Where am I? Let me ask to the real world!

Luciano Serafini   Paolo Traverso

Fondazione Bruno Kessler

Where am I? Let me ask to the real world!

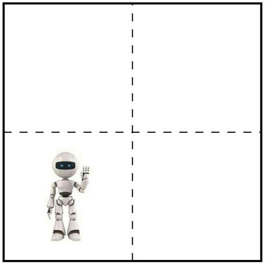
Learning abstract planning domains and mappings to real world perceptions

Luciano Serafini Paolo Traverso

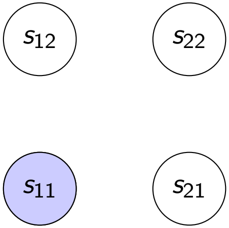
Fondazione Bruno Kessler

A simple example

Real world

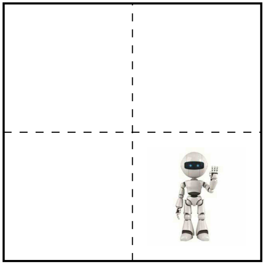


Abstract model



A simple example

Real world

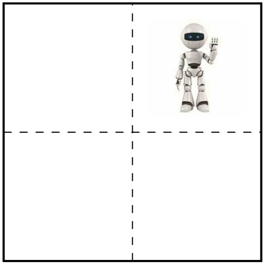


Abstract model

$$\begin{tikzcd}
s_{12} & s_{22} \\
s_{11} \arrow[r, bend left, "e"] & s_{21}
\end{tikzcd}$$

A simple example

Real world

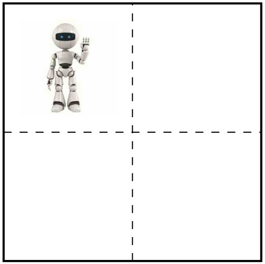


Abstract model

$$\begin{tikzcd}
s_{12} & s_{22} \\
s_{11} \arrow[r, bend left, "e"] & s_{21} \arrow[u, bend left, "n"]
\end{tikzcd}$$

A simple example

Real world

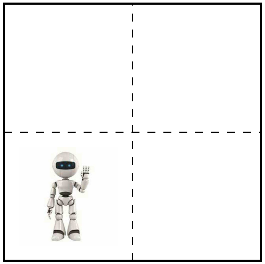


Abstract model

$$\begin{tikzcd}
s_{12} & s_{22} \arrow[l, bend left, "w"] \\
s_{11} \arrow[r, bend left, "e"] & s_{21} \arrow[u, bend left, "n"]
\end{tikzcd}$$

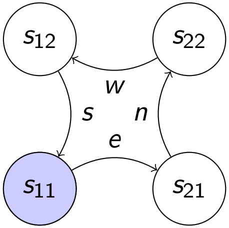
A simple example

Real world



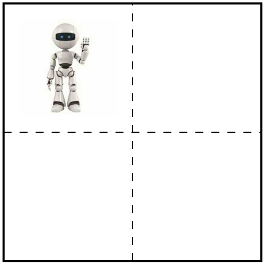
Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \arrow[l, bend left, "w"'] \\
s_{11} \arrow[r, bend left, "e"'] & s_{21} \arrow[u, bend left, "n"]
\end{tikzcd}$$



A simple example

Real world

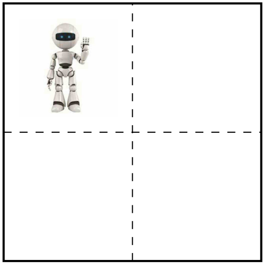


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \arrow[l, bend right, "w"'] \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[u, bend left, "n"]
\end{tikzcd}$$

A simple example

Real world

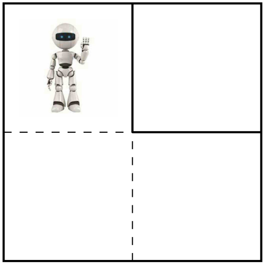


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[r, bend left, "e"] \arrow[d, bend left, "s"] & s_{22} \arrow[l, bend left, "w"] \arrow[d, bend left, "s"] \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"] \arrow[u, bend left, "n"]
\end{tikzcd}$$

A simple example

Real world

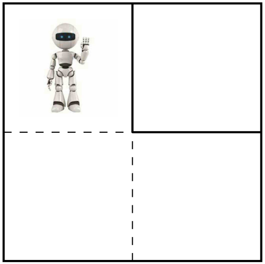


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[r, bend left, "e"] \arrow[d, bend left, "s"] & s_{22} \arrow[l, bend left, "w"] \arrow[d, bend left, "s"] \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"] \arrow[u, bend left, "n"]
\end{tikzcd}$$

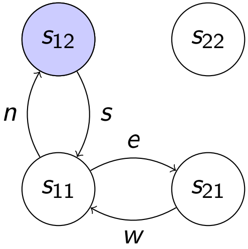
A simple example

Real world



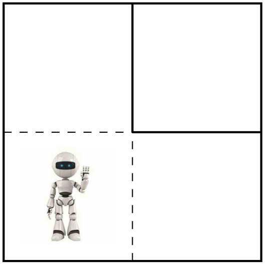
Abstract model

$$\begin{tikzcd}
s_{12} & s_{22} \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"]
\end{tikzcd}$$



A simple example

Real world

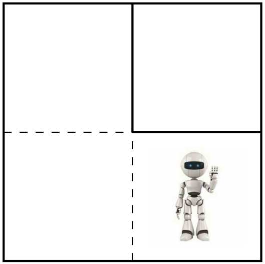


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"]
\end{tikzcd}$$

A simple example

Real world

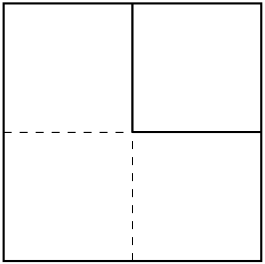


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"]
\end{tikzcd}$$

A simple example

Real world

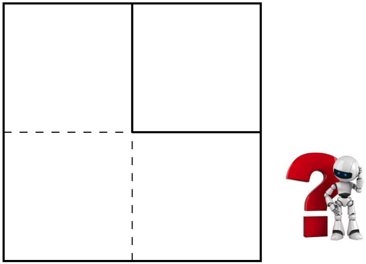


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"]
\end{tikzcd}$$

A simple example

Real world

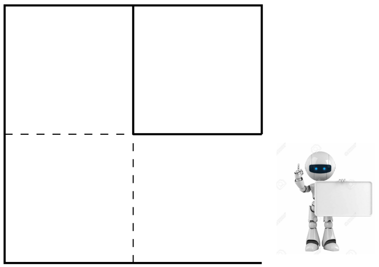


Abstract model

$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"]
\end{tikzcd}$$

A simple example

Real world

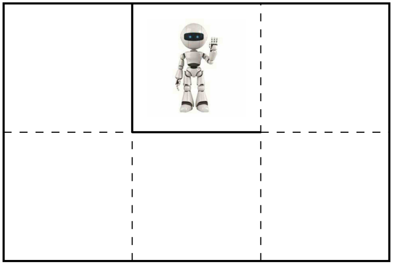


Abstract model

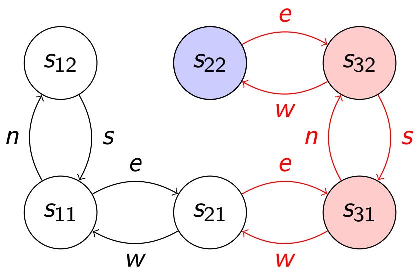
$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} & \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"] \arrow[r, bend left, "e", color=red] & s_{31}
\end{tikzcd}$$

A simple example

Real world



Abstract model



$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \arrow[r, bend left, "e"] & s_{32} \arrow[l, bend left, "w"] \arrow[d, bend left, "s"] \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"] \arrow[r, bend left, "e"] & s_{31} \arrow[l, bend left, "w"] \arrow[u, bend left, "n"]
\end{tikzcd}$$

The Challenge

Abstract world and real world

While an agent can conveniently plan at the abstract level, it perceives the world and acts in it through sensors and actuators that work with data in a continuous world, typically represented with variables on real numbers.

The Challenge

Abstract world and real world

While an agent can conveniently plan at the abstract level, it perceives the world and acts in it through sensors and actuators that work with data in a continuous world, typically represented with variables on real numbers.

Unexpected observations

There may be situations in which the agent perceives data which are not compatible with any of the states of its abstract model.

Objective

A formal framework in which

(i) the agent can learn dynamically new states of the planning domain;

(ii) the mapping between abstract states and the perception from the real world, represented by continuous variables, is part of the planning domain;

(iii) such mapping is learned and updated along the life of the agent.

The Idea

A formal framework in which

- We model agent's perception of the real world by a *perception function* that returns the likelihood of observing some continuous data being in a state of the domain.

The Idea

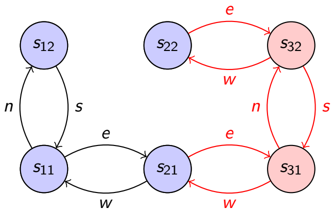
A formal framework in which

- We model agent's perception of the real world by a *perception function* that returns the likelihood of observing some continuous data being in a state of the domain.

- We define an algorithm that interleaves planning, acting, and learning. It is able to discover that the abstract model is not coherent with the real world.

Planning Domains

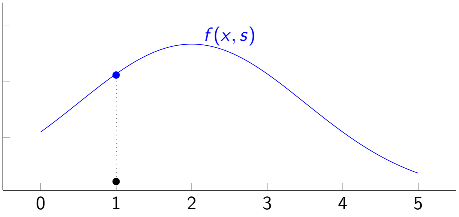
- Deterministic Planning Domain: $\mathcal{D} = \langle S, A, \gamma \rangle$
- State transition function: $\gamma : S \times A \rightarrow S$
- Planning Problem: $\mathcal{P} = \langle \mathcal{D}, s_0, S_g \rangle$
- A Plan π is a policy: partial function from $\pi : S \rightharpoonup A$.



Planning Domains with Perception Functions

- Deterministic Planning Domain: $\mathcal{D} = \langle S, A, \gamma \rangle$
- State transition function: $\gamma : S \times A \to S$
- Planning Problem: $\mathcal{P} = \langle \mathcal{D}, s_0, S_g \rangle$
- A Plan π is a policy: partial function from $\pi : S \rightharpoonup A$.
- Perception Function: $f : \mathbb{R}^n \times S \to R^+$, where $f(\boldsymbol{x}, s) = p(\boldsymbol{x}|s) = \frac{p(\boldsymbol{x},s)}{p(s)}$, with $p(\boldsymbol{x}, s)$ a joint PDF on $\mathbb{R}^n \times S$
  $f(\boldsymbol{x}, s)$ is the likelihood of observing $\boldsymbol{x}$ being in a state s.

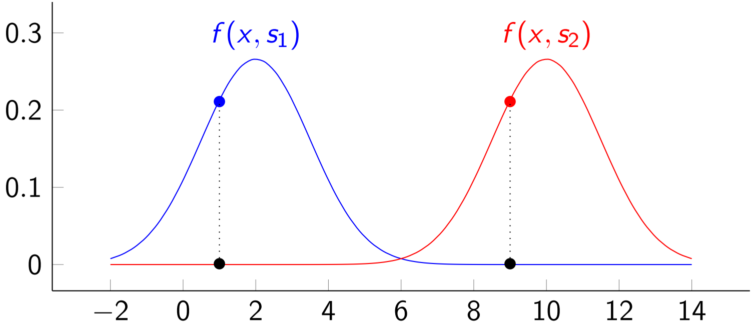
$$\begin{tikzcd}
s_{12} \arrow[d, bend left, "s"] & s_{22} \arrow[r, bend left, "e"] & s_{32} \arrow[l, bend left, "w"] \arrow[d, bend left, "s"] \\
s_{11} \arrow[u, bend left, "n"] \arrow[r, bend left, "e"] & s_{21} \arrow[l, bend left, "w"] \arrow[r, bend left, "e"] & s_{31} \arrow[l, bend left, "w"] \arrow[u, bend left, "n"]
\end{tikzcd}$$



Simple univariate perception function

Example

- $S = \{s_1, s_2\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$



The Planning, Acting, and Learning (PAL) Algorithm

Loop until goal reached:

1. generate a plan (if no plan exists, explore the domain)
2. execute the first action in the plan and observe $\boldsymbol{x}$
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state
4. update the transition function (γ)
5. update the perception function (f)

The Planning, Acting, and Learning (PAL) Algorithm

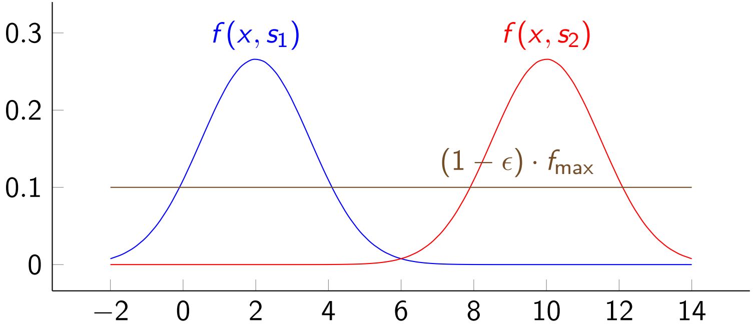
Loop until goal reached:

1. generate a plan (if no plan exists, explore the domain)
2. execute the first action in the plan and observe $\boldsymbol{x}$
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state
4. update the transition function (γ)
5. update the perception function (f)

Simple univariate perception function

Example

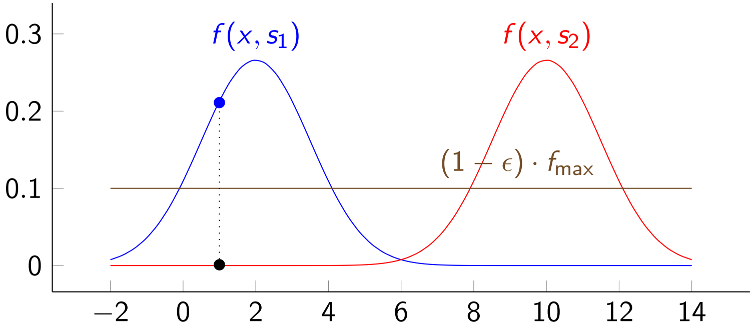
- $S = \{s_1, s_2\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$



Simple univariate perception function

Example

- $S = \{s_1, s_2\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$

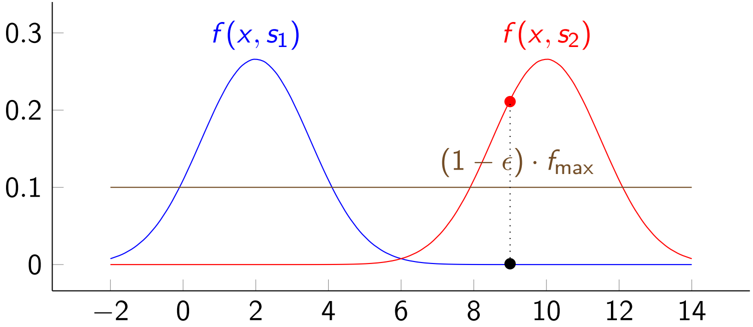


$s_1 = \operatorname{argmax}_{s \in \{s_1, s_2\}} f(1.0, s)$ and $f(1.0, s_1) \geq (1-\epsilon) \cdot f_{\max}$

Simple univariate perception function

Example

- $S = \{s_1, s_2\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$

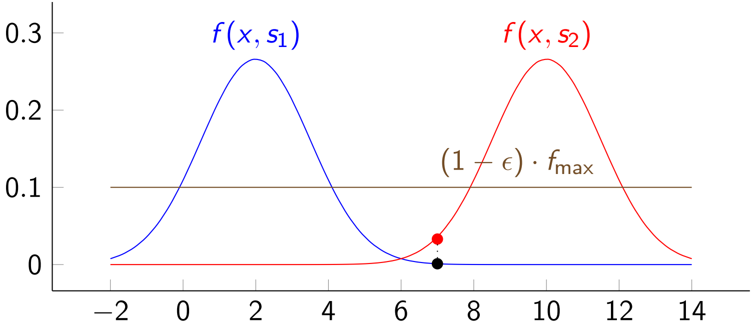


$s_2 = \mathrm{argmax}_{s\in\{s_1,s_2\}}\, f(9.0, s)$ and $f(9.0, s_2) \geq (1-\epsilon)\cdot f_{\max}$

Simple univariate perception function

Example

- $S = \{s_1, s_2\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$

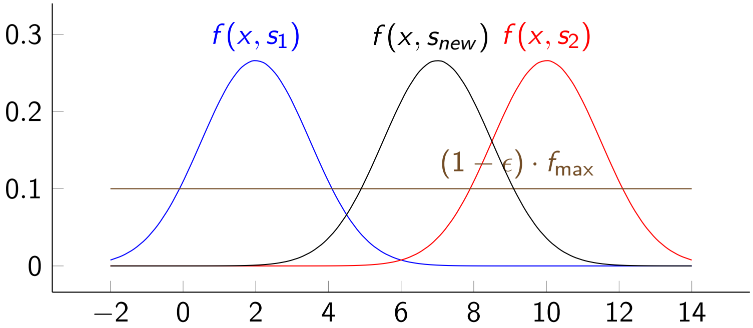


$s_2 = \text{argmax}_{s \in \{s_1, s_2\}} f(7.0, s)$ and $f(7.0, s_2) < (1-\epsilon) \cdot f_{\max}$

Simple univariate perception function

Example

- $S = \{s_1, s_2, s_{new}\}$
- $f(x, s_1) = \mathcal{N}(\mu = 2, \sigma = 0.21)$
- $f(x, s_2) = \mathcal{N}(\mu = 10, \sigma = 0.21)$
- $f(x, s_{new}) = \mathcal{N}(\mu = 7, \sigma = 0.21)$



$s_{new} = \text{argmax}_{s \in \{s_1, s_2, s_{new}\}} f(7.0, s)$ and $f(7, 0, s_{new}) \geq (1-\epsilon)\cdot f_{\max}$

The Planning, Acting, and Learning (PAL) Algorithm

Loop until goal reached:

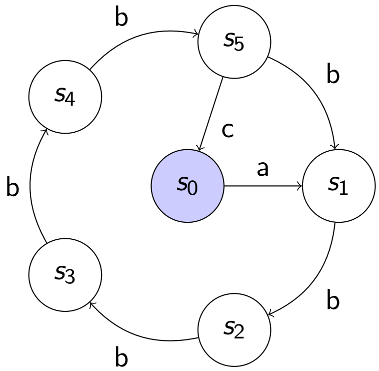
1. generate a plan (if no plan exists, explore the domain)
2. execute the first action in the plan and observe $\boldsymbol{x}$
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state.
4. update the transition function (γ)
5. update the perception function (f)

The Planning, Acting, and Learning (PAL) Algorithm

Loop until goal reached:

1. generate a plan (if no plan exists, explore the domain)
2. execute the first action in the plan and observe $\boldsymbol{x}$
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state.
4. update the transition function (γ)
5. update the perception function (f)

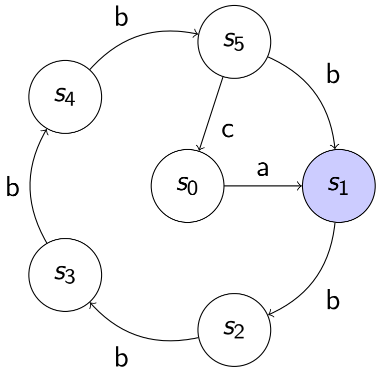
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target

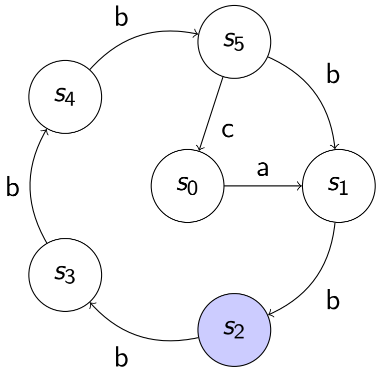
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1

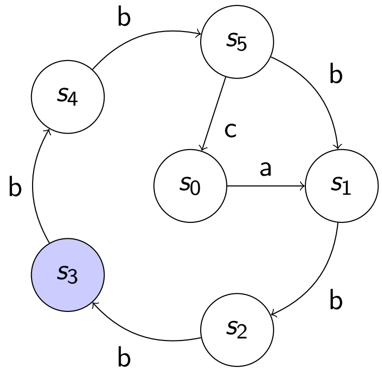
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2

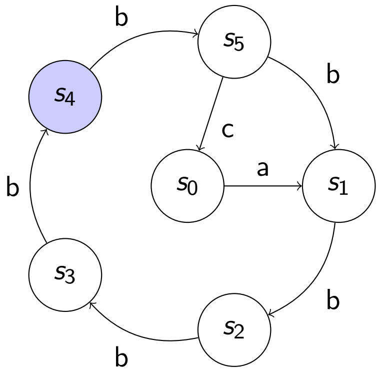
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3

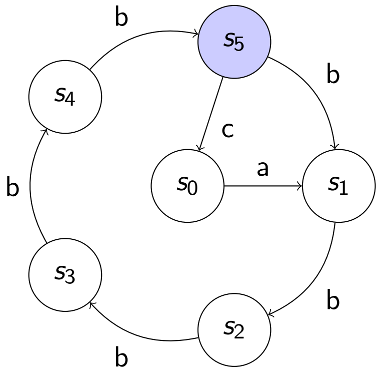
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4

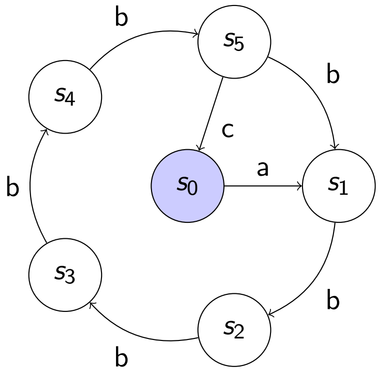
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5

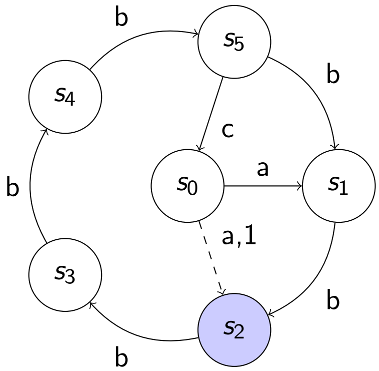
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0

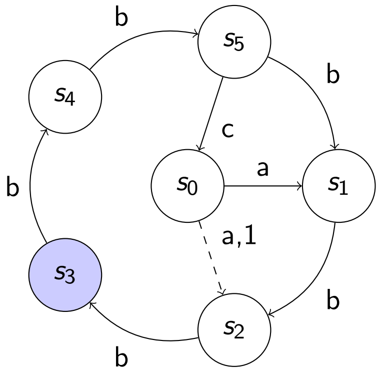
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2

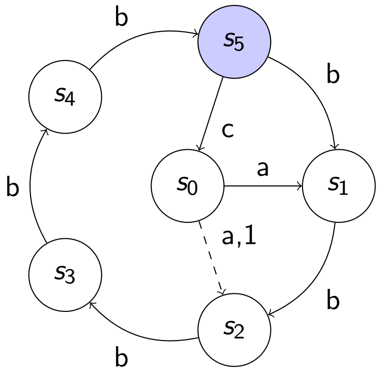
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5

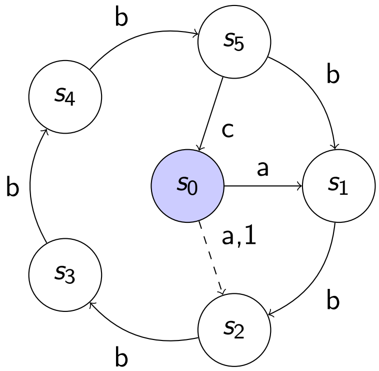
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5
$\mathcal{T}_{11}$	s_5	c	s_0

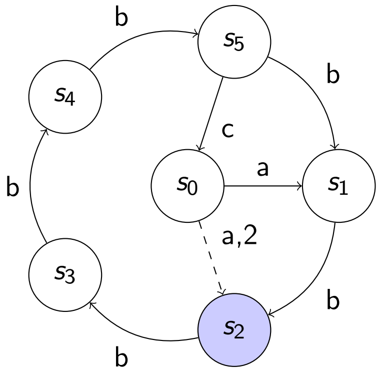
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5
$\mathcal{T}_{11}$	s_5	c	s_0
$\mathcal{T}_{12}$	s_0	a	s_2

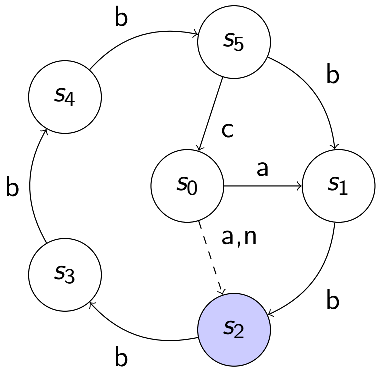
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5
$\mathcal{T}_{11}$	s_5	c	s_0
$\mathcal{T}_{12}$	s_0	a	s_2
$\vdots$			
...	s_0	a	s_2

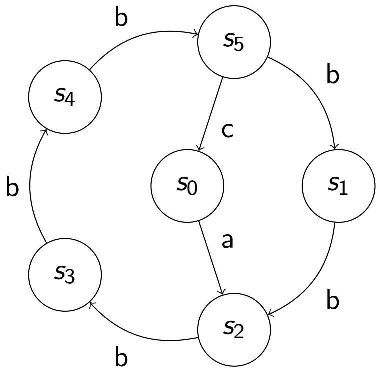
Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5
$\mathcal{T}_{11}$	s_5	c	s_0
$\mathcal{T}_{12}$	s_0	a	s_2
$\vdots$			
...	s_0	a	s_2

Updating Transitions: Example



$\mathcal{T}$

iter	Source	action	Target
$\mathcal{T}_1$	s_0	a	s_1
$\mathcal{T}_2$	s_1	b	s_2
$\mathcal{T}_3$	s_2	b	s_3
$\mathcal{T}_4$	s_3	b	s_4
$\mathcal{T}_5$	s_4	b	s_5
$\mathcal{T}_6$	s_5	c	s_0
$\mathcal{T}_7$	s_0	a	s_2
$\mathcal{T}_8$	s_2	b	s_3
$\mathcal{T}_9$	s_3	b	s_4
$\mathcal{T}_{10}$	s_4	b	s_5
$\mathcal{T}_{11}$	s_5	c	s_0
$\mathcal{T}_{12}$	s_0	a	s_2
$\vdots$			
$\ldots$	s_0	a	s_2

Updating transitions

- The update of transitions is based on the current model $\gamma(s,a)$ and the set of transitions observed so far $\mathcal{T} : \langle s, a, s' \rangle$

- The agent can be more or less careful in the revision: Parameter $\alpha \in [0,1]$ - the higher the value of α the more careful the agent is in the revision.

$$\underset{s' \in S}{\operatorname{argmax}}(\alpha \cdot \overbrace{\mathbb{1}_{s'=\gamma(s,a)}}^{\textbf{current model}} + (1-\alpha) \cdot \overbrace{|\{i \mid \mathcal{T}_i = \langle s, a, s' \rangle\}|}^{\textbf{observations}})$$

The Planning, Acting, and Learning (PAL) Algorithm

Loop until goal reached:

1. generate a plan (if no plan exists explore the domain, e.g., with a random policy)
2. execute the first action in the plan and observe $\boldsymbol{x}$ through sensors
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state (s_{new})
4. update the transition function (γ)
5. update the perception function (f)

The Planning, Acting, and Learning (PAL) Algorithm

Loop until goal reached:

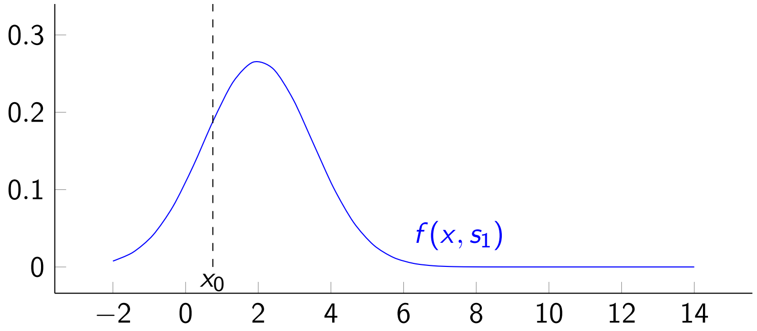
1. generate a plan (if no plan exists explore the domain, e.g., with a random policy)
2. execute the first action in the plan and observe $\boldsymbol{x}$ through sensors
3. decide in which state you are based on maximum likelihood. If the likelihood is too low for all the existing states, create a new state (s_{new})
4. update the transition function (γ)
5. update the perception function (f)

Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.

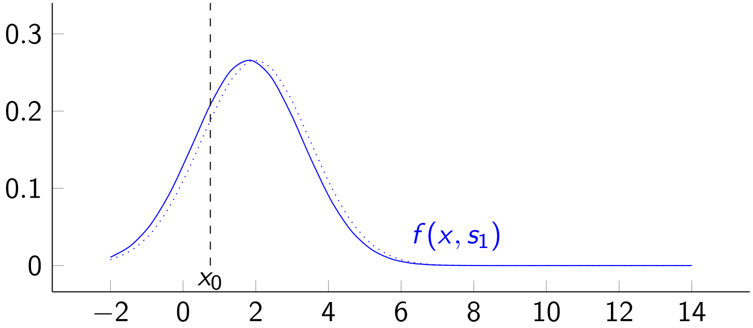
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



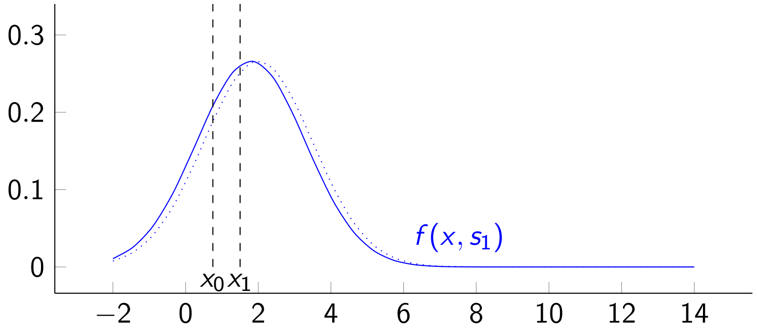
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



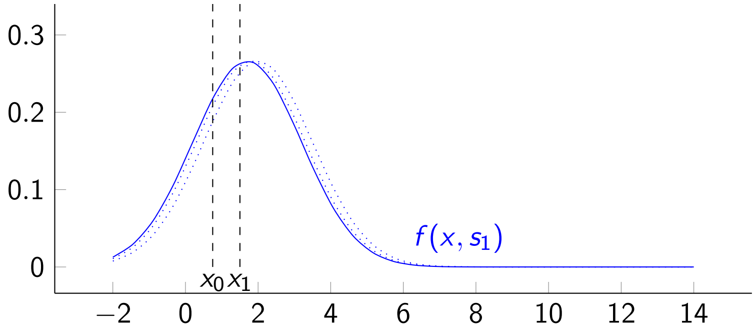
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



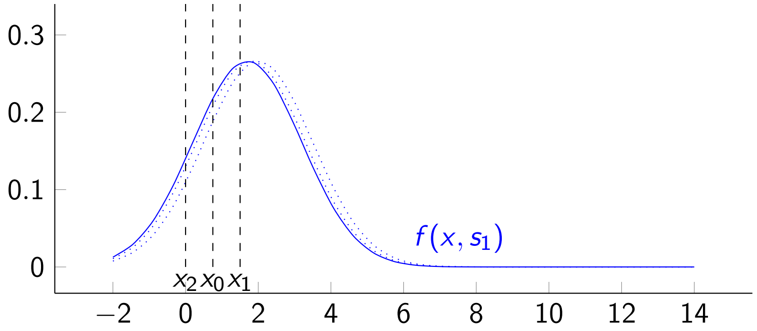
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



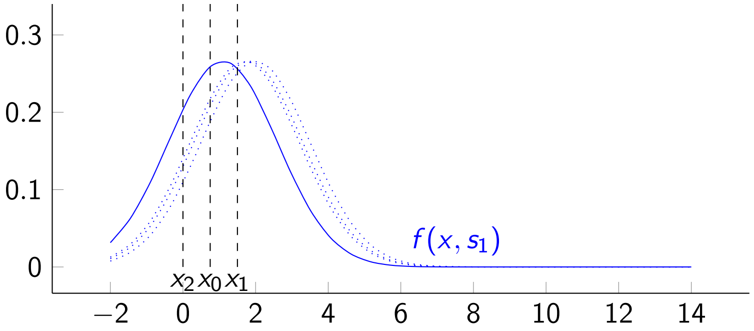
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



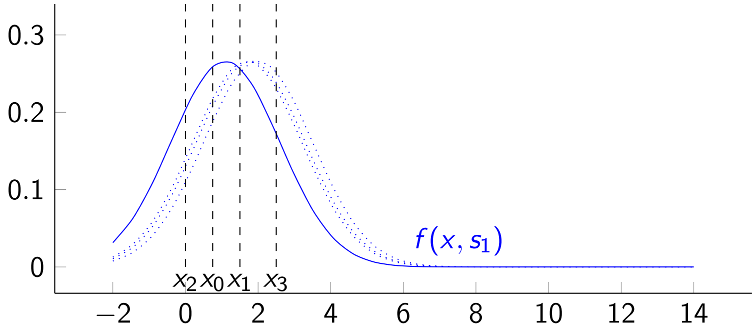
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



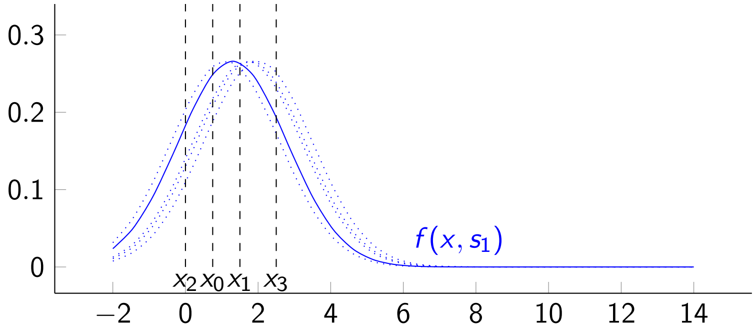
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



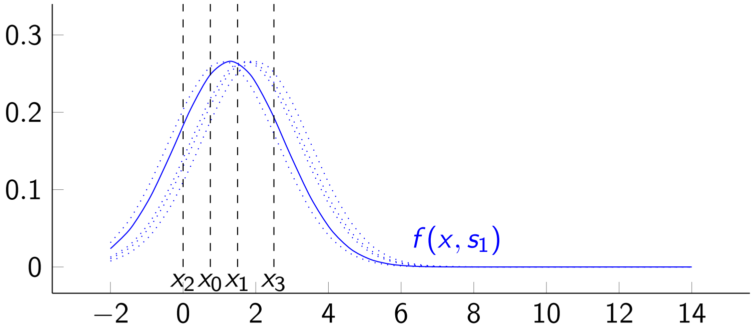
Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



Updating the Perception Function: Example

The idea: We update the perception function $f(\boldsymbol{x}, s)$ to maximize the likelihood of the entire set of observations extended with the new observation.



Updating the Perception Function

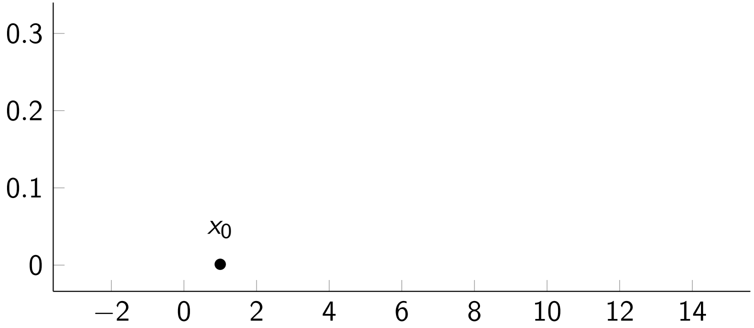
- The update of perception function is based on the current perception function $f(\boldsymbol{x}, s)$ and the set of observations $\mathcal{O} : \langle s, \boldsymbol{x} \rangle$.

- Also in this case the agent can be more or less careful in the revision: Parameter $\beta \in [0, 1]$ - the higher the value of β the more careful the agent is in the revision.

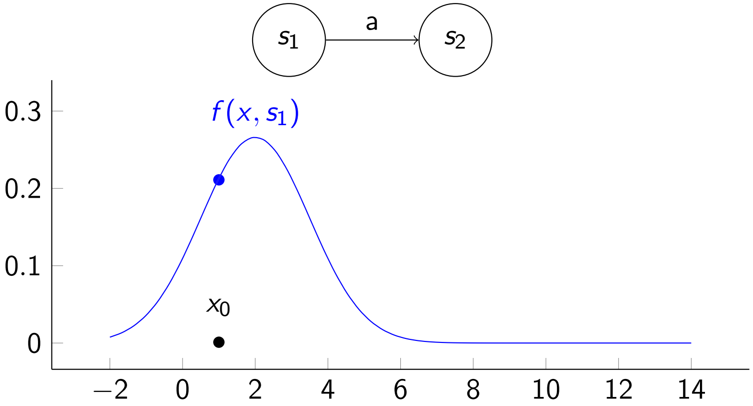
Model coherence

- **Objective:** to estimate the quality of the generated model
- **Approach:** to measure the coherence between an abstract model with perception function and the real world.
- **Idea:** We introduce a measure called *divergence*.

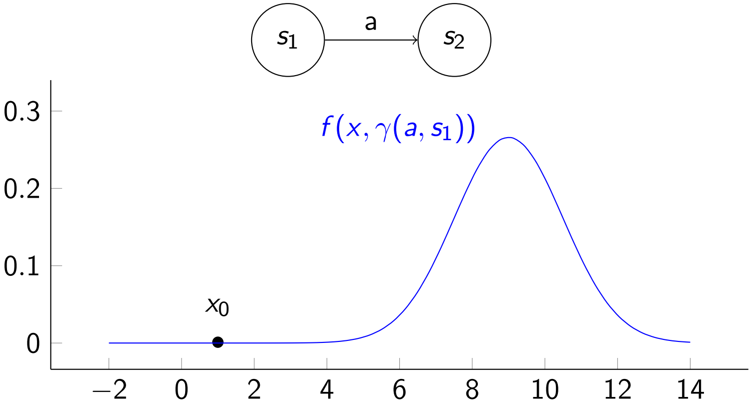
Estimating the coherence of the model



Estimating the coherence of the model

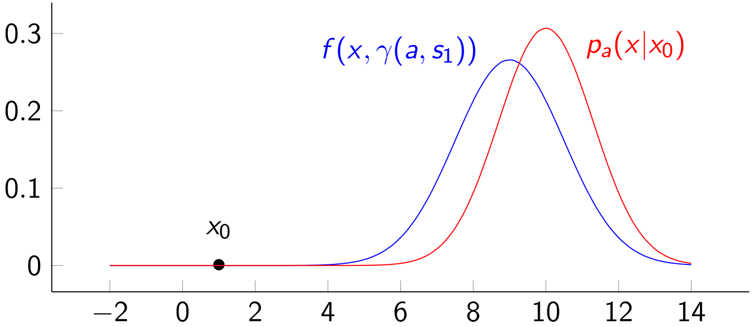


Estimating the coherence of the model



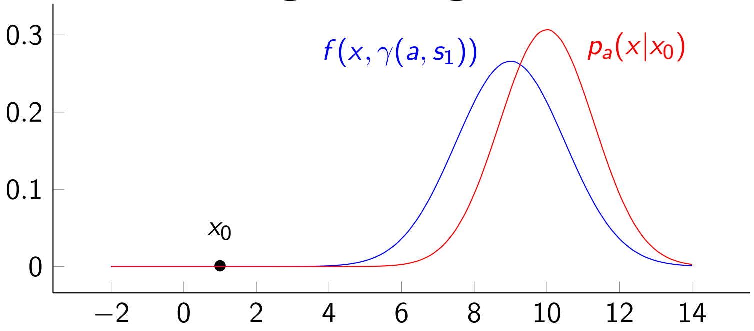
Estimating the coherence of the model

$$\begin{tikzcd} s_1 \arrow[r, "a"] & s_2 \end{tikzcd}$$



Estimating the coherence of the model

$$\begin{tikzcd} s_1 \arrow[r, "a"] & s_2 \end{tikzcd}$$

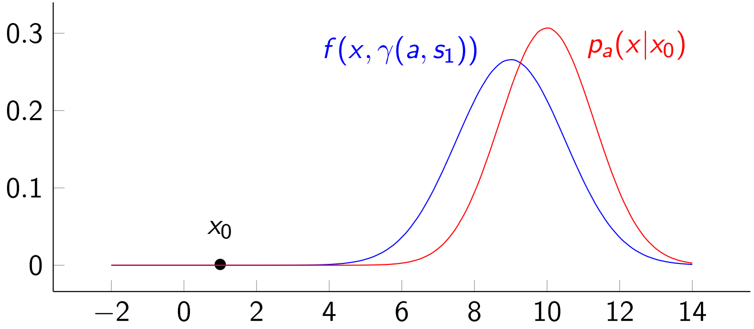


$$\mathrm{KL}(f(x, \gamma(a, s_1))||p_a(x|x_0))$$

$\mathrm{KL}(p(\boldsymbol{x})||q(\boldsymbol{x}))$ is the Kullback-Leibler divergence (also called relative entropy) is a measure of how one probability distribution $p(\boldsymbol{x})$ is different from a second one $q(\boldsymbol{x})$.

Estimating the coherence of the model

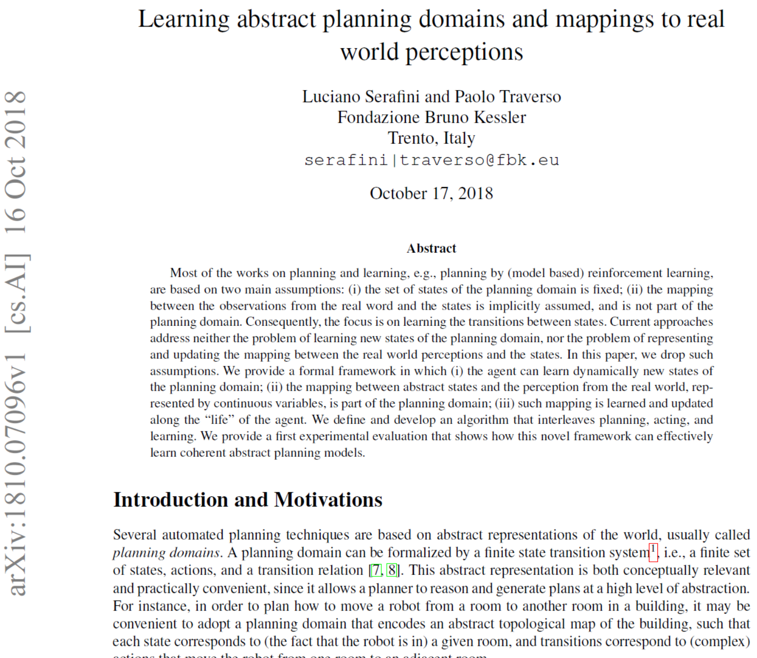
$$\begin{tikzcd} s_1 \arrow[r, "a"] & s_2 \end{tikzcd}$$



$$\sum_{i=1}^{n} \sum_{a \in A} \mathrm{KL}(f(x, \gamma(a, s_i)) || p_a(x|x_i))$$

$\mathrm{KL}(p(\boldsymbol{x})||q(\boldsymbol{x}))$ is the Kullback-Leibler divergence (also called relative entropy) is a measure of how one probability distribution $p(\boldsymbol{x})$ is different from a second one $q(\boldsymbol{x})$.

Reference



Learning abstract planning domains and mappings to real world perceptions

Luciano Serafini and Paolo Traverso
Fondazione Bruno Kessler
Trento, Italy
`serafini|traverso@fbk.eu`

October 17, 2018

Abstract

Most of the works on planning and learning, e.g., planning by (model based) reinforcement learning, are based on two main assumptions: (i) the set of states of the planning domain is fixed; (ii) the mapping between the observations from the real word and the states is implicitly assumed, and is not part of the planning domain. Consequently, the focus is on learning the transitions between states. Current approaches address neither the problem of learning new states of the planning domain, nor the problem of representing and updating the mapping between the real world perceptions and the states. In this paper, we drop such assumptions. We provide a formal framework in which (i) the agent can learn dynamically new states of the planning domain; (ii) the mapping between abstract states and the perception from the real world, represented by continuous variables, is part of the planning domain; (iii) such mapping is learned and updated along the "life" of the agent. We define and develop an algorithm that interleaves planning, acting, and learning. We provide a first experimental evaluation that shows how this novel framework can effectively learn coherent abstract planning models.

Introduction and Motivations

Several automated planning techniques are based on abstract representations of the world, usually called *planning domains*. A planning domain can be formalized by a finite state transition system[1], i.e., a finite set of states, actions, and a transition relation [7, 8]. This abstract representation is both conceptually relevant and practically convenient, since it allows a planner to reason and generate plans at a high level of abstraction. For instance, in order to plan how to move a robot from a room to another room in a building, it may be convenient to adopt a planning domain that encodes an abstract topological map of the building, such that each state corresponds to (the fact that the robot is in) a given room, and transitions correspond to (complex) actions that move the robot from one room to an adjacent room.

Reference

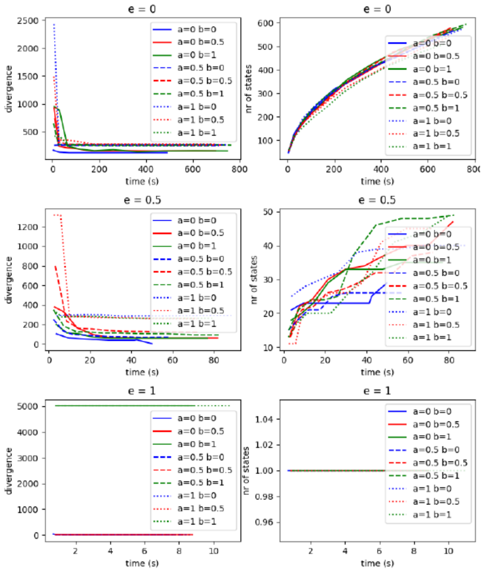


Figure 3: Experiments with 5×5 building. a, b, and e stand for α, β, and ϵ, respectively .

11

Future Challenges

- Non-Deterministic/Probabilistic (MDP) planning domains
- Partially Observable planning domains (POND and POMDP)
- State merging and state elimination
- From abstract states to state variables
- Perception function with NN (e.g., Logic Tensor Network)
- Scalability

THANK YOU FOR YOUR ATTENTION!