PREGO: An Action Language for Belief-Based Cognitive Robotics in Continuous Domains[*]

Vaishak Belle and **Hector J. Levesque**

Dept. of Computer Science
University of Toronto
Toronto, Ontario M5S 3H5, Canada
{vaishak, hector}@cs.toronto.edu

Abstract

The area of cognitive robotics is often subject to the criticism that the proposals investigated in the literature are too far removed from the kind of continuous uncertainty and noise seen in actual real-world robotics. This paper proposes a new language and an implemented system, called PREGO, based on the situation calculus, that is able to reason effectively about degrees of belief against noisy sensors and effectors in continuous domains. It embodies the representational richness of conventional logic-based action languages, such as context-sensitive successor state axioms, but is still shown to be efficient using a number of empirical evaluations. We believe that PREGO is a powerful framework for exploring real-time reactivity and an interesting bridge between logic and probability for cognitive robotics applications.

Introduction

Cognitive robotics, as envisioned in (Levesque and Reiter 1998), is a high-level control paradigm that attempts to apply knowledge representation (KR) technologies to the reasoning problems faced by an autonomous agent/robot in an incompletely known dynamic world. This is a challenging problem; at its core, it requires a clear understanding of the relationships among the beliefs, perception, and actions of the agent. To that end, sophisticated knowledge-based proposals for reasoning about action and change have been investigated in the literature, as demonstrated in (De Giacomo, Levesque, and Sardina 2001; Reiter 2001a; Son and Baral 2001; Herzig, Lang, and Marquis 2003), among many others. One major criticism leveled at this line of work, however, is that the theory seems far removed from the kind of continuous uncertainty and noise seen in robotic applications when real sensors and effectors are deployed. The interpreters are often propositional or resort to the closed-world assumption, and at best, only allow limited forms of incomplete information (Reiter 2001a; 2001b). Rather surprisingly, very little attention has been devoted to the integration of action languages with probability densities or degrees of belief. As far as we know, there has yet to emerge a simple specification language that, (a) has the desirable features of popular action formalisms such as context-dependent successor state axioms (Reiter 2001a), (b) allows for expressing discrete and continuous noise in effectors and sensors, and most significantly, (c) is equipped with an effective computational methodology for handling *projection* (Reiter 2001a), the reasoning problem at the heart of planning and agent programming. For real-time reactivity, it is also desirable that the reasoning that is needed (under suitable representational stipulations) be as practical as common filtering techniques (Thrun, Burgard, and Fox 2005).

This paper proposes a system called PREGO as a first step in this direction. Informally, the language of PREGO is built from the following components:

- symbols for the fluents over which one can define a (continuous or discrete) probability distribution;
- action symbols for effectors and sensors, possibly noisy;
- specifications for the preconditions of actions, successor state of fluents, and the results of sensing operations.

In this paper, we study the formal foundations of PREGO, as well as a computational methodology for projecting belief, that is, for computing degrees of belief after any sequence of actions and sensing operations.[1] We show that the language of PREGO can be interpreted as a situation-suppressed dialect of the situation calculus (Reiter 2001a), and that the embodied projection mechanism is a special form of goal regression. From a logical point of view, PREGO's handling of continuity and uncertainty goes beyond the capabilities of popular interpreters for KR action languages. From a probability point of view, PREGO allows successor state and sensing axioms that can be arbitrarily complex (Reiter 2001a), making it not only significantly more expressive than probabilistic formalisms (Boyen and Koller 1998), but also current probabilistic planning formalisms, *e.g.* (Sanner 2011). To the best of our knowledge, a development of this kind has not been considered in this generality before. PREGO is fully implemented, and we also present empirical evaluations of its behavior on non-trivial projection tasks.

We structure the paper as follows. We first introduce PREGO and discuss some very simple belief change examples. We then introduce the background logical language of the situation calculus and study PREGO's techniques. Then, evaluations, related work and conclusions are presented.

[*]Financial support from the Natural Science and Engineering Research Council of Canada made this research possible.

[1]For simplicity, only projection is considered for the language. High-level control structures (Levesque et al. 1997), such as recursion and loops, is left for the future.

PREGO

The PREGO language is a simple representation language with a LISP-like syntax.[2] A domain in PREGO is modeled as a *basic action theory* (or BAT) made up of the following five expressions (which we will illustrate immediately below):[3]

1. `(define-fluents` *fluent fluent* ...`)`
   A list of all the fluents to be used in the BAT. These can be thought of as probabilistic variables, whose values may or may not be known.
2. `(define-ini-p-expr` *expr*`)`
   Here, *expr* is an expression mentioning the fluents from (1) that should evaluate to a number between 0.0 and 1.0. It indicates the probability density given to any initial state in terms of the values of the fluents in that state.
3. `(define-ss-exprs` *fluent act expr act expr* ...`)`
   This determines the successor-state expressions for the given fluent *fluent*. The *act* parameters are of the form `(`*name var var* ...`)` where the *vars* are the arguments of the action and are used in the corresponding *expr*. The idea is that if the action takes place, the new value of the fluent is the value of *expr*. If an action does not appear on the list, the fluent is unchanged by the action.
4. `(define-l-exprs` *act expr act expr* ...`)`
   The format of *act* and *expr* are as in (3). For each *act*, the *expr* is a numerical expression like in (2) and determines the likelihood of that action. If an action does not appear on the list, it is assumed to have a likelihood of 1.0.
5. `(define-alts` *act altfn act altfn* ...`)`
   The format of the *act* is as in (3) and (4). The *altfn* is a function of one argument that produces noisy versions of *act* for that argument. If an action does not appear on the list, then it is exact, that is, without a noisy version.

To illustrate these, let us consider the simple scenario depicted in Figure 1, where a robot is moving in a 1-dimensional world towards a wall. Its distance to the wall is given by a fluent `h`. Suppose:

- The robot initially does not know how far it is from the wall, but that the distance satisfies $2 \leq$ `h` ≤ 12. In other words, the robot believes that the initial value of `h` is drawn from a (continuous) uniform distribution on [2,12].
- The robot has a distance sensor aimed at the wall. The sensor is noisy in that the value `z` read on the sensor differs from the actual value of `h`, but in a reasonable way. We assume that the likelihood of getting `z` is given by a normal distribution whose mean is the true value `h`.
- The robot has an effector for moving exactly `z` units towards or away from the wall, but this motion stops when the wall is reached.

A BAT for this domain is shown in Figure 2. We have a single fluent `h` and two actions: a sensing action `sonar`, and a physical action `fwd`. Note that we can use RACKET arithmetic

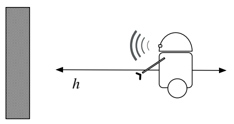

Figure 1: Robot moving towards a wall.

```
(define-fluents h)

(define-ini-p-expr '(UNIFORM h 2 12))

(define-ss-exprs h
    (fwd z) '(max 0 (- h ,z)))

(define-l-exprs
    (sonar z) '(GAUSSIAN ,z h 4.0))
```

Figure 2: PREGO's input for the simple robot domain.

(*e.g.* `max` and `-`) and any other function that can be defined in RACKET (*e.g.* the predefined `UNIFORM` and `GAUSSIAN`). Note also that the *expr* terms are quoted expressions where fluents appear as free variables. (We use backquote and comma to embed the action argument `z` in the expression.)

Once the fluents are defined (here only `h`), the designer provides the initial joint distribution over the fluents using any mathematical function (here, a uniform distribution over one variable).[4] The successor state axiom says that the value of `h` after a `fwd` action is obtained by subtracting `z` from the previous value or 0, whichever is higher. The likelihood expression says that the sonar readings are expected to be centered on the value of `h` with a standard deviation of 4.0. Since the physical action `fwd` is assumed to be noise-free, `define-alts` is not used in this BAT.

Let us now turn to a noisy version of `fwd`. The idea here is that the agent might attempt a move by 3 units but end up actually moving 3.094 units. Unlike sensors, where the reading is nondeterministic, *observable,* but does not affect fluents, the outcome of noisy actions is nondeterministic, *unobservable* and changes fluent properties.

To model this, we imagine a new action symbol `nfwd` with two arguments: the first captures the intended motion amount, and the second captures the actual motion amount. Then, the `ss-exprs` block for the fluent `h` would include:

```
    (nfwd x y) '(max 0 (- h ,y))
```

That is, the true value of `h` changes according to the second argument, not the first. Since the robot does not know the actual outcome, all it knows is that `(nfwd 3 z)` occurred for some value of `z`, which is captured using `define-alts`:

```
(define-alts
    (nfwd x y) (lambda (z) '(nfwd ,x ,z)))
```

Finally, to indicate that this noisy move has (say) Gaussian noise, we would include the following in the `l-exprs` block:

```
    (nfwd x y) '(GAUSSIAN ,y ,x 1.0)
```

In other words, the actual amount moved is normally distributed around the intended value with a variance of 1.

[2]The system is realized in the RACKET dialect of the SCHEME family (`racket-lang.org`). Note that the technical development does not hinge on any feature unique to this programming language.

[3]For space reasons, we omit action preconditions in this paper.

[4]For simplicity, full joint distributions are assumed; when additional structure is available in the form of conditional independencies, then belief networks, among others, can be used (Pearl 1988).

Using PREGO

PREGO can be used to reason about what is believed after any sequence of physical or sensing actions. We use

(eval-bel *expr actions*)

where *expr* is any Boolean expression (with the fluents as free variables) and *actions* is a list of the actions that occurred. For example, after moving 4 units towards the wall, the robot believes it is as likely as not to be within 3 units:

```
> (eval-bel (< h 3) ((fwd 4)))
0.5
```

Note that PREGO is handling a *mixed distribution* here. After the action, the possibility that h = 0 must be accorded a *weight* of .2 (*i.e.* from all points where $h \in [2,4]$ initially), while points where $h \in (0,8]$ retain a *density* of .1.

Likewise, suppose we are interested in the agent's beliefs after sensing the value 5 on its sonar. Clearly, its beliefs should sharpen around 5, and if the robot obtained a second reading of 5.12, its belief would sharpen further. If we were to plot PREGO's computed beliefs for the fluent h after the sequence ((sonar 5) (sonar 5.12)), we would obtain Figure 3. In the following sections, we justify these results and also discuss examples involving the noisy effector.

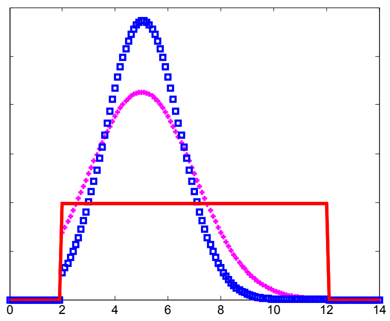

Figure 3: Beliefs about h initially (solid red), after sensing 5 (magenta markers) and after a second sensor reading (blue squares).

Before ending this section, note that, as mentioned, the language allows arbitrary context-dependent successor-state and likelihood specifications (Reiter 2001a). For example, imagine a binary fluent wet which says whether the floor is wet or not. Assume that, initially, it is very likely to be wet:

```
(define-ini-p-expr
     '(* (UNIFORM h 2 12)
         (DISCRETE wet #t .8 #f .2)))
```

Then to model a noisy move whose reliability depends on whether or not the floor is wet, we might have:

```
(define-l-exprs
     (nfwd x y) '(GAUSSIAN ,y ,x (if wet 4.0 1.0)))
```

which says that the actual motion is determined by a Gaussian distribution with variance 4 when the floor is wet but with a variance 1 when the floor is dry. What makes the language rich is that there can be physical actions (*e.g.* mopping up and spilling coffee) that affect this context, as well as sensing actions (*e.g.* observing a reflection on the floor) that sharpen the agent's belief about the context.

Logical Foundations

The PREGO language can be interpreted as a dialect of the situation calculus, extended to reason about degrees of belief. We will not go over that language here, except to note:

- it is many-sorted, with sorts for physical actions, sensing actions, situations, and objects (for everything else);
- a set of initial situations correspond to the ways the world might be initially — a constant S_0 denotes the actual one;
- there is a distinguished binary symbol *do* such that $do(a_1 \cdots a_n, s)$ denotes the situation resulting from performing actions a_1 through a_n at situation s.

Fluents capture changing properties about the world. Here we assume that $f_1, \ldots, f_k$ are all the fluents in the language,[5] and that these take no arguments other than a single situation term. Note that this is not a propositional theory in that we allow the *values* of these fluents to range over any set, including the reals $\mathbb{R}$.

We following two notational conventions. First, we often suppress the situation argument in a logical expression ϕ or use a distinguished variable *now*, and we let $\phi[s]$ denote the formula with that variable replaced by s. Second, we use conditional *if-then-else* expressions in formulas throughout, possibly mentioning quantifiers. We take some liberties with the scope of variables in that $f = \text{If } \exists x.\ \phi \text{ Then } t_1 \text{ Else } t_2$ is to mean $\exists x\ [\phi \wedge f = t_1] \vee [(f = t_2) \wedge \neg\exists x.\ \phi]$.

Basic Action Theory

As hinted in PREGO's presentation, a domain theory $\mathcal{D}$ is formulated as a BAT (Reiter 2001a) which includes sentences $\mathcal{D}_0$ describing what is true initially, successor state axioms (SSA) of the form $\forall a, s.\ f(do(a, s)) = \text{SSA}_f(a)[s]$, and precondition axioms. For example, the effect of the *fwd* action, from the robot domain above, would be expressed as:[6]

$$h(do(a,s)) = (\text{ If } \exists z(a = fwd(z)) \\ \text{Then } \max(0, h - z) \text{ Else } h\ )[s]$$

so as to incorporate Reiter's solution to the frame problem.

For the task of *projection*, we are interested in the entailments of $\mathcal{D}$. Entailment is wrt standard Tarksi models, but we assume that the obvious interpretations are assigned to arithmetic symbols, =, and real constants, such as π and e.

Noise and Degrees of Belief

Our account of probabilistic uncertainty is based on (Belle and Levesque 2013a; Bacchus, Halpern, and Levesque 1999), but augmented here to also deal with *continuous* noisy effectors. These extensions to the situation calculus still benefit from Reiter's solution to the frame problem. They also generalize the Scherl and Levesque (2003) proposal, where actions and sensors are noise-free and beliefs are strictly categorical, that is, non-numeric.

[5]Non-logical symbols, such as fluent and action symbols, in the situation calculus are italicized, *e.g.* fluent h in PREGO is h in the language of the situation calculus.

[6]Free variables are implicitly assumed to be quantified from the outside. Note that SSAs are formulated here using conditional expressions, but they macro expand to Reiter's formulation.

Mirroring PREGO's presentation, our account involves 3 distinguished symbols: l, alt and p. $\mathcal{D}$ would now contain l-axioms of the form $l(\alpha(\vec{x}), s) = \mathrm{L}_\alpha(\vec{x})[s]$, and alt-axioms of the form $alt(a, u) = a'$. For example, $nfwd$ is modeled by including the appropriate SSA and the following in $\mathcal{D}$:[7]

$$l(nfwd(x, y), s) = \mathcal{N}(y; x, 1)[s]. \tag{1}$$

$$alt(nfwd(x, y), z) = nfwd(x, z). \tag{2}$$

Finally, the distinguished symbol p can be seen as a *numeric* variant of the accessibility relation in epistemic logics. Intuitively, $p(s', s)$ is the density that the agent attributes to s' when at s (Belle and Levesque 2013a). As part of $\mathcal{D}_0$, the modeler would provide the probability distribution on the agent's initial worlds, using a sentence of the form

$$p(s, S_0) = \mathrm{INIT}(f_1, \ldots, f_k)[s].$$

For example,

$$p(s, S_0) = \mathcal{U}(h; 2, 12)[s] \tag{3}$$

embodies the initial uncertainty of our robot from Figure 1.

Given such axioms in $\mathcal{D}$, the *degree of belief* in any situation-suppressed ϕ in s is defined using the abbreviation:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \int_{f_1,\ldots,f_k} \int_{u_1,\ldots,u_n} Density(\phi, s^*)$$

where the normalization factor γ is the numerator but with ϕ replaced by *true*, and if $s = do(a_1 \cdots a_n, S_0)$ then $s^* = do(alt(a_1, u_1) \cdots alt(a_n, u_n), S_0)$.

The idea behind *Density* is simple. Starting from s, the density of $do(a_1 \cdots a_n, s)$ is the p-value of s times the likelihoods of each a_i. By integrating over $\vec{u}$, $alt(a_i, u_i)$ is taken into account, and so, all possible successors resulting from noisy actions are also considered.[8] For space reasons, we omit the definition; see (Belle and Levesque 2013a). We simply note that the belief change mechanism subsumes Bayesian conditioning (Pearl 1988), as used in the robotics literature (Thrun, Burgard, and Fox 2005).

Computing Projection

The projection mechanism seen in `eval-bel` is built on a special form of *goal regression*. In other words, we begin by finding a situation-suppressed expression r such that

$$\mathcal{D} \models Bel(\phi, do(a_1 \cdots a_n, S_0)) = r[S_0].$$

Because only $\mathcal{D}_0$ is needed to calculate $r[S_0]$, this reduces the calculation of belief after actions and sensing to a calculation in the initial situation in terms of INIT.

Such a regression operator is formulated in (Belle and Levesque 2013b). They show how *Bel*-expressions about the future reduce to *Density*-expressions about S_0. However, their proposal does not deal with noisy effectors. Moreover, their *Density*-expressions expand into formulas that quantify over initial situations. Consequently, considerable logical machinery is needed to further simplify these sentences to a purely numerical formula.

What we propose here is a new treatment that not only generalizes to both noisy acting and sensing, but one that involves only mathematical (as opposed to logical) expressions. Roughly, this is achieved by processing the logical terms in the goal in a *modular* fashion wrt the situation-suppressed RHS (also interpreted as logical terms) of the axioms in $\mathcal{D}$. The result is a Boolean expression (about S_0), where fluents are free variables, that can be evaluated using any software with numerical integration capabilities. In other words, no logical consequence finding is necessary.

Definition 1: Given a BAT $\mathcal{D}$, a situation-suppressed expression e, and an action sequence σ, we define $\mathcal{R}[e, \sigma]$ as a situation-suppressed formula e' as follows:

1. If e is a fluent:
   - if $\sigma = \epsilon$ (is empty), then $e' = e$;
   - if $\sigma = \sigma' \cdot a$ then $e' = \mathcal{R}(\mathrm{SSA}_e(a), \sigma')$.
2. If e is a number, constant or variable, then $e' = e$.
3. If e is $Bel(\phi, now)$ then
   $$e' = \frac{1}{\gamma} \int_{\vec{f}} \mathrm{INIT} \times \mathcal{G}[\phi, \sigma]$$
   where $\mathcal{G}$ is an operator for obtaining a (mathematical) expression from the belief argument ϕ, defined below.
4. Else e is $(e_1 \circ e_2 \circ \ldots \circ e_n)$ and
   $$e' = (\mathcal{R}[e_1, \sigma] \circ \mathcal{R}[e_2, \sigma] \circ \ldots \circ \mathcal{R}[e_n, \sigma])$$
   where $\circ$ is any mathematical operator over expressions, such as $\neg$, $\wedge$, $=$, $+$, IF, $\mathcal{N}$, etc.

As in (Reiter 2001a), fluents are simplified one action at a time using appropriately instantiated SSAs. The main novelty here is how *Bel* is regressed using INIT, the RHS of the p-axiom in $\mathcal{D}_0$, with the argument ϕ handled separately, and how $\mathcal{R}$ works over arbitrary mathematical functions in a modular manner; *e.g.* $\mathcal{R}[\mathcal{N}(t_1; t_2, t_3), \sigma]$ would give us $\mathcal{N}(\mathcal{R}[t_1, \sigma]; \mathcal{R}[t_2, \sigma], \mathcal{R}[t_3, \sigma])$. Now, $\mathcal{G}$:

Definition 2: Let $\mathcal{D}$ and σ be as above. Given any situation-suppressed fluent formula ϕ, we define $\mathcal{G}[\phi, \sigma]$ to be a situation-suppressed expression e as follows:

1. If $\sigma = \epsilon$, then $e =$ IF ϕ THEN 1 ELSE 0.
2. Else, $\sigma = \sigma' \cdot \alpha(t)$; let $\alpha(t') = alt(\alpha(t), u)$ and
   $$e = \int_u \mathcal{R}[\mathrm{L}_\alpha(t'), \sigma'] \times \mathcal{G}[\mathcal{R}[\phi, \alpha(t')], \sigma'].$$

Essentially, $\mathcal{G}$ integrates over all possible outcomes for a noisy action using alt. The likelihood of these outcomes is determined using L_α, the RHS of the l-axioms.

For our main theorem, $\mathcal{R}$ is shown to have this property:

Theorem 3: *Let $\mathcal{D}$ and σ be as above, and let e be any situation-suppressed expression. Then*

$$\mathcal{D} \models e[do(\sigma, S_0)] = (\mathcal{R}[e, \sigma])[S_0].$$

When e is a belief formula, this allows us to reduce the belief calculation to the initial situation, as desired:

Corollary 4: *Let $\mathcal{D}, \phi$ and σ be as above. Then*

$$\mathcal{D} \models Bel(\phi, do(\sigma, S_0)) = (\mathcal{R}[Bel(\phi, now), \sigma])[S_0].$$

[7]We use $\mathcal{N}$ and $\mathcal{U}$ as abbreviations for mathematical formulas defining a Gaussian and uniform density respectively.

[8]For discrete fluents and discrete noisy effectors, one would replace $\int_f$ and $\int_u$ by $\sum_f$ and $\sum_u$ respectively. Let us also remark that both summations and integrals can be defined as terms in the logical language, as shown in (Belle and Levesque 2013a).

From Specification to PREGO

$\mathcal{R}$ is a reduction operator that takes as input any Boolean expression (where fluents are free variables) and outputs a new one, leading to a surprisingly straightforward implementation that exactly follows Definitions 1 and 2. We demonstrate this using a BAT. Let $\mathcal{D}$ include the situation calculus counterparts for our robot domain, *i.e.* (1), (2), (3) and:

$$l(sonar(z), s) = \mathcal{N}(z; h, 4)[s] \tag{4}$$

$$h(do(a, s)) = (\text{ IF } \exists x, y\ (a = nfwd(x, y)) \text{ THEN } \max(0, h - y) \text{ ELSE } h\ )[s] \tag{5}$$

Assume now that the robot *senses* 5 initially on the sonar, and then it attempts a noisy move of 2 units *away* from the wall (by providing a negative argument to *nfwd*). In reality, assume a move of 2.1 units occurs. (As we shall see, the second argument of *nfwd* determines the change to the *h* fluent, but does not affect beliefs about *h*, and so it can be any arbitrary number.) Suppose we are further interested in the robot's beliefs about $\phi = (h \le 7)$. $\mathcal{R}$ works as follows:

$$\begin{aligned}
&\mathcal{R}[Bel(\phi, now), do(sonar(5) \cdot nfwd(-2, -2.1), S_0)] \\
&= \frac{1}{\gamma} \int_h \mathcal{U}(h; 2, 12) \times \mathcal{G}[\phi, sonar(5) \cdot nfwd(-2, -2.1)] \\
&= \frac{1}{\gamma} \int_h \mathcal{U}(h; 2, 12) \times \int_u \begin{pmatrix} \mathcal{R}[\mathrm{L}_{nfwd}(-2, u), sonar(5)] \ \times \\ \mathcal{G}[\mathcal{R}[\phi, nfwd(-2, u)], sonar(5)] \end{pmatrix} \\
&= \frac{1}{\gamma} \int_h \mathcal{U}(h; 2, 12) \times \int_u \mathcal{N}(u; -2, 1) \times \mathcal{G}[\psi, sonar(5)] \\
&= \frac{1}{\gamma} \int_h \mathcal{U}(h; 2, 12) \times \int_u \mathcal{N}(u; -2, 1) \times \mathcal{N}(5; h, 4) \times \mathcal{G}[\psi, \epsilon]
\end{aligned}$$

where $\psi = (\mathcal{R}[\phi, nfwd(-2, u)]) = (\max(0, h - u) \le 7)$, and $\mathcal{G}[\psi, \epsilon] = \text{IF } \psi \text{ THEN } 1 \text{ ELSE } 0$.

In the PREGO system, the supporting regression can be examined using a second function, `regr-bel`, as follows:

```
> (regr-bel (<= h 7) ((sonar 5) (nfwd -2 -2.1)))
'(/
   (INTEGRATE (h u)
     (* (UNIFORM h 2 12) (GAUSSIAN u -2 1.0)
        (GAUSSIAN 5 h 4.0)
        (if (<= (max 0 (- h u)) 7) 1.0 0.0)))
   (INTEGRATE (h w)
     (* (UNIFORM h 2 12) (GAUSSIAN w -2 1.0)
        (GAUSSIAN 5 h 4.0))))
```

The answer is a quotient of two integrals (or summations in the discrete case), where the denominator is the normalization factor. The integrand of the numerator is a product of four terms: the first coming from INIT, the next two coming from the noisy acting and sensing, and the final due to the regression of the argument of *Bel*. What `eval-bel` then does is to evaluate these integrals numerically using Monte Carlo sampling (Murphy 2012):[9]

```
> (eval-bel (<= h 7) ((sonar 5) (nfwd -2 -2.1)))
0.47595449413426844
```

[9]For standard distributions such as $\mathcal{N}$ and $\mathcal{U}$, points can be generated for $f_1, \ldots, f_k$ (*i.e.* the fluents) and $u_1, \ldots, u_n$ (*i.e.* the new integration variables introduced for noisy actions) using INIT and the *l*-axioms respectively. These points are then tested for the regression of the argument to *Bel* (*e.g.* $\mathcal{G}[\psi, \epsilon]$ above).

Evaluations

We consider the empirical behavior of PREGO in this section, where we measure the CPU time in milliseconds (ms) on non-trivial projection tasks, which is then contrasted with classical regression in a manner explained shortly. For the experimental setup, we imagine a planner to have generated a number of plans of increasing length. For our purpose, these plans are randomly generated action sequences, possibly involving combinations of noisy and noise-free actions and sensors, and thus, are representative of the plan search space. All experiments were run wrt the simple robot domain,[10] on Mac OS X 10.8 using a system with a 2.4 GHz Intel Core 2 Duo processor, 4 GB RAM, and RACKET v5.3.6.

On Regression We first study PREGO's query reduction alone, that is, the CPU time for `regr-bel` over plans with purely noisy actions. The average times, over 50 trials, are reported in the table below: we find that regression is very effective, taking less than 6700 ms for 10^6 noisy actions.

Plan length	20	80	500	5000	10^4	10^6
Time (ms)	0	1	3	31	64	6680

On Noise Next, we consider calculating degrees of beliefs using `eval-bel`: (Q1) plans with purely noisy acting; (Q2) plans with interleaved noisy acting and sensing; (Q3) plans with purely noise-free actions. Average times are shown in Figure 4. Surprisingly, we find Q2 > Q1 > Q3. Note for a plan of length *n*, for Q3, we integrate over just one variable (the fluent *h*), for Q1 we integrate over $n + 1$ variables since each noisy action introduces a new integration variable (*e.g.* 100 integrals for 99 noisy actions), and for Q2 we integrate over $n/2 + 1$ variables since sensing does not introduce a new variable, but rather a new term in the expression being integrated (see `regr-bel` above). So the runtimes reflect the fact that the integrand is much more complex in Q2.

On Determinization In discrete domains, beliefs can also be calculated using classical regression (Reiter 2001a), by means of the following *determinizations*: (D1) reason about the initial uncertainty by regressing plans wrt each of the possible initial states, and (D2) reason about noisy actions in plans by regressing wrt all possible outcomes. We are now interested in demonstrating that PREGO's belief-based methodology outperforms classical regression used for this purpose. To maximize objectivity, classical regression is evaluated using PREGO itself as given by Definition 1's item 1, 2 and 4, which then corresponds to Reiter's (2001a).

In continuous domains, of course, one further needs *a-priori* discretizations. For D2, we crudely discretize the normally-distributed *nfwd* as a discrete noisy action with 3 possible outcomes. We will further discretize the initial belief that *h* is uniformly distributed on [2,12] in two ways: divisions of .5 units thereby leading to 20 possible initial states (D1/20), and divisions of .1 units thereby leading to 100 possible initial states (D1/100). The plots are provided in Fig-

[10]Analogous experiments with similar developments have been performed on real-world planning domains (Meuleau et al. 2009) and context-dependent specifications, which are left for a longer version of the paper.

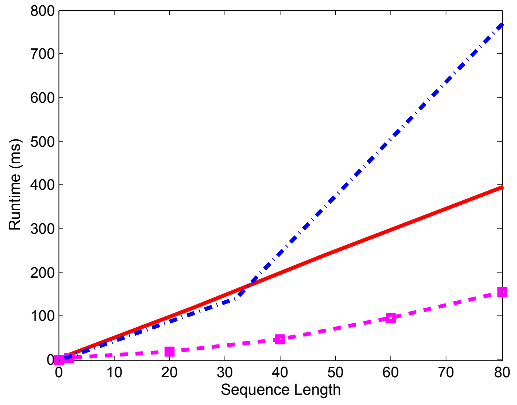

Figure 4: Q1 (red), Q2 (dashed blue), Q3 (magenta with markers).

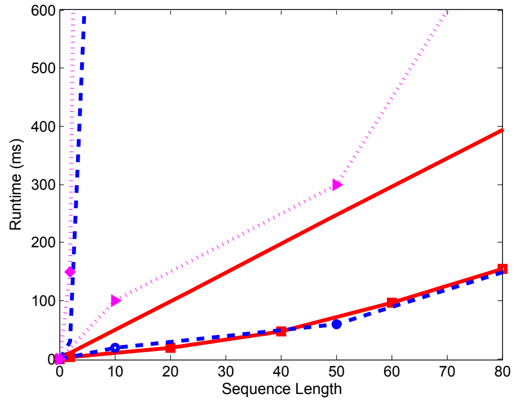

Figure 5: Q1 (solid red), Q3 (solid red with squares), D1/20 over noise-free plans (dashed blue with circles), D2+D1/20 over noisy plans (undecorated dashed blue), D1/100 over noise-free plans (dotted magenta with triangle markers), and D2+D1/100 over noisy plans (dotted magenta with diamond markers).

ure 5: Q3 and D1/20 are comparable, but D1/100 is more expensive than Q1; that is, classical regression with D1/100 over *noise-free* plans is more intensive than *continuous* belief regression over *noisy* actions. Finally, when considering even a 3-outcome *nfwd* discretization, we find that Q1 and D2+D1/100 are far apart: regressing beliefs over a plan length of 96 noisy actions takes 472 ms *vs.* classical regression with D1/100 and 10 noisy actions needs $\approx 3 \times 10^6$ ms.

Related Work

The PREGO framework is based on the situation calculus. While this language is quite expressive, popular interpreters either make the closed-world assumption, or only allow certain kinds of disjunctive knowledge (Reiter 2001a; Finzi, Pirri, and Reiter 2000; Fan et al. 2012). In other words: no degrees of belief. The notable exception to this is the MDP-inspired DTGOLOG and related proposals (Reiter 2001a). Although a full comparison is difficult since DTGOLOG implements a particular planning methodology while PREGO is just a specification language, there are significant differences: DTGOLOG is for fully observable domains and only supports discrete probabilities. There has been recent work on POMDP extensions (Sanner and Kersting 2010; Zamani et al. 2012). However, they assume discrete noise in effectors, and make other strong structural assumptions, such as context-free (STRIPS-style) actions. In our view, context-dependent SSAs are one of the reasons to consider using a language like the situation calculus in the first place.

While the situation calculus has received a lot of attention, there are, of course, other action languages; *e.g.* see (Thielscher 2001; Van Benthem, Gerbrandy, and Kooi 2009; Iocchi et al. 2009) for treatments on probabilities in other formalisms. They are, however, limited to discrete probabilities. Thus, we differ from these and many others (Poole 1998; Mateus et al. 2001; Grosskreutz and Lakemeyer 2003b; 2003a) in being able to address continuity in a general way, except for (Belle and Levesque 2013a) that we build on. See (Bacchus, Halpern, and Levesque 1999; Belle and Levesque 2013a) for more discussions. In addition, PREGO is seen to be more expressive than current probabilistic planning languages (Kushmerick, Hanks, and Weld 1995; Younes and Littman 2004; Sanner 2011). Other continuous models for planning, such as (Meuleau et al. 2009), are procedural rather than declarative, and do not support contextual SSAs. As argued in (Bacchus, Halpern, and Levesque 1999), the same representational limitations also apply to most probabilistic formalisms, such as Kalman Filters and Dynamic Bayesian Networks (Boyen and Koller 1998).[11] Finally, recent work on relational probabilistic languages (Richardson and Domingos 2006; Milch et al. 2005) and probabilistic programming (Goodman et al. 2008) feature sophisticated stochastic models, but do not handle actions.

Conclusions

This paper proposed a new declarative representation language, and studied a formal and computational account for projection with degrees of beliefs. The language allows for discrete and continuous fluents, noisy actions and noisy sensors. It incorporates important features of action languages, thereby providing an interesting bridge between realistic robotic concerns, on the one hand, and logic-based representation languages, on the other. We also reported on empirical studies that demonstrate why we feel that the PREGO system is powerful enough to explore real-time reactivity in cognitive robotics applications. To the best of our knowledge, no other proposal of this generality has been investigated.

There are two main avenues for the future. First, progression. The regression system in PREGO maintains the initial state, and is appropriate for planning. For some applications, it is desirable to periodically update the system (Vassos and Levesque 2008). Following (Belle and Levesque 2014), we would like to explore progression in PREGO and perhaps evaluate that against particle filters (Thrun, Burgard, and Fox 2005). Second, in a companion paper, we intend to consider the equivalent of GOLOG's program structures for PREGO.

[11] As shown in (Belle and Levesque 2013b), when the BAT is restricted to normally-distributed fluents and effectors, regression can be shown to yield expressions identical to a Kalman filter. However, Kalman filters only maintain the current world state, and so they correspond to a kind of progression (Vassos and Levesque 2008).

References

Bacchus, F.; Halpern, J. Y.; and Levesque, H. J. 1999. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence* 111(1–2):171 – 208.

Belle, V., and Levesque, H. J. 2013a. Reasoning about continuous uncertainty in the situation calculus. In *Proc. IJCAI*.

Belle, V., and Levesque, H. J. 2013b. Reasoning about probabilities in dynamic systems using goal regression. In *Proc. UAI*.

Belle, V., and Levesque, H. J. 2014. How to progress beliefs in continuous domains. In *Proc. KR*.

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In *Proc. UAI*, 33–42.

De Giacomo, G.; Levesque, H.; and Sardina, S. 2001. Incremental execution of guarded theories. *ACM Transactions on Computational Logic* 2(4):495–525.

Fan, Y.; Cai, M.; Li, N.; and Liu, Y. 2012. A first-order interpreter for knowledge-based golog with sensing based on exact progression and limited reasoning. In *Proc. AAAI*.

Finzi, A.; Pirri, F.; and Reiter, R. 2000. Open world planning in the situation calculus. In *Proc. AAAI*, 754–760.

Goodman, N. D.; Mansinghka, V. K.; Roy, D. M.; Bonawitz, K.; and Tenenbaum, J. B. 2008. Church: a language for generative models. In *UAI*, 220–229. AUAI Press.

Grosskreutz, H., and Lakemeyer, G. 2003a. ccgolog – a logical language dealing with continuous change. *Logic Journal of the IGPL* 11(2):179–221.

Grosskreutz, H., and Lakemeyer, G. 2003b. Probabilistic complex actions in golog. *Fundam. Inform.* 57(2-4):167–192.

Herzig, A.; Lang, J.; and Marquis, P. 2003. Action representation and partially observable planning using epistemic logic. In *Proc. IJCAI*, 1067–1072.

Iocchi, L.; Lukasiewicz, T.; Nardi, D.; and Rosati, R. 2009. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. *ACM Transactions on Computational Logic* 10:5:1–5:41.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76(1):239–286.

Levesque, H., and Reiter, R. 1998. High-level robotic control: Beyond planning. Position paper at AAAI Spring Symposium on Integrating Robotics Research.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Mateus, P.; Pacheco, A.; Pinto, J.; Sernadas, A.; and Sernadas, C. 2001. Probabilistic situation calculus. *Annals of Math. and Artif. Intell.* 32(1-4):393–431.

Meuleau, N.; Benazera, E.; Brafman, R. I.; Hansen, E. A.; and Mausam. 2009. A heuristic search approach to planning with continuous resources in stochastic domains. *J. Artif. Intell. Res. (JAIR)* 34:27–59.

Milch, B.; Marthi, B.; Russell, S. J.; Sontag, D.; Ong, D. L.; and Kolobov, A. 2005. BLOG: Probabilistic models with unknown objects. In *Proc. IJCAI*, 1352–1359.

Murphy, K. P. 2012. *Machine learning: a probabilistic perspective*. The MIT Press.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Poole, D. 1998. Decision theory, the situation calculus and conditional plans. *Electron. Trans. Artif. Intell.* 2:105–158.

Reiter, R. 2001a. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press.

Reiter, R. 2001b. On knowledge-based programming with sensing in the situation calculus. *ACM Trans. Comput. Log.* 2(4):433–457.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine learning* 62(1):107–136.

Sanner, S., and Kersting, K. 2010. Symbolic dynamic programming for first-order pomdps. In *Proc. AAAI*, 1140–1146.

Sanner, S. 2011. Relational dynamic influence diagram language (rddl): Language description. Technical report, Australian National University.

Scherl, R. B., and Levesque, H. J. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1-2):1–39.

Son, T., and Baral, C. 2001. Formalizing sensing actions– a transition function based approach. *Artificial Intelligence* 125(1-2):19–91.

Thielscher, M. 2001. Planning with noisy actions (preliminary report). In *Proc. Australian Joint Conference on Artificial Intelligence*, 27–45.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT Press.

Van Benthem, J.; Gerbrandy, J.; and Kooi, B. 2009. Dynamic update with probabilities. *Studia Logica* 93(1):67–96.

Vassos, S., and Levesque, H. 2008. On the Progression of Situation Calculus Basic Action Theories: Resolving a 10-year-old Conjecture. In *Proc. AAAI*, 1004–1009.

Younes, H., and Littman, M. 2004. PPDDL 1. 0: An extension to pddl for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University.

Zamani, Z.; Sanner, S.; Poupart, P.; and Kersting, K. 2012. Symbolic dynamic programming for continuous state and observation POMDPs. In *NIPS*, 1403–1411.