Learning the Parameters of Probabilistic Logic Programs from Interpretations

DTAI
Declaratieve Talen en Artificiële Intelligentie

Bernd Gutmann

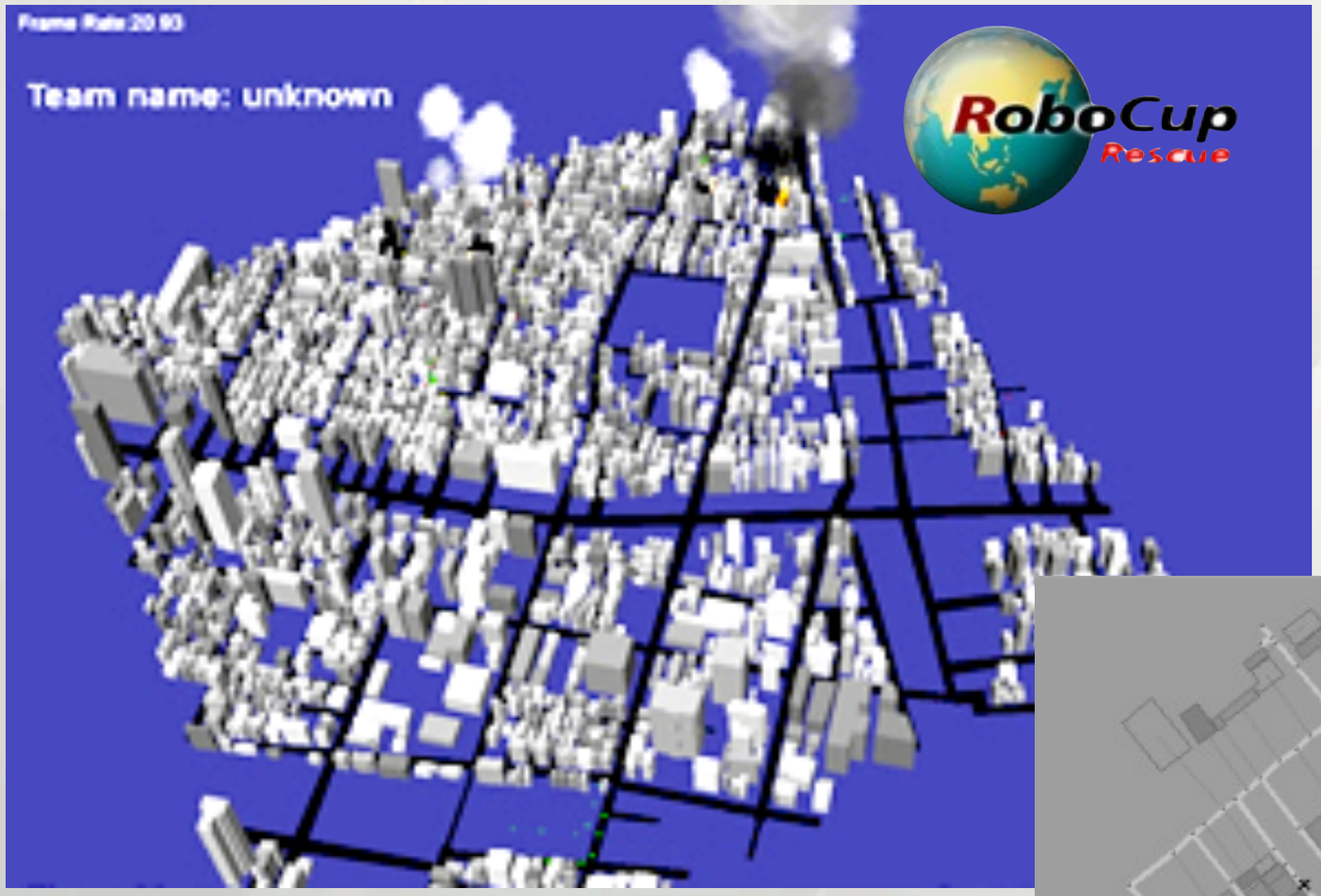
Ingo Thon

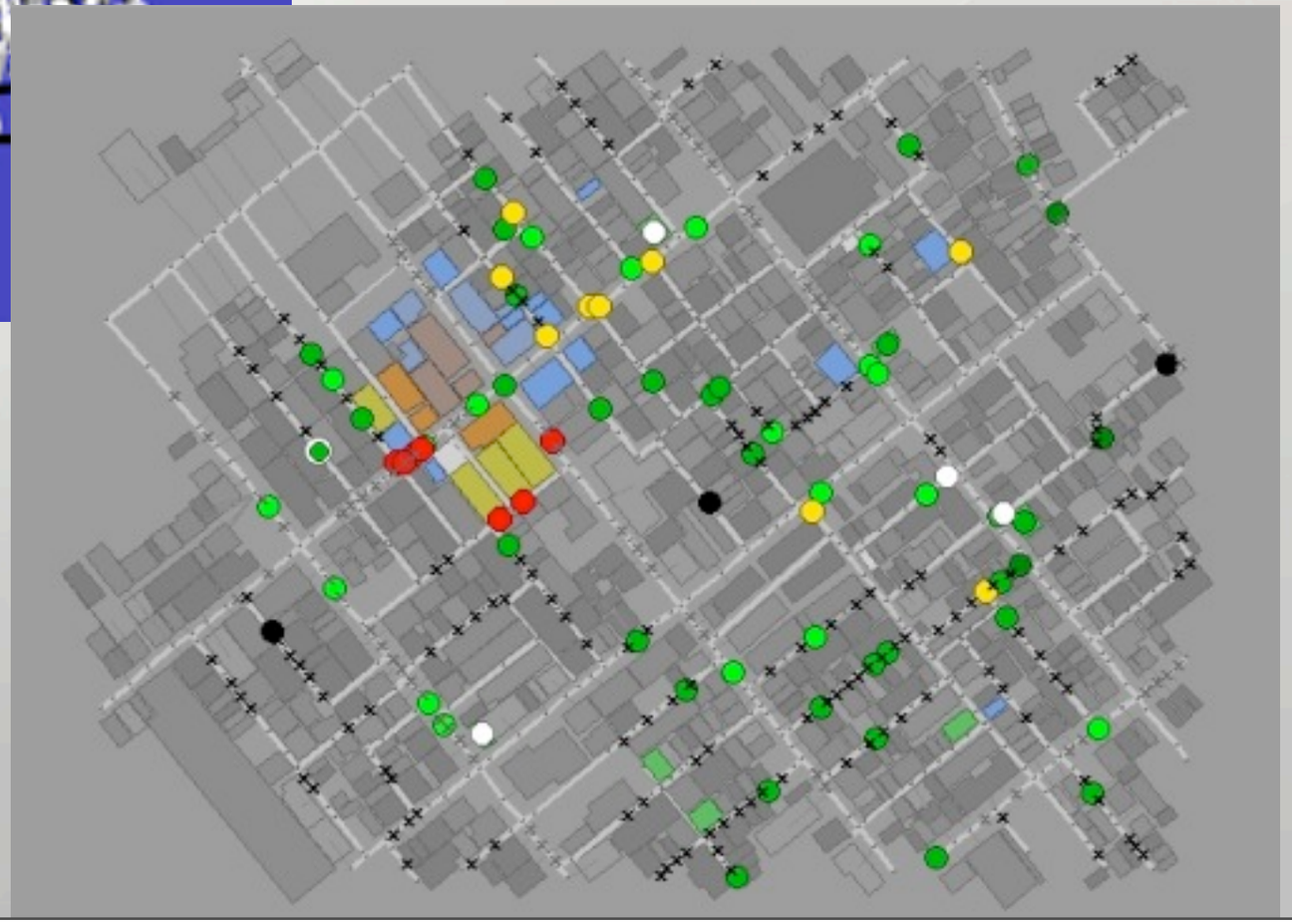
Luc De Raedt

Outline

- Motivation
- Background
- Contribution: LFI-Problog
- Experimental Results
- Conclusions

Consider you are ambulance dispatcher...





Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

$\models$

distribution over programs/return values:

distribution over queries

L. De Raedt et al., IJCAI 2007

Probabilistic Programming

Programming Language:

Prolog

+

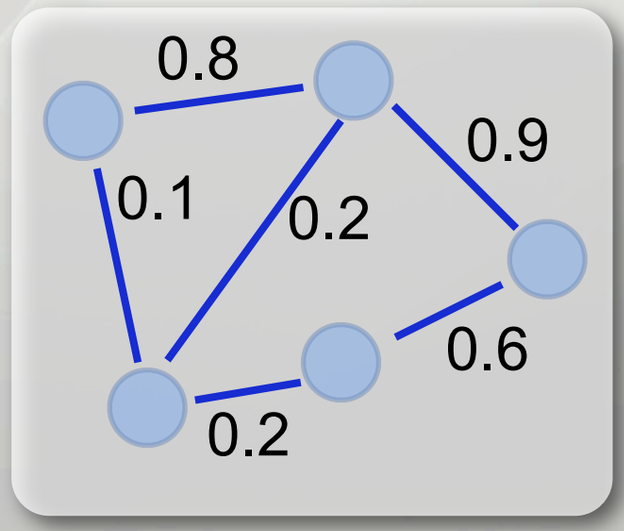
Random Variables:

probabilistic facts

$\models$

distribution over programs/return values:

distribution over queries



L. De Raedt et al., IJCAI 2007

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

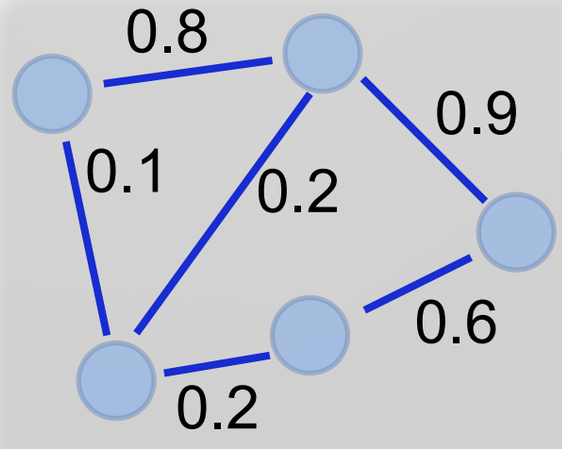
$\models$

distribution over programs/return values:

distribution over queries

path finding algorithm

+



L. De Raedt et al., IJCAI 2007

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

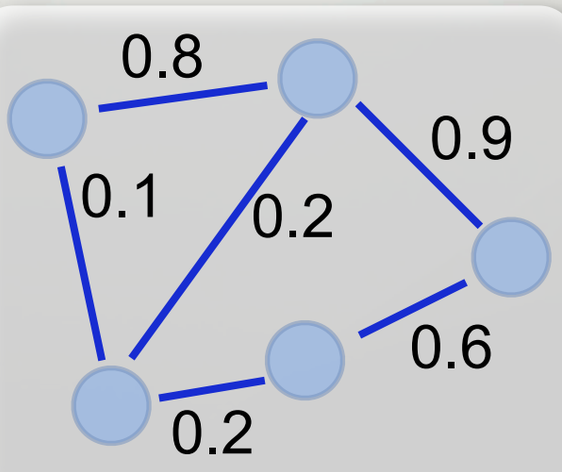
⊨

distribution over programs/return values:

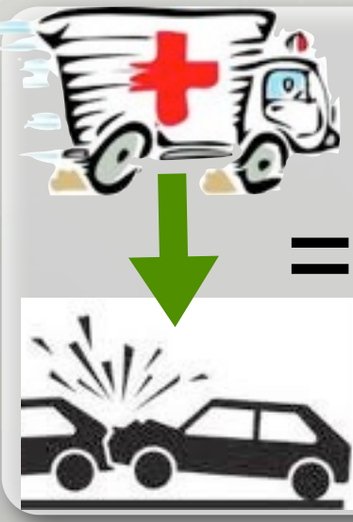
distribution over queries

path finding algorithm

+



⊨



= 0.8

L. De Raedt et al., IJCAI 2007

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

⊨

distribution over programs/return values:

distribution over queries

rule learner

+

soft evidence

0.2

⊨

probabilistic rule learner

L. De Raedt et al., IJCAI 2007

L. De Raedt et al., ILP 2009

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

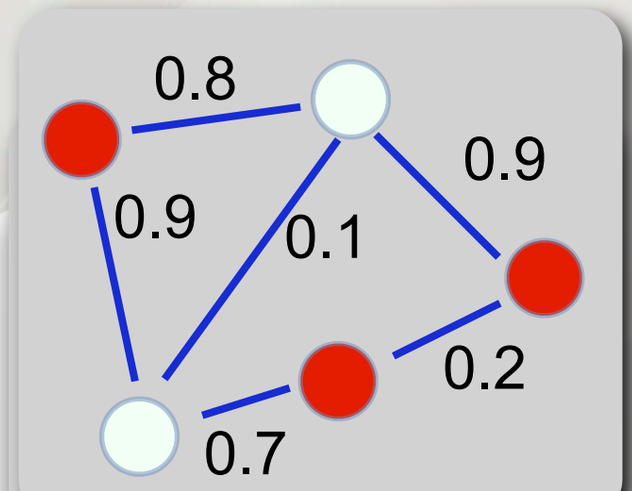
⊨

distribution over programs/return values:

distribution over queries

graph miner

+



⊨

graph patterns

L. De Raedt et al., IJCAI 2007

L. De Raedt et al., ILP 2009

A. Kimmig et al., IJCAI 2009

Probabilistic Programming

Programming Language:

Prolog

lisp interpreter

+

Random Variables:

probabilistic facts

f() =
[0.4 :: true,
0.6 :: false]

0.7

0.2

+

⊨

distribution over programs/return values:

distribution over queries

probabilistic lisp

L. De Raedt et al., IJCAI 2007
L. De Raedt et al., ILP 2009

A. Kimmig et al., IJCAI 2009
to be published

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

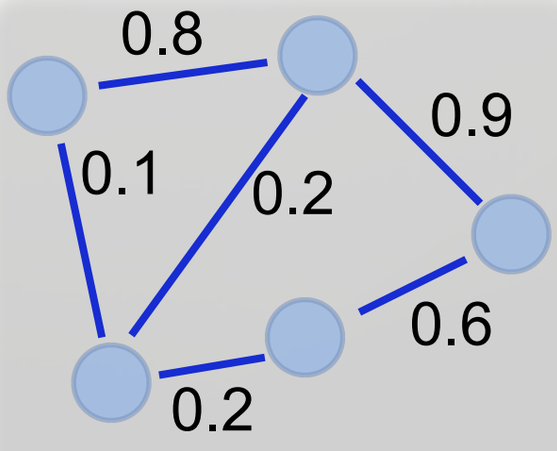
⊨

distribution over return values:

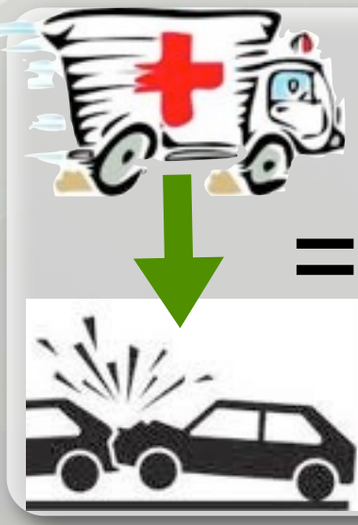
distribution over queries

path finding algorithm

+



⊨



= 0.8

Probabilistic Programming

Programming Language:

Prolog

+

Random Variables:

probabilistic facts

LEARN

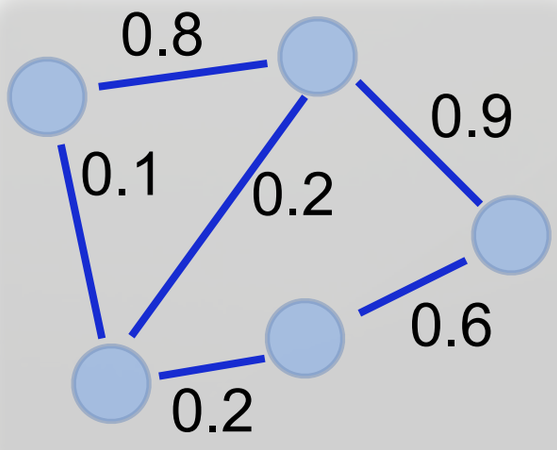
⊨

distribution over return values:

distribution over queries

path finding algorithm

+



⊨

= 0.8

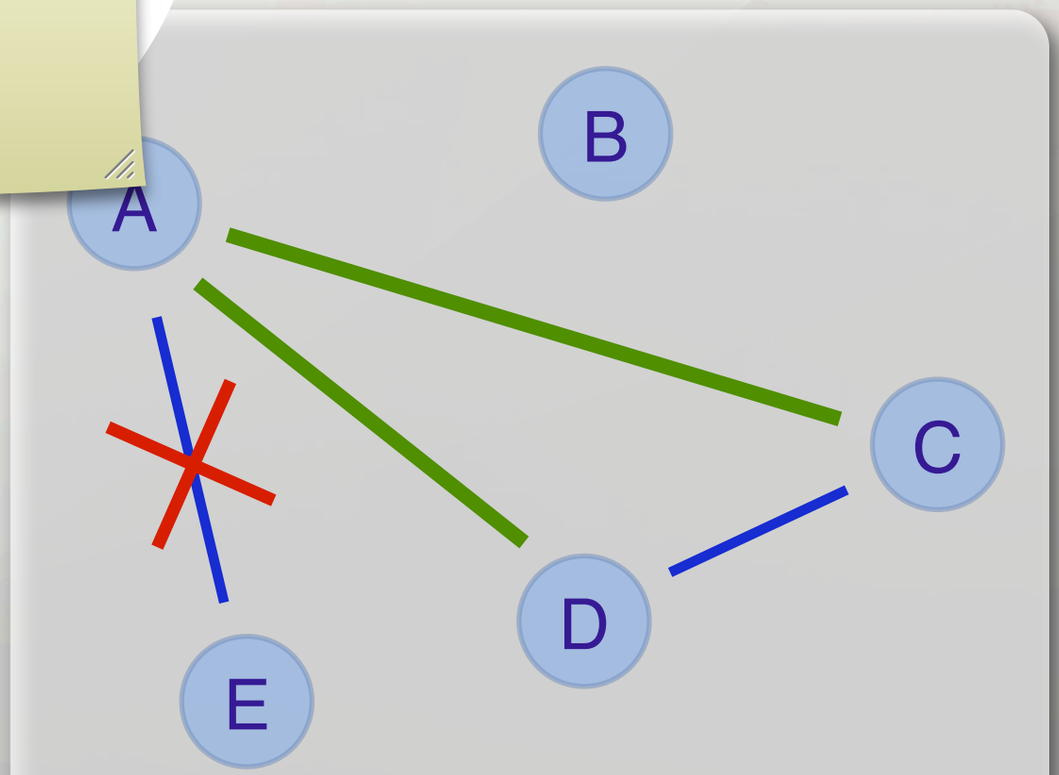
B. Gutmann et al. : ECML 2008

B. Gutmann et al. : ECML 2011!!!!

Training Example

Couldn't pass street from A to E managed to come road from D to C as well....

Work on smoth transition. Non-probabilistic case

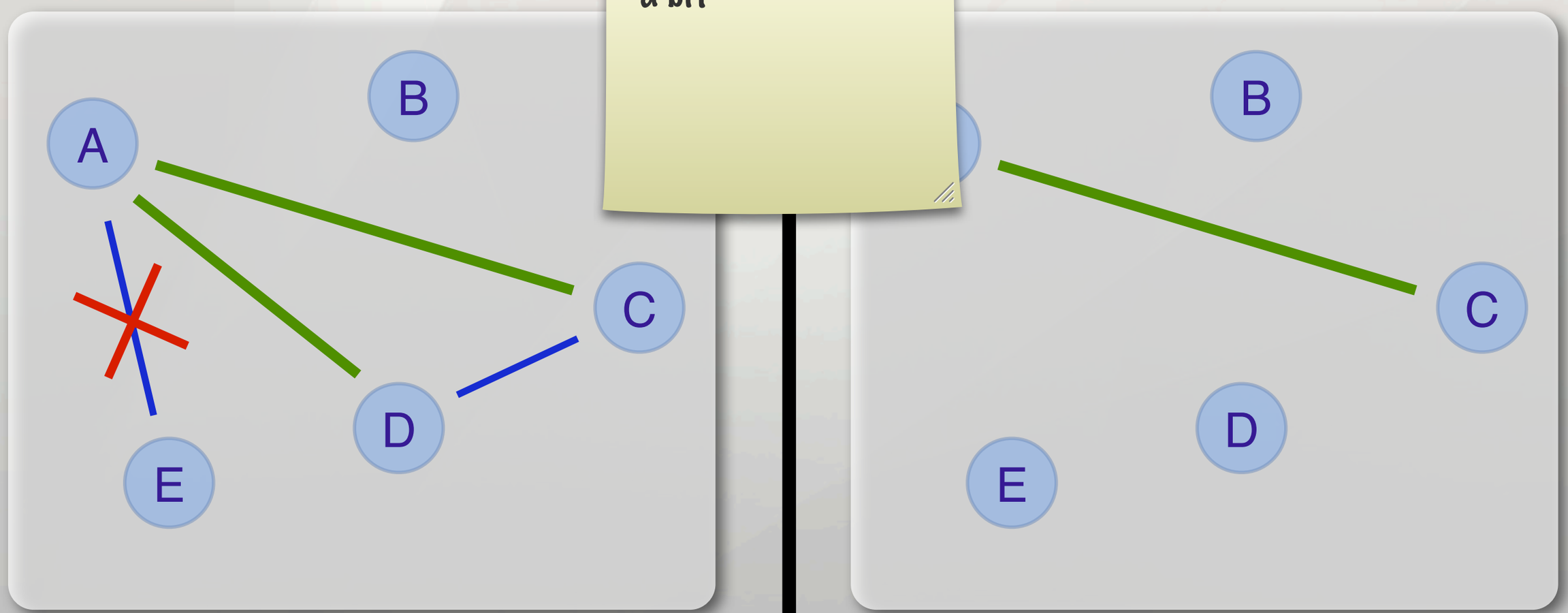


The ILP View on Learning

Learning from interpretations

Learning from entailment

Maybe elobarate a bit



B. Gutmann et al. : ECML 2011

B. Gutmann et al. : ECML 2008

The ILP View on the Alphabet Soup

Learning from interpretations

Markov Logic

BLP

IBAL

PRM

KBMC

Learning from entailment

(Le)ProbLog

SLP

Prism

SDS

SDS: Sato's Distributions Semantics

KBMC: Knowledge Based Model Construction

The ILP View on the Alphabet Soup

Learning from interpretations

Learning from entailment

LFI-ProbLog

(Le)ProbLog

SLP

Prism

SDS

Markov Logic

BLP

IBAL

PRM

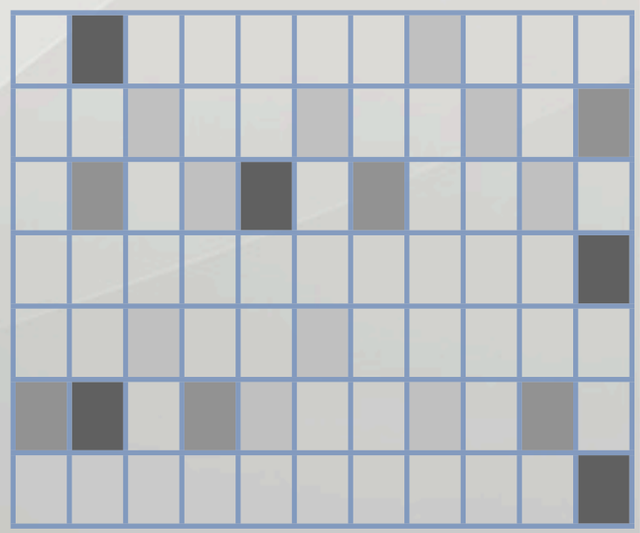
KBMC

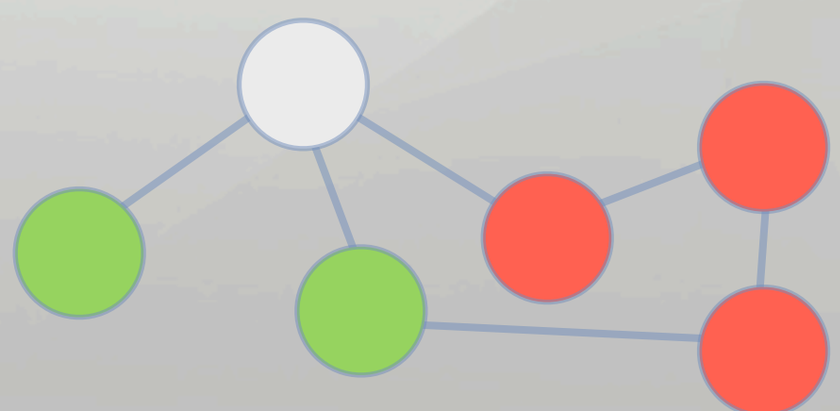
SDS: Sato's Distributions Semantics

KBMC: Knowledge Based Model Construction

Contribution of this Work

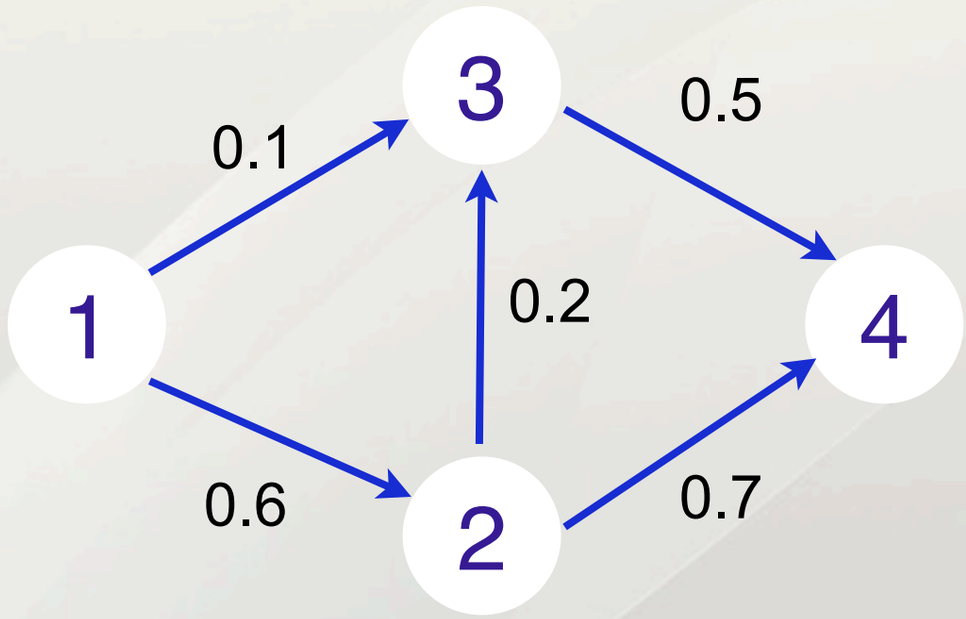
- LFI-ProbLog: First parameter learning approach from partial interpretations for probabilistic logic programming
- Generative setting
- Use (partial) interpretations as training data
- Setting applicable to wide variety of tasks
  - Collective classification
  - Clustering
  - Rule learning
  - Etc.





Toy Example

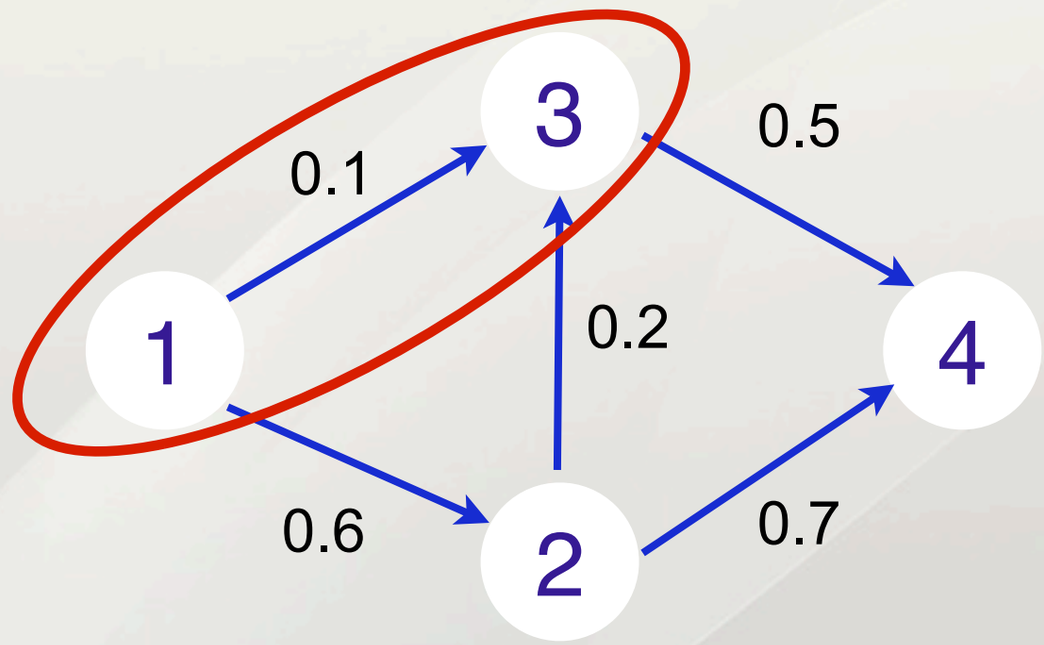
0.1 :: edge(1,3)
0.6 :: edge(1,2)
0.2 :: edge(2,3)
0.5 :: edge(3,4)
0.7 :: edge(2,4)



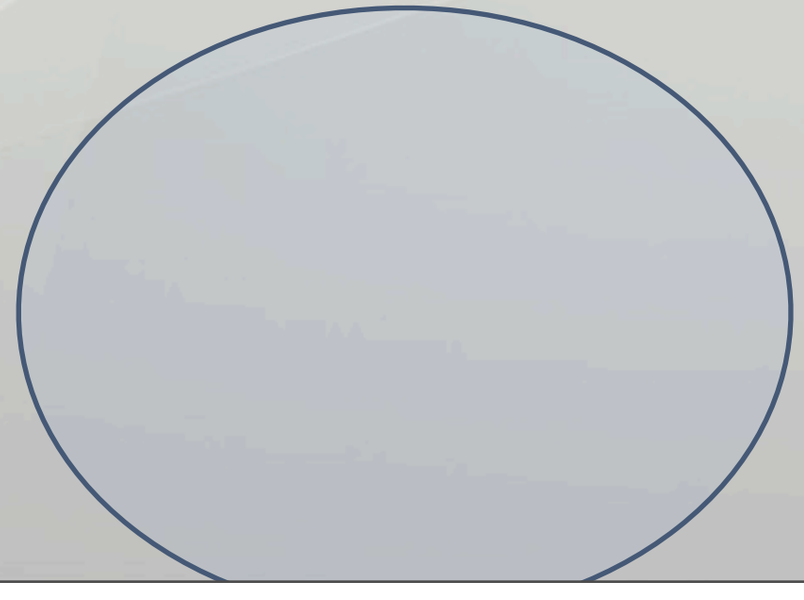

```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```

Toy Example

0.1 :: edge(1,3)
0.6 :: edge(1,2)
0.2 :: edge(2,3)
0.5 :: edge(3,4)
0.7 :: edge(2,4)




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

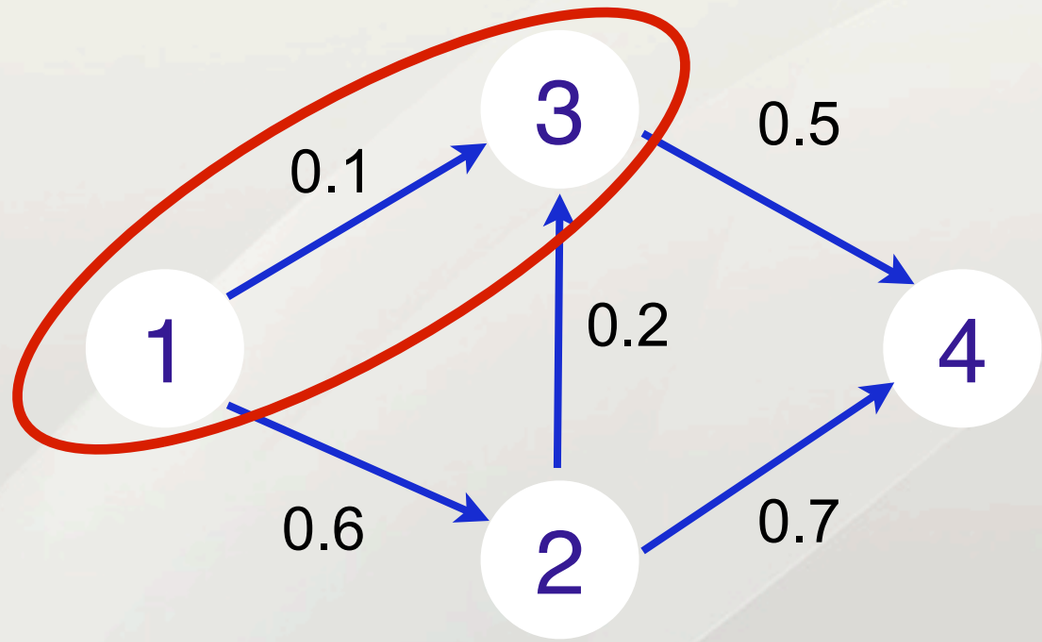
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2)

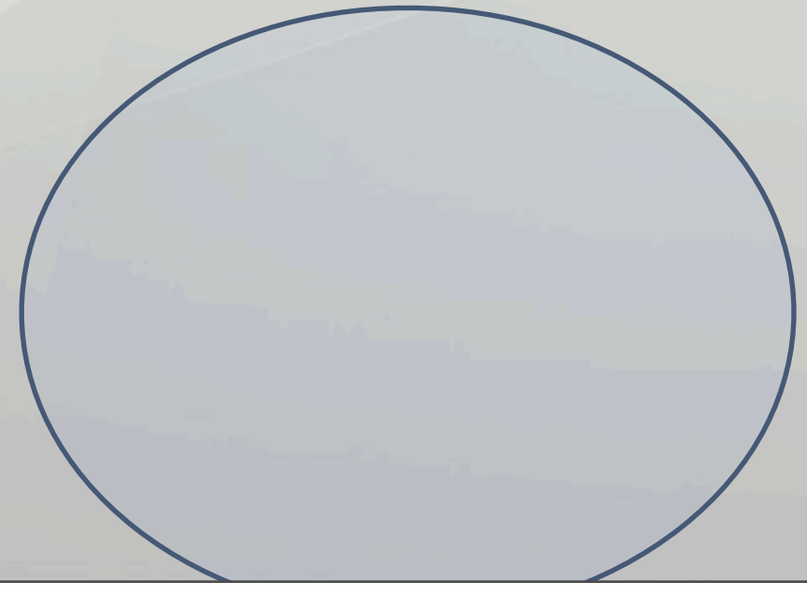
0.2 :: edge(2,3)

0.5 :: edge(3,4)

0.7 :: edge(2,4)




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

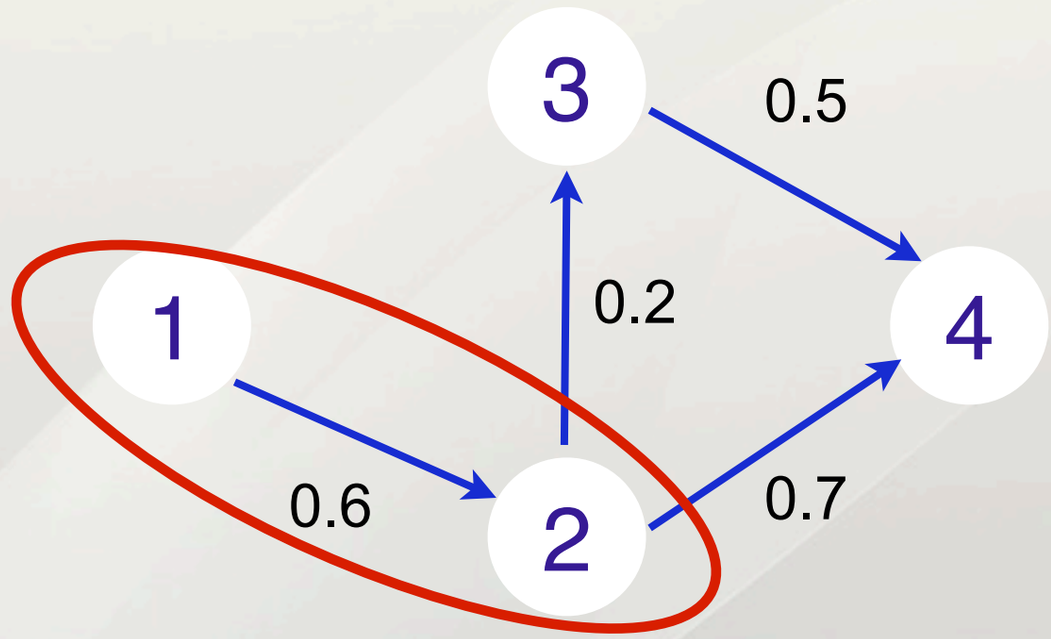
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2)

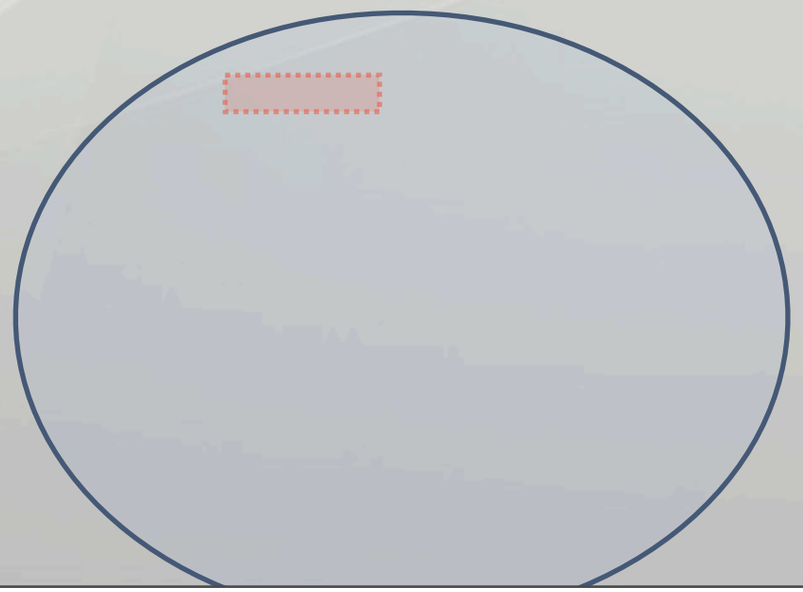
0.2 :: edge(2,3)

0.5 :: edge(3,4)

0.7 :: edge(2,4)




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

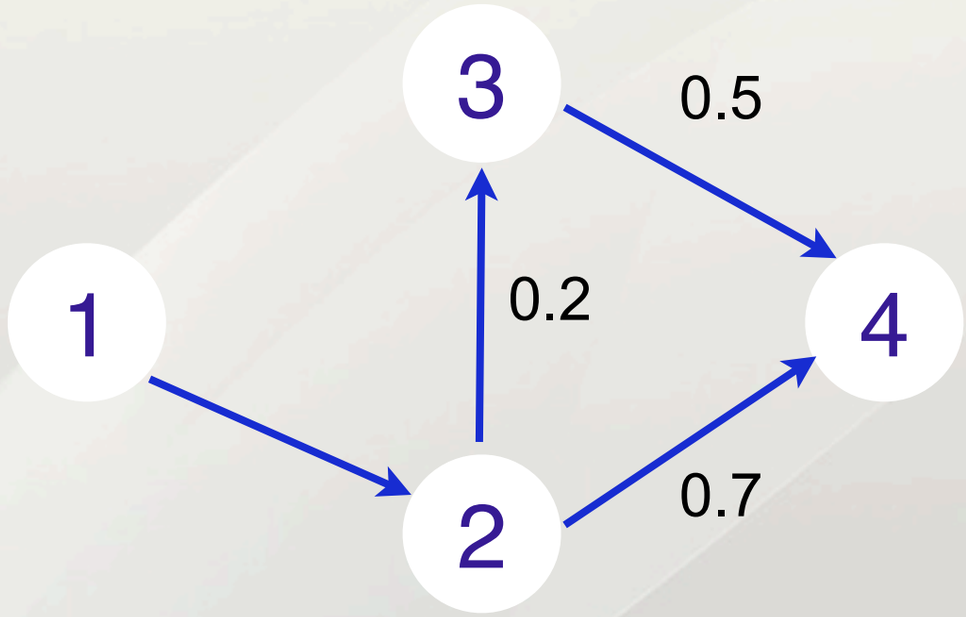
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2) ✔

0.2 :: edge(2,3)

0.5 :: edge(3,4)

0.7 :: edge(2,4)




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```

Toy Example

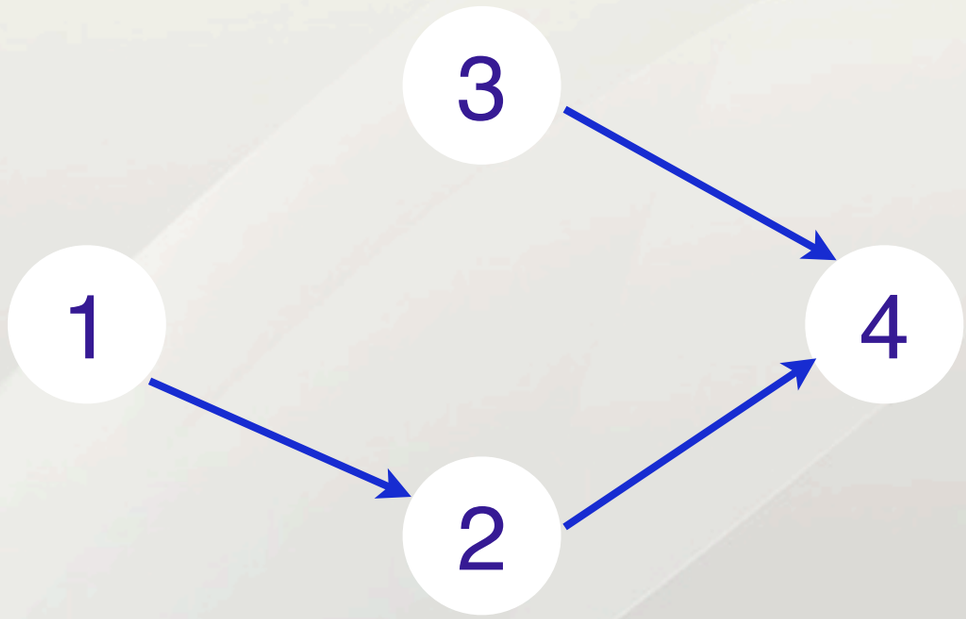
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2) ✔

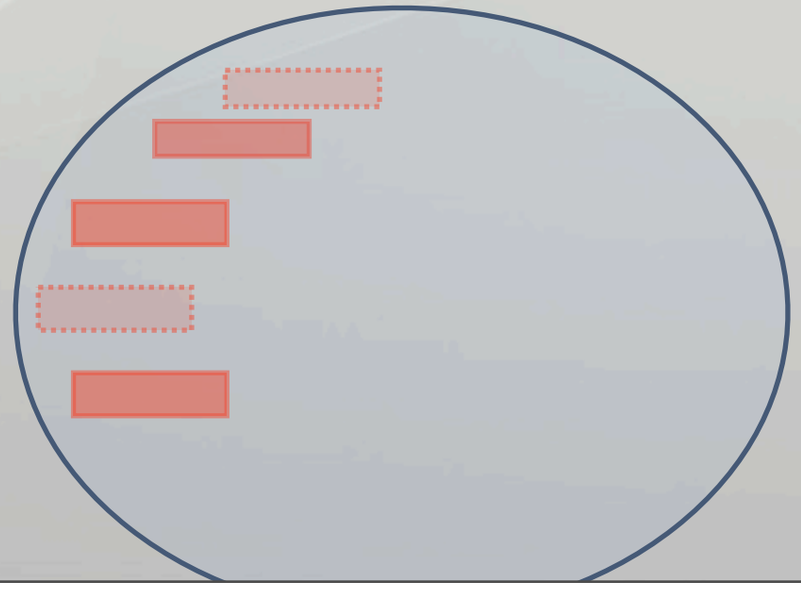
0.2 :: edge(2,3) ✗

0.5 :: edge(3,4) ✔

0.7 :: edge(2,4) ✔




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

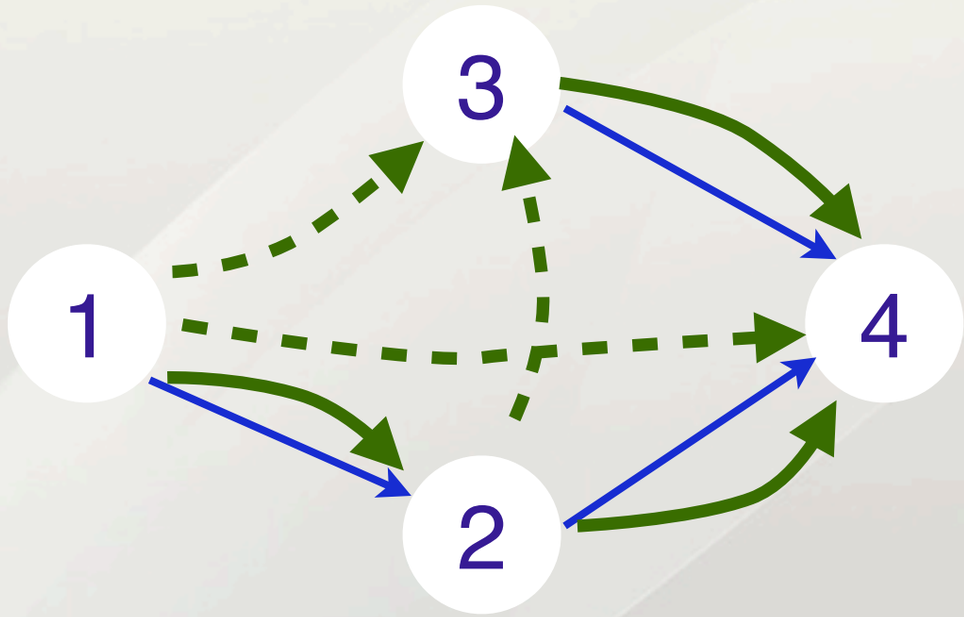
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2) ✔

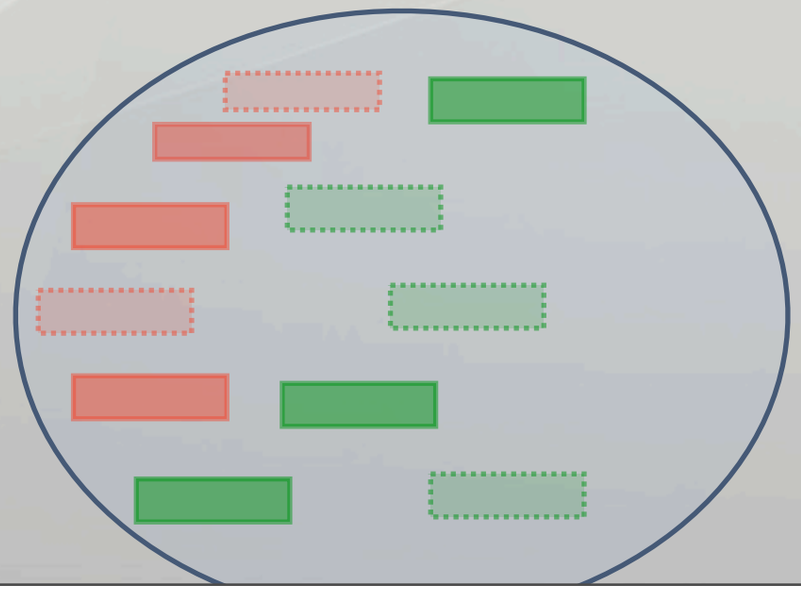
0.2 :: edge(2,3) ✗

0.5 :: edge(3,4) ✔

0.7 :: edge(2,4) ✔




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

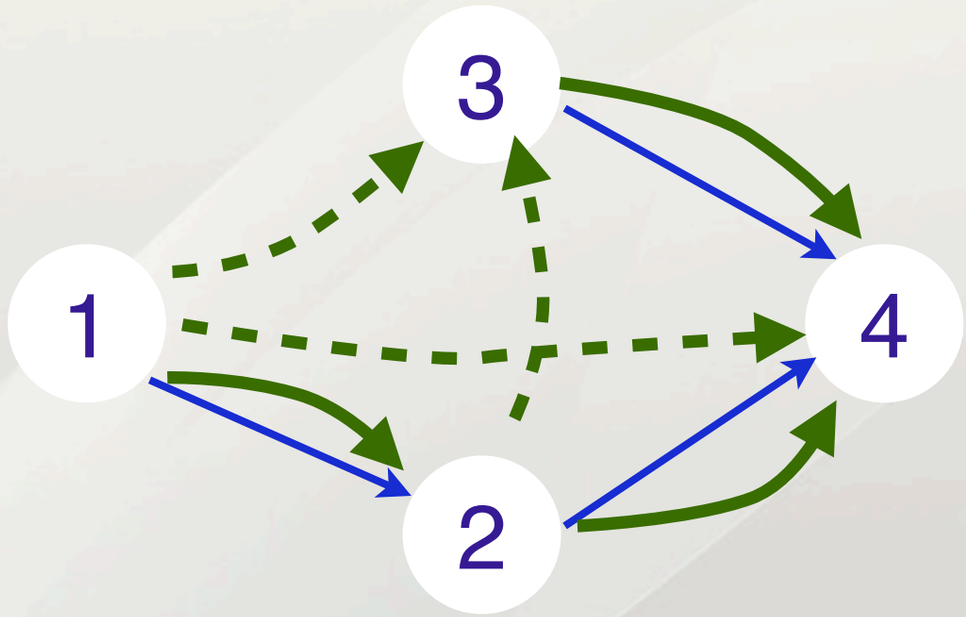
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2) ✓

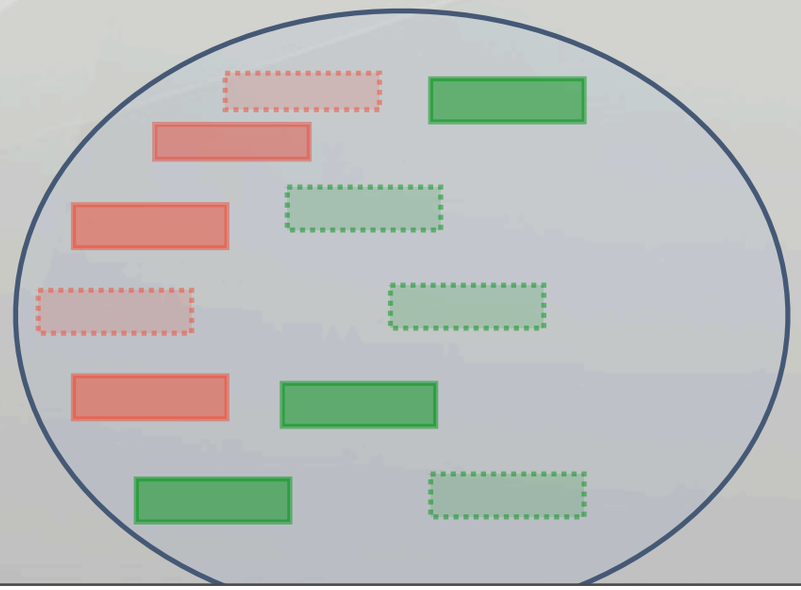
0.2 :: edge(2,3) ✗

0.5 :: edge(3,4) ✓

0.7 :: edge(2,4) ✓




```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```



Toy Example

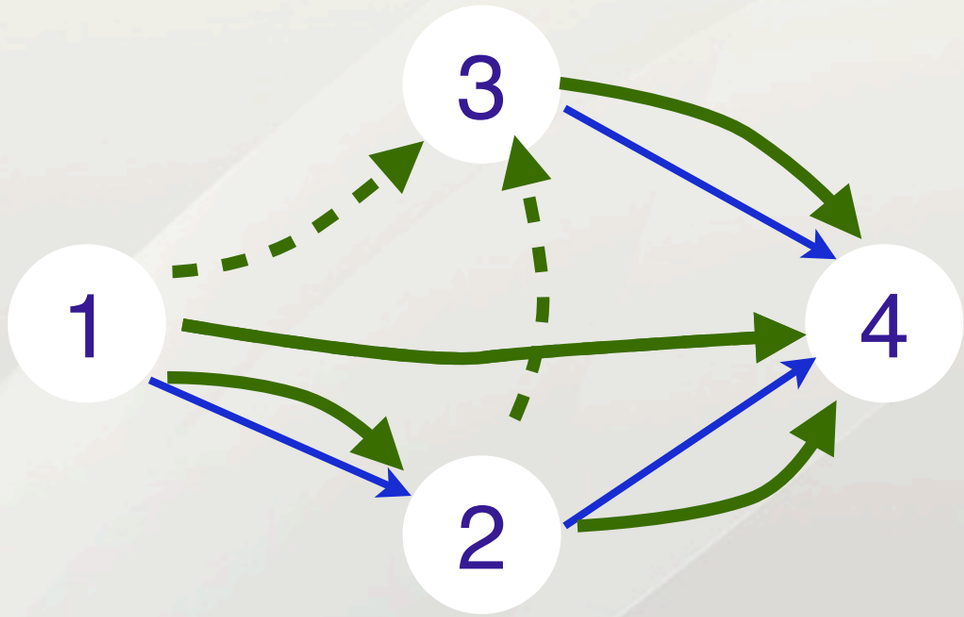
0.1 :: edge(1,3) ✗

0.6 :: edge(1,2) ✓

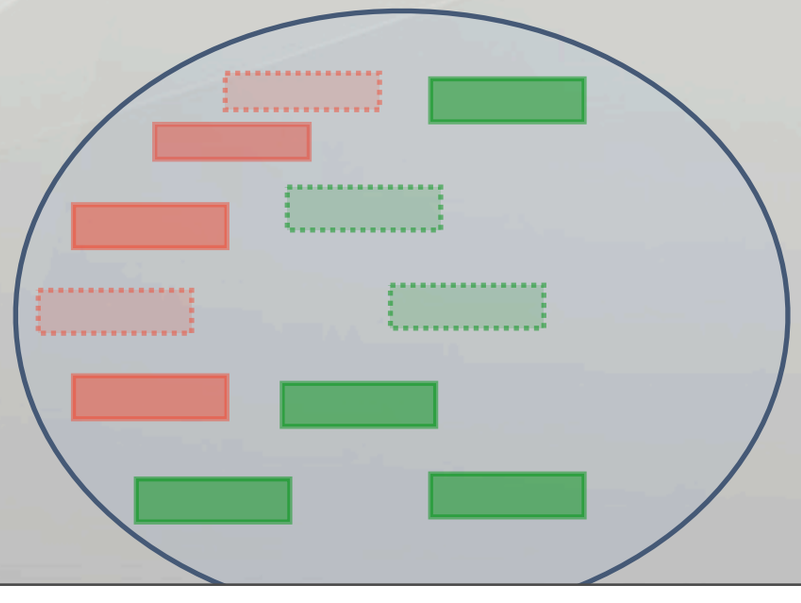
0.2 :: edge(2,3) ✗

0.5 :: edge(3,4) ✓

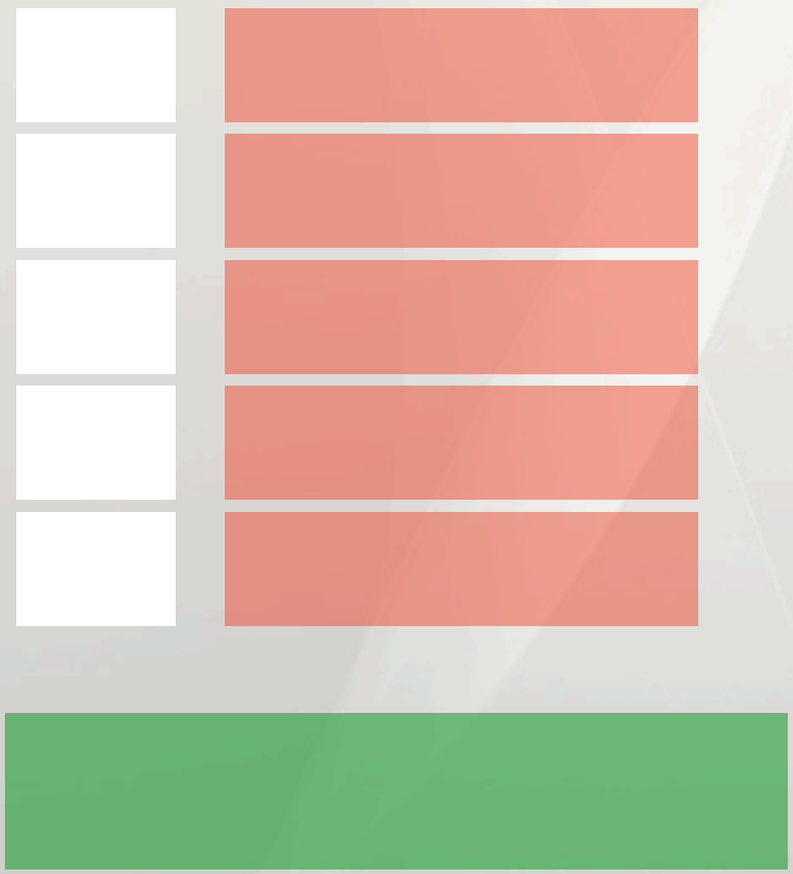
0.7 :: edge(2,4) ✓



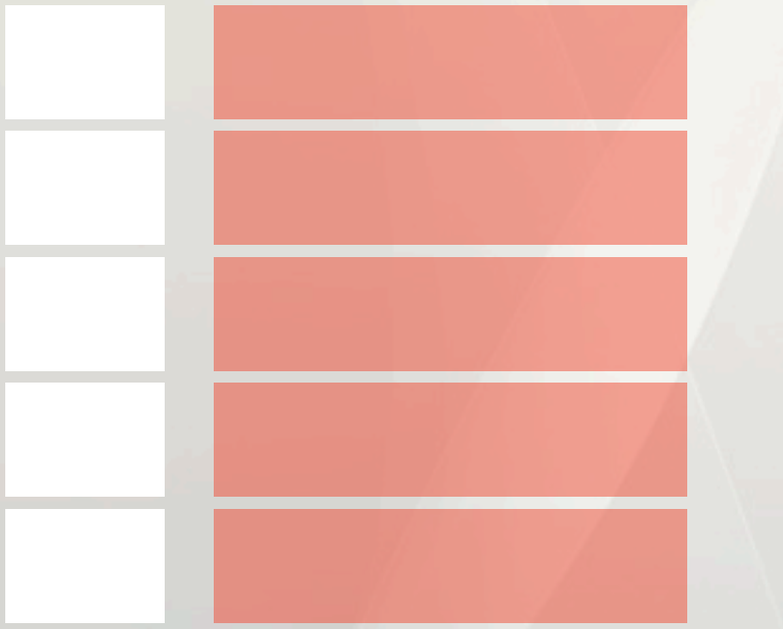

```
path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).
```

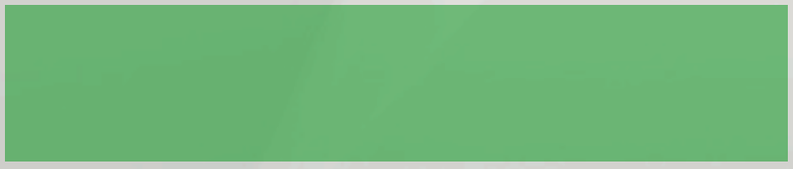


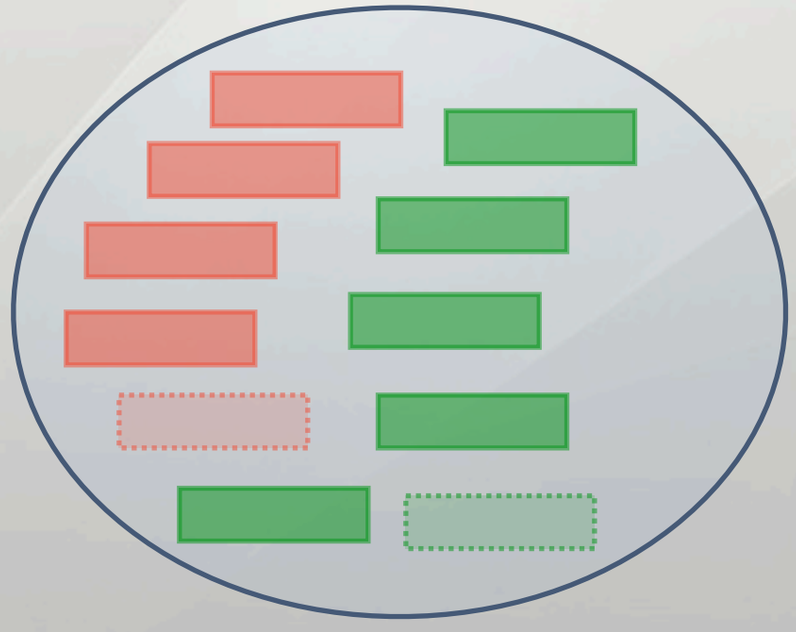
Sampling



Sampling







Sampling

??

??

??

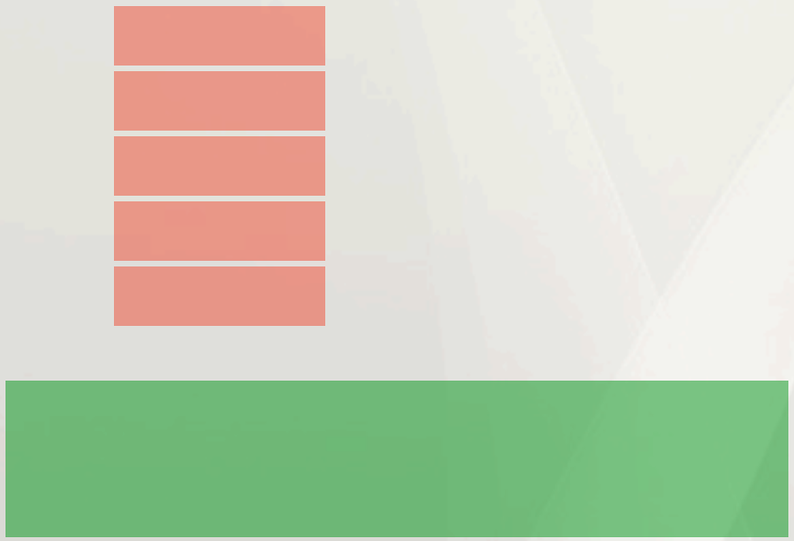
??

??

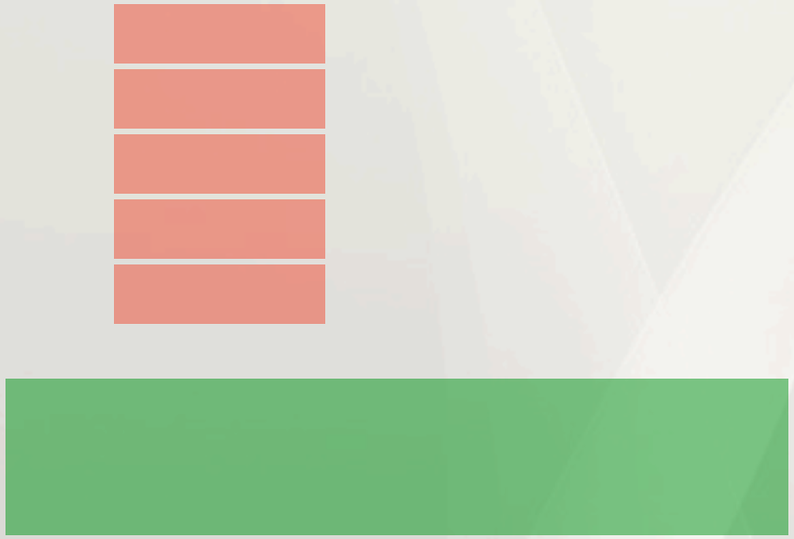
previous

next

Parameter Estimation

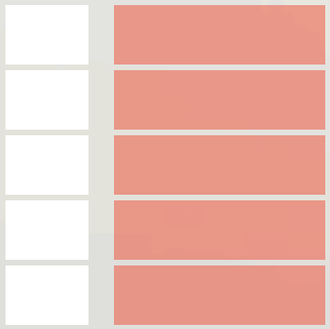


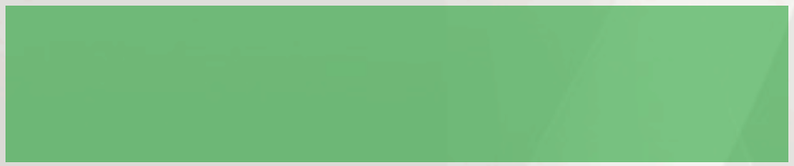
Parameter Estimation

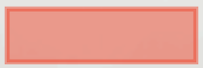


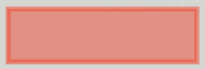
$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of observations}}$$

Parameter Estimation





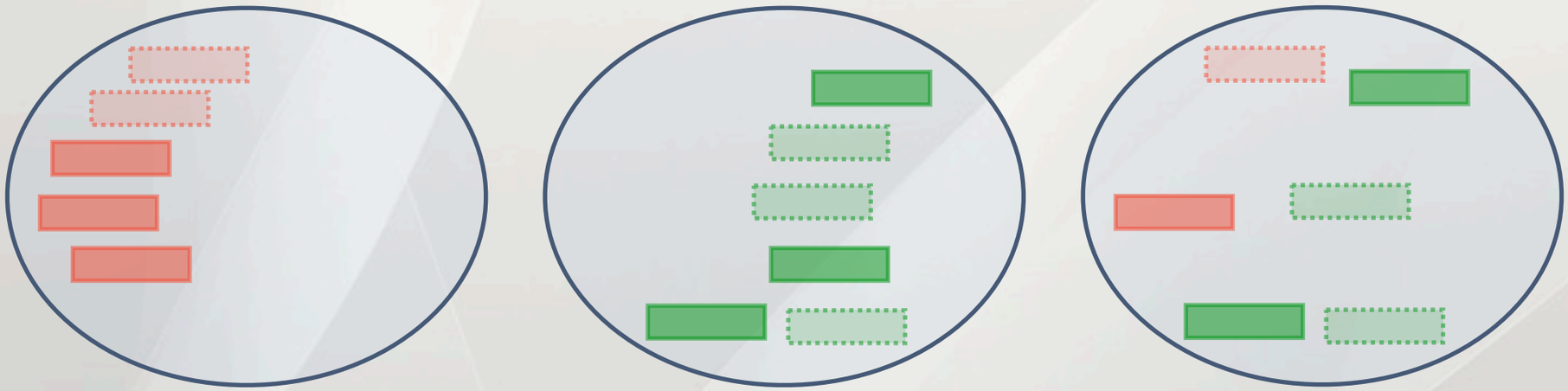




$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of observations}}$$

Missing Values

- Learning from **partial** interpretations



- Facts can be unobserved
- Soft-EM
- Use **expected_count** instead of **count**

LFI ProbLog

data examples

ProbLog program

CNF

diff with

abilistic

ectation
DD

e
probabilities

Wednesday 7 September 2011

LFI ProbLog

data examples

ProbLog
program

CNF

diff with

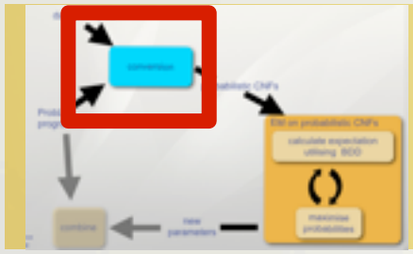
abilistic

ectation
DD

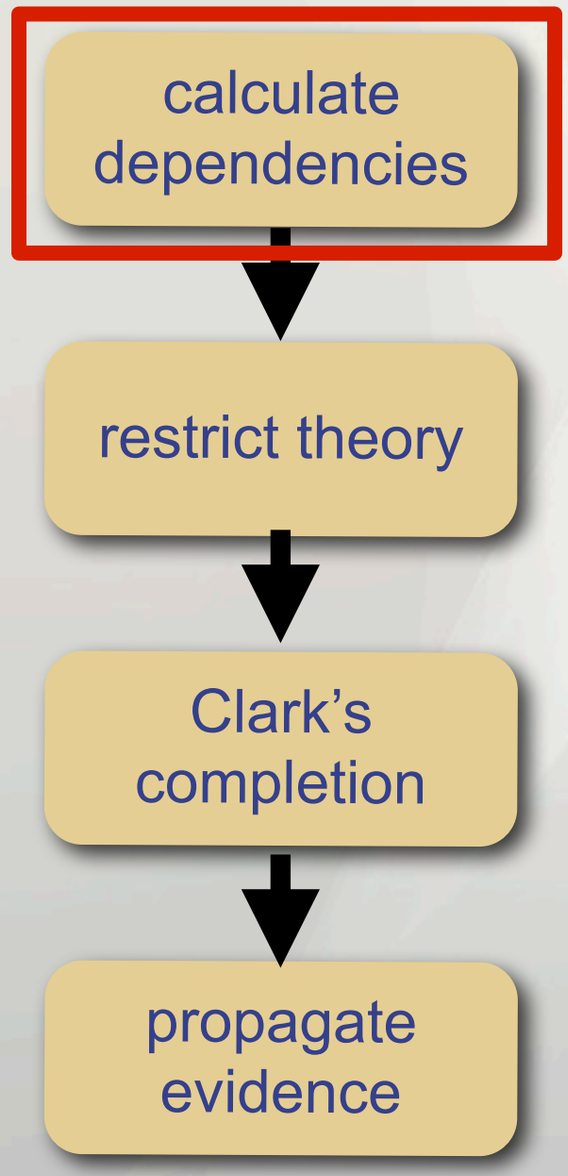
e
probabilities

Related Wo

- A. Darwich
- D. Fierens
- I. Thon et a



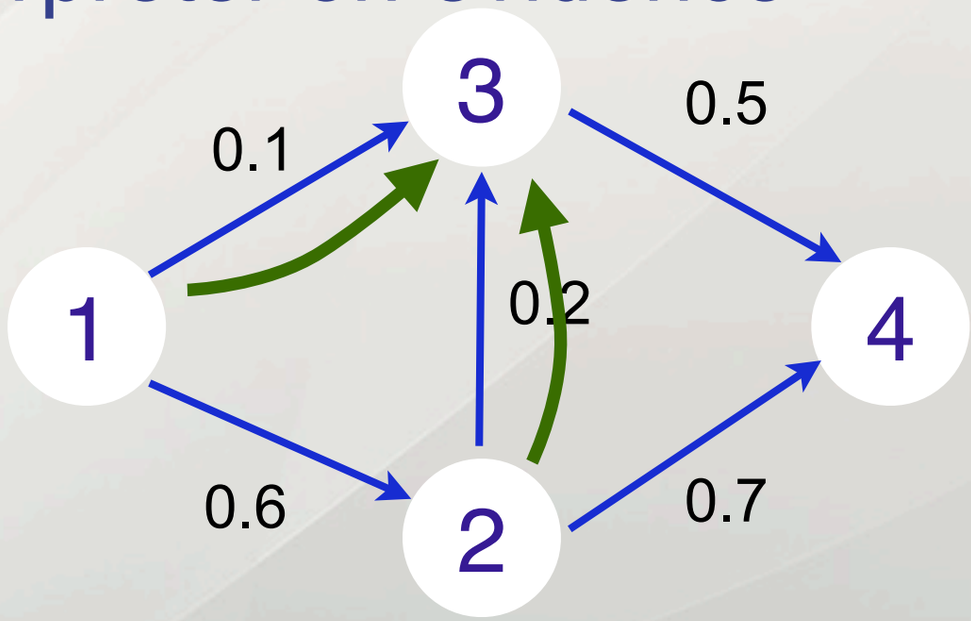
Conversion to Probabilistic CNF

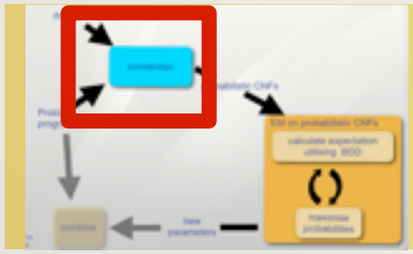


Goal: Find all relevant ground atoms

How: Run meta-interpreter on evidence

evidence:
{path(1,3) = true,
path(2,3) = true}





Conversion to Probabilistic CNF

calculate dependencies

restrict theory

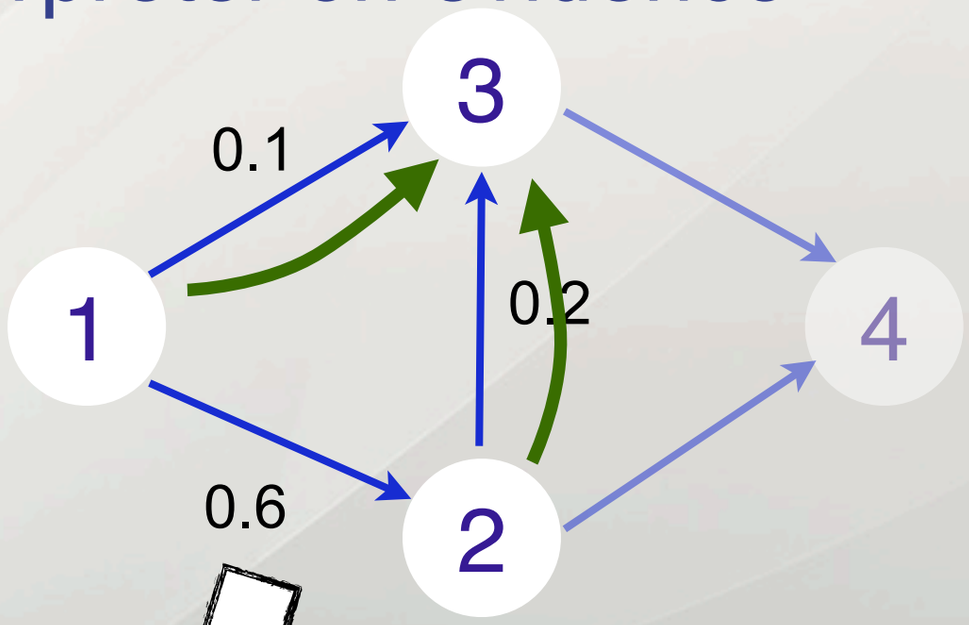
Clark's completion

propagate evidence

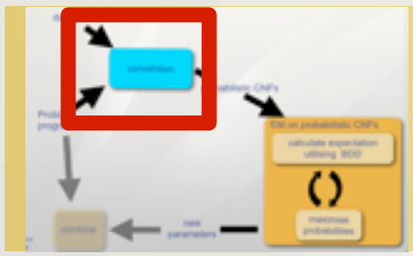
Goal: Find all relevant ground atoms

How: Run meta-interpreter on evidence

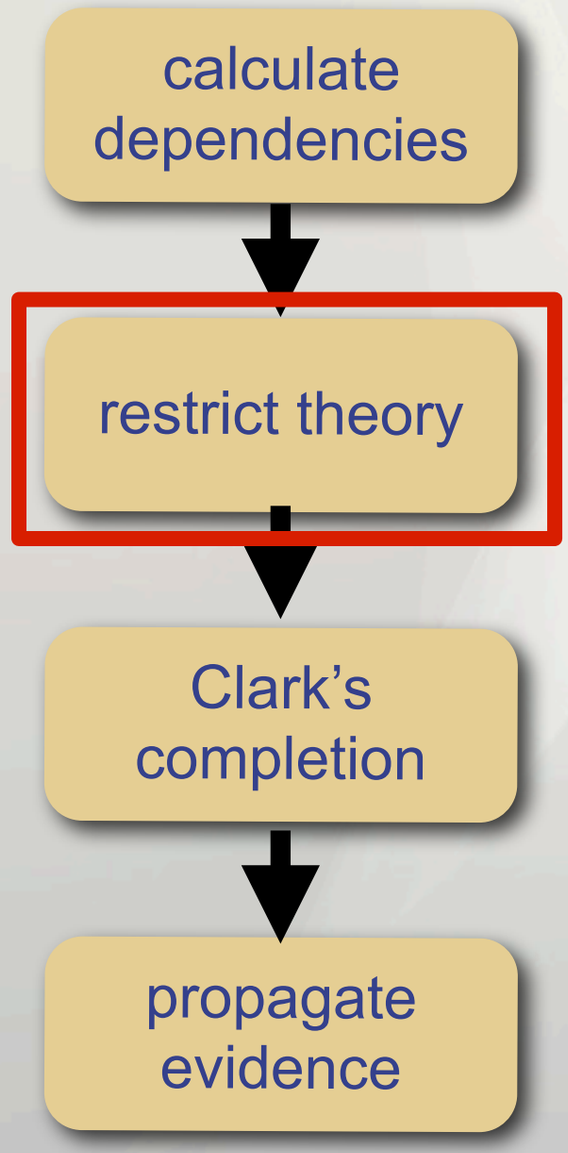
evidence:
{path(1,3) = true,
path(2,3) = true}



edge(1,2), edge(1,3), edge(2,3),
path(1,2),path(1,3), path(2,3),path(1,3)



Conversion to Probabilistic CNF

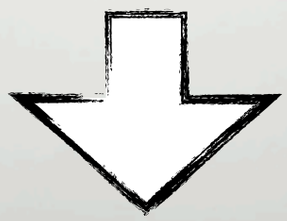


Goal: Finite ground program

How: Ground program using relevant fact

edge(1,2), edge(1,3), edge(2,3),
path(1,2), path(1,3), path(2,3), path(1,3)

path(A,B) :- edge(A,B).
path(A,B) :- edge(A,C),path(C,B).

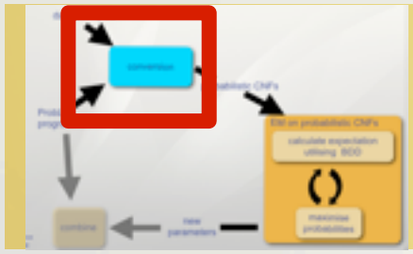


path(1,2) :- edge(1,2).

path(2,3) :- edge(2,3).

path(1,3) :- edge(1,3).

path(1,3) :- edge(1,2),path(2,3).



Conversion to Probabilistic CNF

calculate dependencies

restrict theory

Clark's completion

propagate evidence

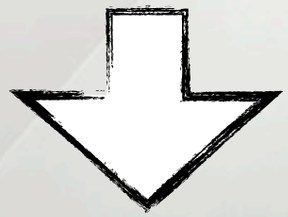
Goal: Propositional CNF

How: Clark's completion

path(1,2) :- edge(1,2). path(2,3) :- edge(2,3).

path(1,3) :- edge(1,3).

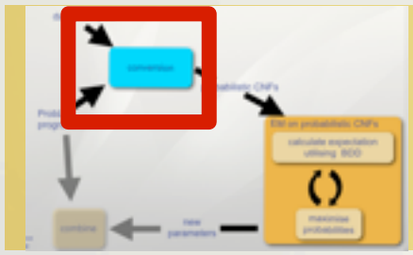
path(1,3) :- edge(1,2),path(2,3).



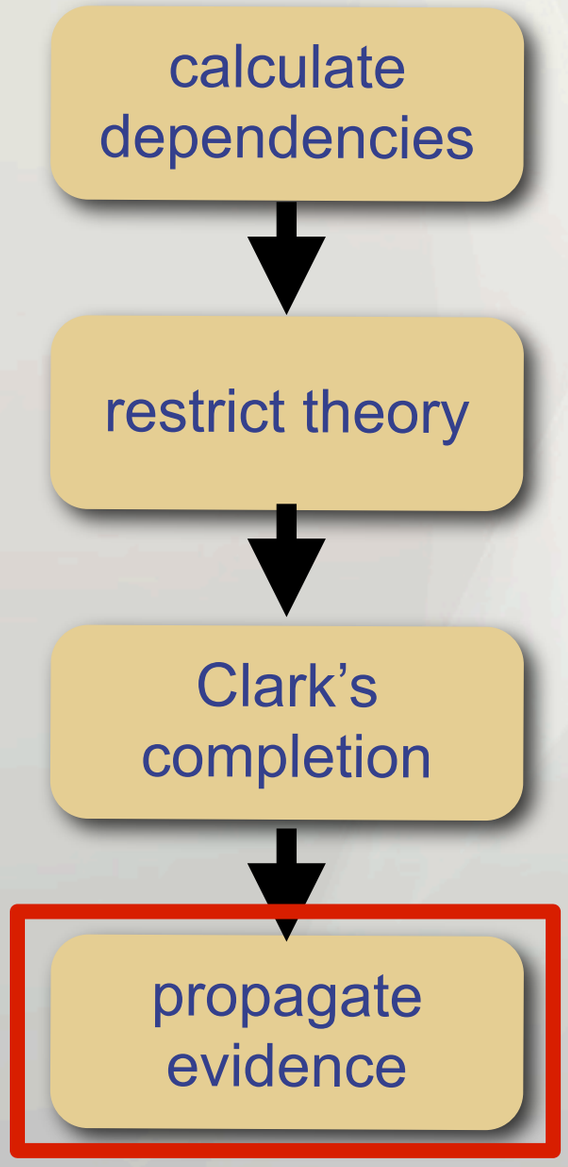
path(1,2) ⇔ edge(1,2)

∧ path(2,3) ⇔ edge(2,3)

∧ path(1,3) ⇔ (edge(1,3) ∨ (edge(1,2) ∧ path(2,3)))



Conversion to Probabilistic CNF



Goal: Simpler CNF

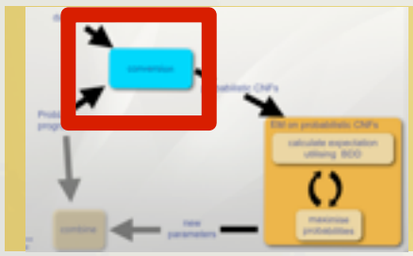
How: Unit propagation

I = {path(1,3) = true,
path(2,3) = true}

$$path(1,2) \Leftrightarrow edge(1,2)$$

$$\wedge\ path(2,3) \Leftrightarrow edge(2,3)$$

$$\wedge\ path(1,3) \Leftrightarrow (edge(1,3) \vee (edge(1,2) \wedge path(2,3)))$$



Conversion to Probabilistic CNF

calculate dependencies

restrict theory

Clark's completion

propagate evidence

Goal: Simpler CNF

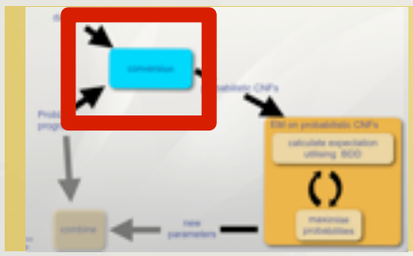
How: Unit propagation

I = {path(1,3) = true,
path(2,3) = true}

$$path(1,2) \Leftrightarrow edge(1,2)$$

$$\Lambda\ \cancel{path(2,3)}^{\text{true}} \Leftrightarrow edge(2,3)$$

$$\Lambda\ \cancel{path(1,3)}^{\text{true}} \Leftrightarrow (edge(1,3) \vee (edge(1,2) \wedge \cancel{path(2,3)}^{\text{true}}))$$



Conversion to Probabilistic CNF

calculate dependencies

restrict theory

Clark's completion

propagate evidence

Goal: Simpler CNF

How: Unit propagation

I = {path(1,3) = true,
path(2,3) = true}

path(1,2) ⇔ edge(1,2)

true

Λ path(2,3) ⇔ edge(2,3)

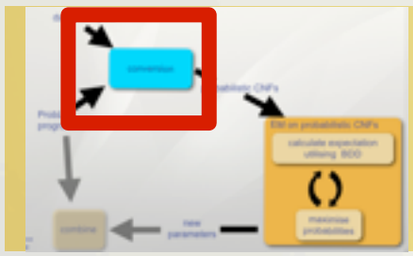
true

true

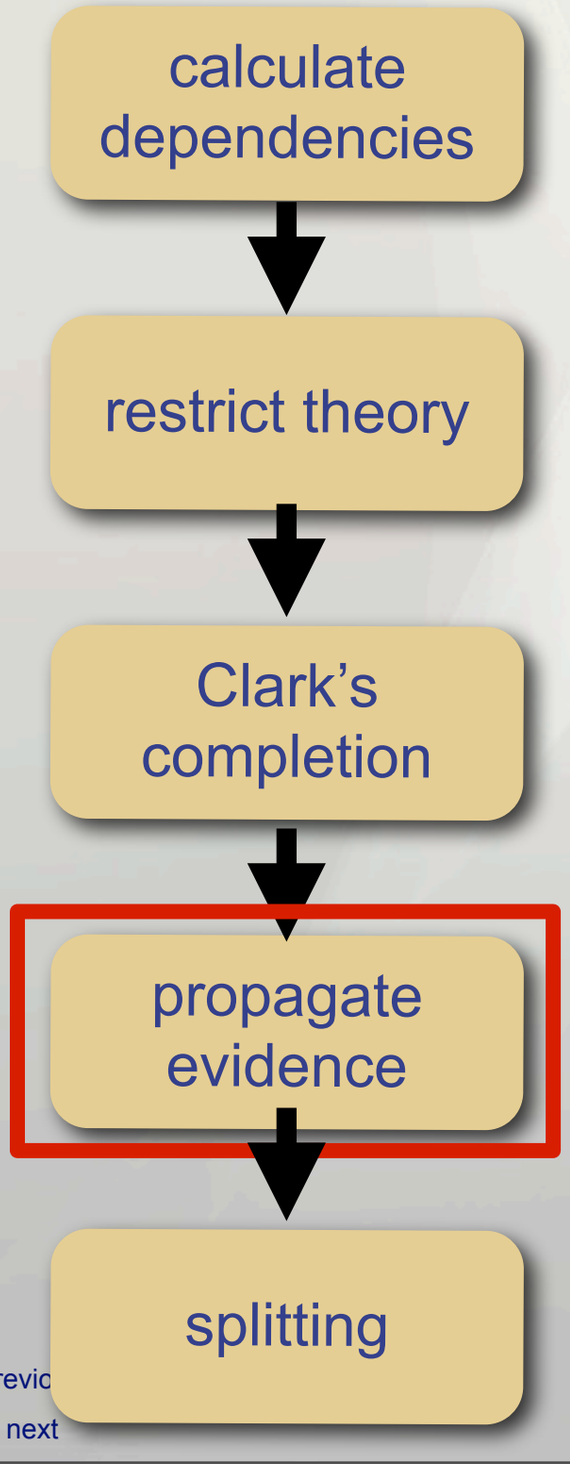
Λ path(1,3) ⇔ (edge(1,3) ∨ (edge(1,2) Λ path(2,3)))

true

propagated = { edge(2,3)= true)



Conversion to Probabilistic CNF



Goal: Simpler CNF

How: Unit propagation

I = {path(1,3) = true,
path(2,3) = true}

path(1,2) ⇔ edge(1,2)

true true

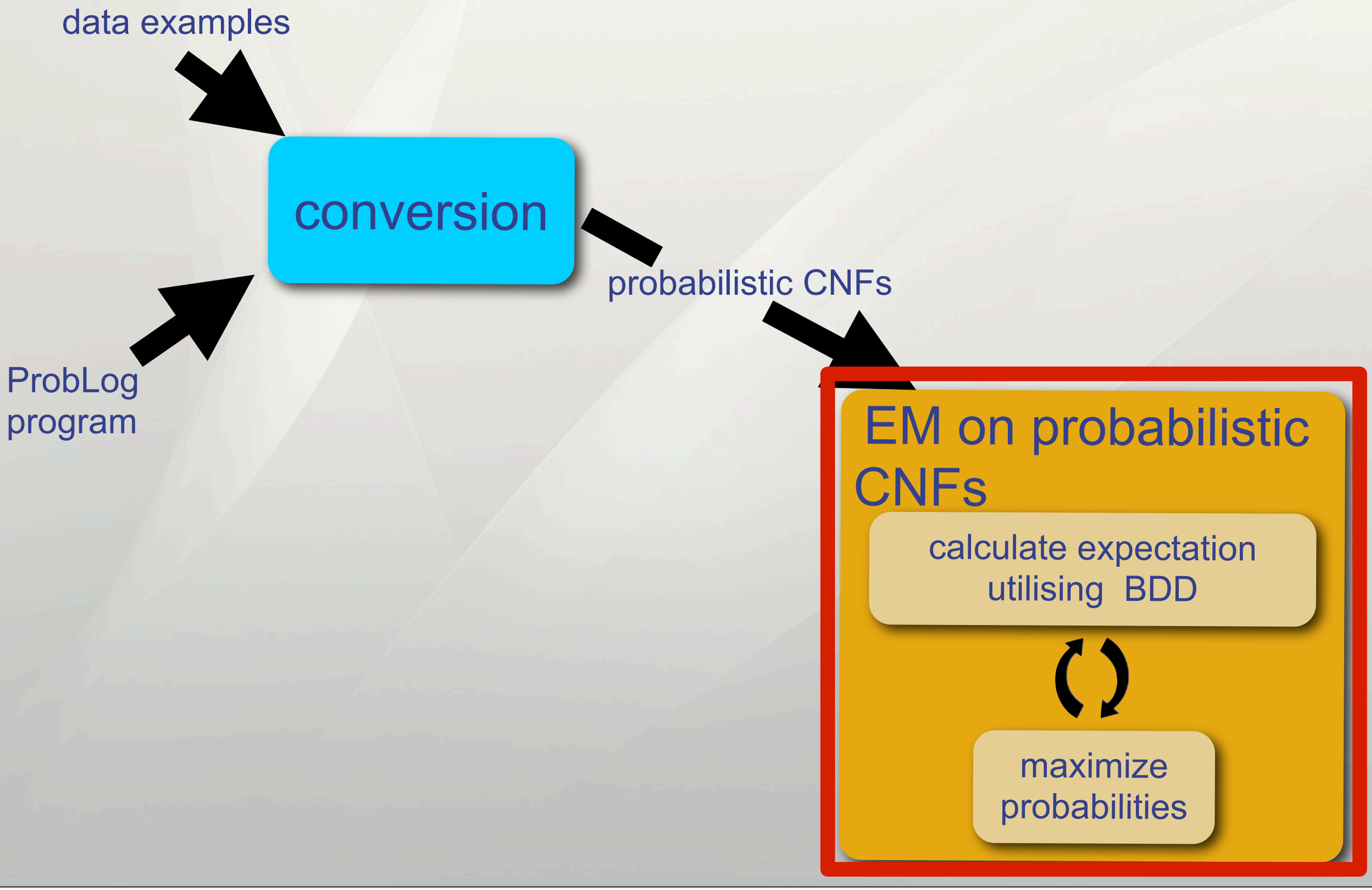
Λ path(2,3) ⇔ edge(2,3)

true true

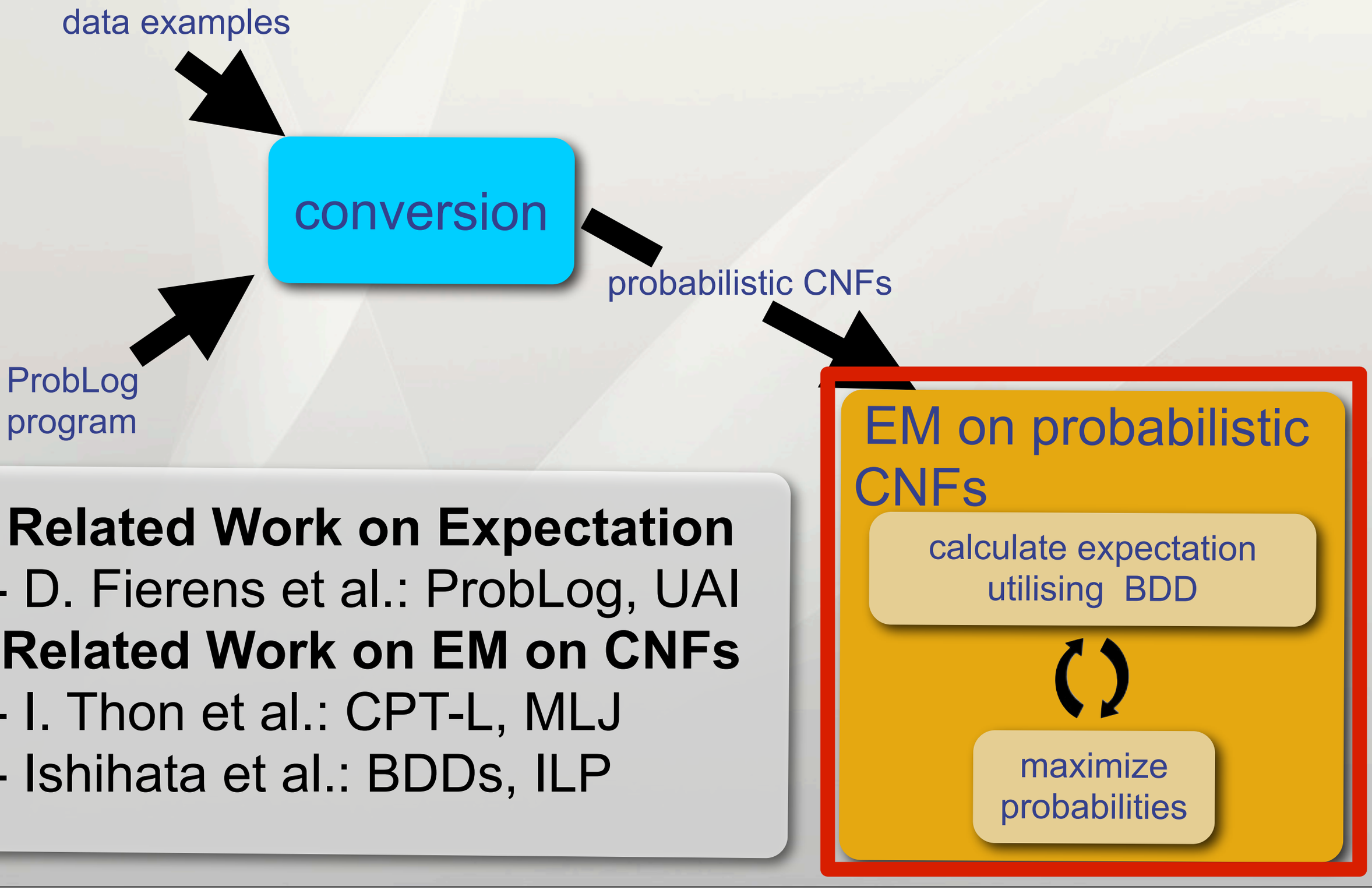
Λ path(1,3) ⇔ (edge(1,3) ∨ (edge(1,2) Λ path(2,3)))

propagated = { edge(2,3)= true)

LFI ProbLog


data examples
conversion
ProbLog program
probabilistic CNFs
EM on probabilistic CNFs
calculate expectation utilising BDD
maximize probabilities

LFI ProbLog

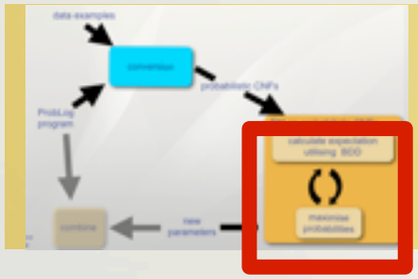


Related Work on Expectation

- D. Fierens et al.: ProbLog, UAI

Related Work on EM on CNFs

- I. Thon et al.: CPT-L, MLJ
- Ishihata et al.: BDDs, ILP



EM on CNFs

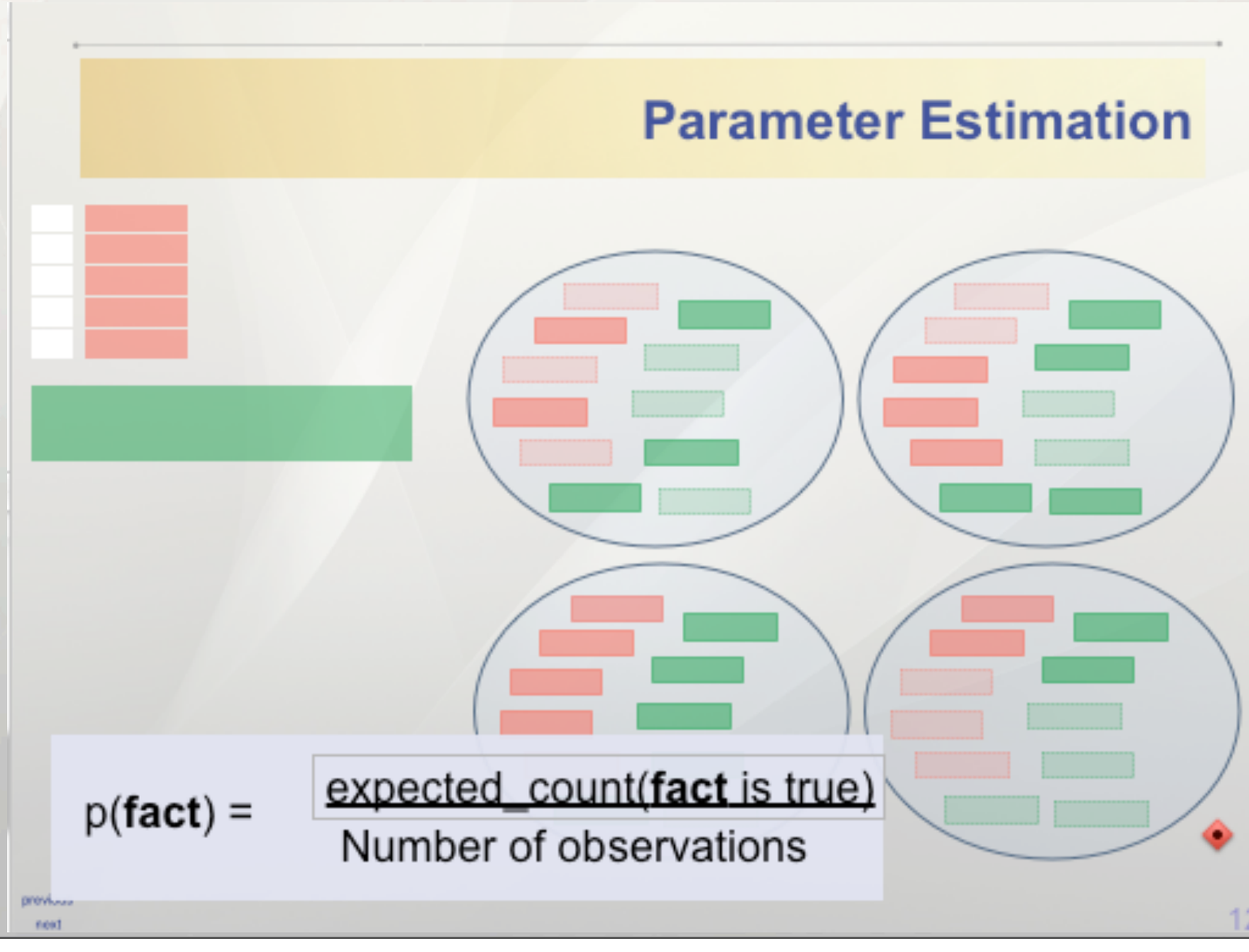
Goal: New parameters

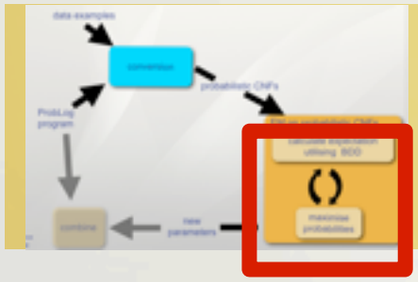
How: Counting expected counts

EM on probabilistic CNFs

calculate expectation utilising BDD

maximise probabilities





EM on CNFs

EM on probabilistic CNFs

calculate expectation utilising BDD

maximise probabilities

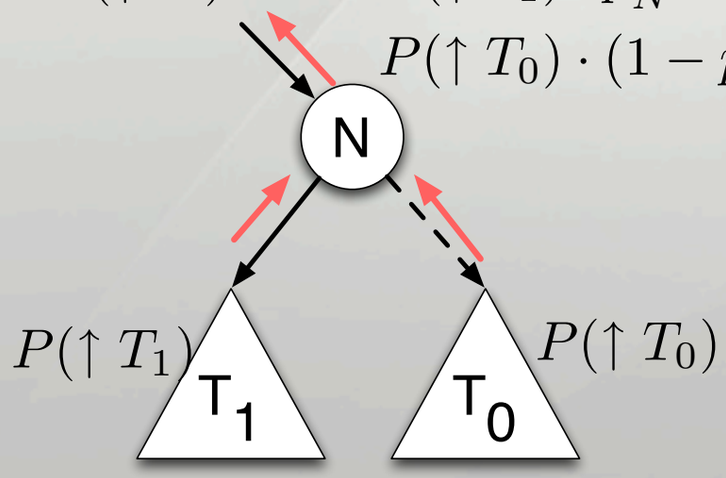
Goal: Marginalize over probabilistic facts

How: “Inside/outside” on BDDs

$$E[\delta^k_{m,n}|I_n] = \sum_{N \text{ represents } f_m} \frac{P(\downarrow N) \cdot p_N \cdot P(\uparrow h(N))}{P(BDD)}$$

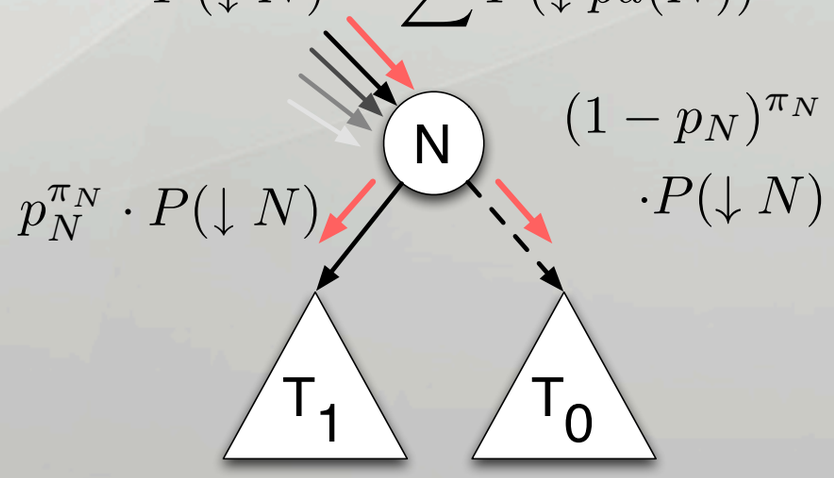
Upward Step

$$P(\uparrow N) = P(\uparrow T_1) \cdot p_N^{\pi_N} + P(\uparrow T_0) \cdot (1 - p_N)^{\pi_N}$$



Downward Step

$$P(\downarrow N) = \sum P(\downarrow pa(N))$$



Experimental Results

Real-world dataset: WebKB
Artificial dataset: Smoker

Q1: Is LFI-ProbLog competitive with existing state-of-the-art frameworks (like MLNs)?

Q2: Is LFI-ProbLog sensitive to the initial probabilities?

Q3: Is LFI-Problog able to recover the parameters of the original model with a reasonable amount of data?

Experimental Results

Real-world dataset: WebKB
Artificial dataset: Smoker

Q1: Is LFI-ProbLog competitive with existing state-of-the-art frameworks (like MLNs)?

YES

Q2: Is LFI-ProbLog sensitive to the initial probabilities?

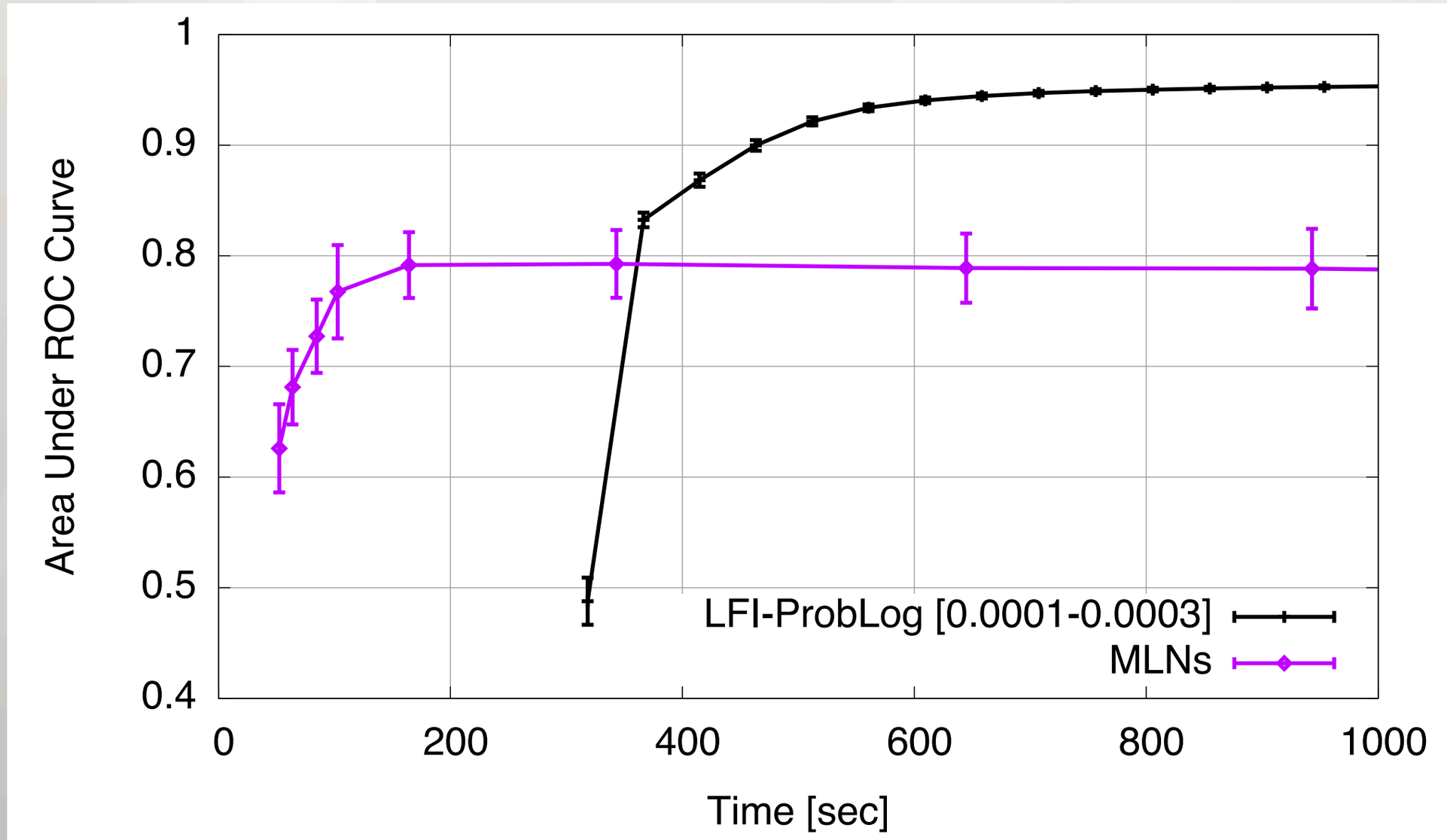
YES

Q3: Is LFI-Problog able to recover the parameters of the original model with a reasonable amount of data?

YES

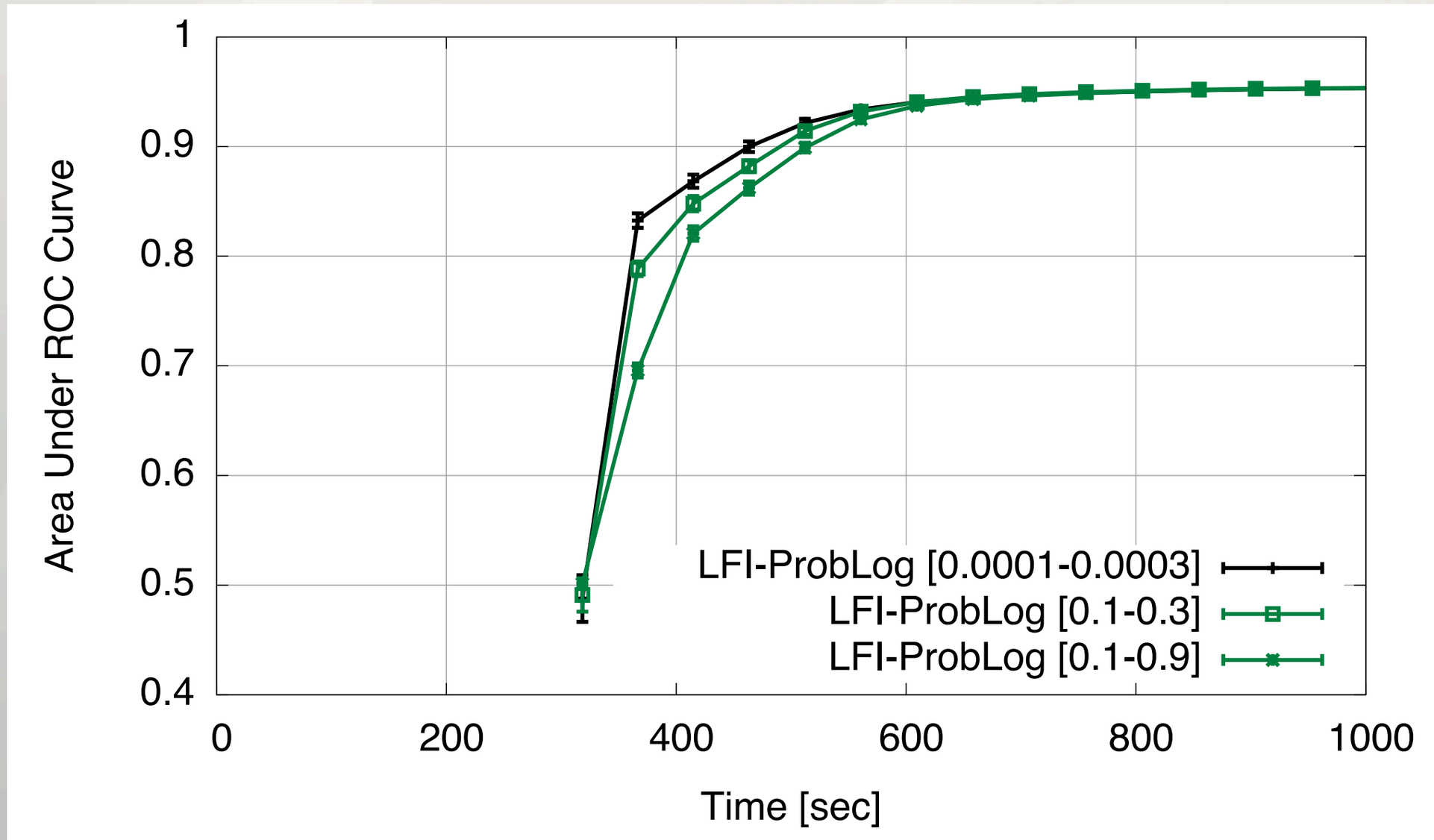
Experimental Results: WebKB

Q1: Is LFI-ProbLog competitive with existing state-of-the-art frameworks?



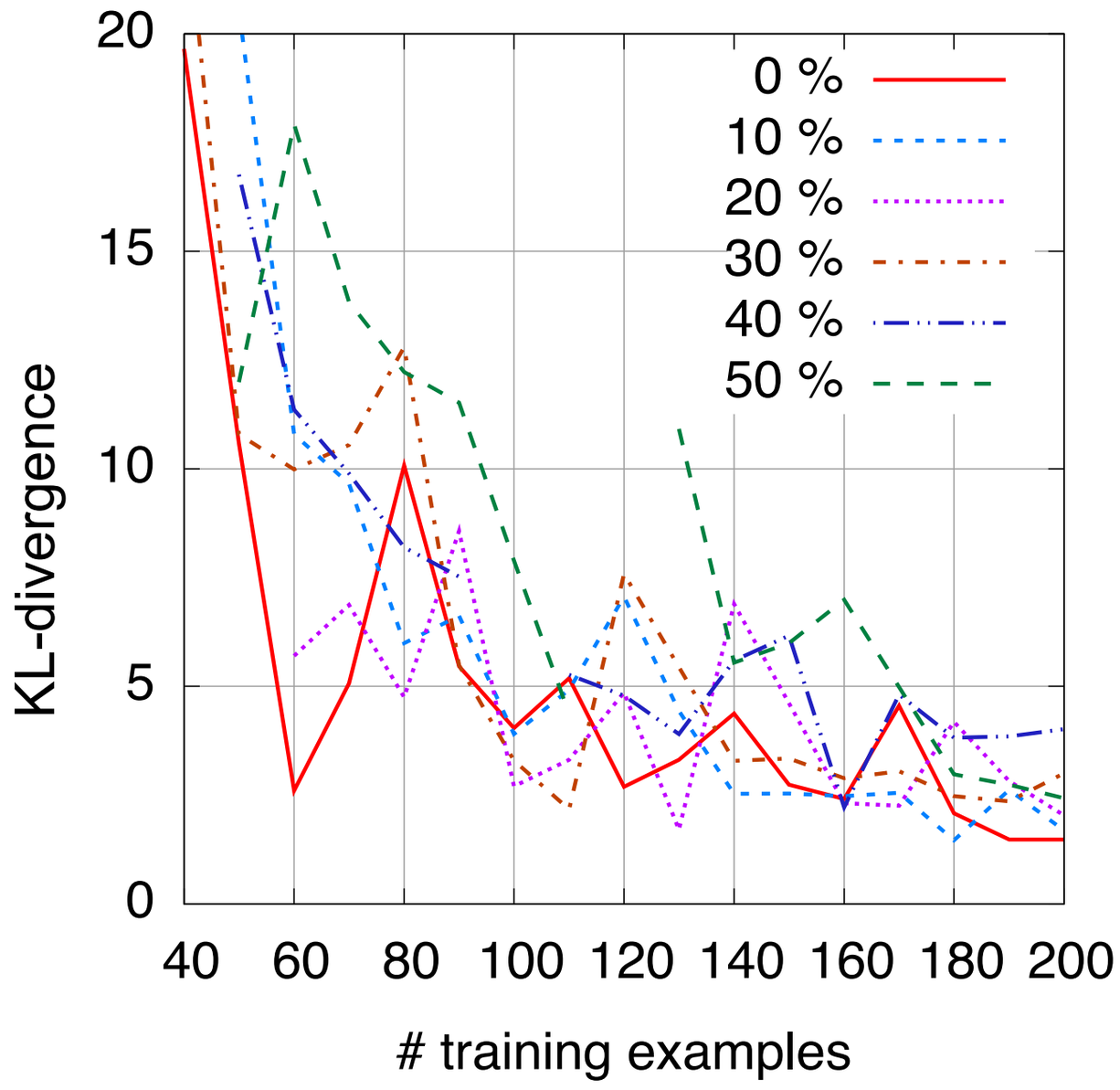
Experimental Results: WebKB

Q2: Is LFI-ProbLog insensitive to the initial probabilities?



Experimental Results: Smokers

Q3: Is LFI-Problog able to recover the parameters of the original model with a reasonable amount of data?



Goal t
sensi
towa
data

Conclusions

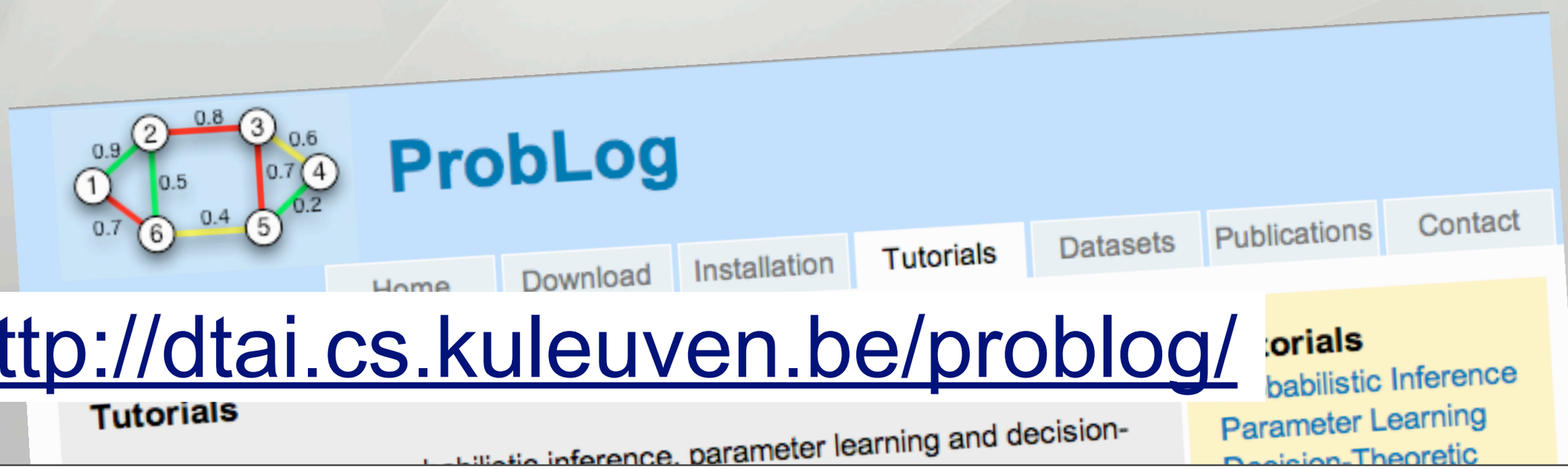
- Presented first algorithm for learning Probabilistic Logic Programs from interpretations
  - Learns in a generative setting
  - Handles missing/incomplete data

- Empirical results:
  - Accuracy competitive with other state of the art
  - Run time efficiency sufficient for real world tasks

- Publicly available implementation: Download and try it out!!
  - http://dtai.cs.kuleuven.be/problog/

Thank you

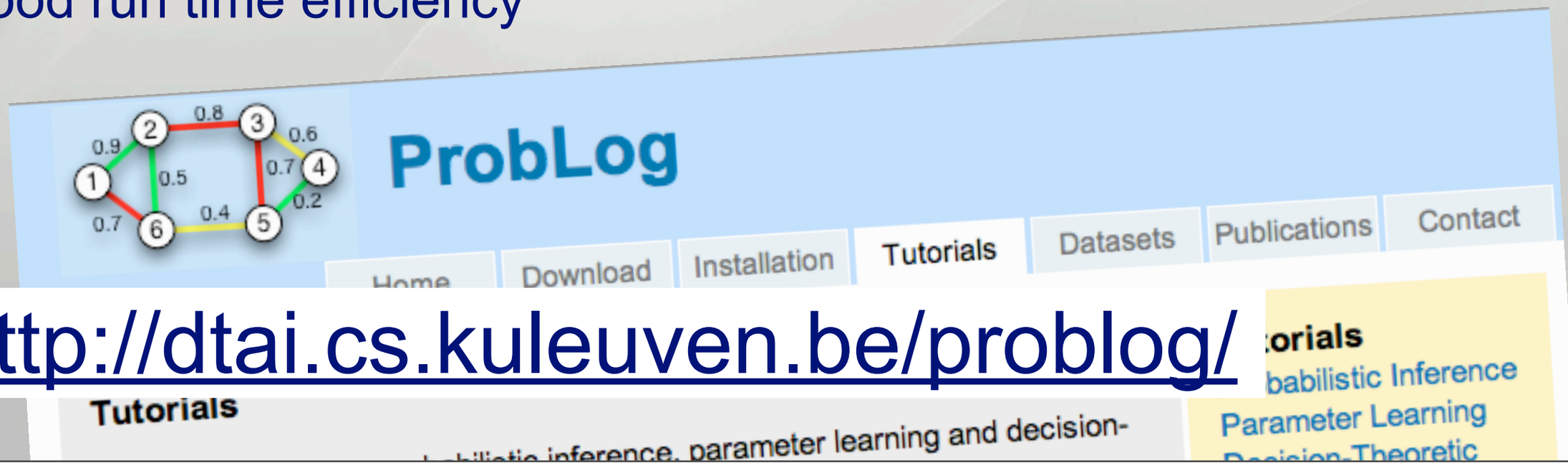
Questions?



http://dtai.cs.kuleuven.be/problog/

Conclusions

- Presented first ProbLog algorithm capable of learning from interpretation
  - Can learn in a generative setting
  - Can handle missing/incomplete data
- Strong empirical results
  - Accuracy competitive with other state of the art
  - Good run time efficiency




STOP