Using Abstraction for Interpretable Robot Programs in Stochastic Domains

KR'22 XLoKR

Till Hofmann, Vaishak Belle

THE UNIVERSITY *of* EDINBURGH

July 30, 2022

Golog Programs

Golog:

- Agent language based on the Situation Calculus
- A basic action theory Σ specifies
  - the initial situation
  - action preconditions
  - action effects (successor state axioms)
- Allows nondeterministic operators:
  - $\delta_1|\delta_2$ nondeterministically executes one of the two branches δ_1 and δ_2
  - $\pi x.\delta$ picks some value for x and substitutes x by the value in the program δ
- Also supports interleaved concurrency: $\delta_1 \| \delta_2$
- High-level specification of a robot's behavior

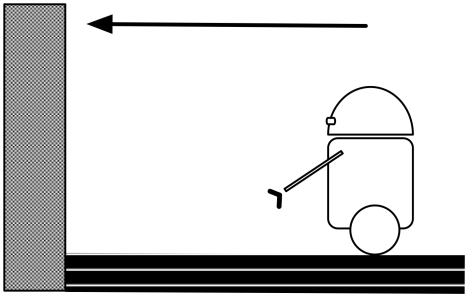
Golog Programs

- The Basic Action Theory:

$$\begin{aligned} \Sigma_0 &= At(near) \vee At(far) \\ \Box Poss(a) &\equiv \exists l.a = goto(l) \wedge \neg At(l) \\ \Box[a]At(l) &\equiv a = goto(l) \vee \\ &\quad At(l) \wedge \neg \exists l' a = goto(l') \end{aligned}$$

- The program:

if $\neg At(near)$ **then**
  $goto(near)$
end if
$goto(far)$



(Belle and Lakemeyer 2017)

Knowledge-Based Programs

- Classical Golog assumes complete knowledge
- If some fact is true in the real world, the robot "knows" about it
- In practice: knowledge is incomplete
- Robot needs to use *sensors*

→ Epistemic Situation Calculus

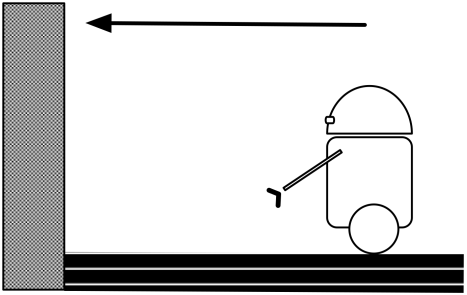
- Possible-worlds semantics: something is *known* if it is true in every possible world

Knowledge-Based Programs

- Classical Golog assumes complete knowledge
- If some fact is true in the real world, the robot "knows" about it
- In practice: knowledge is incomplete
- Robot needs to use *sensors*

→ Epistemic Situation Calculus

- Possible-worlds semantics: something is *known* if it is true in every possible world



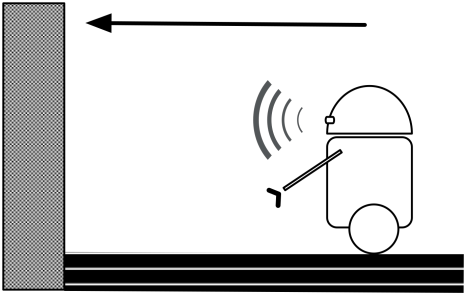
(Belle and Lakemeyer 2017)

Knowledge-Based Programs

- Classical Golog assumes complete knowledge
- If some fact is true in the real world, the robot "knows" about it
- In practice: knowledge is incomplete
- Robot needs to use *sensors*

→ Epistemic Situation Calculus

- Possible-worlds semantics: something is *known* if it is true in every possible world



(Belle and Lakemeyer 2017)

Knowledge-Based Programs

- Classical Golog assumes complete knowledge
- If some fact is true in the real world, the robot "knows" about it
- In practice: knowledge is incomplete
- Robot needs to use *sensors*

→ Epistemic Situation Calculus

- Possible-worlds semantics: something is *known* if it is true in every possible world

sonar()
if ¬Know(*At*(*near*)) **then**
    goto(*near*)
end if
goto(*far*)

Degrees of Belief (Bacchus et al. 1999; Belle and Lakemeyer 2017)

- Knowledge-based programs can deal with incomplete knowledge
- However, we still assume:
  - noiseless sensors without measurement error
  - perfect actions that always have the desired effect

Degrees of Belief (Bacchus et al. 1999; Belle and Lakemeyer 2017)

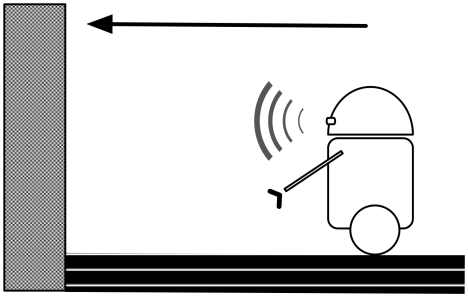
- Knowledge-based programs can deal with incomplete knowledge
- However, we still assume:
  - noiseless sensors without measurement error
  - perfect actions that always have the desired effect
- In practice, both assumptions are idealistic:
  - The *sonar* sensor may measure with an error, e.g., ± 1
  - The robot may get stuck with some probability when trying to move
  - The robot may not be able to observe those errors

Degrees of Belief (Bacchus et al. 1999; Belle and Lakemeyer 2017)

- Knowledge-based programs can deal with incomplete knowledge
- However, we still assume:
  - noiseless sensors without measurement error
  - perfect actions that always have the desired effect
- In practice, both assumptions are idealistic:
  - The *sonar* sensor may measure with an error, e.g., ± 1
  - The robot may get stuck with some probability when trying to move
  - The robot may not be able to observe those errors
- Logic $\mathcal{DS}$ (Belle and Lakemeyer 2017) allows to model such robots:
  - Each world is assigned a weight
  - $\mathrm{B}(\alpha : r)$: α is believed with degree r
  - e.g., $\mathrm{B}(At(near) : 0.9)$: $At(near)$ is true with probability 0.9

A $\mathcal{DS}$ Basic Action Theory

- $Loc(x)$ is true if the distance to the wall is x
- One action $move(x, y)$, where
  - x is the distance the robot intends to move
  - y is the distance that the robot actually moves
  - We will write $move(x)$ for $\pi y.move(x, y)$, where nature nondeterministically chooses y
- One noisy sensor $sonar(x)$ that measures the distance to the wall
  As above, $sonar()$ is short for $\pi x.sonar(x)$ (where nature chooses x)



(Belle and Lakemeyer 2017)

A $\mathcal{DS}$ Program

sonar()
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \leq 3 : 1)$ **do**
    move(−1)
    sonar()
end while
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \geq 5 : 1)$ **do**
    move(1)
    sonar()
end while

A $\mathcal{DS}$ Program

sonar()
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \leq 3 : 1)$ **do**
    move(-1)
    sonar()
end while
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \geq 5 : 1)$ **do**
    move(1)
    sonar()
end while

$sonar(4), move(-1, 0), sonar(3), move(-1, -1),$
$sonar(4), move(-1, -1), sonar(2),$

A $\mathcal{DS}$ Program

sonar()
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \leq 3:1)$ **do**
    move(-1)
    sonar()
end while
while $\neg \mathrm{B}(\exists l.\, Loc(l) \wedge l \geq 5:1)$ **do**
    move(1)
    sonar()
end while

$sonar(4), move(-1,0), sonar(3), move(-1,-1),$
$sonar(4), move(-1,-1), sonar(2),$
$move(1,1), sonar(3), move(1,1),$
$sonar(3), move(1,1), sonar(4), move(1,0),$
$sonar(4), move(1,1), sonar(6)$

Challenges

Probabilistic belief programs have challenges:

1. Correctly designing programs is difficult
2. Reasoning about probabilities is hard
3. **Understanding how such a system operates is not trivial**

Challenges

Probabilistic belief programs have challenges:

1. Correctly designing programs is difficult
2. Reasoning about probabilities is hard
3. **Understanding how such a system operates is not trivial**

We would like to

- model stochastic actions and noisy sensors
- write high-level programs without dealing with probabilities
- obtain execution traces that are easy to understand

Abstraction

$\Rightarrow$ We can use abstraction!

- Idea:
  - define a low-level $\mathcal{DS}$ BAT that includes stochastic actions
  - define a second, high level BAT that abstracts away stochasticity

Abstraction

$\Rightarrow$ We can use abstraction!

- Idea:
  - define a low-level $\mathcal{DS}$ BAT that includes stochastic actions
  - define a second, high level BAT that abstracts away stochasticity
- Map the high-level program to the low-level program:
  1. Map each high-level fluent to a low-level formula, e.g.,:

$$At(far) \mapsto \exists x.\, Loc(x) \wedge x \geq 5$$

  2. Map each high-level action to a low-level procedure, e.g.,

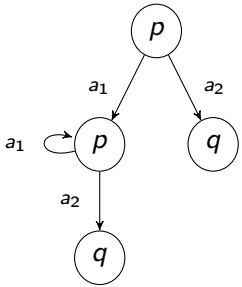
$$goto(far) \mapsto \text{while } \neg\text{Know}(\exists x\,(Loc(x) \wedge x \geq 5)) \text{ do}$$
$$move(1); sonar()$$
$$\text{done}$$

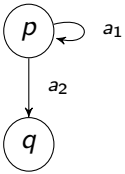
  3. Define a *bisimulation* between high-level and low-level program

Bisimulation

Intuitively, two states s_1, s_1' of two transition systems T, T' are bisimilar if

1. they have the same local properties (e.g., labels) $\rightarrow$ *Isomorphism*
2. if $s_1 \xrightarrow{a} s_2$, then there is s_2' such that $s_1' \xrightarrow{a} s_2'$ and (s_2, s_2') are bisimilar
3. if $s_1' \xrightarrow{a} s_2'$, then there is s_2 such that $s_1 \xrightarrow{a} s_2$ and (s_2, s_2') are bisimilar

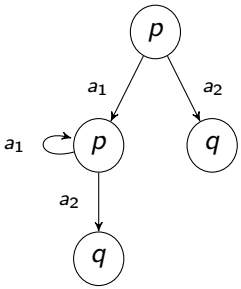


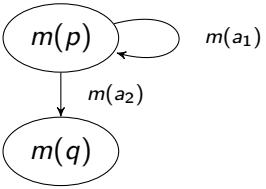


Bisimulation for Belief-Based Programs

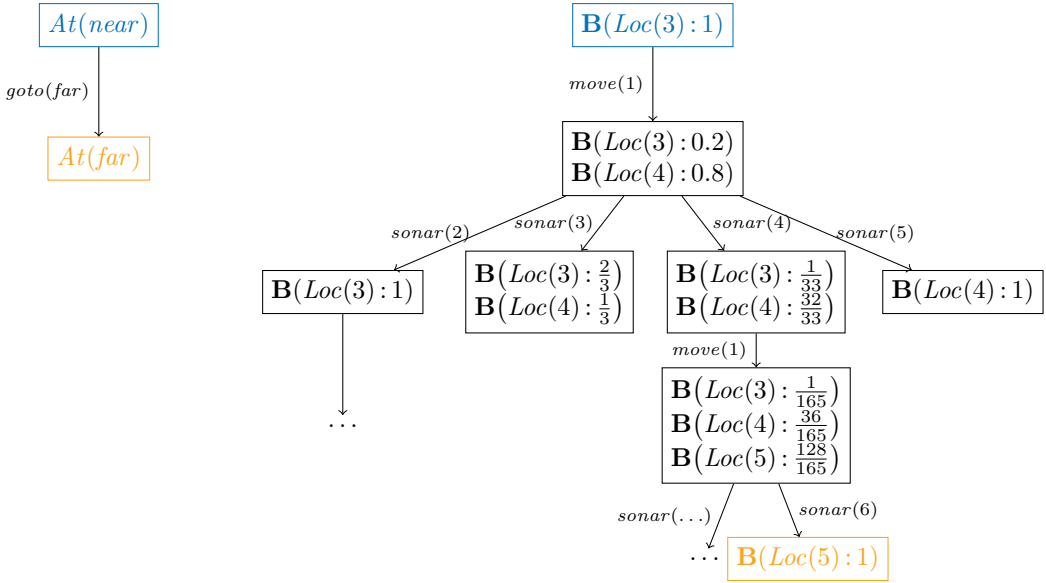
Intuitively, two states s_1, s_1' of two transition systems T, T' are bisimilar if

1. they have the same mapped local properties (atomic propositions) $\rightarrow$ *Isomorphism*
2. if $s_1 \xrightarrow{a} s_2$, then there is s_2' such that $s_1' \xrightarrow{m(a)} s_2'$ and (s_2, s_2') are bisimilar
3. if $s_1' \xrightarrow{m(a)} s_2'$, then there is s_2 such that $s_1 \xrightarrow{a} s_2$ and (s_2, s_2') are bisimilar





Bisimulation for Belief-Based Programs



Abstraction: Program Equivalence

if $\neg At(near)$ **then**
    $goto(near)$
end if
$goto(far)$

$\Leftrightarrow$

$sonar()$
while $\neg$Know$(\exists l.\, Loc(l) \wedge l \leq 3)$ **do**
    $move(-1)$
    $sonar()$
end while
while $\neg$Know$(\exists l.\, Loc(l) \wedge l \geq 5)$ **do**
    $move(1)$
    $sonar()$
end while

Abstraction: Trace Equivalence

$goto(near),$
$goto(far)$

$\Leftrightarrow$

$sonar(4), move(-1, 0), sonar(3), move(-1, -1),$
$sonar(4), move(-1, -1), sonar(2),$
$move(1, 1), sonar(3), move(1, 1),$
$sonar(3), move(1, 1), sonar(4), move(1, 0),$
$sonar(4), move(1, 1), sonar(6)$

Conclusion

- Robot domains are often stochastic because of noisy sensors and actuators

→ Need to model stochastic domains

- However, stochastic Golog programs are hard to interpret
- Program traces are lengthy and contain noise, even in simple domains

Conclusion

- Robot domains are often stochastic because of noisy sensors and actuators

→ Need to model stochastic domains

- However, stochastic Golog programs are hard to interpret
- Program traces are lengthy and contain noise, even in simple domains

Idea: use abstraction to make stochastic programs easier to interpret

- Define a second, nonstochastic BAT
- Map high-level BAT to low-level stochastic BAT
- Write programs in high-level BAT
- Bisimulation establishes equivalence

→ Easier to write programs

→ Easier to understand programs and interpret program traces

A $\mathcal{DS}$ Basic Action Theory

- Initially, the robot is 4 m away from the wall:

$$\forall x(Loc(x) \equiv x = 4)$$

- The robot can move back and forth by 1 m and also always use its sonar:

$$\begin{aligned}\Box Poss(a) \equiv &\exists x, y\,(a = move(x, y) \wedge (x = 1 \vee x = -1)) \\ &\vee \exists z\,(a = sonar(z))\end{aligned}$$

- The location of the robot changes with the second argument of $move(x, y)$:

$$\begin{aligned}\Box[a]Loc(l) \equiv &\exists x, y\,\big(a = move(x, y) \wedge Loc(l') \wedge l = l' + y\big) \\ &\vee \neg\exists x, y\,(a = move(x, y)) \wedge Loc(l)\end{aligned}$$

A $\mathcal{DS}$ Basic Action Theory

- Action likelihood axioms:

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

where

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

- Observational indistinguishability:

$$\begin{aligned}\Box oi(a,a') \equiv\ & \exists x,y,z\,\big(a = move(x,y) \wedge a' = move(x,z)\big) \\ & \vee \exists z\,\big(a = sonar(z) \wedge a = a'\big)\end{aligned}$$

Bisimulation of probabilistic belief programs

In our context, when are two states isomorphic?

Definition (Objective Isomorphism)

We say that (w_h, z_h) is objectively m-isomorphic to (w_l, z_l), written $(w_h, z_h) \sim_m (w_l, z_l)$ iff for every atomic formula α mentioned in Σ_h:

$$w_h, z_h \models \alpha \text{ iff } w_l, z_l \models m(\alpha)$$

Epistemic isomorphism: Intuitively, two states must have the same weight; however, there are some complications ...

Epistemic Isomorphism

- Problem: High-level state is supposed to be more abstract, i.e., we cannot just directly compare the weights of two states

→ Map a state (w_h, z_h) to a set of states $\left\{(w_l^{(i)}, z_l^{(i)})\right\}_i$

- Problem: Norm is defined wrt *oi*, but not all the low-level states need to be *oi*

→ Partition $\left\{(w_l^{(i)}, z_l^{(i)})\right\}_i$ wrt *oi*

Definition (Epistemic Isomorphism)

For $(w_h, z_h) \in \mathcal{S}$ and $S_l \subseteq \mathcal{S}$, we say that (d_h, w_h, z_h) is epistemically m-isomorphic to (d_l, S_l), written $(d_l, w_h, z_h) \sim_e (d_l, S_l)$ iff for the partition $P = S_l / \approx_{\text{oi}}$, for each $S_l^i \in P$ and $(w_l^i, z_l^i) \in S_l^i$:

$$\text{Norm}(d_h, \{(w_h, z_h)\}, S_{\text{True}}^{e_h, w_h, z_h}) = \text{Norm}(d_l, S_l^i, S_{\text{True}}^{e_l, w_l^i, z_l^i})$$

Bisimulation

- Bisimulation is a relation B between high-level states (w_h, z_h) and low-level states (w_l, z_l)
- We have defined *isomorphism* $\rightarrow$ local property
- What about transitions?
- We have two types of transitions:
  1. *Action step:* If we can take an action a in $(w_h, z_h) \rightarrow (w_h, z_h \cdot a)$, then we should be able to execute the mapped program $m(a)$ in (w_l, z_l) so we again get into two bisimilar states
  2. *Epistemic step:* If there is an *oi* state $(w'_h, z'_h) \approx_{\text{oi}} (w_h, z_h)$ with non-zero weight, then there should be a corresponding low-level state $(w'_l, z'_l) \approx_{\text{oi}} (w_l, z_l)$ that is bimisilar to (w'_h, z'_h)

Bisimulation

Theorem

Let $(e_h, w_h) \sim_m (e_l, w_l)$ *with m-bisimulation* B*. Then for every bounded formula* α *and traces* z_h, z_l *with* $(z_h, z_l) \in B$*:*

$$e_h, w_h, z_h \models \alpha \text{ iff } e_l, w_l, z_l \models m(\alpha)$$

Sound and Complete Abstractions

- So far: we have defined bisimulations wrt to specific (e_h, w_h) and (e_l, w_l)
- We would like to go from particular models to basic action theories

→ *Sound abstraction:* For each low-level model e_l, w_l of Σ_l, there exists a high-level model e_h, w_h of Σ_h

→ *Complete abstraction:* For each high-level model e_h, w_h of Σ_h, there exists a low-level model e_l, w_l of Σ_l

Theorem

Let Σ_h be a sound and complete abstraction of Σ_l relative to refinement mapping m. Then, for every bounded formula α,

$$\mathrm{Know}(\Sigma_h) \wedge \Sigma_h \models \alpha \textit{ iff } \mathrm{Know}(\Sigma_l) \wedge \Sigma_l \models m(\alpha)$$

References I

- Bacchus, Fahiem, Joseph Y. Halpern, and Hector J. Levesque (July 1999). "Reasoning about Noisy Sensors and Effectors in the Situation Calculus". In: *Artificial Intelligence* 111.1, pp. 171–208. issn: 0004-3702.
- Belle, Vaishak and Gerhard Lakemeyer (2017). "Reasoning about Probabilities in Unbounded First-Order Dynamical Domains". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 828–836.