Nonmonotonic Reasoning, Action and Change

Proceedings of the Tenth International Workshop (NRAC 2013)

Edited by

Jianmin Ji
University of Science and Technology of China, China

Hannes Strass
Leipzig University, Germany

Xun Wang
University of Technology, Sydney, Australia

Preface

We present here the informal proceedings for the Tenth International Workshop on Non-Monotonic Reasoning, Action and Change (NRAC 2013). NRAC is a well-established forum to foster discussion and sharing of experiences among researchers interested in the broad areas of nonmonotonic reasoning and reasoning about action and change, including belief revision, planning, logic programming, argumentation, causality, probabilistic and possibilistic approaches to KR, and other related topics.

Since its inception in 1995, NRAC has always been held in conjunction with the International Joint Conference on Artificial Intelligence (IJCAI). Previous editions were held in 2011 in Barcelona, Spain; in 2009 in Pasadena, USA; in 2007 in Hyderabad, India; in 2005 in Edinburgh, Scotland; in 2003 in Acapulco, Mexico; in 2001 in Seattle, USA; in 1999 in Stockholm, Sweden; in 1997 in Nagoya, Japan; and in 1995 in Montreal, Canada. This time, NRAC 2013 is held as a one-day satellite workshop of IJCAI 2013, in Beijing, China, and will take place on 5 August 2013.

In addition to the paper sessions, this year's workshop features invited talks by two internationally renowned researchers: Xiaoping Chen from the University of Science and Technology of China, and Joohyung Lee from Arizona State University, USA.

The programme chairs would like to thank all authors for their contributions and are also very grateful to the programme committee for their work during the review phase and for providing excellent feedback to the authors. This is especially notable as the review phase was unusually short this time. The programme chairs are also very grateful to Mary-Anne Williams, Maurice Pagnucco, Gerhard Brewka and Leora Morgenstern from the steering committee for always being available for consultation.

May 2013

Jianmin Ji
Hannes Strass
Xun Wang

Programme Chairs

Jianmin Ji	School of CS, University of Science and Technology of China, China
Hannes Strass	Leipzig University, Germany
Xun Wang	University of Technology, Sydney, Australia

Programme Committee

Xiaoping Chen	University of Science and Technology of China, China
Jens Claßen	RWTH Aachen University, Germany
Eduardo Fermé	University of Madeira, Portugal
Alfredo Gabaldon	New University of Lisbon, Portugal
Laura Giordano	Università del Piemonte Orientale, Italy
Jérôme Lang	Université Paris-Dauphine, France
Joohyung Lee	Arizona State University, USA
Fangzhen Lin	Hong Kong University of Science and Technology, Hong Kong
Thomas Meyer	Meraka Institute, South Africa
Leora Morgenstern	SAIC Advanced Systems and Concepts, USA
Sebastian Sardiña	RMIT University, Australia
Eugenia Ternovska	Simon Fraser University, Canada
Matthias Thimm	University of Koblenz-Landau, Germany
Ivan José Varzinczak	Meraka Institute, South Africa
Stavros Vassos	Sapienza University of Rome, Italy
Renata Wassermann	University of São Paulo, Brazil
Dongmo Zhang	University of Western Sydney, Australia

Additional Reviewers

Gavin Rens	Meraka Institute, South Africa

NRAC Steering Committee

Gerhard Brewka	Leipzig University, Germany
Michael Thielscher	The University of New South Wales, Australia
Leora Morgenstern	SAIC Advanced Systems and Concepts, USA
Maurice Pagnucco	The University of New South Wales, Australia
Pavlos Peppas	University of Patras, Greece
Mary-Anne Williams	University of Technology, Sydney, Australia
Andreas Herzig	Université Paul Sabatier, France
Benjamin Johnston	University of Technology, Sydney, Australia

Table of Contents

Reasoning about Motion Kinematics with Continuous Uncertainty in the Situation Calculus[*]

Vaishak Belle and **Hector J. Levesque**
Dept. of Computer Science
University of Toronto
Toronto, Ontario M5S 3H5, Canada
{vaishak, hector}@cs.toronto.edu

Abstract

Motion kinematics and probabilistic state estimation tasks are fundamental reasoning concerns in robotic applications, where the world is uncertain, and sensors and effectors are noisy. Most systems make various assumptions about the dependencies between state variables, and especially about how these dependencies change as a result of actions. Building on a general framework by Bacchus, Halpern and Levesque for reasoning about degrees of belief in the situation calculus, and a recent extension to it for continuous domains, in this paper we investigate the above reasoning concerns in the presence of a rich theory of actions using an example. We also show that while actions might affect prior distributions in nonstandard ways, suitable posterior beliefs are nonetheless entailed as a side-effect of the overall specification.

1 Introduction

An intelligent agent operating in a dynamic and uncertain world faces at least two major sorts of reasoning problems.[1] First, because the world is *dynamic*, actions perpetually change the properties of the state. Second, because little in the world is definite, the agent has to modify its beliefs based on the actions it performs and its sensor measurements, both of which are prone to noise. At one extreme, for representing incomplete knowledge, dynamical behavior using high-level actions, and control imperatives, logical formalisms seem appropriate. The *situation calculus* [McCarthy and Hayes, 1969], which serves as the foundation for many planning approaches [Reiter, 2001; Fritz and McIlraith, 2007], is one such candidate. The language is first-order and it exploits regularities in the effects that actions have on propositions to describe physical laws compactly. For example, the breaking of a fragile object held by the agent against a broken one getting fixed is written as

$$\begin{aligned}\forall a, s.\ & \mathit{Broken}(x,do(a,s)) \equiv \\ & a = \mathit{drop}(x) \wedge \mathit{Holding}(x,s) \wedge \mathit{Fragile}(x) \vee \\ & \mathit{Broken}(x,s) \wedge a \neq \mathit{repair}(x).\end{aligned}$$

[*]The authors would like to thank the reviewers for constructive feedback, and the Natural Sciences and Engineering Research Council of Canada for financial support.

[1]The following discussion is taken, with modifications, from [Belle and Levesque, 2013c].

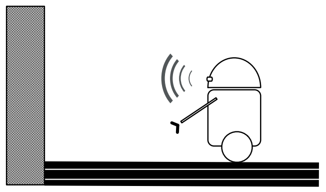


Figure 1: Robot facing a wall (the ground level view).

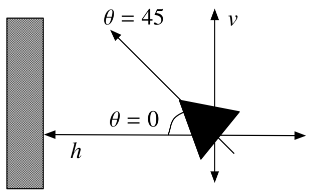


Figure 2: Pose of the robot on the planar surface (the overhead view).

At the other extreme, revising beliefs after noisy observations and similar temporal phenomena is effortlessly addressed using probabilistic models such as Kalman filtering and Dynamic Bayesian Networks [Dean and Kanazawa, 1989; Dean and Wellman, 1991]. Unfortunately, while belief update of known priors over Gaussian and other continuous error models is treated appropriately there, very little is said about how actions might change values of certain state variables while not affecting others. These formalisms also assume a full specification of the dependencies between variables, making it difficult to deal with other forms of incomplete knowledge, and strict uncertainty in particular.

The above issues bring to forefront concerns about integrating high-level actions, incomplete information, and probabilistic state estimation in a general way. To see a simple example, imagine a robot operating in a planar surface, located at a certain distance h to the right of a wall and oriented at a certain angle, as in Figure 1 and Figure 2. Suppose the robot initially believes that $\theta = 0$ and that h is drawn from a uniform distribution on $[2, 12]$. Among the robot's many capabilities, we imagine the ability of moving, which might be predicated on factors such as the ground's slipperiness. Then, a leftwards motion of 1 unit would shift the uniform distribution on h to $[1, 11]$, but a leftward motion of 5 units would change the distribution more radically. The point $h = 0$ would now obtain a weight of .3, while $h \in (0, 7]$ would retain their densities. This mixed distribution would then be preserved by a subsequent rightward motion. Likewise, we might imagine the robot to be equipped with two onboard sensors: a laser unit aimed at the wall estimating h, and a sonar unit that also estimates the distance between the robot and the wall. Each of these might be characterized by Gaussian error models, and the effect of a reading from any sensor would revise the distribution on h from uniform to an appropriate Gaussian. Of course, if the robot is uncertain about θ's value, its belief about h after moving and sensing would be some non-trivial function of θ. In the end, the robot is left with the difficult task of adjusting its beliefs as it operates and obtains competing (perhaps conflicting) measurements from individual sensors.

Perhaps the most general formalism for dealing with probabilistic belief in formulas, and how that should evolve in the presence of noisy acting and sensing, is a logical account by Bacchus, Halpern and Levesque (BHL) [1999]. In the BHL approach, besides quantifiers and other logical connectives, one has the provision for specifying the *degrees of belief* in formulas in the initial state. This specification may be compatible with one or very many initial distributions and sets of independence assumptions. All the properties of belief will then follow at a corresponding level of specificity.

Subjective uncertainty is captured in the BHL scheme using a possible-world model of belief [Fagin *et al.*, 1995]. In classical possible-world semantics, a formula ϕ is believed to be true when ϕ comes out true in all possible worlds that are deemed accessible. In BHL, the degree of belief in ϕ is defined as a normalized sum over the possible worlds where ϕ is true of some nonnegative *weights* associated with those worlds. To reason about belief change, the BHL model is then embedded in a rich theory of action and sensing provided by the situation calculus [McCarthy and Hayes, 1969; Reiter, 2001; Scherl and Levesque, 2003]. The BHL account provides axioms in the situation calculus regarding how the weight associated with a possible world changes as the result of acting and sensing. The properties of belief and belief change then emerge as a direct logical consequence of the initial specifications and these changes in weights.

However, in contrast to the earlier mentioned Bayesian formalisms, one of the limitations of the BHL approach is that it is restricted to fluents whose values are drawn from discrete countable domains. One could say, for example, that $h \in \{2, 3, \ldots, 11\}$ is given an equal weight of .1, but stipulating a continuous uniform distribution and Gaussian sensor error models (and not discrete approximations thereof) is quite beyond the BHL approach. In recent work [Belle and Levesque, 2013a], it is shown how with minimal additional assumptions this serious limitation of BHL can be lifted.

In this paper, we investigate how the (generalized) BHL scheme might be utilized for kinematic configurations and state estimation using our example consisting of a robot, a laser and a sonar device.[2] Our example supposes that the robot is capable of deterministic physical actions, while the sensors are characterized by continuous error models. We first consider the case where the robot is certain about its orientation, and demonstrate properties about belief change. We then move to a case where the robot has uncertainty regarding its orientation, and show how beliefs behave in this more difficult setting. Thus, the domain formalization illustrates belief change wrt shifting densities as logical properties of actions, competing sensors, and uncertain orientations, among others. Since no assumptions need to be made in general regarding the kind of distributions that initial state variables are drawn from, nor about dependencies between state variables, this work illustrates how beliefs about the robot's location would change after acting and sensing in complex noisy domains.

The paper is structured as follows. In the next section, we briefly review formal preliminaries, such as the situation calculus, the BHL scheme as well as the essentials of its generalization to continuous domains. We then model the robot domain and illustrate properties about belief change. In the final sections, we discuss related and future work.

2 Preliminaries

The language $\mathcal{L}$ of the situation calculus [McCarthy and Hayes, 1969] is a many-sorted dialect of predicate calculus, with sorts for *actions*, *situations* and *objects* (for everything else, and includes the set of reals $\mathbb{R}$ as a subsort). A situation represents a world history as a sequence of actions. A set of initial situations correspond to the ways the world might be initially. Successor situations are the result of doing actions, where the term $do(a, s)$ denotes the unique situation obtained on doing a in s. The term $do(\alpha, s)$, where α is the sequence $[a_1, \ldots, a_n]$ abbreviates $do(a_n, do(\ldots, do(a_1, s)\ldots))$. Initial situations are defined as those without a predecessor:

$$Init(s) \doteq \neg\exists a, s'.\ s = do(a, s').$$

We let the constant S_0 denote the actual initial situation, and we use the variable ι to range over initial situations only.

In general, the situations can be structured into a set of trees, where the root of each tree is an initial situation and the edges are actions. In dynamical domains, we want the values of predicate and functions to vary from situation to situation. For this purpose, $\mathcal{L}$ includes *fluents* whose last argument is always a situation. For example, if f is a fluent, we understand $f(s) < 12$ as "$f < 12$ at situation s". Here we assume without loss of generality that all fluents are functional.

Basic action theory

Following [Reiter, 2001], we model dynamic domains in $\mathcal{L}$ by means of a *basic action theory* $\mathcal{D}$, which consists of[3]

1. axioms $\mathcal{D}_0$ that describe what is true in the initial states, including S_0;
2. precondition axioms that describe the conditions under which actions are executable;
3. successor state axioms that describe the changes to fluents on executing actions;

[2]For a demonstration of the formalism in a slightly simpler setting, see [Belle and Levesque, 2013c].

[3]As usual, free variables in any of these axioms should be understood as universally quantified from the outside.

4. domain-independent *foundational* axioms, the details of which need not concern us here. See [Reiter, 2001].

An agent reasons about actions by means of the entailments of $\mathcal{D}$, for which standard Tarskian models suffice. We assume henceforth that models *also* assign the usual interpretations to $=, <, >, 0, 1, +, \times, /, -, e, \pi$ and x^y (exponentials).[4]

Likelihood and degree of belief

The BHL model of belief builds on a treatment of knowledge by Scherl and Levesque [2003]. Here we present a simpler variant based on just two distinguished binary fluents l and p.

The term $l(a, s)$ is intended to denote the likelihood of action a in situation s. For example, suppose $sonar(z)$ is the action of reading the value z from a sensor that measures the distance to the wall, h.[5] We might assume that this action is characterized by a Gaussian error model:[6]

$$\begin{aligned}&l(sonar(z), s) = u \equiv\\&\quad (z \geq 0 \land u = \mathcal{N}(z - h(s); \mu, \sigma^2)) \lor (z < 0 \land u = 0)\end{aligned}$$

which stipulates that the difference between a nonnegative reading of z and the true value h is normally distributed with a variance of σ^2 and mean of μ. In general, the action theory $\mathcal{D}$ is assumed to contain for each action A an additional *action likelihood axiom* of the form

$$l(A(\vec{x}), s) = u \;\equiv\; \phi_A(\vec{x}, u, s)$$

where ϕ_A is a formula that characterizes the conditions under which action A has likelihood u in s. (Actions that have no sensing aspect should be given a likelihood of 1.)

Next, the p fluent determines a probability distribution on situations. The term $p(s', s)$ denotes the relative *weight* accorded to situation s' when the agent happens to be in situation s. The properties of p in initial states, which vary from domain to domain, are specified by axioms as part of $\mathcal{D}_0$. The following nonnegative constraint is also included in $\mathcal{D}_0$:

$$\forall \iota, s.\, p(s, \iota) \geq 0 \;\land\; (p(s, \iota) > 0 \supset Init(s)) \qquad \text{(P1)}$$

While this is a stipulation about initial states ι only, BHL provide a successor state axiom for p, and show that with an appropriate action likelihood axiom, the nonnegative constraint then continues to hold everywhere:

$$\begin{aligned}&p(s', do(a, s)) = u \;\equiv\\&\quad \exists s''\; [s' = do(a, s'') \land Poss(a, s'') \land\\&\qquad\quad u = p(s'', s) \times l(a, s'')]\\&\quad \lor \neg\exists s''\; [s' = do(a, s'') \land Poss(a, s'') \land u = 0]\end{aligned} \qquad \text{(P2)}$$

Now if ϕ is a formula with a single free variable of sort situation,[7] then the *degree of belief* in ϕ is simply defined as the following abbreviation:[8]

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \sum_{\{s' : \phi[s']\}} p(s', s) \qquad \text{(B)}$$

where γ, the normalization factor, is understood throughout as the same expression as the numerator but with ϕ replaced by *true*. For example, here γ is $\sum_{s'} p(s', s)$. We do not have to insist that s' and s share histories since $p(s', s)$ will be 0 otherwise. BHL show how summations can be expressed using second-order logic, see the appendix. That is, neither *Bel*'s definition nor summations are special axioms of $\mathcal{D}$, but simply convenient abbreviations for logical terms. To summarize, in the BHL scheme, an action theory consists of:

1. $\mathcal{D}_0$ as before, but now also including (P1);
2. precondition axioms as before;
3. successor state axioms as before, but now also including one for p *viz.* (P2);
4. foundational domain-independent axioms as before; and
5. action likelihood axioms.

From sums to integrals

While the definition of belief in BHL has many desirable properties, it is defined in terms of a *summation* over situations, and therefore precludes fluents whose values range over the reals. The continuous analogue of (B) then requires *integrating* over some suitable space of values.

As it turns out, a suitable space can be found. First, some notation. We use a form of conditional *if-then-else* expressions, by taking some liberties with notation and the scope of variables as follows. We write $f = \text{If } \exists x.\, \phi \text{ Then } t_1 \text{ Else } t_2$ to mean the logical formula

$$f = u \equiv \exists x.\, [\phi \land (u = t_1)] \lor [(u = t_2) \land \neg\exists x.\, \phi]$$

Now, assume that there are n fluents $f_1, \ldots, f_n$ in $\mathcal{L}$, and that these take no arguments other than a situation.[9] Next, suppose that there is exactly one initial situation for any vector of fluent values [Levesque *et al.*, 1998]:

$$[\forall \vec{x} \exists \iota \bigwedge f_i(\iota) = x_i] \;\land\; [\forall \iota, \iota'. \bigwedge f_i(\iota) = f_i(\iota') \supset \iota = \iota'] \qquad (*)$$

Under these assumptions, it can be shown [Belle and Levesque, 2013a] that the summation over all situations in (B) can be recast as a summation over all possible initial values $x_1, \ldots, x_n$ for the fluents:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \sum_{\vec{x}} P(\vec{x}, \phi, s) \qquad \text{(B}'\text{)}$$

[4]Alternatively, one could specify axioms for characterizing the field of real numbers in $\mathcal{D}$. Whether or not reals with exponentiation is *first-order* axiomatizable remains a major open question.

[5]Naturally, we assume that the value z being read is not under the agent's control. See BHL for a precise rendering of this nondeterminism in terms of GOLOG operators [Reiter, 2001].

[6]Note that $\mathcal{N}$ is a continuous distribution involving π, e, exponentiation, and so on. Therefore, BHL always consider discrete probability distributions that *approximate* the continuous ones.

[7]The ϕ is usually written either with the situation variable suppressed or with a distinguished variable *now*. Either way, $\phi[t]$ is used to denote the formula with that variable replaced by t.

[8]As in probability logics [Fagin and Halpern, 1994], belief in ϕ is simply the total weight of worlds satisfying ϕ.

[9]Basically, if we were to assume that the arguments of all fluents, even k-ary ones, are taken from finite sets then this would allow us to enumerate the n random variables of the domain (for some large n). Note that, from the point of view of situation calculus basic action theories, fluents are typically allowed to take *arguments* from any set, including infinite ones. In probabilistic terms, this would correspond to having a joint probability distribution over infinitely many, perhaps uncountably many, random variables. We know of no existing work of this sort, and we have as yet no good ideas about how to deal with it.

where $P(\vec{t}, \phi, s)$ is the (unnormalized) weight accorded to the *successor* of an initial world where f_i equals t_i:

$$\begin{aligned} & P(\vec{x}, \phi, do(\alpha, S_0)) \doteq \\ & \quad \text{IF } \exists\iota.\ \bigwedge f_i(\iota) = x_i \wedge \phi[do(\alpha, \iota)] \\ & \qquad \text{THEN } p(do(\alpha, \iota), do(\alpha, S_0)) \\ & \quad \text{ELSE } 0 \end{aligned}$$

where α is an action sequence. In a nutshell, because every situation has an initial situation as an ancestor, and because there is a bijection between initial situations and possible fluent values, it is sufficient to sum over fluent values to obtain the belief even for non-initial situations. Note that unlike (B), this one expects the final situation term $do(\alpha, S_0)$ mentioning what actions and observations took place to be explicitly specified, but that is just what one expects when the agent reasons about its belief after doing things, and for the projection problem in particular [Reiter, 2001].

The generalization to the continuous case then proceeds as follows. First, we observe that some (though possibly not all) fluents will be real-valued, and that $p(s', s)$ will now be a measure of *density* not weight. Similarly, the P term above now measures (unnormalized) density rather than weight.

Now suppose fluents are partitioned into two groups: the first k take their values $x_1, \ldots, x_k$ from $\mathbb{R}$, while the rest take their values $y_{k+1}, \ldots, y_n$ from countable domains, then the *degree of belief* in ϕ is an abbreviation for:

$$Bel(\phi, s) \doteq \frac{1}{\gamma} \int_{\vec{x}} \sum_{\vec{y}} P(\vec{x} \cdot \vec{y}, \phi, s)$$

The belief[10] in ϕ is obtained by ranging over all possible fluent values, and integrating and summing the densities of situations where ϕ holds.[11] In [Belle and Levesque, 2013a], it is shown that belief change in this formalism is identical, under certain conditions, to Bayesian conditioning over continuous variables [Pearl, 1988]. This is precisely what is desired to capture standard probabilistic belief update mechanisms.

The appendix shows how integrals can be formulated using second-order quantification. That is, as before, *Bel*, *P*, integrals and sums are simply convenient abbreviations, and do not involve special axioms in $\mathcal{D}$. More precisely, the continuous extension to BHL has the same components from earlier, with a single revision:

1. $\mathcal{D}_0$ additionally includes $(*)$.

Note that likelihood axioms are specified as before, but we will no longer have to approximate Gaussian error models (or any other continuous models) as would BHL.

3 Motion Kinematics and State Estimation

We build a basic action theory $\mathcal{D}$ for a robot on a planar surface. We imagine three fluents h, v and θ in addition to *Poss*, l and p. The fluent h gives the distance to the wall along the horizontal axis and v gives the position of the robot along the vertical axis. The fluent θ gives the orientation of the robot, and is assumed to be between $-180°$ and $180°$. We postulate $\theta = 0$ to mean that the robot is parallel to the X-axis, facing the wall. By extension, when $\theta = 90$ the robot is parallel to the Y-axis, and if the robot is above the X-axis then motion along this orientation would move the robot away from the X-axis. See Figure 2 for the geometric configuration.

We consider two physical actions $mv(z)$ and $orient(z)$, and two sensing actions $sonar(z)$ and $laser(z)$. The mv action moves the robot along its current orientation θ, which can be manipulated in-place by means of *orient*. Both sensors estimate the distance to the wall.

For simplicity, we assume that actions are always executable. Therefore, $\mathcal{D}$ will not contain any precondition axioms. $\mathcal{D}$'s successor state axioms are the following. There is a fixed one for p, which is (P2). For h, we have:

$$\begin{aligned} & h(do(a, s)) = u \equiv \\ & \quad \neg\exists z(a = mv(z)) \wedge u = h(s) \ \vee \\ & \quad \exists z(a = mv(z) \wedge u = \max(0, h(s) - z \cdot \cos(\theta(s)))). \end{aligned} \tag{1}$$

This says an action $mv(z)$ reduces the horizontal distance between the robot and the wall by $z \cdot \cos(\theta(s))$ units, but that the motion stops if the robot hits the wall. It is also assumed that $mv(z)$ is the only action that affects h. Of course, to move away from the wall, z can be any negative value. The $\cos(\theta(s))$ factor is by trigonometry, and comes from the observation that the robot moves z units while being at an angle $\theta(s)$ to the X-axis.

Similarly, for v we have:

$$\begin{aligned} & v(do(a, s)) = u \equiv \\ & \quad \neg\exists z(a = mv(z)) \wedge u = v(s) \vee \\ & \quad \exists z(a = mv(z) \wedge u = v(s) + z \cdot \sin(\theta(s))). \end{aligned} \tag{2}$$

This captures the effect on v after moving, while assuming that $mv(z)$ is the only action affecting v. As for the fluent θ, we naturally assume that *orient* is the only action affecting it, incrementing the current value between $[-180, 180]$:

$$\begin{aligned} & \theta(do(a, s)) = u \equiv \\ & \quad \neg\exists z(a = orient(z)) \wedge u = \theta(s) \vee \\ & \quad \exists z(a = orient(z) \wedge u = [(\theta(s) + z) \bmod 360] - 180). \end{aligned} \tag{3}$$

Finally, we specify the likelihood axioms in $\mathcal{D}$. We will suppose that the laser unit is very accurate:

$$\begin{aligned} & l(laser(z), s) = u \equiv \\ & \quad (z \geq 0 \wedge |\theta(s)| \leq 10 \wedge u = \mathcal{N}(h(s) - z; 0, .25)) \vee \\ & \quad (z < 0 \wedge u = 0) \vee \\ & \quad (|\theta(s)| > 10 \wedge u = 0). \end{aligned} \tag{4}$$

This stipulates that when the robot is *facing* the wall, captured by the θ's absolute value being with a certain range (chosen arbitrarily to be 10 here), the difference between a nonnegative reading of z and the true value h is normally distributed with a variance of .25 and mean of 0. (A mean of 0 indicates that there is no systematic bias in the reading.) If the robot is not in the direction of the wall, due to reflections and similar factors [Thrun *et al.*, 2005] we simply assume that sensors do not return any value.

[10]We write $\vec{x} \cdot \vec{y}$ to mean the concatenation of variables.

[11]We are assuming here that the density function is (Riemann) integrable. If it is not, belief is clearly not defined, nor should it be. Similarly, if the normalization factor is 0, which corresponds to the case of conditioning on an event that has 0 probability, belief should not be (and is not) defined.

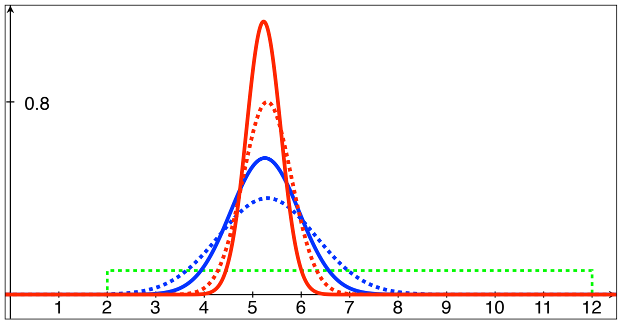


Figure 3: Belief density change for h at S_0 in dotted-green, after a single reading of sonar (item 4) in dotted-blue, after a second sonar reading (item 5) in blue, after a single laser reading (item 6) in dotted-red, and after two laser readings (item 7) in red. Note that a single laser reading is more effective than two sonar readings.

A sonar unit on the robot gives similar readings to the laser unit but is less accurate:

$$\begin{aligned}&l(sonar(z), s) = u \equiv\\&\quad(z \geq 0 \land |\theta(s)| \leq 10 \land u = \mathcal{N}(h(s) - z; 0, 1)) \lor\\&\quad(z < 0 \land u = 0) \lor\\&\quad(|\theta(s)| > 10 \land u = 0).\end{aligned} \tag{5}$$

As mentioned before, physical actions such as $mv(z)$ and $orient(z)$ are assumed to be deterministic for this paper, and so they are given trivial likelihoods:

$$\begin{aligned}&l(mv(z), s) = 1,\\&l(orient(z), s) = 1.\end{aligned}$$

Finally, we turn to the initial theory. $\mathcal{D}_0$ includes the following domain-independent axioms: (∗) and (P1). Specific to the domain, imagine that $\mathcal{D}_0$ says that the robot is parallel to the X-axis in the real world, *i.e.* $\theta(S_0) = 0$. $\mathcal{D}_0$ also includes the following for p:[12]

$$p(\iota, S_0) = \begin{cases} .1 \times \mathcal{N}(v(\iota); 0, 16) & \text{if } 2 \leq h(\iota) \leq 12 \text{ and } \theta(\iota) = 0\\ 0 & \text{otherwise}\end{cases} \tag{6}$$

This says that the value of v is normally distributed about the horizontal axis with variance 16, and independently, $\theta = 0$ and the value of h is uniformly distributed between 2 and 12.[13] This specification assumes that the agent has full belief about $\theta = 0$. (Beliefs when there is uncertainty about θ is illustrated in the next example.) No other sentence is in $\mathcal{D}_0$. This completes the specification of $\mathcal{D}$.

Theorem 1: *The following are logical entailments of $\mathcal{D}$:*

Initial beliefs

1. $Bel(true, S_0) = 1$.

2. $Bel(h = 2 \lor h = 3 \lor h = 4 \lor h = 5, S_0) = 0$
   Intuitively, although we are integrating a density function $q(x_1, x_2, x_3)$ over all real values, $q(x_1, x_2, x_3) = 0$ unless $x_1 \in \{2, 3, 4, 5\}$.

3. $Bel(5 \leq h \leq 5.5, S_0) = .05$
   We are integrating a function that is 0 except when $5 \leq x_1 \leq 5.5$. This is $\int_{\mathbb{R}} \int_5^{5.5} .1 \times \mathcal{N}(x_2; 0, 16)\, dx_1\, dx_2 = .05$.[14]

Sensing by sonar

4. $Bel(5 \leq h \leq 5.5, do(sonar(5.3), S_0)) \approx .19$
   Compared to item 3, belief is sharpened by obtaining a reading of 5.3 on the sonar. This is because the p function incorporates the likelihood of a $sonar(5.3)$ action. Starting with the density function in item 3, the sensor reading multiplies the expression to be integrated by $\mathcal{N}(x_1 - 5.3; 0, 1)$, as given by (5). This amounts to evaluating the expression
   $$\int_{\mathbb{R}} \int_A .1 \times \mathcal{N}(x_1 - 5.3; 0, 1) \times \mathcal{N}(x_2; 0, 16)\, dx_1\, dx_2$$
   with $A = [5, 5.5]$ for the numerator, and $A = [2, 12]$ for the denominator.

5. $Bel(5 \leq h \leq 5.5, do[sonar(5.3), sonar(5.2)], S_0) \approx .27$
   Two successive readings around 5.3 sharpen belief about $[5, 5.5]$ further. Compared to item 4, the density function is additionally multiplied by $\mathcal{N}(x_1 - 5.2; 0, 1)$, and integrated for the denominator and numerator as before.

Sensing by Laser

6. $Bel(5 \leq h \leq 5.5, do(laser(5.3), S_0)) \approx .38$
   Compared to item 3, and even item 4, the highly sensitive laser increases the posterior belief for $[5, 5.5]$ significantly. Using the error model (4), this is a result of
   $$\int_{\mathbb{R}} \int_A .1 \cdot \mathcal{N}(x_2; 0, 16) \cdot \mathcal{N}(x_1 - 5.3; 0, .25)\, dx_2\, dx_1$$
   with A for the numerator and denominator as before.

Competing sensors

7. $Bel(5 \leq h \leq 5.5, do([laser(5.2), laser(5.25)], S_0)) \approx .52$
   $Bel(5 \leq h \leq 5.5, do([laser(5.3), sonar(3)], S_0)) \approx .29$
   The laser is more sensitive than the sonar, and so consecutive readings are far more effective. These changing densities are shown in Figure 3. We also see that after applying the laser sensor, a sonar reading of 3 for h, which is far from the previous observation, only slightly redistributes the density. (Compare this to item 6.)

Physical actions

8. $Bel(h = 0, do([orient(60), mv(10)], S_0)) = .3$
   Here a *continuous* distribution evolves into a *mixed* one. By (1), $h = 0$ holds after the actions iff $h \leq 10 \cdot \cos(60)$ held initially. This results in $\int_{\mathbb{R}} \int_2^5 .1 \times \mathcal{N}(x_2; 0, 16)\, dx_1\, dx_2 = .3$.

[12] Initial beliefs can also be specified for $\mathcal{D}_0$ using Bel directly.

[13] We model a simple distribution for illustrative purposes. In general, neither do the variables have to be independent, nor does the specification need to be complete in the sense of mentioning all the variables.

[14] Strictly speaking, we would need to integrate the density function for $\theta \in [-180, 180]$. Since the density function is 0 when $\theta \neq 0$ according to the p specification (6), we simplify the presentation and show integration wrt the h and v variables only. See the next example for cases where such simplifications are not possible.

9. $Bel(h \leq 5, do([orient(60), mv(10)], S_0)) = .8$
   Bel's definition is amenable to a set of h values, where one value has a weight of .1, and all the other real values have a uniformly distributed density of .1. This change in weights is shown in Figure 4.

10. $Bel(h = 5, do([orient(60), mv(10), mv(-10)], S_0)) = .3$
    $Bel(h = 5, do([orient(60), mv(-10), mv(10)], S_0)) = 0$
    The point $h = 5$ has 0 weight initially (like in item 2). Moving leftwards *first* means many points "collapse", and so this point (now having h value 0) gets .3 weight which is retained on moving away. But not vice versa.

11. $Bel(-1 \leq v \leq 2, do(orient(60), S_0)) =$
    $Bel(-1 \leq v \leq 2, S_0) = \int_{-1}^{2} \mathcal{N}(x_2; 0, 16) \mathrm{d}x_2$
    Owing to Reiter's solution to the frame problem, belief in v is unaffected on changing the orientation. For $v \in [-1, 2]$ it is the area between $[-1, 2]$ bounded by the specified Gaussian.

12. $Bel(v \leq .5, do([orient(30), mv(5)], S_0)) =$
    $Bel(v \leq -2, S_0)$
    $Bel(v \leq -4.5, do([orient(30), mv(-5)], S_0)) =$
    $Bel(v \leq -2, S_0)$
    After the action $mv(5)$ when the robot is oriented at 30, the Gaussian for v's value has its mean "shifted" by $5 \cdot \sin(30)$ because the density associated with $v = x_2$ initially is now associated with $v = x_2 + 2.5$. An analogous shifting occurs when $mv(-5)$ is executed.

Nonstandard properties

13. $Bel(h > 7v, S_0) \approx .6$
    Beliefs about any mathematical expression involving the random variables, even when that does not correspond to well known density functions, are entailed. In this case, we are basically evaluating:
$$\int_{2}^{12} \int_{-\infty}^{x_1/7} .1 \times \mathcal{N}(x_2; 0, 16)\, \mathrm{d}x_2\, \mathrm{d}x_1.$$

14. $Bel([\exists a, s.\ now = do(a, s) \wedge h(s) > 1], do(mv(4), S_0)) = 1.$
    It is possible to refer to earlier or later situations using *now* as the current situation. This says that after moving, there is full belief that $(h > 1)$ held before the action.

Uncertainty about θ

In the previous example, the agent had full belief about θ. The formalism, of course, does not hinge on such assumptions. We now sketch an example in a partial knowledge setting, and discuss a few properties. Features similar to those above can also be observed here.

To build the example, the only sentence we will replace in $\mathcal{D}$ is the p specification. Let $\mathcal{D}^*$ be an action theory exactly as before except for the following new initial axiom for p:
$$p(\iota, S_0) = \begin{cases} .1 \times \mathcal{N}(v(\iota); 0, 16) \times .5 & \ldots \\ 0 & \text{otherwise} \end{cases}$$
where the ellipses stands for:
$$\text{if } h(\iota) \in [2, 12] \text{ and } \theta(\iota) \in [-1, 1].$$

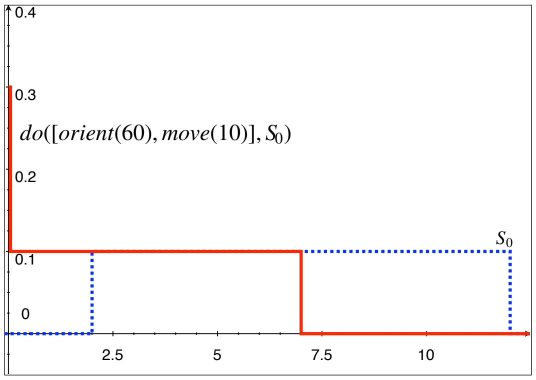


Figure 4: Belief update for h after physical actions. Initial belief at S_0 in blue, and after a move by 10 with an orientation of 60 (in red).

What this says is that the value of v is normally distributed, independently h is uniformly distributed between 2 and 12, and independently θ is uniformly distributed between -1 and 1. (As we pointed out before, variants where the independence assumption does not hold are also easy to sketch, but will not be illustrated here.)

Theorem 2: *The following are entailments of $\mathcal{D}^*$:*

1. $Bel(\theta \leq 0, S_0) = .5$
   This is the integral $\int_{x_1} \int_{x_2} \int_{-180}^{0} q(x_1, x_2, x_3) \mathrm{d}x_3 \mathrm{d}x_2 \mathrm{d}x_1$, with q being the density function at the initial situation. Since $q(x_1, x_2, x_3)$ is 0 except when $x_3 \in [-1, 1]$ in which case it is $.1 \times \mathcal{N}(x_2; 0, 16) \times .5$, the expression evaluates to .5.

2. $Bel(5 \leq h \leq 5.5, do(sonar(5.3), S_0)) \approx .19$
   According to (5), doing the sonar action means that the density function $q(x_1, x_2, x_3)$ is multiplied by $\mathcal{N}(x_1 - 5.3; 0, 1)$ when θ is between -10 and 10, but 0 otherwise. Since $q(x_1, x_2, x_3)$ is 0 for $\theta \notin [-1, 1]$, we are left with the expression:
$$\int_{\mathbb{R}} \int_{A} \int_{-1}^{1} q(x_1, x_2, x_3) \times \mathcal{N}(x_1 - 5.3; 0, 1) \mathrm{d}x_3 \mathrm{d}x_1 \mathrm{d}x_2$$
   for $A = [5, 5.5]$ for the numerator, and for $A = [2, 12]$ for the denominator. The end result is precisely the same belief as in item 4 from the previous example.

3. $Bel(h \leq 5, do(mv(3), S_0)) \approx .55$
   We first observe that if the robot is oriented at angle θ, then by means of (1), the belief in $h \leq 5$ on moving 3 units can be obtained from the initial belief in $h \leq 5 + 3 \cdot \cos\theta$. Therefore, we are to evaluate the integral
$$\int_{\mathbb{R}} \int_{-1}^{1} \int_{2}^{5+3 \cdot \cos x_3} q(x_1, x_2, x_3) \mathrm{d}x_1 \mathrm{d}x_3 \mathrm{d}x_2 =$$
$$\int_{\mathbb{R}} \int_{-1}^{1} \mathcal{N}(x_2; 0, 16) \cdot .1 \cdot .5 \cdot [(5 + 3\cos x_3) - 2] \mathrm{d}x_3 \mathrm{d}x_1.$$
   In contrast, it is easy to verify that the original basic action theory $\mathcal{D}$ entails $Bel(h \leq 5, do(mv(3), S_0)) = .6$. That is, as one would expect, due to the robot's uncertainty regarding its orientation in $\mathcal{D}^*$, it has weaker beliefs about $h \leq 5$.

4 Related Work

Sensor fusion has been a primary concern in state estimation approaches [Thrun *et al.*, 2005]. Popular models include variants of Kalman filtering [Fox *et al.*, 2003], where priors and likelihoods are assumed to be Gaussian. Our formalism, however, does not require any constraints on the nature of distributions, nor about how distributions and dependencies may evolve after actions. Moreover, strict uncertainty is allowed. This distinguishes the current method from numerous probabilistic formalisms [Dean and Wellman, 1991; Fox *et al.*, 2003], including those that handle actions explicitly [Darwiche and Goldszmidt, 1994; Hajishirzi and Amir, 2010]. To the best of our knowledge, none of these handle changes in state variables like those considered here.

Logical formalisms, such as [Halpern, 1990; Bacchus, 1990], are equipped to handle features such as disjunctions and quantifiers, but they do not explicitly address actions. Relational probabilistic languages and Markov logics [Ng and Subrahmanian, 1992; Richardson and Domingos, 2006] also do not model actions. Recent temporal extensions, such as [Choi *et al.*, 2011], specifically treat Kalman filtering, and not complex actions. In this regard, action logics such as dynamic and process logics are closely related. Recent proposals, for example [Van Benthem *et al.*, 2009], treat sensor fusion. However, these and related frameworks [Halpern and Tuttle, 1993; Kushmerick *et al.*, 1995], are mostly propositional. In the last years, there have been extensions to the PDDL planning language, so as to account for probabilistic effects and partial observability [Younes and Littman, 2004; Sanner, 2011], but limited to certain sorts of initial databases rather than a specification that allows full first-order expressivity.

Finally, proposals based on the situation and fluent calculi are first-order [Bacchus *et al.*, 1999; Poole, 1998; Boutilier *et al.*, 2000; Mateus *et al.*, 2001; Shapiro, 2005; Gabaldon and Lakemeyer, 2007; Fritz and McIlraith, 2009; Belle and Lakemeyer, 2011; Thielscher, 2001], but none of them deal with continuous sensor noise, and nor do the extensions for continuous processes [Reiter, 2001; Herrmann and Thielscher, 1996; Fox and Long, 2006]. We are also not aware of any logical formalism that deals with the integration of continuous variables within the language.

5 Conclusions

This paper investigates a kinematic setup and probabilistic state estimation for a robot operating in an incompletely known world, equipped with noisy sensors. In contrast to a number of competing formalisms, where the modeler is left with the difficult task of deciding how the dependencies and distributions of state variables might evolve, here one need only specify the initial beliefs and the physical laws. Suitable posteriors are then entailed. The framework of the situation calculus, and a recent generalization to the BHL scheme, allows us to additionally specify situation-specific biases and realistic continuous error models. Our example demonstrates that belief changes appropriately even when one is interested in nonstandard properties, such as logical relationships of state variables, all of which emerges as a side-effect of the general specification.

For the future, on the representation side, features such as continuous time, exogenous actions, decision theory and durative actions have been proposed in the situation calculus [Reiter, 2001], which could be imported to our formalism. On the more computational side, we are interested in studying formal constraints on action theories that would allow us to estimate posteriors efficiently under the assumption that priors and likelihoods are drawn from tractable distributions [Box and Tiao, 1973]. This would perhaps need, among others, an account of regression [Reiter, 2001; Scherl and Levesque, 2003], generalized for probabilistic beliefs and noisy sensing. Preliminary results in this direction appear in [Belle and Levesque, 2013b].

Appendix: Sums and Integrals in Logic

Logical formulas can be used to characterize sums and a variety of sorts of integrals. Here we show the simplest possible cases: the summing of a one variable function from 1 to n, and the definite integral from $-\infty$ to ∞ of a continuous real-valued function of one variable. Other complications are treated in a longer version of the paper.

First, sums. For any logical term t and variable i, we introduce the following notation to characterize summations:

$$\sum_{i=1}^{n} t = z \doteq \exists f\ [f(1) = t^i_1 \land f(n) = z \land \\ \forall j\ (1 \leq j < n \supset f(j+1) = f(j) + t^i_{(j+1)}\ )]$$

where f is assumed to not appear in t, and j is understood to be chosen not to conflict with any of the variables in t and i.

Now, integrals. We begin by introducing a notation for limits to positive infinity. For any logical term t and variable x, we let $\lim_{x\to\infty} t$ stand for a term characterized by:

$$\lim_{x\to\infty} t = z \ \doteq\ \forall u(u > 0 \supset \exists m \forall n(n > m \supset \left|z - t^x_n\right| < u)).$$

The variables u, m, and n are understood to be chosen here not to conflict with any of the variables in x, t, and z.

Then, for any variable x and terms a, b, and t, we introduce a term INT$[x, a, b, t]$ to denote the definite integral of t over x from a to b:

$$\mathrm{INT}[x, a, b, t] \ \doteq\ \lim_{n\to\infty} h \cdot \sum_{i=1}^{n} t^x_{(a+h\cdot i)}$$

where h stands for $(b-a)/n$. The variable n is chosen not to conflict with any of the other variables. Finally, we define the definite integral of t over all real values of x by the following:

$$\int_x t \ \doteq\ \lim_{u\to\infty} \lim_{v\to\infty} \mathrm{INT}[x, -u, v, t].$$

The main result for this logical abbreviation is the following:

Theorem 3: *Let g be a function symbol of $\mathcal{L}$ standing for a function from $\mathbb{R}$ to $\mathbb{R}$, and let c be a constant symbol of $\mathcal{L}$. Let M be any logical interpretation of $\mathcal{L}$ such that the function g^M is continuous everywhere. Then we have the following:*

$$\textit{If}\ \int_{-\infty}^{\infty} g^M(x)\,.\,dx \ =\ c^M \quad \textit{then}\ \ M \models (c = \int_x g(x)).$$

References

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

[Bacchus, 1990] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge*. MIT Press, 1990.

[Belle and Lakemeyer, 2011] V. Belle and G. Lakemeyer. A semantical account of progression in the presence of uncertainty. In *Proc. AAAI*, pages 165–170, 2011.

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

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

[Belle and Levesque, 2013c] V. Belle and H. J. Levesque. Robot location estimation in the situation calculus. In *Symposium on Logical Formalizations of Commonsense Reasoning*, 2013.

[Boutilier *et al.*, 2000] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI*, pages 355–362, 2000.

[Box and Tiao, 1973] G. E. P. Box and G. C. Tiao. *Bayesian inference in statistical analysis*. Addison-Wesley, 1973.

[Choi *et al.*, 2011] J. Choi, A. Guzman-Rivera, and E. Amir. Lifted relational kalman filtering. In *Proc. IJCAI*, pages 2092–2099, 2011.

[Darwiche and Goldszmidt, 1994] A. Darwiche and M. Goldszmidt. Action networks: A framework for reasoning about actions and change under uncertainty. In *Proc. UAI*, pages 136–144, 1994.

[Dean and Kanazawa, 1989] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational intelligence*, 5(2):142–150, 1989.

[Dean and Wellman, 1991] T. Dean and M. Wellman. *Planning and control*. Morgan Kaufmann Publishers Inc., 1991.

[Fagin and Halpern, 1994] R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *J. ACM*, 41(2):340–367, 1994.

[Fagin *et al.*, 1995] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[Fox and Long, 2006] M. Fox and D. Long. Modelling mixed discrete-continuous domains for planning. *J. Artif. Intell. Res. (JAIR)*, 27:235–297, 2006.

[Fox *et al.*, 2003] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *Pervasive Computing, IEEE*, 2(3):24–33, 2003.

[Fritz and McIlraith, 2007] C. Fritz and S. A. McIlraith. Monitoring plan optimality during execution. In *Proc. ICAPS*, pages 144–151, 2007.

[Fritz and McIlraith, 2009] C. Fritz and S. A. McIlraith. Computing robust plans in continuous domains. In *Proc. ICAPS*, pages 346–349, 2009.

[Gabaldon and Lakemeyer, 2007] A. Gabaldon and G. Lakemeyer. ESP: A logic of only-knowing, noisy sensing and acting. In *Proc. AAAI*, pages 974–979, 2007.

[Hajishirzi and Amir, 2010] H. Hajishirzi and E. Amir. Reasoning about deterministic actions with probabilistic prior and application to stochastic filtering. In *Proc. KR*, 2010.

[Halpern and Tuttle, 1993] J. Y. Halpern and M. R. Tuttle. Knowledge, probability, and adversaries. *J. ACM*, 40:917–960, 1993.

[Halpern, 1990] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

[Herrmann and Thielscher, 1996] C. Herrmann and M. Thielscher. Reasoning about continuous processes. In *AAAI/IAAI, Vol. 1*, pages 639–644, 1996.

[Hughes and Cresswell, 1972] G. E. Hughes and M. J. Cresswell. *An introduction to modal logic*. Methuen London, 1972.

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

[Levesque *et al.*, 1998] H. J. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Electron. Trans. Artif. Intell.*, 2:159–178, 1998.

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

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502, 1969.

[Ng and Subrahmanian, 1992] R. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

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

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

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

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

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

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

[Shapiro, 2005] S. Shapiro. Belief change with noisy sensing and introspection. In *NRAC Workshop*, pages 84–89, 2005.

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

[Thrun *et al.*, 2005] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

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

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

On Decidable Verification of Non-terminating Golog Programs

Jens Claßen, Martin Liebenberg and **Gerhard Lakemeyer**
Dept. of Computer Science
RWTH Aachen University
52056 Aachen
Germany
{classen,liebenberg,gerhard}@kbsg.rwth-aachen.de

Abstract

The high-level action programming language GOLOG has proven to be a useful means for the control of autonomous agents such as mobile robots. Usually, such agents perform open-ended tasks, and their control programs are hence non-terminating. Before deploying such a program to the robot, it is often desirable if not crucial to verify that it meets certain requirements, preferably by means of an automated method. For this purpose, Claßen and Lakemeyer recently introduced algorithms for the verification of temporal properties of non-terminating GOLOG programs, based on the first-order modal Situation Calculus variant $\mathcal{ES}$, and regression-based reasoning. However, while GOLOG's high expressiveness is a desirable feature, it also means that their verification procedures cannot be guaranteed to terminate in general. In this paper, we address this problem by showing that, for a relevant subset, the verification of non-terminating GOLOG programs is indeed decidable, which is achieved by means of three restrictions. First, we use the $\mathcal{ES}$ variant of a decidable two-variable fragment of the Situation Calculus that was introduced by Gu and Soutchanski. Second, we have to restrict the GOLOG program to contain ground action only. Finally, we consider special classes of successor state axioms, namely the context-free ones and those that only admit local effects.

1 Introduction

The GOLOG [De Giacomo *et al.*, 2000; Levesque *et al.*, 1997] family of high-level action programming languages and its underlying logic, the Situation Calculus [McCarthy and Hayes, 1969; Reiter, 2001], have proven to be useful means for the control of autonomous agents such as mobile robots [Burgard *et al.*, 1999]. Usually, the task of such an agent is open-ended, i.e. there is no predefined goal or terminal state that the agent tries to reach, but (at least ideally) the robot works indefinitely, and its corresponding control program is hence *non-terminating*.

As a simple example, consider a mobile robot whose task it is to remove dirty dishes from certain locations in an office on request. A program for this robot might look like this:

```
loop :  while ($\exists x.OnRobot(x)$) do
            $\pi x.unload(x)$
        endWhile;
        $\pi y.goToRoom(y)$;
        while ($\exists x.DirtyDish(x,y)$) do
            $\pi x.load(x,y)$
        endWhile;
        $goToKitchen$
```

We assume that the robot is initially in the kitchen, its home base. There is an infinite loop, where during each iteration the robot first unloads all dishes it carries, then selects a room in the office building, goes to this room, loads all dirty dishes in this room, and returns to the kitchen. Here, $DirtyDish(x,y)$ should be read as "dirty dish x is in room y" and $load(x,y)$ as "load dish x in room y." During the execution of the program, people can send requests indicating that there is a dirty dish in a certain room (not shown here).

Before actually deploying such a program on the robot and executing it in the real world, it is often desirable if not crucial to verify that it meets certain requirements such as safety, liveness and fairness properties, for example that "every request will eventually be served by the robot" or whether "it is possible that no request is ever served." Moreover, the verification is preferably done using an *automated* method, since manual, meta-theoretic proofs such as done in [De Giacomo *et al.*, 1997] tend to be tedious and prone to errors. For this purpose, Claßen and Lakemeyer [2008] recently proposed the logic $\mathcal{ESG}$, an extension of the modal Situation Calculus variant $\mathcal{ES}$ [Lakemeyer and Levesque, 2010] by constructs that allow to express temporal properties of GOLOG programs. They moreover provided algorithms for the verification of a subset of the logic that resembles the branching-time temporal logic CTL. Their methods rely on regression-based reasoning and a newly introduced graph representation of GOLOG programs to do a systematic exploration of a program's configuration space within a fixpoint approximation loop. While the procedures are proven to be sound, no general guarantee can be given for termination.

There are two reasons for this. On the one hand, to detect the convergence of the fixpoint loop, the algorithm has to check the equivalence of formulas that encode reachable program configurations. Since these may be arbitrary first-order formulas, this already amounts to an undecidable problem. Furthermore, even if all equivalence checks can be performed in finite time (or if we assume a first-order oracle), the fixpoint computation may never converge.

A straight-forward approach to remedy this problem is to restrict the input language such that verification becomes decidable, as done for instance by Baader, Liu and ul Mehdi [2010]. Instead of using the full first-order expressiveness of the Situation Calculus or $\mathcal{ES}$, they resort to a dynamic extension [Baader *et al.*, 2005] of the decidable description logic $\mathcal{ALC}$ [Baader *et al.*, 2003] to represent pre- and postconditions of actions, where properties are expressed by a variant of LTL over $\mathcal{ALC}$ assertions [Baader *et al.*, 2008]. Second, they encode programs by finite Büchi automata instead of the fully-fledged GOLOG language. They could show that under these restrictions, verification reduces to a decidable reasoning task within the underlying description logic.

Although this is a step in the right direction, it requires harsh restrictions in terms of expressiveness. In particular, representing programs through Büchi automata loses one important feature of GOLOG, namely the possibility to include test conditions in the form of formulas. Moreover, representing action effects within $\mathcal{ALC}$ only allows for basic STRIPS-style addition and deletion of literals. While decidability can obviously not be achieved without any restrictions on the input languages, the high, first-order expressiveness of the Situation Calculus and GOLOG is typically considered a desirable feature and the reason why these languages were chosen in the first place, and one would rather give up as little as possible of it. Ideally, we could do the verification within the very same expressive formalism and with the same reasoning tools that are used for the actual control of the agent.

In this paper, we show that this is indeed possible for a relevant subset of the formalism. In order to achieve decidability for first-order equivalence checks, we rely on results by Gu and Soutchanski [Gu and Soutchanski, 2010] who presented a modified version of the Situation Calculus built using a two-variable fragment of first-order logic and a variant of Reiter's regression operator such that the reasoning task of projection becomes decidable. Since (as we will see later) this is in itself not sufficient to guarantee the termination of the overall verification method, we moreover consider special classes of successor state axioms from the literature to be used in the agent's basic action theory, namely the *context-free* [Lin and Reiter, 1997] ones as well as those that only admit *local effects* [Liu and Levesque, 2005], and prove that under these prerequisites, a termination guarantee can be given for the verification methods if we restrict the GOLOG program to contain ground actions only. Note that our restrictions allow us to retain a great deal of (first-order) expressiveness, including test conditions in programs and conditional action effects.

The remainder of this paper is organized as follows. In the following section, we briefly recapitulate the logic $\mathcal{ESG}$. Section 3 then presents the verification procedures we consider. In Section 4, we present a decidable subset of $\mathcal{ES}$ that is similar to Gu and Soutchanski's two-variable Situation Calculus fragment. Sections 6 and 5 contain the main results of this paper, namely the decidability of the verification methods for the above mentioned classes of basic action theories. Section 7 reviews related work before we conclude in Section 8.

2 The Logic $\mathcal{ESG}$

2.1 Syntax

The language is a first-order modal dialect with equality and sorts of type *object* and *action*. It includes countably infinitely many standard names for each sort. Also included are both fluent and rigid predicate and function symbols. Fluents vary as the result of actions, but rigids do not. We assume that the fluents include unary predicates $Poss$ and Exo, whose argument is of type action and which will be used to specify when an action is executable or exogenous, respectively.

The logical connectives are $\wedge$, $\neg$, $\forall$, together with these modal operators: $\boldsymbol{X}$, $\boldsymbol{U}$, $[\delta]$, and $[\![\delta]\!]$, where δ is a program as defined below. Other connectives like $\vee$, $\supset$, $\subset$, $\equiv$, and $\exists$ are used as the usual abbreviations.

Program constructs are logical (built-in) symbols with a fixed meaning. The programs we consider are the ones admitted by the following grammar:

$$\delta ::= \; t \mid \alpha? \mid \delta_1;\delta_2 \mid \delta_1|\delta_2 \mid \pi x.\delta \mid \delta_1\|\delta_2 \mid \delta^* \quad (1)$$

That is we allow primitive actions t (where t can be any action term), tests $\alpha?$ (where α is a static situation formula as defined below), sequence, nondeterministic branching, nondeterministic choice of argument, concurrency, and nondeterministic iteration. Moreover, conditionals and loops can be defined in terms of the above constructs:

$$\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \overset{def}{=} [\phi?;\delta_1] \mid [\neg\phi?;\delta_2] \quad (2)$$

$$\textbf{while } \phi \textbf{ do } \delta \textbf{ endWhile} \overset{def}{=} [\phi?;\delta]^*;\neg\phi? \quad (3)$$

The infinite loop, also abbreviated as δ^ω, is further given by:

$$\textbf{loop } \delta \textbf{ endLoop} \overset{def}{=} \textbf{while } \top \textbf{ do } \delta \textbf{ endWhile} \quad (4)$$

Formulas come in two different "flavours", as given by the following definitions:

Definition 1 (Situation Formulas)**.** The *situation formulas* are the least set such that

- if $t_1,\dots,t_k$ are terms and P is a (fluent or rigid) k-ary predicate symbol, then $P(t_1,\dots,t_k)$ is a situation formula;
- if t_1 and t_2 are terms, then $(t_1 = t_2)$ is a situation formula;
- if α and β are situation formulas, x is a variable, P is a (fluent or rigid) predicate symbol, δ is a program, and ϕ is a trace formula (defined below), then $\alpha\wedge\beta$, $\neg\alpha$, $\forall x.\alpha$, $\forall P.\alpha$, $\Box\alpha$, $[\delta]\alpha$ ("α holds after executing δ"), and $[\![\delta]\!]\phi$ ("temporal property ϕ holds for all executions of δ") are situation formulas.

Situation formulas, roughly, express properties wrt a given situation and possibly future situations, that is, the formulas may include references to future situations by means of $[\cdot]$, $\Box$, or $[\![\cdot]\!]$. Moreover, let $\langle\delta\rangle\alpha = \neg[\delta]\neg\alpha$ and $\langle\!\langle\delta\rangle\!\rangle\varphi =$

$\neg[\![\delta]\!]\neg\varphi$. A situation formula α is called *fluent* when it contains no $[\cdot]$, no $\Box$, and no $[\![\cdot]\!]$ operators, nor any of the special fluents $Poss$ and Exo. It is called *static* when it contains no $[\cdot]$, no $\Box$ and no $[\![\cdot]\!]$ operators. It is *bounded* when it contains no $\Box$ operators, no $[\![\cdot]\!]$ operators, and $[t]$ operators only in case the argument is an action term t.

Definition 2 (Trace Formulas)**.** The *trace formulas* are the least set such that

- if α is a situation formula, then it is also a trace formula;
- if ϕ and ψ are trace formulas and x is a variable, then $\phi \wedge \psi$, $\neg\phi$, $\forall x.\phi$, $\boldsymbol{X}\phi$ ("ϕ holds in the next situation"), and $\phi\,\boldsymbol{U}\,\psi$ ("ϕ holds until ψ holds") are also trace formulas.

Trace formulas, as the name suggests, are used to talk about *traces* of situations, i.e. finite or infinite sequences of actions. We will use them for representing the temporal properties of program execution traces. In addition to the usual abbreviations, we also have $\boldsymbol{F}\phi = (\top\,\boldsymbol{U}\,\phi)$ ("eventually ϕ") and $\boldsymbol{G}\phi = \neg\boldsymbol{F}\neg\phi$ ("always ϕ").

2.2 Semantics

Terms and formulas are interpreted with respect to *worlds*:

Definition 3 (Worlds)**.** Let $\mathcal{P}_O$ and $\mathcal{P}_A$ denote the set of primitive terms of sort object, and action, respectively, where a *primitive term* is of the form $f(n_1,\ldots,n_k)$, where all the n_i are standard names. Similarly, let $\mathcal{P}_F$ be the set of all primitive formulas $F(n_1,\ldots,n_k)$. Moreover, let $\mathcal{N}_O$ and $\mathcal{N}_A$ be the sets of all standard names of sort object and action, respectively, $\mathcal{N} = \mathcal{N}_O \cup \mathcal{N}_A$, and $\mathcal{Z} = {\mathcal{N}_A}^*$ the set of all finite sequences of action names. A world w then is a mapping

- $w : \mathcal{P}_O \times \mathcal{Z} \to \mathcal{N}_O$ and
- $w : \mathcal{P}_A \times \mathcal{Z} \to \mathcal{N}_A$ and
- $w : \mathcal{P}_F \times \mathcal{Z} \to \{0,1\}$

satisfying the following constraints:

Rigidity: If R is a rigid function or predicate symbol, then for all $z, z' \in \mathcal{Z}$, $w[R(n_1,\ldots,n_k),z] = w[R(n_1,\ldots,n_k),z']$.

Unique names for actions: If $g(\vec{n})$ and $g'(\vec{n}')$ are two distinct primitive action terms, then for all $z \in \mathcal{Z}$, $w[g(\vec{n}),z] \neq w[g'(\vec{n}'),z]$.

Let $\mathcal{W}$ denote the set of all worlds.

A world thus maps primitive terms to co-referring standard names of the corresponding sort, and primitive formulas to truth values. The rigidity constraint ensures that rigid symbols do not take different values in different situations, as expected. We further incorporate the unique names assumption for actions into our logic's semantics, as opposed to the Situation Calculus where this is typically asserted axiomatically.

Definition 4 (Denotation of Terms)**.** Given a ground term t, a world w, and an action sequence $z \in \mathcal{Z}$, we define $|t|_w^z$ (read: "the co-referring standard name for t given w and z") by:

1. If $t \in \mathcal{N}$, then $|t|_w^z = t$;
2. if $t = f(t_1,\ldots,t_k)$, then $|t|_w^z = w[f(n_1,\ldots,n_k),z]$, where $n_i = |t_i|_w^z$.

To interpret programs, we need the notion of *program configurations*. A configuration $\langle z,\delta\rangle$ consists of an action sequence z and a program δ, where intuitively z is the history of actions that have already been performed, while δ is the program that remains to be executed. Then we define the possible transitions and finality of programs as follows:

Definition 5 (Program Transition Semantics)**.** The transition relation $\xrightarrow{w}$ among configurations, given a world w, is the least set satisfying

1. $\langle z,t\rangle \xrightarrow{w} \langle z\cdot p,\top?\rangle$, if $p = |t|_w^z$;
2. $\langle z,\delta_1;\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\gamma;\delta_2\rangle$, if $\langle z,\delta_1\rangle \xrightarrow{w} \langle z\cdot p,\gamma\rangle$;
3. $\langle z,\delta_1;\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$,
   if $\langle z,\delta_1\rangle \in \mathcal{F}^w$ and $\langle z,\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$;
4. $\langle z,\delta_1|\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$,
   if $\langle z,\delta_1\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$ or $\langle z,\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$;
5. $\langle z,\pi x.\delta\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$,
   if $\langle z,\delta_n^x\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$ for some $n \in \mathcal{N}_x$;
6. $\langle z,\delta^*\rangle \xrightarrow{w} \langle z\cdot p,\gamma;\delta^*\rangle$, if $\langle z,\delta\rangle \xrightarrow{w} \langle z\cdot p,\gamma\rangle$;
7. $\langle z,\delta_1\|\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\|\delta_2\rangle$, if $\langle z,\delta_1\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$;
8. $\langle z,\delta_1\|\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta_1\|\delta'\rangle$, if $\langle z,\delta_2\rangle \xrightarrow{w} \langle z\cdot p,\delta'\rangle$.

Above, $\mathcal{N}_x$ means the set of all standard names of the same sort as x, and δ_n^x refers to δ with x replaced by n.

The set of final configurations $\mathcal{F}^w$ of a world w is the smallest set such that

1. $\langle z,\alpha?\rangle \in \mathcal{F}^w$ if $w,z \models \alpha$;
2. $\langle z,\delta_1;\delta_2\rangle \in \mathcal{F}^w$ if $\langle z,\delta_1\rangle \in \mathcal{F}^w$ and $\langle z,\delta_2\rangle \in \mathcal{F}^w$;
3. $\langle z,\delta_1|\delta_2\rangle \in \mathcal{F}^w$ if $\langle z,\delta_1\rangle \in \mathcal{F}^w$ or $\langle z,\delta_2\rangle \in \mathcal{F}^w$;
4. $\langle z,\pi x.\delta\rangle \in \mathcal{F}^w$ if $\langle z,\delta_n^x\rangle \in \mathcal{F}^w$ for some $n \in \mathcal{N}_x$;
5. $\langle z,\delta^*\rangle \in \mathcal{F}^w$;
6. $\langle z,\delta_1\|\delta_2\rangle \in \mathcal{F}^w$ if $\langle z,\delta_1\rangle \in \mathcal{F}^w$ and $\langle z,\delta_2\rangle \in \mathcal{F}^w$.

Temporal properties that we express by situation formulas refer to *traces*, as defined below.

Definition 6 (Traces)**.** A *trace* is a possibly infinite sequence of *action* standard names. As a notational convention, we use τ to denote arbitrary traces, z for finite ones and π for infinite ones. Let $\Pi = {\mathcal{N}_A}^\omega$ be the set of all infinite traces, and $\mathcal{T} = \mathcal{Z} \cup \Pi$ the set of all traces. Furthermore, let $\pi^{(i)}$ stand for the finite sequence that consists of the first i elements of π, where $\pi^{(0)}$ is the empty sequence $\langle\rangle$.

We can now define the traces admitted by a given program:

Definition 7 (Traces of Programs)**.** Let $\xrightarrow{w}{}^*$ denote the reflexive and transitive closure of $\xrightarrow{w}$. Given a world w and a finite sequence of *action* standard names z, the set of *traces* $\|\delta\|_w^z$ of a program δ is the set

$$\{z' \in \mathcal{Z} \mid \langle z,\delta\rangle \xrightarrow{w}{}^* \langle z\cdot z',\delta'\rangle,\ \langle z\cdot z',\delta'\rangle \in \mathcal{F}^w\} \ \cup$$
$$\{\pi \in \Pi \mid \langle z,\delta\rangle \xrightarrow{w} \langle z\cdot\pi^{(1)},\delta_1\rangle \xrightarrow{w} \langle z\cdot\pi^{(2)},\delta_2\rangle \xrightarrow{w} \cdots$$
$$\text{where for all } i \geq 0,\ \langle z\cdot\pi^{(i)},\delta_i\rangle \notin \mathcal{F}^w\}$$

In words, the finite traces admitted by some δ given w and z are those that correspond to a finite number of transitions by means of which a final configuration is reachable. Its infinite traces are given by all infinite sequences of transitions that never visit any final configuration.

Situation and Trace Formulas

We are now equipped to define the truth of formulas:

Definition 8 (Truth of Situation and Trace Formulas)**.** Given a world $w \in \mathcal{W}$ and a situation formula α, we define $w \models \alpha$ as $w, \langle\rangle \models \alpha$, where for any $z \in \mathcal{Z}$:

1. $w, z \models F(t_1, \ldots, t_k)$ iff $w[F(n_1, \ldots, n_k), z] = 1$, where $n_i = |t_i|_w^z$;
2. $w, z \models (t_1 = t_2)$ iff n_1 and n_2 are identical, where $n_i = |t_i|_w^z$;
3. $w, z \models \alpha \wedge \beta$ iff $w, z \models \alpha$ and $w, z \models \beta$;
4. $w, z \models \neg\alpha$ iff $w, z \not\models \alpha$;
5. $w, z \models \forall x.\alpha$ iff $w, z \models \alpha_n^x$ for all $n \in \mathcal{N}_x$;
6. $w, z \models \Box\alpha$ iff $w, z \cdot z' \models \alpha$ for all $z' \in \mathcal{Z}$;
7. $w, z \models [\delta]\alpha$ iff for all finite $z' \in \|\delta\|_w^z$, $w, z \cdot z' \models \alpha$;
8. $w, z \models [\![\delta]\!]\phi$ iff for all $\tau \in \|\delta\|_w^z$, $w, z, \tau \models \phi$.

The truth of trace formulas ϕ is defined as follows for $w \in \mathcal{W}$, $z \in \mathcal{Z}$, and traces $\tau \in \mathcal{T}$:

1. $w, z, \tau \models \alpha$ iff $w, z \models \alpha$, if α is a situation formula;
2. $w, z, \tau \models \phi \wedge \psi$ iff $w, z, \tau \models \phi$ and $w, z, \tau \models \psi$;
3. $w, z, \tau \models \neg\phi$ iff $w, z, \tau \not\models \phi$;
4. $w, z, \tau \models \forall x.\phi$ iff $w, z, \tau \models \phi_n^x$ for all $n \in \mathcal{N}_x$;
5. $w, z, \tau \models \boldsymbol{X}\phi$ iff $\tau = p \cdot \tau'$ and $w, z \cdot p, \tau' \models \phi$;
6. $w, z, \tau \models \phi\ \boldsymbol{U}\ \psi$ iff there is z' such that $\tau = z' \cdot \tau'$ and $w, z \cdot z', \tau' \models \psi$ and for all $z'' \neq z'$ with $z' = z'' \cdot z'''$, $w, z \cdot z'', z''' \cdot \tau' \models \phi$.

2.3 Basic Action Theories and Regression

Definition 9. A *basic action theory* (BAT) $\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \cup \Sigma_{\text{exo}}$ describes the dynamics of a specific application domain, where

1. Σ_0, *the initial database*, is a finite set of fluent sentences describing the initial state of the world.
2. Σ_{pre} is a *precondition axiom* of the form $\Box Poss(a) \equiv \pi$, with π being a fluent formula, whose only free variable is a, describing precisely the conditions under which a is a possible action.
3. Σ_{post} is a finite set of *successor state axioms* (SSAs), one for each fluent relevant to the application domain, incorporating Reiter's [Reiter, 2001] solution to the frame problem, and encoding the effects the actions have on the different fluents. The SSA for a fluent predicate has the form $\Box[a]F(\vec{x}) \equiv \gamma_F^+ \vee F(\vec{x}) \wedge \neg\gamma_F^-$, whereas the one for a functional fluent is of the form $\Box[a]f(\vec{x}) = y \equiv \gamma_f^+ \vee (f(\vec{x}) = y) \wedge \neg\exists y'\gamma_f^+{}_{y'}^{y}$, where γ_F^+ and γ_F^- are fluent formulas with free variables $\vec{x}$, and γ_f^+ one with free variables among $\vec{x}$ and y.
4. Σ_{exo} is the *exogenous actions axiom*, having the form $\Box Exo(a) \equiv \chi$, where χ is again a fluent formula with the free variable a. It is used to express the necessary and sufficient conditions under which an action is exogenous, i.e. not controlled by the agent, but by "nature".

Our algorithm relies on the equivalent of Reiter's *regression* operator $\mathcal{R}[\alpha]$. Roughly, the idea is that, whenever we encounter a subformula of the form $[t]F(\vec{x})$ within α, where t is an action term, we may substitute it by the right-hand side of the successor state axiom of the fluent F. This is sound in the sense that the axiom defines the two expressions to be equivalent. The result of the substitution will be true in exactly the same worlds satisfying the action theory Σ as the original one, but contains one less modal operator $[t]$. Similarly, $Poss(t)$ and $Exo(t)$ are replaced by the right-hand sides of the corresponding axiom. By iteratively applying such substitutions, we eventually get a fluent formula that describes exactly the conditions on the initial situation under which the original, non-static formula holds:

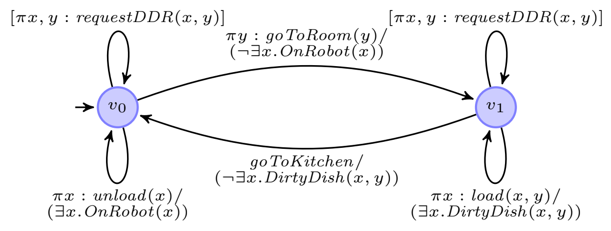
Theorem 10. *Let Σ be a BAT and α a bounded sentence. Then $\mathcal{R}[\alpha]$, the regression of α, is a fluent sentence and* $\Sigma \models \alpha$ *iff* $\Sigma_0 \models \mathcal{R}[\alpha]$.

3 Verification in $\mathcal{ESG}$

We encode the space of reachable program configurations by a *characteristic graph* $\mathcal{G}_\delta = \langle v_0, V, E\rangle$ for a given program δ. The nodes V in such a graph are of the form $\langle\delta', \phi\rangle$, denoting the remaining program of a current run and the condition under which execution may terminate there. v_0 is the initial node. Edges in E are labeled with tuples $\pi\vec{x} : t/\psi$, where $\vec{x}$ is a list of variables (if it is empty, we omit the leading π), t is an action term and ψ is a formula (which we omit when it is $\top$). Intuitively, this means when one wants to take action t, one has to choose instantiations for the $\vec{x}$ and ψ must hold. Due to lack of space, we omit the formal definition of characteristic graphs and refer the interested reader to [Claßen and Lakemeyer, 2008]. Figure 1 shows the graph corresponding to $\delta_{robot}\|\delta_{exo}$, where δ_{robot} denotes the control program presented in the introduction and δ_{exo} is the encoding of exogenous actions. Here it consists simply of the $requestDDR(x, y)$, which should be read as "requesting the removal of dirty dish x from room y." The nodes are $v_0 = \langle\delta_{robot}\|\delta_{exo}, \bot\rangle$ and $v_1 = \langle(\delta_1; \delta_{robot})\|\delta_{exo}, \bot\rangle$, where δ_1 is the program

$$(\pi x.DirtyDish(x, y)?; load(x, y))^*;$$
$$\neg\exists x.DirtyDish(x, y)?; goToKitchen.$$

The verification algorithms work on *labels* of the characteristic graph, where a label is given by $\langle v, \psi\rangle$ with $v \in V$ and ψ being a fluent formula. Intuitively, it represents all program configurations corresponding to v as well as all worlds w and action name sequences z satisfying ψ. A *labelling* is then given by a set of labels, one for each node of the graph. We need the following operations on labellings, as formalized below: Initial labelling with a formula, conjunction and disjunction of labellings, extraction of the label formula from



$[\pi x, y : requestDDR(x,y)]$
$\pi y : goToRoom(y)/$
$(\neg\exists x.OnRobot(x))$
$[\pi x, y : requestDDR(x,y)]$
v_0
v_1
$goToKitchen/$
$(\neg\exists x.DirtyDish(x,y))$
$\pi x : unload(x)/$
$(\exists x.OnRobot(x))$
$\pi x : load(x,y)/$
$(\exists x.DirtyDish(x,y))$

Figure 1: Characteristic graph for the robot example

the initial node, and the pre-image of a labelling:

$$\textsc{Label}[\langle V,E,v_0\rangle,\alpha] \overset{def}{=} \{\langle v,\alpha\rangle \mid v\in V\}$$

$$L_1 \textsc{ And } L_2 \overset{def}{=} \{\langle v,\psi_1\wedge\psi_2\rangle \mid \langle v,\psi_1\rangle\in L_1, \langle v,\psi_2\rangle\in L_2\}$$

$$L_1 \textsc{ Or } L_2 \overset{def}{=} \{\langle v,\psi_1\vee\psi_2\rangle \mid \langle v,\psi_1\rangle\in L_1, \langle v,\psi_2\rangle\in L_2\}$$

$$\textsc{InitLabel}[\langle V,E,v_0\rangle,L] \overset{def}{=} \psi \text{ such that } \langle v_0,\psi\rangle\in L$$

$$\textsc{Pre}[\langle V,E,v_0\rangle,L] \overset{def}{=} \{\langle v,\textsc{Pre}[v,L]\rangle \mid v\in V\}$$

where

$$\textsc{Pre}[v,L] \overset{def}{=} \bigvee \{\mathcal{R}[\exists\vec{x}.\phi\wedge[t]\psi] \mid v \xrightarrow{\pi\vec{x}:t/\phi} v'\in E,\ \langle v',\psi\rangle\in L\}.$$

Roughly, the pre-image of a label gives us a description of the predecessor configuration of that label. (Note the use of regression to eliminate the action term t.)

The verification algorithm works on a CTL-like fragment of $\mathcal{ESG}$:

$$\begin{aligned}\varphi ::= (t_1=t_2) \mid F(\vec{t}) \mid \varphi\wedge\varphi \mid \neg\varphi \mid \exists x.\varphi \mid \\ \langle\!\langle\delta\rangle\!\rangle \boldsymbol{G}\varphi \mid \langle\!\langle\delta\rangle\!\rangle \varphi\,\boldsymbol{U}\,\varphi\end{aligned} \tag{5}$$

where we assume that δ is a non-terminating program of the form ${\delta_1}^\omega\|\cdots\|{\delta_k}^\omega$. The algorithm then applies the following transformation.

Definition 11. Let Σ be a BAT and φ a formula according to (5). Then $\mathcal{C}[\varphi]$, the *verification transformation* of φ wrt Σ, is inductively defined by

1. $\mathcal{C}[(t_1=t_2)] = (t_1=t_2)$;
2. $\mathcal{C}[F(\vec{t})] = F(\vec{t})$;
3. $\mathcal{C}[\varphi_1\wedge\varphi_2] = \mathcal{C}[\varphi_1]\wedge\mathcal{C}[\varphi_2]$;
4. $\mathcal{C}[\neg\varphi] = \neg\mathcal{C}[\varphi]$;
5. $\mathcal{C}[\exists x.\varphi] = \exists x.\mathcal{C}[\varphi]$;
6. $\mathcal{C}[\langle\!\langle\delta\rangle\!\rangle\boldsymbol{G}\,\varphi] = \textsc{CheckEG}[\delta,\mathcal{C}[\varphi]]$;
7. $\mathcal{C}[\langle\!\langle\delta\rangle\!\rangle\varphi\,\boldsymbol{U}\,\psi] = \textsc{CheckEU}[\delta,\mathcal{C}[\varphi],\mathcal{C}[\psi]]$.

The procedure for the case of the "always" operator $\boldsymbol{G}$ is as follows:

Procedure 1 $\textsc{CheckEG}[\delta,\varphi]$

```
$L' := \textsc{Label}[\mathcal{G}_\delta,\bot]$;
$L\ := \textsc{Label}[\mathcal{G}_\delta,\varphi]$;
while $L \not\equiv L'$ do
    $L' := L$;
    $L\ := L' \textsc{ And } \textsc{Pre}[\mathcal{G}_\delta,L']$;
end while
return $\textsc{InitLabel}[\mathcal{G}_\delta,L]$
```

The while loop is exited once $L\equiv L'$ holds, defined as follows:

$$\begin{aligned}L_1\equiv L_2 \text{ iff for all } v \text{ with } \langle v,\psi_1\rangle\in L_1 \text{ and } \langle v,\psi_2\rangle\in L_2,\\ \models \psi_1\equiv\psi_2.\end{aligned}$$

A similar procedure is used for the "until" operator $\boldsymbol{U}$:

Procedure 2 $\textsc{CheckEU}[\delta,\varphi_1,\varphi_2]$

```
$L' := \textsc{Label}[\mathcal{G}_\delta,\bot]$;
$L\ := \textsc{Label}[\mathcal{G}_\delta,\top]$;
while $L \not\equiv L'$ do
    $L' := L$;
    $L\ := L' \textsc{ And } \textsc{Pre}[\mathcal{G}_\delta,L']$;
end while
$L' := \textsc{Label}[\mathcal{G}_\delta,\top]$;
$L\ := \textsc{Label}[\mathcal{G}_\delta,\varphi_2] \textsc{ And } L$;
while $L \not\equiv L'$ do
    $L' := L$;
    $L\ := L' \textsc{ Or } (\textsc{Label}[\mathcal{G}_\delta,\varphi_1] \textsc{ And } \textsc{Pre}[\mathcal{G}_\delta,L'])$;
end while
return $\textsc{InitLabel}[\mathcal{G},L]$
```

The algorithm is sound in the following sense:

Theorem 12 ([Claßen and Lakemeyer, 2008])**.** *Let Σ be a BAT, δ a program and φ a fluent sentence. Then if the computation of $\mathcal{C}[\varphi]$ terminates, it is a fluent sentence and* $\Sigma\models\varphi$ *iff* $\Sigma_0\models\mathcal{C}[\varphi]$.

4 Decidability

The algorithms presented in the previous section cannot be guaranteed to terminate for two reasons. On the one hand, equivalence checks over first-order formulas as applied in the conditions of the while loops are in general undecidable. On the other hand, even if all equivalence checks terminate, the fixpoint approximation loops may never converge.

As for the first source of non-termination, we can exploit results by Gu and Soutchanski [2010] who present a two-variable fragment of the Situation Calculus for which the projection problem (solved by means of regression) is decidable. Here we capture this fragment as a subset of the situation formulas of $\mathcal{ESG}$. We refer to this sublanguage as $\mathcal{ES}^2$.

Definition 13. $\mathcal{ES}^2$ is the subset of situation formulas according to Definition 1 that do not contain any $[\![\cdot]\!]$ operators and where $[t]$ operators are restricted to action terms t. In addition the following constraints are satisfied:

- there are no object terms other than the variables x and y or rigid constant symbols;
- all action function symbols have at most two arguments;
- fluents have at most two arguments.

In $\mathcal{ES}^2$ a *regressable* formula has to be bounded and its action terms have to be ground. Furthermore, the regression operator $\mathcal{R}$ is modified such that by means of appropriate substitutions, no new variable is introduced in the process of regression. For details, the interested reader is referred to Gu and Soutchanski's article [2010]. We then have that projection is decidable in $\mathcal{ES}^2$:

Theorem 14. *Let α be a regressable sentence of $\mathcal{ES}^2$ without standard names and Σ a BAT in $\mathcal{ES}^2$. Then $\Sigma \models \alpha$ is decidable.*

Proof. (Sketch) The proof idea for this theorem is to map $\mathcal{ES}^2$ to the decidable fragment $\mathcal{L}_{SC}^{DL}$ introduced by Gu and Soutchanski. We use a similar reduction as Lakemeyer and Levesque [2010] who embed $\mathcal{ES}$ in the original Situation Calculus. Thus, because $\mathcal{L}_{SC}^{DL}$ is decidable, $\mathcal{ES}^2$ is decidable too. $\square$

Resorting to a decidable base logic is unfortunately not sufficient to also eliminate the second source of non-termination of the verification algorithms. To see why, consider a simple BAT with the single fluent F whose successor state axiom is $\Box[a]F(x) \equiv \exists y.F(y) \wedge S(x,y)$ (where S is rigid). Let δ be the program **loop** : t for some ground action t and $\langle\!\langle\delta\rangle\!\rangle \mathbf{G}F(c)$ the sentence to verify, for some constant c. The characteristic graph of δ has only one node v_0 and one edge from v_0 to v_0 with the label t. Applying Procedure 1, we get the following label sets L in subsequent iterations:

$$\begin{aligned}
&L_0 = \{\langle v_0, F(c)\rangle\},\\
&L_1 = \{\langle v_0, F(c) \wedge \exists y.F(y) \wedge S(c,y)\rangle\},\\
&L_2 = \{\langle v_0, F(c) \wedge [\exists y.F(y) \wedge S(c,y)] \wedge\\
&\qquad \exists y.\exists x.F(x) \wedge S(y,x) \wedge S(c,y)\rangle\},\\
&L_3 = \{\langle v_0, F(c) \wedge [\exists y.F(y) \wedge S(c,y)] \wedge\\
&\qquad \exists y.\exists x.F(x) \wedge S(y,x) \wedge S(c,y)\\
&\qquad \exists y.\exists x.[\exists y.F(y) \wedge S(x,y)] \wedge S(y,x) \wedge S(c,y)\rangle\},\\
&\quad \ldots
\end{aligned}$$

Obviously, none of the formulas in this sequence is equivalent to its predecessor, and hence the algorithm never converges. Note also that we remain within $\mathcal{ES}^2$ due to re-using the two variable symbols x and y.

5 Decidability with context-free BATs

The first possibility is to restrict oneself to BATs with context-free SSAs:

Definition 15 (Context-free Successor State Axioms [Lin and Reiter, 1997])**.** A successor state axiom is *context-free* if its effect conditions, $\gamma_F^+(\vec{x},a)$ and $\gamma_F^-(\vec{x},a)$, contain no fluents (but maybe rigids). A BAT is context-free if each successor state axiom is context-free.

In order to ensure our prerequisite that formulas to be regressed only contain ground terms, we prohibit the usage of the non-deterministic pick operator π. Note that this is not such a harsh restriction as this still allows to use a "pseudo-pick" that quantifies over a finite domain of constants:

$$\pi x : \{c_1, \ldots, c_k\}.\delta \overset{def}{=} \delta_{c_1}^x | \cdots | \delta_{c_k}^x.$$

We then have the following theorem:

Theorem 16. *If Σ is a context-free BAT and δ a program without pick operators, Procedures 1 and 2 will terminate.*

Proof. (Sketch) The central property for the proof of this theorem is the following:

$$\mathcal{R}\big[[t_i]\mathcal{R}\big[[t_n]\ldots[t_i]\ldots[t_1]\varphi\big]\big] \equiv \mathcal{R}\big[[t_n]\ldots[t_i]\ldots[t_1]\varphi\big].$$

That is, regressing φ through the same ground action multiple times produces an equivalent result as only regressing once through that action. Because the program (and thus the characteristic graph) has only finitely many actions all of which are ground, there are ony finitely many such sequences of actions to consider. We then exploit the fact that the bodies of all loops in the procedures are monotone, i.e. they either always produce a subsumer of the previous label formula, or a subsumed one. Hence, eventually the label set converges. $\square$

6 Decidability with local-effect BATs

The other option to ensure termination is to restrict ourselves to BATs whose SSAs are local-effect:

Definition 17 (Local-effect Successor State Axioms [Liu and Levesque, 2005])**.** A successor state axiom is *local-effect* if both $\gamma_F^+(\vec{x},a)$ and $\gamma_F^-(\vec{x},a)$ are disjunctions of formulas of the form $\exists\vec{z}[a = A(\vec{y}) \wedge \phi(\vec{y})]$, where A is an action function, $\vec{y}$ contains $\vec{x}$, $\vec{z}$ is the remaining variables of $\vec{y}$. ϕ is called a *context formula* and contains no quantifiers. A BAT is local-effect if each successor state axiom is local-effect.

Then we have:

Theorem 18. *If Σ is a local-effect BAT and δ a program without pick operators, Procedures 1 and 2 will terminate.*

Proof. (Sketch) The proof of this theorem relies on the fact that we have only finitely many action terms in the graph (all of which are ground) and only finitely many fluents in the action theory. Furthermore, instantiating a successor state axiom by a ground action during regression yields a quantifier-free formula. Since there are only finitely many such instantiations and only finitely many edge condition formulas in the graph, we get finitely many equivalence classes of possible label formulas. Using the monotonicity argument again, termination is guaranteed. $\square$

Example 19. Recapitulating the example from the beginning, we show here a verification run for a local-effect BAT. Fortunately, the example is already in the two-variable fragment. We only need to change the program slightly by replacing the pick operators by the pseudo-picks. Then we have the following successor state axioms:

$$\begin{aligned}&\Box[a]\mathit{DirtyDish}(x,y) \equiv a = \mathit{requestDDR}(x,y) \vee\\ &\quad \mathit{DirtyDish}(x,y) \wedge \neg[a = \mathit{load}(x,y)]\\ &\Box[a]\mathit{OnRobot}(x) \equiv \exists y.\, a = \mathit{load}(x,y) \vee\\ &\quad \mathit{OnRobot}(x) \wedge \neg[a = \mathit{unload}(x)].\end{aligned}$$

We omit other fluents like the location of the robot for simplicity. Additionally, we have the following precondition axiom:

$$\begin{aligned}&\Box\mathit{Poss}(a) \equiv [\exists x,y.\, a = \mathit{requestDDR}(x,y)] \vee\\ &\quad [\exists x,y.\, a = \mathit{load}(x,y)] \vee [\exists x.\, a = \mathit{unload}(x)].\end{aligned}$$

The only exogenous actions axiom is

$$\Box\mathit{Exo}(a) \equiv \exists x,y.\, a = \mathit{requestDDR}(x,y).$$

Finally, the following is the GOLOG program δ'_{robot} with pseudo-picks, where d_i is a constant for a dish and r_i for a room:

```
loop :  while ($\exists x.OnRobot(x)$) do
            $\pi x : \{d_1, d_2, d_3\}.unload(x)$
        endWhile;
        $\pi y : \{r_1, r_2\}.goToRoom(y)$;
        while ($\exists x.DirtyDish(x, y)$) do
            $\pi x : \{d_1, d_2, d_3\}.load(x, y)$
        endWhile;
        goToKitchen
```

We now want to verify the following formula φ for δ'_{robot}: $\neg\exists x,y.\langle\langle\delta'_{robot}\rangle\rangle\boldsymbol{G}\mathit{DirtyDish}(x,y)$. This means there cannot be any infinite run of δ'_{robot} where some dish in some room remains dirty forever. The algorithm starts with $\mathcal{C}[\varphi]$ resulting in $\neg\exists x,y.\text{CHECKEG}[\delta'_{robot}, \mathit{DirtyDish}(x,y)]$. Then Procedure 1 starts with the following label set:

$$L_0 = \{\langle v_0, \mathit{DirtyDish}(x,y)\rangle, \langle v_1, \mathit{DirtyDish}(x,y)\rangle\}$$

$$\begin{aligned}&\text{PRE}[v_0, L_0]\\ &\equiv \mathcal{R}[(\neg\exists x.\mathit{OnRobot}(x)) \wedge\\ &\qquad [\mathit{goToRoom}(r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[(\neg\exists x.\mathit{OnRobot}(x)) \wedge\\ &\qquad [\mathit{goToRoom}(r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_1,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_2,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_3,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_1,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_2,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_3,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[(\exists x.\mathit{OnRobot}(x)) \wedge [\mathit{unload}(d_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[(\exists x.\mathit{OnRobot}(x)) \wedge [\mathit{unload}(d_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[(\exists x.\mathit{OnRobot}(x)) \wedge [\mathit{unload}(d_3)]\mathit{DirtyDish}(x,y)]\\ &\equiv (\neg\exists x.\mathit{OnRobot}(x)) \wedge \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_1 \wedge y = r_1 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_2 \wedge y = r_1 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_3 \wedge y = r_1 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_1 \wedge y = r_2 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_2 \wedge y = r_2 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad x = d_3 \wedge y = r_2 \vee \mathit{DirtyDish}(x,y) \vee\\ &\quad (\exists x.\mathit{OnRobot}(x)) \wedge \mathit{DirtyDish}(x,y)\\ &\equiv x = d_1 \wedge y = r_1 \vee x = d_2 \wedge y = r_1 \vee\\ &\quad x = d_3 \wedge y = r_1 \vee x = d_1 \wedge y = r_2 \vee\\ &\quad x = d_2 \wedge y = r_2 \vee x = d_3 \wedge y = r_2 \vee\\ &\quad \mathit{DirtyDish}(x,y)\end{aligned}$$

$$\begin{aligned}&\text{PRE}[v_1, L_0]\\ &\equiv \mathcal{R}[(\neg\exists x.\mathit{DirtyDish}(x,r_1)) \wedge\\ &\qquad [\mathit{goToKitchen}]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[(\neg\exists x.\mathit{DirtyDish}(x,r_2)) \wedge\\ &\qquad [\mathit{goToKitchen}]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_1,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_2,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_3,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_1,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_2,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\quad \mathcal{R}[[\mathit{requestDDR}(d_3,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_1)) \wedge [\mathit{load}(d_1,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_1)) \wedge [\mathit{load}(d_2,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_1)) \wedge [\mathit{load}(d_3,r_1)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_2)) \wedge [\mathit{load}(d_1,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_2)) \wedge [\mathit{load}(d_2,r_2)]\mathit{DirtyDish}(x,y)] \vee\\ &\mathcal{R}[(\exists x.\mathit{DirtyDish}(x,r_2)) \wedge [\mathit{load}(d_3,r_2)]\mathit{DirtyDish}(x,y)]\\ &\equiv x = d_1 \wedge y = r_1 \vee x = d_2 \wedge y = r_1 \vee x = d_3 \wedge y = r_1 \vee\\ &\quad x = d_1 \wedge y = r_2 \vee x = d_2 \wedge y = r_2 \vee x = d_3 \wedge y = r_2 \vee\\ &\quad \mathit{DirtyDish}(x,y)\end{aligned}$$

$$\begin{aligned}L_1 &= L_0 \text{ AND PRE}[\mathcal{G}_\delta, L_0]\\ &= \{\langle v_0, \mathit{DirtyDish}(x,y)\rangle, \langle v_1, \mathit{DirtyDish}(x,y)\rangle\}\end{aligned}$$

Now, $L_0 \equiv L_1$, i.e. the algorithm terminates and returns $\neg\exists x,y.\mathit{DirtyDish}(x,y)$. Thus, there is no run with some dish forever remaining dirty in some room iff there is no dirty dish initially. Intuitively, this is correct because $\boldsymbol{G}\phi$ means that ϕ persists to hold during the *entire* run, including the initial situation. Therefore, only if a dish is dirty initially it may happen that it never gets cleaned, namely when the robot never visits the corresponding room. Note that excluding this from happening would still allow the case where a dirty dish occurs at a later time of the run (due to some *requestDDR* action) and never gets cleaned from that moment on.

7 Related Work

The verification of non-terminating GOLOG programs was first discussed by De Giacomo, Ternovska and Reiter [1997], but only in the form of manual, meta-theoretic proofs, where properties were expressed using μ-calculus fixpoint formulas instead of temporal modalities. The $\mathcal{ESG}$ language and the automated verification methods used in this paper were introduced by Claßen and Lakemeyer [2008] and later extended to a larger subset [2010]. However, they proved their algorithms only to be sound, but could not give a general termination guarantee. De Giacomo, Lespérance and Pearce [2010] applied the idea of verifying GOLOG programs through iterative fixpoint approximations using characteristic graphs in the context of games and multi-agent systems, where properties are expressed in Alternating-Time Temporal Logic. De Giacomo, Lespérance and Patrizi [2012] define the class of bounded action theories, for which they show that the verification of a certain class of first-order μ-calculus temporal properties is decidable.

8 Conclusion

In this paper, we showed that the problem of verifying non-terminating GOLOG programs is indeed decidable for a relevant subset of the formalism, which was achieved by means of three restrictions. First, we used the $\mathcal{ES}$ variant of a decidable two-variable fragment of the Situation Calculus as introduced by Gu and Soutchanski. Second, we have to restrict the GOLOG program to contain ground action only. Finally, we considered special classes of successor state axioms, namely the context-free ones and those that only admit local effects. Interesting lines of future work would be to come up with a solution for re-introducing the original pick operator and to obtain complexity results for our approach.

References

[Baader *et al.*, 2003] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[Baader *et al.*, 2005] Franz Baader, Carsten Lutz, Maja Miličić, Ulrike Sattler, and Frank Wolter. Integrating description logics and action formalisms: First results. In *Proc. AAAI 2005*, pages 572–577. AAAI Press, 2005.

[Baader *et al.*, 2008] Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over description logic axioms. In *Proc. KR 2008*, pages 684–694. AAAI Press, 2008.

[Baader *et al.*, 2010] Franz Baader, Hongkai Liu, and Anees ul Mehdi. Verifying properties of infinite sequences of description logic actions. In *Proc. ECAI 2010*, pages 53–58. IOS Press, 2010.

[Burgard *et al.*, 1999] Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2):3–55, 1999.

[Claßen and Lakemeyer, 2008] Jens Claßen and Gerhard Lakemeyer. A logic for non-terminating Golog programs. In *Proc. KR 2008*, pages 589–599. AAAI Press, 2008.

[Claßen and Lakemeyer, 2010] Jens Claßen and Gerhard Lakemeyer. On the verification of very expressive temporal properties of non-terminating Golog programs. In *Proc. ECAI 2010*, pages 887–892. IOS Press, 2010.

[De Giacomo *et al.*, 1997] Giuseppe De Giacomo, Evgenia Ternovska, and Raymond Reiter. Non-terminating processes in the situation calculus. In *Working Notes of "Robots, Softbots, Immobots: Theories of Action, Planning and Control", AAAI'97 Workshop*, 1997.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *Proc. KR 2010*, pages 445–455. AAAI Press, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories and decidable verification. In *Proc. KR 2012*. AAAI Press, 2012.

[Gu and Soutchanski, 2010] Yilan Gu and Mikhail Soutchanski. A description logic based situation calculus. *Annals of Mathematics and Artificial Intelligence*, 58(1–2):3–83, 2010.

[Lakemeyer and Levesque, 2010] Gerhard Lakemeyer and Hector J. Levesque. A semantic characterization of a useful fragment of the situation calculus with knowledge. *Artificial Intelligence*, 175(1):142–164, 2010.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

[Lin and Reiter, 1997] Fangzhen Lin and Raymond Reiter. How to progress a database. *Artificial Intelligence*, 92(1–2):131–167, 1997.

[Liu and Levesque, 2005] Yongmei Liu and Hector J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proc. IJCAI 2005*, pages 522–527. Professional Book Center, 2005.

[McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. American Elsevier, New York, 1969.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

Experiments in Infinite States Verification in Game-Theoretic Logics

Slawomir Kmiec and **Yves Lespérance**
Dept. of Computer Science and Engineering,
York University, Toronto, Canada
skmiec@cse.yorku.ca and lesperan@cse.yorku.ca

Abstract

Many practical problems where the environment is not in the system's control such as service orchestration and contingent and multi-agent planning can be modelled in game-theoretic logics (e.g. ATL). But most work on verification methods for such logics is restricted to finite state cases. [De Giacomo *et al.*, 2010] develops a situation calculus-based logical framework for representing such infinite state game-type problems together with a verification method based on fixpoint approximates and regression. In this paper, we describe some case studies we have done to evaluate this method. We specify some example domains and show that the verification method does allow us to verify various properties. We also find some examples where the method must be extended to exploit information about the initial state and state constraints in order to work. Our example domains can be used to evaluate other infinite state verification methods.

1 Introduction

Many practical problems where the environment is not completely under the system's control, such as service orchestration and contingent and multi-agent planning, can be modeled as games and specified in game-theoretic logics. There has been much work to define such logics (e.g. ATL) and develop verification methods for them, mainly model checking techniques [Alur *et al.*, 2002]. However, most such work is restricted to finite state settings. [De Giacomo *et al.*, 2010] develops an expressive logical framework for specifying such problems within the situation calculus [McCarthy and Hayes, 1969]. In their approach, a game-like problem/setting is represented as a *situation calculus game structure*, a special kind of action theory that specifies who are the players, what the legal moves are, etc. They also define a logic that combines the μ-calculus, game-theoretic path quantifiers as in ATL, and first-order quantification, for specifying properties about such game settings. As well, they propose a procedural language for defining game settings, GameGolog, which is based on ConGolog [De Giacomo *et al.*, 2000]. Finally, they propose a method for verifying temporal properties over infinite state game structures that is based on fixpoint approximates and regression. The method is also adapted for GameGolog-defined settings to exploit a compact "characteristic graph" [Claßen and Lakemeyer, 2008] representation of the program's configuration space.

While [De Giacomo *et al.*, 2010] give examples to illustrate the expressiveness and convenience of their formalism, they recognize that their work is essentially theoretical and call for experimental studies to understand whether these techniques actually work in practice. This is what we begin to address in this paper. We develop several example problems involving infinite state domains and represent them as situation calculus game structures. We then examine whether the [De Giacomo *et al.*, 2010] fixpoint approximates verification method works to verify common temporal properties. In many cases, it does indeed work. So to some extent, our work validates the [De Giacomo *et al.*, 2010] proposal.

We do however find other examples where the [De Giacomo *et al.*, 2010] method does not converge in a finite number of steps. We note that the method uses only the simplest part of the action theory, the unique name and domain closure axioms, to try to show that successive approximates are equivalent (after performing regression). Clearly, using the whole action theory is problematic as it includes a second order axiom to specify the domain of situations. We show that in some cases, adding a few key facts that are entailed by the entire theory (from simple axioms about the initial state to state constraints proven by induction) is sufficient to get convergence in a finite number of steps. This means that the method can be used successfully in a wider range of problems if we can rely on the modeler to identify such facts. Thus, our case studies show that the methods introduced in [De Giacomo *et al.*, 2010] often do work for infinite domains, where very few verification methods are available, and allow reasoning about a range of game problems. Note that in our case studies, the fixpoint approximation method was performed manually. We discuss implementation in the conclusion.

2 Situation Calculus Game Structures

2.1 Situation Calculus and Basic Action Theories

The Situation Calculus (SitCalc) is a many sorted predicate logic language for representing dynamically changing worlds in which all changes are the result of named actions [Mc-

Carthy and Hayes, 1969; Reiter, 2001]. Actions are terms in the language, e.g. $pickup(R, X)$ could represent an action where a robot R picks up an object X. Action terms are denoted by α possibly with subscripts to differentiate different action terms. Action variables are denoted by lower case letters *a* possibly with subscripts. Action types, i.e. actions functions, which may require parameters, are denoted by upper case letters *A* possibly with subscripts. Situations represent possible world histories and are terms in the language. The distinguished constant S_0 denotes the initial situation where no action has yet been performed. The distinguished function symbol do is used to build sequences of actions such that $do(a, s)$ denotes the successor situation that results from performing action a in situation s. Fluents are predicates or functions whose values may vary from situation to situation. They are denoted by symbols that take a situation term as their last argument. A distinguished predicate symbol *Poss(a,s)* is used to state that an action *a* is executable in a situation *s*.

Given this language, one can specify action theories that describe how the world changes as the result of the available actions. We focus on *basic action theories* as proposed in [Reiter, 2001]. We assume that there is a *finite number of action types* in the domains we consider. Thus a basic action theory $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent axioms of the situation calculus (Σ); precondition axioms stating when actions can be legally performed ($\mathcal{D}_{poss}$); successor state axioms describing how fluents change between situations ($\mathcal{D}_{ssa}$); unique name axioms for actions and domain closure on action types ($\mathcal{D}_{ca}$); and axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). Successor state axioms specify the value of fluents in situation $do(a, s)$ in terms of the action a and the value of fluents in situation s; they encode the causal laws of the world and provide a solution to the frame problem.

2.2 Situation Calculus Game Structure Definitions

Situation calculus game structures, proposed by [De Giacomo *et al.*, 2010], are a specialization of basic action theories that allow multi-agent game-like settings to be modeled. In SitCalc game structures, every action a has an agent parameter and the distinguished function $agent(a)$ returns the agent of the action. Axioms for the $agent$ function are specified for every action type and by convention the agent parameter is the first argument of any action type. It is assumed that there is a finite set $Agents$ of agents who are denoted by unique names. Actions are divided into two groups: choice actions and standard actions. Choice actions model the decisions of agents and they are assumed to have no effect on any fluent other than $Poss$, $Legal$, and $Control$. $Poss(a, s)$ specifies that an action a is physically possible (i.e. executable) in situation s. Choice actions are always physically possible. Standard actions are the other non-choice actions. There is also a distinguished predicate $Legal(s)$ that is a stronger version of possibility/legality and models the game structure of interest. It specifies what actions an agent may execute and what choices can be made according to the rules of the game. The axioms provided for $Legal$ specify the game of interest. It is required that the axioms for $Legal$ entail 3 properties:

1. $Legal$ implies physically possible
   $Legal(s) \supset s = S_0 \vee \exists a, s'.s = do(a, s') \wedge Poss(a, s')$
2. legal situations are result of an action performed in legal situations
   $Legal(s) \supset s = S_0 \vee \exists a, s'.s = do(a, s') \wedge Legal(s')$
3. only one agent can act in a legal situation
   $Legal(do(a, s)) \wedge Legal(do(a', s)) \supset agent(a) = agent(a')$

$Control(agt, s)$ holds if agent agt is the one that is in control and can act in a legal situation s; it is defined as follows:

$$Control(agt, s) \doteq \exists a.Legal(do(a, s)) \wedge agent(a) = agt$$

As a result of the above constraints on $Legal$, it follows that the predicate $Control$ holds for only one agent in a any given legal situation. As explained in [De Giacomo *et al.*, 2010], games where several agents act simultaneously can be modeled using a round-robin of choice actions; if the result of such simultaneous choices is non-deterministic, a "game master" agent that makes the decision can be introduced. It is worth noting that the state of the game in situation s is captured by the fluents. Finally, [De Giacomo *et al.*, 2010] define a SitCalc game structure to be an action theory $\mathcal{D}_{GS} = \Sigma \cup \mathcal{D}_{poss} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{ca} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{legal}$ where $\mathcal{D}_{legal}$ contains the axioms for $Legal$ and $Control$ and for the function $agent()$, and the other components are as for standard basic action theories. Note that here, a game structure is a type of situation calculus theory and not a single game model as is often the case.

2.3 Property Language

[De Giacomo *et al.*, 2010] introduces a logical language $\mathcal{L}$ for expressing temporal properties of game structures. It is inspired by ATL [Alur *et al.*, 2002] and based on the μ-calculus [Park, 1976], as used over game structures as in [Bradfield and Stirling, 2007]. The key element of the $\mathcal{L}$-logic is the $\langle\langle G\rangle\rangle \bigcirc \varphi$ operator defined as follows:

$$\begin{aligned}
&\langle\langle G\rangle\rangle \bigcirc \varphi \doteq \\
&\quad (\exists agt \in G.\ Control(agt, now) \wedge \\
&\qquad \exists a.\ agent(a) = agt \wedge \\
&\qquad\quad Legal(do(a, now)) \wedge \varphi[do(a, now)]) \vee \\
&\quad (\exists agt \notin G.\ Control(agt, now) \wedge \\
&\qquad \forall a.\ agent(a) = agt \wedge \\
&\qquad\quad Legal(do(a, now)) \supset \varphi[do(a, now)])
\end{aligned}$$

This operator, in essence, specifies that a coalition G of agents can ensure that ϕ holds next, i.e. after one more action, as follows. If an agent from the coalition G is in control in the current situation, then all we need is that there be some legal action that this agent can perform to make the formula ϕ hold. If the agent in control is not in coalition G, then what we need is that regardless of the action taken by the in-control agent (for all) the formula ϕ holds after the action. The whole logic $\mathcal{L}$ is defined as follows:

$$\begin{aligned}
&\Psi \leftarrow \varphi \mid Z(\vec{x}) \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2 \mid \exists x.\Psi \mid \forall x.\Psi \mid \\
&\langle\langle G\rangle\rangle \bigcirc \Psi \mid [[G]] \bigcirc \Psi \mid \mu Z(\vec{x}).\Psi(Z(\vec{x})) \mid \nu Z(\vec{x}).\Psi(Z(\vec{x})).
\end{aligned}$$

In the above φ is an arbitrary, possibly open, situation-suppressed situation calculus uniform formula, Z is a predicate variable of a given arity, $\langle\langle G\rangle\rangle \bigcirc \Psi$ is as defined above,

$[[G]] \bigcirc \Psi$ is the dual of $\langle\langle G\rangle\rangle \bigcirc \Psi$ (i.e., $[[G]] \bigcirc \Psi \equiv \neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$[1]), and μ (resp. ν) is the least (resp. greatest) fixpoint operator from the μ-calculus, where the argument is written as $\Psi(Z(\vec{x}))$ to emphasize that $Z(\vec{x})$ may occur free, i.e., not quantified by μ or ν in Ψ.

The language $\mathcal{L}$ allows one to express arbitrary temporal/dynamic properties. For example, the property that group G can ensure that eventually $\varphi(\vec{x})$ (or has a strategy to achieve $\varphi(\vec{x})$), where $\varphi(\vec{x})$ is a situation suppressed formula with free variables $\vec{x}$, may be expressed by the following least fixpoint construction:

$$\langle\langle G\rangle\rangle\Diamond\varphi(\vec{x}) \doteq \mu Z(\vec{x}).\, \varphi(\vec{x}) \lor \langle\langle G\rangle\rangle \bigcirc Z(\vec{x})$$

Similarly, group G's ability to maintain a property $\varphi(\vec{x})$ can be expressed by the following greatest fixpoint construction:

$$\langle\langle G\rangle\rangle\Box\varphi(\vec{x}) \doteq \nu Z(\vec{x}).\varphi(\vec{x}) \land \langle\langle G\rangle\rangle \bigcirc Z(\vec{x})$$

We say that there is a path where $\varphi(\vec{x})$ holds next if the set of all agents can ensure that $\varphi(\vec{x})$ holds next: $\exists \bigcirc \varphi(\vec{x}) \doteq \langle\langle Agents\rangle\rangle \bigcirc \varphi(\vec{x})$. Similarly there is a path where $\varphi(\vec{x})$ eventually holds if the set of all agents has a strategy to achieve $\varphi(\vec{x})$: $\exists\Diamond\varphi(\vec{x}) \doteq \langle\langle Agents\rangle\rangle\Diamond\varphi(\vec{x})$.

2.4 Fixpoint Iteration Verification Method

[De Giacomo *et al.*, 2010] propose a procedure based on regression and fixpoint approximation to verify formulas of logic $\mathcal{L}$ given a SitCalc game structure theory. This recursive procedure $\tau(\Psi)$ tries to compute a first-order formula uniform in current situation *now* that is equivalent to Ψ:

$$\begin{aligned}
&\tau(\varphi) = \varphi\\
&\tau(Z) = Z\\
&\tau(\Psi_1 \land \Psi_2) = \tau(\Psi_1) \land \tau(\Psi_2)\\
&\tau(\Psi_1 \lor \Psi_2) = \tau(\Psi_1) \lor \tau(\Psi_2)\\
&\tau(\exists x.\Psi) = \exists x.\tau(\Psi)\\
&\tau(\forall x.\Psi) = \forall x.\tau(\Psi)\\
&\tau(\langle\langle G\rangle\rangle \bigcirc \Psi) = \mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(\Psi))\\
&\tau([[G]] \bigcirc \Psi) = \neg\mathcal{R}(\langle\langle G\rangle\rangle \bigcirc \tau(\text{NNF}(\neg\Psi)))\\
&\tau(\mu Z.\Psi) = \mathit{lfp}Z.\tau(\Psi)\\
&\tau(\nu Z.\Psi) = \mathit{gfp}Z.\tau(\Psi)
\end{aligned}$$

In the above, $\mathcal{R}$ represents the regression operator and $\langle\langle G\rangle\rangle \bigcirc \Psi$ is regressable if Ψ is regressable, NNF$(\neg\Psi)$ denotes the negation normal form of $\neg\Psi$, and

- *lfpZ*.Ψ is the formula R resulting from the least fixpoint procedure

```
$R := False$;
$R_{new} := \Psi(False)$;
**while** ($\mathcal{D}_{ca} \not\models R \equiv R_{new}$){
    $R := R_{new}$;
    $R_{new} := \Psi(R)$ }
```

- *gfpZ*.Ψ is the formula R resulting from the greatest fixpoint procedure

```
$R := True$;
$R_{new} := \Psi(True)]$;
**while** ($\mathcal{D}_{ca} \not\models R \equiv R_{new}$){
    $R := R_{new}$;
    $R_{new} := \Psi(R)$ }
```

[1]Although $\neg\langle\langle G\rangle\rangle \bigcirc \neg\Psi$ is not in $\mathcal{L}$ according to the syntax, the equivalent formula in negation normal form is.

The fixpoint procedures test if $R \equiv R_{new}$ is entailed given only the unique name and domain closure for actions axioms $\mathcal{D}_{ca}$. In general, there is no guarantee that such procedures will ever terminate i.e. that for some i $\mathcal{D}_{ca} \models R_i \equiv R_{i+1}$. But if the *lfp* procedure does terminate, then $\mathcal{D}_{GS} \models R_i[S] \equiv \mu Z.\Psi(Z)[S]$ and R_i is first-order and uniform in S (and similarly *gfp*). In such cases, the task of verifying a fixpoint formula in the situation calculus is reduced to that of verifying a first-order formula. We have the following result:

Theorem 1. *[De Giacomo* et al.*, 2010] Let $\mathcal{D}_{GS}$ be a situation calculus game structure and let Ψ be an $\mathcal{L}$-formula. If the algorithm above terminates, then $\mathcal{D}_{GS} \models \Psi[S_0]$ iff $\mathcal{D}_{S_o} \cup \mathcal{D}_{ca} \models \tau(\Psi)[S_0]$.*

3 Case Studies

3.1 Light World (LW)

Our first example domain is the Light World (LW), a simple game we designed that involves an infinite row of lights, one for each integer. A light can be on or off. Each light has a switch that can be flipped, which will turn the light on (off resp.) if it was off (on resp.). There are 2 players, X and O. Players take turns and initially it is X's turn. The goal of player X is to have lights 1 and 2 on in which case player X wins the game. Initially, only light 5 is on. Note that this is clearly an infinite state domain as the set of lights that can be turned on or off is infinite. Note also that the game may go on forever without the goal being reached (e.g., if player O keeps turning light 1 or 2 off whenever X turns them on).

We will show that the fixpoint approximation method of [De Giacomo *et al.*, 2010] can be used to verify some interesting properties in this domain. We apply the method with one small modification: when checking whether the two successive approximates are equivalent, we use a suitable axiomatization of the integers D_Z in addition to the unique names and domain closure axioms for actions D_{ca}^{LW}, as our game domain involves one light for every integer. [2]

The game structure axiomatization for this domain is: $\mathcal{D}_{GS}^{LW} = \Sigma \cup \mathcal{D}_{poss}^{LW} \cup \mathcal{D}_{ssa}^{LW} \cup \mathcal{D}_{ca}^{LW} \cup \mathcal{D}_{S_0}^{LW} \cup \mathcal{D}_{Legal}^{LW} \cup \mathcal{D}_Z$. We have only one action $flip(p,t)$, meaning that player p flips light t, with the precondition axiom (in $\mathcal{D}_{poss}^{LW}$):

$$Poss(flip(p,t),s) \equiv Agent(p)$$

We have the fluents $On(t,s)$, meaning that light t is on in situation s, and $turn(s)$, a function that denotes the agent whose turn it is in s. The successor state axioms (in $\mathcal{D}_{ssa}^{LW}$) are as follows:

$$\begin{aligned}
&On(t,do(a,s)) \equiv \exists p\, a = flip(p,t) \land \neg On(t,s) \lor\\
&\quad On(t,s) \land \forall p.a \neq flip(p,t)
\end{aligned}$$

$$\begin{aligned}
&turn(do(a,s)) = p \equiv\\
&\quad p = O \land turn(s) = X \lor p = X \land turn(s) = O
\end{aligned}$$

[2]Our axioms and the properties we attempt to verify only use a very simple part of integer arithmetic. It should be possible to generate the proofs using the decidable theory of Presburger arithmetic [Enderton, 1972] after encoding integers as pairs of natural numbers in the standard way [Hamilton, 1982]. Most theorem proving systems include sophisticated solvers for dealing with formulas involving integer constraints and it should be possible to use these to perform the reasoning about integers that we require.

The rules of the game are specified using the *Legal* predicate. We have the following axioms in $\mathcal{D}^{LW}_{legal}$:

$$\begin{aligned}&Legal(do(a,s)) \equiv Legal(s) \wedge\\ &\quad \exists p,t.\ Agent(p) \wedge turn(s)=p \wedge a = flip(p,t)\\ &agent(flip(p,t)) = p\\ &Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a)=p\\ &\forall p.\{Agent(p) \equiv (p = X \vee p = O)\}, \qquad X \neq O\end{aligned}$$

Thus legal moves involve the player whose turn it is flipping any switch. We have the following unique name and domain closure axioms for actions in $\mathcal{D}^{LW}_{ca}$:

$$\begin{aligned}&\forall a.\ \{\ \exists p,t.\ a = flip(p,t)\}\\ &\forall p,p',t,t'.\ \{\ flip(p,t) = flip(p',t') \supset p = p' \wedge t = t'\ \}\end{aligned}$$

Finally, the initial state axioms in $\mathcal{D}^{LW}_{S_0}$ are: $turn(S_0) = X$, $\neg On(1,S_0)$, $\neg On(2,S_0)$, $On(5,S_0)$, and $Legal(S_0)$.

For our first verification example, we consider the property that it is possible for X to eventually win (assuming O cooperates), which can be represented by the following formula:

$$\exists\Diamond Wins(X) \doteq \mu Z.Wins(X)\ \vee\ \exists \bigcirc Z,$$

where $Wins(X,s) \doteq Legal(s) \wedge On(1,s) \wedge On(2,s)$. We apply the [De Giacomo *et al.*, 2010] fixpoint iteration verification method to this example. We can show that the regressed approximations simplify as follows (see [Kmiec, 2013] for a more detailed version of all proofs in this paper):

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$
$\quad Legal(s) \wedge On(1,s) \wedge On(2,s)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps), i.e., if light 1 and light 2 are on in s.

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathbf{D_Z} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$
$\quad Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$\quad Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(1,s) \vee$
$\quad Legal(s) \wedge (turn(s) = X \vee turn(s) = O) \wedge On(2,s)$

This approximation evaluates to true if s is such that X can win in at most 1 step; these are legal situations where player X is already winning or where one of lights 1 or 2 is on, as X or O can turn the other light on at the next step.

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathbf{D_Z} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$
$\quad Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$\quad Legal(s) \wedge (turn(s) = X \vee turn(s) = O)$

This approximation evaluates to true if s is such that X can win in at most 2 steps; this is the case if X is winning already or if s is any legal situation where it is one of the players' turn, as the controlling player can turn light 1 on at the next step and the other player can and light 2 on at the second step).

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathbf{D_Z} \models \mathbf{R_3(s)} \equiv \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_2}) \equiv$
$\quad Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$\quad Legal(s) \wedge (turn(s) = X \vee turn(s) = O)$

The fixpoint iteration procedure converges at the 4^{th} step as we have: $\mathcal{D}^{LW}_{ca} \cup D_Z \models R_2(s) \equiv R_3(s)$. By the way, note that it can be shown using the entire theory (by induction on situations) that $\mathcal{D}^{LW}_{GS} \models R_2(s) \equiv Legal(s)$, as it is always either X's or O's turn. In essence, it is possible for X to eventually win in any legal situation. It then follows by Theorem 1 of [De Giacomo *et al.*, 2010] that: $\mathcal{D}^{LW}_{GS} \models \exists\Diamond Wins(X)[S_0]$ iff $\mathcal{D}^{LW}_{GS} \models Legal(S_0) \wedge \{On(1,S_0) \wedge On(2,S_0) \vee turn(S_0) = X \vee turn(S_0) = O\}$. By the initial state axioms, the latter holds so $\mathcal{D}^{LW}_{GS} \models \exists\Diamond Wins(X)[S_0]$, i.e., player X can eventually win in the initial situation.

For our second verification example, we look at the property that X can ensure that he/she eventually wins no matter what O does, i.e., the existence of a strategy that ensures $Wins(X)$. This can be represented by the following formula:

$$\langle\langle\{X\}\rangle\rangle\Diamond Wins(X) \doteq \mu Z.\ Wins(X) \vee \langle\langle\{X\}\rangle\rangle \bigcirc Z$$

We apply the [De Giacomo *et al.*, 2010] method to try verify this property. We can show that the regressed approximations simplify as follows:

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{False})$
$\quad \equiv Legal(s) \wedge On(1,s) \wedge On(2,s)$

This approximation evaluates to true if s is such that X is already winning in s (in no steps); these are situations where lights 1 and 2 are already on.

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_0})$
$\equiv \quad Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$\quad Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$
$\quad Legal(s) \wedge turn(s) = X \wedge On(2,s)$

This approximation evaluates to true if s is such that X can ensure it wins in at most 1 step. This holds if lights 1 and 2 are already on or if either light 1 or 2 is on and it is player X's turn, as X can then turn the other light on at the next step.

$\mathcal{D}^{\mathbf{LW}}_{\mathbf{ca}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \equiv \mathbf{Wins(X,s)} \vee \mathcal{R}(\langle\langle\{\mathbf{X}\}\rangle\rangle \bigcirc \mathbf{R_1})$
$\equiv \quad Legal(s) \wedge On(1,s) \wedge On(2,s) \vee$
$\quad Legal(s) \wedge turn(s) = X \wedge On(1,s) \vee$
$\quad Legal(s) \wedge turn(s) = X \wedge On(2,s)$

Thus the fixpoint iteration procedure converges in the 3^{rd} step as we have: $\mathcal{D}^{LW}_{ca} \cup \mathcal{D}_Z \models R_1(s) \equiv R_2(s)$. Therefore by Theorem 1 of [De Giacomo *et al.*, 2010]: $\mathcal{D}^{LW}_{GS} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0] \equiv R_1(S_0)$ Since both lights 1 and 2 are off initially, it follows by the initial state axioms that $\mathcal{D}^{LW}_{GS} \models \neg\langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_0]$, i.e., there is no winning strategy for X in S_0. However, we also have that $\mathcal{D}^{LW}_{GS} \models \langle\langle\{X\}\rangle\rangle\Diamond Wins(X)[S_1]$, where $S_1 = do(flip(O,3), do(flip(X,1), S_0))$, i.e., X has a winning strategy in the situation S_1 where X first turned light 1 on and then O flipped light 3, as X can turn on light 2 next.

Note that when the fixpoint approximation method is able to show that a coalition can ensure that a property holds eventually, and the theory is complete and we have domain closure, we can always extract a strategy that the coalition can follow to achieve the property: a strategy works if it always selects actions for the coalition that get it from one approximate to a lower approximate (R_i to R_{i-1}).

3.2 Oil Lamp World (OLW)

The [De Giacomo *et al.*, 2010] fixpoint approximation method tries to detect convergence by checking if the i-th approximate is equivalent to the $(i+1)$-th approximate using only the unique name and domain closure axioms for actions D_{ca} (to which we have added the axiomatization of the integers). We now give an example where this method does not converge in a finite number of steps. However, we also show that if we use some additional facts that are entailed by the entire theory D_{GS}^{OLW}, including the initial state axioms, when checking if successive approximates are equivalent, then we do get convergence in a finite number of steps.

Consider the Oil Lamp World (OLW), a variant of the Light World (LW) domain discussed earlier. It also involves an infinite row of lamps one for each integer, which can be on or off. A lamp has an igniter that can be flipped. When this happens, the lamp will go on provided that the lamp immediately to the right is already on, i.e., flipping the igniter for lamp t will turn it on if lamp $t+1$ is already on. There is only one agent, X. The goal of X is to have lamp 1 on, in which case X wins. Observe that the game may go on indefinitely without the goal being reached, e.g., if X keeps flipping a lamp other than lamp 1 repeatedly.

The game structure axiomatization for this domain is: $\mathcal{D}_{GS}^{OLW} = \Sigma \cup \mathcal{D}_{poss}^{OLW} \cup \mathcal{D}_{ssa}^{OLW} \cup \mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_{S_0}^{OLW} \cup \mathcal{D}_{Legal}^{LW} \cup \mathcal{D}_Z$. As in the previous example, we have only one action, $flip(p,t)$, meaning that p flips the igniter on light t, with the following precondition axiom (in $\mathcal{D}_{poss}^{LW}$):

$$Poss(flip(p,t),s) \equiv Agent(p)$$

But there is no turn taking in this game as there is only one agent X. We have the successor state axiom (in $\mathcal{D}_{ssa}^{LW}$):

$$\begin{aligned}&On(t, do(a,s)) \equiv\\ &\quad \exists p\, a = flip(p,t) \wedge On(t+1,s) \vee On(t,s)\end{aligned}$$

Note that once a lamp is turned on it remains on. The rules of the game are specified by the axioms in $\mathcal{D}_{legal}^{LW}$ as follows:

$$\begin{aligned}&Legal(do(a,s)) \equiv\\ &\quad Legal(s) \wedge \exists p,t.\ Agent(p) \wedge a = flip(p,t)\\ &agent(flip(p,t)) = p\\ &Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = p\\ &\forall p.\{Agent(p) \equiv p = X\}\end{aligned}$$

Thus legal moves involve X flipping any igniter. The unique name and domain closure axioms for actions and the initial state axioms are exactly as in the Light World example.

We are interested in verifying the property that it is possible for X to eventually win $\exists\Diamond Wins(X)$, where $Wins(X,s) \doteq Legal(s) \wedge On(1,s)$. We begin by applying the [De Giacomo *et al.*, 2010] method and try to show that successive approximates are equivalent using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{OLW}$ and the axiomatization of the integers $\mathcal{D}_Z$. We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv$
$\quad Legal(s) \wedge On(1,s)$

This approximation evaluates to true if s is such that X is already winning (in no steps); these are situations where lamp 1 is on.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv$
$\quad Legal(s) \wedge (On(1,s) \vee On(2,s))$

This approximation evaluates to true if s is such that X can win in at most 1 step; these are legal situations where either lamp 1 is on or where lamp 2 is on, and then X can turn lamp 1 on at the next step.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{OLW}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv$
$\quad Legal(s) \wedge (On(1,s) \vee On(2,s) \vee On(3,s))$

This approximation evaluates to true if s is such that X can win in at most 2 steps; these are legal situations where either lamp 1 is on, or where lamp 2 is on (and then X can turn lamp 1 on at the next step), or where lamp 3 is on (and then X can turn on lamps 2 and 1 at the next steps).

We can generalize and show that for all natural numbers i

$$D_{ca}^{OLW} \cup D_Z \models R_i \equiv Legal(s) \wedge \bigvee_{1 \le j \le i+1} On(j,s)$$

That is, X can win in at most i steps if some lamp between 1 and $i+1$ is on. It follows that for all i, $D_{ca}^{OLW} \cup D_Z \not\models R_i \equiv R_{i+1}$, since one can always construct a model of $D_{ca}^{OLW} \cup D_Z$ where every light except $i+2$ is off . Thus, the plain [De Giacomo *et al.*, 2010] method fails to converge in a finite number of steps.

Nonetheless, there is a way to beef up the [De Giacomo *et al.*, 2010] method to get convergence in a finite number of steps. The idea is to use some facts that are entailed by the entire theory in addition to the the unique name and domain closure axioms for actions D_{ca}^{OLW} and the integer axioms D_Z. First, we can show by induction on situations that any lamp that is on in the initial situation will remain on forever:

$$\begin{aligned}&\mathcal{D}_{GS}^{OLW} \models \phi_{op}\\ &\text{where } \phi_{op} \doteq \forall k\{On(k,S_0) \supset \forall s\, On(k,s)\}\end{aligned}$$

Then, it follows that for any natural numbers $i, j, i \le j$,

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(i+1,S_0), \phi_{op}\} \models R_j \equiv Legal(s)$$

In essence, X can eventually win in any legal situation where some lamp n is known to be on. It follows that:

$$\mathcal{D}_{ca}^{OLW} \cup \mathcal{D}_Z \cup \{On(i+1,S_0), \phi_{op}\} \models R_i \equiv R_{i+1}$$

Thus the fixpoint approximation method converges in a finite number of steps if we use the facts that some lamp n is known to be on initially and that a lamp that is on initially remains on forever. Moreover, our initial state axioms include $On(5,S_0)$. Thus, $\mathcal{D}_{GS}^{OLW} \models \exists\Diamond Wins(X)[S_0]$, i.e., X can eventually win in the initial situation, as it is legal and lamp 5 is on.

We can also show by induction on situations that if all lamps are off initially, they will remain so forever:

$$\mathcal{D}_{GS}^{OLW} - \mathcal{D}_{S_0}^{OLW} \models (\forall k\, \neg On(k,S_0)) \supset (\forall s \forall k\, \neg On(k,s))$$

Then, we can show by a similar argument as above that the fixpoint approximation method converges in a finite number of steps if we use the facts that all lamp are off initially and that if all lamps are off initially, they remain off forever.

3.3 In-Line Tic-Tac-Toe (TTT1D)

Our final example domain is more like a traditional game. It involves a one-dimensional version of the well known tic-tac-toe game that is played on an infinite vector of cells, one for each integer. We show that the [De Giacomo *et al.*, 2010] fixpoint approximation method does work to verify both winnability and the existence of a winning strategy in this game, although in the former case the proof is long and tedious. There are two players, X and O, that take turns, with X playing first. All cells are initially blank, i.e. marked B. Players can only put their mark at the left or right edge of the already marked area. The functional fluent $curn$ denotes the marking position on the left (negative) side of the marked area and similarly $curp$ denotes the marking position on the right (positive) side of the marked area. Initially, $curn$ refers to cell 0 and $curp$ to cell 1. Player p can put its mark in the cell on the left (negative) side of the marked area, i.e. the cell referred to by $curn$, by doing the action $markn(p)$. This also decreases the value $curn$ by 1 so that afterwards, it points to the next cell on the left. There is an analogous action $markp(p)$ for marking the the cell on the right (positive) side of the marked area denoted by $curp$. A player wins if it succeeds in putting its mark in 3 consecutive cells. For example, if initially we have the following sequence of moves:

$$[markp(X), markn(O), markp(X), markn(O), markp(X)],$$

then in the resulting situation the board is as follows:

$$\ldots, B_{-3}, B_{-2}, O_{-1}, O_0, X_1, X_2, X_3, B_4, B_5, \ldots$$

(with the subscript indicating the cell number) and X wins. Note that the game may go on indefinitely without either player winning, for instance if player O always mimics the last move of player X.

The game structure axiomatization for this domain is: $\mathcal{D}_{GS}^{T^31D} = \Sigma \cup \mathcal{D}_{poss}^{T^31D} \cup \mathcal{D}_{ssa}^{T^31D} \cup \mathcal{D}_{ca}^{T^31D} \cup \mathcal{D}_{S_0}^{T^31D} \cup \mathcal{D}_{Legal}^{T^31D} \cup \mathcal{D}_Z$. The precondition axioms (in $\mathcal{D}_{poss}^{T^31D}$) state that the actions $markn(p)$ and $markp(p)$ are always possible if p is an agent. The successor state axioms (in $\mathcal{D}_{ssa}^{LW}$) are as follows:

$$\begin{aligned}
&curn(do(a,s)) = k \equiv \\
&\quad \exists p.\{a = markn(p)\} \wedge curn(s) = k+1 \\
&\quad \vee curn(s) = k \wedge \forall p.\{a \neq markn(p)\} \\
&curp(do(a,s)) = k \equiv \\
&\quad \exists p.\{a = markp(p)\} \wedge curp(s) = k-1 \\
&\quad \vee curp(s) = k \wedge \forall p.\{a \neq markn(p)\} \\
&cell(k, do(a,s)) = p \equiv \\
&\quad a = markp(p) \wedge curp(s) = k \ \vee \\
&\quad a = markn(p) \wedge curn(s) = k \vee \\
&\quad cell(k,s) = p \\
&\qquad \wedge \neg\exists p'.\{a = markp(p') \wedge curp(s) = k\} \\
&\qquad \wedge \neg\exists p'.\{a = markn(p') \wedge curn(s) = k\} \\
&turn(do(a,s)) = p \equiv \\
&\quad agent(a) = X \wedge p = O \wedge turn(s) = X \\
&\quad \vee agent(a) = O \wedge p = X \wedge turn(s) = O
\end{aligned}$$

The rules of the game are specified (in $\mathcal{D}_{legal}^{T^31D}$) as follows:

$$\begin{aligned}
&Legal(do(a,s)) \equiv Legal(s) \wedge \\
&\quad \exists p.\{\ turn(s) = p \wedge agent(a) = p \\
&\qquad \wedge (a = markn(p) \ \vee a = markp(p))\ \} \\
&agent(markn(p)) = p, \qquad agent(markp(p)) = p \\
&Control(p,s) \doteq \exists a.Legal(do(a,s)) \wedge agent(a) = p \\
&\forall p.\ \{\ Agent(p) \equiv (p = X \vee p = O)\}, \qquad X \neq O
\end{aligned}$$

The unique name and domain closure axioms for actions are specified in the usual way. Finally, we have the following initial state axioms in $\mathcal{D}_{S_0}^{T^31D}$: $curn(S_0) = 0$, $curp(S_0) = 1$, $turn(S_0) = X$, and $Legal(S_0)$.

We first consider the property that it is possible for X to eventually win $\exists\Diamond Wins(X)$, where

$$\begin{aligned}
&Wins(p,s) \doteq \exists k(Legal(s) \wedge \\
&\quad ((curn(s) = k-2 \wedge cell(k-1,s) = p \wedge \\
&\qquad cell(k,s) = p \wedge cell(k+1,s) = p) \vee \\
&\quad (curp(s) = k+2 \wedge cell(k+1,s) = p \wedge \\
&\qquad cell(k,s) = p \wedge cell(k-1,s) = p)))
\end{aligned}$$

(Note that this simple definition allows both players to win.) If we apply the original [De Giacomo *et al.*, 2010] method to this property (using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{T^31D}$ and the axiomatization of the integers $\mathcal{D}_Z$ to show that successive approximates are equivalent), the fixpoint approximation procedure does eventually converge, but only after 11 steps. The proof is very long and tedious and there are numerous cases to deal with. The reason for this is that we cannot use the fact that $curn$ is always less than $curp$ and that the cells that are between them are non-blank and that the other cells are blank, which are consequences of the initial state axioms. So our proof has to deal with numerous cases where there are non-blank cells to the left of $curn$ or to the right of $curp$ (if we can rule these out, the proof becomes much simpler). Let us sketch the proof. We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{False}) \equiv Wins(X,s)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps); these are legal situations where there are 3 X marks in a row on either side.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_0}) \equiv R_0(s) \vee XCanPlayToWinNext(s)$

where $XCanPlayToWinNext(s) \doteq$

$$\begin{aligned}
&Legal(s) \wedge turn(s) = X \wedge \exists k\{ \\
&(curn(s) = k-1 \wedge cell(k,s) = X \wedge cell(k+1,s) = X) \vee \\
&(cell(k-1,s) = X \wedge cell(k,s) = X \wedge curp(s) = k+1) \vee \\
&(cell(k-1,s) = X \wedge curn(s) = k \wedge \\
&\quad cell(k+1,s) = X \wedge curp(s) = k+2) \vee \\
&(cell(k-1,s) = X \wedge cell(k,s) = X \wedge \\
&\quad curn(s) = k+1 \wedge curp(s) = k+2) \vee \\
&(curn(s) = k-2 \wedge curp(s) = k-1 \wedge \\
&\quad cell(k,s) = X \wedge cell(k+1,s) = X) \vee \\
&(curn(s) = k-2 \wedge cell(k-1,s) = X \wedge \\
&\quad curp(s) = k \wedge cell(k+1,s) = X)\}
\end{aligned}$$

This approximation evaluates to true if s is such that it is possible for X to win in at most 1 step. These are legal situations where there are 3 X marks in a row on either side, i.e. $\uparrow_n XXX$ or $XXX\uparrow_p$ ($\uparrow_n$ representing the position of $curn$ and similarly for $\uparrow_p$ and $curp$), or where it is X's turn and there are 2 X marks already and X can fill in the missing cell to get 3 in a row, i.e. $\uparrow_n XX$ or $XX\uparrow_p$ or $\uparrow_n\uparrow_p XX$ or $XX\uparrow_n\uparrow_p$ or $\uparrow_n X \uparrow_p X$ or $X \uparrow_n X \uparrow_p$.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X,s)} \vee \mathcal{R}(\exists \bigcirc \mathbf{R_1}) \equiv R_1(s) \vee$
$Legal(s) \wedge turn(s) = O \wedge \exists k\{$

$(curp(s) < k-1 \land curn(s) = k-1 \land$
$\quad cell(k,s) = X \land cell(k+1,s) = X) \lor$
$(cell(k-1,s) = X \land cell(k,s) = X \land$
$\quad curp(s) = k+1 \land k+1 < curn(s)) \lor$
$(curn(s) < k-1 \land cell(k-1,s) = X \land$
$\quad cell(k,s) = X \land curp(s) = k+1) \lor$
$(curn(s) = k-1 \land cell(k,s) = X \land$
$\quad cell(k+1,s) = X \land k+1 < curp(s)) \lor$
$cell(k-1,s) = X \land cell(k,s) = X \land$
$\quad k+1 = curn(s) \land curp(s) = k+1) \lor$
$(cell(k-1,s) = X \land cell(k,s) = X \land$
$\quad curn(s) = k+2 \land curp(s) = k+2) \lor$
$(curn(s) = k-1 \land curp(s) = k-1 \land$
$\quad cell(k,s) = X \land cell(k+1,s) = X) \lor$
$(curn(s) = k-2 \land curp(s) = k-2 \land$
$\quad cell(k,s) = X \land cell(k+1,s) = X)\}$

This approximation evaluates to true if s is such that it is possible for X to win in at most 2 steps. These are legal situations where X can win in at most 1 step as above, or where it is O's turn, and O can do a move that doesn't interfere and the result is a situation where X can win in at most 1 step; for this we have 8 cases: $\uparrow_n XX$ with $\uparrow_p < \uparrow_n$ or $XX \uparrow_p$ with $\uparrow_n > \uparrow_p$ or $X_k X \uparrow_p$ with $\uparrow_n < k$ or $\uparrow_n XX_k$ with $\uparrow_p > k$ or $XX \uparrow_{n,p}$ or $XX\_ \uparrow_{n,p}$ or $\uparrow_{n,p} XX$ or $\uparrow_{n,p}\, \_XX$.

We can keep going in this way. As we allow more steps, we have more and more classes of configurations where X can win. We can show that:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{10}(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\exists \bigcirc \mathbf{R_9}) \equiv$
$\quad Legal(s)$

Thus, it is possible for X to win in at most 10 steps in all legal situations. And we also have that:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_{11}(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\exists \bigcirc \mathbf{R_{10}}) \equiv$
$\quad Legal(s)$

i.e., it is possible for X to win in at most 11 steps in all legal situations. Thus, the fixpoint approximation procedure converges in the 11^{th} step as we have: $\mathcal{D}_{ca}^{T^31D} \cup \mathcal{D}_Z \models R_{10}(s) \equiv R_{11}(s)$. There are situations where it does take at least 10 steps/moves for X to win, for instance if we have $\uparrow_p < \uparrow_n$ with two blank cells in between, i.e., $\uparrow_p BB \uparrow_n$, and it is O's turn. The fact that $\uparrow_p < \uparrow_n$ means that the initial marks that are made will later be overwritten. It is straightforward to check that it takes at least 10 moves for X to have 3 X's in a row and win (O wins as well), for instance if O keeps playing $markn$ and X keeps playing $markp$. It follows from our convergence result by Theorem 1 of [De Giacomo *et al.*, 2010] that: $\mathcal{D}_{GS}^{T^31D} \models \exists \Diamond Wins(X)[S_0] \equiv R_{10}(S_0) \equiv Legal(S_0)$. Since we have $Legal(S_0)$ in the initial state axioms, it follows that $\mathcal{D}_{GS}^{T^31D} \models \exists \Diamond Wins(X)[S_0]$, i.e., it is possible for X to win in the initial situation.

Finally, we consider the property that X can ensure that it eventually wins $\langle\langle \{X\} \rangle\rangle \Diamond Wins(X)$. We can apply the original [De Giacomo *et al.*, 2010] method to this property (using only the unique name and domain closure axioms for actions $\mathcal{D}_{ca}^{T^31D}$ and the axiomatization of the integers $\mathcal{D}_Z$ to show that successive approximates are equivalent), and the fixpoint iteration converges after only 3 steps. We can show that the regressed approximations simplify as follows:

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_0(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{False})$
$\quad \equiv Wins(X,s)$

This approximation evaluates to true if s is such that X is winning in s already (in no steps); these are situations where there are 3 X marks in a row on either side.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_1(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{R_0})$
$\quad \equiv R_0(s) \lor XCanPlayToWinNext(s)$

This approximation evaluates to true if s is such that X can ensure to win in at most 1 step. These are legal situations where there are 3 X marks in a row on either side, or where it is X's turn and there are 2 X marks already and X can fill in the missing cell to get 3 in a row next as discussed in the possibility of winning case.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_2(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{R_1})$
$\equiv \quad R_1(s) \lor$
$\quad Legal(s) \land turn(s) = O \land$
$\quad \exists m.(curn(s) < m-2 \land cell(m-2,s) = X \land$
$\quad\quad cell(m-1,s) = X \land curp(s) = m) \land$
$\quad \exists n.(curn(s) = n-1 \land cell(n,s) = X \land$
$\quad\quad cell(n+1,s) = X \land n+1 < curp(s))$

This approximation evaluates to true if s is such that X can ensure to win in at most 2 steps. These are legal situations where X can ensure to win in at most 1 step as above, or where it is O's turn and we have both $X_k X \uparrow_p$ with $\uparrow_n < k$ and $\uparrow_n XX_k$ with $\uparrow_p > k$; then if O plays $markn$ then X can play $markp$ to win afterwards, and if O plays $markp$ then X can play $markn$ to win afterwards.

$\mathcal{D}_{\mathbf{ca}}^{\mathbf{T^31D}} \cup \mathcal{D}_{\mathbf{Z}} \models \mathbf{R_3(s)} \doteq \mathbf{Wins(X,s)} \lor \mathcal{R}(\langle\langle \{\mathbf{X}\} \rangle\rangle \bigcirc \mathbf{R_2})$
$\equiv \quad R_1(s) \lor$
$\quad Legal(s) \land turn(s) = O \land$
$\quad \exists m.(curn(s) < m-2 \land cell(m-2,s) = X \land$
$\quad\quad cell(m-1,s) = X \land curp(s) = m) \land$
$\quad \exists n.(curn(s) = n-1 \land cell(n,s) = X \land$
$\quad\quad cell(n+1,s) = X \land n+1 < curp(s))$

This approximation evaluates to true if s is such that X can ensure to win in at most 3 steps. It simplifies to exactly the same formula as $R_2(s)$. Thus the fixpoint iteration procedure converges in the 3^{rd} step as we have: $\mathcal{D}_{GS}^{T^31D} \cup \mathcal{D}_Z \models R_2(s) \equiv R_3(s)$. Therefore by Theorem 1 of [De Giacomo *et al.*, 2010]: $\mathcal{D}_{GS}^{T^31D} \models \langle\langle \{X\} \rangle\rangle \Diamond Wins(X)[S_0] \equiv R_2(S_0)$. It follows by the initial state axioms that $\mathcal{D}_{GS}^{T^31D} \models \neg \langle\langle \{X\} \rangle\rangle \Diamond Wins(X)[S_0]$ i.e., there is no winning strategy for X in S_0. But $\mathcal{D}_{GS}^{T^31D} \models \langle\langle \{X\} \rangle\rangle \Diamond Wins(X)[S_1]$ where $S_1 = do([markp(X), markn(O), markp(X), markn(O)], S_0)$ i.e., there is a winning strategy for X in a situation where X has marked twice on the right and O has marked twice on the left.

4 Discussion

In this paper, we described the results of some case studies to evaluate whether the [De Giacomo *et al.*, 2010] verification method actually works. We developed various infinite state game-type domains and applied the method to them. Our example domains are rather simple, but have features present in practical examples (e.g., the TTT1D domain is 1D version of tic-tac-toe on an infinite board). Our experiments do confirm that the method does work on several non-trivial verification problems with infinite state space. We also identify some examples where the method, which only uses the simplest part of the domain theory, the unique names and domain closure for action axioms, fails to converge in a finite number of steps. We show that in some of these cases, extending the method to use some selected facts about the initial situation and some state constraints does allow us to get convergence in a finite number of steps. Finally, our example domains and properties should be useful for evaluating other approaches to infinite state verification and synthesis. See [Kmiec, 2013] for more details about our verification experiments and proofs. It also develops an evaluation-based Prolog implementation of a version of the method for complete initial state theories with the closed world assumption. It generates successive approximates and checks if they hold in the situation of interest, but does not check if the sequence of approximates converges.

Among related work that deals with verification in infinite-states domains, let us mention [Claßen and Lakemeyer, 2008; 2010], which also uses methods based on fixpoint approximation. There, characteristic graphs are introduced to finitely represent the possible configurations that a Golog program representing a multi-agent interaction may visit. However their specification language is not a game structure logic. Also closely related is [Sardina and De Giacomo, 2009], which uses a fixpoint approximation method to compose a target process expressed as a ConGolog program out of a library of available ConGolog programs. Earlier, [Kelly and Pearce, 2007] proposed a fixpoint approximation method to verify a class of temporal properties in the situation calculus called property persistence formulas. [Shapiro *et al.*, 2002] show how a theorem proving tool can be used to verify properties of multi-agent systems specified in ConGolog and an extended situation calculus with mental states. A leading example of a symbolic model checker for multi-agent systems is MCMAS [Lomuscio *et al.*, 2009]. [Belardinelli *et al.*, 2012] show that model checking of an expressive temporal language on infinite state systems is decidable if the active domain in all states remains bounded. As well, [De Giacomo *et al.*, 2012] show that verification of temporal properties in bounded situation calculus theories where there is a bound on the number of fluent atoms that are true in any situation is decidable.

In future work, we would like to automate the symbolic fixpoint approximation method that we performed manually, perhaps by writing proof tactics in a theorem proving environment. This would require some symbolic manipulation procedures for regression, FOL simplification of the resulting formulas, and checking if two successive approximations are equivalent. It would also be desirable to develop techniques for identifying initial state properties and state constraints that can be used to show finite convergence in cases where these are needed. More generally, we need a better characterization of when this kind of method can be used successfully. Note that the [De Giacomo *et al.*, 2010] framework assumes that every agent has access to all the information specified in the theory. The framework should be generalized to deal with private knowledge and partial observability. Finally, the approach should be evaluated on real practical problems.

References

[Alur *et al.*, 2002] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[Belardinelli *et al.*, 2012] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. An abstraction technique for the verification of artifact-centric systems. In *KR*, 2012.

[Bradfield and Stirling, 2007] Julien Bradfield and Colin Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, volume 3, pages 721–756. Elsevier, 2007.

[Claßen and Lakemeyer, 2008] Jens Claßen and Gerhard Lakemeyer. A logic for non-terminating Golog programs. In *Proc. of KR'08*, pages 589–599, 2008.

[Claßen and Lakemeyer, 2010] Jens Claßen and Gerhard Lakemeyer. On the verification of very expressive temporal properties of non-terminating Golog programs. In *Proc. of ECAI'10*, pages 887–892, 2010.

[De Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *AIJ*, 121(1–2):109–169, 2000.

[De Giacomo *et al.*, 2010] G. De Giacomo, Y. Lesperance, and A. R. Pearce. Situation calculus-based programs for representing and reasoning about game structures. In *Proc. KR 2010*, pages 445–455, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded Situation Calculus Action Theories and Decidable Verification. In *KR*, 2012.

[Enderton, 1972] Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972.

[Hamilton, 1982] A.G. Hamilton. *Numbers, Sets and Axioms: The Apparatus of Mathematics*. Cambridge University Press, 1982.

[Kelly and Pearce, 2007] Ryan F. Kelly and Adrian R. Pearce. Property persistence in the situation calculus. In *Proc. IJCAI'07*, pages 1948–1953, 2007.

[Kmiec, 2013] Slawomir Kmiec. Infinite states verification in game-theoretic logics. Master's thesis, Dept. of Computer Science and Engineering, York University, 2013. To appear.

[Lomuscio *et al.*, 2009] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: A model checker for the verification of multi-agent systems. In *Proc. CAV'09*, pages 682–688, 2009.

[McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, pages 463–502. 1969.

[Park, 1976] David Park. Finiteness is mu-ineffable. *Theor. Comput. Sci.*, 3(2):173–181, 1976.

[Reiter, 2001] Ray Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[Sardina and De Giacomo, 2009] Sebastian Sardina and Giuseppe De Giacomo. Composition of ConGolog programs. In *Proc. IJCAI'09*, pages 904–910, 2009.

[Shapiro *et al.*, 2002] Steven Shapiro, Yves Lespérance, and Hector J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. In *Proc. AAMAS*, pages 19–26. Bologna, Italy, 2002.

Towards Action Languages with Norms and Deadlines

Matthias Knorr **Alfredo Gabaldon** **Ricardo Gonçalves** **João Leite** **Martin Slota**

CENTRIA & Departamento de Informática
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract

Action Languages are simple logical formalisms to describe the properties of a domain and the behavior of an agent and to reason about it. They offer an elegant solution to the frame problem, but are inapt to reason with norms in which an obligation deadline may require the agent to adapt its behavior even though no action occurred. In this paper we extend the Action Language $\mathcal{A}$ with features that allow reasoning about norms and time in dynamic domains. Unlike previous extensions of Action Languages with norms, our resulting language is expressive enough to handle different kinds of obligations with deadlines that explicitly refer to time, as well as norm violations and contrary-to-duty obligations.

1 Introduction

Open dynamic systems, e.g., systems interacting on the Web, social systems and open agent communities, have attracted increased attention in recent years. In these systems, constraints on the behavior of the participants cannot be hardwired in their specification. Instead, desirable properties are promoted by *normative systems* [Esteva *et al.*, 2001; Boella and van der Torre, 2004; Boella *et al.*, 2008; Alechina *et al.*, 2012; Bulling and Dastani, 2011; Sadiq *et al.*, 2007]. Norms, when used to govern *autonomous* agents, do not simply act as hard constraints that prevent the agent from adopting some behavior, but rather provide an indication as to how the agent *should* behave which, if not adhered to, can result in the application of sanctions or other normative effects.

In general, *norms* can be seen as a specification of what is expected to follow from a specific state of affairs, e.g., in the form of *obligations*. There are *obligations to-do*, i.e., obligations to execute an action before a deadline—e.g., to reply within one day after receiving a request, or to register before logging in; *obligations to-achieve*, i.e., obligations to bring about, before the deadline, a state of the world in which some proposition holds—e.g., to achieve a certain amount of credits within the academic year; and *obligations to-maintain*, i.e., obligations to maintain a state of the world in which some proposition holds until the deadline—e.g., to keep the contract with your mobile phone company for one year.

One important characteristic of realistic systems of norms is the prominent role of *time* and *deadlines*. Another feature of complex systems of norms are *reparative obligations*, or *contrary-to-duty obligations* [Carmo and Jones, 2002], i.e., obligations imposed as a consequence of the violation of some other obligation—e.g., to pay a fine within 10 days if the obligation to return an item by a deadline is violated.

Action Languages [Gelfond and Lifschitz, 1998] are simple logical formalisms for modeling dynamic systems, with application in *Artificial Intelligence*, *Robotics* and *Multi-Agent Systems*. A theory of an *Action Language* describes the properties of a domain and the abilities of an agent, compactly specifying a transition diagram containing all possible trajectories of the system. Action Languages provide a simple and elegant solution to the *frame problem* and enable an agent to encode the applicability of actions, their effects, describe complex interrelations between fluents and use automated planning to achieve some goal.

The combination of Action Languages and norms has received some attention in the literature. The line of work in [Craven and Sergot, 2008; Artikis *et al.*, 2009] extends action language $\mathcal{C}+$ [Giunchiglia *et al.*, 2004] for representing norms and institutional aspects of normative societies, focusing on aspects such as power and count-as rules. In [Gelfond and Lobo, 2008], an action language is extended with propositions for specifying defeasible authorization and obligation policies, but only obligations to-do are considered. However, none of the previous approaches deals with explicit time, deadlines or contrary-to-duty obligations. As it turns out, existing Action Languages, and these extensions, cannot capture the dynamics of deadlines because of the fundamental role they assign to physical *actions*, whose execution is *the only way* to cause a state change. With the introduction of norms with deadlines, and contrary-to-duty obligations, state change needs to also be triggered by the violation of other obligations, resulting from the expiration of the deadlines, which cannot be encoded in existing Action Languages.

To address this limitation, in this paper, we extend the Action Language $\mathcal{A}$[1] [Gelfond and Lifschitz, 1998] with features that allow reasoning about norms and time in dynamic do-

[1]We restrict ourselves to $\mathcal{A}$ and focus on explaining the technical details related to norms with explicit time deadlines, leaving more expressive Action Languages for future work.

mains, resulting in the Normative Action Language $\mathcal{A_N}$ and a query language that can deal with

- *obligations to-do*, *to-achieve* and *to-maintain*;
- deadlines that explicitly refer to time;
- norm violations and satisfactions;
- contrary-to-duty obligations.

At the same time, our approach solves the frame problem also for obligations and it is more amenable to implementation than other related work based on more complex formalisms (see related work in Sect. 4).

After introducing, in Sect. 2, the syntax and semantics of our normative Action Language $\mathcal{A_N}$, illustrating its use, and presenting some basic properties, in Sect. 3 we present a query language for $\mathcal{A_N}$, discuss its complexity and equivalence between theories in $\mathcal{A_N}$, before we conclude in Sect. 4.

2 Normative Action Language $\mathcal{A_N}$

We introduce the syntax and semantics of $\mathcal{A_N}$, a simple language for specifying norms, yet expressive enough to handle different kinds of obligations with deadlines, their satisfaction and violation, and contrary-to-duty obligations. We start from the deterministic Action Language $\mathcal{A}$ [Gelfond and Lifschitz, 1998] whose semantics builds on transition systems in which nodes correspond to states of the environment and edges correspond to transitions between states and are labeled by the action that causes the transition. To capture the meaning of a set of norms, we extend this transition system by expanding the states with a deontic component, and by adding a temporal dimension to transitions.

2.1 Syntax

Action Languages provide two disjoint, non-empty sets of function-free first-order atoms[2] defined over a given signature $\Sigma = \langle \mathcal{P}, \mathcal{C}, \mathcal{V} \rangle$ of pairwise disjoint sets of predicates ($\mathcal{P}$), constants ($\mathcal{C}$) and variables ($\mathcal{V}$): a set $\mathcal{A}$ of *elementary actions* and a set $\mathcal{F}$ of *physical fluents*. An *action* is a finite, possibly empty subset of $\mathcal{A}$ and can be understood as a set of elementary actions that are executed simultaneously. If convenient, we denote a singleton action $\{ \alpha \}$ with the elementary action α. Physical fluents $f \in \mathcal{F}$ and their negations $\neg f$ form the set of *physical literals*, used to represent states of the "world."

To allow for deontic expressions, we extend the signature Σ with sets of *time points* $\mathcal{T}$ and *time variables* $\mathcal{V}_t$, resulting in the deontic signature $\Sigma_d = \langle \mathcal{P}, \mathcal{C} \cup \mathcal{T}, \mathcal{V} \cup \mathcal{V}_t \rangle$. From now on, we assume an arbitrary but fixed deontic signature Σ_d.

Both additions to the signature are related to time, and we explain them in the following. The set of time points $\mathcal{T}$ represents the time domain, and we assume that $\mathcal{T}$ is a countable subset of non-negative real numbers, including 0, such as natural numbers $\mathbb{N}$. The set of time variables $\mathcal{V}_t$ relates specifically to $\mathcal{T}$ in the same standard way as $\mathcal{V}$ relates to $\mathcal{C}$, and we reserve a special time variable $\mathbf{now} \in \mathcal{V}_t$ which we always associate with the time point representing the current time.

Both $\mathcal{T}$ and $\mathcal{V}_t$ are used to define *time expressions*, which allow us to shift time points into the future by a specific time interval. The set of *time expressions* $\mathcal{T}^*$ is defined as $\mathcal{T}^* = \mathcal{T} \cup \{ V + c \mid V \in \mathcal{V}_t \wedge c \in \mathcal{T} \}$.[3] Where convenient, we simply abbreviate $V + 0$ by V.

We now introduce *deontic literals* to represent three types of obligations: 1. *obligations to-do*, requiring that an action be executed; 2. *obligations to-achieve*, requiring that a physical literal become true; 3. *obligations to-maintain*, requiring that a physical literal remain true; all three strictly before a specified deadline.[4]

Definition 1 (Deontic literal)**.** Let $A \subseteq \mathcal{A}$ be a non-empty action, l a physical literal, and $t \in \mathcal{T}^*$, called the *deadline*. An *obligation* is of the following three forms:

- *obligation to-do* $\mathsf{O}^{\mathsf{d}}_t\, A$;
- *obligation to-achieve* $\mathsf{O}^{\mathsf{a}}_t\, l$;
- *obligation to-maintain* $\mathsf{O}^{\mathsf{m}}_t\, l$.

Obligations and their negations form the set of *deontic literals*. The expression $\mathsf{O}_t\, l$ represents both $\mathsf{O}^{\mathsf{a}}_t\, l$ and $\mathsf{O}^{\mathsf{m}}_t\, l$.

Note that obligations to-achieve and to-maintain can be understood as dual: the intention for the former is to require that literal l holds for (at least) one time point strictly in between the introduction of the obligation and its deadline, and for the latter that l holds for all time points from the time point of the introduction until (immediately before) the deadline. This will be accordingly reflected in the semantics later.

A *literal* is either a physical literal or a deontic literal. A literal is *ground* if it contains no variables. Literals f and $\neg f$ are called *complementary*. The literal complementary to l is denoted by $\bar{l}$.

In Action Languages, only actions can cause state changes, but the introduction of obligations with deadlines should allow a state change to be triggered also by the violation (V) or the satisfaction (S) of obligations, resulting from the expiration of their deadlines. Deontic events accommodate for that.

Definition 2 (Event)**.** Let d be an obligation. *Deontic events* are expressions of the form $\mathsf{V}d$ and $\mathsf{S}d$. An *event* is an action or a deontic event.

We recall propositions in $\mathcal{A}$ and at the same time extend them to include norms.

Definition 3 (Norm and normative specification)**.** Let e be an event, l a deontic or physical literal and, for all i with $1 \leq i \leq n$, l_i literals. A *proposition* n takes the following form:

$$e \textbf{ causes } l \textbf{ if } l_1, \ldots, l_n \ . \tag{1}$$

[2] In [Gelfond and Lifschitz, 1998], only the propositional case is considered. We use function-free first-order atoms here to ease the presentation of our formalization of time.

[3] Here we abuse the set of time points and time variables to also represent *time intervals*. The expression $V + c$ always represents the addition of a time interval to a time point, or of two time intervals.

[4] The restriction to non-inclusive deadlines is an arbitrary decision and it would be reasonable to consider inclusive deadlines instead, or even to introduce both types of deadlines. For simplicity, we consider only non-inclusive deadlines.

We say that e is the *event* of n, l the *effect* of n and $l_1, \dots, l_n$ its *condition*. If the condition is empty, we write (e **causes** l). If l is a deontic literal, then n is a *norm*. If l is a physical literal, then e is an action, and all l_i in the condition are physical literals.

A proposition is *safe* if every variable (different from **now**) appearing in its effect also appears in its event or in an obligation within its condition. A *normative specification* $\mathcal{N}$ is a finite set of safe propositions of the form (1).

Intuitively, a norm of the form (1) adds or removes the obligation specified by l if the *event* occurs and the *condition* is satisfied. Also note that a proposition with physical literal l matches a proposition in $\mathcal{A}$ [Gelfond and Lifschitz, 1998] and the rationale for the applied restrictions is that normative information should not affect the physical world. This is indeed the case and in line with the idea that obligations are meant to represent only guidelines of desired behavior for an agent (including penalties for non-compliance), unlike the line of work in which obligations can be used to prohibit the execution of an action (see, e.g., [Cholvy, 1999]). Finally, safeness of variables occurring in l prevents from the specification of propositions with non-ground effects.[5]

Example 4. Consider a set of norms in a university library scenario:

$$\mathit{borrow}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{\mathbf{now}+4}\, \mathit{ret}(X) \textbf{ if } \mathit{ugrad} \tag{2}$$

$$\mathit{borrow}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{\mathbf{now}+12}\, \mathit{ret}(X) \textbf{ if } \mathit{grad} \tag{3}$$

$$\mathit{renew}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{T+4}\, \mathit{ret}(X) \textbf{ if } \mathsf{O}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \tag{4}$$

$$\mathit{renew}(X) \textbf{ causes } \neg\mathsf{O}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \textbf{ if } \mathsf{O}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \tag{5}$$

$$\mathsf{VO}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{\mathbf{now}+1}\, \mathit{pay} \tag{6}$$

$$\mathsf{VO}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{\mathbf{now}+1}\, \mathit{ret}(X) \textbf{ if } \mathit{ugrad} \tag{7}$$

$$\mathsf{VO}^{\mathsf{d}}_{T}\, \mathit{ret}(X) \textbf{ causes } \mathsf{O}^{\mathsf{d}}_{\mathbf{now}+3}\, \mathit{ret}(X) \textbf{ if } \mathit{grad} \tag{8}$$

Norms (2) and (3) specify that borrowing a book creates the obligation to return that book within the period specified depending on the student's status (4 and 12 weeks for undergraduate and graduate students respectively). A book may be renewed for 4 more weeks, which means updating the obligation with the new deadline (4–5). Finally, a contrary-to-duty norm specifies that, if a user fails to return the book on time, a fine has to be paid within one week (6) and the book has to be returned (7–8).

On different domains, an example of a norm with an achievement obligation is that one has the obligation to achieve 30 credits within the academic year, and an example of a norm with a maintenance obligation is that one has the obligation to maintain a contract with a mobile carrier for (at least) 24 months.

$$\mathit{enterAcademicYear} \textbf{ causes } \mathsf{O}^{\mathsf{a}}_{\mathbf{now}+12}\, \mathit{sumCredits}(30)$$

$$\mathit{startMobileContract} \textbf{ causes } \mathsf{O}^{\mathsf{m}}_{\mathbf{now}+24}\, \mathit{mobileContract}$$

[5] A less restrictive condition could be applied to propositions whose effect is a physical literal, but this would only affect the action language which is not our major concern.

2.2 Semantics

The semantics of Action Language $\mathcal{A}$ is defined as a transition system T modeling the physical environment. A node σ of T represents a possible physical state and a transition $\langle \sigma, A, \sigma' \rangle$ represents that state σ' can be reached from σ by executing action A. We extend such T with deontic features and time to define the semantics of normative specifications as follows. We augment states σ with deontic states δ and we define when literals are satisfied in such a combined state σ/δ. Next, we present a relation that captures how a deontic event is caused by either reaching a deadline or due to an executed action. We proceed by specifying which positive and negative normative effects are triggered in a state σ/δ when executing an action A at time t w.r.t. $\mathcal{N}$, using a Boolean function $\rho_{A,t}()$ in the former case to avoid introducing meaningless obligations. This enables us to define a resulting new deontic state and, subsequently, transitions and paths in the resulting transition system $T_{\mathcal{N}}$.

Let $\mathcal{N}$ be a normative specification. The states of the transition system $T_{\mathcal{N}}$ consist of two parts: a set of physical literals representing the physical "world," and a set of obligations representing the deontic state of the agent. Additionally, we require that obligations be part of the state only if they are not immediately satisfied or violated.

Definition 5 (State of $T_{\mathcal{N}}$)**.** Let σ be a complete and consistent set of ground physical literals, i.e., for each ground physical fluent f, exactly one of f and $\neg f$ belongs to σ, and δ a finite set of ground obligations. Then, σ/δ is a *state* of the transition system $T_{\mathcal{N}}$ if the following conditions are satisfied for every physical literal l and deadline t: ($\mathsf{O}^{\mathsf{a}}_t\, l \notin \delta$ or $l \notin \sigma$) and ($\mathsf{O}^{\mathsf{m}}_t\, l \notin \delta$ or $l \in \sigma$). We call σ the *physical state* and δ the *deontic state*.

Note that, unlike σ, δ is not complete, since it would be impractical to require that d or $\neg d$ occur in δ for each d due to the usually infinite set of time points $\mathcal{T}$. This is also why we consider a separate set δ and do not merge the two parts into one.

To deal with the satisfaction of non-ground literals in a state σ/δ, we introduce a *variable assignment* z as a function mapping variables to constants ($\mathcal{V} \to \mathcal{C}$) and time variables to time points ($\mathcal{V}_t \to \mathcal{T}$). For every time point t, we denote the set of variable assignments z such that $\mathsf{z}(\mathbf{now}) = t$ by $\mathcal{Z}_t$. Hence, the index t in $\mathcal{Z}_t$ is not merely notation, but defines the value that is assigned to **now**.

For any literal or event λ, we denote by $\lambda|_{\mathsf{z}}$ the literal or event obtained from λ by substituting every variable according to z, and, subsequently, replacing every time expression $t + c$ with the time point t' such that $t' = t + c$. E.g., $\mathsf{O}^{\mathsf{d}}_9\, \mathit{ret}(\mathit{book})$ is the result of $(\mathsf{O}^{\mathsf{d}}_{\mathbf{now}+4}\, \mathit{ret}(X))|_{\{X \to \mathit{book},\, \mathbf{now} \to 5\}}$.

Satisfaction for ground literals in a state σ/δ is defined as follows for a physical literal l and a ground obligation d:

$$\sigma/\delta \models l \text{ iff } l \in \sigma \ ,$$
$$\sigma/\delta \models d \text{ iff } d \in \delta \ ,$$
$$\sigma/\delta \models \neg d \text{ iff } d \notin \delta \ .$$

Furthermore, given a variable assignment z, and a set of literals $L = \{l_1, \ldots, l_n\}$, we define $\sigma/\delta \models L|_{\mathsf{z}}$ iff for all i with $i \in \{1, \ldots, n\}$, $\sigma/\delta \models l_i|_{\mathsf{z}}$.

Each transition of $T_{\mathcal{N}}$ is a tuple $\langle \sigma/\delta, (A, t), \sigma'/\delta' \rangle$, where A is a ground action and $t \in \mathcal{T}$ a time point, meaning that A occurred at time t, causing the transition from state σ/δ to σ'/δ'. Since the physical effects are independent of the deontic ones, we first define a relation $R_{\mathcal{N}}$ that, for a given $\mathcal{N}$, associates each physical state σ and ground action A with a new physical state σ':

$$\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}} \text{ iff } \sigma' = (\sigma \cup \mathsf{E}_A(\sigma)) \setminus \{\, \bar{l} \mid l \in \mathsf{E}_A(\sigma) \,\} \quad ,$$

where $\mathsf{E}_A(\sigma)$ stands for the set of all physical literals $l|_{\mathsf{z}}$ such that $(e$ **causes** l **if** $C) \in \mathcal{N}$ and there is $\mathsf{z} \in \mathcal{Z}_t$ with $\sigma/\delta \models C|_{\mathsf{z}}$ and $e|_{\mathsf{z}} \subseteq A$. If $A = \emptyset$, then $\sigma' = \sigma$, which allows us to handle deontic updates resulting from deadline expirations at time points in which no action occurs. Note that the requirement that σ' be a physical state ensures that $\langle \sigma, A, \sigma' \rangle \notin R_{\mathcal{N}}$ if A has contradictory effects in state σ.

We proceed by specifying how to obtain a new deontic state. First, we define the conditions for the occurrence of deontic events, which are satisfactions/violations of obligations occurring in the current deontic state δ w.r.t. the new physical state σ'.

Definition 6 (Occurrence of deontic event)**.** Let σ/δ be a state of $T_{\mathcal{N}}$, A, B ground actions, $\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}}$ and t, t' time points. The occurrence relation for ground deontic events under action A at time t, $\vdash_{A,t}$, is defined for tuples $\langle \delta, \sigma' \rangle$ as follows:

$$\begin{array}{lll}
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{VO}^{\mathsf{d}}_{t'} B & \text{iff} & \mathsf{O}^{\mathsf{d}}_{t'} B \in \delta \wedge t \geq t' \\
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{VO}^{\mathsf{a}}_{t'} l & \text{iff} & \mathsf{O}^{\mathsf{a}}_{t'} l \in \delta \wedge t \geq t' \\
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{SO}^{\mathsf{m}}_{t'} l & \text{iff} & \mathsf{O}^{\mathsf{m}}_{t'} l \in \delta \wedge t \geq t' \\
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{SO}^{\mathsf{d}}_{t'} B & \text{iff} & \mathsf{O}^{\mathsf{d}}_{t'} B \in \delta \wedge t < t' \wedge B \subseteq A \\
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{SO}^{\mathsf{a}}_{t'} l & \text{iff} & \mathsf{O}^{\mathsf{a}}_{t'} l \in \delta \wedge t < t' \wedge l \in \sigma' \\
\langle \delta, \sigma' \rangle \vdash_{A,t} \mathsf{VO}^{\mathsf{m}}_{t'} l & \text{iff} & \mathsf{O}^{\mathsf{m}}_{t'} l \in \delta \wedge t < t' \wedge \bar{l} \in \sigma'
\end{array}$$

Additionally, $\epsilon_{A,t}(\delta, \sigma') = \{\, e \mid \langle \delta, \sigma' \rangle \vdash_{A,t} e \,\}$.

The above conditions encode the dynamics of violations and satisfactions, and depend on the type of obligation involved. The first three represent events generated by a deadline expiration. The last three represent events that occur before the expiration of the respective deadline. Namely, either action B is executed (as part of A) at time t, or a state change affects the literal l to be achieved (or cease to be maintained). We explain the latter case in more detail for an obligation to-achieve l. Such an obligation can only be part of a state σ/δ if $\bar{l} \in \sigma$. If executing action A at time t introduces l, i.e., adds it to the new state σ' (and removes $\bar{l}$), then an event occurs, which (as we will see below) is used to trigger the removal of the corresponding obligation, but also possibly the introduction of new obligations.

Before defining the normative effects of executing action A at time t in state σ/δ, we need to introduce an auxiliary function $\rho_{A,t}(d, \sigma')$ that determines whether, given σ' with $\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}}$, an obligation d, which would be introduced to the new deontic state, is *relevant*: $\rho_{A,t}(d, \sigma') = \bot$ if either (1) d is an obligation with deadline $t' \leq t$, (2) $d = \mathsf{O}^{\mathsf{d}}_{t'} B \wedge B \subseteq A$, (3) $d = \mathsf{O}^{\mathsf{a}}_{t'} l \wedge l \in \sigma'$, or (4) $d = \mathsf{O}^{\mathsf{m}}_{t'} l \wedge \bar{l} \in \sigma'$; otherwise $\rho_{A,t}(d, \sigma') = \top$. Condition (1) matches the first part of Def. 6, while (2-4) matches the second. We thus avoid the introduction of obligations that would be satisfied/violated immediately, following the rationale to only consider obligations whose satisfaction can be influenced by the agent's behavior.

We now define the normative effects of executing an action A at time t in a given state σ/δ. We say that an effect of a norm is positive (negative) if it is an obligation (its negation). For each instance of a norm in $\mathcal{N}$ we need to evaluate its condition in σ/δ, check whether the respective event is a subset of action A or a deontic event, and, in case of the positive effects, check if the effect of the norm is an obligation that is relevant (or can be safely ignored). The latter and the check for deontic events occur w.r.t. the new physical state σ' (obtained by executing A on σ) as already indicated.

Definition 7 (Normative effect)**.** Let σ/δ be a state of $T_{\mathcal{N}}$, $\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}}$, t a time point and d an obligation. The set of *positive normative effects* $\mathsf{E}^+_{A,t}(\sigma/\delta, \sigma')$ and the set of *negative normative effects* $\mathsf{E}^-_{A,t}(\sigma/\delta, \sigma')$ are defined as follows:

$$\begin{aligned}
\mathsf{E}^+_{A,t}(\sigma/\delta, \sigma') = \{\, (d|_{\mathsf{z}}) \mid (e \textbf{ causes } d \textbf{ if } C) \in \mathcal{N} \wedge \exists \mathsf{z} \in \mathcal{Z}_t : \\
\sigma/\delta \models C|_{\mathsf{z}} \wedge (e|_{\mathsf{z}} \subseteq A \vee \langle \delta, \sigma' \rangle \vdash_{A,t} e|_{\mathsf{z}}) \\
\wedge\, \rho_{A,t}(d|_{\mathsf{z}}, \sigma') \,\}; \\
\mathsf{E}^-_{A,t}(\sigma/\delta, \sigma') = \{\, (d|_{\mathsf{z}}) \mid (e \textbf{ causes } \neg d \textbf{ if } C) \in \mathcal{N} \wedge \exists \mathsf{z} \in \mathcal{Z}_t : \\
\sigma/\delta \models C|_{\mathsf{z}} \wedge (e|_{\mathsf{z}} \subseteq A \vee \langle \delta, \sigma' \rangle \vdash_{A,t} e|_{\mathsf{z}}) \,\}.
\end{aligned}$$

The new deontic state δ' can now be computed from σ/δ by first detecting which deontic events occur (and removing the corresponding obligations), then adding the positive effects of these events and finally removing their negative effects.

Definition 8 (New deontic state)**.** Let σ/δ be a state of $T_{\mathcal{N}}$, $\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}}$, t a time point and d an obligation. We define $\mathsf{G}(\mathsf{V}d) = \mathsf{G}(\mathsf{S}d) = d$, for any set of deontic events E, $\mathsf{G}(E) = \{\, \mathsf{G}(e) \mid e \in E \,\}$ and the new deontic state

$$\delta' = \Big[(\delta \setminus \mathsf{G}(\epsilon_{A,t}(\delta, \sigma'))) \cup \mathsf{E}^+_{A,t}(\sigma/\delta, \sigma') \Big] \setminus \mathsf{E}^-_{A,t}(\sigma/\delta, \sigma').$$

Three consequences follow immediately: first, if an obligation is introduced and removed simultaneously by different norms, then the removal prevails, following a generalization of the *in dubio pro reo* principle; second, it may happen that the occurrence of a deontic event removes some obligation, which is immediately re-introduced in $\mathsf{E}^+_{A,t}()$ if a corresponding norm exists, such as for example if you pay a fine and, at the same time, commit an offense that incurs in the same penalty; and third, the frame problem for obligations is trivially solved in this equation—whatever appears in δ and is not removed on purpose, persists in δ'.

We show that σ' and δ' indeed form a state of $T_{\mathcal{N}}$.

Proposition 9. *Let σ/δ be a state of $T_{\mathcal{N}}$, $\langle \sigma, A, \sigma' \rangle \in R_{\mathcal{N}}$, and δ' as defined in Def. 8. Then σ'/δ' is a state of $T_{\mathcal{N}}$.*

Furthermore, considering the definition of deontic events, whenever a deadline of an existing obligation is reached, a deontic event always takes place. A consequence of this observation is that a transition from σ/δ must not occur at a time point that exceeds the deadline of some obligation in δ. We define this time point as the earliest deadline among the current obligations, or infinity if there are no obligations in δ. Formally, let $d(\delta) = \{\, t \in \mathcal{T} \mid \mathsf{O}_t\, l \in \delta \text{ or } \mathsf{O}^{\mathsf{d}}_t\, B \in \delta \,\}$. Then, $\mathsf{ltp}(\delta) = \min(d(\delta))$ if $d(\delta) \neq \emptyset$ and $\mathsf{ltp}(\delta) = \infty$ if $d(\delta) = \emptyset$. Note that, since δ is assumed finite, this notion of least time point is well-defined, i.e., if $d(\delta) \neq \emptyset$, then $\mathsf{ltp}(\delta) \in d(\delta)$, which, along with Proposition 9, allows us to define transitions of $T_{\mathcal{N}}$:

Definition 10 (Transition)**.** A *transition of* $T_{\mathcal{N}}$ is a tuple $\langle \sigma/\delta, (A, t), \sigma'/\delta' \rangle$ where σ, δ, σ', δ' and A are as in Def. 8, t is a time point s.t. $t \leq \mathsf{ltp}(\delta)$ and if $A = \emptyset$, then $t = \mathsf{ltp}(\delta)$.

Example 11. The following are transitions of $T_{\mathcal{N}}$ for Example 4 in Sect. 2.1.

$\langle \{ugrad\}/\emptyset, (borrow(b), 1), \{ugrad\}/\{\mathsf{O}^{\mathsf{d}}_5\, ret(b)\} \rangle$

$\langle \{ugrad\}/\{\mathsf{O}^{\mathsf{d}}_5\, ret(b)\}, (ret(b), 4), \{ugrad\}/\emptyset \rangle$

$\langle \{ugrad\}/\{\mathsf{O}^{\mathsf{d}}_5\, ret(b)\}, (\emptyset, 5), \{ugrad\}/\{\mathsf{O}^{\mathsf{d}}_6\, pay, \mathsf{O}^{\mathsf{d}}_6\, ret(b)\} \rangle$.

The tuple $\langle \{ugrad\}/\{\mathsf{O}^{\mathsf{d}}_5\, ret(b)\}, (ret(b), 8), \{ugrad\}/\emptyset \rangle$ is not a transition because $\mathsf{ltp}(\{\mathsf{O}^{\mathsf{d}}_5\, ret(b)\}) = 5 \not\geq 8$.

We can show that the transition system $T_{\mathcal{N}}$ is deterministic.

Proposition 12. *$T_{\mathcal{N}}$ is deterministic, i.e., if $\langle \sigma/\delta, (A, t), \sigma'/\delta' \rangle$ and $\langle \sigma/\delta, (A, t), \sigma''/\delta'' \rangle$ are transitions of $T_{\mathcal{N}}$, then $\sigma'/\delta' = \sigma''/\delta''$.*

Now, a path is an alternating sequence of states in $T_{\mathcal{N}}$ and pairs (A, t) corresponding to the transitions of $T_{\mathcal{N}}$.

Definition 13 (Path)**.** A *path* is a sequence of the form

$$\sigma_0/\delta_0, (A_1, t_1), \sigma_1/\delta_1, \ldots, (A_n, t_n), \sigma_n/\delta_n \quad , \qquad (9)$$

where σ_j/δ_j is a state of $T_{\mathcal{N}}$ for every $0 \leq j \leq n$, $\langle \sigma_j/\delta_j, (A_{j+1}, t_{j+1}), \sigma_{j+1}/\delta_{j+1} \rangle$ is a transition of $T_{\mathcal{N}}$ for every $0 \leq j < n$, and $t_j < t_{j+1}$ for every $1 \leq j < n$.

The last condition states the assumption that the time points in a path are ordered.

The satisfaction of an obligation to-do or to-achieve and the violation of an obligation to-maintain always indicate some relevant change w.r.t. the previous state.

Proposition 14. *Let P be a path of the form (9).*

If $\langle \delta_{j-1}, \sigma_j \rangle \vdash_{A_j, t_j} \mathsf{SO}^{\mathsf{d}}_t\, B$, then $B \not\subseteq A_{j-1}$ and $B \subseteq A_j$;

if $\langle \delta_{j-1}, \sigma_j \rangle \vdash_{A_j, t_j} \mathsf{SO}^{\mathsf{a}}_t\, l$, then $l \notin \sigma_{j-1}$ and $l \in \sigma_j$;

if $\langle \delta_{j-1}, \sigma_j \rangle \vdash_{A_j, t_j} \mathsf{VO}^{\mathsf{m}}_t\, l$, then $l \in \sigma_{j-1}$ and $l \notin \sigma_j$.

A symmetric result for the other three deontic events does not hold, simply because these occur due to a deadline that is reached with the progress of time.

3 Query Language and Equivalence

We now define a query language for $\mathcal{A}_{\mathcal{N}}$ that can be used to check whether a certain literal/event occurs in a specific time interval given a normative specification and a description of the initial state. We consider decidability and complexity of answering queries. Then, we also discuss equivalence between different normative specifications.

3.1 Syntax of the Query Language

A query language in the case of action languages usually consists of statements describing initial conditions and statements to query the domain description w.r.t. these initial conditions. We adapt the notion of axioms for our purpose.

Definition 15 (Axiom)**.** Let $\mathcal{N}$ be a normative specification and l a ground physical literal or a ground obligation. An *axiom* is of the form **initially** l. Given a set of axioms Γ, a physical state σ in $T_{\mathcal{N}}$ *satisfies* Γ if, for every physical literal l, (**initially** l) $\in \Gamma$ implies $l \in \sigma$.

Let δ be the set of obligations d such that (**initially** d) $\in \Gamma$. A set of axioms Γ is an *initial specification for* $\mathcal{N}$ if, for every physical state σ that satisfies Γ, σ/δ forms a state of $T_{\mathcal{N}}$. Such states σ/δ are called *initial w.r.t.* Γ.

We thus specify that an initial specification for $\mathcal{N}$ aligns with Def. 5, i.e., if Γ contains an axiom for an obligation to achieve (maintain) l, then it must also contain an axiom for $\neg l$ (l). Note that a set of axioms may not *fully specify* the physical state σ, i.e., there may be several states σ that satisfy Γ, hence several initial states.

An *action sequence* is a finite sequence $((A_1, t_1), \ldots, (A_k, t_k))$ such that, for all i with $1 \leq i \leq k$, A_i is a non-empty action, and $t_1, \ldots, t_k \in \mathcal{T}$ with $0 < t_1 < \cdots < t_k$. Given an action sequence, queries are defined as follows:

Definition 16 (Query)**.** Let l be a deontic literal, a deontic event—both without any occurrence of **now**—or a physical literal, $t_\alpha, t_\beta \in \mathcal{T}$ with $0 \leq t_\alpha \leq t_\beta$, and S an action sequence. A *query* is of the form $l : [t_\alpha, t_\beta] : S$.

Note that even though our query language is quite simple, it is rather versatile and allows for expressive queries due to the usage of variables in queries. Not only may we query for non-ground fluents occurring in a certain time interval, such as whether a user had some book in her possession, but also whether there occurred any obligation or violation in a given time interval without having to specify the deadline.

3.2 Semantics of the Query Language

The semantics of the query language is defined w.r.t. paths of the transition system $T_{\mathcal{N}}$. First, we establish that a path P of the form (9) *satisfies* an initial specification Γ for $\mathcal{N}$ if σ_0/δ_0 is an initial state relative to Γ. The idea is to restrict the paths considered to answer a query to those which match the initial specification.

Next, we link the action sequence in a query to a path by matching each pair (A_i, t_i) in the sequence to exactly one in the path. All other actions in the path have to be empty, i.e., they occur due to deontic events.

Definition 17 (Satisfiability of an Action Sequence)**.** Let S be an action sequence $(A'_1, t'_1), \ldots, (A'_k, t'_k)$ and P a path of the form (9). *P satisfies S* if there is an injective mapping $\mu : \{1, \ldots, k\} \mapsto \{1, \ldots, n\}$ (from S to P) such that

1. for each i with $1 \leq i \leq k$, $A'_i = A_{\mu(i)}$ and $t'_i = t_{\mu(i)}$,
2. for each j with $1 \leq j \leq n$, if $\mu(i) \neq j$ for all i with $1 \leq i \leq k$, then $A_j = \emptyset$.

Given the definition of action sequences and paths, if such an injective mapping μ exists, then it is clearly unique, and so is the path corresponding to an action sequence for a fixed initial state.

To evaluate whether a certain literal or event holds while executing a sequence of actions, we need to collect all states that fall into the time interval $[t_\alpha, t_\beta]$ given in the query. That is, we collect the state at t_α and all the states inside the interval, or alternatively the final state in the path if the last transition occurs before t_α. In the former case, if there is no action occurring precisely at t_α, then we have to consider the state prior to t_α, because that is then the current state at t_α. Formally, given a path P of the form (9) and time points $t_\alpha \leq t_\beta$, we define the set $s(P, [t_\alpha, t_\beta]) = \{\sigma_i/\delta_i \mid t_i < t_\alpha < t_{i+1}\} \cup \{\sigma_i/\delta_i \mid t_\alpha \leq t_i \leq t_\beta\} \cup \{\sigma_n/\delta_n \mid t_n < t_\alpha\}$. Additionally, we want to ensure that only those paths are considered that cover the entire interval so that we do not miss any states. Therefore, we define that path P *reaches* time point t if either $t_n \geq t$ or $\mathsf{ltp}(\delta_n) = \infty$.

Finally, we can define how queries are evaluated.

Definition 18 (Query satisfaction)**.** Let Q be a query of the form $l : [t_\alpha, t_\beta] : S$, $\mathcal{N}$ a normative specification and Γ an initial specification for $\mathcal{N}$. *Q is a consequence of Γ w.r.t. $\mathcal{N}$*, denoted by $\Gamma \models_{\mathcal{N}} Q$, if, for every path P that satisfies Γ and S and that reaches t_β, there exists a variable assignment z such that one of these conditions holds:

(a) for some $\sigma/\delta \in s(P, [t_\alpha, t_\beta])$, $\sigma/\delta \models l|_{\mathsf{z}}$ if l is a literal;

(b) for some j with $t_\alpha \leq t_j \leq t_\beta$, $\langle \delta_{j-1}, \sigma_j \rangle \vdash_{A_j, t_j} e|_{\mathsf{z}}$ if l is a deontic event.

Note that our definition of query satisfaction implies that if the action sequence is not executable, then the query holds automatically for all paths in the transition system satisfying the conditions, simply because there are none. That is related to the question of consistent action descriptions [Zhang *et al.*, 2002] and also implicit domain constraints [Herzig and Varzinczak, 2007; Thielscher, 2011], and we refer to the literature for ways to avoid such problems.

Example 19. Recall Example 4 and $\Gamma = \{\textbf{initially}\ \mathit{ugrad}\}$:

$$Q_1 = \mathsf{VO}^{\mathsf{d}}_X\, \mathit{ret}(b) : [1, 8] : \langle(\mathit{borrow}(b) : 1), (\mathit{ret}(b) : 4)\rangle;$$

$$Q_2 = \mathsf{O}^{\mathsf{d}}_5\, \mathit{ret}(Y) : [0, 4] : \langle(\mathit{borrow}(b) : 1)\rangle;$$

$$Q_3 = \mathit{ugrad} : [0, 9] : \langle(\mathit{borrow}(b) : 1), (\mathit{ret}(b) : 4)\rangle.$$

We obtain that $\Gamma \not\models_{\mathcal{N}} Q_1$, but $\Gamma \models_{\mathcal{N}} Q_2$ and $\Gamma \models_{\mathcal{N}} Q_3$.

We analyze decidability and computational complexity of answering queries where we measure the input in the size of the set of axioms Γ.

Theorem 20. *Let Q be a query, $\mathcal{N}$ a normative specification and Γ an initial specification for $\mathcal{N}$. If the physical states in $T_{\mathcal{N}}$ are finite, then answering $\Gamma \models_{\mathcal{N}} Q$ is decidable in* coNP. *If Γ additionally fully specifies σ, then answering $\Gamma \models_{\mathcal{N}} Q$ is in* P.

3.3 Equivalence

Equivalence is an important problem in the area of normative systems. It can be used, for example, for simplifying normative systems, which usually tend to have redundant norms. In our approach, we define equivalence of normative specifications w.r.t. the answers they provide to queries.

Definition 21 (Equivalence)**.** We say that normative specifications $\mathcal{N}_1, \mathcal{N}_2$ are *equivalent* if for every set of axioms Γ and every query Q, $\Gamma \models_{\mathcal{N}_1} Q$ if and only if $\Gamma \models_{\mathcal{N}_2} Q$.

We can show that two normative specifications being equivalent is the same as them having the same transition system.

Theorem 22. *The following conditions are equivalent for any normative specifications $\mathcal{N}_1$, $\mathcal{N}_2$:*

1) $\mathcal{N}_1$, $\mathcal{N}_2$ are equivalent.

2) $T_{\mathcal{N}_1} = T_{\mathcal{N}_2}$.

3) The sets of paths of $T_{\mathcal{N}_1}$ and of $T_{\mathcal{N}_2}$ coincide.

A stronger notion of equivalence requires equivalence in the presence of additional norms, important when modularly analyzing subsets of norms of a larger system. Two strongly equivalent subsets of a normative specification can be safely replaced by one another.

Definition 23 (Strong equivalence)**.** We say that normative specifications $\mathcal{N}_1, \mathcal{N}_2$ are *strongly equivalent* if for every normative specification $\mathcal{N}$, $\mathcal{N}_1 \cup \mathcal{N}$ is equivalent to $\mathcal{N}_2 \cup \mathcal{N}$.

Strong equivalence implies equivalence but not vice-versa.

Theorem 24. *Let $\mathcal{N}_1$, $\mathcal{N}_2$ be normative specifications. If $\mathcal{N}_1$ is strongly equivalent to $\mathcal{N}_2$, then $\mathcal{N}_1$ is also equivalent to $\mathcal{N}_2$, but the converse implication does not hold.*

4 Conclusions

We have extended Action Language $\mathcal{A}$ with features that allow reasoning about norms, time and deadlines in dynamic domains. We have shown how our language can be used to express norms involving obligations with deadlines that explicitly refer to time and actions, including obligations to-do, to-achieve and to-maintain but also contrary-to-duty situations, which previous action languages and their extensions to norms did not cover. We have defined a semantics for this language and a query language along with its semantics. Moreover, we studied the complexity and equivalence of normative specifications.

Notably, our framework may serve as a basis for introducing norms to other AI action formalisms where norms with explicit time deadlines and contrary-to-duty obligations have received little consideration so far. Interesting examples include the Event Calculus [Kowalski and Sergot, 1986], the Situation Calculus [Reiter, 1991], the Fluent Calculus [Thielscher, 1999] and extensions of Dynamic Logic [Harel, 1979] that have a solution to the frame problem [Zhang and Foo, 2001; Zhang and Foo, 2002; Castilho *et al.*, 2002; Demolombe *et al.*, 2003].

Our query language can be used to define interesting planning problems, such as finding plans which prevent violations, or whose violations are within certain limits. Additionally, our language has important applicability in the development of electronic institutions. Electronic institutions are virtual entities that maintain, promote and enforce a set of norms. They observe agent's actions to determine norm violations (resp. satisfactions), e.g., to enforce sanctions (resp. give rewards). Given its formal semantics, and its strong links to dynamic systems, $\mathcal{A}_{\mathcal{N}}$ can be used as the language to specify and disseminate the norms and the query language used to determine violations and satisfactions.

Related work on normative systems resulted in frameworks that combine obligations and time. The proposals in [Dignum and Kuiper, 1997; Dignum and Kuiper, 1998; Broersen and Brunel, 2007; Balbiani *et al.*, 2009], which combine dynamic, deontic and temporal logic, have a rich language, but they have difficulties in dealing with the frame problem, relevant in the propagation of obligations that have not been fulfilled yet [Broersen and Brunel, 2007], and with dealing with contrary-to-duty obligations. Also, no axiomatization exists for the proposals in [Dignum and Kuiper, 1997; Dignum and Kuiper, 1998], and hence automatic reasoning is not possible, while the approaches in [Broersen and Brunel, 2007; Balbiani *et al.*, 2009] do not deal with actions. In [Ågotnes *et al.*, 2010], robustness of normative systems is studied building on temporal logic, but neither deadlines nor contrary-to-duty obligations are considered. The work in [Governatori and Rotolo, 2011] aims at studying the dynamics of normative violations. However, without an explicit representation of actions, they cannot properly deal with obligations to-do, nor integrate the normative part of the system with the dynamics resulting from the execution of actions provided by Action Languages. Finally, in [Dastani *et al.*, 2012] the focus is set on an operational semantics to be able to modify a normative system during runtime. Yet, there are no time deadlines. Instead, deadlines are state conditions, which may be an interesting extension of our work, but does not cover the expressiveness provided by our formalism.

Our work opens several interesting paths for future research. First of all, we would like to design an implementation. Of course, an encoding in ASP is always possible, but perhaps more efficient solutions exist. We would also like to extend the language with other deontic constructs such as *prohibition* and *permission*. We already have some notion of prohibition, since an obligation to-maintain $\neg l$ can be seen as a prohibition to bring about l, and some notion of permission, since the removal of an obligation to-maintain $\neg l$ can be seen as a weak permission to bring about l. On the other hand, the counterpart of obligations to-do, forbidden actions, has not been considered here. Accommodating forbidden actions would require a new normative fluent $\mathsf{F}_t\, a$ meaning that action a is forbidden until time t. Moreover, we may consider extending our framework to more expressive Action Languages, more complex deadlines, and actions with different durations.

Acknowledgments We would like to thank the anonymous reviewers whose comments helped to improve the paper.

Matthias Knorr, João Leite and Martin Slota were partially supported by Fundação para a Ciência e a Tecnologia under project "ERRO – Efficient Reasoning with Rules and Ontologies" (PTDC/EIA-CCO/121823/2010). Matthias Knorr was also partially supported by FCT Grant SFRH/BPD/86970/2012 and Ricardo Gonçalves by FCT Grant SFRH/BPD/47245/2008.

References

[Ågotnes *et al.*, 2010] Thomas Ågotnes, Wiebe van der Hoek, and Michael Wooldridge. Robust normative systems and a logic of norm compliance. *Logic Journal of the IGPL*, 18(1):4–30, 2010.

[Alechina *et al.*, 2012] Natasha Alechina, Mehdi Dastani, and Brian Logan. Programming norm-aware agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 2 of *AAMAS '12*, pages 1057–1064. IFAAMAS, 2012.

[Artikis *et al.*, 2009] Alexander Artikis, Marek Sergot, and Jeremy Pitt. Specifying norm-governed computational societies. *ACM Trans. Comput. Log.*, 10(1):1–42, 2009.

[Balbiani *et al.*, 2009] Philippe Balbiani, Jan Broersen, and Julien Brunel. Decision procedures for a deontic logic modeling temporal inheritance of obligations. *Electr. Notes Theor. Comput. Sci.*, 231:69–89, 2009.

[Boella and van der Torre, 2004] Guido Boella and Leendert W. N. van der Torre. Regulative and constitutive norms in normative multiagent systems. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pages 255–266. AAAI Press, 2004.

[Boella *et al.*, 2008] Guido Boella, Leendert van der Torre, and Harko Verhagen. Introduction to the special issue on normative multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 17(1):1–10, 2008.

[Broersen and Brunel, 2007] Jan Broersen and Julien Brunel. Preservation of obligations in a temporal and deontic framework. In Edmund Durfee, Makoto Yokoo, Michael Huhns, and Onn Shehory, editors, *Autonomous Agents and Multi-Agent Systems*, page 177, 2007.

[Bulling and Dastani, 2011] Nils Bulling and Mehdi Dastani. Verifying normative behaviour via normative mechanism design. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, volume 1 of *IJCAI'11*, pages 103–108. AAAI Press, 2011.

[Carmo and Jones, 2002] José Carmo and Andrew Jones. Deontic logic and contrary-to-duties. In Dov Gabbay and Franz Guenthner, editors, *Handbook of Philosophical Logic*, volume 8, pages 265–343. Kluwer Academic Publishers, Dordrecht, Holland, 2002.

[Castilho *et al.*, 2002] Marcos A. Castilho, Andreas Herzig, and Ivan José Varzinczak. It depends on the context! a decidable logic of actions and plans based on a ternary dependence relation. In Salem Benferhat and Enrico Giunchiglia, editors, *NMR*, pages 343–348, 2002.

[Cholvy, 1999] Laurence Cholvy. Checking regulation consistency by using SOL-resolution. In *ICAIL*, pages 73–79, 1999.

[Craven and Sergot, 2008] Robert Craven and Marek Sergot. Agent strands in the action language nC+. *J. Applied Logic*, 6(2):172–191, 2008.

[Dastani *et al.*, 2012] Mehdi Dastani, John-Jules Ch. Meyer, and Nick A. M. Tinnemeier. Programming norm change. *Journal of Applied Non-Classical Logics*, 22(1-2):151–180, 2012.

[Demolombe *et al.*, 2003] Robert Demolombe, Andreas Herzig, and Ivan José Varzinczak. Regression in modal logic. *Journal of Applied Non-Classical Logics*, 13(2):165–185, 2003.

[Dignum and Kuiper, 1997] Frank Dignum and Ruurd Kuiper. Combining dynamic deontic logic and temporal logic for the specification of deadlines. In *HICSS (5)*, pages 336–346, 1997.

[Dignum and Kuiper, 1998] Frank Dignum and Ruurd Kuiper. Obligations and dense time for specifying deadlines. In *HICSS (5)*, pages 186–195, 1998.

[Esteva *et al.*, 2001] Marc Esteva, Juan A. Rodríguez-Aguilar, Carles Sierra, Pere Garcia, and Josep Lluís Arcos. On the formal specifications of electronic institutions. In Frank Dignum and Carles Sierra, editors, *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer, 2001.

[Gelfond and Lifschitz, 1998] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[Gelfond and Lobo, 2008] Michael Gelfond and Jorge Lobo. Authorization and obligation policies in dynamic systems. In Maria de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2008.

[Giunchiglia *et al.*, 2004] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1):49–104, 2004.

[Governatori and Rotolo, 2011] Guido Governatori and Antonino Rotolo. Justice delayed is justice denied: Logics for a temporal account of reparations and legal compliance. In João Leite, Paolo Torroni, Thomas Ågotnes, Guido Boella, and Leon van der Torre, editors, *CLIMA*, volume 6814 of *Lecture Notes in Computer Science*, pages 364–382. Springer, 2011.

[Harel, 1979] David Harel. *First-Order Dynamic Logic*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1979.

[Herzig and Varzinczak, 2007] Andreas Herzig and Ivan José Varzinczak. Metatheory of actions: Beyond consistency. *Artif. Intell.*, 171(16-17):951–984, 2007.

[Kowalski and Sergot, 1986] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4(1):67–95, 1986.

[Reiter, 1991] Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[Sadiq *et al.*, 2007] Shazia Wasim Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2007.

[Thielscher, 1999] Michael Thielscher. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial intelligence*, 111(1):277–299, 1999.

[Thielscher, 2011] Michael Thielscher. A unifying action calculus. *Artif. Intell.*, 175(1):120–141, 2011.

[Zhang and Foo, 2001] Dongmo Zhang and Norman Y. Foo. EPDL: A logic for causal reasoning. In Bernhard Nebel, editor, *IJCAI*, pages 131–138. Morgan Kaufmann, 2001.

[Zhang and Foo, 2002] Dongmo Zhang and Norman Y. Foo. Interpolation properties of action logic: Lazy-formalization to the frame problem. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 357–368. Springer, 2002.

[Zhang *et al.*, 2002] Dongmo Zhang, Samir Chopra, and Norman Y. Foo. Consistency of action descriptions. In Mitsuru Ishizuka and Abdul Sattar, editors, *PRICAI*, volume 2417 of *Lecture Notes in Computer Science*, pages 70–79. Springer, 2002.

Reasoning about Robot Epistemic Ability to Use the Cloud

David Rajaratnam[1], Hector J. Levesque[2], Maurice Pagnucco[1], Michael Thielscher[1]

[1]University of New South Wales
Sydney, Australia
{daver,morri,mit}@cse.unsw.edu.au

[2]University of Toronto
Toronto, Canada
hector@cs.toronto.edu

Abstract

Robots are often computationally constrained due to size and power limitations. To counter these limitations recent research has considered the use of off-board resources such as cloud computing facilities. These efforts typically focus on the cloud as a computational resource and data repository. However, to date, decisions about how and when cloud resources should be used must be scripted in advance. We argue that a robot should be able to make these decisions for itself in a dynamic way by reasoning introspectively about gaps in its own knowledge and ability. This paper develops formal techniques for this reasoning, providing a more principled approach to the utilisation of cloud computing resources.

1 Introduction

Mobile robot platforms often lack extensive computational resources in order to minimise their size and power usage. Consequently, access to off-board resources like those provided by cloud platforms can be a distinct advantage and has been the focus of recent research [Arumugam *et al.*, 2010; Kuffner, 2010; Guizzo, 2011; Waibel *et al.*, 2011]. These approaches typically focus on the cloud as a computational resource or as a repository for data that can be used by robots.

However, existing proposals provide for only a very limited use of cloud resources and services. In particular, they lack the ability to reason introspectively about the robot's abilities and knowledge. Such introspection is critical if the robot is to reason about gaps in its knowledge and abilities and therefore determine what external resources are required in order to achieve its objectives. *The aim of this paper is to advance this research by developing formal techniques for representing and reasoning about robot abilities and knowledge that can more fully utilise cloud computing resources.*

In the following section (Section 2) we briefly outline current developments in cloud robotics. In particular we highlight the RoboEarth project [Tenorth *et al.*, 2012] and some of the reasoning capabilities that it offers. We then highlight limitations with this model and argue that a more formal account of reasoning for cloud robotics is required (Section 3). In Section 4 we present background to the Situation Calculus [McCarthy, 1963; Reiter, 2001], a known formal model for reasoning about dynamic systems, and highlight an extension that can deal with epistemic reasoning and reasoning about agent abilities [Lesperance *et al.*, 1995].

Unfortunately, the current Situation Calculus model for epistemic reasoning does not allow for agents with limited resources and is consequently unrealistic for cloud robotic applications. Therefore we propose an extension to this theory that allows a robot to forget information acquired from the cloud (Section 5). We then apply this extension to formalise a hypothetical problem of a robot that needs to use information in the cloud in order to carry out its objectives (Section 6). Finally we prove and discuss properties that highlight the benefits of this formal account of reasoning for cloud robotics.

2 Related Work

Early discussions on cloud robotics have examined its transformative potential to endow robots with a degree of intelligence not possible in conventional robotic systems [Kuffner, 2010; Guizzo, 2011]. With this goal in sight, the research has followed two distinct paths. Firstly, much of the existing research has focused on architectural considerations for enabling cloud robotics. For example, recent work has examined robot communication protocols [Hu *et al.*, 2012; Hunziker *et al.*, 2013] as well as robot architectures for cloud-based distributed computation [Arumugam *et al.*, 2010].

In parallel with the architectural considerations, other research has considered specific problems for which cloud-based robotics can provide a benefit. For example, combining Google's object recognition services and a large online hand grasp database [Goldfeder *et al.*, 2009] to allow a robot to learn to recognise and grasp unfamiliar objects in its environment [Kehoe *et al.*, 2013]. Such applications provide important functionality and highlight the potential for cloud robotics. However, they lack the generality to show how cloud robotics can be applied in a broader context to enable a robot to know *how* and *when* to use these vast new resources available at its wireless fingertips.

Of the current research efforts only the RoboEarth project [Tenorth *et al.*, 2012] comes close to considering the problem of a robot reasoning about its knowledge and abilities. Here a knowledge exchange language is developed, and actions models are defined using OWL-based ontologies allowing for the specification of taxonomies and relationships between different entities [Kunze *et al.*, 2011].

Action *recipes* define the steps that a robot needs to take in-order to fulfill a task. Furthermore a step within a recipe can itself be a sub-action, thus allowing for the arbitrary composition of actions. Importantly, actions can contain *requirements* that must be satisfied before the recipe can be undertaken. Satisfying these requirements may involve the robot simply possessing a particular sensor or actuator but may also involve the robot undertaking further actions to gain information; such as *build a map of an environment*. Reasoning is performed to ensure that all requirements can be satisfied by the *capabilities* of the robot and ultimately an action recipe is decomposed into the primitive actions to be executed.

While the RoboEarth framework does not have a strict formalisation of epistemic reasoning, nevertheless the authors argue that it does enable some reasoning tasks related to its knowledge and abilities. For example the robot can be asked questions such as: "Can you set the table with cups and plates?" or "Can you set the table with silverware?" [Kunze *et al.*, 2011]. The robot may answer in the affirmative to the first question, providing it satisfies all the necessary requirements such as an appropriate robotic hand and the algorithms and models to recognise cups, plates and tables, but may have to answer in the negative for the second question if it doesn't have the correct model to recognise silverware.

3 Motivation

Unfortunately, the previous RoboEarth example masks some important ambiguities about the types of questions that the robot is able to answer regarding its knowledge and abilities. In particular, it is not clear whether the human in the example is asking the robot whether it *can in principle* set a table or whether it can do so in some concrete instance. The robot's affirmative answer is clearly for the "in-principle" question, since by matching its capabilities to the task requirements it is indeed able to state that in a suitable room, where a table and various cups and plates are located, it would be able to set the table. However the recipe requirements in the example only ensures that the robot is able to recognise cups and plates, it doesn't require the robot to *know* that such cups and plates are in fact readily available for its use.

In order for the robot to answer questions about a concrete instance it would not only need all the capabilities mentioned earlier but it would require knowledge that there are in fact the necessary objects nearby for it to use. This highlights that simply having the necessary abilities to perform an action are not enough to determine if that action can be performed in a concrete situation. Instead, to genuinely answer such questions requires reasoning about the robot's knowledge of its environment.

While it is conceivable that these epistemic requirements could be explicitly added to the RoboEarth action recipes, it is not clear that this could be done without a dramatic increase in complexity of the system. Rather we argue that epistemic reasoning must be embedded implicitly within the system to avoid a blow-out in the specification and modelling of the robot. Consequently it is necessary to provide a clear and unambiguous account of reasoning for cloud robotics.

In order to highlight the requirement for epistemic reasoning and to show how such reasoning can be performed we adopt the following scenario throughout the rest of the paper. We consider a robot that needs to enter a room. There is a door that is initially closed and unlocked, and two buttons that the robot can push. One of these buttons will open the door while the other button will lock the door; and once locked the door cannot be unlocked. Initially, the robot does not know which button will have which result. Fortunately, it is able to access information in the cloud to determine the identity of the button that will open the door.

While simple, this example scenario has the essential features of realistic situations where more detailed information from the cloud (like a map) is needed by a robot to achieve a demanding goal (like getting to a location).

4 Situation Calculus

The Situation Calculus provides a formal language based on classical first-order logic to describe dynamic domains [McCarthy, 1963; Reiter, 2001]. It distinguishes three types of terms: *situations* representing histories as the world evolves; *fluents* denoting domain properties that may change as a result of actions; and *actions* that can be performed by the reasoner.

The function $do(a, s)$ represents the situation that results from performing action a at situation s, while S_0 denotes the initial situation where no actions have taken place. For each action a *precondition axiom* $Poss(a, s)$ specifies the conditions under which action a is possible in situation s and *successor state axioms* specify how the value of fluents change as the result of actions. For a more comprehensive technical formulation of what is required of a Situation Calculus basic action theory, we refer to [Reiter, 2001].

4.1 Knowledge

The Situation Calculus has been extended with a possible world semantics to capture a notion of knowledge in the face of sensing actions [Scherl and Levesque, 1993]. An agent can be said to *know* that some fact ϕ is true if and only if ϕ is true in all possible states of the world that the agent can be in. This notion can be formalised using a special epistemic predicate $K(s', s)$ to represent the fact that in situation s the agent thinks the world could equally be in situation s'. Situation s' is said to be *accessible* from s. An agent in situation s that knows that fluent ϕ holds can be defined by[1]:

$$\mathbf{Know}(\phi, s) \overset{\text{def}}{=} \forall s'.K(s', s) \supset \phi[s']$$

Note that the predicate K is required to be both transitive and Euclidean, ensuring that the agent has introspection about its knowledge; that is, it knows whether it knows something.

Of course reasoning about knowledge is not very interesting unless that knowledge can change over time. The introduction of sensing actions provides a mechanism by which knowledge can be acquired by an agent. The process of knowledge acquisition is encoded through the successor state

[1]We adopt the commonly used notation that $\phi[s]$ represents the formula ϕ as holding in the situation s.

axiom of the *K* relation. An example of such a successor state axiom, containing a single hypothetical sensing action *sense* that senses the value of a fluent *Sensed*, would be:

$$\begin{aligned}K(s^*, do(a, s)) \equiv& \exists s'.s^* = do(a, s') \wedge Poss(a, s') \wedge K(s', s)\\ &\wedge [a = sense \supset Sensed(s') \equiv Sensed(s)]\end{aligned}$$

After a non-sensing action, the accessible situations are the successors of all the situations that were previously accessible. However, for sensing actions only those situations that agree with the actual world on the sensed fluent remain accessible. Effectively, after performing the *sense* action, the agent will know the value of the *Sensed* fluent.

4.2 Ability

The formalisation of agent ability builds on the notion of knowledge to allow an agent to reason about when it is, or is not, able to achieve a goal. In this paper we only provide a summary of the relevant formalisms and the interested reader is referred to [Lesperance *et al.*, 1995] for more details.

The ability to achieve a goal involves an agent knowing what to do and when to do it. Central to this ability is the notion of an *action selection function* which prescribes the action that an agent should take in a given situation. For convenience the following definitions are introduced:[2]

$$\begin{aligned}&\mathbf{OnPath}\ (\sigma, s, s') \overset{\text{def}}{=} s \leq s' \wedge\\ &\qquad \forall a \forall s^*.(s < do(a, s^*) \leq s' \supset \sigma(s^*) = a)\\ &\mathbf{CanGet}\ (\phi, \sigma, s) \overset{\text{def}}{=}\\ &\qquad \exists s'.(\mathbf{OnPath}(\sigma, s, s') \wedge \mathbf{Know}(\phi, s') \wedge\\ &\qquad \forall s^*.[s \leq s^* < s' \supset \exists a \mathbf{Know}(\sigma(\text{now}) = a, s^*)])\\ &\mathbf{Can}(\phi, s) \qquad \overset{\text{def}}{=} \exists \sigma.\mathbf{Know}(\mathbf{CanGet}(\phi, \sigma, \text{now}), s)\\ &\mathbf{KnowIf}(\phi, s) \overset{\text{def}}{=} \mathbf{Know}(\phi, s) \vee \mathbf{Know}(\neg\phi, s)\end{aligned}$$

The first defines that situation s' is *on the path* from s as specified by the action selection function σ when all the intermediate actions on that path are those prescribed by σ. The second shows the conditions under which an agent, following an action selection function σ, *can get* to a situation where ϕ holds. The third states that an agent *can achieve* a goal ϕ if there is an action selection function such that the agent knows that following the prescribed actions will lead to ϕ holding. Finally, the last states that to *know if* something holds means to either know that it holds or to know that it doesn't hold.

5 Forgetting in the Cloud

The formalism developed in [Lesperance *et al.*, 1995] provides a model for an agent to acquire knowledge about the world. However, it assumes that the acquired knowledge is remembered indefinitely. This is a reasonable assumption in cases where the acquired knowledge is fairly limited and/or the agent has no limits on its computational resources. Unfortunately, cloud information is vast and robots have practical restrictions on their computational capabilities, so such a model is no longer applicable. Instead we need to consider the possibility that the robot will need to forget some of its acquired knowledge.

The notion of knowledge forgetting that we introduce allows knowledge that has been acquired from the cloud to be forgotten after a certain number of subsequent actions. To do this we provide an alternative account of the epistemic relation for the Situation Calculus extended with knowledge.

Axiom 1 *Consider the set of all initial situations* S_0, $S_1, \ldots, S_m$, *and an upper bound* n *on the longevity of acquired knowledge. Let Z be a ternary relation that is transitive and Euclidean with respect to the second and third parameters. The* longevity successor state axioms *for Z are:*

$$\begin{aligned}Z(k, s^*, do(a, s)) \equiv \exists s'.(&s^* = do(a, s') \wedge\\ &Z(k-1, s', s) \wedge Poss(a, s')),\end{aligned}$$
where $k \in 1 \ldots n$.

These successor state axioms work in a similar manner to the axioms for the previous *K* relations; as they track the relation between possible worlds as a result of actions. However, unlike the previous *K* relation, the new successor state axioms do not get changed as a result of sensing actions. Instead they effectively attach an action counter to each knowledge state. This counter can then be used as a means of controlling how the *Z* relation defines the epistemic commitments of an agent; acquiring knowledge through sensing but also forgetting that knowledge after some number of subsequent actions.

To do this we first introduce a *sensing equivalence* relation *SEQ* as a means of separating the generic aspects of the formalism from those that are specific to a particular scenario.

Definition 1 *Consider a Situation Calculus basic action theory* Σ, *a set of sensing actions* $a_1, \ldots, a_n$ *and fluents* $\phi_1, \ldots, \phi_n$. *Then* $SEQ(a, s', s)$ *is a* sensing equivalence *relation if there exists an axiom in* Σ *of the form:*

$$SEQ(a, s', s) \equiv \bigwedge_{i=1}^{n} a = a_i \supset [\nabla_i.(\phi_i[s'] \equiv \phi_i[s])],$$
where each ∇_i *is a composition of first order quantifiers over the non-situation free variables in* ϕ_i.

The equivalence specified by an *SEQ* relation is problem specific and captures the knowledge that is acquired by the sensing actions. For example, the following represents a sensing equivalence relation for the running example.

Example 1 *The scan action queries the cloud for available information, while reading the Keymap allows the robot to discover which button is the key to opening the door:*

$$\begin{aligned}&SEQ(a, s', s) \equiv\\ &\quad (a = read(Keymap) \supset [\forall b.(Key(b, s') \equiv Key(b, s))]) \wedge\\ &\quad (a = scan \supset [\forall m.(InCloud(m, s') \equiv InCloud(m, s))])\end{aligned}$$

A new relation *K* can now be defined in terms of the *Z* successor state axioms and a sensing equivalence relation.

[2] $s \leq s'$ is shorthand for $s < s' \vee s = s'$; the pseudo-variable "now" is used to represent the situation bound by the enclosing **Know** (e.g., $\mathbf{Know}(\sigma(\text{now}) = a, s^*)$ stands for $\forall s'.K(s', s^*) \supset \sigma(s') = a$); we use the name **KnowIf** rather than **KnowWhether** introduced in the original.

Definition 2 *Consider a Situation Calculus basic action theory, extended with the Z successor state axioms and a sensing equivalence relation SEQ. Let latency(a) represent the longevity value for a sensing action (note: by definition non-sensing actions are assigned a longevity value higher than the highest value of the parameterised Z relation). Then K is a* knowledge acquisition relation*:*

$$\begin{aligned}K(s', s) \overset{\text{def}}{=} \; & \exists n.Z(n, s', s) \wedge \{n = 0 \vee \\ & [\neg\exists \hat{n}, \hat{s}', \hat{s}, a.Z(\hat{n}, \hat{s}', \hat{s}) \wedge \hat{n} < n \wedge \\ & \; do(a, \hat{s}') \leq s' \wedge do(a, \hat{s}) \leq s \wedge \\ & \; \neg \mathit{SEQ}(a, \hat{s}', \hat{s}) \wedge n - \hat{n} \leq \mathit{latency}(a)]\}\end{aligned}$$

The knowledge acquisition definition works by placing restrictions over the *Z* relation so that the latency of a sensing action determines the expiry for knowledge acquired from that action. A *Z* instance will also be a *K* instance provided that it is not being *blocked* by some previous, still active, sensing action. This blocking reduces the *K* accessibility between situations resulting in the agent gaining knowledge. However, when the sensing action expires, then the block will be removed and so the once inaccessible situations will become accessible again and the agent will effectively have forgotten what it had previously learned by sensing.

The new version of the *K* relation is applied directly to the previous definition of knowing ($\mathbf{Know}$) by [Lesperance *et al.*, 1995] discussed earlier. Furthermore it is fairly trivial to observe that this new formulation of knowledge is a direct generalisation of the old (although we do not prove this here for space reasons). Namely we can attain the original notion of knowledge without forgetting by simply requiring the latency of all actions to be some value higher than the total number of actions in the action sequence. We shall refer to this special case as *ideal knowledge* ($\mathbf{Know_I}$) and use it to define a notion of the *epistemic independence* of actions.

Definition 3 *For Situation Calculus basic action theory Σ, a set of actions $a_1, \ldots, a_n$, a fluent ϕ. A sequence of actions $a_1, \ldots, a_n$ is said to be be* epistemically independent *of ϕ if for any situation s: $\Sigma \models \mathbf{Know_I}(\phi, s)$ iff $\Sigma \models \mathbf{Know_I}(\phi, do([a_1, \ldots, a_n], s))$*

Validating this new formulation of knowledge requires the establishment of a number of properties. Firstly, the introspective nature of knowledge is preserved. This is an intuitively important property of what it means *to know* something; namely, that if you know something then you know that you know it.

Theorem 1 *Consider a Situation Calculus basic action theory, extended with the longevity successor state axioms for Z. Then the defined relation K is* transitive *and* Euclidean*.*

Proof Sketch: Consider transitive case only as Euclidean case is identical. The $n = 0$ case follows by the definition. For $n > 0$ consider if $K(s, s')$ and $K(s', s'')$ but $\neg K(s, s'')$. *Z* is transitive so as $Z(s, s')$ and $Z(s', s'')$ therefore $Z(s, s'')$. So $K(s, s'')$ is being blocked from holding by some a, $\hat{s}$, $\hat{s}''$, $\hat{n}$ such that $\mathit{SEQ}(a, \hat{s}, \hat{s}'')$ holds and $n - \hat{n} \leq \mathit{latency}(a)$. However, if this were true then there must also be an identical block on either $K(s, s')$ or $K(s', s'')$. □

Next, it is important to show that knowledge can be both acquired through sensing and forgotten after some number of subsequent actions. Naturally, the subsequent actions must be independent of the knowledge in question. For example, if the agent simply repeats a sensing action, then unsurprisingly the knowledge gained by the first action will be re-affirmed by the second action and no forgetting takes place.

Theorem 2 *Let Σ be a Situation Calculus basic action theory extended with the Z successor state axioms, and SEQ be a sensing equivalence relation such that sensing action a_s detects the value of ground fluent ϕ (i.e., $\Sigma \models \mathit{SEQ}(a_s, s', s) \equiv [\phi(s') \equiv \phi(s)]$), where latency$(a_s) = n$ $(n \geq 1)$. Let ϕ hold but be unknown in situation s (i.e., $\Sigma \models \phi[s]$ and $\Sigma \not\models \mathbf{Know}(\phi, s)$). For the set of actions $a_1, \ldots, a_n, a_{n+1}$ that are epistemically independent of ϕ then:*

1. *For any $s^* = do([a_s, a_1, \ldots, a_k], s)$ where $k \leq n$:* $\Sigma \models \mathbf{Know}(\phi, s^*)$.
2. *For any $s^* = do([a_s, a_1, \ldots, a_n, a_{n+1}], s)$:* $\Sigma \not\models \mathbf{Know}(\phi, s^*)$.

Proof Sketch: (1) As $\Sigma \models \phi[s]$ and $\Sigma \not\models \mathbf{Know}(\phi, s)$, so $\phi[s]$ holds but there exists some s' such that $K(s', s)$ and $\neg\phi[s']$. As a_s detects the value of ϕ therefore $\neg\mathit{SEQ}(a_s, s', s)$ and combined with $\mathit{latency}(a_s) \geq 1$ will block any $K(do(a_s, s'), do(a_s, s))$ from holding. Hence $\Sigma \models \mathbf{Know}(\phi, do(a_s, s))$. Since $a_1, \ldots, a_{n+1}$ is epistemically independent of ϕ it cannot change the value of ϕ and so the block will continue to hold for any $k \leq n$. (2) Let $\vec{a} = a_1, \ldots, a_n, a_{n+1}$. Since $\vec{a}$ is epistemically independent of ϕ it cannot change the value of $\mathbf{Know_I}(\phi)$. So, there exists s' where $\neg\phi[s']$ and $Z(i, s', s)$ (for some i), and where $(Z, i + 1 + n + 1, do([a_s, \bar{a}], s'), do([a_s, \bar{a}], s))$ also holds. Since $n + 1 - i \leq n$ so the block caused by $\neg\mathit{SEQ}(a_s, s', s)$ will have expired. Hence $\mathbf{Know}(\phi)$ will no longer hold. □

The importance of the notion of epistemic independence (Definition 3) in allowing acquired knowledge to be forgotten should not be understated. If a piece of acquired knowledge is a precondition for some action, then performing that action will prevent the acquired knowledge from being forgotten. For example, if a robot first senses that a door is open, and if that door being open is a precondition for the robot switching rooms, then the action of switching rooms is not epistemically independent of the acquired knowledge that the door is in fact open. Hence if the robot were to subsequently switch rooms then it would not forget that the door must have been open in order for it to do so.

In such cases it is useful to draw a distinction between *semantic* and *episodic* memory (see, for example, [Schacter *et al.*, 2011]). The knowledge acquired through sensing that the door is open becomes part of the robot's semantic memory. The knowledge acquired as a consequence of the robot switching rooms is part of its episodic memory. In this paper we are concerned with forgetting semantic memory. Forgetting of episodic memory corresponds to the robot forgetting some of its past actions. In a Situation Calculus context this is commonly achieved through the use of *progression*.

The new formalisation of the Situation Calculus epistemic relation developed here satisfies both the introspective nature of knowledge (Theorem 1) as well as the ability to acquire and forget knowledge (Theorem 2). It therefore provides a strong basis on which to model the reasoning needs of a robot with access to the cloud.

6 Scenario Formalisation

To show how reasoning about knowledge and ability for a cloud enable robot works, we return to the scenario described earlier of a robot that needs to enter a room but doesn't know which button to press to open the door. Firstly, the precondition and successor state axioms need to be defined:

$$\begin{array}{lcl}
\mathit{Poss}(\mathit{enter}, s) & \equiv & \mathit{Open}(s) \\
\mathit{Poss}(\mathit{push}(b), s) & \equiv & \mathit{True} \\
\mathit{Poss}(\mathit{read}(m), s) & \equiv & \mathit{HaveMap}(m, s) \\
\mathit{Poss}(\mathit{getmap}(m), s) & \equiv & \mathit{InCloud}(m, s) \wedge \\
 & & \forall m. \neg \mathit{HaveMap}(m, s) \\
\mathit{Poss}(\mathit{dropmap}(m), s) & \equiv & \mathit{HaveMap}(m, s) \\
\mathit{Poss}(\mathit{scan}, s) & \equiv & \mathit{True} \\
\mathit{InRoom}(\mathit{do}(a, s)) & \equiv & a = \mathit{enter} \vee \mathit{InRoom}(s) \\
\mathit{Locked}(\mathit{do}(a, s)) & \equiv & \mathit{Locked}(s) \vee \\
 & & \exists b. a = \mathit{push}(b) \wedge \neg \mathit{Key}(b, s) \\
\mathit{Open}(\mathit{do}(a, s)) & \equiv & \neg \mathit{Locked}(s) \wedge \\
 & & \exists b. a = \mathit{push}(b) \wedge \mathit{Key}(b, s) \\
\mathit{Key}(b, \mathit{do}(a, s)) & \equiv & \mathit{Key}(b, s) \\
\mathit{InCloud}(m, \mathit{do}(a, s)) & \equiv & \mathit{InCloud}(m, s) \\
\mathit{HaveMap}(m, \mathit{do}(a, s)) & \equiv & a = \mathit{getmap}(m) \vee \\
 & & (\mathit{HaveMap}(m, s) \wedge \\
 & & \ a \neq \mathit{dropmap}(m))
\end{array}$$

There are a number of salient aspects to this encoding. Firstly, pushing the wrong button will lock the door and a locked door cannot subsequently be unlocked. Given this constraint it becomes important that the robot should check the cloud to see if there is information about which button is the *Key* to opening the door. This is performed through the *scan* sensing action. Once the robot knows which maps are available it is able to retrieve a map with *getmap*. The map that contains the button key information is the *Keymap*, although there is a second map, *Othermap*, that contains other information not relevant to identifying the key button. Only if the robot has gotten the *Keymap* can it *read* the information necessary to determine the correct button to press.

The sensing equivalence relation models the information that is gain by sensing actions. We adopt the equivalence relation *SEQ* defined in Example 1. Furthermore, sensing actions require latency to model the length of time that a robot can retain knowledge. To model a robot with very limited memory we choose low latency values:

$$\begin{array}{r}
\mathit{latency}(\mathit{scan}) = 2 \\
\forall m. \mathit{latency}(\mathit{read}(m)) = 2
\end{array}$$

Having defined some basic axioms we can specify the facts that are true about the initial state. Essentially this describes the world as it is, which is a separate question to what the robot knows about the world:

$$\begin{array}{c}
\neg \mathit{Open}(S_0), \neg \mathit{Locked}(S_0), \forall m. \neg \mathit{HaveMap}(m, S_0), \\
\neg \mathit{InRoom}(S_0), \neg \mathit{Key}(B_1, S_0), \mathit{Key}(B_2, S_0), \\
\mathit{InCloud}(\mathit{Keymap}, S_0), \mathit{InCloud}(\mathit{Othermap}, S_0)
\end{array}$$

The knowledge that the robot has of the real world can be specified in terms of a set of possible initial situations $S_I = \{S_0, S_{1a}, S_{1b}, S_{1c}, S_{2a}, S_{2b}, S_{2c}\}$ accessible from S_0:

$$Z(0, s', S_0) \equiv \bigvee_{s_x \in S_I} s_x = s'$$

We want the robot to initially know that it is not in the room, the door is not open or locked, and the robot has no maps:

$$\bigcup_{s_x \in S_I} \left\{ \begin{array}{l} \neg \mathit{InRoom}(s_x), \neg \mathit{Open}(s_x), \neg \mathit{Locked}(s_x), \\ \forall m. \neg \mathit{HaveMap}(m, s_x) \end{array} \right\}$$

Properties of the initial situations in S_I are defined in such a way as to indicate that the robot does not know which of the two buttons is the key to opening the door and does not know which maps are currently stored in the cloud:

$$\begin{array}{ll}
S_{1a}: & \neg \mathit{Key}(B_1, S_{1a}), \mathit{Key}(B_2, S_{1a}), \\
 & \mathit{InCloud}(\mathit{Keymap}, S_{1a}), \mathit{InCloud}(\mathit{Othermap}, S_{1a}) \\
S_{1b}: & \neg \mathit{Key}(B_1, S_{1b}), \mathit{Key}(B_2, S_{1b}), \\
 & \neg \mathit{InCloud}(\mathit{Keymap}, S_{1b}), \mathit{InCloud}(\mathit{Othermap}, S_{1b}) \\
S_{1c}: & \neg \mathit{Key}(B_1, S_{1c}), \mathit{Key}(B_2, S_{1c}), \\
 & \mathit{InCloud}(\mathit{Keymap}, S_{1c}), \neg \mathit{InCloud}(\mathit{Othermap}, S_{1c}) \\
S_{2a}: & \mathit{Key}(B_1), S_{2a}), \neg \mathit{Key}(B_2, S_{2a}), \\
 & \mathit{InCloud}(\mathit{Keymap}, S_{2a}), \mathit{InCloud}(\mathit{Othermap}, S_{2a}) \\
S_{2b}: & \mathit{Key}(B_1, S_{2b}), \neg \mathit{Key}(B_2, S_{2b}), \\
 & \neg \mathit{InCloud}(\mathit{Keymap}, S_{2b}), \mathit{InCloud}(\mathit{Othermap}, S_{2b}) \\
S_{2c}: & \mathit{Key}(B_1, S_{2c}), \neg \mathit{Key}(B_2, S_{2c}), \\
 & \mathit{InCloud}(\mathit{Keymap}, S_{2c}), \neg \mathit{InCloud}(\mathit{Othermap}, S_{2c})
\end{array}$$

With the above formalisation Figure 1 shows the progression of the robot's actions and knowledge based on a sequence that will lead to it entering the room:

$$[\mathit{scan}, \mathit{getmap}(\mathit{Keymap}), \mathit{read}(\mathit{Keymap}), \mathit{push}(B_2), \mathit{enter}]$$

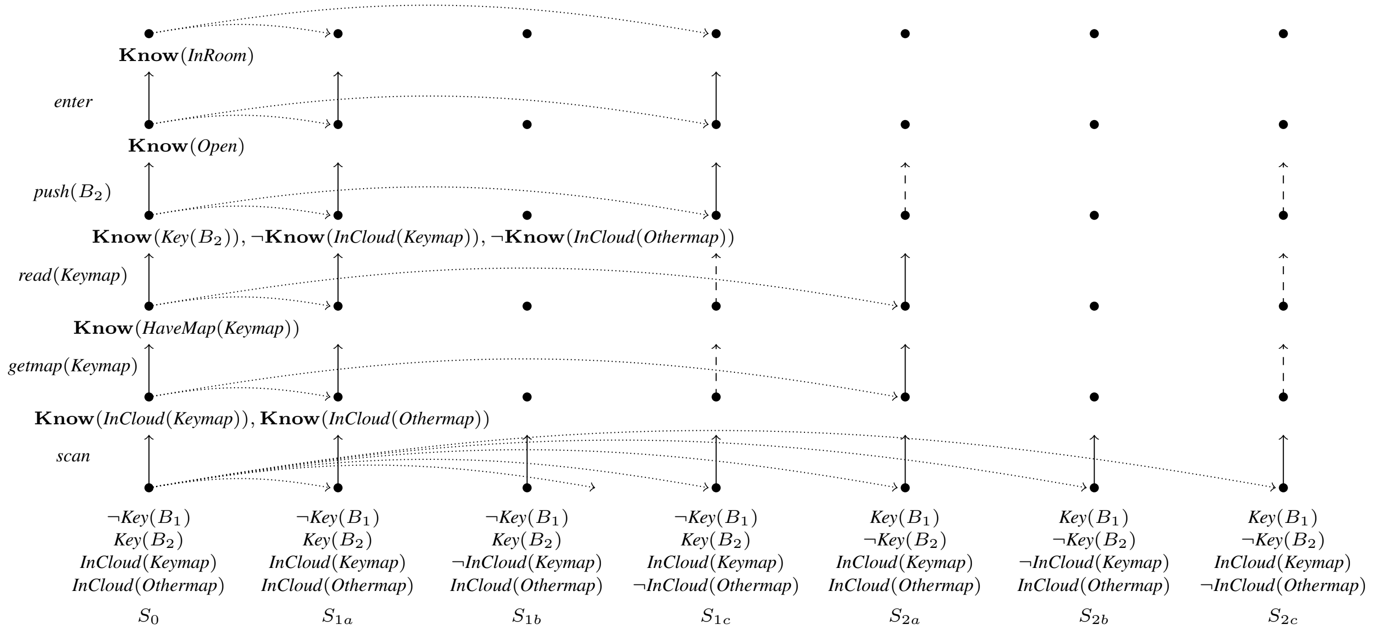
The strength of this model is that it goes beyond simply computing a plan for the robot to achieve its goal. Rather it allows the robot to reason introspectively about what it knows and what it can do with this knowledge. In particular there are a number of interesting properties that can be established. Firstly, it becomes provable that initially the robot does not know what information is in the cloud and is therefore not able to say whether it can in fact find a plan to enter the room.

Theorem 3 $\Sigma \models \neg \mathbf{Can}(\mathit{InRoom}(now), S_0)$

Proof Sketch: We expand and prove by contradiction. So assume there exists an action selection function σ such that:

$$\bigwedge_{s_x \in S_I} \begin{array}{l} \exists s'. (\mathbf{OnPath}(\sigma, s_x, s') \wedge \mathbf{Know}(\mathit{InRoom}, s') \wedge \\ \quad \forall s^*. [s_x \leq s^* < s' \supset \\ \qquad \exists a. \mathbf{Know}(\sigma(\text{now}) = a, s^*)]) \end{array}$$

Initial situations S_{1b} and S_{2b} (K accessible from each other) have no *Keymap* in the cloud and are identical except



Know(InRoom)
enter
Know(Open)
push(B2)
Know(Key(B2)), ¬Know(InCloud(Keymap)), ¬Know(InCloud(Othermap))
read(Keymap)
Know(HaveMap(Keymap))
getmap(Keymap)
Know(InCloud(Keymap)), Know(InCloud(Othermap))
scan
¬Key(B1) Key(B2) InCloud(Keymap) InCloud(Othermap) S0
¬Key(B1) Key(B2) InCloud(Keymap) InCloud(Othermap) S1a
¬Key(B1) Key(B2) ¬InCloud(Keymap) InCloud(Othermap) S1b
¬Key(B1) Key(B2) InCloud(Keymap) ¬InCloud(Othermap) S1c
Key(B1) ¬Key(B2) InCloud(Keymap) InCloud(Othermap) S2a
Key(B1) ¬Key(B2) ¬InCloud(Keymap) InCloud(Othermap) S2b
Key(B1) ¬Key(B2) InCloud(Keymap) ¬InCloud(Othermap) S2c

Figure 1: The progression of actions leading to the robot being in the room. Dotted lines represent the epistemic accessibility from the actual situation S_0 to the alternative possible situations. Solid vertical lines indicate actions that are performed from epistemically accessible situations. Actions performed from situations currently inaccessible from the real situation are represented with broken vertical lines. These remain relevant because forgetting can result in some subsequent situation regaining its epistemic accessibility (e.g., the sequence from S_{1c} leads to the robot forgetting which maps are in the cloud).

for the identity of the key button. So, any action sequence not containing *enter* that is legal from the one situation will also be legal from the other. Hence the situations in every step of such a sequence will also be *K* accessible from each other. Therefore σ must prescribe identical actions from both these situations leading up to the *enter* action. But the key in both these situations is different so pushing the correct button for one will be the wrong button for the other and vice-versa. Hence no such action selection function exists. □

A second interesting property is that while the robot doesn't initially know if it can enter the room, after scanning the cloud it will know the *Keymap* is in the cloud and so will be able to know that it can devise a plan to enter the room.

Theorem 4 $\Sigma \models \mathbf{Can}(\mathit{InRoom}(\mathit{now}), \mathit{do}(\mathit{scan}, S_0))$

Proof Sketch: Prove by constructing an appropriate action selection function σ. Consider two action sequences:

$$\vec{a}_1 = [\mathit{scan}, \mathit{getmap}(\mathit{Keymap}), \mathit{read}(\mathit{Keymap}), \mathit{push}(B_2), \mathit{enter}]$$
$$\vec{a}_2 = [\mathit{scan}, \mathit{getmap}(\mathit{Keymap}), \mathit{read}(\mathit{Keymap}), \mathit{push}(B_1), \mathit{enter}]$$

From $\mathit{do}(\mathit{scan}, S_0)$, only $\mathit{do}(\mathit{scan}, S_{1a})$ and $\mathit{do}(\mathit{scan}, S_{2a})$ are *K* accessible so only consider these three paths. Let σ produce $\mathit{do}(\vec{a}_1, S_0)$ and $\mathit{do}(\vec{a}_1, S_{1a})$. Both paths lead to the robot being in the room and the action sequence in both is identical. Let σ produce $\mathit{do}(\vec{a}_2, S_{2a})$. This follows an identical action sequence up to the $\mathit{read}(\mathit{Keymap})$. At this point the resulting situation is no longer *K* accessible from the S_0 and S_{1a} paths so is free to follow a different action sequence; in which case pushing $\mathit{push}(B_1)$ will allow it to enter the room. □

After querying the cloud the robot knows that it can enter the room. However, it still doesn't know which button is the key and hence the robot doesn't actually know that pressing button B_2 will open the door.

Theorem 5

$$\Sigma \models \neg\mathbf{KnowIf}(\mathit{Open}(\mathit{do}(\mathit{push}(B_2), \mathit{now})), \mathit{do}(\mathit{scan}, S_0))$$

Proof Sketch: From $\mathit{do}(\mathit{scan}, S_0)$ situation $\mathit{do}(\mathit{scan}, S_{2a})$ is *K* accessible, hence it doesn't know that B_2 is the key. If it pushed B_2 then the door would be open in the actual situation but closed in the *K* accessible path leading from S_{2a}. Hence the robot would not know if the door was open or not. □

Finally, the robot is able to perform more complicated feats of introspective reasoning such as reasoning about what it would know if it were to undertake some action. For example, it can know that after it scans the cloud it will know if the door would be open after retrieving and reading the *Keymap* and pushing B_2.

Theorem 6

$$\Sigma \models \mathbf{Know}(\mathbf{KnowIf}(\mathit{Open}(\mathit{do}([\mathit{getmap}(\mathit{Keymap}), \mathit{read}, \mathit{push}(B_2)], \mathit{now})), \mathit{do}(\mathit{scan}, S_0))$$

Proof Sketch: Reduces to needing to show that for situations $s_x \in \{\mathit{do}(\mathit{scan}, S_0), \mathit{do}(\mathit{scan}, S_{1a}), \mathit{do}(\mathit{scan}, S_{2a})\}$ then $\mathbf{KnowIf}(\mathit{Open}(\mathit{do}([\mathit{getmap}(\mathit{Keymap}), \mathit{read}, \mathit{push}(B_2)], s_x)$ holds. Arguments are similar to the previous theorems. □

The scenario and results presented here highlight the subtle introspective reasoning that is enabled by this formalism. The robot is able to reason about what it can or can't do based not only on the information it currently has but also on the information that it knows that it can get. This represents a powerful mechanism for the robot to use information in the cloud to make decisions dynamically to achieve its objectives.

7 Conclusion and Future Work

In this paper we have motivated the need for robots to be able to use cloud based resources. We then examined some existing approaches to cloud robotics and highlighted a gap in current technologies concerned with a robot's ability to reason introspectively about its knowledge and ability. To address this gap we provided a formalism for reasoning about knowledge and ability using the Situation Calculus, and extended it with a notion of forgetting to cope with the high data demands of the cloud and limited capabilities of robots. Finally we formalised an example scenario with many of the essential features of a realistic situation in which a robot might find itself. We then used this scenario to prove properties of introspective reasoning that allow a robot to use the cloud in a flexible manner in order to achieve its goals.

This work opens up a four specific avenues for future research. Firstly, the scope of the framework can be extended to move beyond reasoning only about *information* stored in the cloud. As highlighted by the RoboEarth project [Kunze *et al.*, 2011], a robot that uses the cloud should also be able to download *instructions* from the cloud, for example instructions on how to perform some task. It is therefore necessary to extend the formal account of introspective reasoning to model a robot that reasons about and retrieves both data and instructions from the cloud.

The second area for future research concerns the link between different notions of forgetting. While we have developed a notion of knowledge forgetting, the more commonly studied form is that of logical forgetting [Lin and Reiter, 1994]. This latter notion of forgetting involves the replacement of a formula with one that is logically weaker and is a mechanism to reduce the underlying expressivity of a language. While superficially unrelated to knowledge forgetting, more recent work has established a connection between logical forgetting and belief erasure [Nayak *et al.*, 2007]. Consequently, it would be interesting to explore any potential relationships that may exist between the Situation Calculus based knowledge forgetting developed here and these other forms of forgetting.

The third area for future research involves the operational behaviour of determining when forgetting should occur. In this paper we explored an implicit notion of forgetting, where the robot forgets after a certain number of subsequent actions. However a model where the agent forgets as a result of an explicit forgetting action is also possible. The choice of forgetting formalism would be dictated by the requirements of the agent being modelled.

The final avenue for future research concerns the need to consider the practical aspects of imbuing real robots with introspective reasoning abilities. This can either take the form of a direct implementation of the Situation Calculus based formalism within a robot, or alternatively to treat the formalism as normative principles and to see the extent to which other approaches can be made to satisfy these principles.

Acknowledgements

This research was supported under Australian Research Council's (ARC) *Discovery Projects* funding scheme (project number DP 120102144). The fourth author is the recipient of an ARC Future Fellowship (project number FT 0991348) and is also affiliated with the University of Western Sydney.

References

[Arumugam *et al.*, 2010] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit. DAvinCi: A cloud computing framework for service robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3084–3089, 2010.

[Goldfeder *et al.*, 2009] Corey Goldfeder, Matei T. Ciocarlie, Hao Dang, and Peter K. Allen. The columbia grasp database. In *ICRA*, pages 1710–1716. IEEE, 2009.

[Guizzo, 2011] E. Guizzo. Robots with their heads in the clouds. *Spectrum, IEEE*, 48(3):16 –18, march 2011.

[Hu *et al.*, 2012] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. Cloud robotics: architecture, challenges and applications. *IEEE Network*, 26(3):21–28, 2012.

[Hunziker *et al.*, 2013] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea. Rapyuta: The RoboEarth cloud engine. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[Kehoe *et al.*, 2013] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg. Cloud-based robot grasping with the google object recognition engine. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[Kuffner, 2010] J. Kuffner. Cloud-enabled robots, 2010. In conjunction with the 10th IEEE-RAS International Conference on Humanoid Robots. http://i61www.ira.uka.de/users/asfour/Workshop-Humanoids2010/talks/James-Kuffner-Humanoids2010.pdf.

[Kunze *et al.*, 2011] Lars Kunze, Tobias Roehm, and Michael Beetz. Towards semantic robot description languages. In *ICRA*, pages 5589–5595. IEEE, 2011.

[Lesperance *et al.*, 1995] Y. Lesperance, H. J. Levesque, F. Lin, and R. B. Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66:2000, 1995.

[Lin and Reiter, 1994] Fangzhen Lin and Ray Reiter. Forget it! In *In Proceedings of the AAAI Fall Symposium on Relevance*, pages 154–159, 1994.

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Stanford University Artificial Intelligence Project Memo 2, 1963.

[Nayak *et al.*, 2007] Abhaya C. Nayak, Yin Chen, and Fangzhen Lin. Forgetting and update – an exploration. In Giacomo Bonanno, James P. Delgrande, Jérôme Lang, and Hans Rott, editors, *Formal Models of Belief Change in Rational Agents*, volume 07351 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[Schacter *et al.*, 2011] Daniel L. Schacter, Daniel T. Gilbert, and Daniel M. Wegner. *Psychology*, chapter 6, pages 240–241. Worth Publishers, 2011.

[Scherl and Levesque, 1993] Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In Richard Fikes and Wendy G. Lehnert, editors, *AAAI*, pages 689–695. AAAI Press / The MIT Press, 1993.

[Tenorth *et al.*, 2012] Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 14–18 2012. Best Cognitive Robotics Paper Award.

[Waibel *et al.*, 2011] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. Roboearth. *Robotics Automation Magazine, IEEE*, 18(2):69 –82, june 2011.

Point-Sensitive Circumscription Computation via Answer Set Programming and Applications
(Preliminary Report)

Hai Wan[1] **and Zhanhao Xiao**[2] **and Rui Yang**[2] **and Pu Wang**[2]
[1]Software School, Sun Yat-Sen University, P.R. China 510275
[2]School of Information Science and Technology, Sun Yat-Sen University, P.R. China 510275
wanhai@mail.sysu.edu.cn, zhanhaoxiao@gmail.com, yangrui_yr90@163.com, wangpu96@gmail.com

Abstract

As a generalization of both McCarthy's circumscription and Lifschitz's pointwise circumscription, Amir's point-sensitive circumscription is a useful tool for formalizing solutions for Theories of Action. Point-sensitive circumscription can maintain the control of the minimization process over selective fine-grained variance of predicates and functions, whereas the computation and applications still remain unsatisfactory. Thus, the practical value of point-sensitive circumscription has been severely restricted. In this paper, we propose translation Tr^{ps} from point-sensitive circumscription to first-order stable model semantics over arbitrary structures. Based on the reduction from stable model semantics to answer set programming over finite structures, a point-sensitive circumscription solver, named psc2lp, is developed. We also provide a situation calculus style encoding method by examining a variant of the Yale Shooting Scenario, to solve the Frame Problem and the Qualification Problem. These problems in situation calculus represented naturally by point-sensitive circumscription, can be handled by our approach using existing answer set solvers effectively.

1 Introduction

Point-sensitive circumscription, devised by Amir in [Amir, 1997; 1998], is a nonmonotonic method defined along the intuitions of pointwise circumscription [Lifschitz, 1986; 1987a; 1987b] with control of the minimization process over selective fine-grained variance of predicates and functions. Amir's point-sensitive circumscription can be considered as a generalization of both McCarthy's circumscription [McCarthy, 1980; 1986] and Lifschitz's pointwise circumscription, and is a useful tool for finding the Frame Problem and the Qualification Problem solutions for Theories of Action.

Pointwise circumscription has been used in formalizing some Entailment Classes in the theory of Features and Fluents [Sandewall and Shoham, 1995; Iwanuma and Oota, 1996; Iwanuma *et al.*, 2009], and in [Amir, 1997]. It was argued in [Lifschitz, 1987a] and [Doherty and Łukaszewicz, 1994] that pointwise circumscription has the power to be a tool for formalizing solutions for the Frame Problem.

The unintended minimal model which pointwise circumscription can capture is the insistence on minimizing one fluent at a time, while not allowing other changes for other fluents for the same time point in situation calculus. Compared with pointwise circumscription, point-sensitive circumscription might be to minimize changes for all the fluents once for a given time point. However, despite the progresses on theoretical aspects, the computation of point-sensitive circumscription still remains unsatisfactory, which encounters difficulties from a practical viewpoint. On the other hand, the basic case (circumscribing predicates with no other constants varied) for pointwise circumscription can be rewrite as a first-order sentence, whereas this property of generalized pointwise circumscription and point-sensitive circumscription disappears, which makes the computation of point-sensitive circumscription difficult.

This paper intends to address this issue by translating point-sensitive circumscription into Answer Set Programming (ASP) [Gelfond and Lifschitz, 1988], a promising approach that has been successfully implemented by a number of sophisticated solvers [Drescher *et al.*, 2008; Leone *et al.*, 2006]. To the authors' knowledge, we have found no solvers for computing point-sensitive circumscription.

In this paper, we propose a computation approach and implement such a solver. First, we present a translation from point-sensitive circumscription to first-order stable model semantics over arbitrary structures: Tr^{ps}. The translation has been proved faithful. Over finite structures, point-sensitive circumscription under stable model semantics can be translated into answer set programming. Based on this translation, we can compute point-sensitive circumscription by using existing ASP solvers, over finite structures. Secondly, based on the first-order stable model solver - T2LP [Zhang *et al.*, 2011; Zhang, 2011], we implement a solver, named psc2lp, for computing arbitrary point-sensitive circumscription. Thirdly, we also provide a situation calculus style encoding method for Reiter's basic action theories by examining a variant of the Yale Shooting Scenario, which demonstrates the Frame Problem and the Qualification Problem in situation calculus represented naturally by point-sensitive circumscription can be handled by our approach effectively.

2 Preliminaries

2.1 McCarthy's Circumscription

McCarthy's circumscription [1980] is one of the first and major nonmonotonic reasoning tools. We follow the notions of *parallel circumscription* in [Lifschitz, 1994]. Logic symbols used in this paper are defined as usual. Let φ be a first-order sentence, σ_i be a tuple of $minimized\ predicate\ constants$ and σ_v be a tuple of $individual,\ function$, or $predicate\ constants$ totally differing from σ_i. The rest of the vocabulary of φ are called $fixed\ constants$. Let σ_i^* be a tuple of predicate variables with the same arity as predicate constants in σ_i respectively; similarly, let σ_v^* be a tuple of corresponding variables of same arity as constants in σ_v respectively.

Before defining circumscription, we introduce a comparison relation $<$ between two predicate tuples. Moreover, we abbreviate the conjunction of $\forall\bar{x}(P^*(\bar{x}) \leftrightarrow P(\bar{x}))$ (*resp.* $\forall\bar{x}(P^*(\bar{x}) \rightarrow P(\bar{x}))$) for all $P^* \in \sigma_i^*$ and $P \in \sigma_i$, to $\sigma_i^* = \sigma_i$ (*resp.* $\sigma_i^* \leq \sigma_i$). Therefore, we let the comparison $\sigma_i^* < \sigma_i$ stand for the formula $(\sigma_i^* \leq \sigma_i) \wedge \neg(\sigma_i^* = \sigma_i)$. Then parallel circumscription of σ_i for φ with σ_v varied is defined by a second-order formula:

$$\text{Circ}[\varphi;\sigma_i;\sigma_v] = \varphi \wedge \forall\sigma_i^*\sigma_v^*(\sigma_i^* < \sigma_i \rightarrow \neg\varphi(\sigma_i^*,\sigma_v^*)) \quad (1)$$

where $\varphi(\sigma_i^*,\sigma_v^*)$ is obtained by substituting variables in σ_i^* (*resp.* σ_v^*) for corresponding constants in σ_i (*resp.* σ_v).

Intuitively, circumscription makes the extension of some predicates minimal under the precondition guaranteing the validity of φ, where the extension of predicate P is a set of elements in the domain letting P be true. A structure μ is a σ_i*-minimal model* of φ with σ_v varied if it is a model of CIRC$[\varphi;\sigma_i;\sigma_v]$.

2.2 Lifschitz's Pointwise Circumscription

Pointwise circumscription (basic case) was first proposed in [Lifschitz, 1986] and then expanded in [Lifschitz, 1987a; 1987b], called *generalized pointwise circumscription*. The basic case for pointwise circumscription is the formula:

$$\varphi(P) \wedge \forall x\neg[P(x) \wedge \varphi(\lambda y(P(y) \wedge x \neq y))]$$

where we minimize the predicate P with no other constants varied. Intuitively, it shows that it is impossible to make the extension of exactly one minimized predicate smaller by changing it at exactly one point. One of the benefits of such an approach involves general first-order circumscriptive theories. This property disappears in generalized pointwise circumscription and point-sensitive circumscription.

Let $\varphi(P,Z)$ be a sentence, where P represents a predicate constant and Z denotes a list of predicate constants or function constants Z_i(in particular, a 0-arity function constant is an individual constant). We write $EQ_V(P,Q)$ for

$$EQ_V(P,Q) \overset{\text{def}}{=} \forall x(\neg V(x) \rightarrow (P(x) \leftrightarrow Q(x))) \quad (2)$$

where P,Q,V are predicates with same arity. If P,Q are function constants of the same arity as predicate constant V, then $EQ_V(P,Q)$ stands for $\forall x(\neg V(x) \rightarrow (P(x) = Q(x)))$. Intuitively, the formula $EQ_V(P,Q)$ denotes that P and Q are equal outside V.

The generalized pointwise circumscription of P in φ with Z_i varied on V_i is, by definition,

$$\begin{aligned}\text{C}_{\text{PW}}[\varphi;P;Z_1/V_1,\ldots,Z_n/V_n] = \\ \varphi(P,Z) \wedge \forall xP^*Z^*\neg[P(x) \wedge \neg P^*(x) \wedge \\ \bigwedge_{1\leq i\leq k} EQ_{V_ix}(Z_i^*,Z_i) \wedge \varphi(P^*,Z^*)]\end{aligned} \quad (3)$$

where P^* is an auxiliary predicate corresponding to minimized predicate P, Z^* is a list $Z_1^*,\ldots,Z_n^*$ of predicate and function variables corresponding to the predicate and function constants Z, and $\lambda xuV_i(x,u)(i = 1,\ldots,n)$ is a predicate without parameters which does not contain $Z_1,\ldots,Z_n$ and whose arity is the arity of P plus the arity of Z_i.

For a model $\mathcal{M}$ of $\varphi(P,Z)$, let $|\mathcal{M}|$ be the associated universe, and for every term, function or predicate a, $a^{\mathcal{M}}$ is the realization of a in $\mathcal{M}$.

Definition 1. (Definition 2.1 in [Amir, 1998]) *Let $\mathcal{M}_1$, $\mathcal{M}_2$ have the same universe U, and let $\xi \in U^k$, where k is the arity of P. We say that $\mathcal{M}_1 \leq^\xi \mathcal{M}_2$ iff:*

1. *$K^{\mathcal{M}_1} = K^{\mathcal{M}_2}$ for every function or predicate constant K that is neither P nor in Z,*
2. *for any $i = 1,\ldots,n$, $Z_i^{\mathcal{M}_1}$ and $Z_i^{\mathcal{M}_2}$ coincide on $\{\eta|\neg V_i^{\mathcal{M}_1}(\xi,\eta)\}$,*
3. *$P^{\mathcal{M}_1}(\xi) \rightarrow P^{\mathcal{M}_2}(\xi)$.*

Let $[\varphi(P,Z)]$ be the set of models of $\varphi(P,Z)$. The following proposition 1 says that every model of circumscription for $\varphi(P,Z)$ is minimal in $[\varphi(P,Z)]$ according to the orders $\leq^\xi$.

Proposition 1. ([Lifschitz, 1987a]) *Let $\mathcal{M}$ be a model of $\varphi(P,Z)$. $\mathcal{M} \models \text{C}_{\text{PW}}[\varphi;P;Z_1/V_1,\ldots,Z_n/V_n]$ iff for each $\xi \in \mathcal{M}^k$, $\mathcal{M}$ is minimal relative to $\leq^\xi$.*

$$\begin{aligned}\mathcal{M} \models \text{C}_{\text{PW}}[\varphi;P;Z_1/V_1,\ldots,Z_n/V_n] \equiv \\ \forall\mathcal{M}' \in [\varphi(P,Z)]\,\forall\xi \in |\mathcal{M}|^k \\ \neg(\mathcal{M}' \leq^\xi \mathcal{M} \wedge \mathcal{M} \not\leq^\xi \mathcal{M}')\end{aligned}$$

2.3 Amir's Point-Sensitive Circumscription

Amir [1998] presented *point-sensitive circumscription*, a modified version of pointwise circumscription in which the minimized predicate is minimized according to a minimization domain. This minimization domain may be a point and may be the complete set of elements, which preserve the ability to select/vary parts of the theory/domain dynamically.

We use similar notations to those used in generalized pointwise circumscription. We want to minimize a predicate constant P in a appropriate region. In addition to the definition of EQ_V in (2), let us write $LS_R(P,Q)$ for

$$\begin{aligned}LS_R(P,Q) \overset{\text{def}}{=} \forall x(R(x) \rightarrow (P(x) \rightarrow Q(x)) \\ \wedge\exists x(R(x) \wedge \neg P(x) \wedge Q(x)))\end{aligned} \quad (4)$$

where P,Q predicates or functions, and R a predicate, all with same arity. Intuitively, the formula $LS_R(P,Q)$ denotes that $P \cap R \subsetneqq Q \cap R$ or in the other word, the predicate P is smaller than Q in the region R.

The point-sensitive circumscription of P in φ with Z_i varied on V_i and P minimized using R is defined below,

$$\begin{aligned} C_{\text{PS}}[\varphi; P/R; Z_1/V_1, \dots, Z_n/V_n] = \\ \varphi(P, Z) \wedge \forall x P^* Z^* \neg[LS_{Rx}(P^*, P) \wedge \\ \bigwedge_{i=1}^{n} EQ_{V_i x}(Z_i^*, Z_i) \wedge \varphi(P^*, Z^*)] \end{aligned} \tag{5}$$

where P^* is an auxiliary predicate corresponding to minimized predicate P, Z^* is a list $Z_1^*, \dots, Z_n^*$ of predicate and function variables corresponding to the predicate and function constants Z, and $\lambda xuR(x, u)$, $\lambda xuV_i(x, u)(i = 1, \dots, n)$ are predicates without parameters which do not contain $Z_1, \dots, Z_n$ and whose arity both are the arity of P plus the arity Z_i.

For a model $\mathcal{M}$ of $\varphi(P, Z)$, let $|\mathcal{M}|$ be the associated universe, and for every term, function or predicate a, $a^{\mathcal{M}}$ is the realization of a in $\mathcal{M}$.

Definition 2. (Definition 4.1 in [Amir, 1998]) *Let $\mathcal{M}_1$, $\mathcal{M}_2$ have the same universe U, and let $\xi \in U^k$, where k is the arity of P. We say $\mathcal{M}_1 \ll^\xi \mathcal{M}_2$ (a strict partial order) iff:*

1. *$K^{\mathcal{M}_1} = K^{\mathcal{M}_2}$ for every function or predicate constant K that is neither P nor in Z,*
2. *for any $i = 1, \dots, n$, $Z_i^{\mathcal{M}_1}$ and $Z_i^{\mathcal{M}_2}$ coincide on $\{\eta | \neg V_i^{\mathcal{M}_1}(\xi, \eta)\}$,*
3. *$LS_{R(\xi)}(P^{\mathcal{M}_1}, P^{\mathcal{M}_2})(R(\xi) = \lambda uR(\xi, u))$.*

The following proposition says that every model of the circumscription formula for $\varphi(P, Z)$ is minimal in $[\varphi(P, Z)]$ according to all of the orders $\ll^\xi$, and vice versa.

Proposition 2. (Proposition 4.2 in [Amir, 1998]) *Let $\mathcal{M}$ be a model of $\varphi(P, Z)$.*

$$\begin{aligned} \mathcal{M} \models C_{\text{PS}}[\varphi; P/R; Z_1/V_1, \dots, Z_n/V_n] \equiv \\ \forall \mathcal{M}' \in [\varphi(P, Z)]\, \forall \xi \in |\mathcal{M}|^k \neg(\mathcal{M}' \ll^\xi \mathcal{M}) \end{aligned}$$

Example 1. Let $\varphi(P) \equiv (P(a) \vee P(b)) \wedge (a \neq b)$, let $U = \{1, 2\}$ be the set of elements in the universe. Let M_a, M_b, M_{ab}, and M_ϕ be the models with universe U, with a, b interpreted to 1, 2, respectively, and the following interpretations for the predicate P: $P^{M_a} = \{1\}$, $P^{M_b} = \{2\}$, $P^{M_{ab}} = \{1, 2\}$, $P^{M_\phi} = \phi$.

When it comes to pointwise circumscription of P, it becomes counter-intuitive since $C_{PW}[\varphi; P; P/\lambda x.True]$ is unsatisfiable. According to Definition 1, we can obtain $M_a \leq^2 M_b$ and $M_b \not\leq^2 M_a$. As $M_a \in [\varphi(P)]$, M_b is not a model of $C_{PW}[\varphi; P; P/\lambda x.True]$. Similarly, M_a and M_{ab} do not satisfy $C_{PW}[\varphi; P; P/\lambda x.True]$. Because M_ϕ is not a model of $\varphi(P)$, pointwise circumscription of P in $\varphi(P)$ has no model. But when it is applied to point-sensitive circumscription, we can get two models: M_a and M_b, with same models of McCarthy's parallel circumscription, based on Proposition 2.

Compared to pointwise circumscription, point-sensitive circumscription tends to minimize predicates in a minimum view, where predicates be required to be smaller than all other predicates, by controlling the minimization domain.

2.4 Stable Model Semantics

Similar to Parallel circumscription's definition in (1), stable model semantics was recently generalized to first-order language in [Ferraris *et al.*, 2007; Lin and Zhou, 2011]. Given a first-order sentence φ and a tuple σ_i of predicate constants, let SM$[\varphi; \sigma_i]$ stand for the second-order sentence:

$$\text{SM}[\varphi; \sigma_i] = \varphi \wedge \forall \sigma_i^* (\sigma_i^* < \sigma_i \to \neg \text{St}(\varphi; \sigma_i)) \tag{6}$$

where $\text{St}(\varphi; \sigma_i)$ is defined recursively as follows:

- $\text{St}(P(\bar{x}); \sigma_i) = P^*(\bar{x})$ if $P \in \sigma_i$;
- $\text{St}(F(\bar{x}); \sigma_i) = F(\bar{x})$ if $F \notin \sigma_i$;
- $\text{St}(\psi \circ \chi; \sigma_i) = \text{St}(\psi; \bar{P}) \circ \text{St}(\chi; \bar{P})$ if $\circ \in \{\wedge, \vee\}$;
- $\text{St}(\psi \to \chi; \sigma_i) = (\text{St}(\psi; \sigma_i) \to \text{St}(\chi; \sigma_i)) \wedge (\psi \to \chi)$;
- $\text{St}(Qx\psi; \sigma_i) = Qx\text{St}(\psi; \sigma_i)$ if $Q \in \{\forall, \exists\}$.

A structure μ is called a *σ_i-stable model* of φ if it is a model of SM$[\varphi;\sigma_i]$. A predicate constant is *intensional* if it occurs in σ_i; otherwise, it is *extensional*. According to [Cabalar and Ferraris, 2007], every universal formula without existential quantifiers under stable model semantics is equivalent to a logic program, which is a foundation of computing stable model semantics via existing ASP solvers.

Building on the result of [Cabalar *et al.*, 2005], Lee and Palla [2009; 2012] defined a translation that turns an "almost universal" formula under the stable model semantics into a logic program under the assumption that every positive (negative, respectively) occurrence of a formula $\exists x \varphi(x)$ ($\forall x \varphi(x)$, respectively) in the original formula ϑ belongs to a subformula ϕ of ϑ such that ϕ contains no strictly positive occurrence of any intensional predicates. System F2LP[1] is an implementation of the translation above [Lee and Palla, 2009].

Zhang *et al.* showed an embedding of first-order circumscription in first-order stable model semantics and also introduce a translation that turns arbitrary first-order formulas into logic programs under finite structures, implemented as system T2LP[2] [Zhang *et al.*, 2011; Zhang, 2011].

3 Translating Point-Sensitive Circumscription into Stable Model Semantics

An embedding of first-order parallel circumscription without varied constants in first-order stable model semantics has been shown in Section 4 of [Zhang *et al.*, 2011]. In this section, we propose translation from point-sensitive circumscription into stable model semantics over arbitrary structures.

Note that the equivalence between formulas in classical first-order logic is still retained in circumscription. So for every first-order formula, there always exists a formula in negation normal form equivalent to it in circumscription. Negation normal form guarantees that $\neg$ only occurs directly ahead of predicates. Here $\neg P$ is treated as $P \to \bot$, which is called negative literal conveniently. The implication always follows predicates, so that it is handled easily when taking into account the operator St. Thus the translations in this section take formulas in negation normal form as inputs.

[1] http://reasoning.eas.asu.edu/f2lp

[2] http://ss.sysu.edu.cn/~wh/T2LP.html

Our main idea is in brief to introduce auxiliary predicates to simulate the varied predicate constants and their corresponding varying domains $Z_1/V_1, \ldots, Z_n/V_n$, as well as minimization domain R in the second-order sentence.

Definition 3. *Let φ be any first-order sentence in negation normal form. Then we define $Tr^{ps}(\varphi; P/R; Z_1/V_1, \ldots, Z_n/V_n)$ be the conjunction of below formulas with omitting universal quantifiers:*

$$\varphi^{\neg\neg} \wedge \tilde{\varphi} \tag{7}$$

$$\gamma \leftrightarrow \forall \bar{x}(P(\bar{x}) \vee \neg P(\bar{x})) \tag{8}$$

$$(\gamma \rightarrow P_R(\bar{x})) \wedge (\gamma \rightarrow T(\bar{x}, \bar{y})) \wedge \bigwedge_{1 \leq i \leq k} \gamma \rightarrow Q_i(\bar{x}) \tag{9}$$

$$R(\bar{x}, \bar{y}) \wedge P(\bar{y}) \rightarrow P_R(\bar{y}) \tag{10}$$

$$R(\bar{x}, \bar{y}) \wedge (P(\bar{y}) \rightarrow \gamma) \rightarrow (P_R(\bar{y}) \rightarrow \gamma) \tag{11}$$

$$\begin{aligned} T(\bar{x}, \overline{\min}) \vee (R(\bar{x}, \overline{\min}) \wedge (P(\overline{\min}) \rightarrow \gamma) \\ \wedge \neg\neg P(\overline{\min})) \end{aligned} \tag{12}$$

$$\begin{aligned} succ(\bar{x}, \bar{z}) \rightarrow [T(\bar{x}, \bar{y}) \leftrightarrow (R(\bar{x}, \bar{\bar{z}}) \\ \wedge (P(\bar{z}) \rightarrow \gamma) \wedge \neg\neg P(\bar{z})) \vee T(\bar{x}, \bar{z})] \end{aligned} \tag{13}$$

$$\bigwedge_{1 \leq i \leq n} \neg V_i(\bar{x}, \bar{y}) \wedge \neg\neg Z_i(\bar{y}) \rightarrow Q_i(\bar{y}) \tag{14}$$

$$\bigwedge_{1 \leq i \leq n} \neg V_i(\bar{x}, \bar{y}) \wedge \neg Z_i(\bar{y}) \rightarrow (Q_i(\bar{y}) \rightarrow \gamma) \tag{15}$$

where $\varphi^{\neg\neg}$ is obtained from φ by substituting $\neg\neg P(\bar{x})$ for each positive literal $P(\bar{x})$; $\tilde{\varphi}$ is obtained from φ by substituting $P_R(\bar{x})$ for each positive literal $P(\bar{x})$, $(P_R(\bar{x}) \rightarrow \gamma)$ for each negative literal $\neg P(\bar{x})$, $Q_i(\bar{x})$ for each positive literal $Z_i(\bar{x})$ and $(Q_i(\bar{x}) \rightarrow \gamma)$ for each negative literal $\neg Z_i(\bar{x})$ such that $1 \leq i \leq n$; succ describes the successor relation on the domain based on a total order and $\overline{\min}$ is the minimal tuple in the order; P_R, T, Q_i and γ are auxiliary predicates without occurrence in φ.

Actually, Definition 3 provides a syntactic translation from a first-order sentence to another one, which can be achieved in polynomial time. Next, the soundness and completeness of the translation are illustrated by Proposition 3 and then it is proved to be faithful.

Proposition 3. *Let φ be any first-order sentence in negation normal form. Let ψ denote $Tr^{ps}(\varphi; P/R; Z_1/V_1, \ldots, Z_n/V_n)$. Then, over finite structures, $\mathrm{SM}[\psi; P, P_R, T, \gamma, Q_1, \ldots, Q_n]$ is equivalent to $\mathrm{C_{PW}}[\varphi; P/R; Z_1/V_1, \ldots, Z_n/V_n]$, where P_R, T, γ and $Q_1, \ldots, Q_n$ are auxiliary predicates introduced by the translation.*

Proof(sketch). Intuitively, the translation resulting ψ under stable model semantics simulates φ in point-sensitive circumscription. As it is defined above, $\mathrm{SM}[\psi; \sigma_i]$ is equivalent to the formula $\psi \wedge \forall \sigma_i^* \neg(\sigma_i^* < \sigma_i \wedge St(\psi; \sigma_i))$. Compared with the definition of point-sensitive circumscription (5), that of S-M is similar on the second-order part. Based on the similarity, we propose a translation from point-sensitive circumscription into SM with auxiliary predicates.

Indeed, Formula (7) is equivalent to φ because each substitution in $\tilde{\varphi}$ must be true on account of Formulas (8) and (9). In addition, the validation of auxiliary predicates makes formulas (10)-(15) be always true and guarantees their corresponding predicate variables within the range of them, such as $T^* \leq T$. Thus, the conjunction of these formulas, *i.e.*, ψ, is equivalent to φ and it describes the equivalence of translation in the first-order part.

Next, these formulas change after applying the operator St. Specifically, $St(\gamma; \sigma_i) \equiv \gamma^* = \bot$ if $P^* < P$ and otherwise $\gamma^* = \gamma$. Actually, $P^* < P$ must be true or it will make all predicate variables equal to their corresponding predicate constants. Then $\sigma_i^* < \sigma_i$ is false and it makes the second-order part be true which has no influence on the translation. When $P^* < P$, the implication with an antecedent of γ^* is true and the implication in from of $P^*(\bar{x}) \rightarrow \gamma^*$ is equivalent to $\neg P^*(\bar{x})$.

Additionally, $St((10) \wedge (11); \sigma_i)$ describes the property that P_R^* is equivalent to P^* in the region Rx. Because of $P^* < P$, it equals to $\forall \bar{y}(Rx(\bar{y}) \rightarrow (P_R^*(\bar{y}) \rightarrow P(\bar{y})))$. In $St((12) \wedge (13); \sigma_i)$, predicate variable T^* actually describes the existential quantifier over successor structure, according to the idea of Eiter *et al.* and Zhang *et al.*. So it is equivalent to $\exists \bar{y}(Rx(\bar{y}) \wedge P_R^*(\bar{y}) \wedge P(\bar{y}))$. Consequently the conjunction of them is equivalent to $LS_R(P_R^*, P)$.

Similarly, $St((14) \wedge (15); \sigma_i)$ describes the property that Q_i^* and Z_i are equal outside $V_i x$ such that $1 \leq i \leq k$. Then each its conjunctive simulates $EQ_i x(Q_i^*, Z_i)$. Since predicate constants P_R and Q_i are always valid, corresponding predicate variables P_R^* and Q_i^* can change arbitrarily in the domain and it actually describes the second-order universal quantifier. Here predicate variables P_R^* and Q_i^* simulate p and z_i in point-sensitive circumscription respectively. According to the substitution rule, $St(\tilde{\varphi}; \sigma_i)$ is equivalent to φ^*. Because of $St(\varphi^{\neg\neg}; \sigma_i) = \varphi$, the second-order part of $\mathrm{SM}[\psi; \sigma_i]$ is equal to that of $\mathrm{C_{PW}}[\varphi; P/R; Z_1/V_1, \ldots, Z_n/V_n]$. So far the faithfulness of the translation Tr^{ps} is proved. □

Remark 1. *For the special form of point-sensitive circumscription $\mathrm{C_{PS}}[\varphi; P; Z_1/V_1, \ldots, Z_n/V_n]$, whose R actually is treated as $True$, Formulas (10)-(13) can be omitted, which can make the translation Tr^{ps} more efficient by reducing the number of rules generated.*

Remark 2. *As for Tr^{ps}, we have not mentioned function and individual constants because predicates can simulate functions easily. For each n-arity function, we can introduce a n+1-arity predicate to represent it. Particularly, individual constants varied actually can be simulated by individual variables in scope of first-order existential quantifiers.*

4 Application in Situation Calculus

So far, in this paper we have shown how we can translate point-sensitive circumscription into first-order stable model semantics. In this section we will use point-sensitive circumscription representation to reformulate the Frame Problem and the Qualification Problem in situation calculus.

4.1 Situation Calculus

The situation calculus [Reiter, 2001; Lin, 2008] is one of the most well-known formalisms for reasoning about actions. The situation calculus is a many-sorted first-order language (with some second-order ingredients) suitable for representing changes. Prolog can be used to implement the situation calculus, based on the fact that Clark's completion semantics accounts for definitional axioms.

The basic special sorts in situation calculus are $situations$, $actions$, and $fluents$ ($relational$ fluents and $functional$ fluents), situations and actions are represented as individuals that can be quantified over. There could be other sorts, some of them domain dependent like $block$ for blocks in the blocks world and others domain independent like $truth$ for truth values. Assume the following special domain independent predicates and functions: a binary predicate $Holds(P(x), s)$ denoting fluent P is true in situation s, usually using $P(x, s)$ as shorthand for $Holds(P(x), s)$; a binary function $do(a, s)$ denoting the successor situation to s resulting from performing action a; a binary predicate $Poss(a, s)$ meaning that action a is possible in situation s.

We assume that a description $\mathcal{D}$ consists of a finite number of the following sets of axioms. We often identify $\mathcal{D}$ with the conjunction of the universal closures of all axioms in $\mathcal{D}$. In the following, F, F_i are fluent names, A is an action name, V , V_i are truth values, s, s' are situation variables, $\phi(s)$ is a simple state formula about s, constants a, a' are action variables, f is a variable of sort fluent, v is a variable of sort truth value, and x, x_i, y, y_i are lists of variables.

Reiter's $basic\ action\ theory$ (BAT) is of the form

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \tag{16}$$

where

- Σ: the set of the foundational axioms;
- $\mathcal{D}_{ss}$: a set of successor state axioms of the form
  $$F(x, do(a, s)) \leftrightarrow \Phi_F(x, a, s),$$
  where $\Phi_F(x, a, s)$ is a formula that is uniform in s [Reiter, 2001] and whose free variables are among x, a, s;
- $\mathcal{D}_{ap}$: a set of action precondition axioms of the form
  $$Poss(A(x), s) \leftrightarrow \Pi_A(x, s),$$
  where $\Pi_A(x, s)$ is a formula that is uniform in s and whose free variables are among x, s;
- $\mathcal{D}_{una}$: the set of unique name axioms for fluents and actions;
- $\mathcal{D}_{S_0}$: a set of first-order sentences that are uniform in S_0.

4.2 The Frame Problem, the Ramification Problem, and the Qualification Problem

McCarthy and Hayes identified the Frame Problem as the problem of expressing a dynamical domain without explicitly specifying which conditions are not affected by an action [McCarthy and Hayes, 1969]. McCarthy [1986] initially proposed to solve the Frame Problem by the following *generic frame axiom*:

$$Holds(p, s) \wedge \neg abnormal(p, a, s) \rightarrow Holds(p, do(a, s))$$

with the abnormality predicate abnormal circumscribed. However, Hanks and McDermott showed that McCarthy's approach does not work [Hanks and McDermott, 1987]. Reiter proposed a simple syntactic manipulation that turns a set of effect axioms into a set of successor state axioms that completely captures the true value of each fluent in any successor situation [Reiter, 1991].

$$F(x, do(a, s)) \equiv \gamma^+(a, x, s) \vee (F(x, s) \wedge \neg\gamma^-(a, x, s))$$

The Ramification Problem, first discussed by Finger [1986], is about how to encode constraints like this in an action domain, and how these constraints can be used to derive the effects of the actions in the domain. Lin [1995] represents this constraint as a causal constraint, axiomatizing this by introduced a ternary predicate $Caused(p, v, s)$, meaning that fluent p is caused to have truth value v in situation s.

- $\mathcal{D}_{caused}$ is a set of axioms of the form
  $$Poss(A(x), s) \rightarrow$$
  $$(\phi(s) \rightarrow Caused(F(y), V, do(A(x), s))$$
  (direct effects) and
  $$\phi(s) \wedge Caused(F_1(x_1), V_1, s) \wedge \ldots$$
  $$\wedge Caused(F_n(x_n), V_n, s) \rightarrow Caused(F(x), V, s)$$
  (indirect effects).

The Qualification Problem is concerned with the impossibility of listing all the preconditions required for a real-world action to have its intended effect [McCarthy, 1977]. One possible solution to this problem is to assume that an action is always executable unless explicitly ruled out by the theory. This can be achieved by maximizing the predicate $Poss$, or in terms of circumscription, circumscribing $Poss$. The problem becomes more complex when some domain constraints-like axioms can influence $Poss$. Lin and Reiter [1994] called those constraints that yield indirect effects of actions ramification constraints, and those that yield additional qualifications of actions qualification constraints. They are both represented as sentences of the form $\forall s C(s)$, and it is up to the user to classify which category they belong to. Under this framework, only constraints represented as causal rules using $Caused$ can derive new effects of actions, and ordinary situation calculus sentences of the form $\forall s C(s)$ can only derive new qualifications on actions.

4.3 Representing the Frame and the Qualification Problem with Point-Sensitive Circumscription

Amir [1997] adjusted the discrete situation calculus [Lin and Reiter, 1994] to fit his set theoretic language and proposed his solution to the Frame and the Qualification problems. Amir gave a theoretic solution with point-sensitive circumscription and we achieve it practically in this section.

Suppose $Ab(l, a, s)$, which is an predicate constant on fluent l, action a, and situation s, means an abnormality. Normally, fluent l remains after performing action a in situation s. In other word, $Ab(l, a, s)$ denotes l changes in situation $do(a, s)$, which is represented by the following axioms:

$$\neg Ab(l, a, s) \rightarrow (Holds(l, s) \leftrightarrow Holds(l, do(a, s)))$$

To find all fluents keeping persistence in the next situation, we minimize Ab on one situation at a time. Let $V_1(x, y)$ be the formula $\lambda x, y\ \exists l_x, a_x, s_x\ \exists l_y, a_y, s_x\ x =< l_x, a_x, s_x > \wedge\ y =< l_y, a_y, s_y > \wedge\ s_x = s_y$, and $V_2(x, y)$ be the formula $\lambda x, y\ \exists l_x, a_x, s_x\ \exists l_y, a_y, s_x\ x =< l_x, a_x, s_x > \wedge\ y =< l_y, a_y, s_y > \wedge\ do(a_x, s_x) = s_y$. Intuitively, V_1 considers that situation s_x while V_2 considers the next situation. According to the meaning of V in pointwise view, predicates can change arbitrarily inside V with remaining unchanged outside V. In point-sensitive circumscription, Ab/V_1 means that the abnormality in different situations and Ab is allowed to vary in the same situation, while $Holds/V_2$ means that fluents can be changed in the next situation.

$$EQ_{V_1 x}(Ab^*, Ab) \equiv s_x \neq s_y \rightarrow (Ab^*(y) \leftrightarrow Ab(y))$$

$$EQ_{V_2 x}(Holds^*, Holds) \equiv \\ do(a_x, s_x) \neq s_y \rightarrow (Holds^*(y) \leftrightarrow Holds(y))$$

With this circumscription policy, for each situation, there is only one situation, the next situation, being considered rather than all situations. To minimize Ab in all fluents and actions, we let R be $True$. The point-sensitive circumscription can solve the Frame Problem one situation at a time:

$$\mathrm{C}_{\mathrm{PS}}[\varphi; Ab/R; Ab/V_1, Holds/V_2] \tag{17}$$

The models of this point-sensitive circumscription coincide with minimal models of discrete situation calculus, no matter whether nondeterministic actions are allowed to be done.

As far as the Qualification Problem is concerned, the point-sensitive circumscription can solve it. Similarly, we let Abq describe the abnormality of allowance to preform actions. Action a is executable in situation s normally and $Abq(a, s)$ denote a can not be preformed in s.

$$\neg Abq(a, s) \rightarrow Poss(a, s)$$

In situation calculus, there is a class of formulas called constraints, which consist of ramification constraints RC and quantification constraints QC. We use a predicate constant $AllowedS(s)$ to denote situation s satisfying all quantification constraints. The following axiom can guarantee every situation not in $AllowedS$, its next situation is also not in it.

$$\neg AllowedS(s) \rightarrow \neg AllowedS(do(a, s))$$

Next, an action is said to be applicable if and only if its preconditions are met and it does not lead to the violation of QC, which is denoted as follows.

$$App(a, s) \leftrightarrow Poss(a, s) \wedge AllowedS(do(a, s))$$

To consider all situations, we need consider as many as possible action. Thus, we must maximize the predicates $Poss$. Furthermore, predicate Abq should be circumscribed. The point-sensitive circumscription can solve the Quantification Problem with the following policy.

$$\mathrm{C}_{\mathrm{PS}}[\varphi; Abq/R; Abq/V_1, Ab/V_1, Poss/V_1, \\ Holds/V_2, AllowedS/True, App/True] \tag{18}$$

As a result, action quantifications is expressed explicitly by a conjunction of simple formulas of the following form:

$$Poss(a, s) \leftrightarrow \theta_1(a, s) \wedge ... \wedge \theta_n(a, s)$$

5 Implementation and Example

This section shows how to implement a point-sensitive circumscription solver psc2lp[3]. One example: a variant of the Yale Shooting Scenario is presented how psc2lp can be used in solving the Qualification Problem in situation calculus.

5.1 Implementation

A point-sensitive circumscription solver psc2lp is developed based on our approach. psc2lp firstly accepts a point-sensitive circumscriptive theory, then translates it into a logic program, and finally invokes an ASP solver with a corresponding finite extensional database[4].

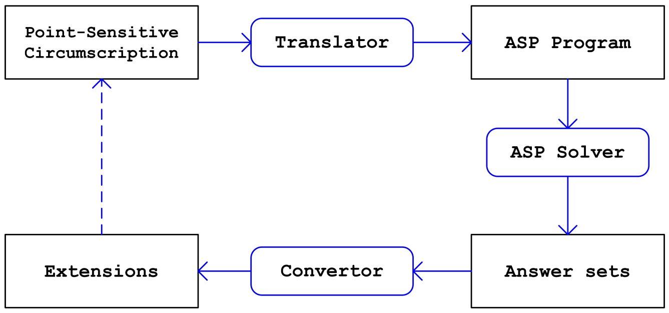


Figure 1: Outline of psc2lp

Figure 1 illustrates how psc2lp works. An input point-sensitive circumscriptive theory is firstly translated to an answer set program by the *translator* in psc2lp. Then, an *ASP solver* is called to compute the answer sets of the program. Finally, the answer sets will be interpreted back to all solutions of the original point-sensitive circumscriptive theory by an *convertor*. For the *ASP solver* module in psc2lp, we just use claspD [5] [Gebser *et al.*, 2007]. The *convertor* in psc2lp is trivial. Hence, the main issue in psc2lp is the *translator*.

Indeed, after the translation Tr^{ps}, the fixed and varied predicates in circumscription are treated as the extensional predicates under stable model semantics which need to be eliminated when translating into a logic program by introducing a sentence of the form $\forall \bar{x}(Q(\bar{x}) \vee \neg Q(\bar{x}))$ for each extensional predicate Q.

[3] http://ss.sysu.edu.cn/∼wh/psc2lp.html

[4] An extensional database is a structure consisting of extensional predicate and function constants under stable model semantics.

[5] http://www.cs.uni-potsdam.de/clasp/

More precisely, we can compute point-sensitive circumscription by 4 steps:

1. Turn the input into the sentence in both prenex normal form and negation normal form;
2. Apply the translation Tr^{ps} to obtain a first-order sentence under stable model semantics;
3. Use Zhang's first order stable model semantics solver T2LP repeatedly till a logic program can be obtained;
4. Add $\forall \bar{x}(Q(\bar{x}) \vee \neg Q(\bar{x}))$ for each fixed and varied predicate.

5.2 Example

Let us examine a variant of the Yale Shooting Scenario (YSS) [Hanks and McDermott, 1987]. Assume that there are two turkeys. As a result of the gun's being shot, exactly one turkey dies. There is only one bullet can be loaded. For this, we have initially, both turkeys are alive, and the gun is loaded. The knowledge representation is provided below with some formulas omitted:

$\mathcal{D}_{ap}$:

$$Holds(loaded, s) \rightarrow Poss(shoot(x), s) \tag{19}$$

$$Holds(loaded, s) \vee \neg Holds(loaded, s) \rightarrow Poss(load, s) \tag{20}$$

$\mathcal{D}_{ss}$:

$$Holds(loaded, do(a, s)) \leftrightarrow (Holds(loaded, s) \wedge a \neq load) \vee (\neg Holds(loaded, s) \wedge a = load) \tag{21}$$

$$Holds(alive(x), do(a, s)) \leftrightarrow Holds(alive(x), s) \wedge a \neq shoot(x) \tag{22}$$

Especially, in this kind of circumscription policy, R is $True$ and $LS_{Rx}(Ab^*, Ab)$ reduces to $Ab^* < Ab$. So Formulas (10)-(13) are not necessary in the translation Tr^{ps}. When the translation is applied on (18), according to (14) and (15) in Tr^{ps} we can get:

$$s_x \neq s_y \wedge \neg\neg Abq(y) \rightarrow Abq'(y)$$
$$s_x \neq s_y \wedge \neg Abq(y) \rightarrow \neg Abq'(y)$$

where $x =< a_x, s_x >$, $y =< a_y, s_y >$ and Abq' is an auxiliary predicate corresponding to Abq, introduced like the introduction of Q. Additionally, according to (14) and (15) we can obtain:

$$do(a_x, s_x) \neq s_y \wedge \neg\neg Holds(y) \rightarrow Holds'(y)$$
$$do(a_x, s_x) \neq s_y \wedge \neg Holds(y) \rightarrow \neg Holds'(y)$$

where $Holds'$ is an auxiliary predicate corresponding to $Holds$. Other similar formulas are omitted because of the limited space.

Besides, (7) in Tr^{ps} applied to (19) is denoted below:

$$\varphi^{\neg\neg} : \quad \neg Holds(loaded, s) \vee Poss(shoot(x), s)$$
$$\tilde{\varphi} : \quad (Holds'(loaded, s) \rightarrow \gamma) \vee Poss'(shoot(x), s)$$

where $Holds'$ and $Poss'$ are auxiliary predicates corresponding to $Holds$ and $Poss$ respectively.

After applying the translation Tr^{ps}, we can get a first-order theory under stable model semantics. Next, via T2LP we can reduce it into an universal theory, which is equivalent to an answer set program. When we obtain a logic program, we can invoke an existing ASP solver to find all solutions.

6 Conclusion

The relationship among McCarthy's circumscription, Lifschitz's pointwise circumscription and Amir's point-sensitive circumscription was clarified in this paper. Furthermore, we proposed and proved a translation Tr^{ps} from point-sensitive circumscription to stable model semantics over finite structures. We can compute point-sensitive circumscription over finite structures by reducing stable model semantics to ASP. Our approach is not only theoretically interesting but also of practical relevance with an example in situation calculus. With point-sensitive circumscription, the Frame problem and the Quantification problem can be solved and we can find all solutions via our solver.

Now we summarize the contributions of this paper. First, we propose a translation from point-sensitive circumscription to stable model semantics over finite structures. Secondly, with psc2lp, we can compute practical problems represented by point-sensitive circumscription effectively, such as the Frame problem and the Quantification problem. Thirdly, compared with propositional case, we can represent problems in a flexible and natural way with allowing existential quantifiers in point-sensitive circumscription.

Acknowledgments

We would like to thank Heng Zhang for his valuable suggestions. Yongmei Liu and her group commented on this paper. Our deep thanks to all of them. We also would like to thank anonymous reviewers for their valuable comments. This research has been partially supported by the Doctoral Program Foundation of Institutions of Higher Education of China under Grant 20110171120041, the Natural Science Foundation of Guangdong Province under Grant S2012010009836, and the Guangzhou Science and Technology Project under Grant 13200059.

References

[Amir, 1997] Eyal Amir. Formalizing action using pointwise circumscription and set theory. In *NRAC'97*, pages 1–17, 1997.

[Amir, 1998] Eyal Amir. Pointwise circumscription revisited. In *KR'98*, pages 202–211, 1998.

[Cabalar and Ferraris, 2007] Pedro Cabalar and Paolo Ferraris. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming*, 7(6):745–759, 2007.

[Cabalar *et al.*, 2005] Pedro Cabalar, David Pearce, and Agustın Valverde. Reducing propositional theories in equilibrium logic to logic programs. In *EPIA'05*, pages 4–17, 2005.

[Doherty and Łukaszewicz, 1994] Patrick Doherty and Witold Łukaszewicz. Circumscribing features and fluents: A fluent logic for reasoning about action and change. In *ISMIS'94*, pages 521–530, 1994.

[Drescher *et al.*, 2008] Christian Drescher, Martin Gebser, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, and Torsten Schaub. Conflict-driven disjunctive answer set solving. In *KR'08*, pages 422–432, 2008.

[Ferraris *et al.*, 2007] Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz. A new perspective on stable models. In *IJCAI'07*, pages 372–379, 2007.

[Finger, 1986] Jeff Finger. *Exploiting constraints in design synthesis.* PhD thesis, Department of Computer Science, Stanford University Stanford, CA, 1986.

[Gebser *et al.*, 2007] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *IJCAI'07*, pages 386–392, 2007.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP'88*, pages 1070–1080, 1988.

[Hanks and McDermott, 1987] Steve Hanks and Drew V. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Iwanuma and Oota, 1996] Koji Iwanuma and Kazuhiko Oota. An extension of pointwise circumscription. *Artifical Intelligence*, 86(2):391–402, 1996.

[Iwanuma *et al.*, 2009] Koji Iwanuma, Katsumi Inoue, and Hidetomo Nabeshima. Reconsideration of circumscriptive induction with pointwise circumscription. *Journal of Applied Logic*, 7(3):307–317, 2009.

[Lee and Palla, 2009] Joohyung Lee and Ravi Palla. System F2LP–computing answer sets of first-order formulas. In *LPNMR'09*, pages 515–521, 2009.

[Lee and Palla, 2012] Joohyung Lee and Ravi Palla. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Intelligence Research*, 43(1):571–620, 2012.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[Lifschitz, 1986] Vladimir Lifschitz. Pointwise circumscription (preliminary report). In *AAAI'86*, pages 406–410, 1986.

[Lifschitz, 1987a] Vladimir Lifschitz. Circumscriptive theories: A logic-based framework for knowledge representation (preliminary report). In *AAAI'87*, pages 364–368, 1987.

[Lifschitz, 1987b] Vladimir Lifschitz. Readings in nonmonotonic reasoning. chapter Pointwise circumscription, pages 179–193. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[Lifschitz, 1994] Vladimir Lifschitz. Circumscription. In D.M.Gabbay et al., editor, *Handbook of Logic in Artificial Intelligence and Logic Programming-Nonmonotonic Reasoning and Uncertain Reasoning*, pages 297–352. Clarendon Press, Oxford, 1994.

[Lin and Reiter, 1994] Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of logic and computation*, 4(5):655–678, 1994.

[Lin and Zhou, 2011] Fangzhen Lin and Yi Zhou. From answer set logic programming to circumscription via logic of GK. *Artificial Intelligence*, 175(1):264–277, 2011.

[Lin, 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *IJCAI'95*, pages 1985–1993, 1995.

[Lin, 2008] Fangzhen Lin. Situation calculus. *Foundations of Artificial Intelligence*, 3:649–669, 2008.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer, Donald Michie, and Michael Swann, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[McCarthy, 1977] John McCarthy. Epistemological problems of artificial intelligence. In *IJCAI'77*, pages 1038–1044, 1977.

[McCarthy, 1980] John McCarthy. Circumscription a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1,2):27–39,171–172, 1980.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, 1986.

[Reiter, 1991] Raymond Reiter. Artificial intelligence and mathematical theory of computation. chapter The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression, pages 359–380. Academic Press Professional, Inc., San Diego, CA, USA, 1991.

[Reiter, 2001] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, Massachusetts, MA, 2001.

[Sandewall and Shoham, 1995] Erik Sandewall and Yoav Shoham. Handbook of logic in artificial intelligence and logic programming (vol. 4). chapter Non-monotonic Temporal Reasoning, pages 439–498. Oxford University Press, Oxford, UK, 1995.

[Zhang *et al.*, 2011] Heng Zhang, Yan Zhang, Mingsheng Ying, and Yi Zhou. Translating first-order theories into logic programs. In *IJCAI'11*, pages 1126–1131, 2011.

[Zhang, 2011] Heng Zhang. *Decidability and translatability of nonmonotonic logics.* PhD thesis, The University of TsingHua, Nov. 2011.

Translating Action Knowledge into High-Level Semantic Representations for Cognitive Robots

Jiongkun Xie, Xiaoping Chen, and Zhiqiang Sui
Multi-Agent Systems Lab.,
University of Science and Technology of China
devilxjk@mail.ustc.edu.cn, xpchen@ustc.edu.cn, zqsui@mail.ustc.edu.cn

Abstract

This paper presents an approach to translating action knowledge into its underlying semantics for action reasoning. To facilitate the translation we propose a new semantic representation which is higher-level and can be explicitly converted into the underlying one. Based on this representation an existing semantic parser is employed as the core of our action knowledge translation. Finally we evaluate this approach and show that it can translate the action knowledge into proposed representations.

1 Introduction

Being able to non-monotonically reason about actions provides high-level cognitive capabilities for an intelligent robot exploring a rich, dynamic and unpredictable world. The action model, which is usually pre-programmed by human designers, is the core of such robots. When an indoor robot is acting in an open environment, it is inevitable to consider what tasks the robot is competent for and how the robot plans to automatically accomplish them. Nevertheless, enumerating all the possible tasks and all the ways to accomplishments is not tractable. Therefore instead of pre-programming all action knowledge that robots might need, dynamically acquiring the corresponding action knowledge to fill the gaps in a specific task sounds more practical.

There are more and more open-source knowledge resources being available including knowledge bases, ontologies, household appliance manuals, etc. Such open knowledge provides a new opportunity for intelligent robots to dynamically acquire the missing action knowledge as long as the knowledge is translated and accumulated into the action model of a robot. Nonetheless, the open knowledge is usually unformalized. For example, the knowledge in the Open Mind Indoor Common Sense (OMICS)[1] database, which is employed as the main source of action knowledge in this paper, is semi-structured meaning that most parts of it are in natural language. The challenge of translating open knowledge lies in the formalization of natural language [Chen *et al.*, 2012].

[1]http://commonsense.media.mit.edu

Recent research provides a number of effective approaches, so called *semantic parsing*, to translating natural language into formal expressions [Zettlemoyer and Collins, 2005; Wong and Mooney, 2007; Kwiatkowski *et al.*, 2011]. In this paper we adopt the semantic parser proposed by [Zettlemoyer and Collins, 2005] to formalize the action knowledge in OMICS. Previous works had their focuses on the semantic parsing for question answering (QA) domain in which the semantics inferences were monotonic. Since in our case the action reasoning is non-monotonic, it is necessary to propose a new semantic representation to capture the non-monotonic aspect of action knowledge. At the meanwhile, the proposed representation has a good interface to the semantics of action knowledge, which facilitates the parsing.

The rest of this paper is organized as follows: Section 2 will give an overview our action knowledge translation. Related works will be discussed in Section 3. The proposed semantic representation and its conversion to the lower-level semantic-equivalent representations will be presented in Section 4. The technical approach to the action knowledge translation will be elaborated in Section 5. Finally, we will evaluate our action knowledge translation in Section 6 and conclude in Section 7.

2 Overview

In the OMICS project [Gupta and Kochenderfer, 2004], extensive collections of several kinds of commonsense knowledge were collected from Internet users. Several sentence templates were designed for capturing different categories of knowledge and provided to users to fill in. Then the complete sentences were examined by administrators and converted into tables whose elements are English sentences. At this point, there are 48 tables in OMICS capturing different sorts of knowledge, including *Help* table (each tuple mapping a user desire to a concrete task that might meet it), *Tasks* table (containing possible tasks in the indoor environment), *Steps* table (decomposing a task into its steps), and so on. Some of tables are so related that they have to be merged together as a joint table to represent the knowledge. For example, the *Tasks* table includes tuples of $\langle id,task \rangle$ and the *Steps* table includes tuples of $\langle id,taskid,num,step \rangle$. And the *taskid* elements in *Steps* table correspond to the *id* elements in *Task* table. Hence the knowledege about the task *heat food in microwave* and its steps could be represented in a *Tasks/Steps* as shown in Table 1.

task	*num*	*step*
heat food in microwave	0	put food in microwaveable dish
heat food in microwave	1	put dish in microwave
heat food in microwave	2	select power level and time
heat food in microwave	3	press start
heat food in microwave	4	wait for food to heat up
heat food in microwave	5	remove dish from microwave

Table 1: A joint table *Tasks/Steps*.

The action knowledege is the main part of commonsense knowledge in OMICS. There are 17,481 pairs of a task and a set of its steps (Table 1 shows one of these pairs). Hence, OMICS database could provide sufficient quantities of action knowledge for indoor robots to solve unpredicted tasks from human. In this paper, we focus on translating the action knowledge of OMICS into its formal representation.

The underlying representation for action reasoning in our robot is in Answer Set Programming (ASP) language. ASP is a logic programming language with Prolog-like syntax under the stable model semantics, originally proposed by [Gelfond and Lifschitz, 1988]. An ASP program is a finite set of *ASP rules* of the form:

$$H \leftarrow p_1, \ldots, p_k, not\, q_1, \ldots, not\, q_m,$$

where $p_i (1 \leq i \leq k)$, and $q_j (1 \leq j \leq m)$ are literals, and H is either empty or a literal. A *literal* is a formula of the form a or $\neg a$, where a is an atom. If H is empty, this ASP rule is also called a *constraint*. An ASP rule consisting of only H is called a *fact*. There are two kinds of negations in ASP, the classical negation $\neg$ and non-classical one not, i.e., *negation as failure*. The meaning of the formula $not\, a$ could be interpreted that a is not derivable from the program. Similarly, a constraint $\leftarrow p_1, \ldots, p_k$ specifies that $p_1, \ldots, p_k$ cannot be jointly derived from the program. ASP provides the non-monotonic mechanism of action reasoning for our robot.

Our translation from the action knowledge in OMICS to the ASP rules is not immediate. Since a piece of action knowledge, even its elements (i.e., English sentences), will be associated with multiple ASP rules, the direct translation is complex and implicit. Therefore we propose a new semantic representation for the action knowledge, which is called Action Knowledge Representation Structure (AKRS). This representation has a good interface to the ASP rules, which means that it could be semantic-equivalently converted into some ASP rules. Our AKRS translation is implemented by adapting the existing approach [Zettlemoyer and Collins, 2005], which takes the Combinatory Categorial Grammar (CCG) [Steedman, 1996; 2000] as the underlying parsing formalism. The advantage of this semantic parser is that it is capable of not only formalizing the natural language but also inducing automatically the semantic lexicon from the annotated data.

3 Related Work

Building systems being able to acquire open knowledge has recently received some attentions. [Kunze *et al.*, 2010] present a robotic system that tranlates the OMICS knowledge into a formal representation. [Tenorth *et al.*, 2010] propose extracting the action knowledge from the natural language instructions from the World Wide Web. [Chen *et al.*, 2010] demonstrate the high-level cognitive functions for a robot to acquire knowledge from human-robot dialog. [Chen *et al.*, 2012] propose a whole architecture for identifying knowledge gaps, searching for the missing knowledge, translating and integrating them for action reasoning. However, the approaches to open knowledge formalization in these works are limited in the usage of some syntactic patterns for translation or their manual construction for the natural language semantic lexicon. Unlike previous works, in this paper, our work improves the open knowledge formalization in terms of automatically learning from some annotated data to semantically parse the open knowledge.

There have been a number of approaches to semantic parsing. These previous works have focused on various semantic representations, including the λ-calculus representations [Zettlemoyer and Collins, 2005], the one based on Prolog [Wong and Mooney, 2007], and the robot control language [Matuszek *et al.*, 2012]. The work most simliar to ours is [Baral and Gonzalez, 2011], in which the puzzles in natural language are translated into ASP language. Although we share the same underlying formalism of semantic parsing (i.e., CCG) and the same forms of output semantics (i.e., in ASP language), what distinguish us from them are our application which is oriented to action reasoning and our way to the final action semantics through the intermediate representation AKRS. There are also other works on semantic parsing based on CCG [Zettlemoyer and Collins, 2007; Lu *et al.*, 2008; Zettlemoyer and Collins, 2009; Kwiatkowski *et al.*, 2010; 2011]. However they focused on how to learn the semantic parser instead of formalizing the action knowledge for indoor robots. In this paper our contribution is not in the improvement for state of the art of semantic parsing, but rather in the novel exploration of action knowledge translation with semantic parsing.

4 Modeling Action Knowledge

As proposed in [Chen *et al.*, 2012], we classify the action knowledge for indoor robots into the *functional* knowledge and the *procedural* knowledge. The functional knowlege describes the effects of a task and the procedural knowledge expresses the steps of how to accomplish a task. Most of the action knowledge in OMICS is procedural as shown as Table 1. We follow the architecture proposed by [Chen *et al.*, 2012] to enable the mixed reasoning of functional knowledge and procedural knowledge. Given a model $M = \langle A, T, O, F, P \rangle$, where A is a action model in which the preconditions and effects of a primitive action of a robot are defined, T is a set of tasks to be accomplished, O is a set of facts having been observed, and F and P are respectively the functional knowledge and procedural knowledge having been obtained, the action reasoning on M is cast to find answer sets of an ASP program M^{Π}. Based on this architecture, the action knowledge translated into ASP rules could be non-monotonically reasoned about by our robot to enhance its capability of accomplishing tasks.

4.1 Action Knowledge Representation Structure

The Action Knowledge Representation Structure is designed to be an intermediate for the translation from action knowledge to its semantically equivalent ASP rules. This representation has a direct conversion to ASP rules. While our underlying reasoning mechanism of ASP is non-monotonic, AKRS is endowed with the ability to capture non-monotonic aspects of action knowledge.

The complete syntax definition of AKRS is listed in Appendix A. The form of AKRSs is Lisp-like. Some keywords are predefined to identify the types of AKRSs and to indicate the attributes in them. Two main types of AKRSs are (:prock T ψ) denoting the procedural knowledge about task T and its steps ψ, and (:funck T ϕ) denoting the functional knowledge about the effects ϕ of task T. The predicates and fluents of AKRS are used to describe the properties of objects. They are the atomic formulae. The complex formulae include the negation, conjunction, and disjunction. A "procedure" of AKRS could be an action, a task, a test, a sequence, non-deterministic choice, a conditional, or a loop. For example, an AKRS action of the form (:act *move* :params (X) :conds (:pred *room* X)) would be interpreted as an action *move* whose destination is a room. The ":params" section of an AKRS action is its arguments. And the ":conds" section is the conditions that action's arguments should satisfy. In the *move* case, the AKRS predicate (:pred *room* X) constrains the argument X to be of *room* type. The same interpretation applies to an AKRS task. Figure 1 illustrates the AKRS of the action knowledge shown as in Table 1.

```
(:prock (:task heat :params (X Y)
            :conds (and (:pred food X)
                (:pred microwave_oven Y)))
(:seq
    (:task place :params (X Y)
            :conds (and (:pred food X)
                    (:pred microwaveable Y)
                    (:pred dish Y)))
    (:task place :params (X Y)
            :conds (and (:pred dish X)
                    (:pred microwave_oven Y)))
    (:seq
            (:task set :params (X) :conds (:pred power X))
            (:task set :params (X) :conds (:pred timer X)))
    (:task press :params (X) :conds (:pred start_button X))
    (:do (:task wait :params ())
            :until (and (:pred food X) (:fluent heated X)))
    (:task take_out :params (X Y)
            :conds (and (:pred dish X)
                    (:pred microwave_oven Y)))))
```

Figure 1: The AKRS of the action knowledge as shown in Table 1.

4.2 ASP Conversion

In this section, we will demonstrate how an AKRS be converted into its semantic-equivalent ASP rules.

An ASP predicate or fluent translated from an AKRS s is denoted as $\mathcal{T}_t(s)$. If s = (:pred *name* $term_0 \ldots term_n$), then $\mathcal{T}_t(s) = name(term_0, \ldots, term_n)$. If s = (:fluent *name* $term_0 \ldots term_n$), then $\mathcal{T}_t(s) = holds(name(term_0, \ldots, term_n), t)$. t is the meta-variable ranging over time names $\{0, 1, \ldots, d\}$. The conversion for AKRS procedures is shown as follows:

- If s = (:test ϕ) where ϕ is a AKRS formula, let p_s be the index of s and $\bigvee_{j=1}^{m}(\bigwedge_{i=1}^{o^j} f_i^j \wedge \bigwedge_{i=o^j+1}^{n^j} \neg f_i^j)$ a DNF of ϕ where fs are either predicates or fluents, then s is translated to the following ASP rules ($1 \le j \le m$):
$$proc(p_s, t_1, t_2) \leftarrow \bigwedge_{1\le i\le o^j} \mathcal{T}_{t_1}(f_i^j), \bigwedge_{o^j< i\le n^j} \neg\mathcal{T}_{t_1}(f_i^j),$$
$$t_1 \le t_2.$$
- If s = (:act *name* :params ($X_0 \ldots X_n$) :conds ϕ), then s is translated to the following ASP rules ($1 \le j \le m$):
$$proc(p_s, t_1, t_2) \leftarrow \bigwedge_{1\le i\le o^j} \mathcal{T}_{t_1}(f_i^j), \bigwedge_{o^1< i\le n^j} \neg\mathcal{T}_{t_1}(f_i^j),$$
$$name(\tau, X_0, \ldots, X_n), occurs(\tau, t_1), t_1 \le t_2.$$
- If s = (:task *name* :params ($X_0 \ldots X_n$) :conds ϕ), then s is translated to the following ASP rules ($1 \le j \le m$):
$$proc(p_s, t_1, t_2) \leftarrow \bigwedge_{1\le i\le o^j} \mathcal{T}_{t_1}(f_i^j), \bigwedge_{o^1< i\le n^j} \neg\mathcal{T}_{t_1}(f_i^j),$$
$$name(\tau, X_0, \ldots, X_n), completed(\tau, t_1, t_2), t_1 \le t_2.$$
- If s = (:seq $s_1 \ldots s_m$), where $s_i (1 \le i \le m)$ are procedures, then we translate each s_i to corresponding ASP rules, with the addition of the following ASP rule:
$$proc(p_s, t_1, t_2) \leftarrow proc(p_{s_1}, t_1, t_{n^1}),$$
$$proc(p_{s_2}, t_{n^1}, t_{n^2}), \ldots, proc(p_{s_m}, t_{n^{m-1}}, t_2),$$
$$t_1 \le t_{n^1}, t_{n^1} \le t_{n^2}, \ldots, t_{n^{m-1}} \le t_2.$$
- If s = (:choice $s_1 \ldots s_m$), where $s_i (1 \le i \le m)$ are procedures, then we translate each s_i to corresponding ASP rules, with the addition of the following ASP rules ($1 \le i \le m$):
$$proc(p_s, t_1, t_2) \leftarrow proc(p_{s_i}, t_1, t_2), t_1 \le t_2.$$
- If s = (:if ϕ :then s_1), where s_1 is procedure and ϕ is an AKRS formula, then we translate the procedure (:seq (:test ϕ) s_1) (indexed as p'_{s_1}) to corresponding ASP rules, with the addition of the following ASP rule:
$$proc(p_s, t_1, t_2) \leftarrow proc(p'_{s_1}, t_1, t_2), t_1 \le t_2.$$
- If s = (:if ϕ :then s_1 :else s_2), where s_1 and s_2 are procedures and ϕ is an AKRS formula, then we translate the procedures (:seq (:test ϕ) s_1) (indexed as p'_{s_1}) and (:seq (:test (neg ϕ)) s_2) (indexed as p'_{s_2}) to corresponding ASP rules, with the addition of the following ASP rules:
$$proc(p_s, t_1, t_2) \leftarrow proc(p'_{s_1}, t_1, t_2), t_1 \le t_2.$$
$$proc(p_s, t_1, t_2) \leftarrow proc(p'_{s_2}, t_1, t_2), t_1 \le t_2.$$

- If s = (:while ϕ :do s_1), where s_1 is a procedure and ϕ is an AKRS formula, then we translate the procedures (:test ϕ) (indexed as p_ϕ) and s_1 to corresponding ASP rules, with the addition of the following ASP rules:

$$proc(p_s, t_1, t_2) \leftarrow not\, proc(p_\phi, t_1, t_2), t_1 \leq t_2.$$
$$proc(p_s, t_1, t_2) \leftarrow proc(p_{s_1}, t_1, t'), proc(p_s, t', t_2),$$
$$t_1 \leq t', t' \leq t_2.$$

- If s = (:do s_1 :until ϕ), where s_1 is a procedure and ϕ is an AKRS formula, then we translate the procedures (:seq s_1 (:while (neg ϕ) :do s_1)) (indexed as p'_{s_1}) to corresponding ASP rules, with the addition of the following ASP rule:

$$proc(p_s, t_1, t_2) \leftarrow proc(p'_{s_1}, t_1, t_2), t_1 \leq t_2.$$

Finally, for each piece of procedural knowledge of the form (:prock (:task *name* :params ($X_0 \ldots X_n$) :conds ϕ) ψ), where ψ is an AKRS procedure, let p_ψ be the index of ψ and $\bigvee_{j=1}^{m}(\bigwedge_{i=1}^{o^j} f_i^j \wedge \bigwedge_{i=o^j+1}^{n^j} \neg f_i^j)$ a DNF of ϕ, it is converted into the following ASP rule:

$$completed(\tau, t_1, t_2) \leftarrow \bigwedge_{1 \leq i \leq o^j} \mathcal{T}_{t_1}(f_i^j), \bigwedge_{o^1 < i \leq n^j} \neg \mathcal{T}_{t_1}(f_i^j),$$
$$name(\tau, X_0, \ldots, X_n), proc(p_\psi, t_1, t_2), t_1 \leq t_2.$$

And for each piece of functional knowledge of the form (:funck (:task *name* :params ($X_0 \ldots X_n$) :conds ϕ) φ), where ϕ and φ are AKRS formulae, let $\bigvee_{j=1}^{m}(\bigwedge_{i=1}^{o^j} f_i^j \wedge \bigwedge_{i=o^j+1}^{n^j} \neg f_i^j)$ be a DNF of the AKRS conjunctive formula (and ϕ φ), it is converted into the following ASP rules ($1 \leq j \leq m$):

$$completed(\tau, t_1, t_2) \leftarrow \bigwedge_{1 \leq i \leq o^j} \mathcal{T}_t(f_i^j), \bigwedge_{o^j < i \leq n^j} \neg \mathcal{T}_t(f_i^j),$$
$$name(\tau, X_0, \ldots, X_n), t_1 \leq t, t \leq t_2.$$

The example result of ASP conversion for an AKRS

```
(:prock (:task get :params (X Y)
              :conds (and (:pred food X)
                          (:pred refrigerator Y)))
(:seq
      (:task open :params (X)
              :conds (:pred refrigerator_door X))
      (:task pick_up :params (X)
              :conds (:pred food X)))
```

is shown as follows:

$$completed(\tau_0, t_1, t_2) \leftarrow food(X), refrigerator(Y),$$
$$get(\tau_0, X, Y), proc(p_0, t_1, t_2), t_1 \leq t_2.$$
$$proc(p_0, t_1, t_2) \leftarrow proc(p_1, t_1, t_2), proc(p_2, t_2, t_3),$$
$$t_1 \leq t_2, t_2 \leq t_3.$$
$$proc(p_1, t_1, t_2) \leftarrow refrigerator\_door(X),$$
$$open(\tau_1, X), completed(\tau_1, t_1, t_2), t_1 \leq t_2.$$
$$proc(p_2, t_1, t_2) \leftarrow food(X), pick\_up(\tau_2, X),$$
$$completed(\tau_2, t_1, t_2), t_1 \leq t_2.$$

5 Action Knowledge Translation

For the reason of the explicit conversion (as shown in the previous section) from AKRSs to ASP rules, we now turn to formalizing the action knowledge into AKRSs. The action knowledge translation on OMICS consists of two phases: the semantic parsing and the AKRS combination. The action knowledge in OMICS as shown in Table 1 is the procedural knowledge. We could combine all the AKRSs of elememts in *Tasks/Steps* table by using the following template:

(:prock T (:seq $[S_0, \ldots, S_n]$))

where T is the AKRS of English sentence in *task* column of *Tasks/Steps* Table and $[S_0, \ldots, S_n]$ is a list (orderd by the number in *num* column) of AKRS results of *step* column. Figure 1 shows an AKRS results from our action knowledge translation.

5.1 Semantic Parsing

Our approach to semantic parsing is based on CCG. CCG is a linguistic formalism that provides a tight interface between natural language syntax and its semantics and has beed used to model a wide range of language phenomena. The core of any CCG is the semantic lexicon. In the lexicon, a word with its syntactic category and semantic form is defined. For example, the lexicon for AKRS would be as follows:

button := NP : $\lambda x.$*(:pred button x)*
press := S/NP : $\lambda f.$(:task *press* :params (X) :conds $f@X$)
if := $(S/S)/S$: $\lambda f.\lambda g.$(:if f :then g)

The syntactic categories in CCG could be either *primitive* (e.g., NP or S), or *complex* (of the form A/B or $A\backslash B$). In the above lexicon *button* has the syntactic category NP which stands for the linguistic notions of noun-phrase. In our case the semantic forms are the AKRSs. For example, the semantic form of *press* is a semi-AKRS task of which the conditions are λ-abstracted to a λ-application $f@X$.

A CCG also has a set of *combinatory rules* to combine syntactic categories ad semantic forms which are adjacent in a string. The basic combinatory rules are the application rules:

$$X/Y : f \quad Y : g \Rightarrow X : f@g \qquad (>)$$
$$Y : g \quad X\backslash Y : f \Rightarrow X : f@g \qquad (<)$$

The first rule is the forward ($>$) application indicating that a syntactic category X/Y with a semantic form f can be combined with a syntactic category Y with a semantic form g to generate a new syntactic category X whose semantic form is formed by λ-applying f to g. Symmetrically the second rule, backward ($<$) application, generates a new syntactic category and semantic form by applying the right one to its left. For example, in the above lexicon, *press* can be combined with *button* to generate a phrase *press button* with syntactic category S and semantic form (:task *press* :params (X) :conds (:pred *button* X)).

The other rules, like the composition rules, and the type-raising rules, give the allowance of an unrestricted notion of constituency which is useful for dealing with the long-range dependencies inherent in certain constructions, such as coordination [Steedman, 2000]. In our semantic parser the above

heat	food	in	microwave
$(S/AP)/NP$	NP	AP/NP	NP
$\lambda f.\lambda g.$(:task *heat* :params $(X\ Y)$:conds (and $f@X\ g@Y$))	$\lambda x.$(:pred *food* x)	$\lambda x.x$	$\lambda x.$(:pred *microwave_oven* x)

S/AP	AP
$\lambda g.$(:task *heat* :params $(X\ Y)$:conds (and (:pred *food* X) $g@Y$))	$\lambda x.$(:pred *microwave_oven* x)

S

(:task *heat* :params $(X\ Y)$:conds (and (:pred *food* X) (:pred *microwave_oven* Y)))

Figure 2: An example derivation of semantic parsing for *heat food in microwave*.

combinatory rules are all employed. One example of the CCG derivation for the natural language sentence *heat food in microwave* is illustrated in Figure 2.

However a CCG parser will generate a number of parses for each natural language sentences. Hence a conditional log-linear model is employed to select the best scored parses. It defines the joint probability of an AKRS z constructed with a syntactic parse y, given a sentence x:

$$P(y,z|x;\theta,\Lambda) = \frac{\exp[\theta \cdot f(x,y,z)]}{\sum_{(y',z')} \exp[\theta \cdot f(x,y',z')]}$$

where Λ is the semantic lexicon. $f(x,y,z)$ is a feature vector evaluating on the sub-structures within (x,y,z). It is parameterized by $\theta \in \mathbb{R}^d$. In this paper we make use of the lexical features. Each lexical feature counts the number of times that a lexical entry is used in y. With the probabilistic model, the semantic parsing then turn to compute:

$$\arg\max_z P(z|x;\theta,\Lambda) = \arg\max_z \sum_y P(y,z|x;\theta,\Lambda)$$

In this formalism the distribution over the syntactic parses y is modeled as a hidden variable. The sum over y can be calculated efficiently by using the dynamic programming (i.e., CKY-style) algorithms. In addition, the beam-search during parsing is employed, in which sub-derivations with low probability under some level are removed.

5.2 Semantic Lexicon Induction

To induce the semantic lexicon, the training examples, $\{(x_i,z_i) : i = 1,\ldots,n\}$, are assumed having been accessed. The task of semantic lexicon induction consists of generating possible lexical items that are generalized enough, and estimating the parameter vector θ over the training examples. [Zettlemoyer and Collins, 2005] gives a lexicon learning algorithm to fulfill this task.

Given an example pair (x_i,z_i), all possible lexical items will be generated first by a lexical item generation function GENLEX. Through picking out the lexical items used in the highest scored parse(s) leading z_i, a compact lexicon Λ_i for paring x_i is obtained. After collecting all compact lexicons, a complete lexicon $\Lambda = \cup_i \Lambda_i$ is used to estimate the parameters over the training examples. The objective of estimating on parameters is to maximize the log-likelihood of the training examples $\mathcal{L} = \sum_{i=1}^{n} \log P(z_i|x_i;\theta,\Lambda)$. Hence the parameters are updated in term of the following formula:

$$\frac{\partial \mathcal{L}}{\partial \theta_j} = \sum_{i=1}^{n} E_{p(y|x_i,z_i;\theta,\Lambda)}[f_j(x_i,y,z_i)] - \sum_{i=1}^{n} E_{p(y,z|x_i;\theta,\Lambda)}[f_j(x_i,y,z)]$$

It involves calculating the expectations of features under the distributions $p(y|x_i,z_i;\theta,\Lambda)$ and $p(y,z|x_i;\theta,\Lambda)$. This calculation again can be reached by using a dynamic programming algorithm, a variant of the inside-outside algorithm. A stochastic gradient ascent algorithm is employed to maximize the likelihood.

The core of the learning algorithm proposed by [Zettlemoyer and Collins, 2005] is the lexical item generation function GENLEX. To adapt their learning algorithm to our action knowledge translation, it needs a new GENLEX function that could be able to generate the lexical items for parsing AKRSs. The GENLEX function consists of a set of trigger rules where each rule takes as input a semantic form and outputs the possible syntactic categories with corresponding semantic forms. The GENLEX takes as input the semantic form z (in our case it is AKRS), then calls each trigger rule to produce all the category outputs of z, and finally generate the lexical items by combining each category output with all possible sub-string of the sentence x. Some of our trigger rules for the new GENLEX function are presented in Table 2.

6 Experiment

The semantic parsing for natural language sentences is the core of our action knowlege translation. Once the sentences are correctly parsed into the AKRSs, through the template described in Section 5, the action knowledge is correctly translated. Therefore, our experimental evaluation mainly focused on the performance of the semantic parser.

There are 17,481 pairs of a task and a set of its steps in OMICS. These tasks and steps consist of 28,846 literally unique natural language sentences. To evaluate the semantic parser, sentences needed to be annotated with their AKRSs. The quantity of sentences is tremendous and the annotation would be exhausting. Hence we picked out 471 sentences that are in the action knowledge most commonly appearing in the domain, such as the knowledge about task *heat food in microwave*. These sentences were annotated manually and were

Trigger Rules		Examples	
Input AKRS	Output Category	Input	Output
a task named *n* with no parameter	S : (:task *n* :params ())	(:task *wait* :params ())	S :(:task *wait* :params ())
a task named *n* with one parameter	S/NP : λf.(:task *n* :params (X) :conds $f@X$)	(:task *press* :params (X) :conds (:pred *button* X))	S/NP : λf.(:task *press* :params (X) :conds $f@X$)
a predicate named *p* with one term	NP : λx.(:pred *p* x)	(:pred food X)	NP : λx.(:pred *food* x)
a predicate named *p* with one term	NP/NP : $\lambda f.\lambda x.$ (and (:pred *p* x) $f@x$)	(:pred *microwaveable* X)	NP/NP : $\lambda f.\lambda x.$ (and (:pred *microwaveable* x) $f@x$)

Table 2: Some examples of trigger rules in GENLEX used for generating the AKRS lexical items.

randomly split into 330 training examples and 141 test examples. We evaluate the performance of semantic parser by giving its *precision* (percentage of parsed AKRSs that were correct), *recall* (percentage of test examples with correctly parsed AKRSs) and *F1-measure* (harmonic mean of precision and recall). The sentences were correctly parsed only if their parsed AKRSs were totally matched the annotated ones. In addition, we adopted the two-pass parsing strategy meaning that if the sentence fails to parse, it would be parsed again by deleting some words at some cost. This strategy is for the purpose of dealing with the words that were never observed in the training examples. The experimental results are shown in Table 3.

	Test set	Training set
Precision	64.84%	87.42%
Recall	41.84%	84.24%
F1	50.86%	85.80%

Table 3: Performance of the semantic parser for AKRS.

There were 64.84% of sentences that were parsed and correctly translated into AKRSs, meaning that our parser could translate the action knowledege into AKRSs, then converted into ASP representations. However, there were only 64.54% of unseen sentences that had returned AKRSs. Compared to the results on the training set, our parser performed worse on the unseen examples. The main reason is that the sentences collected in OMICS are open ended. They are full of ellipses and the same meaning could be expressed in various ways. This would result in the sparsity of training examples, especially our small amount of trainig sentences. However, this is our first step to the wide coverage formalization of action knowledge from an open knowledge database. More annotated examples would be added to the training. Our approach to translating action knowledge is still promising.

7 Conclusion

In this paper, we presented an approach to translating the action knowledge into the underlying semantic representation of cognitive robots. To facilitate the translation we proposed a new semantic representation for action knowledge, i.e., Action Knowledge Representation Structure. This representation captures most phenomena in action knowledge and has been shown its direct and explicit conversion into ASP language. Based on this semantic representation, we employed an existing semantic parser to formalize the natural language elements in the action knowledge. The results then are combined together to obtain the whole semantic representation of the action knowledge. The experimental evaluation on our action knowledge translation implies that there are lots of rooms for improvement.

Acknowledgments

This work is supported by the National Hi-Tech Project of China under grant 2008AA01Z150 and the Natural Science Foundation of China under grant 60745002 and 61175057, as well as the USTC Key Direction Project and the USTC 985 Project. We are grateful to the anonymous reviewers for their helpful comments on the earlier version of this paper.

A BNF Definition of AKRS Syntax

⟨*term*⟩ ::= ⟨*constant*⟩ | ⟨*variable*⟩
⟨*predicate*⟩ ::= (:pred ⟨*predicate name*⟩ ⟨*term*⟩*)
⟨*fluent*⟩ ::= (:fluent ⟨*fluent name*⟩ ⟨*term*⟩*)
⟨*formula*⟩ ::= ⟨*predicate*⟩ | ⟨*fluent*⟩ | (neg ⟨*formula*⟩) | (and ⟨*formula*⟩$^+$) | (or ⟨*formula*⟩$^+$)
⟨*conditions*⟩ ::= [:conds ⟨*formula*⟩]
⟨*action*⟩ ::= (:act ⟨*action name*⟩ :params (⟨*variable*⟩*) ⟨*conditions*⟩)
⟨*task*⟩ ::= (:task ⟨*task name*⟩ :params (⟨*variable*⟩*) ⟨*conditions*⟩)
⟨*test*⟩ ::= (:test ⟨*formula*⟩)
⟨*sequence*⟩ ::= (:seq ⟨*procedure*⟩ ⟨*procedure*⟩$^+$)
⟨*choice*⟩ ::= (:choice ⟨*procedure*⟩ ⟨*procedure*⟩$^+$)
⟨*conditional*⟩ ::= (:if ⟨*formula*⟩ :then ⟨*procedure*⟩) | (:if ⟨*formula*⟩ :then ⟨*procedure*⟩ :else ⟨*procedure*⟩)
⟨*loop*⟩ ::= (:while ⟨*formula*⟩ :do ⟨*procedure*⟩) | (:do ⟨*procedure*⟩ :until ⟨*formula*⟩)
⟨*procedure*⟩ ::= ⟨*action*⟩ | ⟨*task*⟩ | ⟨*test*⟩ | ⟨*sequence*⟩ | ⟨*choice*⟩ | ⟨*conditional*⟩ | ⟨*loop*⟩
⟨*proc_knowledge*⟩ ::= (:prock ⟨*task*⟩ ⟨*procedure*⟩)
⟨*func_knowledge*⟩ ::= (:funck ⟨*task*⟩ ⟨*formula*⟩)

References

[Baral and Gonzalez, 2011] Chitta Baral and Marcos Alvarez Gonzalez. Using Inverse Lambda and Generalization to Translate English to Formal Languages. In *Proceedings of the Ninth International Conference on Computational Semantics*, number 2000, 2011.

[Chen *et al.*, 2010] Xiaoping Chen, Jianmin Ji, Jiehui Jiang, Guoqiang Jin, Feng Wang, and Jiongkun Xie. Developing High-level Cognitive Functions for Service Robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 989–996, Toronto, Canada, 2010. IFAAMAS.

[Chen *et al.*, 2012] Xiaoping Chen, Jiongkun Xie, Jianmin Ji, and Zhiqiang Sui. Toward Open Knowledge Enabling for Human-Robot Interaction. *Journal of Human-Robot Interaction (JHRI)*, 1(2):100–117, 2012.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic programming (ICLP-88)*, pages 1070–1080, 1988.

[Gupta and Kochenderfer, 2004] Rakesh Gupta and Mykel J. Kochenderfer. Common Sense Data Acquisition for Indoor Mobile Robots. In *Proceedings of the 19th National Conference on Artificial Intelligence and the 16th Conference on Innovative Applications of Artificial Intelligence (AAAI-04)*, volume 300, pages 605–610, San Jose, California, USA, 2004. AAAI.

[Kunze *et al.*, 2010] Lars Kunze, Moritz Tenorth, and Michael Beetz. Putting People's Common Sense into Knowledge Bases of Household Robots. In *Proceedings of KI 2010: Advances in Artificial Intelligence, 33rd Annual German Conference on AI*, pages 151–159, Karlsruhe, Germany, 2010. Springer.

[Kwiatkowski *et al.*, 2010] Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP-10)*, pages 1223–1233, MIT Stata Center, Massachusetts, USA, 2010. ACL.

[Kwiatkowski *et al.*, 2011] Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. Lexical Generalization in CCG Grammar Induction for Semantic Parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-11)*, pages 1512–1523, John McIntyre Conference Centre, Edinburgh, UK, 2011. ACL.

[Lu *et al.*, 2008] Wei Lu, Hwee Tou Ng, Wee Sun Lee, and Luke S. Zettlemoyer. A Generative Model for Parsing Natural Language to Meaning Representations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*, pages 783–792, Honolulu, Hawaii, USA, 2008. ACL.

[Matuszek *et al.*, 2012] Cynthia Matuszek, Evan Herbst, Luke S. Zettlemoyer, and Dieter Fox. Learning to Parse Natural Language Commands to a Robot Control System. In *Proceedings of the 13th International Symposium on Experimental Robotics (ISER-12)*, 2012.

[Steedman, 1996] Mark Steedman. *Surface Structure and Interpretation*. MIT Press, 1996.

[Steedman, 2000] Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

[Tenorth *et al.*, 2010] Moritz Tenorth, Daniel Nyga, and Michael Beetz. Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA-10)*, pages 1486–1491, Anchorage, Alaska, USA, 2010. IEEE.

[Wong and Mooney, 2007] Yuk Wah Wong and Raymond J. Mooney. Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-07)*, number June, pages 960–967, Prague, Czech Republic, 2007. ACL.

[Zettlemoyer and Collins, 2005] Luke S. Zettlemoyer and Michael Collins. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI-05)*, number x, pages 658–666, Edinburgh, Scotland, 2005. AUAI Press.

[Zettlemoyer and Collins, 2007] Luke S. Zettlemoyer and Michael Collins. Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL-07)*, pages 678–687, Prague, Czech Republic, 2007. ACL.

[Zettlemoyer and Collins, 2009] Luke S. Zettlemoyer and Michael Collins. Learning Context-Dependent Mappings from Sentences to Logical Form. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL/AFNLP-09)*, pages 976–984, Singapore, 2009. ACL.