Using Abstraction for Interpretable Robot Programs in Stochastic Domains

Till Hofmann[1] and Vaishak Belle[2]

[1] Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany
hofmann@kbsg.rwth-aachen.de
[2] University of Edinburgh, Edinburgh, UK
vbelle@ed.ac.uk

Abstract. A robot's actions are inherently stochastic, as its sensors are noisy and its actions do not always have the intended effects. For this reason, the agent language GOLOG has been extended to models with degrees of belief and stochastic actions. While this allows more precise robot models, the resulting programs are much harder to comprehend, because they need to deal with the noise, e.g., by looping until some desired state has been reached with certainty, and because the resulting action traces consist of a large number of actions cluttered with sensor noise. To alleviate these issues, we propose to use *abstraction*. We define a high-level and nonstochastic model of the robot and then map the high-level model into the lower-level stochastic model. The resulting programs are much easier to understand, often do not require belief operators or loops, and produce much shorter action traces.

1 Introduction

Classical approaches to model robot behavior such as GOLOG [7] assume complete knowledge of the word state as well as deterministic actions. However, both assumptions are often violated on real robots: the robot's sensor cannot completely capture the world, thus requiring some way to represent incomplete knowledge, and a robot's sensors and effectors are inherently noisy, necessitating a model of stochastic actions. Consider a simple robot equipped with a sonar sensor that is driving towards a wall (inspired from [1]). The sensor reading is imprecise and may produce incorrect sensor reading. Additionally, when doing a *move* action, the robot may get stuck with some probability. One approach to model such systems is an extension of the situation calculus [9] with stochastic actions and degrees of belief [1,3]. The core idea is to model the agent's belief with possible worlds, where each world has a certain weight, defining the probability that this world is the actual world. This allows to model degrees of belief, e.g., saying "the robot beliefs to be 2 m away with certainty 0.5". An action may then possibly have several outcomes, each specified with some likelihood.

While there has been progress on reasoning about belief-based programs [8] and programming languages such as ALLEGRO [4] allow to write belief-based programs, including stochastic actions in the model has the disadvantage that the resulting programs become significantly harder to understand. Dealing with noisy actions often requires many actions and loops to reach a desired state with certainty. Additionally, such stochastic programs often have many possible traces, making it more difficult to interpret one particular run of a program.

In this paper, we illustrate how *abstraction* can be used to deal with these issues. Extending on abstraction of basic action theories (BATs) in the classical situation calculus [2], we propose to use abstractions of stochastic domains, where the resulting abstract BAT does not contain any noisy sensors or effectors and is therefore nonstochastic. The resulting programs are easier to write and understand and the resulting traces are free of stochastic effects, thus making them much easier to comprehend.

The remainder of the paper is structured as follows [3]: In Section 2, we introduce belief-based programs based on the logic $\mathcal{DS}$ [3]. We present an example for the robot described above and we show how even simple programs induce traces that are non-trivial to understand. In Section 3, we define a more abstract and nonstochastic model that can be mapped to the lower-level model, resulting in more comprehensible programs. We conclude in Section 4.

2 Belief-Based Programs with Stochastic Actions

$\mathcal{DS}$ [3] is a modal variant of the situation calculus with degrees of belief and stochastic actions. Similar to $\mathcal{ES}$ [6], it is based on a possible worlds semantics, where worlds are part of the semantics and do not occur as terms in the language. In $\mathcal{DS}$, all worlds have a weight, which defines the probability of each world being the true world. The modal operator $\mathbf{B}(\alpha : r)$ expresses that α is believed with degree r, e.g., $\mathbf{B}(Loc(2) : 0.5)$ expresses that the agent believes "the distance to the wall is 2 m" with degree 0.5. We also write $\mathbf{Know}(\alpha)$ for $\mathbf{B}(\alpha : 1)$.

$\mathcal{DS}$ has two action modalities: $[a]$ and $\square$, where $[a]\alpha$ is to be read as "α holds after doing action a" and $\square\alpha$ is to be read as "α holds after any sequence of actions". As in the situation calculus, a BAT defines a domain by axiomatizing the initial situation, action preconditions, and effects. Additionally, noisy actions are modeled with the *action likelihood* l, where $l(a, u)$ expresses that the likelihood of action a is u. After doing an action, the agent may not always distinguish which instance of the action has actually been executed. This is expressed with *observational indistinguishability* (oi) axioms, where $oi(a, a')$ expresses that the agent cannot distinguish the actions a and a'.

We do not present a full account of $\mathcal{DS}$, but rather present an example. We model the robot's movement with the action $move(x, y)$, where x is the intended distance and y is the distance that the robot actually moved, and with a single fluent predicate $Loc(x)$ that describes the position of the robot. A BAT Σ_{move} defining the scenario from above may look as follows:

[3] In this paper, we focus on motivating the use of abstraction for interpretable programs. We refer to [5] for a full technical discussion.

- After doing action a, the robot is at position x if a is a *move* action that moves the robot to location x, if a is a *sonar* action that measures distance x, or if a is neither of the two actions and the robot was at location x before[4]:

$$\begin{aligned}\square[a]Loc(x) \equiv\ & \exists y, z, (a = move(y, z) \land Loc(l) \land x = l + z) \lor a = sonar(x) \\ & \lor \neg\exists y, z\, (a = move(y, z) \lor a = sonar(y)) \land Loc(x)\end{aligned}$$

- A *move* action is possible if the robot moves either one step to the back or to the front. A *sonar* action is always possible:

$$\square Poss(a) \equiv \exists x, y\, (a = move(x, y) \land (x = 1 \lor x = -1)) \lor \exists z\, (a = sonar(z))$$

- Action likelihood axioms: For the *sonar* action, the likelihood that the robot measures the correct distance is 0.8, the likelihood that it measures a distance with an error of ± 1 is 0.1. Furthermore, for the *move* action, the likelihood that the robot moves the intended distance x is 0.6, the likelihood that the actual movement y is off by ± 1 is 0.2:

$$\begin{aligned}\square l(a, u) \equiv\ & \exists z\, (a = sonar(z) \land Loc(x) \land u = \Theta(x, z, .8, .1)) \\ & \lor \exists x, y\, (a = move(x, y) \land u = \Theta(x, y, .6, .2)) \\ & \lor \neg\exists x, y, z\, (a = move(x, y) \lor a = sonar(z)) \land u = .0\end{aligned}$$

where $\Theta(u, v, c, d) = \begin{cases} c & \text{if } u = v \\ d & \text{if } |u - v| = 1\,. \\ 0 & \text{otherwise} \end{cases}$

- The robot cannot detect the distance that it has actually moved, i.e., any two actions $move(x, y)$ and $move(x, z)$ are o.i.:

$$\square oi(a, a') \equiv \exists x, y, z\, (a = move(x, y) \land a' = move(x, z)) \lor a = a'$$

- Initially, the robot is 3 m away from the wall: $\forall x (Loc(x) \equiv x = 3)$

Based on this BAT, we define a program that first moves the robot close to the wall and then back[5]:

$$\begin{aligned}& sonar(); \textbf{while } \neg\textbf{Know}(\exists x\, (Loc(x) \land x \leq 2)) \textbf{ do } move(-1); sonar() \textbf{ done }; \\ & \quad \textbf{while } \neg\textbf{Know}(\exists x\, (Loc(x) \land x > 5)) \textbf{ do } move(1); sonar() \textbf{ done}\end{aligned}$$

The robot first measures its distance to the wall and then moves closer until it knows that its distance to the wall is less than 2 m. Afterwards, it moves away

[4] We assume that free variables are universally quantified from the outside and that $\square$ has lower syntactic precedence than the logical connectives, so that $\square[a]Loc(x) \equiv \gamma$ stands for $\forall a. \square\, ([a]Loc(x) \equiv \gamma)$.

[5] The unary $move(x)$ can be understood as abbreviation $move(x) := \pi y\, move(x, y)$, where nature nondeterministically picks the distance y that the robot really moved (similarly for $sonar()$).

until it knows that is more than 5 m away from the wall. As the robot's *move* action is noisy, each *move* is followed by *sonar* to measure how far it is away from the wall. One possible execution trace of this program may look as follows:

$$\begin{aligned} z_l = \langle & \mathit{sonar}(3), \mathit{move}(-1,0), \mathit{sonar}(3), \mathit{move}(-1,-1), \mathit{sonar}(2), \mathit{move}(-1,-1), \\ & \mathit{sonar}(1), \mathit{move}(1,1), \mathit{sonar}(3), \mathit{move}(1,1), \mathit{sonar}(2), \mathit{move}(1,1), \\ & \mathit{sonar}(4), \mathit{move}(1,0), \mathit{sonar}(4), \mathit{move}(1,1), \mathit{sonar}(6)\rangle \end{aligned} \tag{1}$$

First, the robot (correctly) senses that it is 3 m away from the wall and starts moving. However, the first *move* does not have the desired effect: the robot intended to move by 1 m but actually did not move (indicated by the second argument being 0). After the second *move*, the robot is at $\mathit{Loc}(2)$, as it started at $\mathit{Loc}(3)$ and moved successfully once. However, as its sensor is noisy and it measured $\mathit{sonar}(2)$, it believes that it could also be at $\mathit{Loc}(3)$. For safe measure, it executes another *move* and then senses $\mathit{sonar}(1)$, after which it knows for sure that it is at a distance ≤ 2 m. In the second part, the robot moves back until it knows that it has reached a distance > 5 m. As this simple example shows, the trace z_l is already quite hard to understand. While it is clear from the BAT what each action does, the robot's intent is not immediately obvious and the trace is cluttered with noise and low-level details.

3 Using Abstraction to Obtain Interpretable Programs

To hide away the low-level details such as noisy *move* actions, we propose to use *abstraction*. Similar to [2], abstraction in $\mathcal{DS}$ is a mapping of a high-level BAT to a low-level BAT. We first define the high-level BAT and then map each fluent and action to the lower level. Continuing our example, we can define a second, high-level BAT that consists of the locations *near* and *far*, the fluent *At* that specifies the current location of the robot, and the single action *goto*, which is an idealized *move* action without noise. The high-level BAT Σ_{goto} looks as follows:

- After doing action a, the robot is at location l if a is the action $\mathit{goto}(l)$ or if a is no *goto* action and the robot has been at l before:

$$\Box[a]At(l) \equiv a = \mathit{goto}(l) \lor \neg\exists x\, (a = \mathit{goto}(x)) \land At(l)$$

- The robot may do action a if a is a *goto* action to a valid location:

$$\Box \mathit{Poss}(a) \equiv a = \mathit{goto}(\mathit{near}) \lor a = \mathit{goto}(\mathit{far})$$

- The *goto* action is not noisy:

$$\Box l(a,u) \equiv \exists x\, (a = \mathit{goto}(x)) \land u = 1.0 \lor \neg\exists x\, (a = \mathit{goto}(x)) \land u = 0.0$$

- The agent can distinguish all actions: $\Box oi(a,a') \equiv a = a'$
- Initially, the robot is at the location *near*: $\forall l (At(l) \equiv l = \mathit{near})$

Next, we define a *refinement mapping* that maps each high-level fluent to a low-level formula and each high-level action to a low-level program:

– The high-level fluent $At(l)$ is mapped to a low-level formula by translating the distance to the two locations *near* and *far*:

$$At(l) \mapsto \big(l = near \land \exists x\, (Loc(x) \land x \leq 2) \lor l = far \land \exists x\, (Loc(x) \land x > 5)\big)$$

– The action *goto* is mapped to a program that guarantees that the robot reaches the right position:

$$\begin{aligned}
&goto(x) \mapsto sonar();\\
&\qquad \textbf{if } x = near \textbf{ then}\\
&\qquad\quad \textbf{while } \neg\textbf{Know}(\exists x\, (Loc(x) \land x \leq 2)) \textbf{ do } move(-1); sonar() \textbf{ done}\\
&\qquad \textbf{elif } x = far \textbf{ then}\\
&\qquad\quad \textbf{while } \neg\textbf{Know}(\exists x\, (Loc(x) \land x > 5)) \textbf{ do } move(1); sonar() \textbf{ done} \textbf{ ; fi}
\end{aligned}$$

Using the refinement mapping, we can translate a high-level program based on the BAT Σ_{goto} to a low-level program based on the BAT Σ_{move}. As Σ_{goto} does not contain any sensing or noisy actions, we obtain a nonstochastic program that is much simpler to understand than the low-level program shown above. The program that first moves close to the wall and then moves back only needs two actions and does not require any belief operators:

$$goto(near); goto(far)$$

Note that when the program is executed, each *goto* action is translated into a corresponding low-level program, as defined by the mapping above. However, from the high-level perspective, the program only allows a single trace $z_h = \langle goto(near), goto(far)\rangle$. Compared to the low-level trace z_l from Equation 1, z_h is much easier to understand. It only consists of two actions, which are exactly the two actions from the program. It does not contain any noise, which is the reason why the trace is also unique. Furthermore, we were able to abstract away all sensing actions, further simplifying the resulting traces.

4 Conclusion

In this paper, we demonstrated how *abstraction* can be used to map a low-level Golog program with stochastic actions to a high-level program that is nonstochastic, does not require any sensing or belief operators, and thus is much easier to understand. While this requires some additional work to define the mapping between the two models, the mapping is not specific for a given program and thus can be re-used for other programs in the same domain.

Acknowledgements T. Hofmann was partly supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 2236/1 and the EU ICT-48 2020 project TAILOR (No. 952215). Part of this work was created during a research visit of T. Hofmann at the University of Edinburgh, which was funded by the German Academic Exchange Service (DAAD). V. Belle was partly supported by a grant from the UKRI Strategic Priorities Fund to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020-2024). He was also supported by a Royal Society University Research Fellowship.

References

1. Bacchus, F., Halpern, J.Y., Levesque, H.J.: Reasoning about noisy sensors and effectors in the situation calculus. Artificial Intelligence **111**(1), 171–208 (Jul 1999)
2. Banihashemi, B., De Giacomo, G., Lespérance, Y.: Abstraction in situation calculus action theories. In: Proceedings of the 31st Conference on Artificial Intelligence (AAAI) (2017)
3. Belle, V., Lakemeyer, G.: Reasoning about probabilities in unbounded first-order dynamical domains. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI). pp. 828–836 (2017)
4. Belle, V., Levesque, H.J.: ALLEGRO: Belief-based programming in stochastic dynamical domains. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI). pp. 2762–2769 (2015)
5. Hofmann, T., Belle, V.: Abstracting noisy robot programs. arXiv:2204.03536 [cs] (2022)
6. Lakemeyer, G., Levesque, H.J.: A semantic characterization of a useful fragment of the situation calculus with knowledge. Artificial Intelligence **175**(1), 142–164 (Jan 2011)
7. Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31**(1-3) (1997)
8. Liu, D., Lakemeyer, G.: Reasoning about beliefs and meta-beliefs by regression in an expressive probabilistic action logic. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI). vol. 2, pp. 1951–1958 (2021)
9. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)