$

$$x < y \supset cdf(i, s)(x) \leq cdf(i, s)(y), \tag{17}$$

$$\lim_{y \to x^+} cdf(i, s)(y) = cdf(i, s)(x). \tag{18}$$

Axioms (15) and (16) state that the *cdf* functions starts in 0 and ends in 1. Axiom (17) states that *cdf* are non-decreasing for increasing values of their parameters parameter. Finally, axiom (18) states that the *cdf* functions are continuous from the right.

Axioms about *prob***:** Let f be a fluent, then $prob(f, i, s)$ denotes the probability that f holds after input i is given in situation s. We have:

$$prob(f, i, s) = \int_{-\infty}^{+\infty} cf(holds(f, do(\langle i, x\rangle, s)))cdf'(i, s)(x)dx, \tag{19}$$

where cf stands for characteristic function. So, if φ is a logical sentence, then:

$$\varphi \supset cf(\varphi) = 1,$$
$$\neg\varphi \supset cf(\varphi) = 0.$$

2.4 Examples

2.4.1 The Casino Coin

This is a simple example in which a gambler goes to a casino with a bag of coins. The gambler tosses one coin at a time, as long as the bag is not empty. If the gambler obtains heads, then she wins a coin; if she obtains tails, then she loses a coin.

To model this example we use the sort $\mathcal{N}$ for natural numbers, and introduce the constant *Toss* of type $\mathcal{I}$, the constants *Heads* and *Tails* of type $\mathcal{R}$, the function $bag : \mathcal{N} \to \mathcal{F}$, and the constant *Winning* of type $\mathcal{F}$. The axiomatization follows:

Domain Axioms:

$$Heads = +1, \tag{20}$$
$$Tails = -1. \tag{21}$$

Axioms about the initial situation:

$$holds(bag(K), S_0). \tag{22}$$

CDF axioms:

$$iposs(i, s) \wedge s \geq S_0 \supset cdf(i, s) = Bernoulli(0.5). \tag{23}$$

Here, we refer to the Bernoulli distribution with a meaning different from the usual. Namely, for $p \in [0, 1]$:

$$Bernoulli(p)(y) = \begin{cases} 0 & \text{whenever } y < -1 \\ 1 - p & \text{whenever } -1 \leq y < 1 \\ 1 & \text{otherwise} \end{cases}$$

Action precondition axioms: Action precondition axioms, which, in Reiter's style of axiomatization, are expressed in terms of conditions for *Poss*, are now expressed in terms of *iposs*. I.e., we specify constraints for the possibility of giving some input. Thus, for this example, we write:

$$iposs(Toss, s) \equiv holds(bag(n), s) \wedge n > 0. \tag{24}$$

Effect axioms:

$$\begin{aligned}&[legal(do(\langle Toss, Tails\rangle, s)) \wedge \\ &\quad holds(bag(n), s) \wedge n > 0] \supset holds(bag(n-1), do(\langle Toss, Tails\rangle, s)),\end{aligned} \tag{25}$$

$$\begin{aligned}&[legal(do(\langle Toss, Heads\rangle, s)) \wedge \\ &\quad holds(bag(n), s) \supset holds(bag(n+1), do(\langle Toss, Heads\rangle, s).\end{aligned} \tag{26}$$

Ramification State Constraints: The definition of *winning situation* is given as an equivalence (27). Also, constraint (28) states that bag is a functional fluent.

$$holds(Winning, s) \equiv holds(bag(n), s) \wedge n > K. \tag{27}$$

$$holds(bag(n), s) \wedge holds(bag(m), s) \supset n = m. \tag{28}$$

Based on the approach presented in [5], the effect axioms and the ramification state constraints are replaced by *successor state axioms.* These axioms are derived, by syntactic transformations, from the effect axioms and ramification constraints. The following axioms are obtained after some simplifications, and assuming that the only actions that exist are coin tosses:

Successor State Axioms:

$$\begin{aligned}legal(do(a, s)) \supset [&holds(bag(n), do(a, s)) \equiv \\ &\quad holds(bag(n+1), s) \wedge a = \langle Toss, Tails\rangle \vee \\ &\quad holds(bag(n-1), s) \wedge a = \langle Toss, Heads\rangle].\end{aligned} \tag{29}$$

$$\begin{aligned}legal(do(a, s)) \supset [&holds(Winning, do(a, s)) \equiv \\ &\quad (holds(bag(n+1), s) \wedge a = \langle Toss, Tails\rangle \vee \\ &\quad holds(bag(n-1), s) \wedge a = \langle Toss, Heads\rangle) \wedge \\ &\quad n > K].\end{aligned} \tag{30}$$

Therefore, the coin casino theory Σ_{cc} is composed by the DTT and SCT axioms, along with axioms (20)-(24), and (29)-(30).

Based on this axiomatization, let us see how to compute some probabilities. For example, let us compute the probability that after one toss we are in a winning situation. Therefore, we need to find the denotation of the term:

$$prob(Winning, Toss, S_0).$$

From axiom (19), we can write:

$$\begin{aligned} & prob(f, Toss, S_0) = \\ & \qquad \int_{-\infty}^{+\infty} cf(holds(Winning, do(\langle Toss, x\rangle, S_0)))cdf'(Toss, S_0)(x)dx, \end{aligned} \tag{31}$$

Now,

$$cdf'(Toss, S_0)(x) = 0.5DiracDelta(x+1) + 0.5DiracDelta(x-1). \tag{32}$$

From (31) our oracle would tell us that:

$$\begin{aligned} & prob(Winning, Toss, S_0) = \\ & \qquad 0.5\ cf(holds(Winning, do(\langle Toss, Heads\rangle, S_0))) + \\ & \qquad 0.5\ cf(holds(Winning, do(\langle Toss, Tails\rangle, S_0))). \end{aligned} \tag{33}$$

Now, from (30), and the initial conditions:

$$holds(Winning, do(\langle Toss, x\rangle, S_0)) \equiv x = Heads.$$

Therefore:

$$prob(Winning, Toss, S_0) = 0.5.$$

Notice that this calculations can be performed analytically by our oracle (Mathematica).

2.5 Reasoning

In this subsection we present an approach for the computational realization of a system that reasons about actions based on theories written in the Probabilistic Situation Calculus. The approach is based on the encoding of the logical theories into rewrite rules of Mathematica[11]. There are three examples to illustrate the approach. First, we present the *Casino Coin*, which was used as a running example earlier in the article. This is an example of a discrete system. We show how certain simple probabilities can be estimated using a Monte–Carlo approach. The second example, a random walk, shows an example with continuous distributions. Finally, in the third example we present a random walk with random turns controlled by coin throw. This example combines discrete and continuous distributions.

We have explored the use of Mathematica as our oracle that is able to evaluate complex analytical expressions. Mathematica can also be used to perform certain logical reasoning by encoding propositional truth values as numerical constants, and propositional operators as numerical operations. Furthermore, Mathematica provides a rewrite rule inferential mechanism.

For simplicity of exposition, we present three examples in which the outcomes to inputs are univariate. These examples are the *Coin Casino*, a *Random Walk*, and a *Random Walk with Random Turns*. As shown in section 3 it is not difficult to extend the definitions presented in both types of examples to a generalized multivariate case. For the implementation in Mathematica we present a bivariate example.

2.5.1 Mathematica Encoding of Domain Independent Definitions

In order to uniformly handle discrete and continuous distributions, we model discrete distributions with real functions by making use of the *Dirac delta*. Recall that the Dirac delta is a function that is zero over $\mathbb{R}$ except in an infinitesimal neighborhood around 0, whose indefinite integral is 1. In the examples, we will use *Bernoulli*, *exponential*, and the *logistic* probability distributions. To specify the distributions, we explicitly encode their *cummulative distribution functions* (`cdf`) and its inverse (`idf`). In the examples, we'll use the following:

```
cdfbern[p_] := Function[x,(1-p)UnitStep[x+1]+p UnitStep[x-1]];
idfbern[p_] := Function[z, If[z<p,+1,-1]];
cdfexp[m_] := Function[x, UnitStep[x] (1 -  E^(-x/m))];
idfexp[m_] := Function[z, m Log[1/(1-z)]];
cdflogistic[k_] := Function[x, 1 - (1/(1 + E^(x/k))];
idflogistic[k_]:= Function[z,k Log[z/(1-z)];
```

The Monte–Carlo simulation for the univariate case is fairly straightforward. Given a fluent, and an input sequence, we want to determine the probability that the fluent will hold after executing the input sequence with the corresponding reactions. Since the reactions are random, the simulation generates the random reaction from the corresponding inverse cumulative probability distribution, this is simply done as:

```
gen[i_,s_]:=idf[i,s][Random[]];
```

Thus, given an input and a situation, the `gen` function gives a random reaction. Notice that the probability distribution of the reaction might depend on the situation where the input is performed. The `idf` function is domain dependent, hence is specified in the domain axiomatization below.

A Monte–Carlo run is defined as a Bernoulli experiment consisting of taking an input sequence and simulating the execution of the ordered sequence of inputs, each followed by its random reaction. The experiment is successful, if the fluent of interest (parameter of the experiment) holds after the experiment ends. The run is defined recursively as follows:

```
mcrun[f_,{},s_]:= holds[f,s];
mcrun[f_,w_,s_]:= mcrun[f,Rest[w],do[{First[w],gen[First[w],s]},s]];
```

In order to estimate the probability that fluent `f` will hold after an input sequence `w`, we use a frequency count. Thus, we run the Monte–Carlo experiment n times, and the estimate of the probability that the outcome will be such that `f` holds is the proportion of times that the experiment succeeds. This is defined by defining a `prob` function as follows:

```
prob[f_,w_,s_,n_]:=N[Count[Table[mcrun[f,w,s],{jrun,1,n}],True]/n];
```

Recall that when estimating the probability p of a Bernoulli distribution, the (random) estimation error has an asymptotic Gaussian distribution with mean 0. Its standard deviation is given by:

$$\sqrt{\frac{4p(1-p)}{n}}$$

This standard deviation reaches a maximum when p = 1/2. Therefore, an upper bound for this standard deviation is $1/\sqrt{n}$.

The domain independent situation calculus axioms required are direct translations of logical axioms introduced in section 2.3. These are: check ref

```
poss[{i_,r_},s_]:=iposs[i,s]∧ cdf[i,s]'[r]>0;
legal[s0]:=True;
legal[do[{i_,r_},s_]]:=legal[s]∧ poss[{i,r},s];
holds[f_,do[a_,s_]]:=holds[f,s]/;¬legal[do[a,s]];
```

2.5.2 The Casino Coin in Mathematica

First, `heads` and *tails* are real constants that correspond to the reactions that can follow a coin toss. We introduce the constant `ibconts` to denote the initial contents of the bag:

```
heads := +1;
tails := -1;
ibconts := 2;
```

The cumulative distribution function and its inverse are defined for any input and situation to be the ones corresponding to the Bernoulli distribution:

```
cdf[i_,s_] := cdfbern[1/2];
idf[i_,s_] := idfbern[1/2];
```

We'll make use of the following abbreviation:

```
bconts[s_] := iota[n,holds[bag[n],s]];
```

`bconts` gives the bag contents at any given situation. For this definition, we appeal to the `iota` function. In this case, `iota` gives the value of `n` such that `holds[bag[n],s]` is true. This value has to be unique, in the implementation it gives the first solution it encounters, if any. The definition is:

```
iota := Function[z,c,Solve[c,z][[1,1,2]]];
```

The following expressions correspond to the specifications of the axioms about the initial situation (`s0`), the `iposs` input preconditions and a definition state constraint, which establishes whether or not the gambler is considered to be winning:

```
holds[bag[n_],s0] := n==ibconts
iposs[i_,s_] := i===toss ∧ bconts[s]≥1
holds[winning,s_]:=bconts[s]≥ ibconts;
```

The following is the successor state axiom for the fluent `bag`:

```
holds[bag[n_], do[{i_, y_},  s_]] := ((i === toss ∧
          holds[bag[n - y], s]) ∨ (i =!= toss ∧
          holds[bag[n], s])) /; legal[do[{i, y}, s]];
```

Given the domain specific definitions above, along with the domain independent definitions of the previous subsection, we can already perform probabilistic temporal projection. In the following sample interaction with Mathematica, the slanted text is the response given by Mathematica.

First, we ask whether some situations are legal. Notice that it is not legal to perform a `toss` and obtain a reaction different from `-1` or `1`:

```
legal[do[{toss,heads},s0]]
True
legal[do[{toss,π},s0]]
False
```

Now, we ask whether it is possible to have a `toss` input in some situations:

```
iposs[toss,s0]
True
iposs[toss,do[{toss,tails},s0]]
True
iposs[toss,do[{toss,π},s0]]
True
iposs[toss,do[{toss,tails},do[{toss,tails},s0]]]
False
```

It might be surprising to obtain a positive answer in the penultimate question. Indeed, we are being told that it is possible to perform a `toss` after a first `toss` is performed and a non `0` or `1` reaction is obtained. This is explained because, although the input is possible, the situation is not legal (see above).

In the following interaction, we show how the reasoning is done with regards to the bag contents. These are straightforward logical inferences:

```
holds[bag[n],do[{toss,tails},do[{toss,tails},do[{toss,tails},s0]]]]
2+n==2
legal[do[{toss,tails},do[{toss,tails},do[{toss,tails},s0]]]]
False
holds[bag[n],do[{toss,tails},do[{toss,tails},s0]]]
2+n==2
holds[winning,do[{toss,tails},s0]]
False
```

Finally, the following interaction illustrates the results obtained when performing probability computations. Recall that these probabilities are actually estimates of the real probability computations. This is done by using a Monte–Carlo approach. The `1000` number is the number of experiments that are conducted in order to obtain the estimates. Also, recall that the probability that is being computed is the probability that certain fluent holds after a sequence of inputs is performed in a given situation. For instance, in the penultimate query, we are asking for the probability that the bag contents is `0` after performing three tosses starting in `s0`.

```
prob[winning,{toss},s0,1000]
0.501
prob[winning,{toss,toss},s0,1000]
0.745
prob[bag[0],{toss,toss,toss},s0,1000]
0.245
prob[bag[0],{toss},do[{toss,tails},do[{toss,tails},s0]],1000]
1.
```

All the previous probability estimates can also be obtained by using the integration methods available in mathematica, both numerical and analytical. However, the complexity of the numerical method increases exponentially with the length of the input sequence. On the other hand, the analytical method it becomes, in general, intractable.

2.5.3 A Random Walk

Now we present an example in which we illustrate the reasoning capabilities that we obtain from our oracle in a simple domain in which the probabilities have a continuous distribution. The example is an *exponential random walk*, in which the walking man has the ability to move as long as its position is at the right of where he starts from (along a single dimension, starting at position 0). The walking man can produce two possible inputs, `moveleft` and `moveright`. When an input is performed, the random reaction obtained is the distance by which it moves. The distances (random reactions) have an exponential distribution. This fact is given to our oracle in the following manner:

```
cdf[moveleft,s_] := cdfexp[3] ;
idf[moveleft,s_] := idfexp[3] ;
cdf[moveright,s_] := cdfexp[2] ;
idf[moveright,s_] := idfexp[2] ;
```

Notice that, as in the *coin casino*, we specify the cumulative distribution function and its inverse. The latter is necessary for performing the simulation in the Monte–Carlo approach to the probability estimations of fluents values.

We will make use of the following auxiliary definition:

```
pos[s_]:=iota[x,holds[position[x],s]];
```

this means that pos is a functional fluent, whose value corresponds to the position of the walking man in a given situation. As explained before, the function iota gives, in this case, the value of x such that x is the position of the walking man in the specified situation.

Next, we specify the preconditions for the inputs and the initial situation

```
iposs[i_,s_] := (i===moveright∨i===moveleft)∧pos[s]≥0;
holds[position[x_],s0]:=(x==0);
```

The successor state axioms for the position fluent in this domain is as follows:

```
holds[position[x_],  do[{i_, y_},  s_]] := ((i === moveleft ∧
              holds[position[x + y], s]) ∨ (i === moveright ∧
              holds[position[x - y], s]) ∨ (i =!= moveleft ∧
              i =!= moveright ∧ holds[position[x], s])) /; legal[do[{i, y}, s]];
```

In this example, we define two fluents in terms of the position of the walking man. First, hawk is a fluent that tells us whether or not the walking man has a tendency to move to the right. Also, the fluent center is true whenever the position of the walking man is no further than 1 unit from the center. Notice that these two fluents are acting as abbreviation for their corresponding defintions.

```
holds[hawk,s_]:=pos[s]≥0;
holds[center,s_] :=pos[s]≥-1 ∧ pos[s]≤+1;
```

Given the above domain specific specification, along with the general domain independent specification of section 2.5.1, we can query our oracle. First, we ask general situation calculus questions that take a specific situation (built with specific action executions, always starting at s0) and asking various properties of such a situation:

```
legal[do[{moveright,7},s0]]
True
holds[position[x],do[{moveright,7},s0]]
-7+x==0
holds[hawk,do[{moveright,7},do[{moveleft,2},s0]]]
False
holds[hawk,do[{moveright,7},do[{moveright,2},s0]]]
True
holds[center,do[{moveleft,1},s0]]
True
```

Now, as with the *coin casino example*, we estimate the real probabilities that the fluents `hawk` and `center` hold at the specified situations, by using a Monte–Carlo approach with `1000` points. It is easy to verify that the results are very close to the theoretical results expected.

```
prob[hawk,{moveright},s0,1000]
1.
prob[hawk,{moveleft},s0,1000]
0.
prob[hawk,{moveright,moveleft},s0,1000]
0.41
prob[center,{moveleft},s0,1000]
0.3
prob[center,{moveleft,moveright},s0,1000]
0.299
```

2.5.4 Random Walk with Random Turns

The following example is a variation on the random walk presented in the previous subsection. The purpose is to illustrate how discrete random reaction can be easily combined with continuous random reactions. The walking man, in this case, has two actions at his disposal. First, he can move, but always in the direction towards which he is currently moving. Secondly, he can throw a coin, which leads to changing the direction of motion. First, we define some constants:

```
right:=+1;
left:=-1;
heads := +1; (* no direction change *)
tails := -1; (* change direction *)
```

Now, we define the probability distributions for the move (continuous exponential distribution) and the toss (discrete Bernoulli distribution):

```
cdf[move,s_] := cdfexp[1] ;
idf[move,s_] := idfexp[1] ;
cdf[toss,s_] := cdfbern[1/2] ;
idf[toss,s_] := idfbern[1/2] ;
```

For reasons of efficiency, it is easier for the oracle to deal with functional fluents. However, this can be directly defined in terms of the fluents defined previously. Thus, we have:

```
pos[s_]:=iota[x,holds[position[x],s]];
dir[s_]:=iota[j,holds[direction[j],s]];
trn[s_]:=iota[n,holds[turns[n],s]];
```

Next, we specify the input preconditions. Simply, any input is possible, as long as the position is at the initial position or to its right.

```
iposs[i_,s_] := (i===move∨i===toss)∧pos[s]≥0;
```

Now, we specify the initial situation:

```
holds[position[x_],s0]:=(x==0);
holds[direction[j_],s0]:=(j==right);
holds[turns[n_],s0]:=(n==0);
```

The successor state axiom for the fluents in this domain are the following:

```
holds[position[x_],
    do[{i_,y_},
      s_]]:=((i===move∧holds[position[x-dir[s]y],s])∨(
          i=!=move∧holds[position[x],s]))/;legal[do[{i,y},s]];
holds[direction[j_],do[{i_,k_},s_]]:=
        ((i===toss∧j==k dir[s])∨(i=!=toss∧holds[direction[j],s]))/;
    legal[do[{i,k},s]];
holds[turns[n_],do[{i_,k_},s_]]:=
        ((i===toss∧k==tails∧holds[turns[n-1],s])
          ∨((i=!=toss∨k!=tails)∧holds[turns[n],s]))/;
    legal[do[{i,k},s]];
```

Finally, as in the previous example, we define the fluents `hawk` and `center`:

```
holds[hawk,s_]:=pos[s]≥0;
holds[center,s_] :=pos[s]≥-1 ∧ pos[s]≤+1;
```

Based on the previous definition, we can interact, in the same way as before with our oracle. Below, there are some probability computations calculated using the Monte–Carlo approach:

```
prob[hawk,{move},s0,1000]
1.
prob[hawk,{toss},s0,1000]
```

```
1.
prob[hawk,{toss,move},s0,1000]
0.485
prob[hawk,{toss,move,toss,move},s0,1000]
0.373
prob[center,{toss,move,toss,move},s0,1000]
0.533
prob[turns[2],{toss,toss},s0,1000]
0.238
prob[turns[2],{toss,move,toss},s0,1000]
0.
```

2.6 Semantics

In order to define the semantics of the Probabilistic Situation Calculus, we introduce the concept of a Randomly Reactive Automaton. Conceptually, a Randomly Reactive Automaton is defined by a set of situations, each situation corresponds to a sequence of $\langle input, reaction\rangle$ pairs. The inputs come from a set (arbitrary, domain dependent) and the reactions are the real numbers. Each situation has a *menu* which corresponds to the inputs that are considered possible at each situation. Given a situation and an input, a probability distribution is defined for the set of reactions. The Randomly Reactive Automata are later used in order to provide the semantic structures for the Probabilistic Situation Calculus.

Here we present Randomly Reactive Automata in two different ways. Firstly, we present an outcome based approach. Although the presentation only considers reactions that are real numbers, in this approach, Randomly Reactive Automata can have reactions of any type. The presentation for arbitrary sets of reactions is possible but more complex. Thus, the *outcome based approach* is mainly of theoretical interest, but it shows how to define completely general automata for the Probabilistic Situation Calculus. Secondly, we present Randomly Reactive Automata with a *cumulative distribution function* approach. This is the approach that maps directly to our Situation Calculus language specification of the previous subsection. As discussed in the next section, the disadvantage of this approach is that it is only practical in the univariate case.

2.6.1 Outcome approach

For introducing the outcome approach we need some preliminary concepts like probability space and random variable.

A *probability space* is a triple $\Pi = \langle \Omega, \mathcal{B}, P\rangle$ where Ω is a set (of outcomes), $\mathcal{B}$ is a σ-algebra (that is a set of subsets of Ω containing Ω and being closed for complements and countable unions) and P is a probability defined over $\langle \Omega, \mathcal{B}\rangle$ (that is a map $P : \mathcal{B} \to [0, 1]$ such that

- $P(\Omega) = 1$;
- $P(B^c) = 1 - P(B)$;
- $P(\cup_{i=1}^{\infty} B_i) = \sum_{i=1}^{\infty} P(B_i)$ for pairwise disjoint sets).

No restrictions are assumed on the cardinality of Ω. When Ω is countable then $\mathcal{B}$ is usually $\wp\Omega$. In our case it is not enough to consider Ω countable because we want to consider random reactions that may assume any positive real value. Accordingly we are going to recall the definition of the Borel σ-algebra over the real numbers as the appropriate σ-algebra for our situation. Moreover, we will introduce the definition of random variable.

The *Borel σ-algebra* over the real numbers, $\mathcal{B}(\mathbb{R})$, is the σ-algebra generated by the open intervals with rational endpoints. A *random variable* X over Π is a measurable mapping from $\langle \Omega, \mathcal{B} \rangle$ to $\langle \mathbb{R}, \mathcal{B}(\mathbb{R}) \rangle$, that is, a map $X : \Omega \to \mathbb{R}$ such that $X^{-1}(B) \in \mathcal{B}$ for every $B \in \mathcal{B}(\mathbb{R})$. A *random vector* with indexed (finite) set U is a measurable map $\langle \Omega, \mathcal{B} \rangle \to \langle \mathbb{R}^U, \mathcal{B}(\mathbb{R}^U) \rangle$. We denote by $\mathcal{B}(\mathbb{R}^U)$ the σ-algebra generated by rectangles over $\mathbb{R}^U$ with rational endpoints.

In the sequel, we denote by A^* the set of finite sequences of elements in a set A. The empty sequence will be denoted by ϵ.

Definition 2.1 A *randomly reactive automaton* is a tuple $\langle I, \Pi, \approx, M, X \rangle$ where:

- I is a set; it corresponds to the universe of possible inputs.
- $\Pi = \langle \Omega, \mathcal{B}, P \rangle$ is a probability space;
- $\approx$ is an equivalence relation over S;
- $M = \{M^{[s]}\}_{[s] \in [S]}$ where $M^{[s]} \subseteq I$; given the situation s, $M^{[s]}$ is the *menu* that is available in s (the inputs that are possible).
- $X = \{X_i^{[s]}\}_{i \in I, [s] \in [S]}$, where $X_i^{[s]}$ is a random variable over Π

where $S = (I\mathbb{R})^*$, $[S] = S/\approx$, $[s] = \{s' \in S : s \approx s'\}$ and such that

- $s \approx six$ whenever $s \in S$ and $six \in S \setminus L$;

where $L \subseteq S$ is defined inductively as follows: $\epsilon \in L$ and $six \in L$ iff $s \in L$, $i \in M^{[s]}$ and

$$\lim_{h \to 0^+} \frac{P(\{\omega \in \Omega : x - h < X_i^s(\omega) \leq x + h\})}{h} > 0. \tag{34}$$

The $\approx$ relation establishes an equivalence relation between legal and non-legal. If s is not legal, then it is related by $\approx$ with the last legal situation in the path from S_0 to s. The non legal situations inherit the characteristics of the last legal situation mentioned before.

The elements in I are the *inputs*. The elements in S are the *situations* and the elements of L are the *legal* situations. The set M_s is the *menu* in situation s. The random variable X_i^s gives the random *reaction* (time) of the automaton to input i in situation s.

The evolution consists in obtaining the situations that result from a situation and a sequence of inputs. For this purpose it is useful to work with probability spaces where the outcomes are sequences.

Given a probability space $\Pi = \langle \Omega, \mathcal{B}, P \rangle$, we denote by $\Pi^{\mathbb{N}} = \langle \Omega^{\mathbb{N}}, \mathcal{B}^{\bullet}, P^{\bullet} \rangle$ the probability space such that $\Omega^{\mathbb{N}}$ is the set of all infinite sequences of elements of Ω, $\mathcal{B}^{\bullet}$ is the σ-algebra generated by finite Cartesian products of sets in $\mathcal{B}$ and $P^{\bullet}$ is the probability induced by $P^{\bullet}(B_{k_1} \times \cdots \times B_{k_n}) = \prod_{j=1}^{n} P(B_{k_j})$ for every $n \geq 1$, $1 \leq k_1 < ... < k_n$ and $B_{k_j} \in \mathcal{B}$ for every $j = 1, \ldots, n$. Thus, we are assuming that the evolution of the Randomly Reactive Automata is *Markovian*, that is, the probability distribution of the reactions depends only on the situation in which the input is received, and on the input itself.

Definition 2.2 Let $\vec{i} \in I^n$ and $s \in S$. The *situation by $\vec{i}$ from s* is the map $\zeta_{\vec{i}}^s : \mathbb{R}^n \to (I\mathbb{R})^*$ defined as $\zeta_{\vec{i}}^s = \lambda \vec{x}.s\vec{i}_1\vec{x}_1 \ldots \vec{i}_n\vec{x}_n$.

Thus, the situation by $\vec{i}$ from s is a map, that takes a reaction vector and yields the situation that results from combining the inputs in $\vec{i}$ with the reactions given as a parameter to the map. Notice that when the input sequence has length zero, there is no change of situation.

Definition 2.3 Let $\vec{i} \in I^n$, $n \geq 1$ and $s \in S$. The *reaction vector by $\vec{i}$ from s* is the map $\vec{Y}_{\vec{i}}^{[s]} : \Omega^{\mathbb{N}} \to \mathbb{R}^n$ inductively defined as follows:

- $\vec{Y}_{i}^{[s]} = \lambda\sigma.\ X_i^{[s]}(\sigma_1)$, if $\vec{i} = i$;
- $\vec{Y}_{\vec{i}}^{[s]} = \lambda\sigma.\ \vec{Y}_{\vec{j}}^{[s]}(\sigma) X_i^{[\zeta_{\vec{j}}^s(\vec{Y}_{\vec{j}}^{[s]}(\sigma))]}(\sigma_n)$, if $\vec{i} = \vec{j}i$.

Furthermore, we can define the *situation by $\vec{i}$ from s* as follows: $Z_{\vec{i}}^s = \zeta_{\vec{i}}^s \circ \vec{Y}_{\vec{i}}^s$.

Given a randomly reactive automaton $\langle I, \Pi, \approx, M, X \rangle$ where $\Pi = \langle \Omega, \mathcal{B}, P \rangle$, we describe its situation evolution over $\Pi^{\mathbb{N}}$ as follows.

Definition 2.4 Let $R \subseteq S$ closed for $\approx$, $\vec{i} \in I^n$, $n \geq 1$ and $s \in S$. The *target set of reaction vectors for R by $\vec{i}$ from s* is $R_{\vec{i}}^s = \{\vec{x} \in \mathbb{R}^n : \zeta_{\vec{i}}^s(\vec{x}) \in R\}$. The *probability of reaching R by $\vec{i}$*

from s is

$$Prob(R, \vec{i}, s) = P^{\bullet}(\{\sigma : \vec{Y}_{\vec{i}}^{[s]}(\sigma) \in R_{\vec{i}}^{s}\}).$$

This last definition establishes the meaning of the probability of being in some situation given that some specific input is performed. Thus, for each possible input, it yields the probability of reaching some situation in a subregion R. In the Situation Calculus logical theory, the subregion R is specified as a fluent.

2.6.2 Cumulative distribution approach

For introducing the cumulative distribution approach we need to introduce the definition of distribution function.

A *distribution function* is a map $F : \mathbb{R} \to [0, 1]$ such that

1. F is nondecreasing;
2. F is continuous from the right;
3. $\lim_{x \to -\infty} F(x) = 0$, and $\lim_{x \to +\infty} F(x) = 1$.

Given a random variable X over the probability space $\langle \Omega, \mathcal{B}, P \rangle$, we can obtain the associated distribution function as follows:

$$F_X(x) = P(\{\omega \in \Omega : X(\omega) \leq x\}).$$

Consider the probability space $\Pi' = \langle]0, 1[, \mathcal{B}(]0, 1[), \mu \rangle$ where $\mathcal{B}(]0, 1[)$ is the Borel σ-algebra over $]0,1[$ and μ is the Lebesgue measure, that is $\mu(]a, b[) = b - a$ for $]a, b[\subseteq]0, 1[$. Given a distribution function F, we can obtain a random variable X_F over Π' having F as its associated distribution function F as follows:

$$X_F = \lambda \omega . F^{-1}(\omega)$$

where F^{-1} is the generalized inverse of the distribution function F, that is

$$F^{-1} = \lambda \omega . \inf\{x \in \mathbb{R} : F(x) \geq \omega\}$$

The last procedure is used to simulate values from a given distribution function in a computational way. The outcomes of Π' are generated from a uniform (pseudo)-random number generator.

Now we are ready to give an alternative distribution function definition of a randomly reactive automaton.

Definition 2.5 A *randomly reactive automaton* is a tuple $\langle I, \approx, M, F\rangle$ where:

- I is a countable set;
- $\approx$ is an equivalence relation over S;
- $M = \{M^{[s]}\}_{[s]\in[S]}$ where $M^{[s]} \subseteq I$;
- $F = \{F_i^{[s]}\}_{i\in I,[s]\in[S]}$, where $F_i^{[s]}$ is a distribution function

where $S = (I\mathbb{R})^*$, $[S] = S/\approx$, $[s] = \{s' \in S : s \approx s'\}$ and such that

- $s \approx six$ whenever $s \in S$ and $six \in S \setminus L$;

where $L \subseteq S$ is defined inductively as follows: $\epsilon \in L$ and $six \in L$ iff $s \in L$, $i \in M^{[s]}$ and

$$\lim_{h\to 0^+} \frac{F_i^{[s]}(x+h) - F_i^{[s]}(x-h)}{h} > 0. \tag{35}$$

Given an outcome based definition of a randomly reactive automaton $\langle I, \Pi, \approx, M, X\rangle$, we can obtain a distribution based definition of the same automaton $\langle I, \approx, M, F(I,X)\rangle$ where $F(I,X) = \{F_i^{[s]}\}_{i\in I, s\in S}$ as follows:

$$F_i^{[s]} = F_{X_i^{[s]}}.$$

Conversely, given a distribution based definition of a randomly reactive automaton $\langle I, \approx, M, F\rangle$, we can obtain an outcome based definition for the same automaton $\langle I, \Pi', \approx, M, X(F,I)\rangle$ as follows:

- $\Pi' = \langle]0,1[, \mathcal{B}(]0,1[), \mu\rangle$;
- $X(F,I) = \left\{X_i^{[s]}\right\}_{i\in I,[s]\in[S]}$ with $X_i^{[s]} = \lambda\omega'.(F_i^{[s]})^{-1}(\omega')$.

Therefore the two definitions of randomly reactive automaton are equivalent from a distributional point of view.

In order to characterize the evolution of the automaton it is useful to consider the distribution functions of the random vectors $\{\vec{Y}_{\vec{i}}^{[s]}\}$ with $\vec{i} \in I^*$, with $|\vec{i}| = n$, and $s \in S$:

$$\{\vec{G}_{\vec{i}}^{[s]}\} = \lambda\vec{x}{\in}\mathbb{R}^n.P^\bullet(\{\sigma \in \Omega^{\mathbb{N}} : \vec{Y}_{\vec{i}}^{[s]}(\sigma) \le \vec{x}\}),$$

The distribution functions $\{\vec{G}_{\vec{i}}^{[s]}\}_{i \in I, s \in S}$ can be defined in terms of the distribution functions $F_i^{[s]}$ as follows:

$$\vec{G}_{\vec{i}}^{[s]} = \lambda \vec{x} {\in} \mathbb{R}^n . \int_{-\infty}^{\vec{x}_1} \cdots \int_{-\infty}^{\vec{x}_n} F_{\vec{i}_n}^{[s \vec{i}_1 y_1 \vec{i}_2 y_2 \ldots \vec{i}_{n-1} y_{n-1}]}(dy_n) \ldots F_{\vec{i}_1}^{[s]}(dy_1) \tag{36}$$

where the integrals are Lebesgue-Stieltjes (refer, for example, to [8]) and $\vec{z}_j$ denotes the j-th component of $\vec{z}$.

Definition 2.6 *Let $R \subseteq S$ closed for $\approx$, $\vec{i} \in I^n$, with $\vec{i} = \vec{j}i$, $n \geq 1$ and $s \in S$. The probability of reaching R by $\vec{i}$ from s is*

$$Prob(R, \vec{j}i, s) = \int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty} 1_{R^s_{\vec{j}i}}(\vec{y}x) F_i^{[\zeta^s_{\vec{j}}(\vec{y})]}(dx) F_{\vec{j}_{n-1}}^{[\zeta^s_{\vec{j}_{n-2}}(\vec{y}_{n-2})]}(d\vec{y}_{n-1}) \ldots F_{\vec{j}_1}^{[s]}(d\vec{y}_1).$$

Here $1_{R^s_{\vec{i}}}(z_1, \ldots, z_n)$ is the indicator function for the set $R^s_{\vec{i}}$, that is it is 1 if $\langle z_1, \ldots, z_n \rangle \in R^s_{\vec{i}}$ and 0 otherwise.

In the section devoted to reasoning within the Probabilistic Situation Calculus, we develop the implementation based on this cumulative distribution function approach. In fact, when computing the probabilities that certain fluents hold after an arbitrary input sequence, one could attempt to solve the integral above analytically. However, very quickly (after 2 inputs) the task becomes mathematically intractable. Thus, instead of solving these integrals analytically, we appeal to a Monte–Carlo approach, which proves to be quite efficient.

2.6.3 Examples Revisited

We are going to consider again two of the previous examples. For both cases we present the Randomly Reactive Automaton adopting the the distribution function approach. In one of them the distribution functions are discrete and in the other they are absolutely continuous.

Casino coin

As an example, let us model the following *casino coin*. In this casino, a player arrives with a number of coins. At the casino, she tosses a coin, if the coin lands *Tails*, then she loses the coin, if it lands *Heads*, then the casino gives her another coin. The player can toss coins only if she has at least one.

For describing the automaton corresponding to this example, we use the distribution approach of subsection 2.6.2. Thus, we need to define I, $\approx$, M and F:

- $I = \{toss\}$;
- if $i_1x_1 \ldots i_nx_n \approx i'_1x'_1 \ldots i'_{n'}x'_{n'}$ then

$$\sum_{k=1}^{n} x_k = \sum_{k'=1}^{n'} x'_{k'}$$

  and $i_1x_1 \ldots i_nx_n, i'_1x'_1 \ldots i'_{n'}x'_{n'} \in L$ (L is the set of legal situations. Thus, no coin $toss$es are performed unless there are coins available.).

- $F_i^{[s]} = Bernoulli(0.5)$ for each $i \in I$ and $s \in S$, that is:

$$F_i^{[s]}(x) = \begin{cases} 0 & \text{provided that } x < -1 \\ 1/2 & \text{provided that } -1 \leq x < 1, \\ 1 & \text{elsewhere} \end{cases};$$

- $M^{[s]}$ is equal to $\{toss\}$ whenever there is $i_1x_1 \ldots i_nx_n \in L \cap [s]$ such that $K + \sum_{k=1}^{n} x_k > 0$ and equal to $\emptyset$ otherwise.

Exponential random walk

Consider a robot that makes moves to the right or to the left. The shift to either side is an exponential random variable. Using again the distribution approach we can specify the automaton corresponding to this example in the following way:

- $I = \{-1, 1\}$ (1 corresponds to a move to the right and -1 to a move to the left);
- if $i_1x_1 \ldots i_nx_n \approx i'_1x'_1 \ldots i'_{n'}x'_{n'}$ then

$$\sum_{k=1}^{n} i_kx_k = \sum_{k'=1}^{n'} i'_kx'_{k'}$$

  and $i_1x_1 \ldots i_nx_n, i'_1x'_1 \ldots i'_{n'}x'_{n'} \in L$;

- for every $s \in S$, $F_i^{[s]} = cdf(exponential(\lambda_i))$, where

$$\lambda_i = \begin{cases} 3 & \text{if } i = -1, \\ 2 & \text{if } i = +1. \end{cases}$$

- $M^{[s]}$ is equal to $\{-1, 1\}$ whenever there is $i_1x_1 \ldots i_nx_n \in L \cap [s]$ such that $\sum_{k=1}^{n} i_kx_k \geq 0$ and equal to $\emptyset$ otherwise.

2.7 Random Reactive Automaton Semantics for Extended Situation Calculus Theories

The extended situation calculus language introduced above is used for the specification of a randomly reactive automaton $\langle I, \approx, F, M\rangle$ (see section 2.6.2). The specification SPEC is composed of DTT (data type theory), SCT (situation calculus axioms), proper operations (constants) for the sort $\mathcal{I}$ (for every $i \in \mathbb{I}$, $i : \mathcal{I}$), proper operations for the sort $\mathcal{F}$ (for every $f \in \mathbb{F}$, $f : \mathcal{F}$), along with proper axioms, e.g. $holds(bag(3), S_0)$.

A specification will be a tuple $\langle \mathbb{I}, \mathbb{F}, Ax\rangle$, whose signature is the pair $\mathbb{I}, \mathbb{F}$.

Let *Int* be an interpretation structure for the many sorted higher order language specification, such that: $\mathcal{I}_{Int} = \mathbb{I}$, $\mathcal{I}^*_{Int} = \mathbb{I}^*$, $\mathcal{A}_{Int} = \mathbb{I} \times \mathbb{R}$, $\mathcal{F}_{Int} = \mathbb{F}$, $\mathcal{R}_{Int} = \mathbb{R}$, $\mathcal{S}_{Int} = (\mathbb{I}\mathbb{R})^*$.

For each such *Int*, we can build a machine $m_{Int} = \langle I, \approx, F, M\rangle$, such that:

- $I = \mathcal{I}_{Int}$;
- if $[\![s]\!]_{Int} \approx [\![s']\!]_{Int}$ then $Int \Vdash holds(f, s)$ iff $Int \Vdash holds(f, s')$ for all $f \in T_{\mathcal{F}}$;
- $F^{[[\![s]\!]_{Int}]}_{[\![i]\!]_{Int}} = [\![cdf(i, s)]\!]_{Int}$ for each $i \in T_{\mathcal{I}}$, and $s \in T_{\mathcal{S}}$;
- $M^{[[\![s]\!]_{Int}]} = \{[\![i]\!]_{Int} : Int \Vdash iposs(i, s)\}$,

where T_σ denotes the set of terms of sort σ for signature $\langle \mathbb{I}, \mathbb{F}\rangle$.

We say that a consistent SPEC is categoric on the *cdf*s, iff we end up with the same machine no matter what *Int* is used. Notice that a consistent SPEC need not be categoric. Naturally, we must ensure that there is at least one interpretation for the *cdf*s.

Now, we can provide a semantics for *prob*:

$$[\![prob(f, \vec{i}, s)]\!]_{Int} = Prob_{m_{Int}}(\{[\![s']\!]_{Int} : Int \Vdash holds(f, s')\}, [\![\vec{i}]\!]_{Int}, [\![s]\!]_{Int}). \tag{37}$$

In the previous discussion, we have considered the simpler case in which the operations of sorts $\mathcal{F}$ and $\mathcal{I}$ are constants (zero-ary functions). This can be easily extended, although in a cumbersome manner, to the general case in which the language contains n-ary function symbols for both sorts. In the rest of the paper, we consider the more general case.

3 Probabilistic Situation Calculus Multivariate Case

3.1 Preliminaries

The previous section was restricted to cases in which the reaction obtained from an input was univariate; i.e., the reaction depends on a single random variable. In this section, we discuss the more general case in which reactions may depend on more than one random variable. Our work is limited to the case in which all the random variables involved in a reaction are mutually independent. As before, there are three aspects that we have to deal with. First, the language of the Situation Calculus needs to be extended. Second, we need to specify a Randomly Reactive Automata with multivariate reactions. Third, we need to show how the reasoning mechanism needs to be extended to accommodate these multivariate reactions.

Extending the Situation Calculus is fairly straightforward as shown below. The presentation of the Randomly Reactive Automata for multivariate reactions is more involved. In particular, we need to introduce an approach based on density functions. This is because the approach based on the cumulative distribution functions is much more difficult to develop. Although it is not too difficult to extend the implementation to the multivariate case, we present only an implementation for a bivariate case.

3.2 The Language and Axiomatization

As mentioned earlier, the extensions to the language of the situation calculus are simple. First, we externally define a constant K_U for the language, which specifies the maximum number of independent variables involved in a reaction. Then, we introduce constants and variables of sort $\mathcal{R}^{K_U}$. Furthermore, we need to modify the following definitions from section 2.2:

- $rposs \subseteq \mathcal{R}^{K_U} \times \mathcal{I} \times \mathcal{S}$.
- $\langle \_, \_ \rangle \colon \mathcal{I} \times \mathcal{R}^{K_U} \to \mathcal{A}$.
- $\_{}_2 \colon \mathcal{A} \to \mathcal{R}^{K_U}$.

Furthermore, we replace the *cdf* function for:

- $gdf \colon \mathcal{I} \times \mathcal{S} \to [\mathcal{R}^{K_U} \to \mathcal{R}]$.

Finally, if $\vec{r}$ is a term of type $\mathcal{R}^{K_U}$, we write r_i to denote its i^{th} component.

For the general axiomatization, we incorporate the following:

$$rposs(\vec{r}, i, s) \equiv gdf(i, s)(\vec{r}) > 0. \tag{38}$$

Furthermore, since we have assumed that the random variables associated to a single reaction are mutually independent, we write:

$$gdf(i, s)(\vec{r}) = gdf_1(i, s)(r_1) \cdots gdf_{K_U}(i, s)(r_{K_U}). \tag{39}$$

Take into account that K_U is a fixed constant, which has to be set for each possible axiomatization. For instance, in our example below, the value of K_U is 2. Therefore, we will have:

$$gdf(i, s)(\vec{r}) = gdf_1(i, s)(r_1)gdf_2(i, s)(r_2). \tag{40}$$

Notice that the assumption of independence is not necessary from a theoretical standpoint. Indeed, all we need is a specification of the multidimensional *gdf*. However, for practical purposes, this *gdf* might be difficult or impossible to state. If we were able to produce a *gdf* specification for some specific domain, then this assumption can be dropped.

3.3 Example

Assume that a jumping robot (*FrogBot*) has the ability to jump in a bidimensional space from one position to another. When the *FrogBot* wishes to jump, it issues a `move` command with two parameters: distance to jump, and the jump angle relative to the *FrogBot*'s North. Furthermore, there are two special areas defined in the *FrogBot* bidimensional space: a *bog* and a *target* area. If at any time the *FrogBot* lands in the bog, then the *FrogBot* gets stuck and further jumps become impossible. Due to the nature of the *FrogBot*'s machinery, the move commands are imprecise. Both, the angle and the distance of the actual jump incorporate an error (mutually independent). We have considered that both errors have a logistic distribution. So, the type of questions that we would like to answer in this domain are, for instance, what is the probability of landing in the bog after two jumps have been performed (with given distances and angles).

For the language, we introduce the term $move : \mathcal{R}^2 \to \mathcal{I}$. Thus, a term $move(d, \theta)$ specifies a command to move d units in the θ direction. Also, we will specify the position of the *FrogBot* with two functions $posx, posy : \mathcal{R} \to \mathcal{F}$.

Now, we only present a few axioms related to the specification of this example in the Situation Calculus. In this case, the value of the constant K_U is 2. First, we specify the *generalized density function* for both parameters:

$$gdf_1(move(d, \theta), s) = gdflogistic(d^3/1000), \tag{41}$$

$$gdf_2(move(d, \theta), s) = gdflogistic(\pi/30), \tag{42}$$

Where the function *gdflogistic* is a standard logistic density function.

The successor state axiom for *posx* follows (the successor state axiom for *posy* is analogous):

$$\begin{aligned}
&legal(do(\langle i, \vec{r}\rangle, s)) \supset \\
&\quad holds(posx(x), do(\langle i, \vec{r}\rangle, s)) \equiv \\
&\qquad [(\exists\, d, \theta)\ i = move(d, \theta) \wedge holds(posx(x'), s) \wedge x = x' + dcos(\theta) + r_1 cos(r_2)] \vee \\
&\qquad holds(posx(x), s) \wedge \neg(\exists\, d, \theta)\ i = move(d, \theta).
\end{aligned} \tag{43}$$

As we will show later in the implementation of the *FrogBot* in Mathematica, we also have defined fluents to specify whether the *FrogBot* is in the target, in the bog or elsewhere.

3.4 Semantics for Multivariate Reactions

The two approaches that we saw for the univariate case can also be adopted for multivariate reactions. However, multivariate distribution functions are not so well behaved and consequently, they are not very much used in practice. Herein, we will concentrate on a third automata definition through generalized density functions which is more feasible for the point of view of multivariate reactions. For pedagogical reasons, we present the generalized density approach for the simpler case of univariate reactions.

3.4.1 Generalized Density Approach for Univariate Reactions

For introducing the generalized density approach recall the notion of probability measure. Observe that there is a one to one correspondence between distribution functions and probability measures on $\langle \mathbb{R}, \mathcal{B}(\mathbb{R})\rangle$.

Let μ be a probability measure on $\langle \mathbb{R}, \mathcal{B}(\mathbb{R})\rangle$. The associated distribution function is defined as follows:

$$F(x) = \int_{(-\infty, x]} d\mu \tag{44}$$

The distribution functions that usually arise in practice are mixtures of discrete and absolutely continuous distributions: $F(x) = \alpha F_d(x) + (1-\alpha)F_c(x)$ where $0 \leq \alpha \leq 1$, F_d is a discrete distribution over a (countable) set $B \subseteq \mathbb{R}$ and F_c is an absolutely continuous distribution, as explained below.

F_d is of the form:

$$F_d = \lambda x. \sum_{\{y \in B : y \leq x\}} p(y)$$

where $p : \mathbb{R} \to [0, 1]$ is such that

- $p(y) > 0$, for $y \in B$;
- $p(y) = 0$, for $y \notin B$;
- $\sum_y p(y) = 1$.

The function p is called the *probability mass* function associated with F_d with support B.

F_c is of the form:

$$F_c = \lambda x. \int_{-\infty}^{x} f(y)dy$$

for some $f : \mathbb{R} \to [0, \infty)$ such that

- $\int_{-\infty}^{+\infty} f(y)dy = 1$.

The function f is a *probability density* function associated with F_c.

Herein we only consider distribution functions that are mixtures of discrete and absolutely continuous distribution functions.

Now we are ready to give the generalized density function definition of a randomly reactive automaton.

Definition 3.1 A *randomly reactive automaton* is a tuple $\langle I, \approx, M, D \rangle$ where:

- I is a set;
- $\approx$ is an equivalence relation over S;
- $M = \{M^{[s]}\}_{[s] \in [S]}$ where $M^{[s]} \subseteq I$;
- $D = \{D_i^{[s]}\}_{i \in I, [s] \in [S]}$, where $D_i^{[s]} = \langle \alpha_i^{[s]}, p_i^{[s]}, f_i^{[s]} \rangle$ where
  - $\alpha_i^{[s]} \in [0, 1]$;
  - $p_i^{[s]}$ is a probability mass function with support $B_i^{[s]} \subseteq \mathbb{R}$;
  - $f_i^{[s]}$ is a probability density function in $\mathbb{R}$

where $S = (I\mathbb{R})^*$, $[S] = S/\approx$, $[s] = \{s' \in S : s \approx s'\}$ and such that

- $s \approx six$ whenever $s \in L$ and $six \in S \setminus L$;

where $L \subseteq S$ is defined inductively as follows: $\epsilon \in L$ and $six \in L$ iff $s \in L$, $i \in M^{[s]}$ and

$$\alpha_i^{[s]} p_i^{[s]}(x) + \left(1 - \alpha_i^{[s]}\right) f_i^{[s]}(x) > 0. \tag{45}$$

For a Borel measurable function g and a Borel set A, we have

$$\int_A g(x) F_i^{[s]}(dx) = \alpha \sum_{\{x : x \in B_i^{[s]} \cap A\}} g(x) p_i^{[s]}(x) + (1-\alpha) \int_A g(x) f_i^{[s]}(x) dx. \tag{46}$$

Thus if each $F_i^{[s]}$ is absolutely continuous, then we can express the integrals in terms of the probability density functions $\{f_i^{[s]}\}_{i \in I, [s] \in [S]}$ corresponding to the distribution functions $F_i^{[s]}$. In this case, (36) can be rewritten as:

$$\vec{G}^s_{\vec{i}} = \lambda \vec{x} {\in} \mathbb{R}^n . \int_{-\infty}^{\vec{x}_1} \cdots \int_{-\infty}^{\vec{x}_n} f_{\vec{i}_n}^{[s\vec{i}_1 y_1 \vec{i}_2 y_2 \ldots \vec{i}_{n-1} y_{n-1}]}(y_n) \ldots f_{\vec{i}_1}^{[s]}(y_1) dy_n \ldots dy_1.$$

Also if each $F_i^{[s]}$ is discrete, then we can express the integrals in terms of the probability mass functions $\{p_i^{[s]}\}_{i \in I, [s] \in [S]}$ corresponding to the distribution functions $F_i^{[s]}$. In this case, (36) can be rewritten as:

$$\vec{G}^s_{\vec{i}} = \lambda \vec{x} {\in} \mathbb{R}^n . \sum_{\langle y_1, \ldots, y_n \rangle \leq \vec{x}} p_{\vec{i}_n}^{[s\vec{i}_1 y_1 \vec{i}_2 y_2 \ldots \vec{i}_{n-1} y_{n-1}]}(y_n) \ldots p_{\vec{i}_1}^{[s]}(y_1).$$

For a randomly reactive automaton $\langle I, \approx, M, D \rangle$ as defined in Definition 3.1, we have the following.

Definition 3.2 *Let $R \subseteq S$ closed for $\approx$, $\vec{i} \in I^n$, with $\vec{i} = \vec{j}i$, $n \geq 1$ and $s \in S$. The probability of reaching R by $\vec{i}$ from s*

$$Prob(R, \vec{j}i, s) = \int_{-\infty}^{+\infty} ... \int_{-\infty}^{+\infty} 1_{R^s_{\vec{j}i}}(\vec{y}x) F_i^{[\zeta^s_{\vec{j}}(\vec{y})]}(dx) \ldots F_{\vec{j}_1}^{[s]}(d\vec{y}_1).$$

is computed using 46.

3.4.2 Random Reaction Automaton Semantics for the Multivariate Case

We now are able to extend the general density function approach to multivariate reactions. We start by establishing the notion of random reactive automaton and then show how to calculate the relevant probabilities.

Definition 3.3 A *randomly reactive automaton* is a tuple $\langle I, \approx, M, U, D\rangle$ where:

- I is a set;
- $\approx$ is an equivalence relation over S;
- $M = \{M^{[s]}\}_{[s]\in[S]}$ where $M^{[s]} \subseteq I$;
- U is a finite set;
- $D = \{D_i^{[s]}\}_{i\in I,[s]\in[S]}$, where $D_i^{[s]} = \langle \alpha_i^{[s]}, p_i^{[s]}, f_i^{[s]}\rangle$ where
  - $\alpha_i^{[s]} \in [0,1]$;
  - $p_i^{[s]}$ is a probability mass function with support $B_i^{[s]} \subseteq \mathbb{R}^U$;
  - $f_i^{[s]}$ is a probability density function in $\mathbb{R}^U$

where $S = (I\mathbb{R}^U)^*$, $[S] = S/\approx$, $[s] = \{s' \in S : s \approx s'\}$ and such that

- $s \approx si\vec{x}$ whenever $s \in L$ and $si\vec{x} \in S \setminus L$;

where $L \subseteq S$ is defined inductively as follows: $\epsilon \in L$ and $si\vec{x} \in L$ iff $s \in L$, $i \in M^{[s]}$ and

$$\alpha_i^{[s]} p_i^{[s]}(\vec{x}) + \left(1 - \alpha_i^{[s]}\right) f_i^{[s]}(\vec{x}) > 0. \tag{47}$$

The elements in U are the *coordinates.*

Observe that $p_i^{[s]}$ is a map from $\mathbb{R}^U$ to $[0,1]$ with the same properties as in the univariate case and that $f_i^{[s]}$ is a map from $\mathbb{R}^U$ to $[0,+\infty[$ such that $\int_{\mathbb{R}^U} f(\vec{y})d\vec{y} = 1$.

The *probability* of reaching *R by $\vec{i}$ from s*, $Prob(R, \vec{j}i, s)$, is obtained by simple generalization of Definition 3.2.

3.5 Random Reactive Automaton Semantics for Situation Calculus Theories

The random reactive automaton semantics for the multivariate case is almost the same as for the univariate one. Thus, with the multivariate extended situation calculus version we can specify a multivariate randomly reactive automaton $\langle I, \approx, M, U, D \rangle$. Furthermore, the specification SPEC is composed by a finite set U representing the required coordinates, DTT, SCT, proper operation for the sort $\mathcal{I}$, proper operation for the sort $\mathcal{F}$ and proper axioms.

A specification will be a tuple $\langle \mathbb{I}, \mathbb{F}, U, Ax \rangle$ whose signature is the triple $\mathbb{I}, \mathbb{F}, U$.

Consider *Int* to be an interpretation structure for the many sorted higher order language specification, such that: $\mathcal{I}_{Int} = \mathbb{I}$, $\mathcal{I}^*_{Int} = \mathbb{I}^*$, $\mathcal{A}_{Int} = \mathbb{I} \times \mathbb{R}^U$, $\mathcal{F}_{Int} = \mathbb{F}, \mathcal{S}_{Int} = (\mathbb{I}\mathbb{R}^U)^*$.

As expected, for each such *Int* we can build a random reactive automaton $m_{Int} = \langle I, \approx, M, U, D \rangle$, where:

- $I, \approx$ and M are obtained in a similar way to the univariate case (as presented in 2.7);
- $D^{[[\![s]\!]_{Int}]}_{[\![i]\!]_{Int}} = [\![gdf(i,s)]\!]_{Int}$ for each $i \in T_{\mathcal{I}},$ and $s \in T_{\mathcal{S}}$;
- $U = \{1, \ldots, K_U\}$.

Along the same lines of section 2, we say that a consistent SPEC is categoric on the *cdf*s, iff we end up with the same machine no matter what *Int* is used.

The semantics for *prob* is similar to the univariate case:

$$[\![prob(f, \vec{i}, s)]\!]_{Int} = Prob_{m_{Int}}(\{[\![s']\!]_{Int} : Int \Vdash holds(f, s')\}, [\![\vec{i}]\!]_{Int}, [\![s]\!]_{Int}). \tag{48}$$

3.6 Domain Independent Encoding for Multivariate Reactions

As mentioned before, it is possible to write a general encoding for a multivariate reaction probabilistic situation calculus. For simplicity, however, we only provide an encoding for the specific case of bivariate reactions. Also, to illustrate the reasoning capabilities obtained, we make use of the *FrogBot* example of section 3.3.

Here we only present the part of the Mathematica specification that is specific to the multivariate case. Most of the specification remains as in section 2.5.1. For the generation of a vector of random reactions we have:

```
gen[i_,s_] := {idf1[i,s][Random[]],idf2[i, s][Random[]]};
```

where idf1 and idf2 are domain dependent.

The definitions of mcrun for the Monte–Carlo simulations, and prob for the probability estimation are as in section 2.5.1. poss now is as follows:

```
poss[{i_,r_},s_]:=iposs[i,s]∧gdf1[i,s][r[[1]]]gdf2[i,s][r[[2]]]>0;
```

The definition of legal and holds remain as in section 2.5.1.

3.7 Example

The *FrogBot* example of section 3.3 follows. First, we have a set of constants to specify the bog and the target. Both areas are defined by a sector, defined with min and max values for the distance (from the origin) and angles:

```
bogMinDist:=10;
bogMaxDist:=15;
bogMinAngl:=0;
bogMaxAngl:=π/2;

trgMinDist:=20;
trgMaxDist:=25;
trgMinAngl:=π/6;
trgMaxAngl:=π/3;
```

The *gdf* and inverse specifications are:

```
gdf1[move[d_,θ_],s_]:=cdflogistic[d^3/1000]';
idf1[move[d_,θ_],s_]:=idflogistic[d^3/1000];

gdf2[i_,s_]:=cdflogistic[π/30]';
idf2[i_,s_]:=idflogistic[π/30];
```

For ease of implementation, we define some functional fluents:

```
dist[s_]:=Cart2Pol[pos[s]][[1]];
angl[s_]:=Cart2Pol[pos[s]][[2]];
pos[s_]:={posx[s],posy[s]};

posx[s_]:=iota[x, holds[positionx[x],s]];
posy[s_]:=iota[y,holds[positiony[y],s]];
```

The input preconditions:

```
iposs[i_,s_]:=dist[s]≥bogMaxDist∨ dist[s]<bogMinDist∨
              angl[s]≥bogMaxAngl∨ angl[s]<bogMinAngl;
```

Initial conditions:

```
holds[positionx[x_],s0]:= x==0;
holds[positiony[y_],s0]:= y==0;
```

Successor state axioms:

```
 holds[positionx[x_],do[{i_,r_},s_]]:=
      ((i[[0]] === move ∧
            holds[positionx[x - i[[1]]Cos[ i[[2]] ] - r[[1]]Cos[ r[[2]] ]], s])
      ∨ (i[[0]] =!= move ∧ holds[positionx[x, s]]))/;legal[do[{i, r}, s]];

holds[positiony[y_],do[{i_,r_},s_]]:=
     ((i[[0]] === move ∧
            holds[positiony[y - i[[1]]Sin[ i[[2]] ] - r[[1]]Sin[ r[[2]] ]], s])
     ∨ (i[[0]] =!= move ∧ holds[positiony[y, s]]))/;legal[do[{i, r}, s]]
```

Defined Fluents:

```
holds[onbog, s_]:=
     dist[s] < bogMaxDist∧ dist[s]≥ bogMinDist ∧ angl[s] < bogMaxAngl
     ∧ angl[s]≥ bogMinAngl;

holds[ontarget, s_] :=
    dist[s] < trgMaxDist ∧ dist[s] ≥ trgMinDist ∧
      angl[s] < trgMaxAngl ∧ angl[s] ≥ trgMinAngl;
```

It is straightforward to obtain the previous specification from the theoretical specification in the Situation Calculus. Now, we are ready to interact with our oracle. As seen below, the sequence of actions {`move[12,π/4],{1,π/30}`} is legal, and lands the *FrogBot* in the bog. Therefore, the subsequent moves are illegal:

```
legal[do[{move[12,π/4],{1,π/30}},s0]]
True
holds[onbog,do[{move[12,π/4],{1,π/30}},s0]]
```

```
True
legal[do[{move[2,π/4],{0.1,π/20}}, do[{move[12,π/4],{1,π/30}},s0]]]
False
```

Now, by using our Monte–Carlo approach, we obtain estimates for the probabilities of being in the bog or in the target after some specific sequences of inputs:

```
prob[onbog,{move[12,π/4]},s0,1000]
0.779
prob[onbog,{move[12,π/4],move[10,π/4]},s0,1000]
0.79
prob[ontarget,{move[12,π/4],move[10,π/4]},s0,1000]
0.039
%prob[ontarget, {move[22, −π/4], move[22  2, π/2]}, s0, 1000]
%0.035
```

Notice that the error obtained from a move is proportional to the distance that the *FrogBot* attempts jump. Therefore, as shown here, it is more effective to approach the target in smaller jumps. In fact, if one attempts to get to the target in one jump, the probability of actually reaching it is small. This probability improves drastically using a more conservative, less daring, approach:

```
prob[ontarget,{move[22,π/4]},s0,1000]
0.161
prob[ontarget,{move[8,π/4],move[8,π/4],move[8,π/4]},s0,1000]
0.692
prob[ontarget,{move[8,π/4],move[9,π/4],move[5.5,π/4]},s0,1000]
0.941
```

4 Summary, Conclusions and Final Remarks

In this article we addressed the problem of devising a logical language for writing theories of action in domains in which actions may have non-deterministic probabilistic outcomes. The basic ontology includes, among others, a sort for actions, a sort for *inputs*, and a sort for *reactions*. Furthermore, each non-deterministic probabilistic action is modeled as a pair $\langle$*input, reaction*$\rangle$. Given an input, the language allows one to specify a probability distribution on the set of possible outcomes. The set of possible outcomes can be discrete or continuous.

mention that indepence assumptions are common. in applications people use transf. to get indep. vars. from dep. ones.

The contributions of this article can be summarized as follows:

- We developed the notion of *Random Reactive Automata*: These automata provide a semantic structure for the probabilistic situation calculus. The presentation of the automata was separated in two cases: *unidimensional* reaction and *multidimensional* reaction. The first, corresponding to reactions to inputs that can be characterized with a single random variable; the second...
- Probabilistic Situation Calculus: A language of second order logic in which several standard sorts are interpreted (reals, natural numbers, etc.). A method to axiomatize domains is presented.

References

[1] Bacchus, F., Halpern, J., and Levesque, H. Reasoning about Noisy Sensors and Effectors in the Situation Calculus. *Artificial Intelligence 111*, 1-2 (1999), 171–208.

[2] McCarthy, J., and Hayes, P. J. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence* **4**, B. Meltzer and D. Michie, Eds. Edinburgh University Press, Edinburgh, Scotland, 1969, pp. 463–502.

[3] Pinto, J., Sernadas, A., Sernadas, C., and Mateus, P. Probabilistic reasoning within the situation calculus. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (1999). Submitted.

[4] Pinto, J., Sernadas, A., Sernadas, C., and Mateus, P. Non-determinism and Uncertainty in the Situation Calculus. *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems* (2000). To appear.

[5] Pinto, J. A. Compiling Ramification Constraints into Effect Axioms. *Computational Intelligence 15*, 3 (1999), 280–307.

[6] Pirri, F., and Reiter, R. Some Contributions to the Metatheory of the Situation Calculus. *Journal of the ACM* (1999). To appear.

[7] Poole, D. Decision Theory, the Situation Calculus, and Conditional Plans. *Linköping Electronic Articles in Computer and Information Science 3*, 8 (1998). `http://www.ep.liu.se/ea/cis/1998/008/`.

[8] Port, S. *Theoretical Probability for Applications.* Wiley, 1993.

[9] Reiter, R. *The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression.* Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. Academic Press, San Diego, CA, 1991, pp. 359–380.

[10] Reiter, R. KNOWLEDGE IN ACTION: Logical Foundations for Describing and Implementing Dynamical Systems. Book Draft, available from `http://www.cs.utoronto.ca/~cogrobo`, 1999.

[11] Wolfram, S. *The Mathematica Book*, 3 ed. Wolfram Media, 1996.