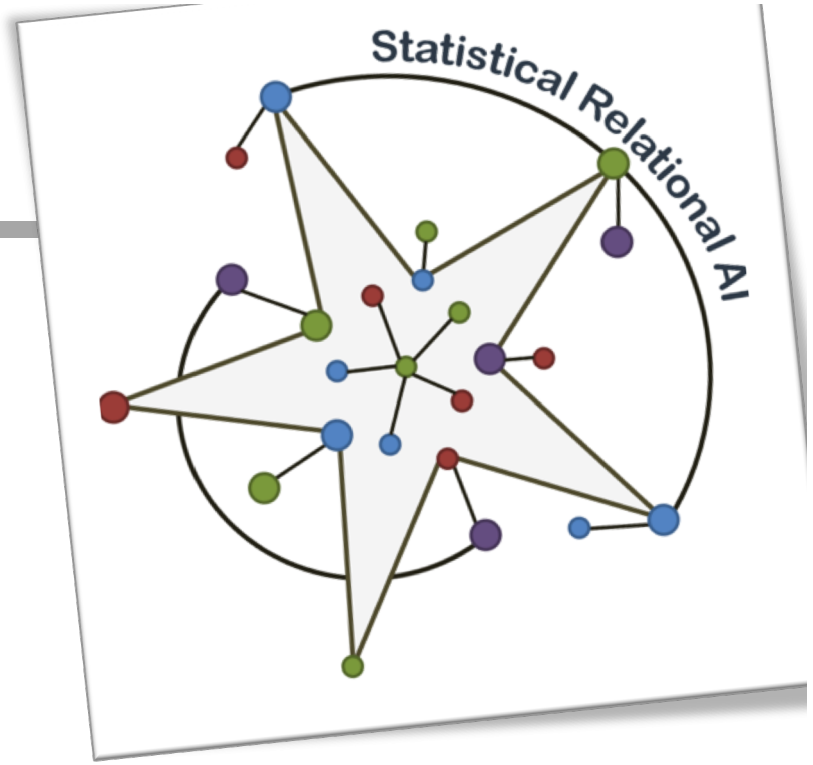
Learning Probabilistic Relational Models



Statistical Relational AI

Tutorial at KI-2018

Tanya Braun, Universität zu Lübeck

Kristian Kersting, Technische Universität Darmstadt

Ralf Möller, Universität zu Lübeck

UNIVERSITÄT ZU LÜBECK

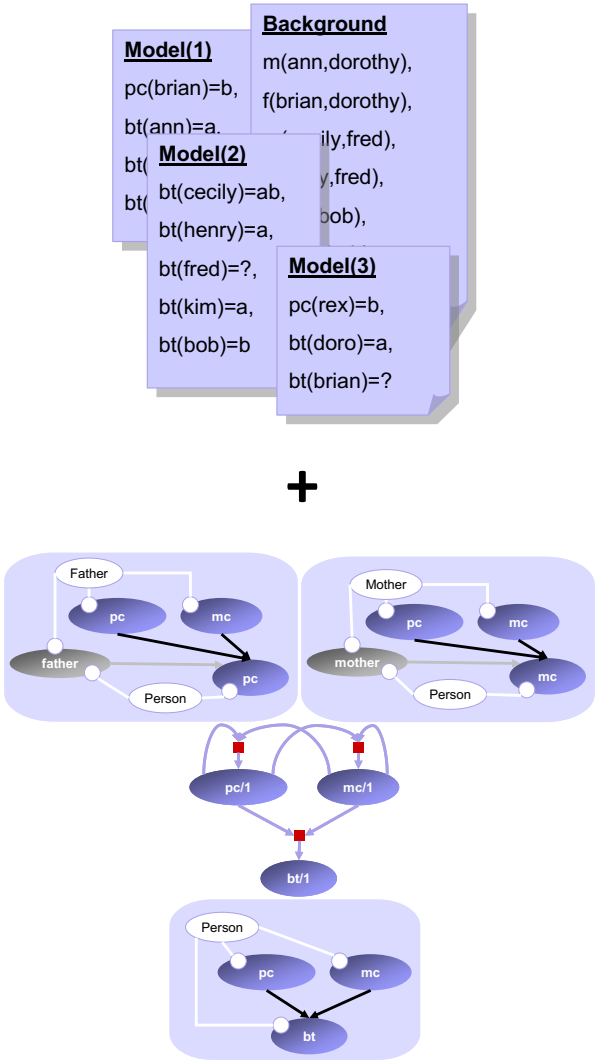
TECHNISCHE UNIVERSITÄT DARMSTADT

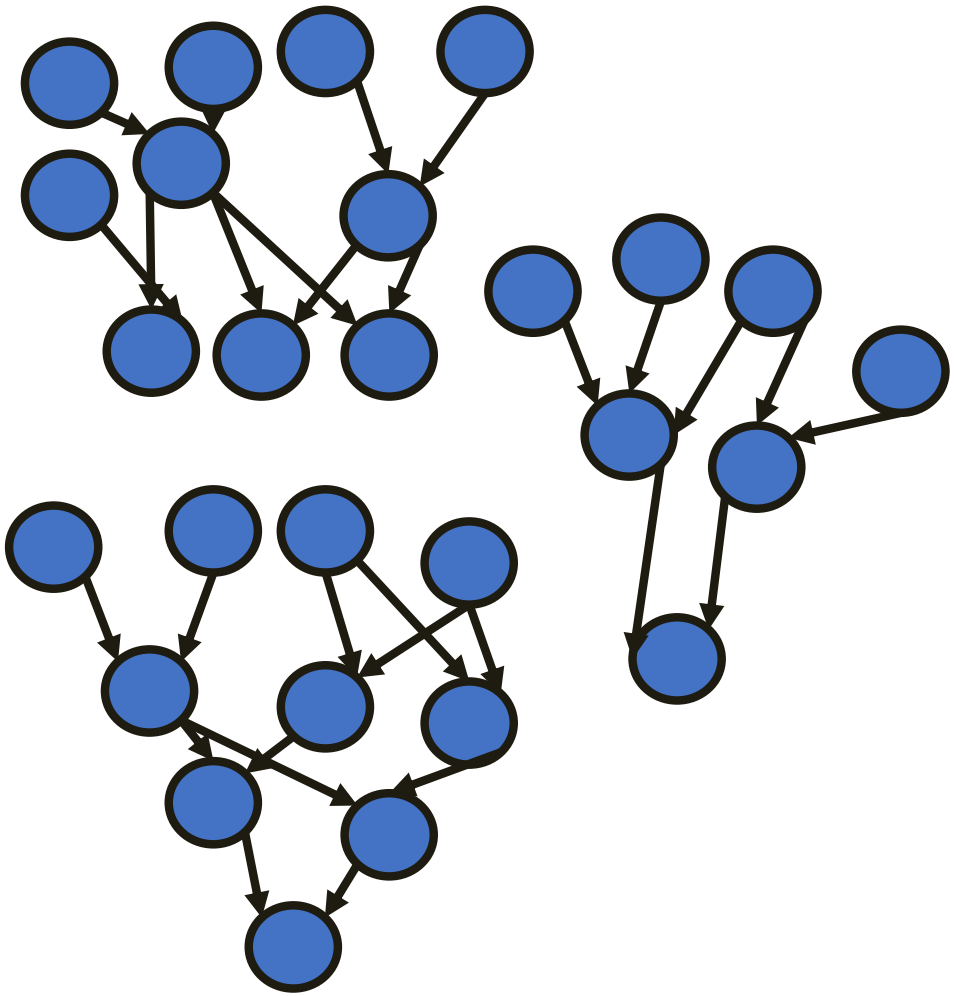
How do we learn relational models?

1. Parameter estimation
2. Vanilla relational (structure) learning
3. Boosting

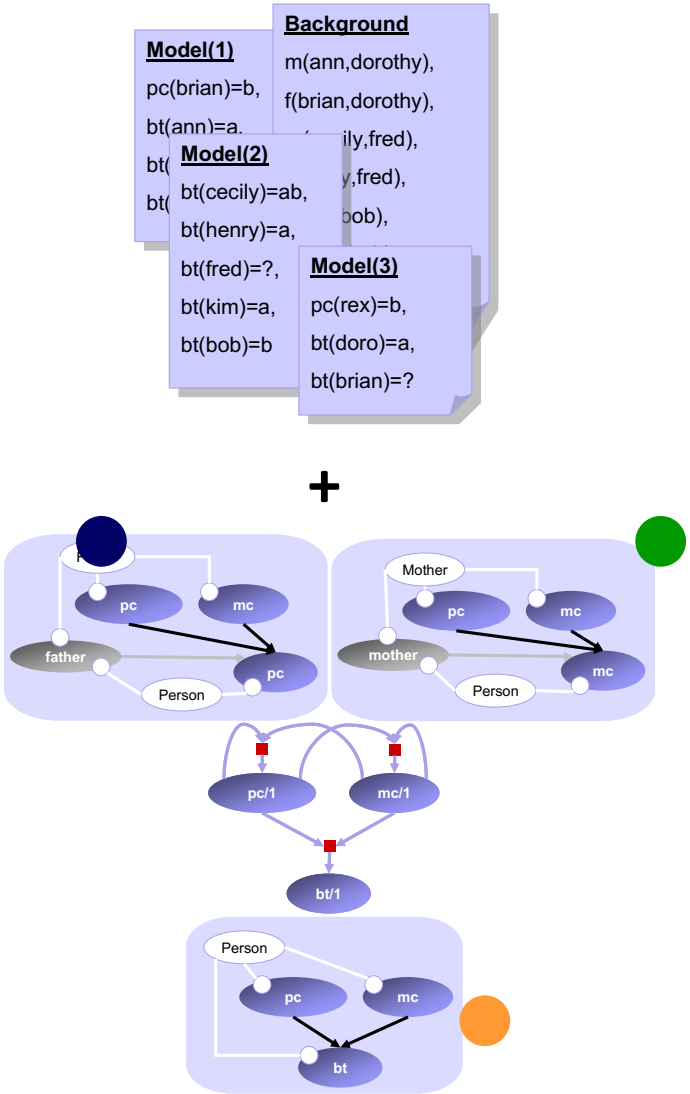
Parameter estimation for relational models

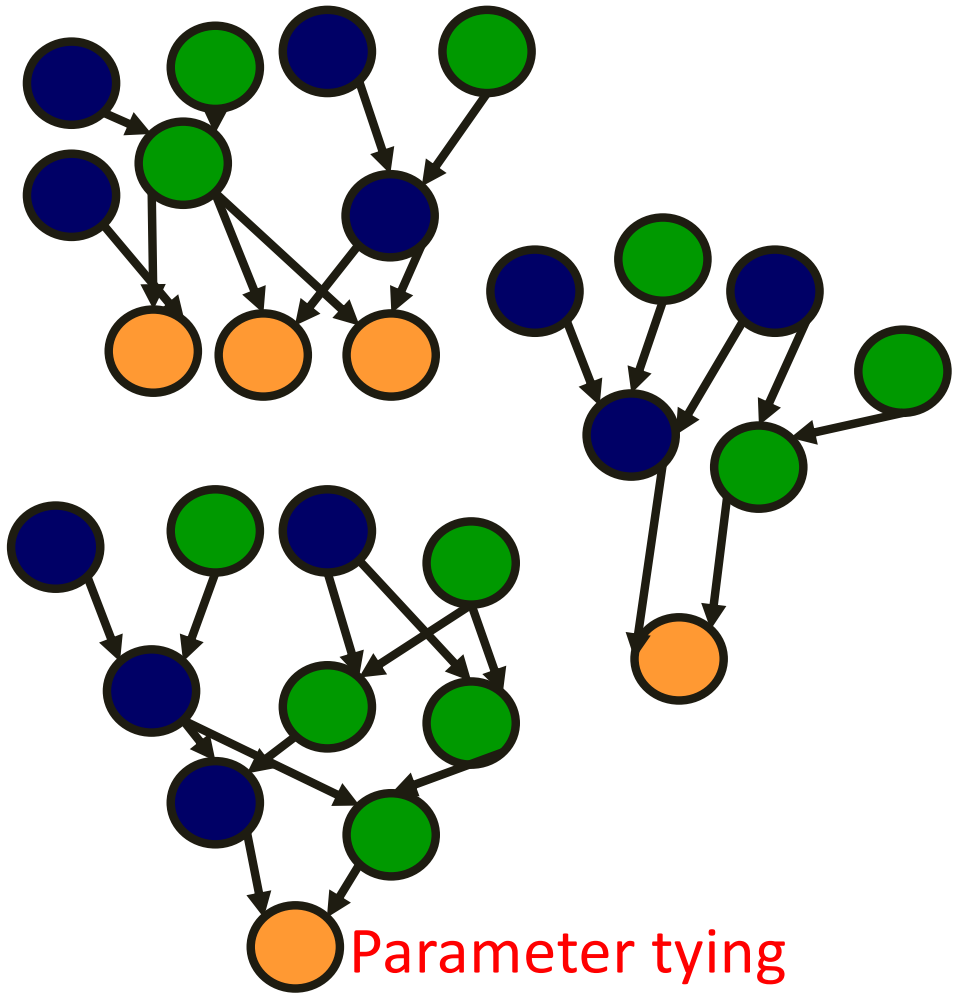
Relational Parameter Estimation



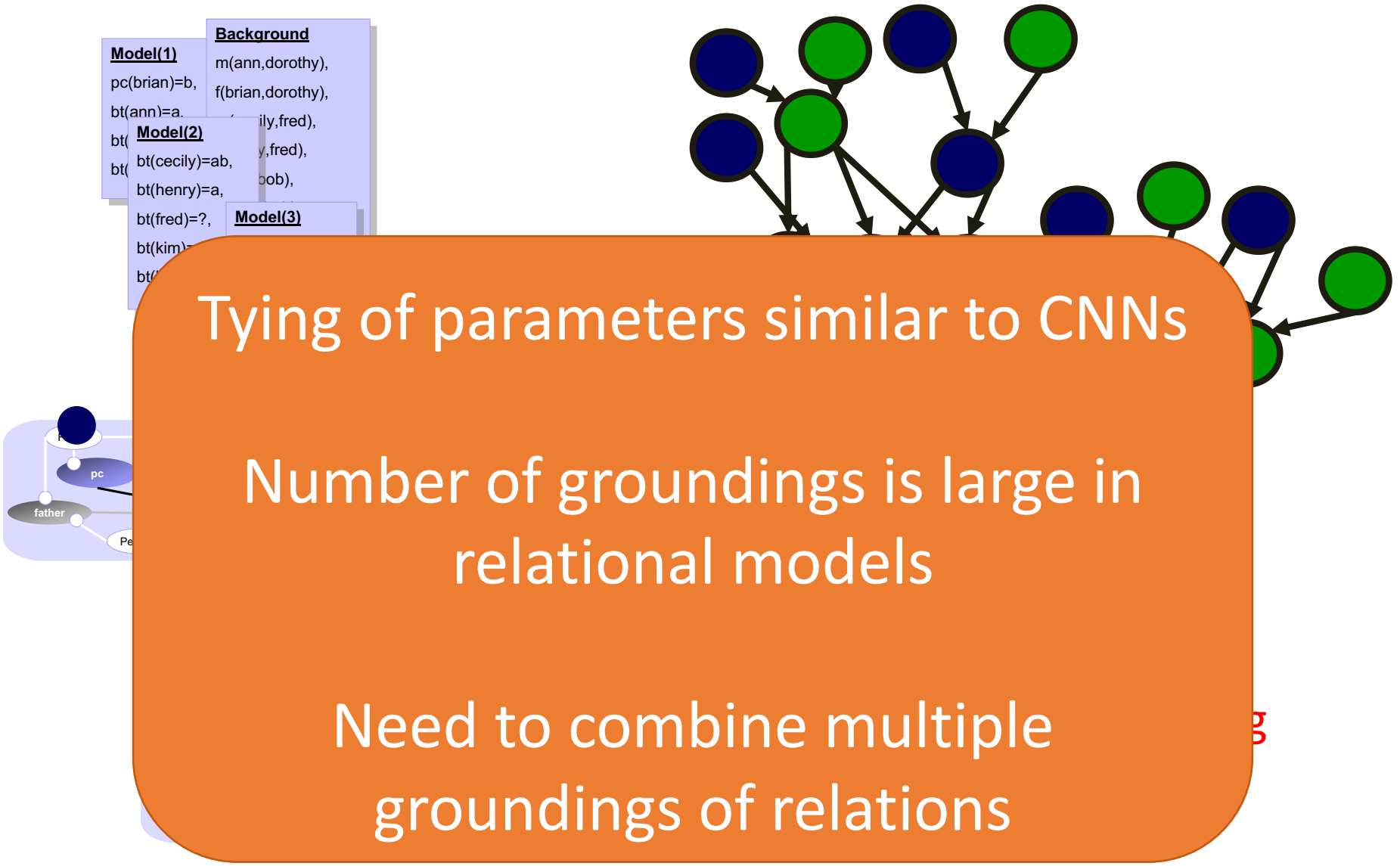


Relational Parameter Estimation





Relational Parameter Estimation



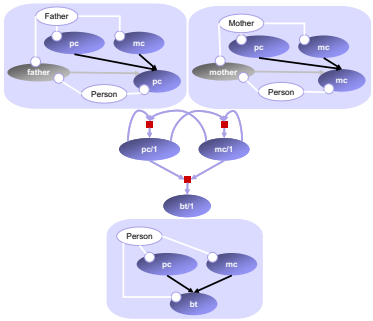
Tying of parameters similar to CNNs

Number of groundings is large in relational models

Need to combine multiple groundings of relations

So, we can apply „standard“ EM

Logic Program L



Initial Parameters q0

Current Model (M,qk)

iterate until convergence

Expectation

Inference

Background
m(ann,dorothy),
f(brian,dorothy),
...ily,fred),

Model(1)
pc(brian)=b,
bt(ann)=a,
bt(
bt(

Model(2)
bt(cecily)=ab,
bt(henry)=a,
bt(fred)=?,
bt(kim)=a,
bt(bob)=b

Model(3)
pc(rex)=b,
bt(doro)=a,
bt(brian)=?

Expected counts of a clause

$$\sum_{\text{Ground Instance GI}} \sum_{\text{DataCase DC}} P(\text{head(GI)}, \text{body(GI)} \mid \text{DC})$$

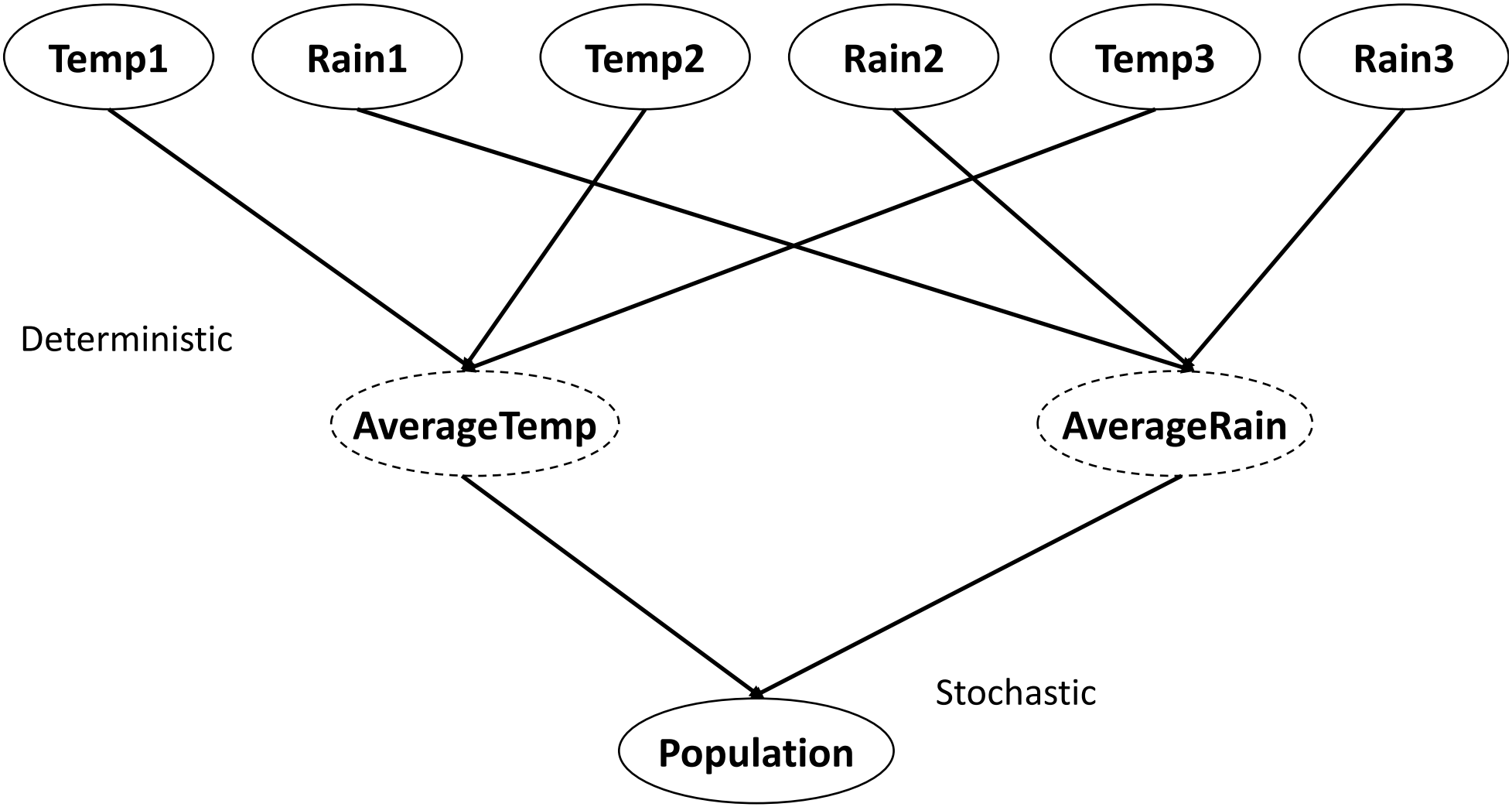
$$\frac{\sum_{\text{Ground Instance GI}} \sum_{\text{DataCase DC}} P(\text{head(GI)}, \text{body(GI)} \mid \text{DC})}{\sum_{\text{Ground Instance GI}} \sum_{\text{DataCase DC}} P(\text{body(GI)} \mid \text{DC})}$$

Maximization

Update parameters (ML, MAP)

Aggregators:

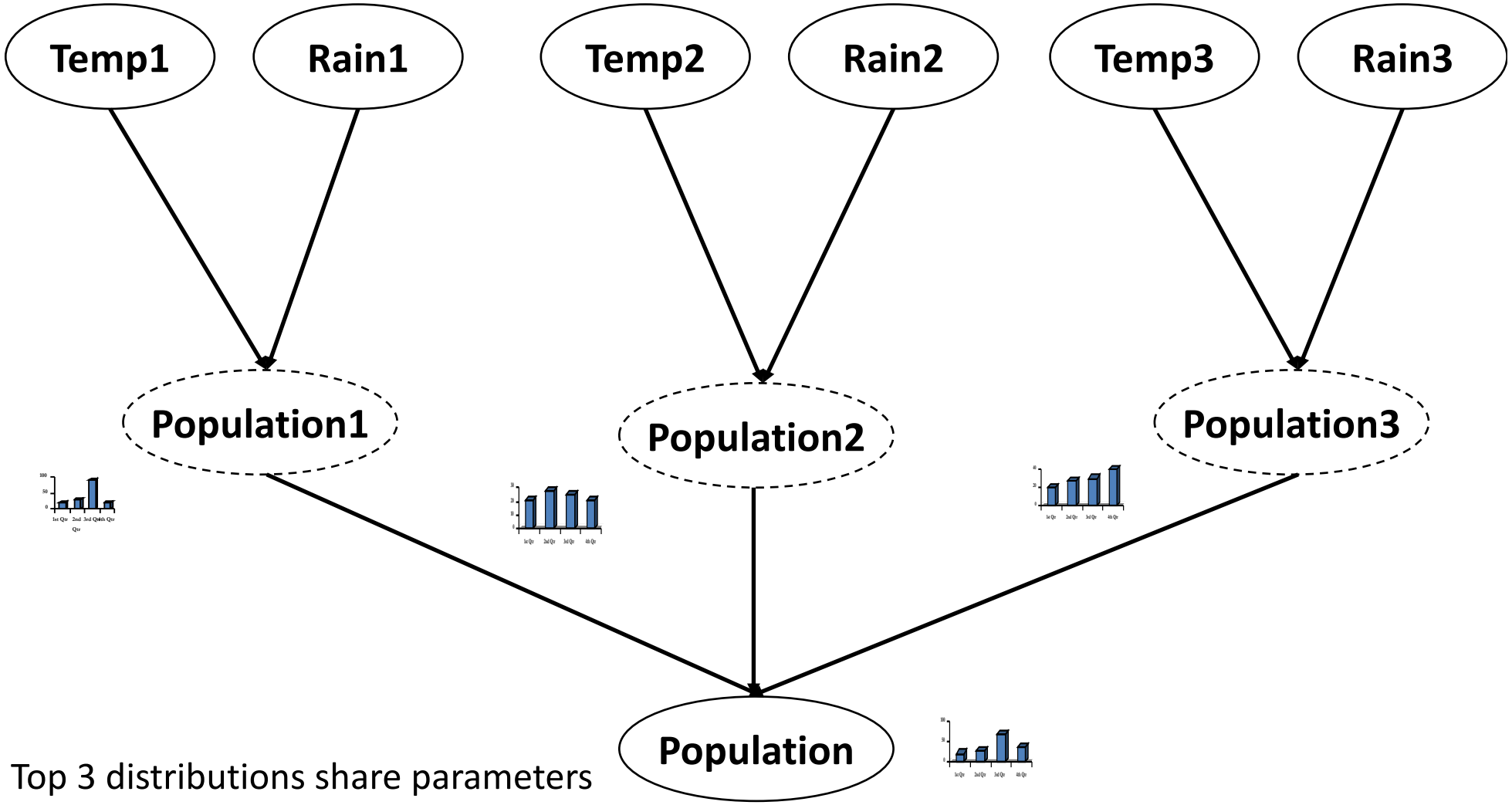
We may also have to combine instances of the same rule



Problem: Does not take into account the interaction between Rain and Temp

Combining Rules:

We may also have to combine instances of the same rule



- Top 3 distributions share parameters
- The 3 distributions are combined into one final distribution
- Gradient-descent and EM Methods exist

Of course, we can also make use of gradients: E.g. MLN Weight Learning

- Parameter tying: Groundings of same clause

$$\frac{\partial}{\partial w_i} \log P_w(x) = n_i(x) - E_w[n_i(x)]$$

No. of times clause *i* is true in data

Expected no. times clause *i* is true according to MLN

- It is #P-complete to count the number of true groundings. Therefore, one often sticks to approximations such as
  - Generative learning: Pseudo-likelihood
  - Discriminative learning: Cond. likelihood

Pseudo-likelihood

Function to optimize:

$$PL(x) = \prod_l P(X_l = x_l \mid MB(x_l))$$

$$\log PL(x) = \sum_l \log P(X_l = x_l \mid MB(x_l))$$

$$P(X_l = x_l \mid MB(x_l)) = \frac{P(x)}{P(x_{[X_l=0]}) + P(x_{[X_l=1]})}$$

$$= \frac{1/Z \exp(\Sigma w_i n_i(x))}{1/Z \exp(\Sigma w_i n_i(x_{[X_l=0]})) + 1/Z \exp(\Sigma w_i n_i(x_{[X_l=1]}))}$$

Gradient:

$$\frac{\partial}{\partial w_j} \log PL(x) = \sum_l n_j(x) - P(X_l = 0 \mid MB(X_l))\, n_j(x_{[X_l=0]}) - P(X_l = 1 \mid MB(X_l))\, n_j(x_{[X_l=1]})$$

$$= \sum_l n_j(x) - E_{x'_l}[n_j(x_{[X_l=x'_l]})]$$

Pseudo-likelihood

Function to optimize:

$$PL(x) = \prod_l P(X_l = x_l \mid MB(x_l))$$

$$\log PL(x) = \sum_l \log P(X_l = x_l \mid MB(x_l))$$

While effective, still hard to count in many data sets

Approximate counting techniques exist (Sarkhel et al. AAAI 2016, Das et al. SDM 2016)

What's Different to standard graphical models?

- Counting groundings need to be efficient
- Ensure parameter tying
- Population growth

Top-down approach:

- GSL[Kok & Domingos, 2005], DSL[Biba et al., 2008]
- Start from unit clauses and search for new clauses

Bottom-up approach:

- BUSL [Mihalkova & Mooney, 2007], Hypergraph Lifting [Kok & Domingos, 2009], Structural Motifs [Kok & Domingos , 2010]
- Use data to generate candidate clauses

Max-Margin Approach:

- Discriminative learning [Huynh & Mooney, 2008]
- Effectively learns horn clauses
- Uses regularization to force parameters to zero
- Later extended to online setting

Structure Learning of Relational Models

Inductive Logic Programming = Machine Learning + Logic Programming

The Problem Specification

[Muggleton, De Raedt JLP96]

- *Given*:
  - *Examples:* first-order atomic formulas (atoms), each labeled positive or negative.
  - *Background knowledge:* definite clause (if-then rules) theory.
  - *Language bias:* constraints on the form of interesting new rules (clauses).

ILP Specification

- *Find:*

  A hypothesis *h* that meets the language constraints and that, when conjoined with *B*, implies (lets us prove) all of the positive examples but none of the negative examples.

- To handle real-world issues such as noise, we often relax the requirements, so that *h* need only entail significantly more positive examples than negative examples.

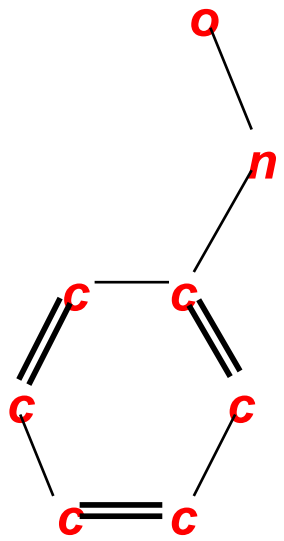
Illustration

Find set of general rules

mutagenic(X) :- atom(X,A,c),charge(X,A,0.82)
mutagenic(X) :- atom(X,A,n),...

Examples E

Pos(mutagenic(m1)
Pos(mutagenic(m2)
Neg(mutagenic(m3)



Background Knowledge B

molecule(m1) molecule(m2)
atom(m1,a11,c) atom(m2,a21,o)
atom(m1,a12,n) atom(m2,a22,n)
bond(m1,a11,a12) bond(m2,a21,a22)
....

A Common Approach

- Use a greedy covering algorithm.

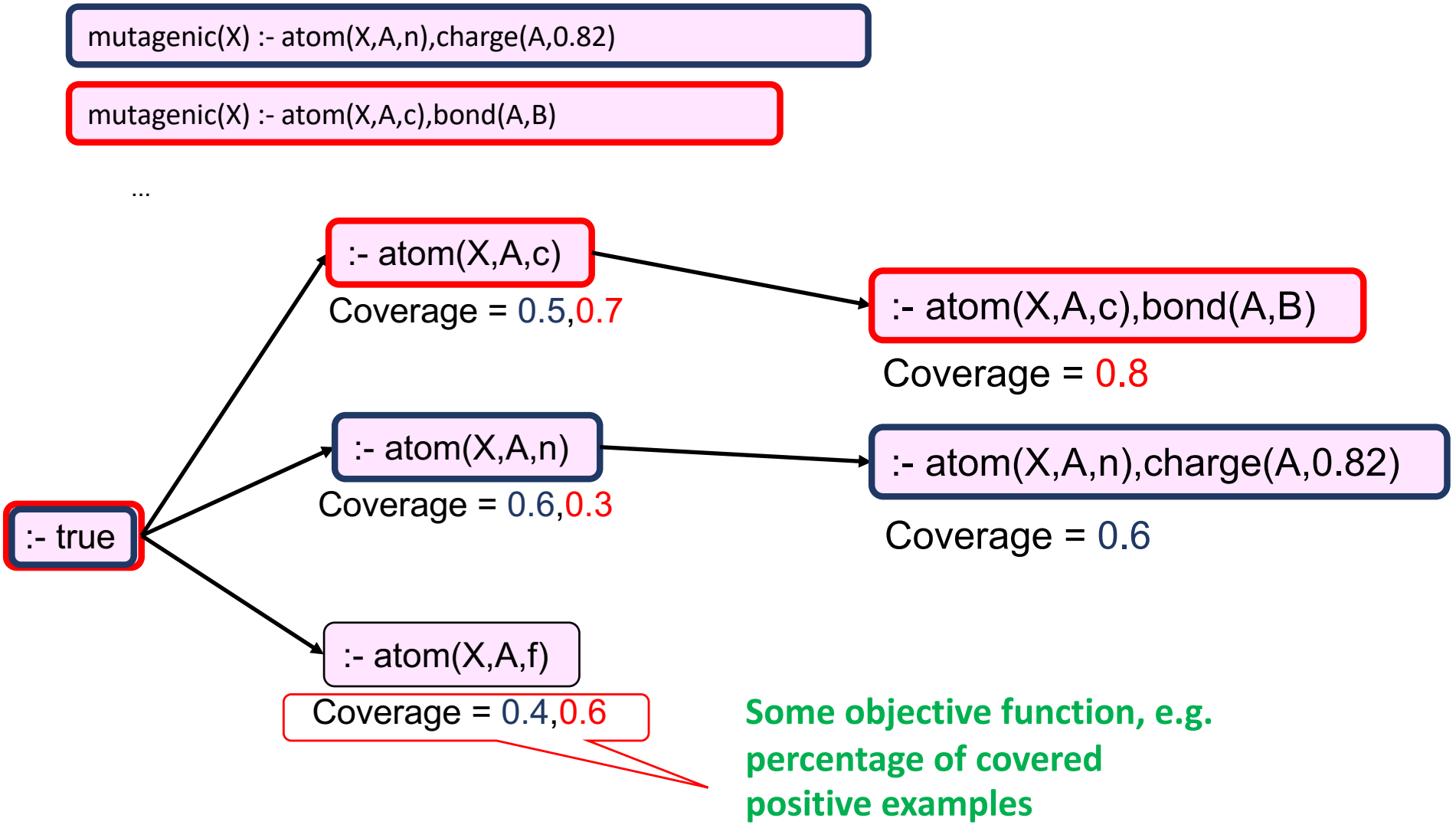
  Repeat while some positive examples remain uncovered (not entailed):

  1. Find a *good clause* (one that covers as many positive examples as possible but no/few negatives).
  2. Add that clause to the current theory, and remove the positive examples that it covers.

- ILP algorithms use this approach but vary in their method for finding a *good clause*.

Example ILP Algorithm: FOIL

[Quinlan MLJ 5:239-266, 1990]



Vanilla Structure learning for Probabilistic relational models

Vanilla SRL Approach [De Raedt, Kersting ALT04]

mutagenic(X) :- atom(X,A,n),charge(A,0.82)

mutagenic(X) :- atom(X,A,c),bond(A,B)

=0.882

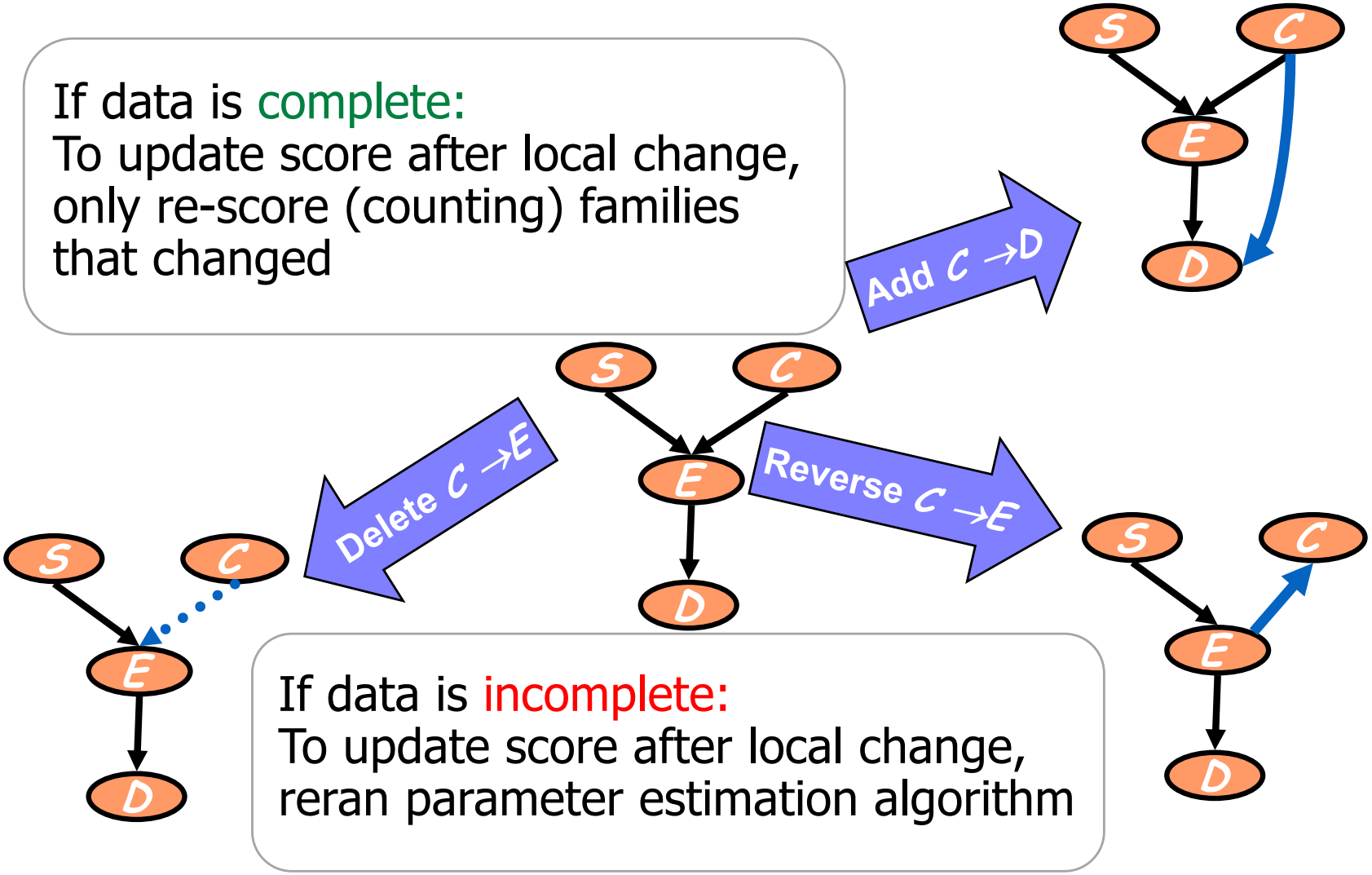
…

- Traverses the hypotheses space a la ILP
- Replaces ILP's 0-1 covers relation by a "smooth", probabilistic one [0,1]

$$\begin{aligned} \text{cover}(e, H, B) &= P(e|H, B) \\ \text{cover}(E, H, B) &= \prod_{e \in E} \text{cover}(e, H, B) \end{aligned}$$

So, essentially like in the propositional case !

If data is complete:
To update score after local change, only re-score (counting) families that changed

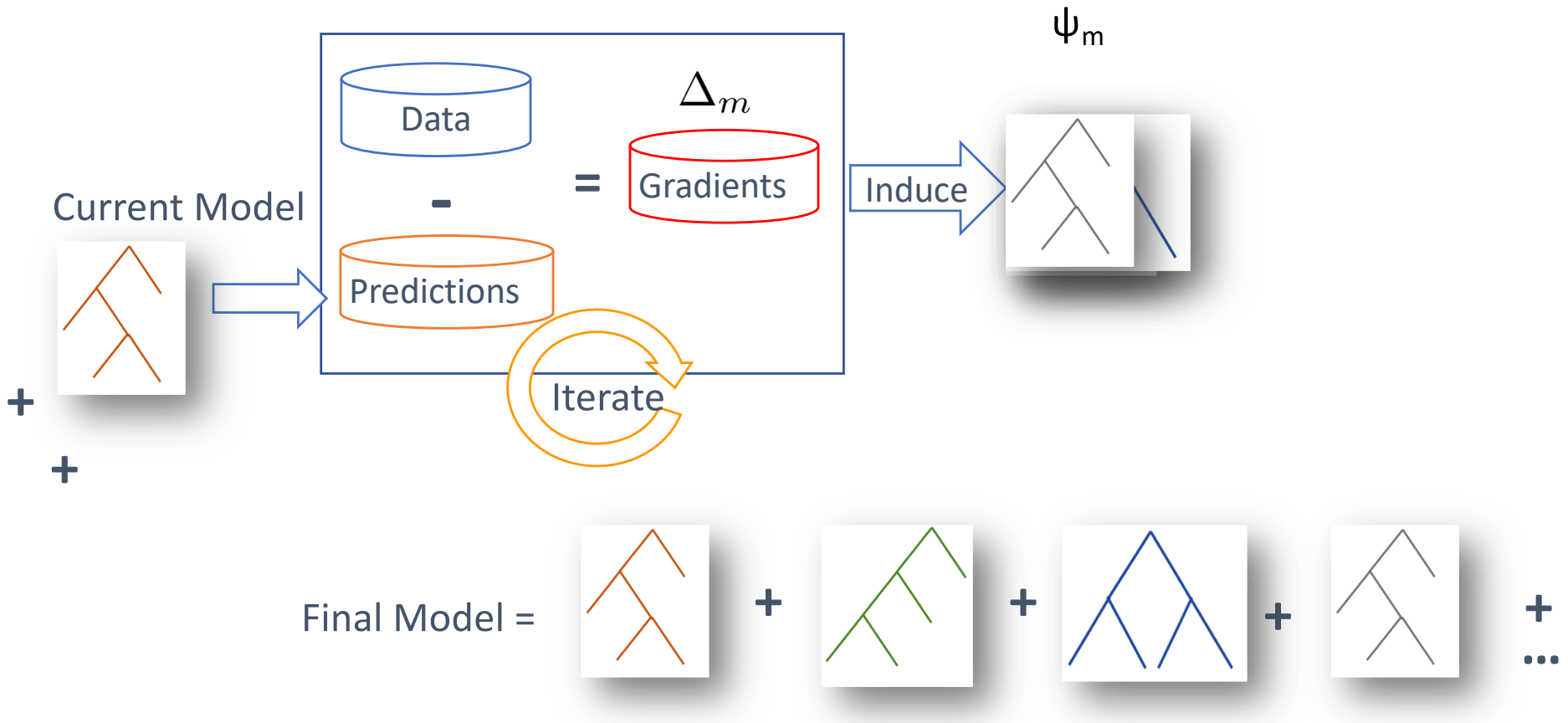


If data is incomplete:
To update score after local change, reran parameter estimation algorithm

Relational Boosting

Relational Gradient Boosting

Learn multiple weak models rather than a single complex model



Friedman et al 2001, Dietterich et al. 2004, Natarajan et al. MLJ 2012

Functional Gradients for SRL Models

- Probability of an example

$$P(x_i = true|\mathbf{Pa}(x_i)) = \frac{e^{\psi(x_i;\mathbf{Pa}(x_i))}}{e^{\psi(x_i;\mathbf{Pa}(x_i))} + 1}$$

x	Δ
target(x1)	0.7
target(x2)	-0.2
target(x3)	-0.9

- Functional gradient
  - Maximize

$$LL(\mathbf{X} = \mathbf{x}) = \sum_{x_i \in \mathbf{x}} \log P(x_i|\mathbf{Pa}(x_i))$$

  - Gradient of log-likelihood w.r.t ψ

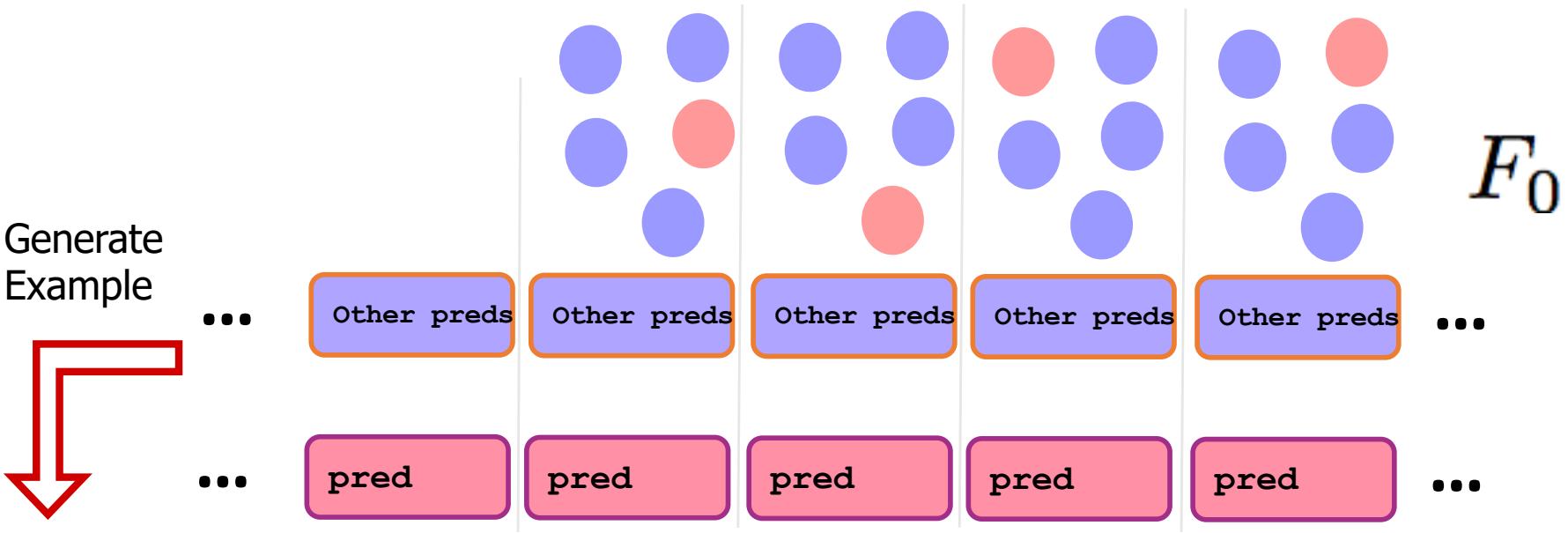
$$\Delta(x_i) = \frac{\partial \log P(\mathbf{X} = \mathbf{x})}{\partial \psi(x_i; \mathbf{Pa}(x_i))} = I(x_i = true; \mathbf{Pa}(x_i)) - P(x_i = true; \mathbf{Pa}(x_i))$$

  - Sum all gradients to get final ψ

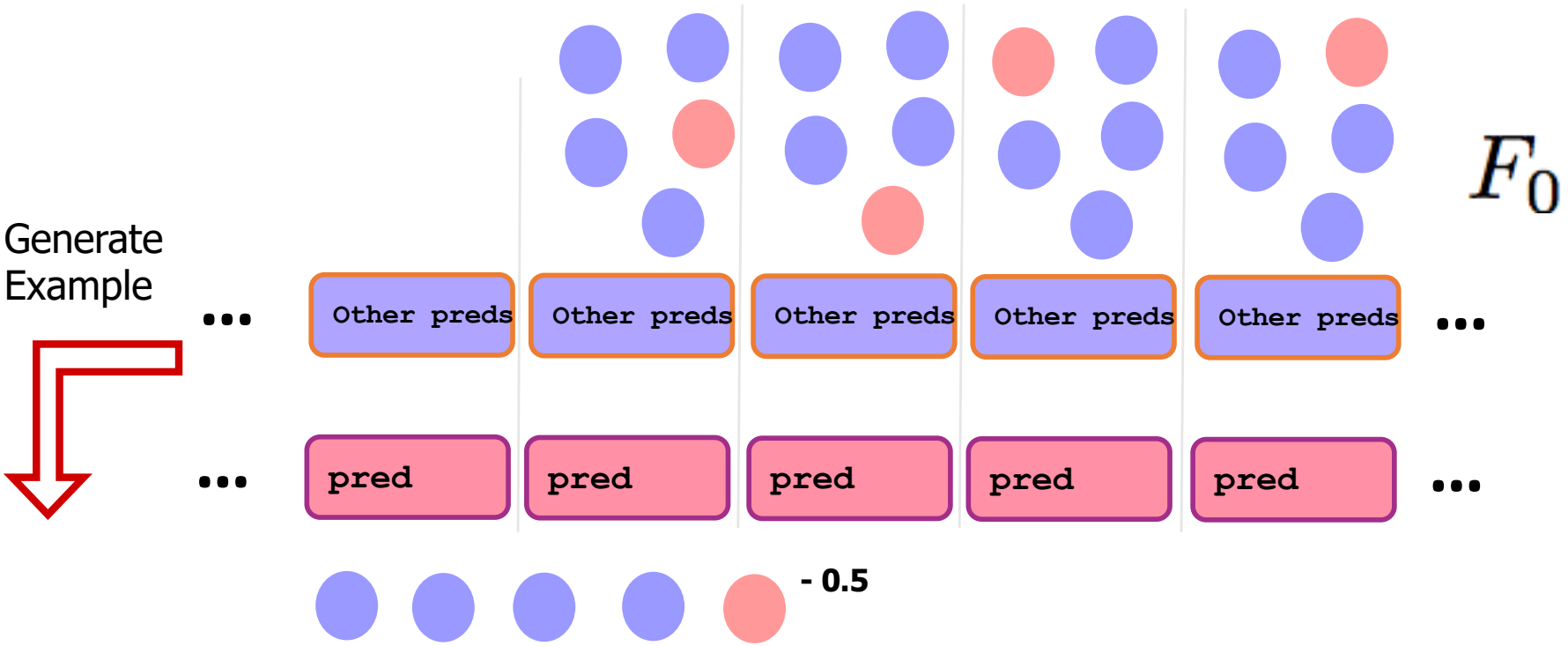
$$\psi_m = \psi_0 + \Delta_1 + \ldots + \Delta_m$$

Can be extended to multiple SRL models & in presence of hidden data

Boosting RDNs/MLNs

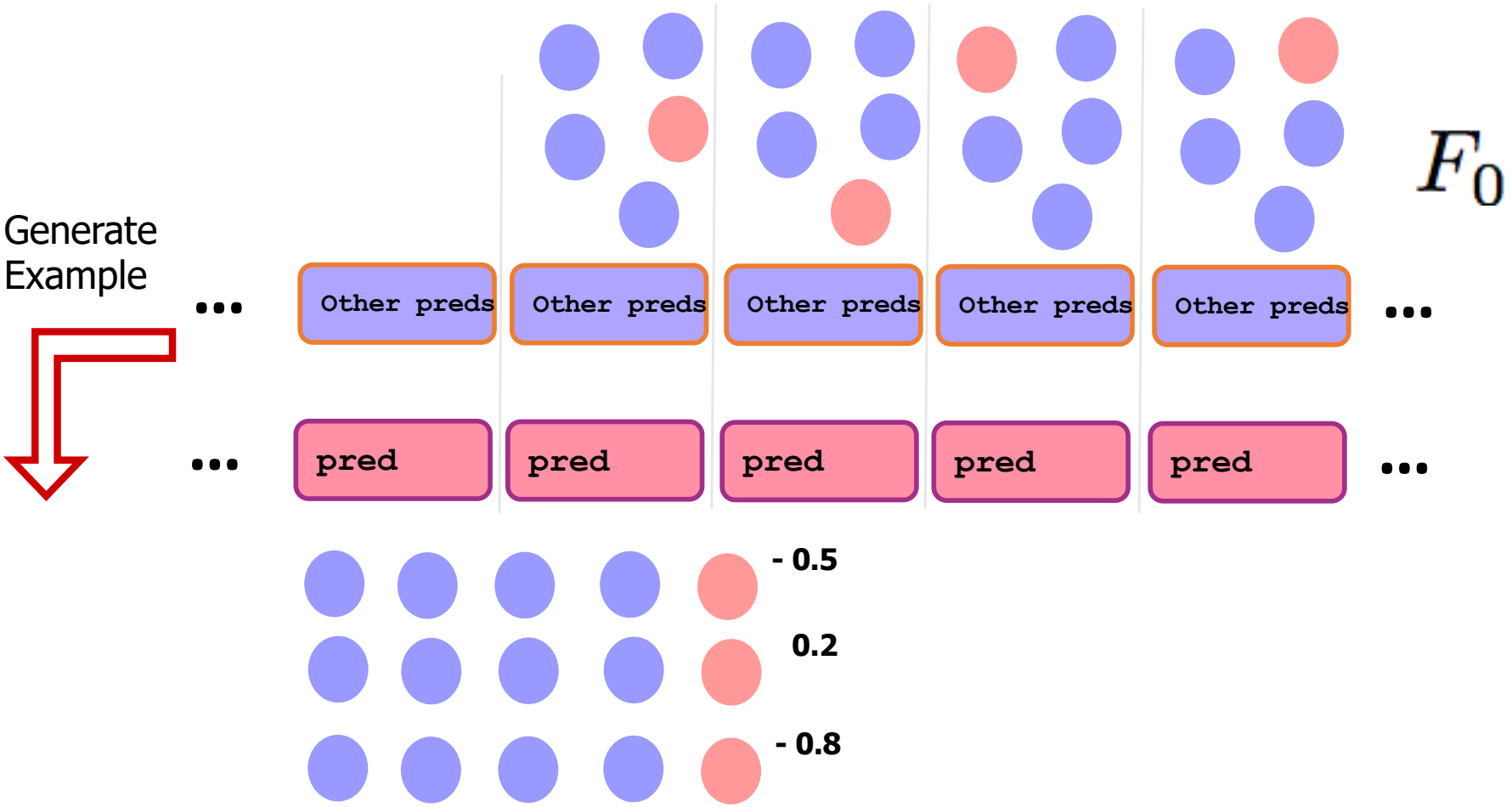


Boosting RDNs/MLNs

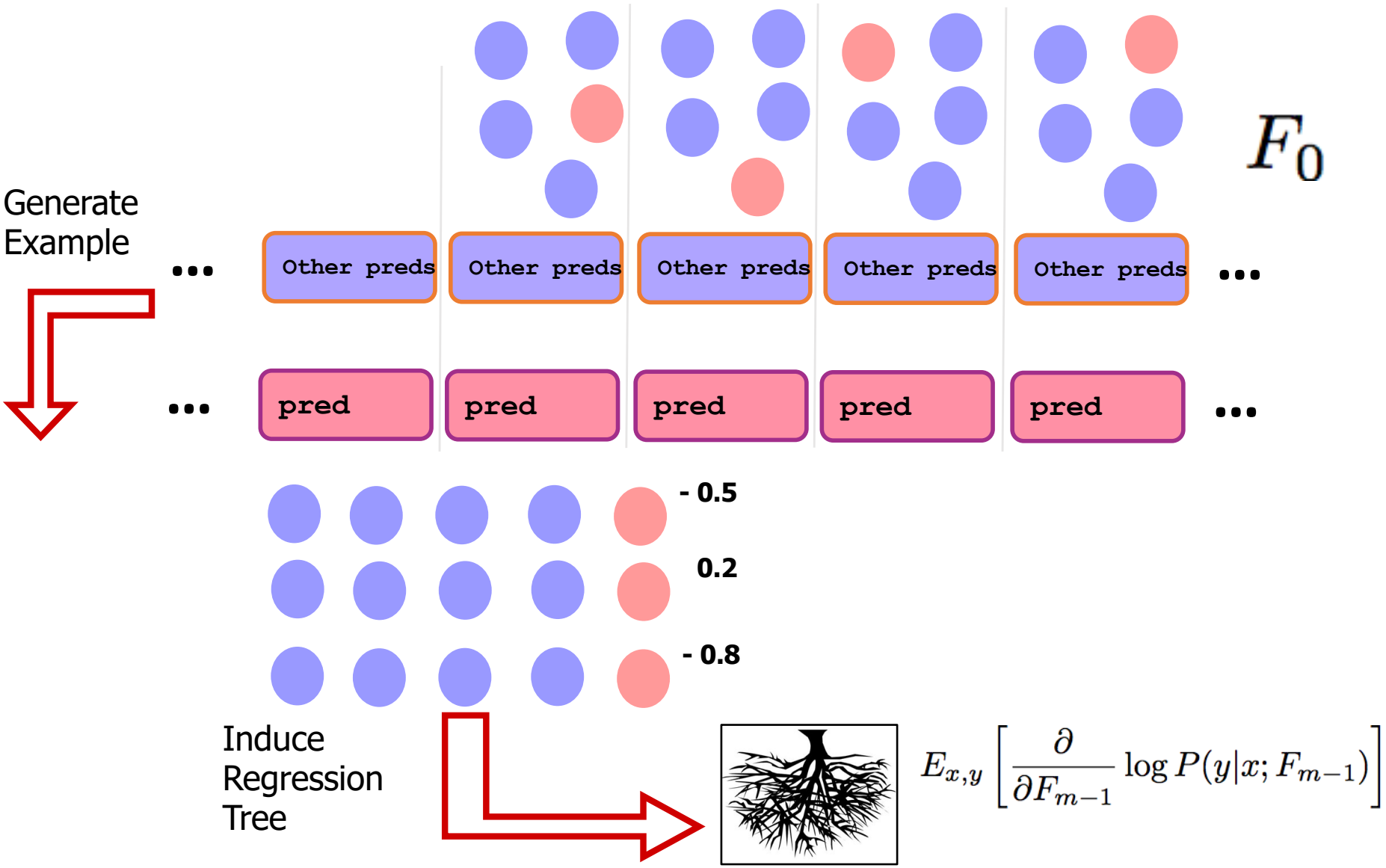


$$\frac{\partial \log P(y_i|x_i)}{\partial \phi(y_i=1|x_i)} = I(y_i=1|x_i) - \frac{e^{\phi(y_i=1|x_i)}}{\sum_{y'} e^{\phi(y'|x_i)}}$$

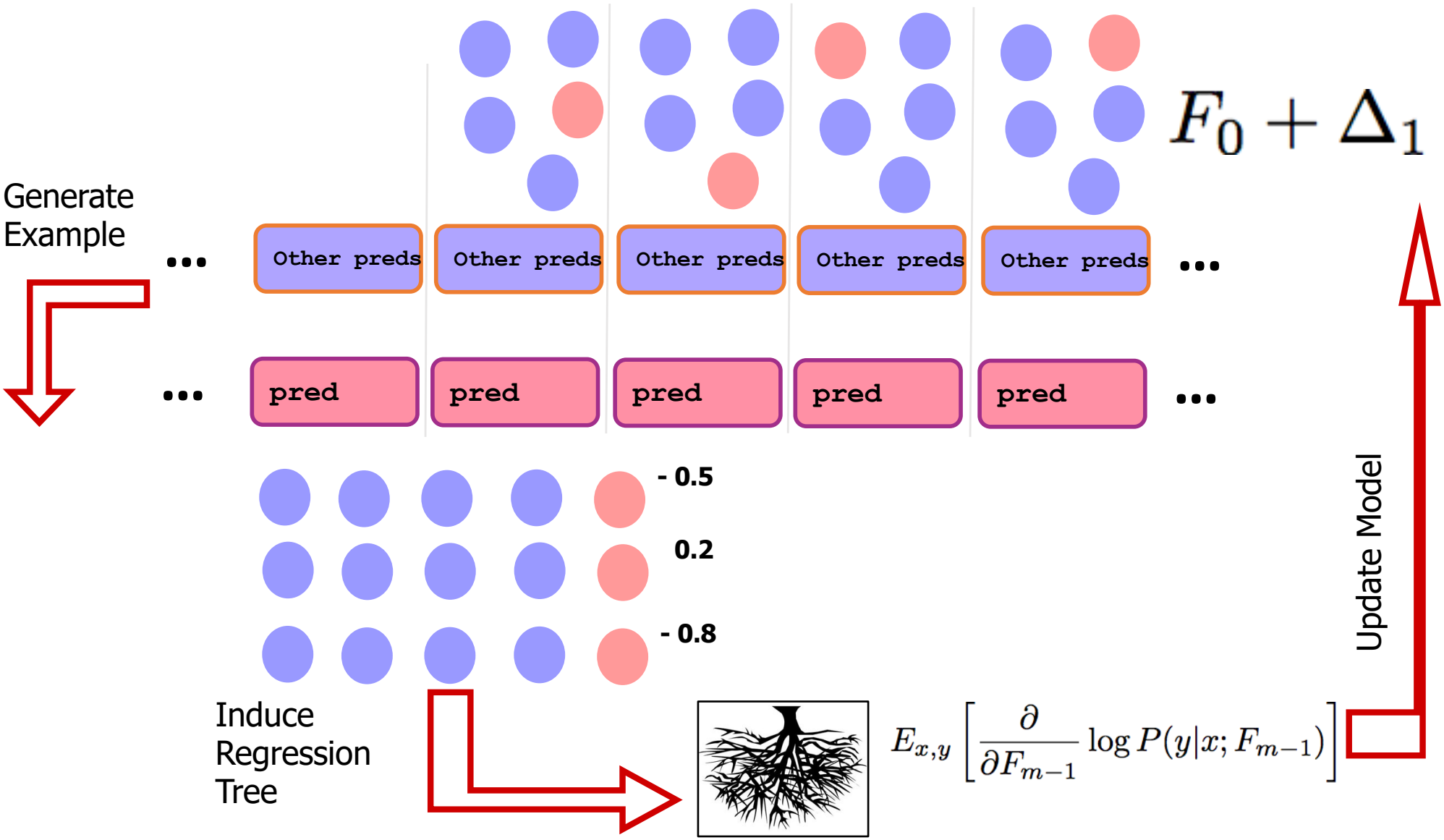
Boosting RDNs/MLNs



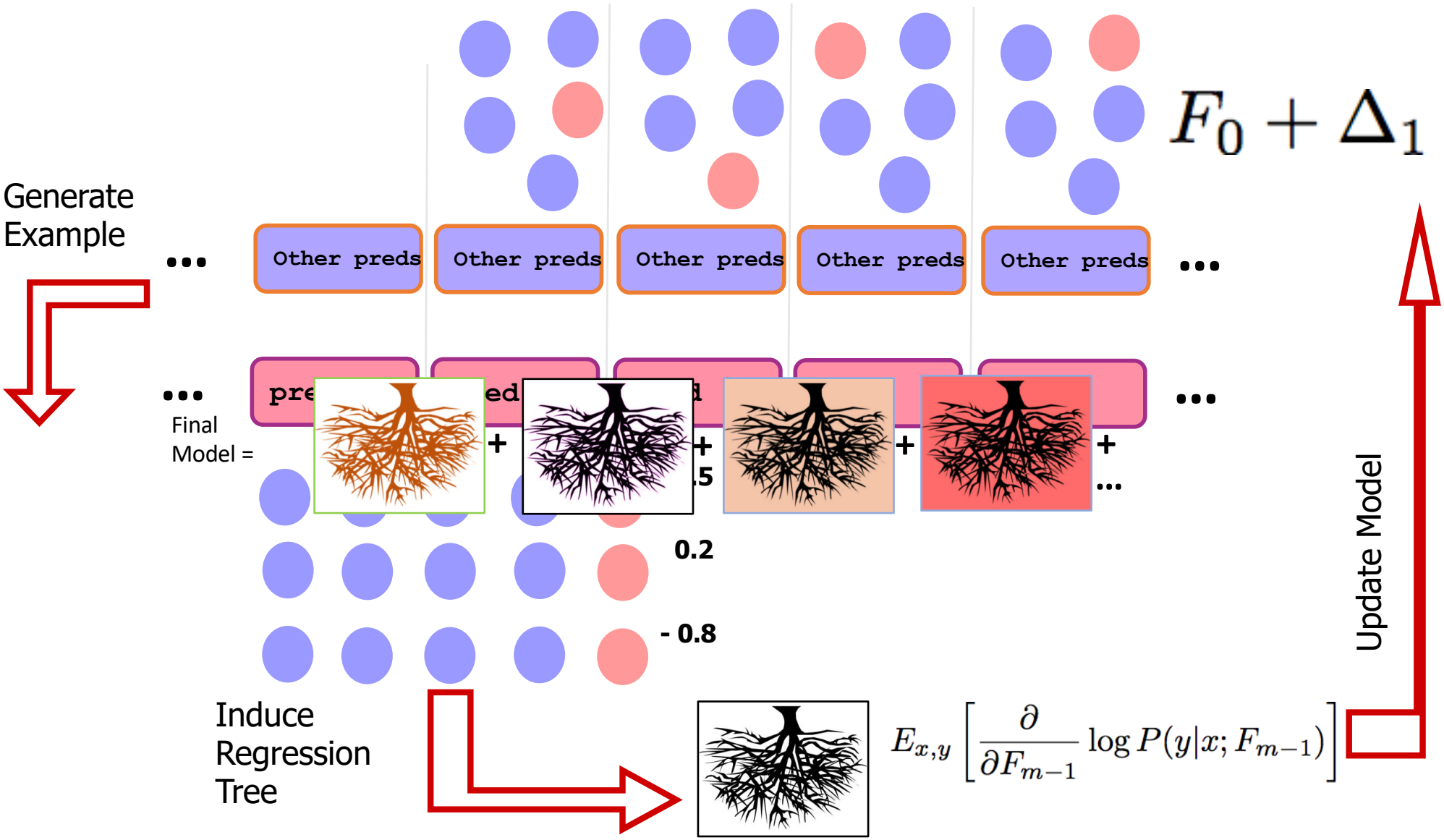
Boosting RDNs/MLNs



Boosting RDNs/MLNs



Boosting RDNs/MLNs

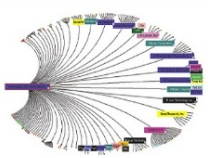


It works

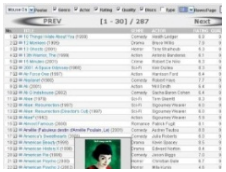
Predicting the advisor for a student

Algo	Likelihood	AUC-ROC	AUC-PR	Time
Boosting	0.810	0.961	0.930	9s
RPT	0.805	0.894	0.863	1s
MLN	0.730	0.535	0.621	93 hrs

Movie Recommendation

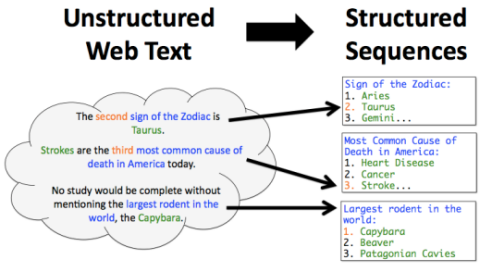


Citation Analysis



Discovering Relations

Learning from Demonstrations



Information Extraction

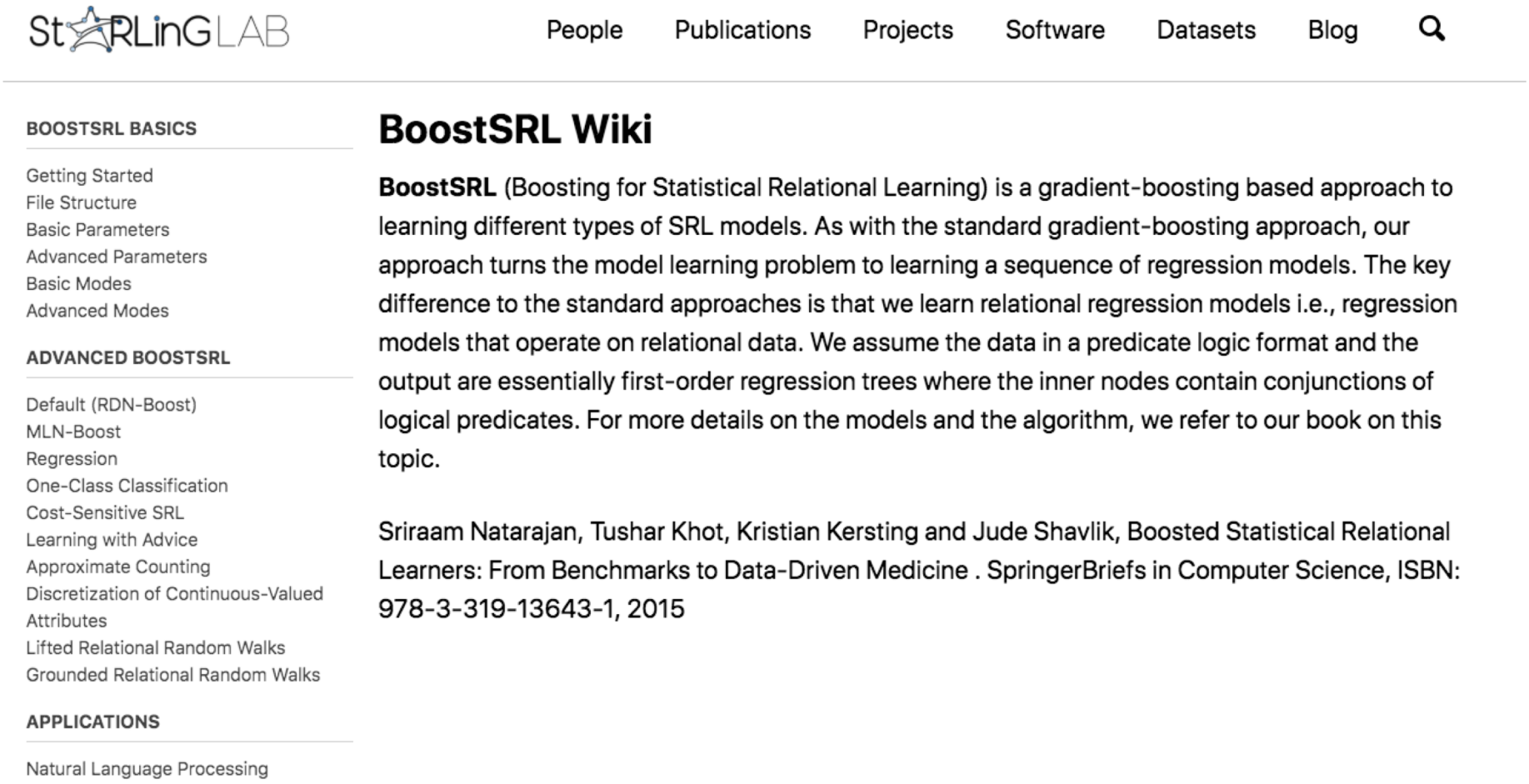
Scale of Learning Structure

- 150 k facts describing the citations
- 115k drug-disease interactions
- 11 M facts on NLP tasks

Natarajan et al. MLJ'12, Khot et al. ICDM '11, Natarajan et al. IJCAI '11, Natarajan et al. IAAI '13
Weiss et al. IAAI '12 AI Magazine '12, Natarajan et al. IJMLC '13, Khot et al. MLJ' 14

Try it out yourself!

https://starling.utdallas.edu/software/boostsrl/wiki/

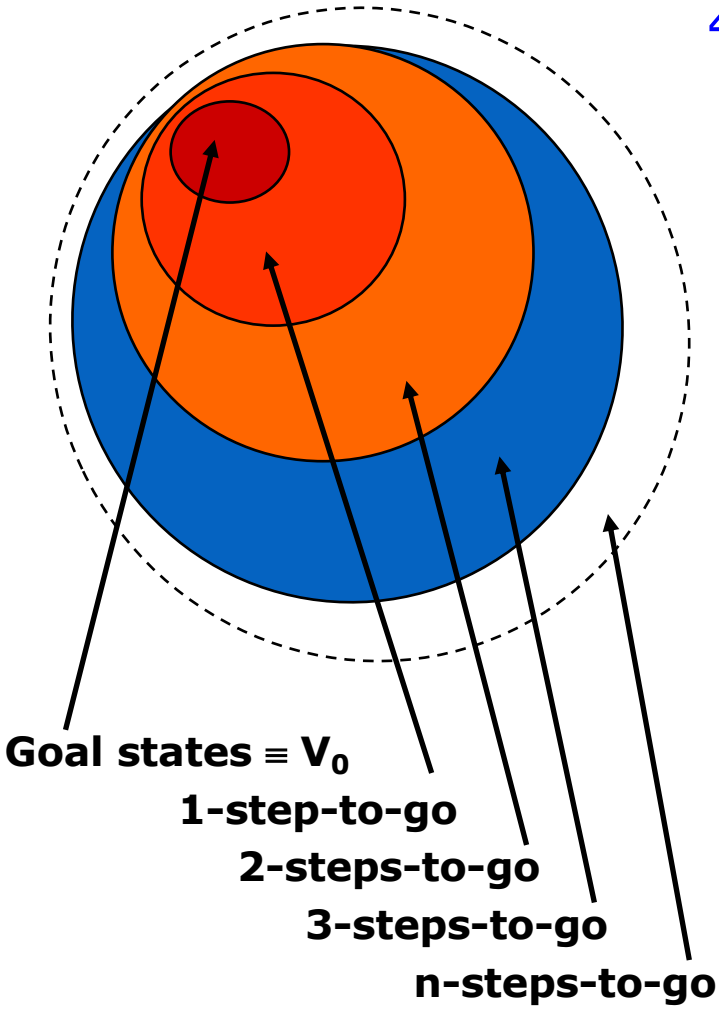


Much more exists: relational random walks (Cohen et al.), relational embeddings (Riedel et al.), relational neural networks (Niepert et al., Zelezny et al., d'Avila Garcez et al.) and so on

Generally, many AI tasks are amenable to relational modelling and lifting. The LP/QP approaches already show this. Let’s consider e.g.

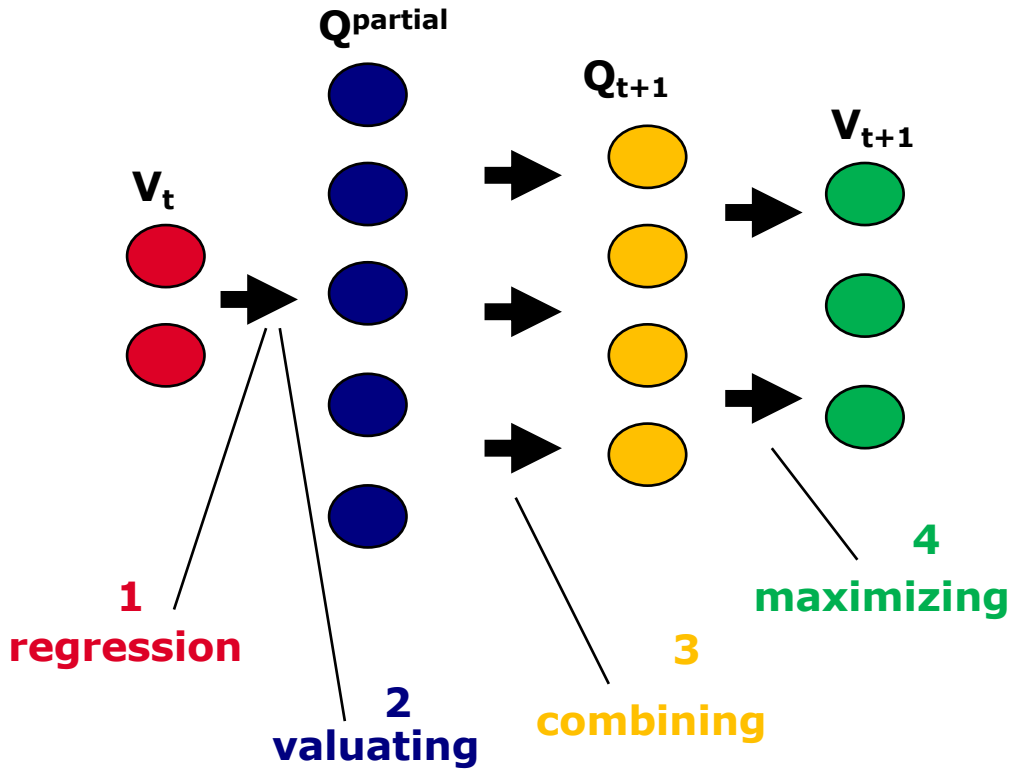
Learning to act optimally

Relational Variant of Dynamic Programming for solving relational Markov Decision Processes (MDPs)



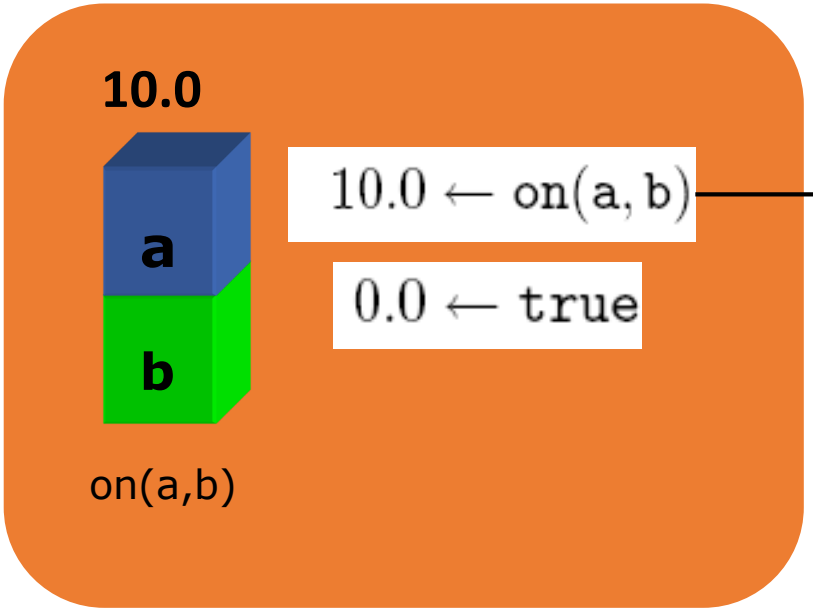
4 steps (indicated by color)

$$V_{t+1}(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_t(s')]$$



Given a relational encoding of a MDP, compute the value function using dynamic programming

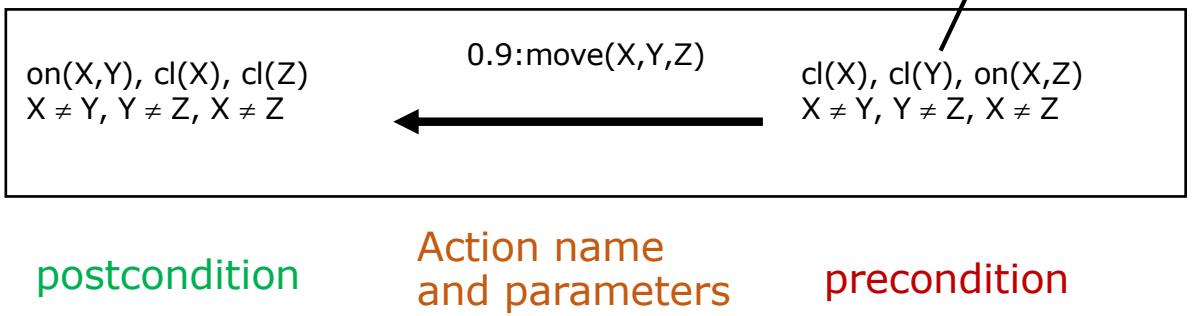
Relational MDPs (using a simplified representation)



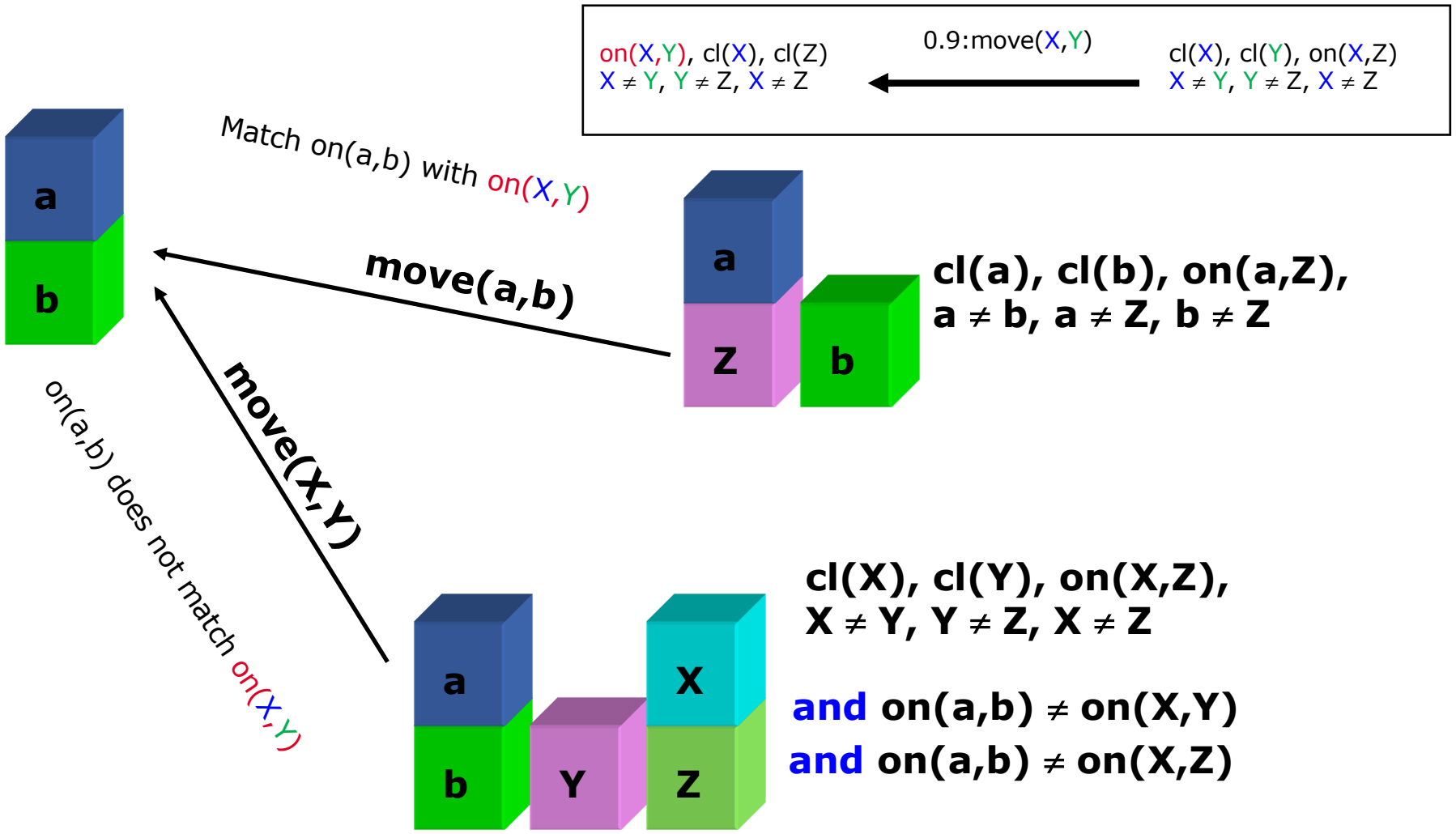
$\mathtt{cl(a), cl(b), on(a,c)}$

- **States: interpretations**, i.e., set of ground atoms
- **Abstract states**: **conjunction** of atoms („query“)
- **Actions**: each outcome a probabilistic STRIP rule
- **Abstract value functions**: set of rules of the form

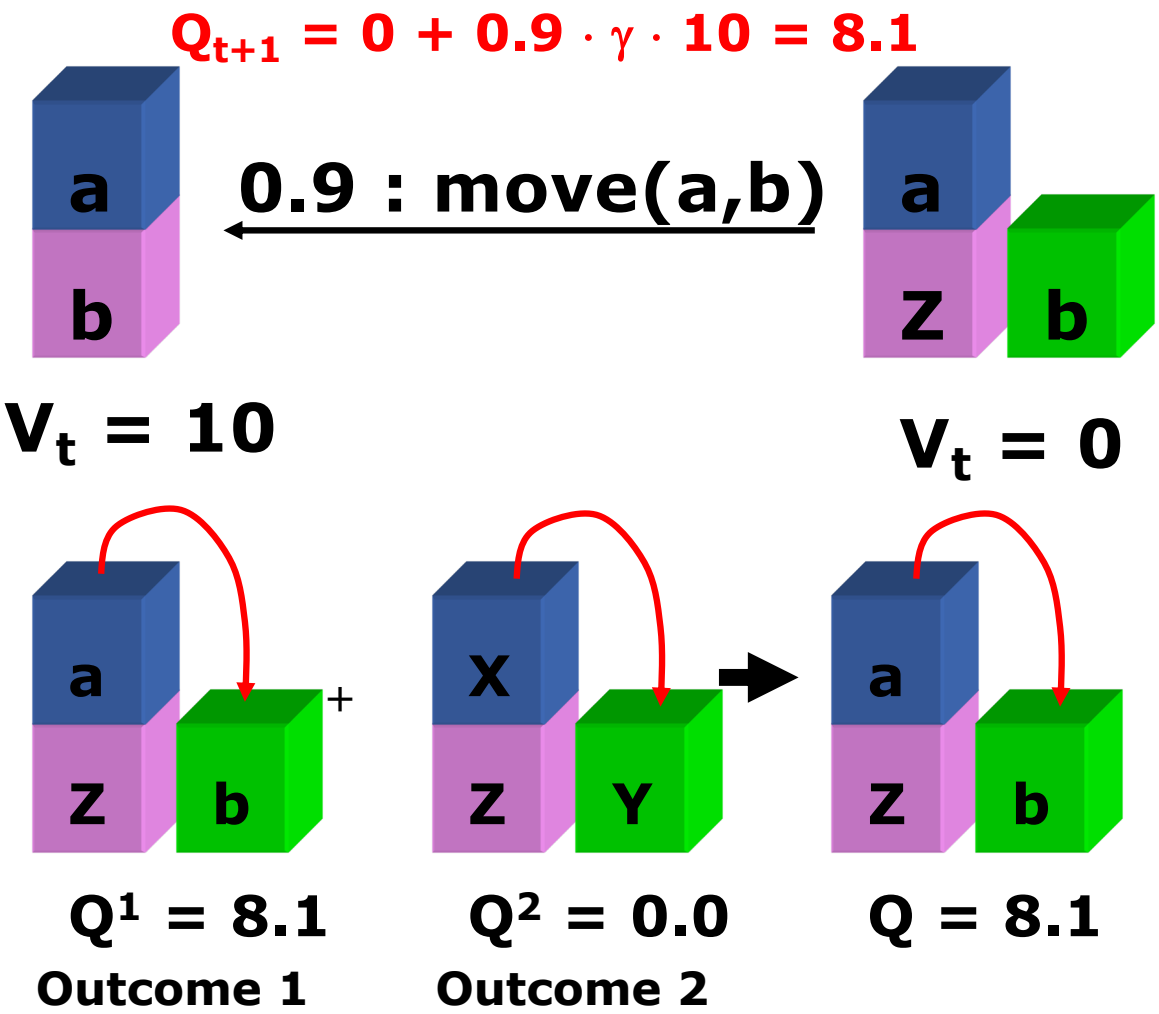
  $c \leftarrow B$ where c is a value and B a conjunction
- **Reward Function**: initial value function $\mathbf{V_0}$



Step 1: Regression

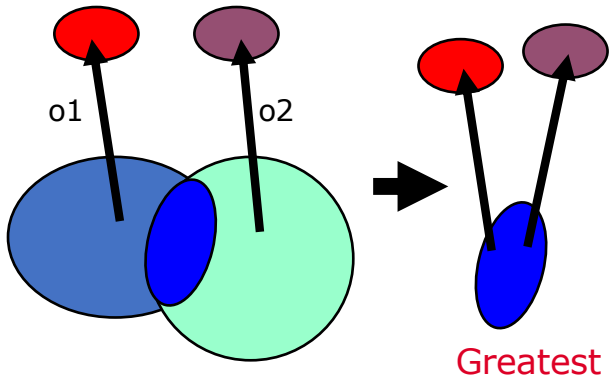


Steps 2&3: Valuation & Combination, i.e., computing Q-rules



Step 2:
Computing values for a single outcome

Step 3:
Combining outcomes



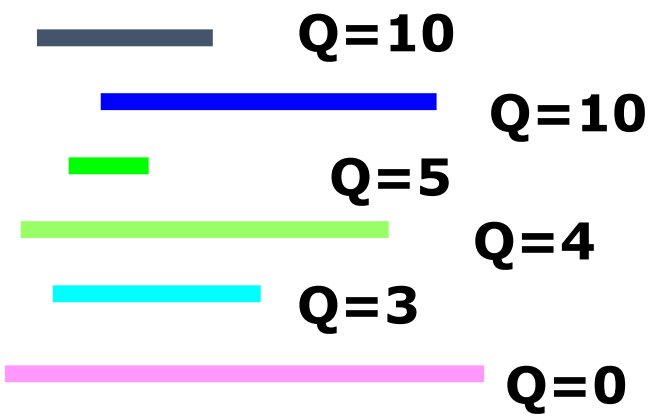
Step 3: Maximizing, i.e., computing the next value function

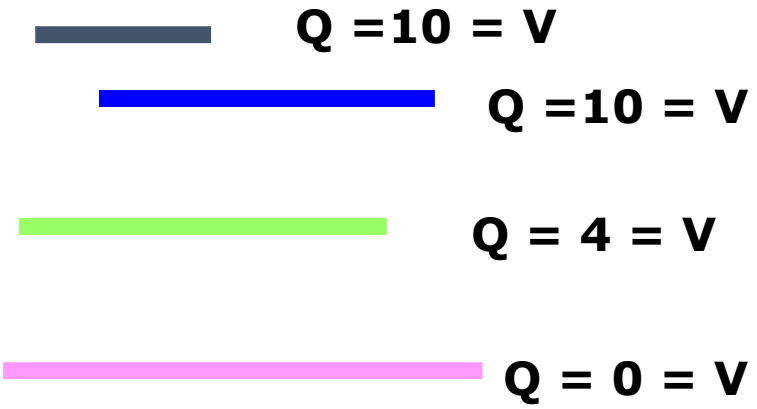
- Maximizing Q-rules, i.e., $V_{t+1}(s) = \max_a Q_{t+1}(s,a)$

10 : move(X,Y) ← cl(X), cl(Y)
5 : move(a,b) ← cl(a), cl(b)

10 : move(X,Y) ← cl(X), cl(Y)

V(cl(X), cl(Y)) = 10





Logistics Domain

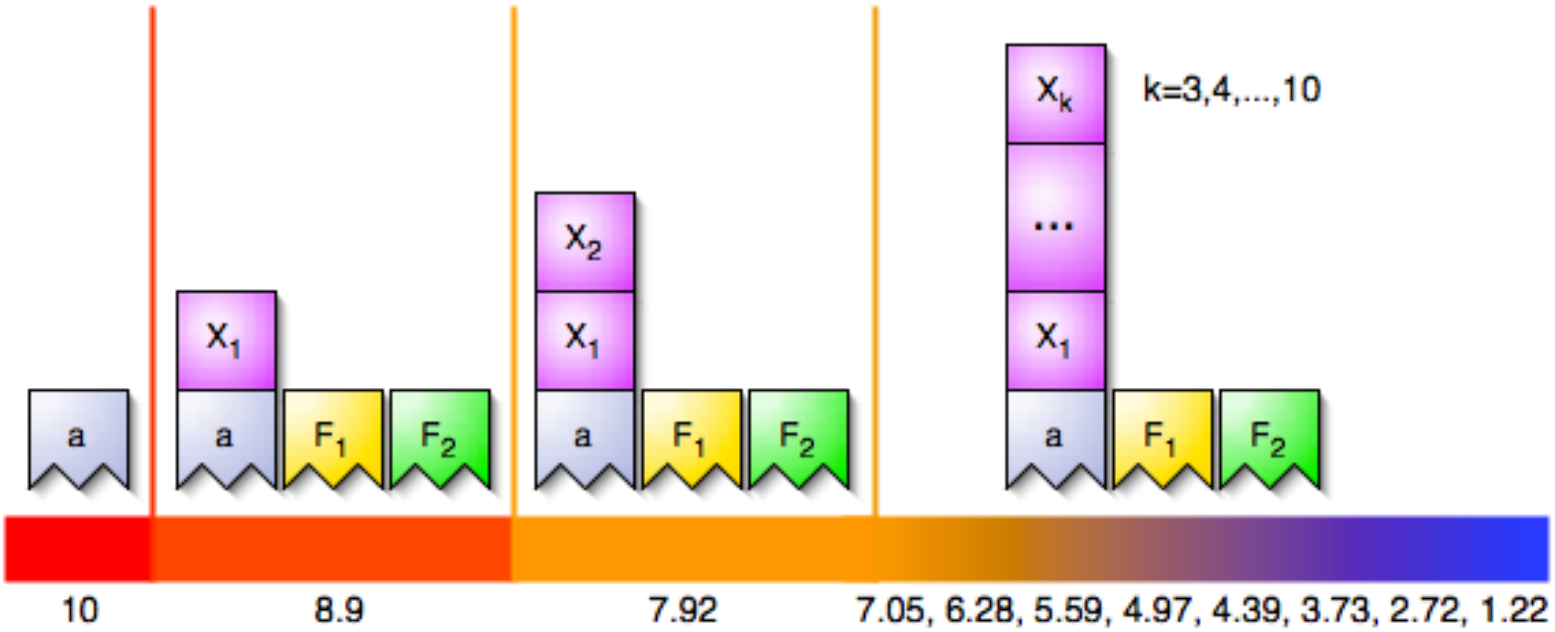
	V_t									
abstract states	1	2	3	4	5	6	7	8	9	10
bin(b, p).	**10.000**	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
tin(A, p), on(b, A), not_rain.	**8.100**	**8.829**	**8.895**	**8.901**	8.901	8.901	8.901	8.901	8.901	8.901
tin(A, p), on(b, A), rain.	**6.300**	**8.001**	**8.460**	**8.584**	**8.618**	**8.627**	**8.629**	**8.630**	8.630	8.630
tin(A, B), on(b, A), not_rain.		**7.290**	**7.946**	**8.005**	**8.010**	**8.011**	8.011	8.011	8.011	8.011
tin(A, B), on(b, A), rain.		**5.670**	**7.201**	**7.614**	**7.726**	**7.756**	**7.764**	**7.766**	**7.767**	7.767
tin(A, B), bin(b, B), not_rain.			**5.905**	**6.968**	**7.111**	**7.128**	**7.130**	**7.131**	7.131	7.131
tin(A, B), bin(b, B), rain.			**3.572**	**5.501**	**6.282**	**6.563**	**6.658**	**6.689**	**6.699**	**6.702**
tin(A, B), bin(b, C), not_rain.				**5.314**	**6.271**	**6.400**	**6.416**	**6.417**	**6.418**	6.418
tin(A, B), bin(b, C), rain.				**3.215**	**4.951**	**5.654**	**5.907**	**5.993**	**6.020**	**6.029**
tin(A, B).	**0.000**	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Convergence on both structural and value level.

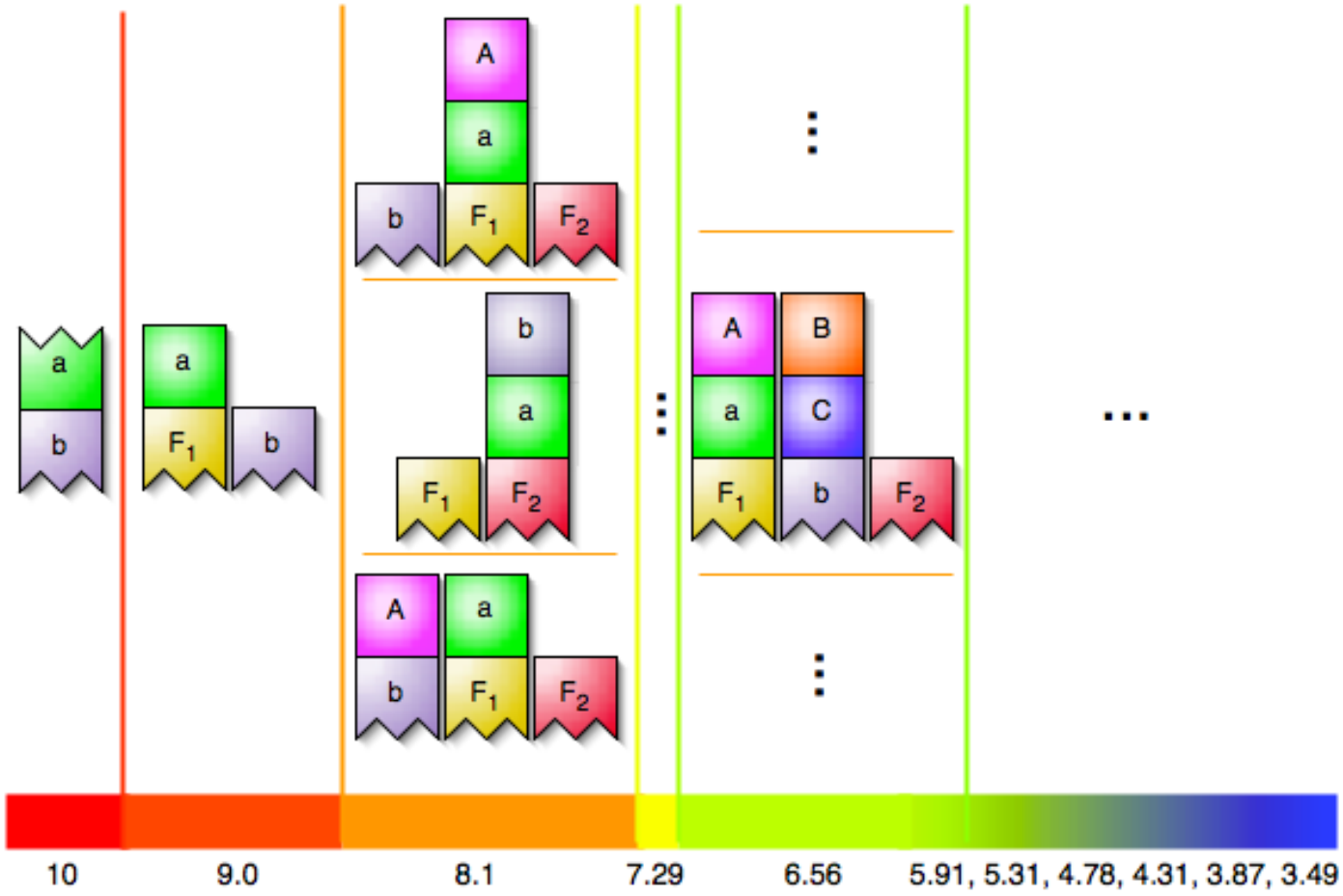
(In about 2 min.)

Resulting value function is independent of the actual domain size. This is lifted inference!

Blocks World: cl(a)



Blocks World: on(a,b)

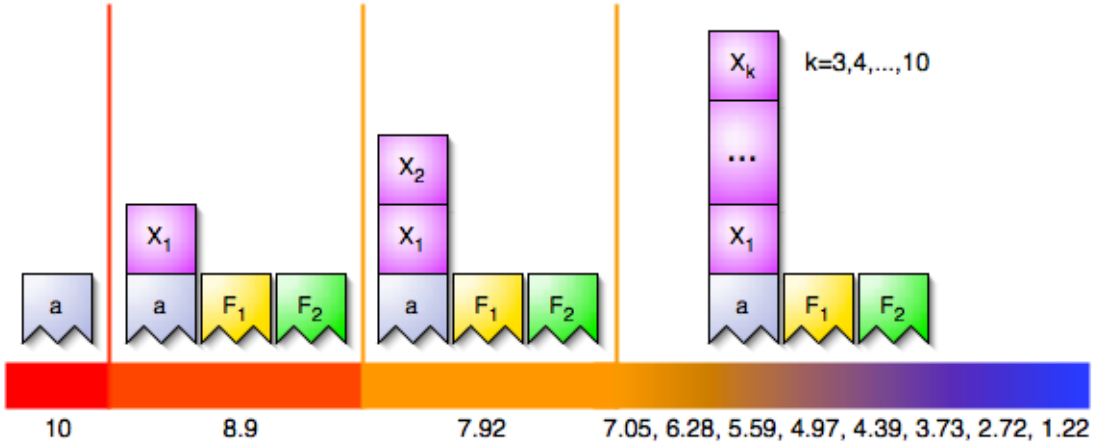


Value functions can often be much more complex to represent than the corresponding policy

Direct Policy Learning

When policies have much simpler representations than the corresponding value functions, **direct search in policy space can be a good idea**

Goal: cl(a)



Policy: put each block on top of a on the floor

Let's apply boosting as used already for learning relational probabilistic models

Non-Parametric Policy Gradients

[Kersting, Driessens ICML08]

- Assume policy to be expressed using an arbitray potential function

$$\pi(s,a,\Psi) = \frac{e^{\Psi(s,a)}}{\sum_b e^{\Psi(s,b)}}$$

- Do functional gradient search w.r.t. world-value

$$\frac{\partial \rho}{\partial \Psi} = \frac{\partial}{\partial \Psi} \sum_{s,a} d^{\pi}(s)\pi(s,a)Q^{\pi}(s,a)$$

$$= \sum_{s,a} d^{\pi}(s) Q^{\pi}(s,a) \frac{\partial \pi(s,a)}{\partial \Psi}$$

sample

compute locally

Local Evaluation

$$Q^{\pi}(s,a)$$

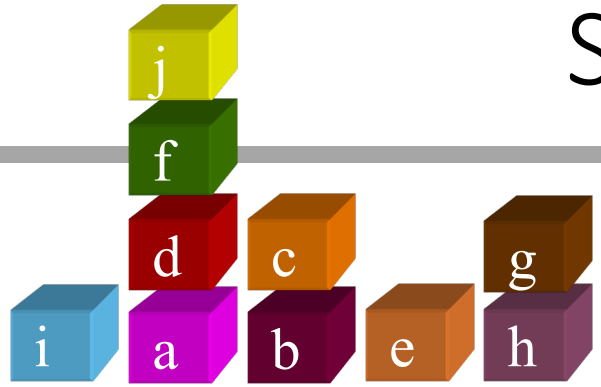
Monte-Carlo estimate or actor critic

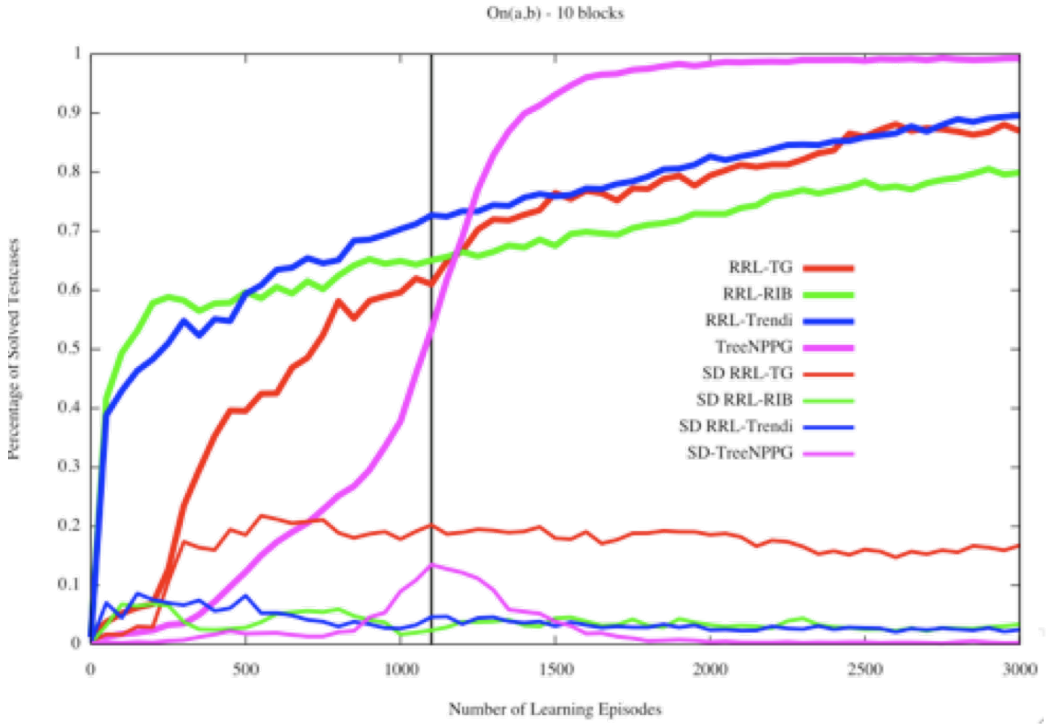
$$\pi(s,a) = \frac{e^{\Psi(s,a)}}{\sum_{b} e^{\Psi(s,b)}}$$

$$\frac{\partial \pi(s,a)}{\partial \Psi(s,a)} = \pi(s,a)(1-\pi(s,a))$$

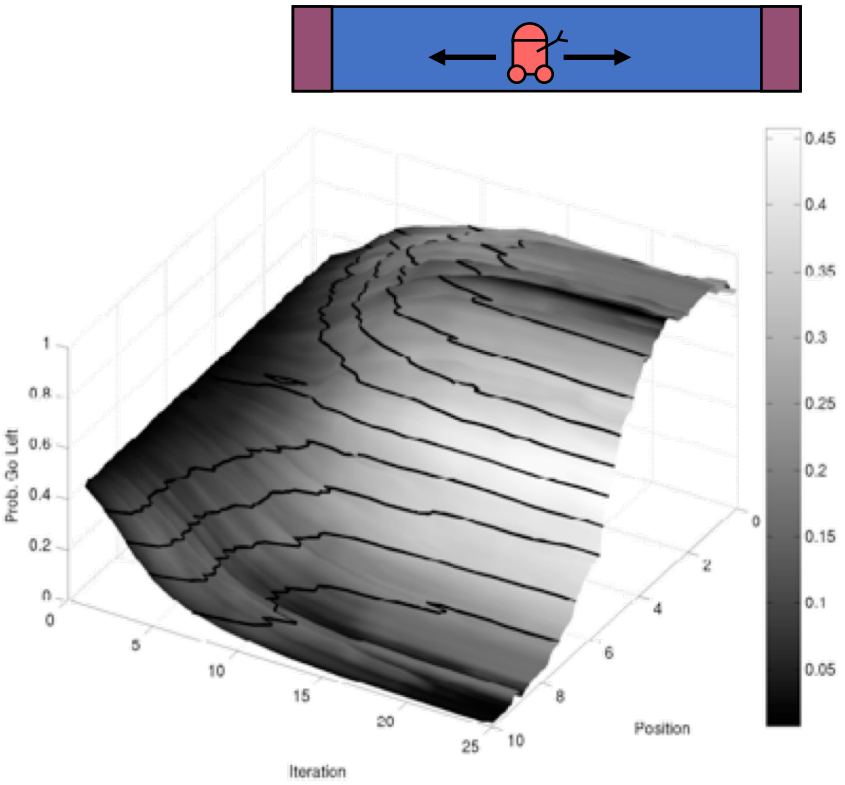
$$\frac{\partial \pi(s,a)}{\partial \Psi(s,b)} = -\pi(s,a)\pi(s,b)$$

Some Experimental Results





Relational Regression Trees



Standard Regression Trees

Allows one to treat propositional, continuous and relational features in a unified way!

What have we learnt?

- Early learning methods extended standard graphical model learning
- Parameter tying exploited when learning relational models
- Vanilla relational learning approach does a greedy search by adding/deleting literals/clauses using some (probabilistic) scoring function
- Learning many weak rules of how to change a model can be much faster
- Many if not most AI algorithms can be lifted to the relational level