Action-Centric Probabilistic Programming*

Vaishak Belle
KU Leuven
Belgium
vaishak@cs.kuleuven.be

Abstract

Probabilistic models are dominant in data management, but are often described in a combination of natural and mathematical language, and algorithms need to be engineered for an individual application. Probabilistic programming languages have received considerable attention recently as they contribute formal languages that enable re-use and clarity in the problem specification. Our interest in this paper is the development of a programming language, called ALLEGRO, that is in service of the high-level control of an autonomous system, such a mobile robot. The language's mathematical foundations rests of Bayesian conditioning, and its programs are interpreted over a sophisticated logical theory of actions supplemented with user-defined axioms. While many existing probabilistic programming languages can easily be shown to capture stochastic transitions, we argue that an action-centric probabilistic programming has many valuable properties. As a modeling language, ALLEGRO can be seen as a basis for relating high-level control specifications, including plans with loops, on the one hand, and high-level probabilistic models, such as relational graphical models, on the other.

1 Introduction

Probabilistic models are dominant in data management, but are often described in a combination of natural and mathematical language, and algorithms need to be engineered for an individual application. Probabilistic programming languages have received considerable attention recently as they contribute formal languages that enable re-use and clarity in the problem specification (Raedt, Kimmig, and Toivonen 2007; Milch et al. 2005; Goodman et al. 2008; Pfeffer 2001). The underlying technical idea in probabilistic programming languages is to extend (traditional) programming languages with primitives for random choices to enable the modeling of, and learning with, structured probability distributions. The end goal, then, is to support *generic inference* techniques that can circumvent the need to engineer specialized algorithms.

Our interest in this paper is the development of a programming paradigm that is in service of the high-level control of an autonomous system, such a mobile robot. Clearly, such a paradigm would need to take into account the probabilistic beliefs of the system, and moreover, when operating in an inherently dynamic world, the system would also need to know the actions (*e.g.,* pick up a block, clear the table) at its disposal, the effects of executing such actions, and the ways in which it can learn more about the world by making observations of one form or another. Note, for example, when these actions are of a nondeterministic nature, that is, when defined over a stochastic transition model, there is an implicit notion of random choice, except that on executing the action, the system is not aware about the exact nature of the outcome and has to reason about all of them. In this sense, what we are after is much like a probabilistic program, except that beliefs need to be *tracked* in an *online* fashion: as the program executes, actions are carried out in the world leading to new beliefs.

In this paper, we report on the development of such a language ALLEGRO, also referred to a belief-based program (in the knowledge representation literature). The language's mathematical foundations rests of Bayesian conditioning, and its programs are interpreted over a sophisticated logical theory of actions supplemented with user-defined axioms (and constraints).

Of course, existing probabilistic programming languages can easily be shown to capture stochastic transitions (Goodman et al. 2008; Nitti, Laet, and Raedt 2013), via a planning-as-inference implementation. So, why is there a need for an alternative proposal? Here are a few noteworthy benefits:

- Actions are first-class citizens in our approach. This means that:
  - program execution corresponds to the agent doing things in the world, and it can branch on observations made about the world like a conditional plan; and
  - complex actions can be further defined by means of constructs such as recursion and sequence allowing for action repetition and modularity.

*This work is an abridged version of a paper entitled "ALLEGRO: Belief-based Programming in Stochastic Dynamical Domains," co-authored with Hector J. Levesque, that appeared in the proceedings of IJCAI, 2015. In Section 1, moreover, an attempt is made to emphasize the connections between statistical relational learning and probabilistic programming, on the one hand, and high-level programs of the sort considered in this work, on the other. The author is supported by the Research Foundation-Flanders (FWO-Vlaanderen).

- A long-standing concern is ensure that AI systems, such as robots, behave reliably. To date, many powerful algorithms have been studied for the correctness of temporal behaviours, such as safety, liveness and fairness (Baier and Katoen 2008), which can be lifted to action-centric programming languages (Claßen et al. 2014).
- Actions often depend on complicated features of the world: for example, a pick up action assumes the robot's arms are free, and a sensor's accuracy may depend on the temperature. A rich logical theory of action makes the modeling of such features straightforward by allowing logical connectives.
- Most significantly, ALLEGRO is based on a general probabilistic logic (Bacchus, Halpern, and Levesque 1999) that allows discrete and continuous random variables (Belle and Levesque 2013), and in that sense, rich representations of structured probability distributions, such as statistical relational learning languages, can be further incorporated.

The origins of ALLEGRO can be traced to two influential proposals: the action-centric high-level programming language GOLOG (Levesque et al. 1997), and knowledge-based programming (Fagin et al. 1995; Reiter 2001a), where program execution is conditioned on beliefs. However, these accounts did not address probabilistic beliefs and stochastic transitions, which ALLEGRO does. More recently, we developed a system called PREGO (Belle and Levesque 2014), which is simply an inference engine, and not a programming formalism (Belle and Levesque 2015). In this sense, ALLEGRO can be seen as the realization of ALGOL-like features in PREGO.

In terms of organization, we introduce ALLEGRO, before turning to related work and conclusions. We refer readers to (Belle and Levesque 2015) for discussions on ALLEGRO's mathematical foundations and empirical behavior.

2 The ALLEGRO System

The ALLEGRO system[1] is a programming language with a simple LISP-like syntax.[2] In this section, we discuss domain axiomatizations, introduce the grammar of ALLEGRO programs, and discuss how ALLEGRO is used.

Domain Axiomatization

When modeling a domain, the user provides a *basic action theory* (or BAT) (Reiter 2001a) to describe the domain in terms of fluents, actions and sensors, possibly characterized by discrete and continuous distributions. As a small example, Figure 2 shows a BAT for the domain illustrated in Figure 1, which can be read as follows:[3]

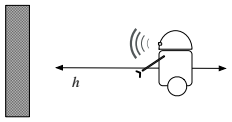

Figure 1: Robot moving towards a wall.

```
(define-fluents h)

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

(define-ss-exprs h
    (nfwd x y) '(max 0 (- h ,y))

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

(define-l-exprs
    (nfwd x y) '(GAUSSIAN ,y ,x 1.0)
    (sonar z) '(GAUSSIAN ,z h 4.0))
```

Figure 2: A BAT for the simple robot domain.

1. The domain has a single fluent `h`, the distance to the wall, whose initial value is taken by the agent to be drawn from a (continuous) uniform distribution on [2, 12].
2. The successor state axiom for `h` says that it is affected only by the `nfwd` action. The first argument of `nfwd` is the amount the agent intends to move, and the second is the amount that is actually moved (which determines how much the value of `h` is to be reduced).
3. The `alt` axiom is used to say that if an action of the form `(nfwd 2 2.79)` occurs, the agent will only know that `(nfwd 2 z)` happened for some value of `z`.
4. The likelihood axiom for `nfwd` says that the actual amount moved by `nfwd` will be expected by the agent to be centered (wrt a normal distribution) around the intended amount with a standard deviation of `1.0`.
5. The likelihood axiom for `sonar` says that sonar readings are noisy but will be expected to be centered around the true value of `h` with a standard deviation of `4.0`.

The idea here is that starting with some initial beliefs, executing `(sonar 5)` does not guarantee that the agent is actually `5` units from the wall, although it should serve to increase the robot's confidence in that fact. Analogously, performing a noisy move with an intended argument `2` means that the robot may end up moving (say) `2.79` units. Nevertheless, its degree of belief that it is closer to the wall should increase.

[1] Available at `vaishakbelle.com`

[2] ALLEGRO is realized in the RACKET dialect of the SCHEME family (`racket-lang.org`). We use RACKET arithmetic freely, such as the `max` function, as well as any other function that can be defined in RACKET, like `GAUSSIAN`. However, the technical development does not hinge on any feature unique to that language.

[3] For ease of exposition, we limit discussions in the following ways. First, we omit any mention of action preconditions here. Second, fluents are assumed to be real-valued, and moreover, only basic features of the language are explained in the paper. In general (Belle and Levesque 2014), fluent values can range over any set, BATs are not limited to any specific family of discrete/continuous distributions, and the language supports notions not usually seen in standard probabilistic formalisms, such as *contextual* likelihood axioms. (For example, if the floor is slippery, then the noise of a move action may be amplified.) All of these are fully realized in ALLEGRO.

Belief-based Programs

The BAT specification describes the noise in actions and sensors wrt the actual outcomes and values observed. But a robot executing a program need not know these outcomes and values. For this reason, the *primitive programs* of ALLEGRO are actions that suppress these parameters. So for the actions `(nfwd x y)` and `(sonar z)` appearing in a BAT, the primitive programs will be `(nfwd x)` and `(sonar)`.

The basic ALLEGRO language uses five program constructs:

prim	primitive programs;
(`begin` $prog_1 \ldots prog_n$)	sequence;
(`if` *form* $prog_1$ $prog_2$)	conditional;
(`let` $((var_1\ term_1) \ldots (var_n\ term_n))$ *prog*)	assignments;
(`until` *form prog*)	until loop.

Here *form* stands for formulas built from this grammar:

$$form ::= (\circ\ term_1\ term_2) \mid (\bullet\ form_1\ form_2) \mid (\texttt{not}\ form)$$

where $\circ \in \{<, >, =\}$ and $\bullet \in \{\texttt{and}, \texttt{or}\}$. Here *term* stands for terms built from the following grammar:

$$term ::= (\texttt{exp}\ term) \mid number \mid fluent \mid var \mid (\diamond\ term_1\ term_2) \mid (\texttt{if}\ form\ term_1\ term_2)$$

where $\diamond$ is any arithmetic operator (*e.g.,* + and -). The primary "epistemic" operator in ALLEGRO is `exp`: `(exp` *term*`)` refers to the *expected value* of *term*. (Reasoning about the expected value about a fluent allows the robot to monitor how it changes with sensing, for example.) The *degree of belief* in a formula *form* can then be defined in terms of `exp` as follows:

$$(\texttt{bel}\ form) \doteq (\texttt{exp}\ (\texttt{if}\ form\ \texttt{1.0}\ \texttt{0.0}))$$

For our purposes, it is convenient to also introduce `(conf` *term number*`)`, standing for the *degree of confidence* in the value of a term, as an abbreviation for:

$$(\texttt{bel}\ (\texttt{>}\ number\ (\texttt{abs}\ (\texttt{-}\ term\ (\texttt{exp}\ term)))))$$

For example, given a fluent `f` that is normally distributed, `(conf f .1)` is higher when the curve is narrower.

Usage

The ALLEGRO system allows programs to be used in three ways: in *online* mode − the system displays each primitive program as it occurs, and prompts the user to enter the sensing results; in *network* mode − the system is connected to a robot over TCP, the system sends primitive programs to the robot for execution, and the robot sends back the sensing data it obtains; and finally, in *offline* mode − the system generates ersatz sensing data according to the given error model. In all cases, the system begins in an initial belief state, and updates this belief state as it runs the program.

As a simple illustration, imagine the robot from Figure 2 would like to get within 2 and 6 units from the wall. It might proceed as follows: sharpen belief about current position (by sensing), (intend to) move by an appropriate amount, adjust beliefs for the noise of the move, and repeat these steps until the goal is achieved. This intuition is given as a program in Figure 3: here, `conf` is used to first become confident about `h` and then `exp` is used to determine the distance for getting

```
(until (> (bel (and (>= h 2) (<= h 6))) .8)
    (until (> (conf h .4) .8) (sonar))
    (let ((diff (- (exp h) 4)))
       (nfwd diff)))
```

Figure 3: A program to get between 2 and 6 units from the wall.

midway between 2 and 6 units. (An arbitrary threshold of `.8` is used everywhere wrt the robot's beliefs.) For an online execution of this program `prog` in ALLEGRO, we would do:

```
> (online-do prog)
Execute action: (sonar)
Enter sensed value: 4.1
Enter sensed value: 3.4
Execute action: (nfwd 1.0)
Enter sensed value: 3.9
Enter sensed value: 4.2
Execute action: (nfwd 0.0)
```

We see the robot first applying the sonar sensor, for which the user reports a value of `4.1`. After updating its beliefs for this observation, the robot is not as confident as required by `prog`, and so, a second sensing action is executed for which `3.4` is read. Then the robot attempts a (noisy) move, but its confidence degrades as a result. So these steps are repeated once more, after which the program terminates. On termination, the required property can be tested in ALLEGRO using:

```
> (bel (and (>= h 2) (<= h 6)))
0.8094620133032484
```

Semantics

The logical foundation of ALLEGRO is based on the *situation calculus* (Reiter 2001a). The interpretation of belief operators, in particular, is based on an extension of the situation calculus for *degrees of belief* and *noisy sensors* (Belle and Levesque 2013). The interpretation of programs is an extension to the online version of GOLOG (Sardina et al. 2004). For these details, and a formal result relating a sampling-based interpreter for ALLEGRO, on the one hand, and situation calculus basic action theories, on the other, we refer interested readers to (Belle and Levesque 2015).

3 Related Work and Discussions

We first relate ALLEGRO to high-level agent programming proposals, and then to planning proposals.

The ALLEGRO system is a programming model based on the situation calculus, and so is a new addition to the GOLOG family of high-level programming languages (Levesque et al. 1997; Sardina et al. 2004). In particular, it follows in the tradition of *knowledge-based* GOLOG with sensing in an online context (Reiter 2001b; Claßen and Lakemeyer 2006; Fan et al. 2012), but generalizes this in the sense of handling degrees of belief and probabilistic noise in the action model.

The GOLOG family has been previously extended for probabilistic nondeterminism, but there are significant differences. For example, in the PGOLOG model (Grosskreutz and Lakemeyer 2003), the outcome of a nondeterminism action is

immediately *observable* after doing the action, and continuous distributions are not handled. This is also true of the notable DTGOLOG approach (Boutilier et al. 2000) and its variants (Ferrein and Lakemeyer 2008). In this sense, the ALLEGRO model is more general where the agent cannot observe the outcomes and sensors are noisy. Moreover, these proposals do not represent beliefs explicitly, and so do not include a query language for reasoning about nested belief expressions.

Outside of the situation calculus, an alternative to GOLOG, called FLUX (Thielscher 2004; 2005), has been extended for knowledge-based programming with noisy effectors in (Thielscher 2001; Martin and Thielscher 2009); continuous probability distributions and nested belief terms are, however, not handled. The concept of knowledge-based programming was first introduced in (Fagin et al. 1995). These have been extended for Spohn-style [1988] ordinal functions in (Laverny and Lang 2005). For discussions on how high-level programming relates to standard agent programming (Shoham 1993), see (Sardina and Lespérance 2010).

Programs, broadly speaking, generalize sequential and tree-like plan structures, but can also be used to limit plan search (Baier, Fritz, and McIlraith 2007); we refer interested readers to (Lakemeyer and Levesque 2007) for discussions. There are, of course, many planning approaches in online contexts, including knowledge-based planning (Petrick and Bacchus 2004; Van Ditmarsch, Herzig, and De Lima 2007), decision-theoretic proposals (Puterman 1994; Kaelbling, Littman, and Cassandra 1998), corresponding relational abstractions (Sanner and Kersting 2010; Zamani et al. 2012), and belief-based planning (Kaelbling and Lozano-Pérez 2013). See (Lang and Zanuttini 2012) on viewing knowledge-based programs as plans.

In the same vein, let us reiterate that the BAT syntax is based on an inference system called PREGO (Belle and Levesque 2014), and from the viewpoint of a representation language, see that work for discussions on its expressiveness relative to other formalisms, such as probabilistic planning languages (Sanner 2011), action formalisms (Van Benthem, Gerbrandy, and Kooi 2009; Thielscher 2001; Iocchi et al. 2009), and related efforts on combining logic and probability (Bacchus 1990; Richardson and Domingos 2006). Notably, its support for continuous distributions and arbitrary successor state axioms makes it a tractable but expressive language.

In the context of probabilistic models, we remark that there are other realizations of program-based probabilistic behavior, such as *probabilistic programming* (Milch et al. 2005; Goodman et al. 2008). These are formal languages that provide program constructs for probabilistic inference. While they are expressive enough to capture dynamical probabilistic models (Nitti, Laet, and Raedt 2013), they belong to a tradition that is different from the GOLOG and FLUX families. For example, atomic programs in GOLOG are actions taken from a basic action theory whereas in (Milch et al. 2005), atomic constructs can be seen as random variables in a Bayesian Network. In other words, in GOLOG, the emphasis is on high-level control, whereas in many probabilistic programming proposals, the emphasis is on inference. So, a direct comparison is difficult; whether these traditions can be combined is an open question.

4 Conclusions

This paper proposes an online account of belief-based programs that handles discrete and continuous probability distributions. It is intended as an alternative to GOLOG in stochastic dynamical domains where sensors and effectors are noisy.

Beginning with the expressive logical language of the situation calculus, we presented and implemented ALLEGRO using a rich BAT syntax. This fragment is interesting because it embodies the desirable features of logic-based action languages, such as non-trivial successor state and likelihood axioms. The latter property together with the rich query mechanism and program syntax in ALLEGRO suggests that it could be seen as a basis for relating high-level agent programming (Lakemeyer and Levesque 2007) and probabilistic robotics (Thrun, Burgard, and Fox 2005) in a general way.

There are many interesting avenues for future work, such as relating belief-based programs to decision-theoretic high-level programming (Boutilier et al. 2000), particle filters (Thrun, Burgard, and Fox 2005), and probabilistic programming (Goodman et al. 2008), among others. Integrating the ALLEGRO system with software frameworks such as ROS,[4] as shown in (Ferrein et al. 2015), is also planned for the future.

References

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

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

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking*. The MIT Press.

Baier, J. A.; Fritz, C.; and McIlraith, S. A. 2007. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proc. ICAPS*, 26–33.

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

Belle, V., and Levesque, H. J. 2014. PREGO: An Action Language for Belief-Based Cognitive Robotics in Continuous Domains. In *Proc. AAAI*.

Belle, V., and Levesque, H. J. 2015. Allegro: Belief-based programming in stochastic dynamical domains. In *IJCAI*.

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

Claßen, J., and Lakemeyer, G. 2006. Foundations for knowledge-based programs using ES. In *Proc. KR*, 318–328.

Claßen, J.; Liebenberg, M.; Lakemeyer, G.; and Zarrieß, B. 2014. Exploring the boundaries of decidable verification of non-terminating golog programs. In *AAAI*, 1012–1019.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. MIT Press.

[4] `ros.org`

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

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11):980–991.

Ferrein, A.; Maier, C.; Mühlbacher, C.; Niemueller, T.; Steinbauer, G.; and Vassos, S. 2015. Controlling logistics robots with the action-based language yagi. In *Proceedings of the 2015 IROS Workshop on Workshop on Task Planning for Intelligent Robots in Service and Manufacturing*.

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

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

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

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *I. J. Robotic Res.* 32(9-10):1194–1227.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99 – 134.

Lakemeyer, G., and Levesque, H. J. 2007. Cognitive robotics. In *Handbook of Knowledge Representation*. Elsevier. 869–886.

Lang, J., and Zanuttini, B. 2012. Knowledge-based programs as plans - the complexity of plan verification. In *Proc. ECAI*, 504–509.

Laverny, N., and Lang, J. 2005. From knowledge-based programs to graded belief-based programs, part i: On-line reasoning. *Synthese* 147(2):277–321.

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

Martin, Y., and Thielscher, M. 2009. Integrating reasoning about actions and Bayesian networks. In *International Conference on Agents and Artificial Intelligence (ICAART)*.

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

Nitti, D.; Laet, T. D.; and Raedt, L. D. 2013. A particle filter for hybrid relational domains. In *IROS*, 2764–2771.

Petrick, R., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. ICAPS*, 2–11.

Pfeffer, A. 2001. Ibal: A probabilistic rational programming language. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, 733–740. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

Raedt, L. D.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *Proc. IJCAI*, 2462–2467.

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

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

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

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

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

Sardina, S., and Lespérance, Y. 2010. Golog speaks the bdi language. In *Programming Multi-Agent Systems*, volume 5919 of *LNCS*. Springer Berlin Heidelberg. 82–99.

Sardina, S.; De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in indigolog—from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 41(2-4):259–299.

Shoham, Y. 1993. Agent-oriented programming. *Artificial intelligence* 60(1):51–92.

Spohn, W. 1988. Ordinal conditional functions: A dynamic theory of epistemic states. In *Causation in Decision, Belief Change, and Statistics*, volume 42 of *The University of Western Ontario Series in Philosophy of Science*. Springer Netherlands. 105–134.

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

Thielscher, M. 2004. Logic-based agents and the frame problem: A case for progression. In *First-Order Logic Revisited*, 323–336. Berlin, Germany: Logos.

Thielscher, M. 2005. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming* 5(4-5):533–565.

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

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

Van Ditmarsch, H.; Herzig, A.; and De Lima, T. 2007. Optimal regression for reasoning about knowledge and actions. In *Proc. AAAI*, 1070–1075.

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