DTAI

MACHINE LEARNING

Capita Selecta AI:

Probabilistic Programming

Wannes Meert

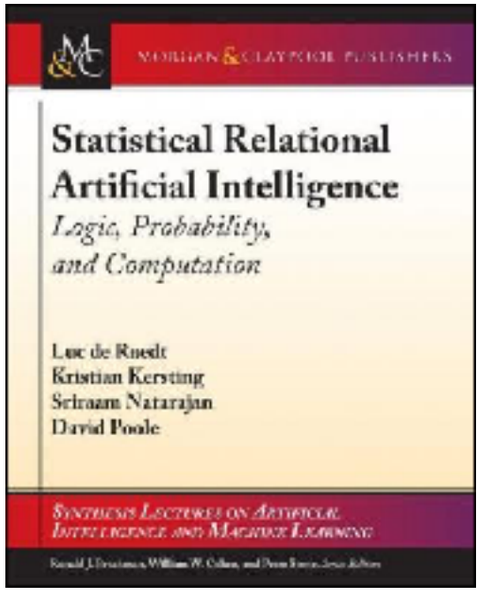
19/10/2016

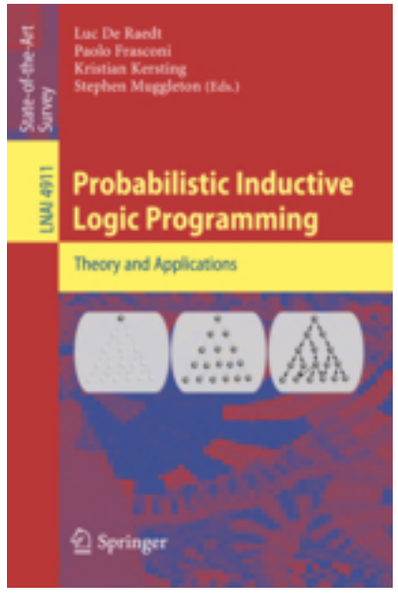
Slides courtesy of Angelika Kimmig, Luc De Raedt and Guy Van den Broeck

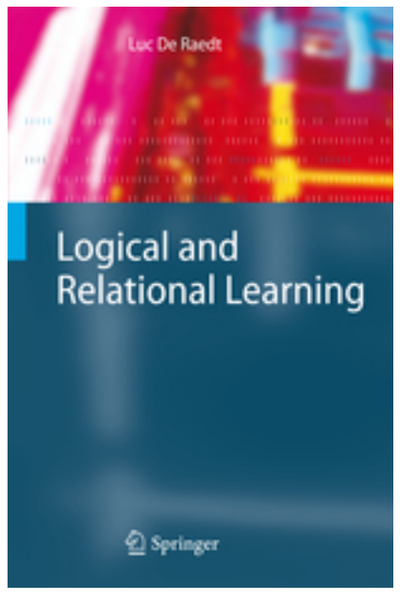
KU LEUVEN

What?

- Probabilistic (logic) learning & related fields
- A core research topic in the DTAI research group
- Two lectures
  - Today: Probabilistic Programming how-to (ProbLog)
  - December 7: broader view and advanced topics
- Assignment: Inference for ProbLog hands-on
  - Evaluation based on short written report







Today

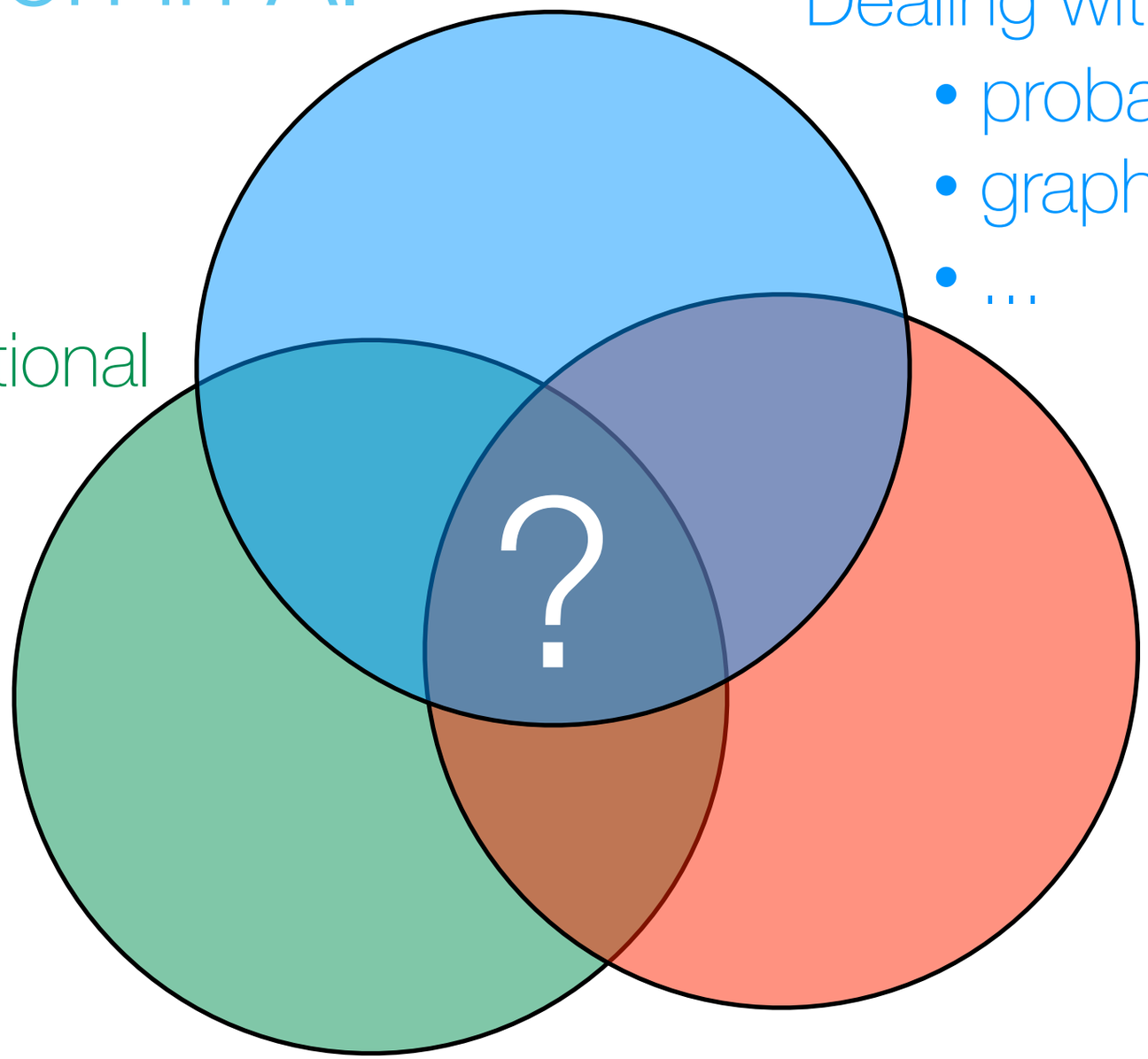
- What is Probabilistic Programming?
- ProbLog: introduction and inner workings
- Learning: parameters and structure
- Language Extensions and Variants
- Assignment

Probabilistic Programming

A key question in AI

Dealing with uncertainty
- probability theory
- graphical models
- …

Reasoning with relational data
- logic
- code
- databases
- programming
- …



Learning
- parameters
- structure

Statistical relational learning, probabilistic logic learning, probabilistic programming, …

KU LEUVEN

Many formalisms exist

~Par-factor based

FACTORIE
ProbLog
Markov Logic
PRISM
Bayesian Logic Programs
Probabilistic Soft Logic
CLP(BN)
ICL
Relational Bayesian networks
Probabilistic Relational Models
…

~MCMC based

Church
Stan-MC
PyMC
Anglican
Venture
BayesDB
Bayesian Logic (BLOG)
…

http://www.starai.org http://probabilistic-programming.org

KU LEUVEN

Why Relational?

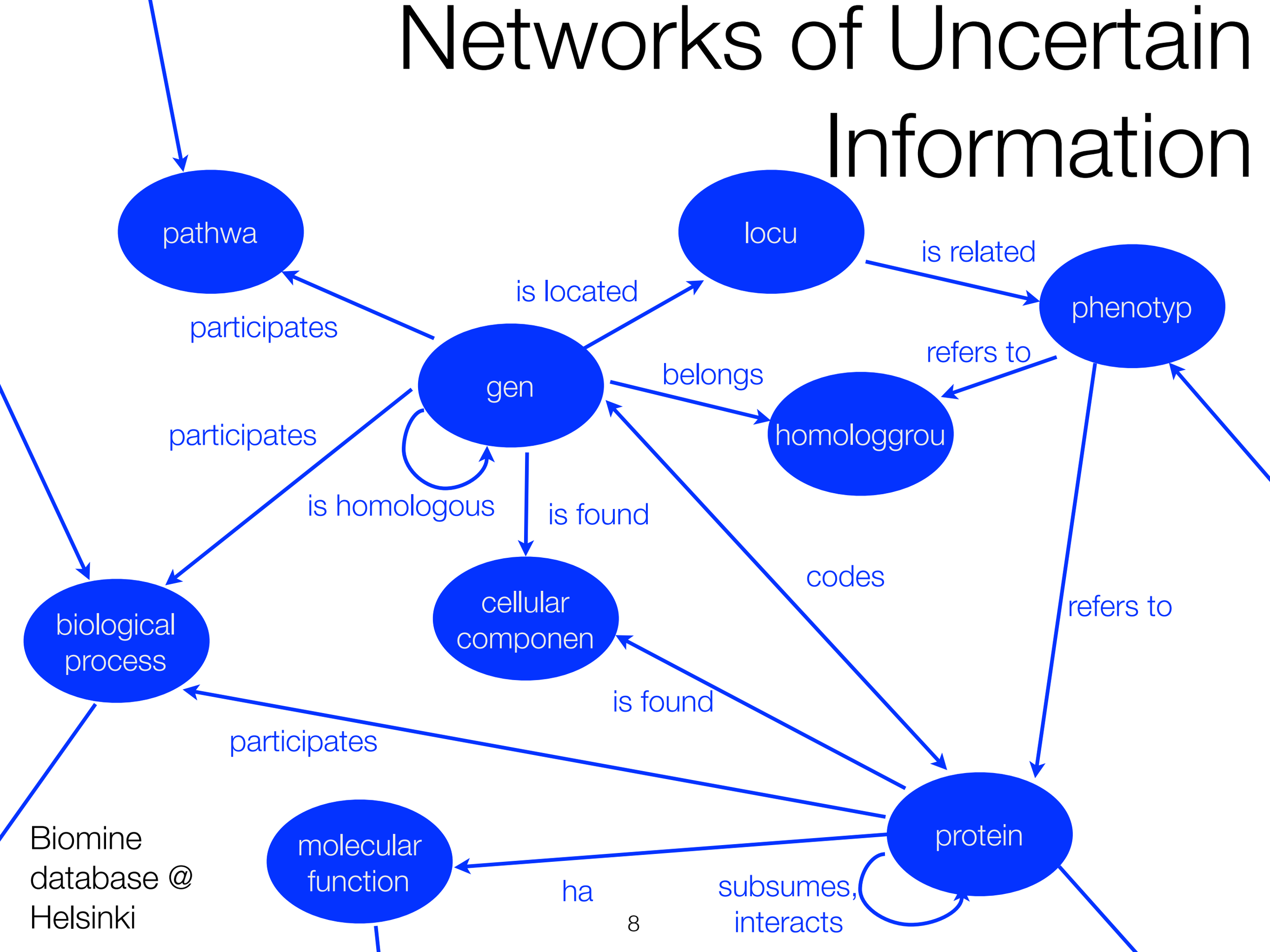
- Our data is already relational!
  - Companies run relational databases
  - Scientific data is relational:
    - Large Hadron Collider generated 25PB in 2012
    - LSST Telescope will produce 30TB per night
  - Here: relational ≠ (normalized) structured
    - Relations link heterogeneous data sources (tables/db/…)

- Big data is big business:
  - Oracle: $7.1BN in sales
  - IBM: $3.2BN in sales
  - Microsoft: $2.6BN in sales

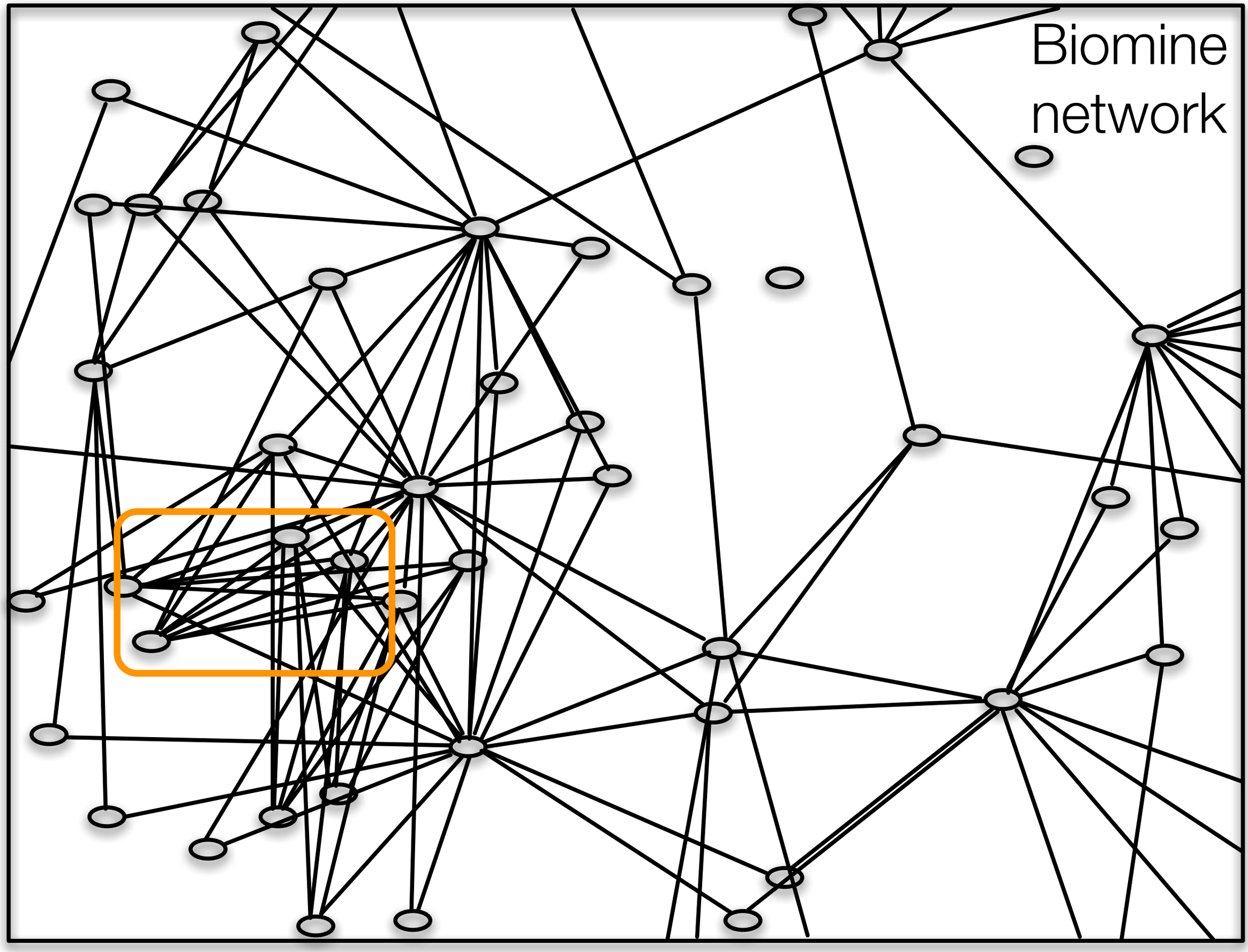
≈ GDP of Iceland!

KU LEUVEN

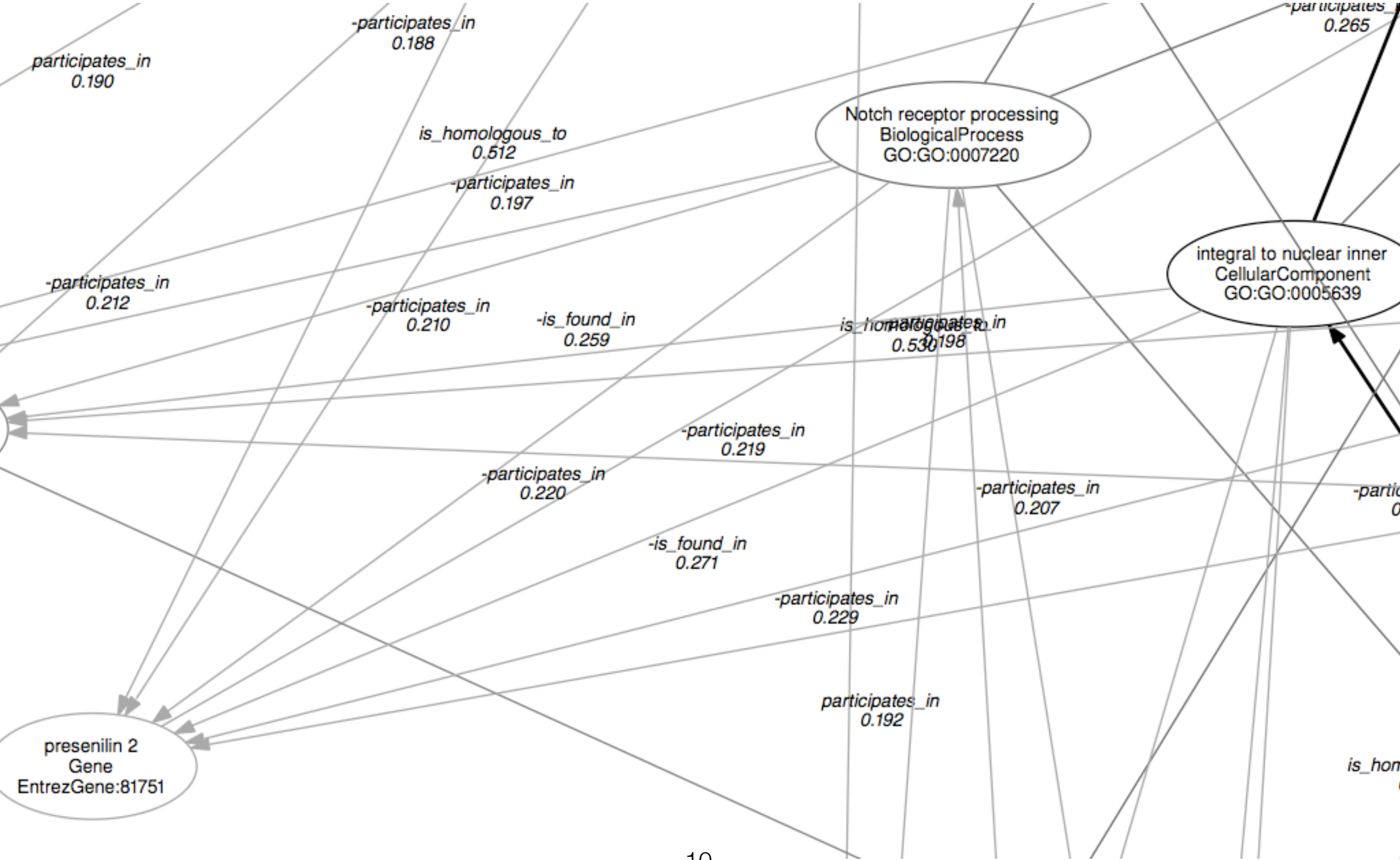
Networks of Uncertain Information



Biomine database @ Helsinki

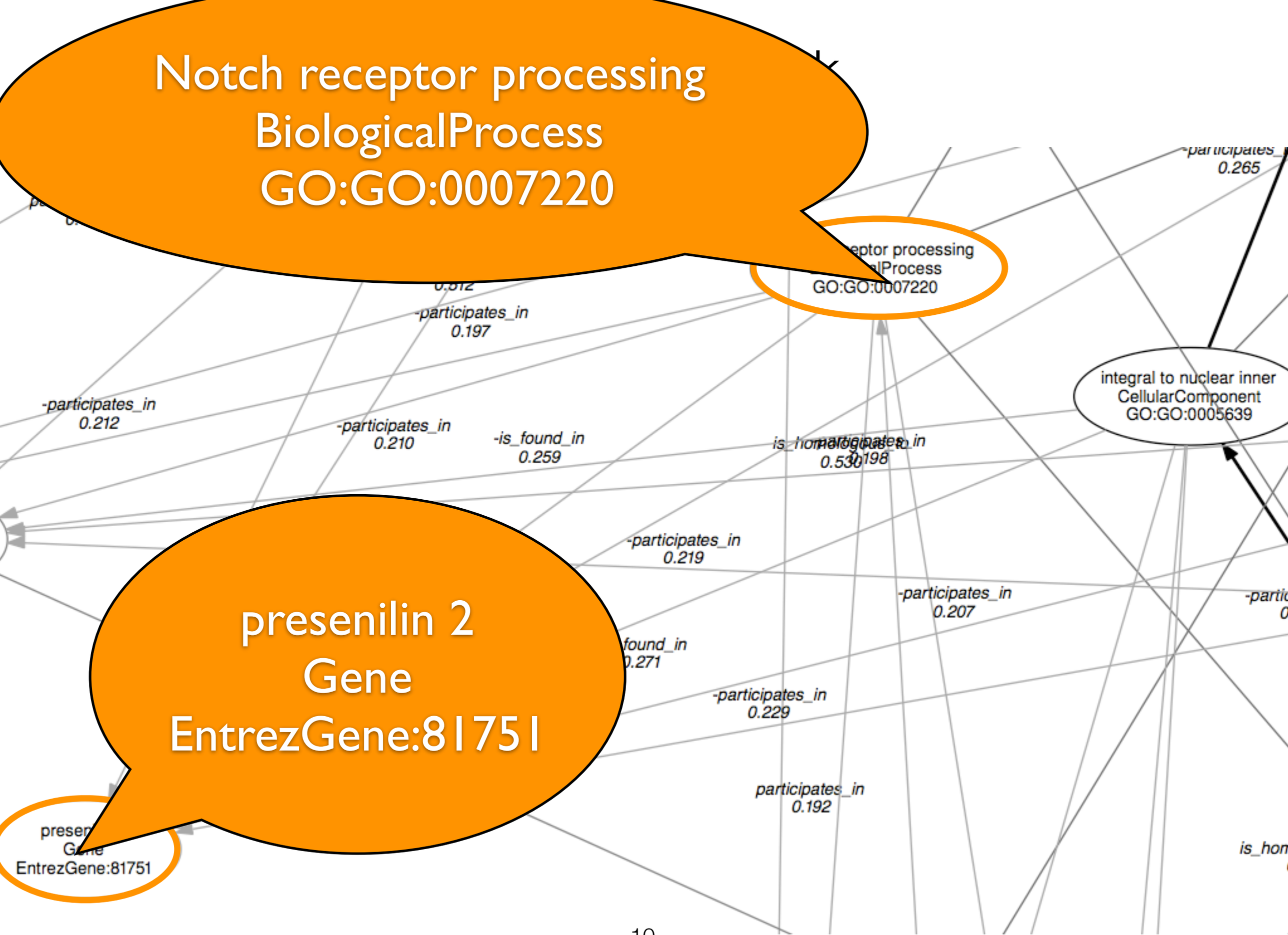

Biomine
network

Biomine Network

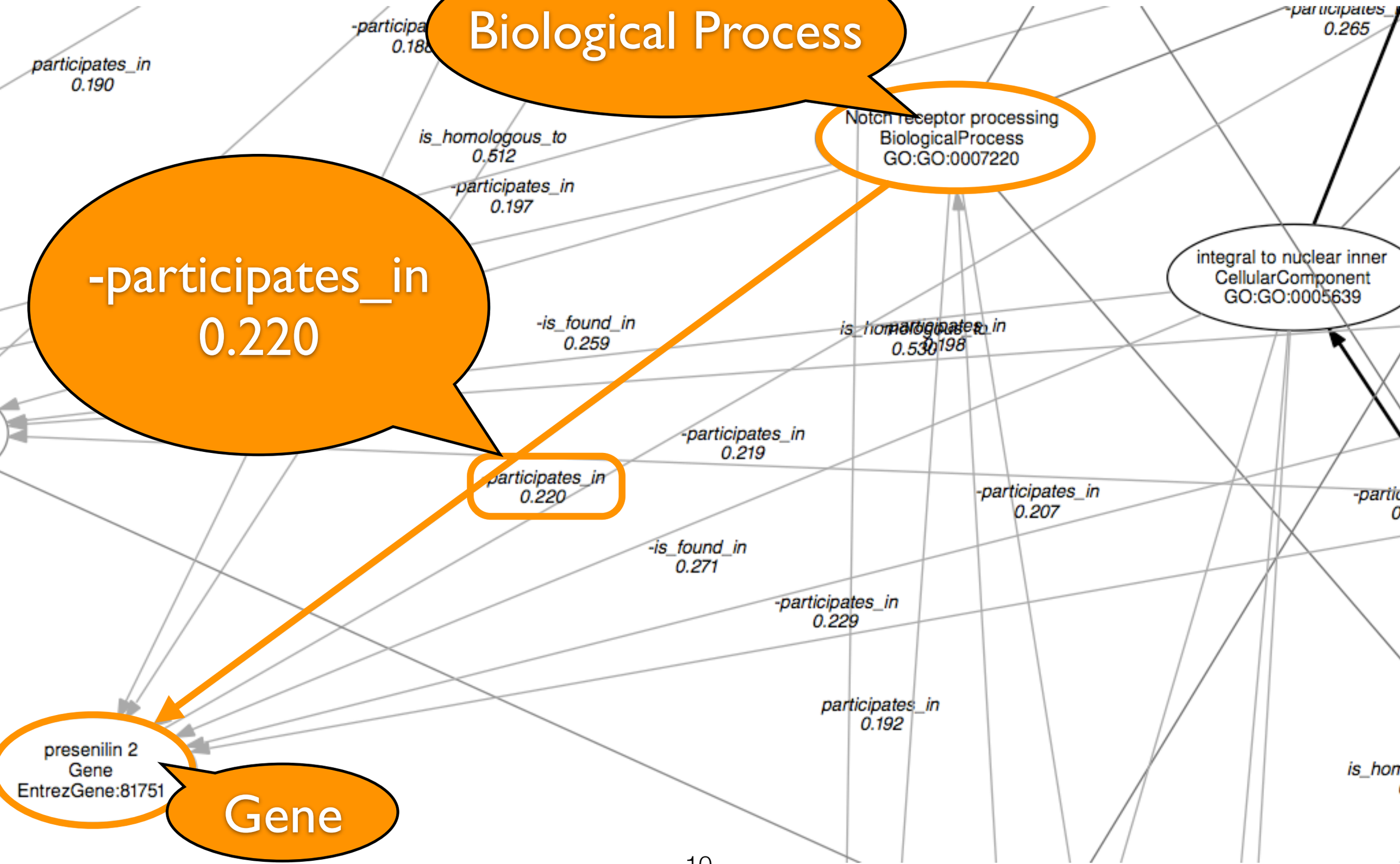


Notch receptor processing
BiologicalProcess
GO:GO:0007220

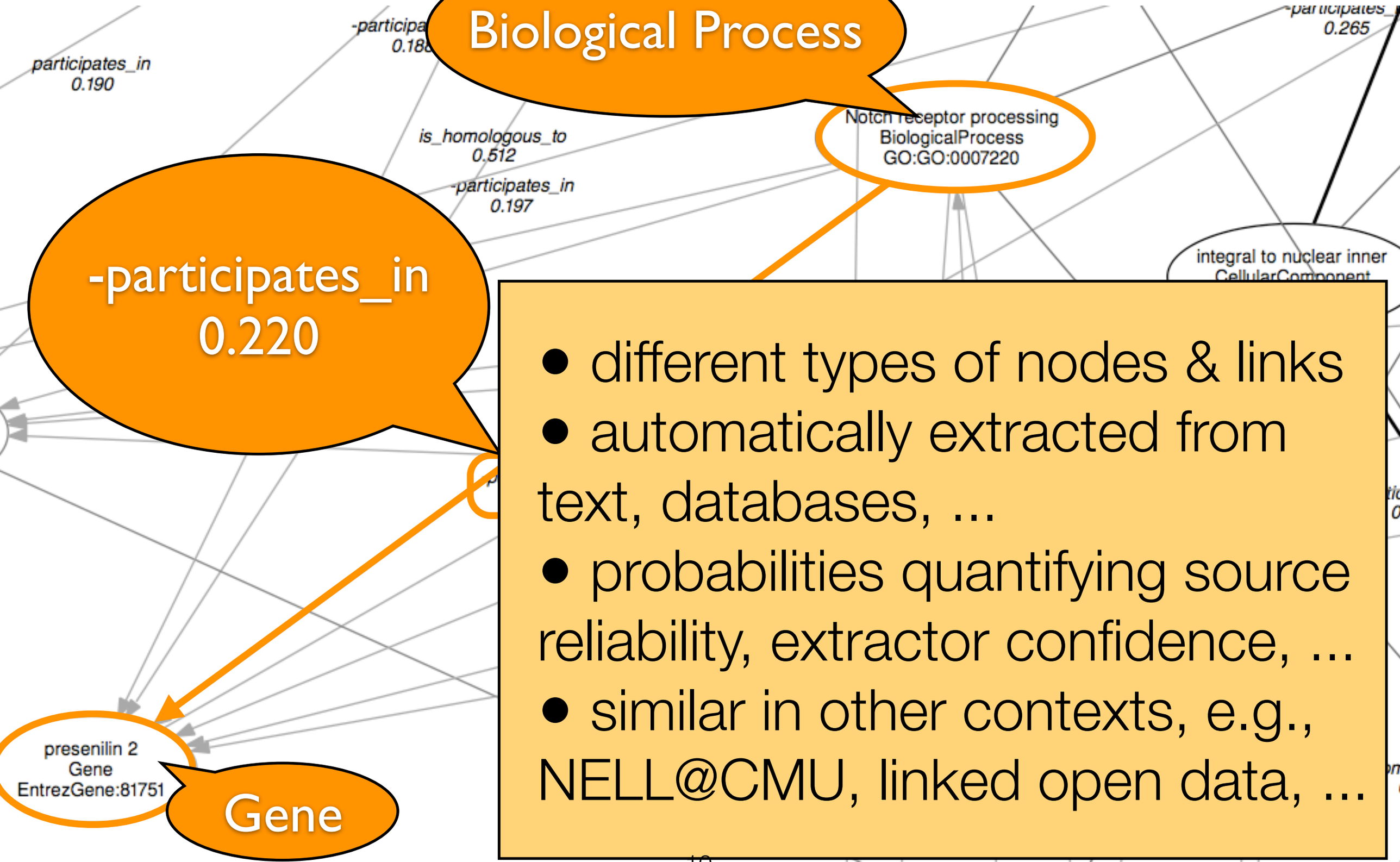
presenilin 2
Gene
EntrezGene:81751



Biomine Network



Biomine Network

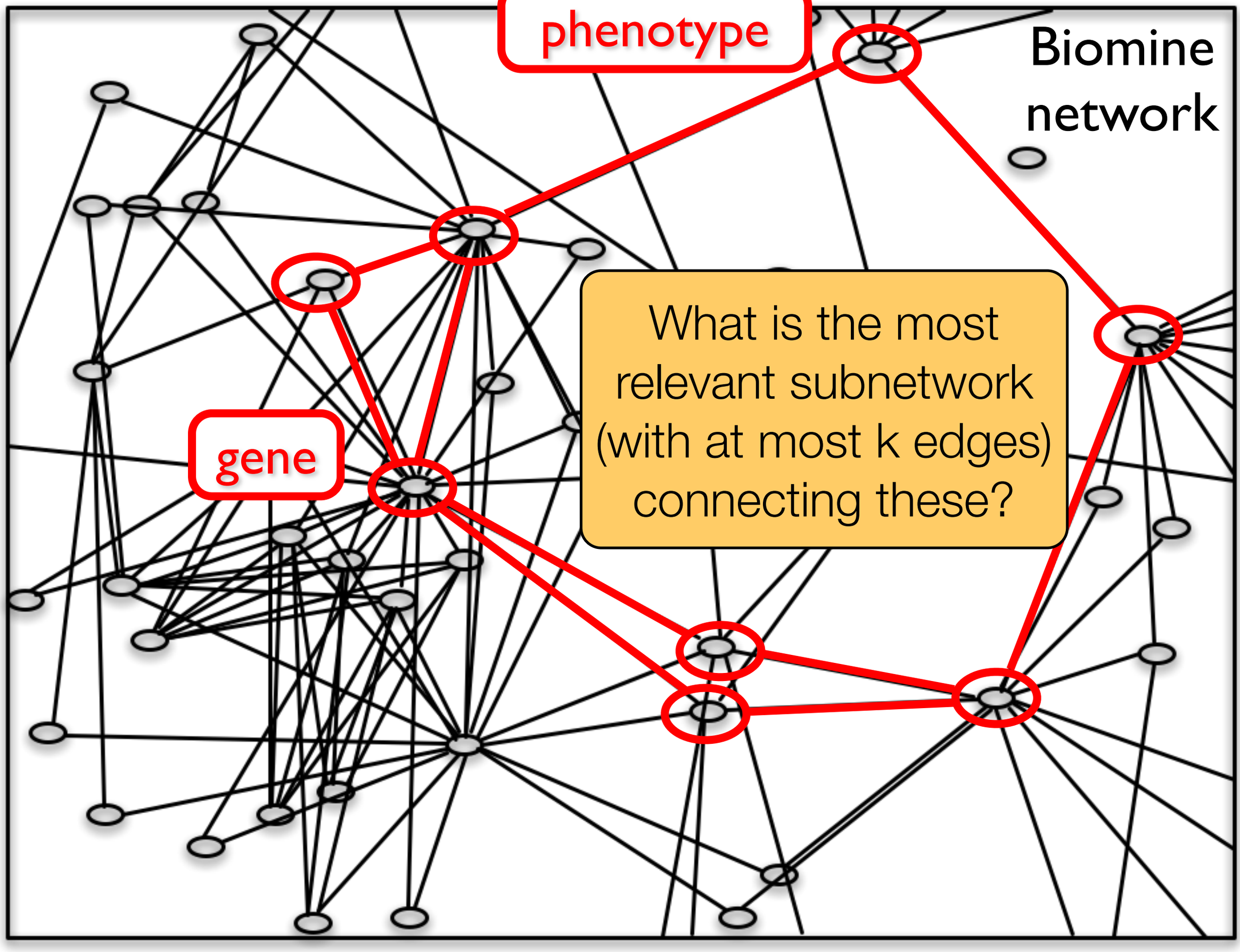


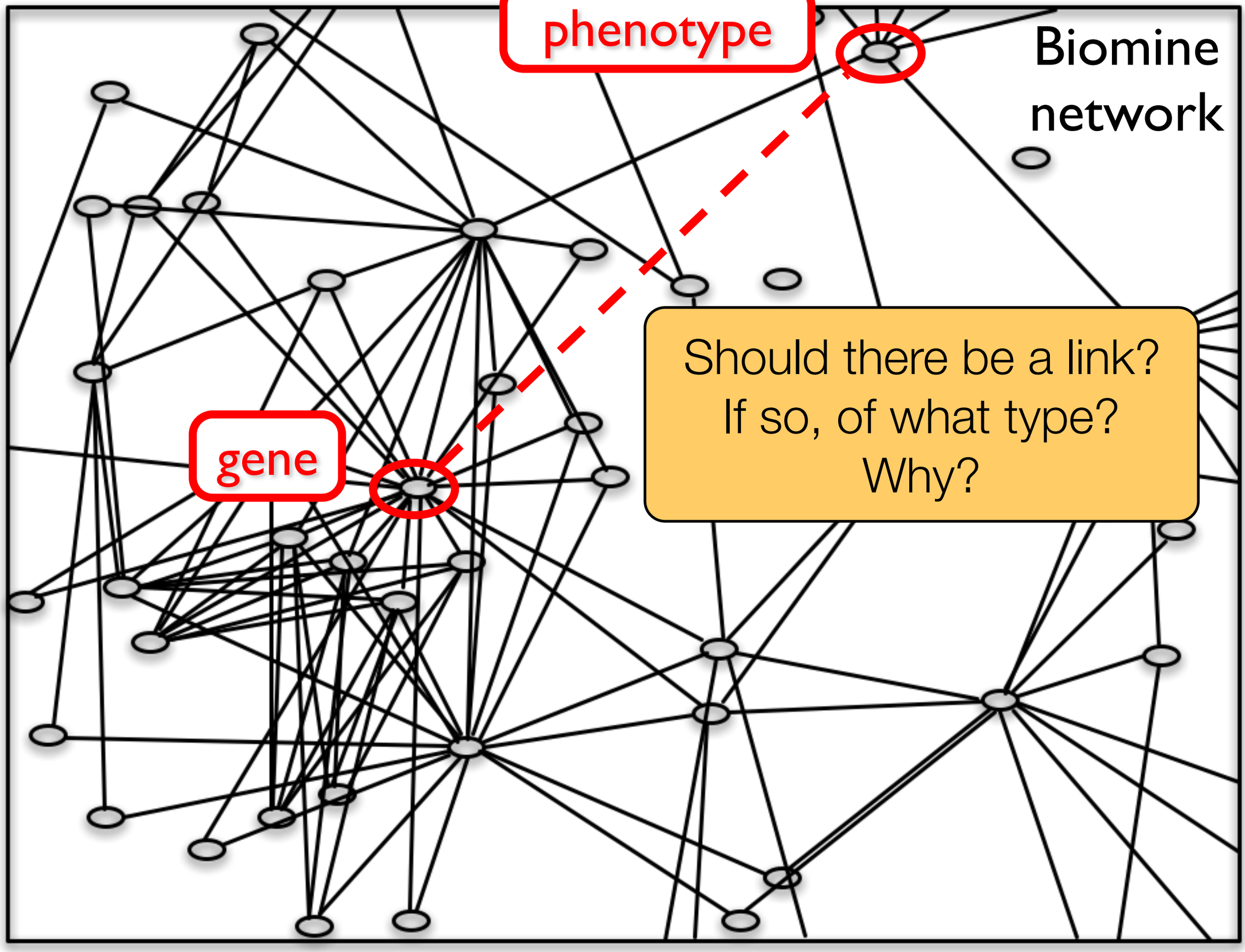
Biological Process

-participates_in
0.220

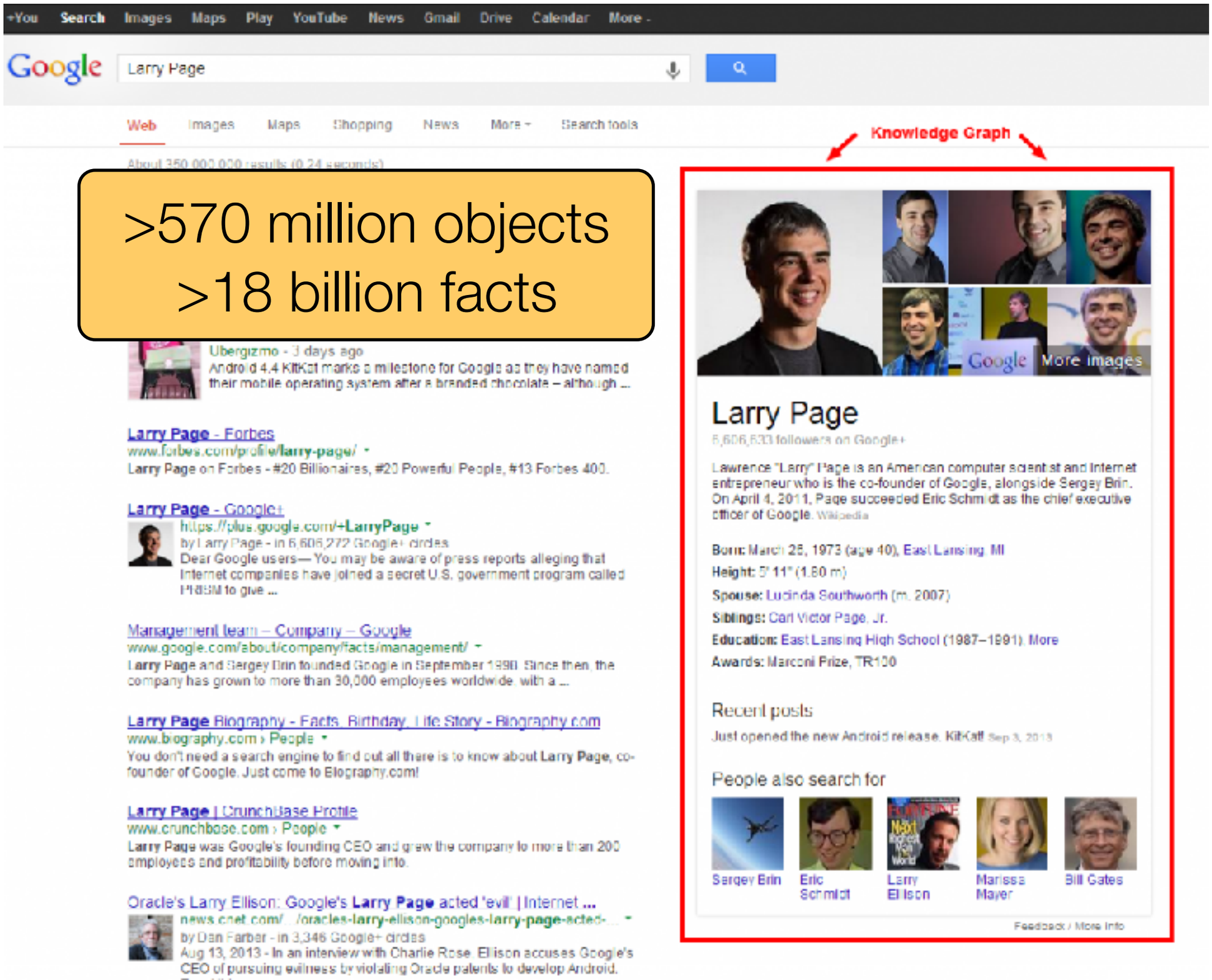
Gene

- different types of nodes & links
- automatically extracted from text, databases, ...
- probabilities quantifying source reliability, extractor confidence, ...
- similar in other contexts, e.g., NELL@CMU, linked open data, ...


phenotype
Biomine network
gene
What is the most relevant subnetwork (with at most k edges) connecting these?


phenotype
gene
Biomine network
Should there be a link?
If so, of what type?
Why?

Google Knowledge Graph



Example: Information Extraction

Recently-Learned Facts twitter Refresh

instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson is an automobile manufacturer	829	10-apr-2014	95.2
social_work is an academic program at the university rutgers_university	827	02-apr-2014	93.8
dante wrote the book the_divine_comedy	826	29-mar-2014	93.8
willie_aames was born in the city los_angeles	831	16-apr-2014	100.0
kitt_peak is a mountain in the state or province arizona	831	16-apr-2014	96.9
greenwich is a park in the city london	831	16-apr-2014	100.0

instances for many different relations

degree of certainty

NELL: http://rtw.ml.cmu.edu/rtw/

Why Statistical Relational Data?

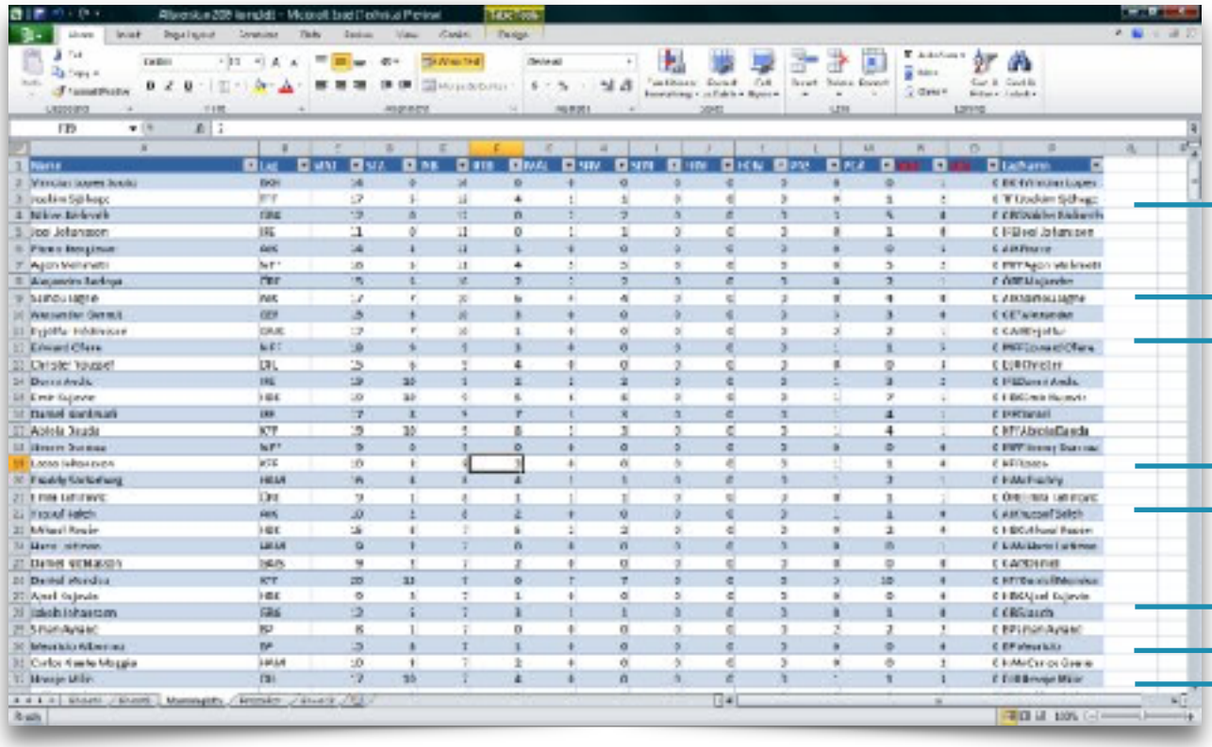
- Relational data is increasingly probabilistic
  - NELL machine reading (>50M facts)
  - Google Knowledge Vault (>2BN facts)
  - DeepDive (>7M facts)
  - …

- Data is inferred from unformatted information using statistical models
  - Learned from the web, large text corpora, ontologies, etc.
  - The learned/extracted data is relational

KU LEUVEN

One table is not enough

Traditional data format



Knowledge about world

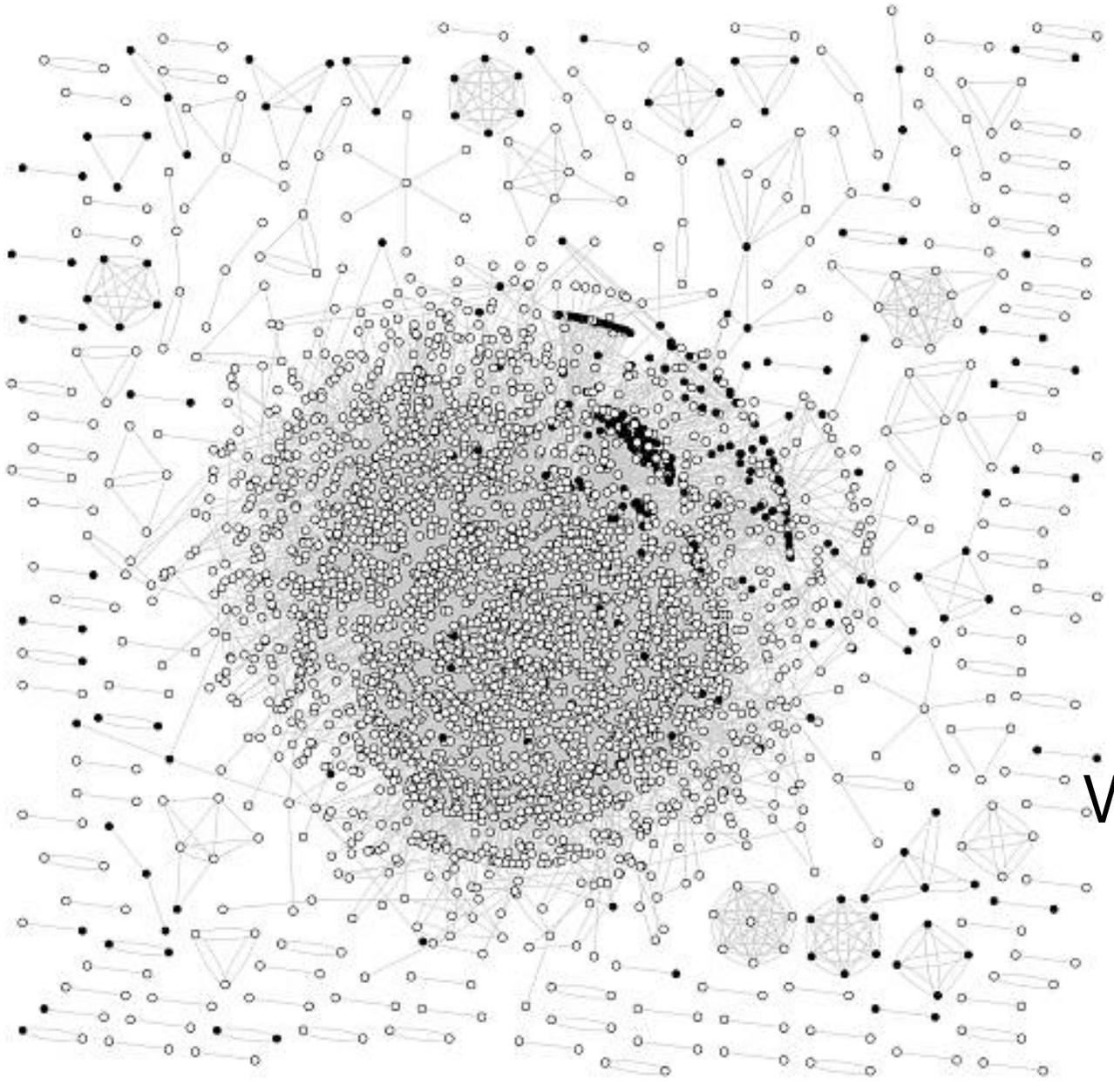
Teammate ..... *Relations*

Family ..... *Hierarchical*

Stands in between ..... *Relative position*

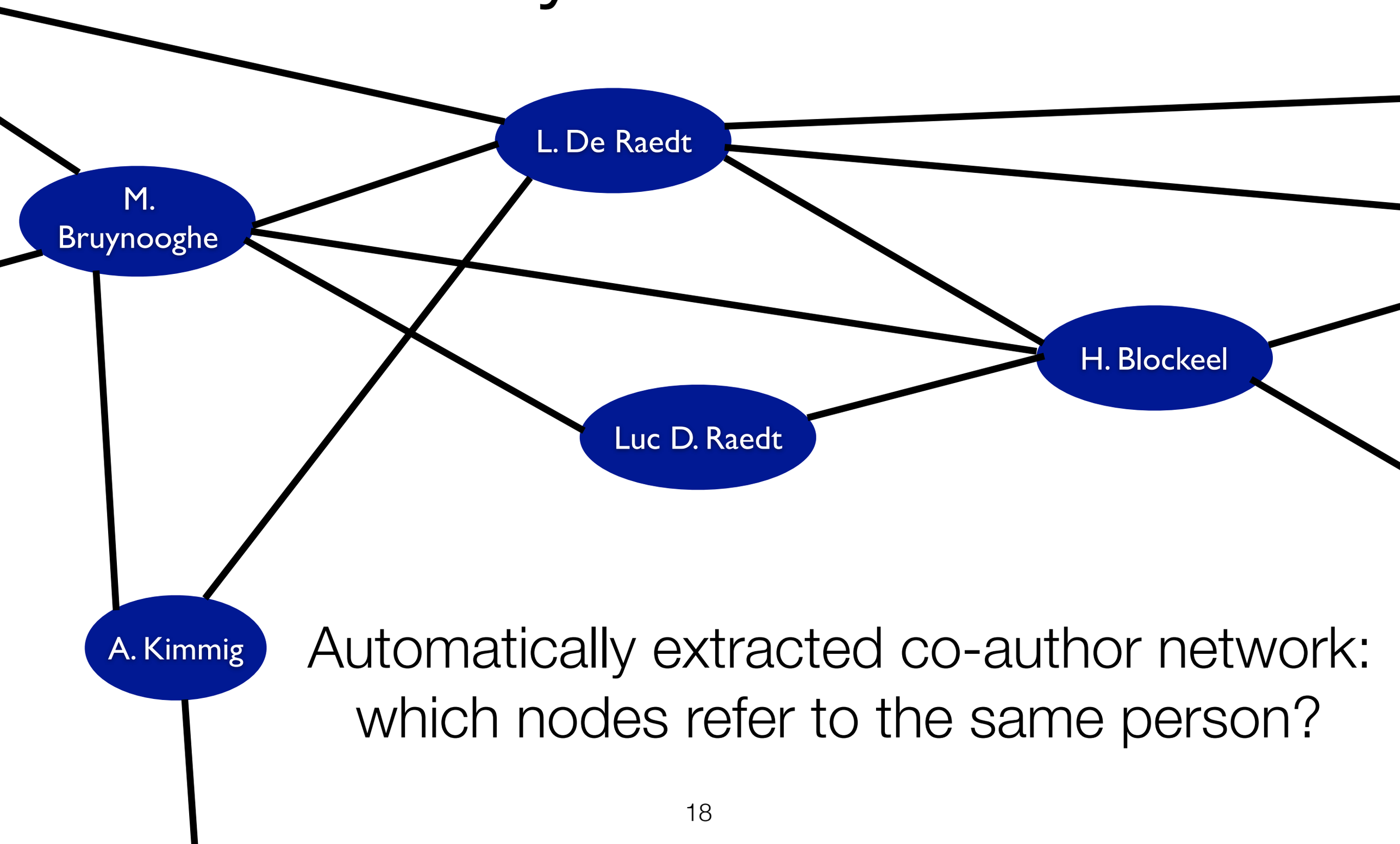
Previous training ..... *Temporal*

Node Classification



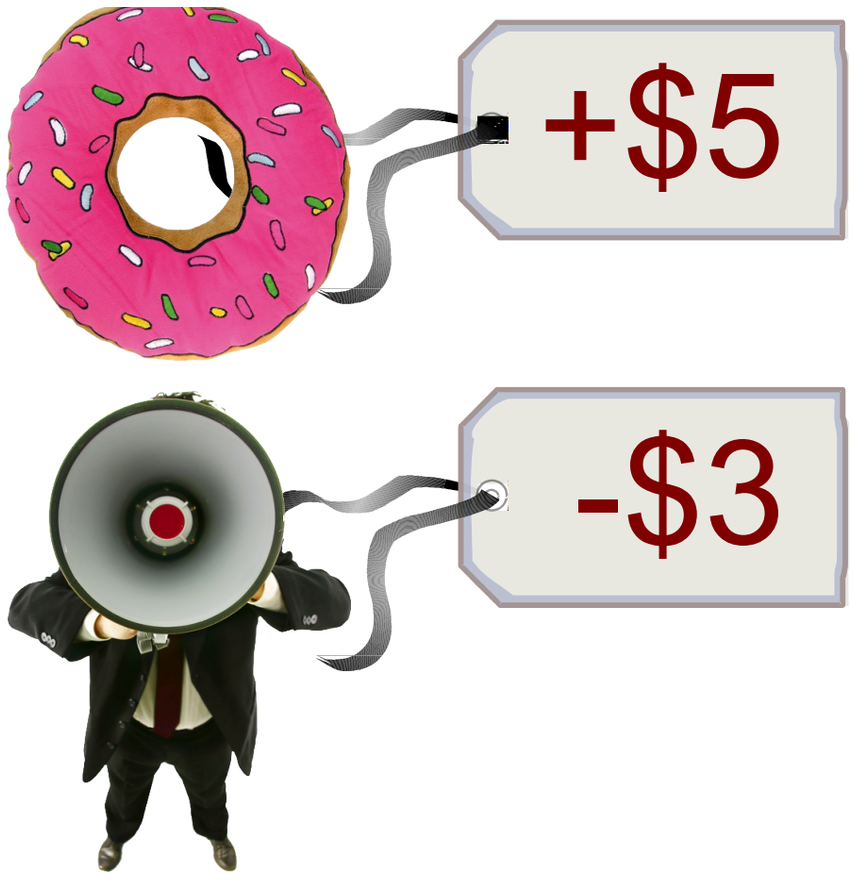
Can we predict the type of a node given information on its neighbors?
e.g., the type of a webpage given its links and the words on the page?

Entity Resolution

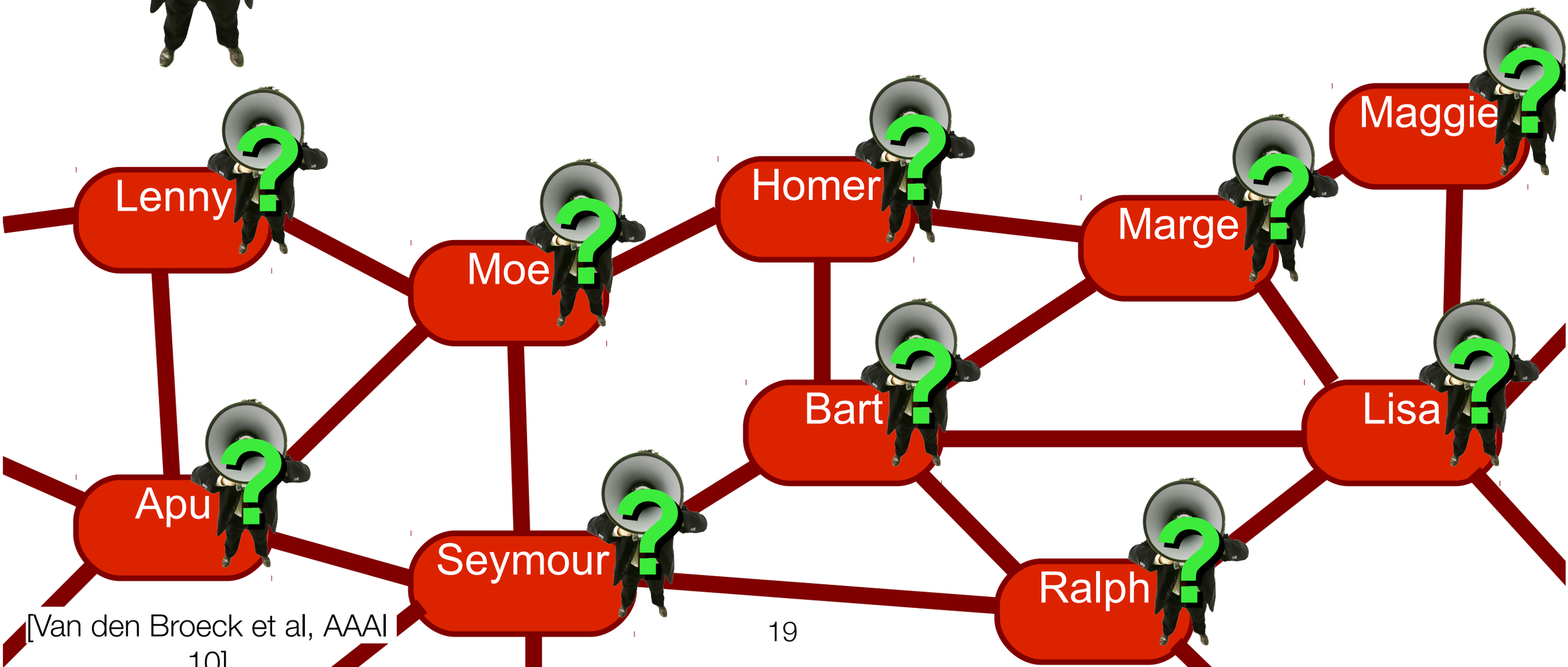


Automatically extracted co-author network: which nodes refer to the same person?

Viral Marketing

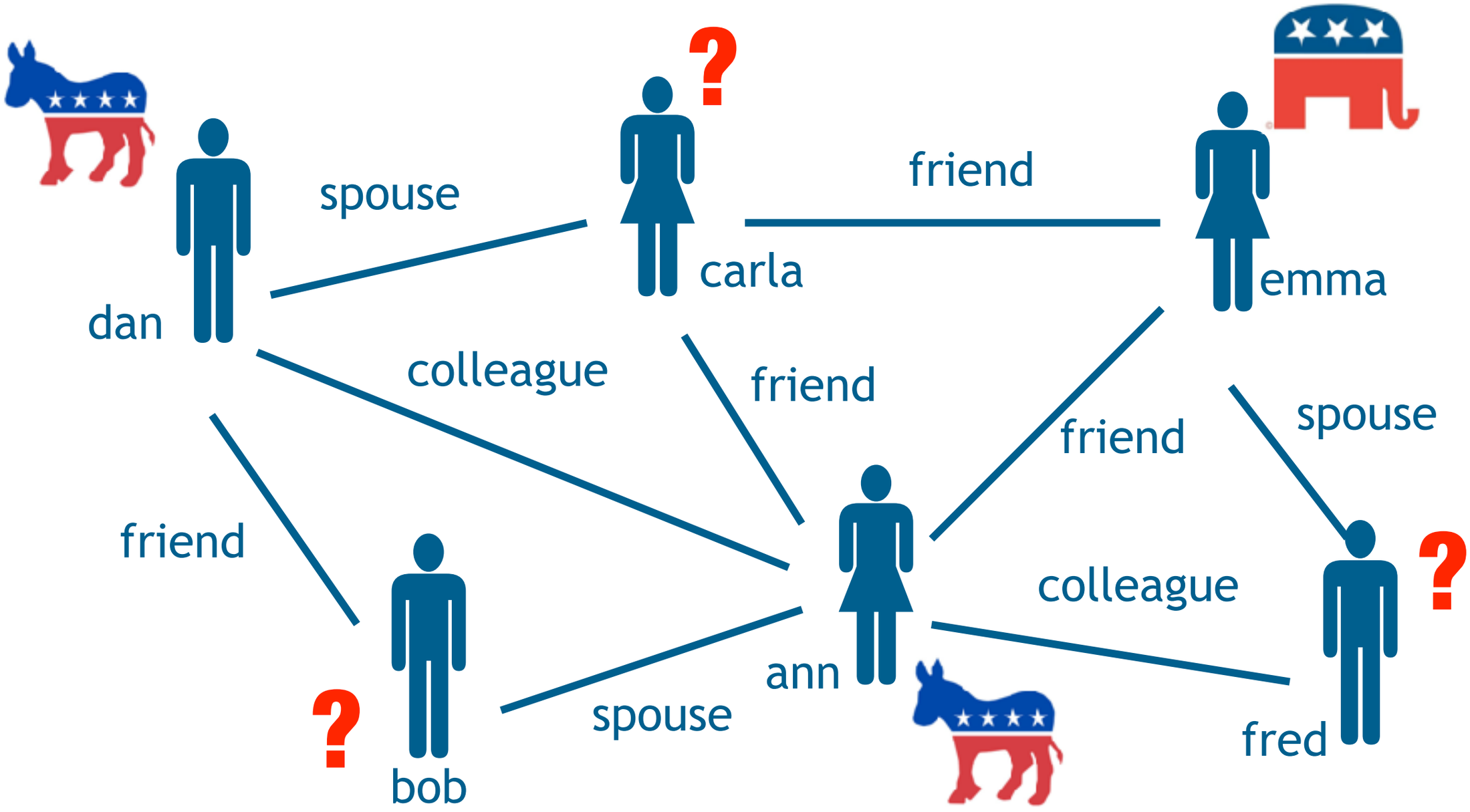


Which advertising strategy maximizes expected profit?



[Van den Broeck et al, AAAI 10]

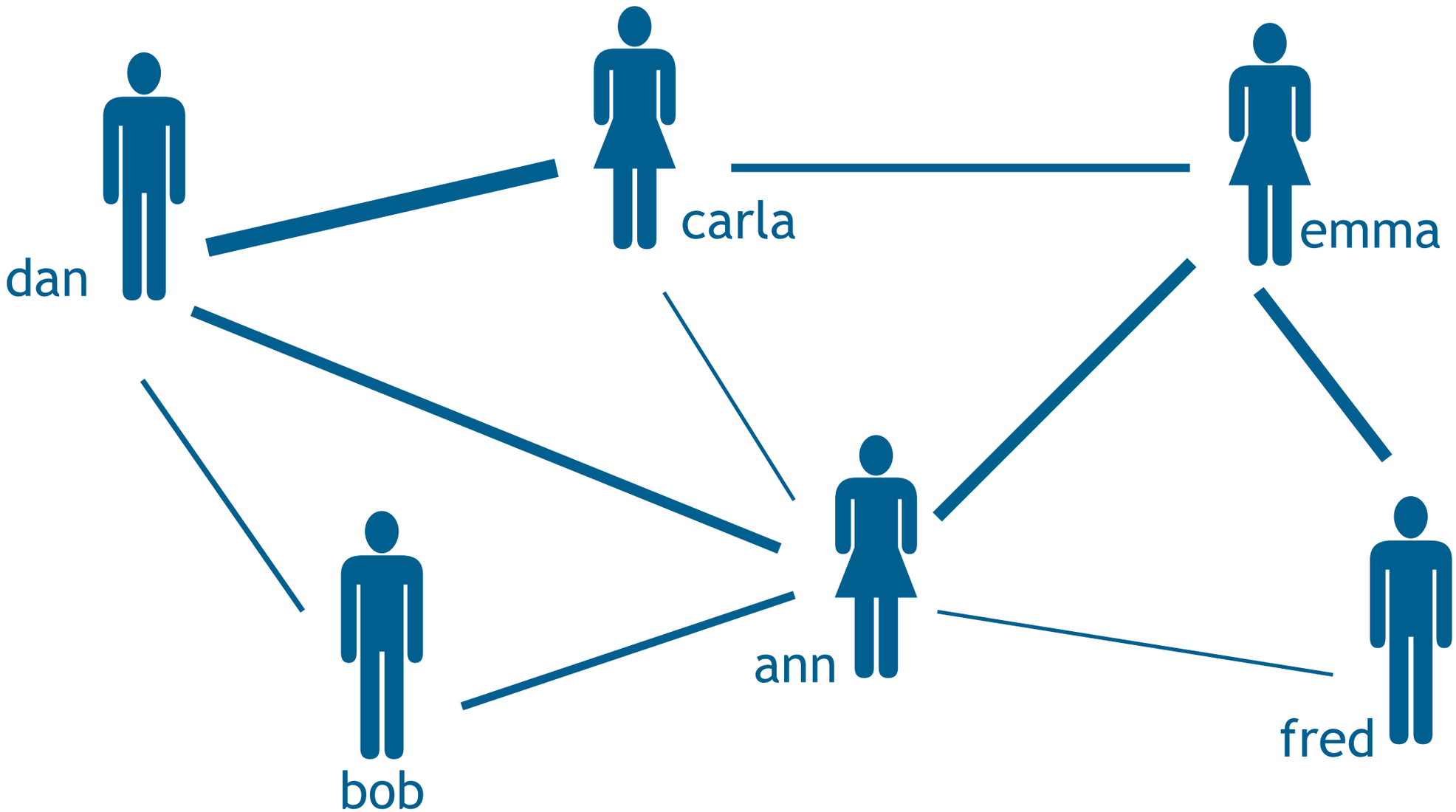
Voter Opinion Modeling



Can we predict preferences?

[Bach et al, NIPS 12]

Social Trust



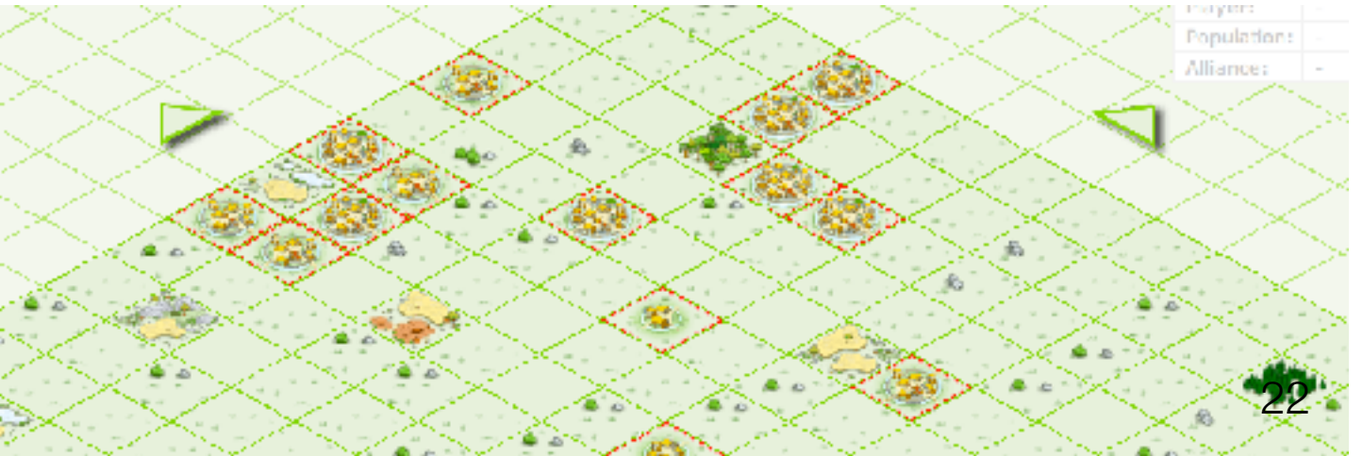
Can we predict strength of ties?

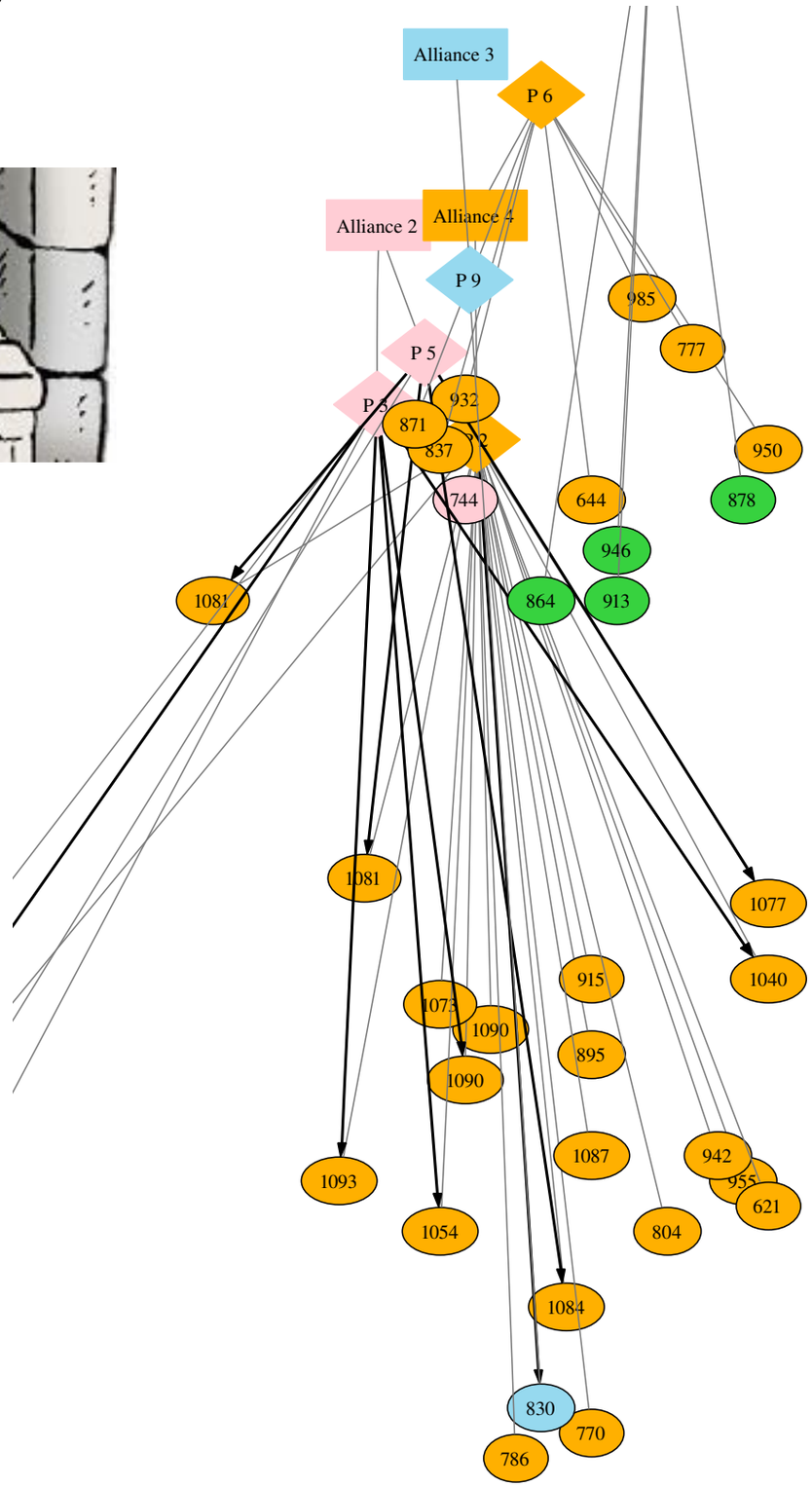
[Huang et al, SBP 13]

Dynamic networks

Travian: A massively multiplayer real-time strategy game

Can we build a model
of this world ?
Can we use it for playing
better ?



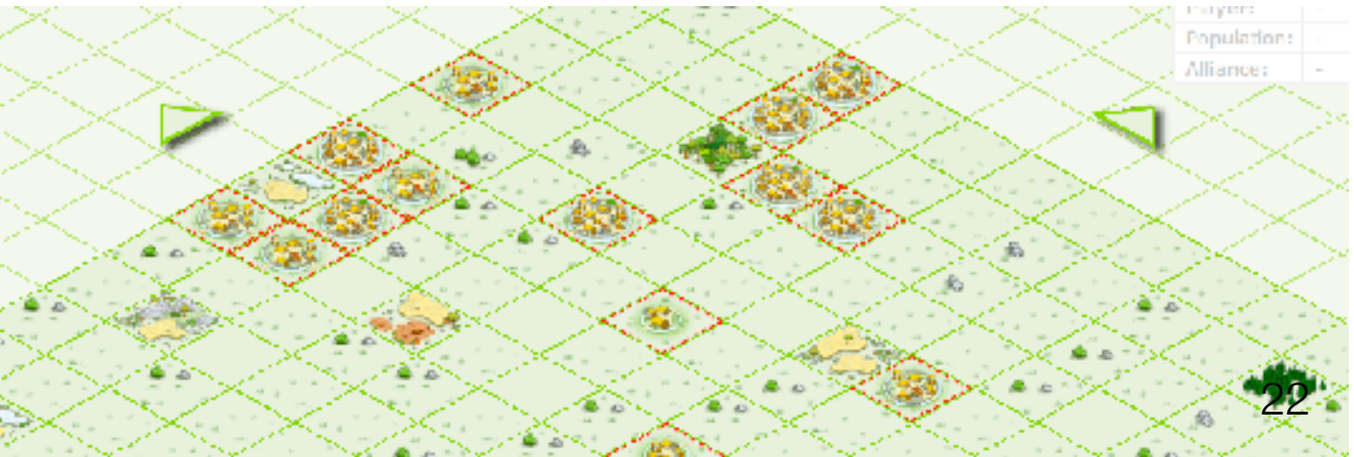


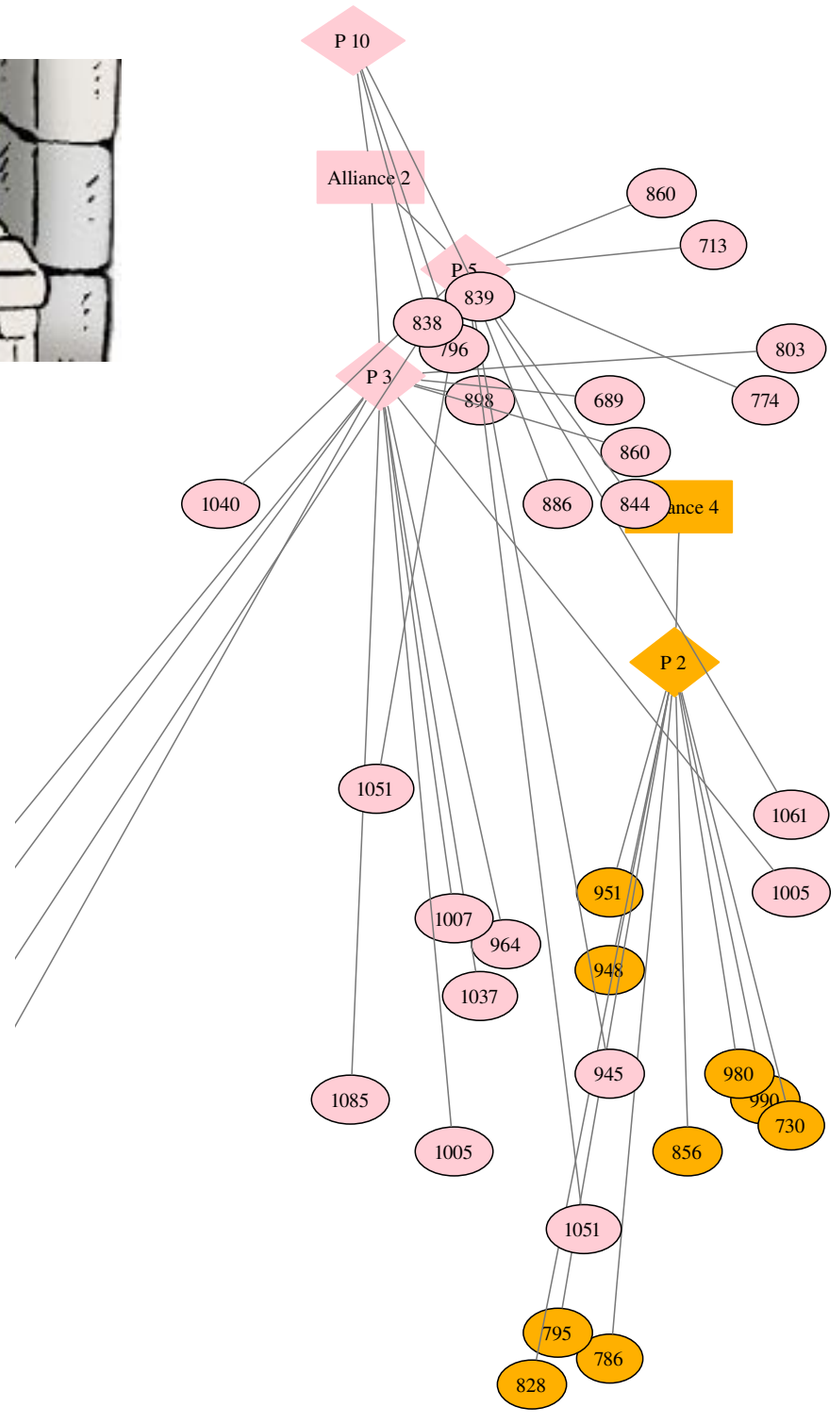
[Thon et al, MLJ 11]

Dynamic networks

Travian: A massively multiplayer real-time strategy game

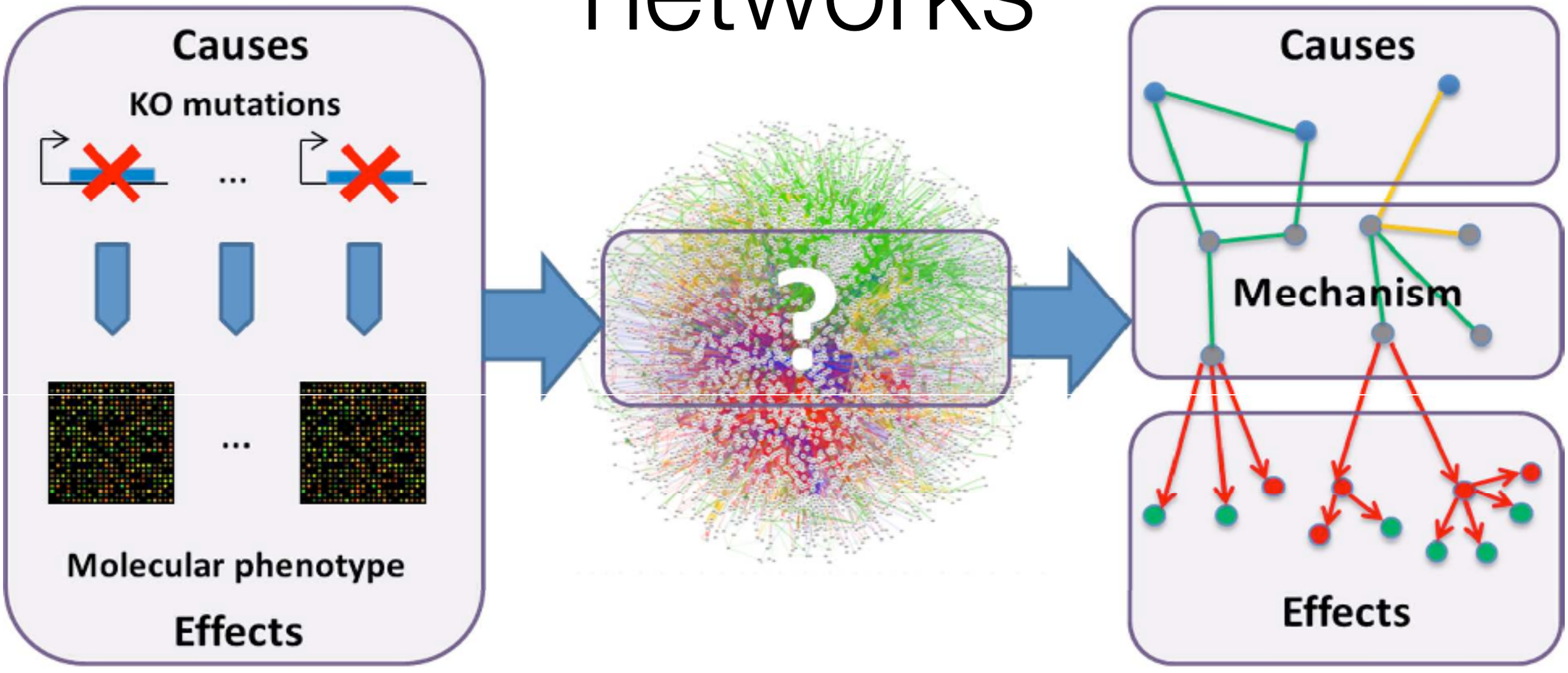
Can we build a model
of this world ?
Can we use it for playing
better ?





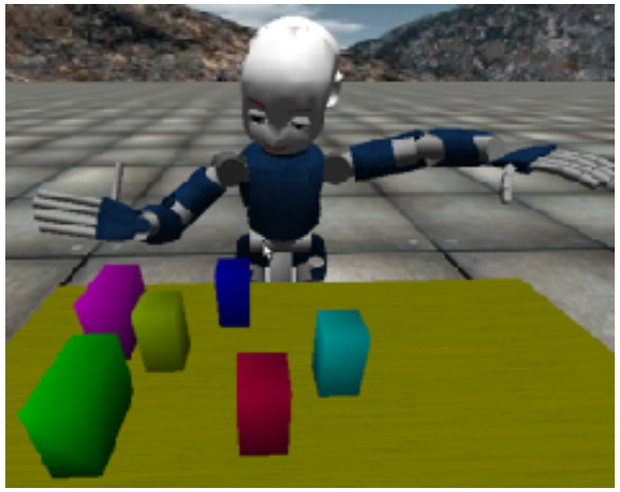
[Thon et al, MLJ 11]

Molecular interaction networks

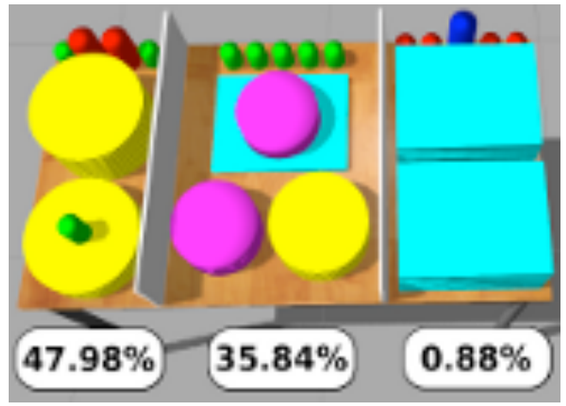


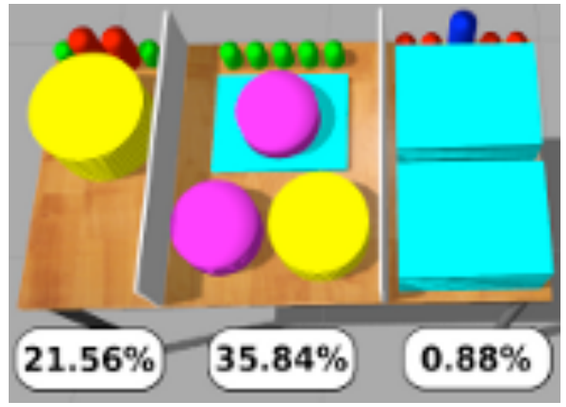
Can we find the mechanism connecting causes to effects?

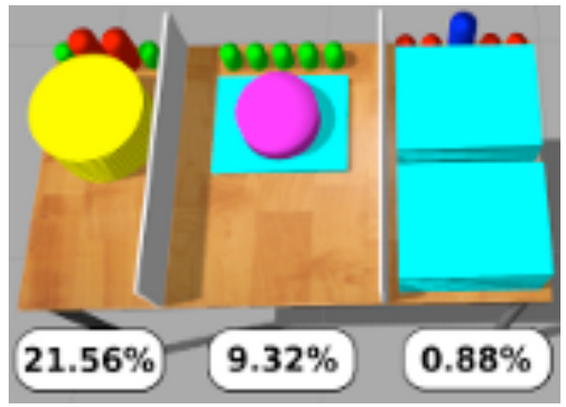
[De Maeyer at al, Molecular Biosystems 13]

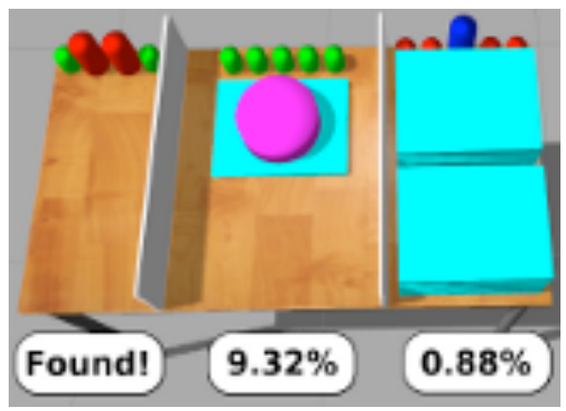


Robotics





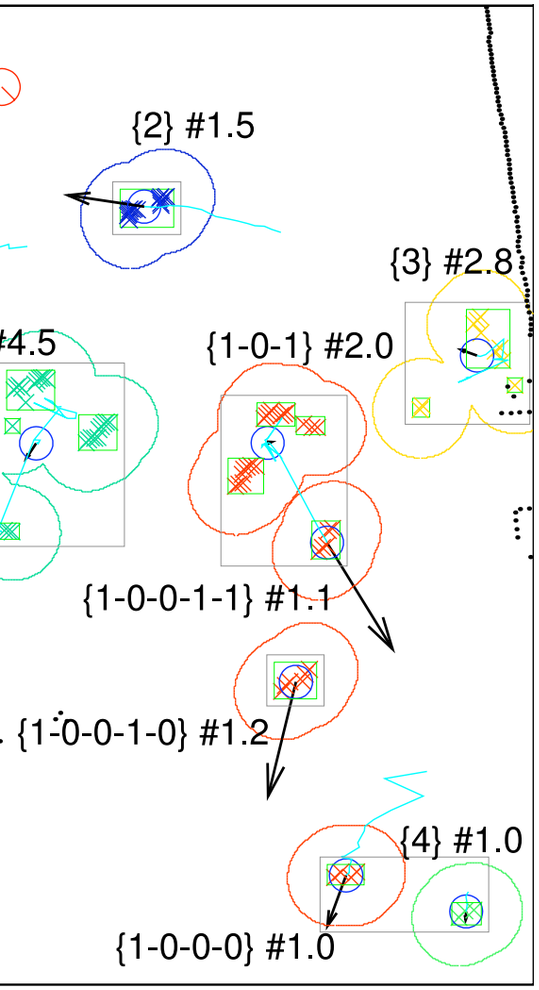




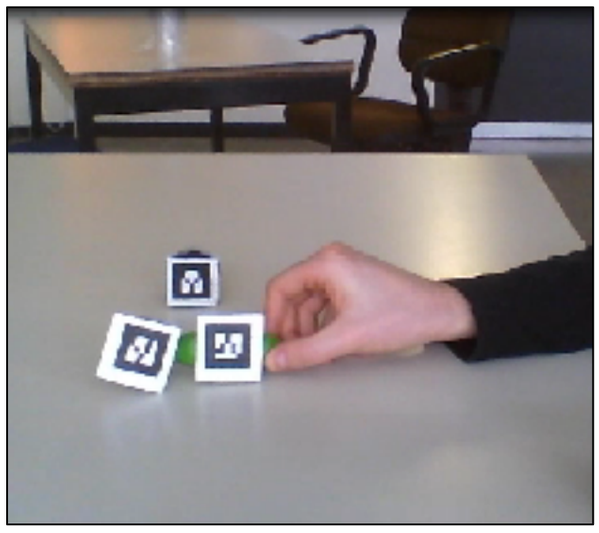
- How to achieve a specific configuration of objects on the shelf?
- Where's the orange mug?
- Where's something to serve soup in?

[Moldovan et al, ICRA 12, 14]

Analyzing Video Data



- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?



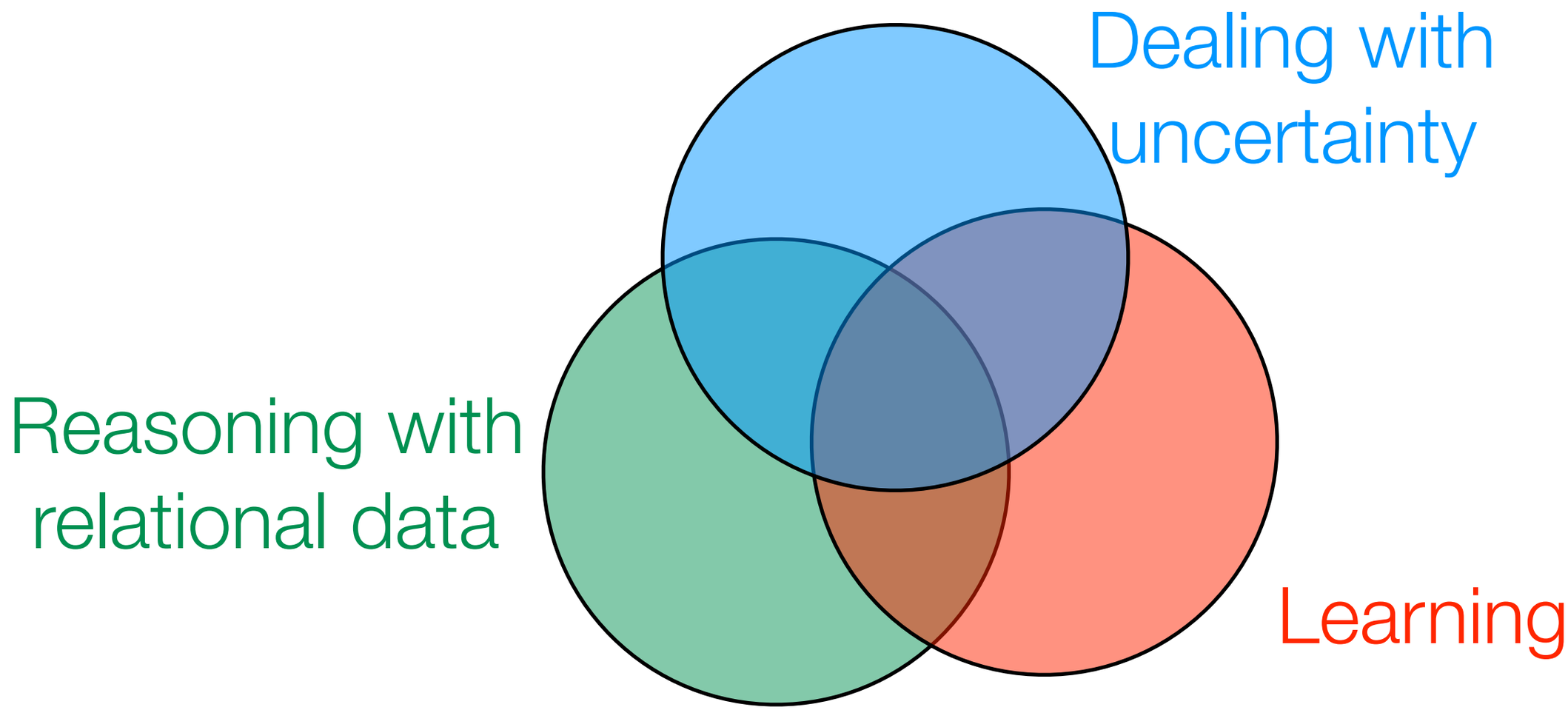
ProbLog

- Semantics by example
- Strategies for inference

KU LEUVEN

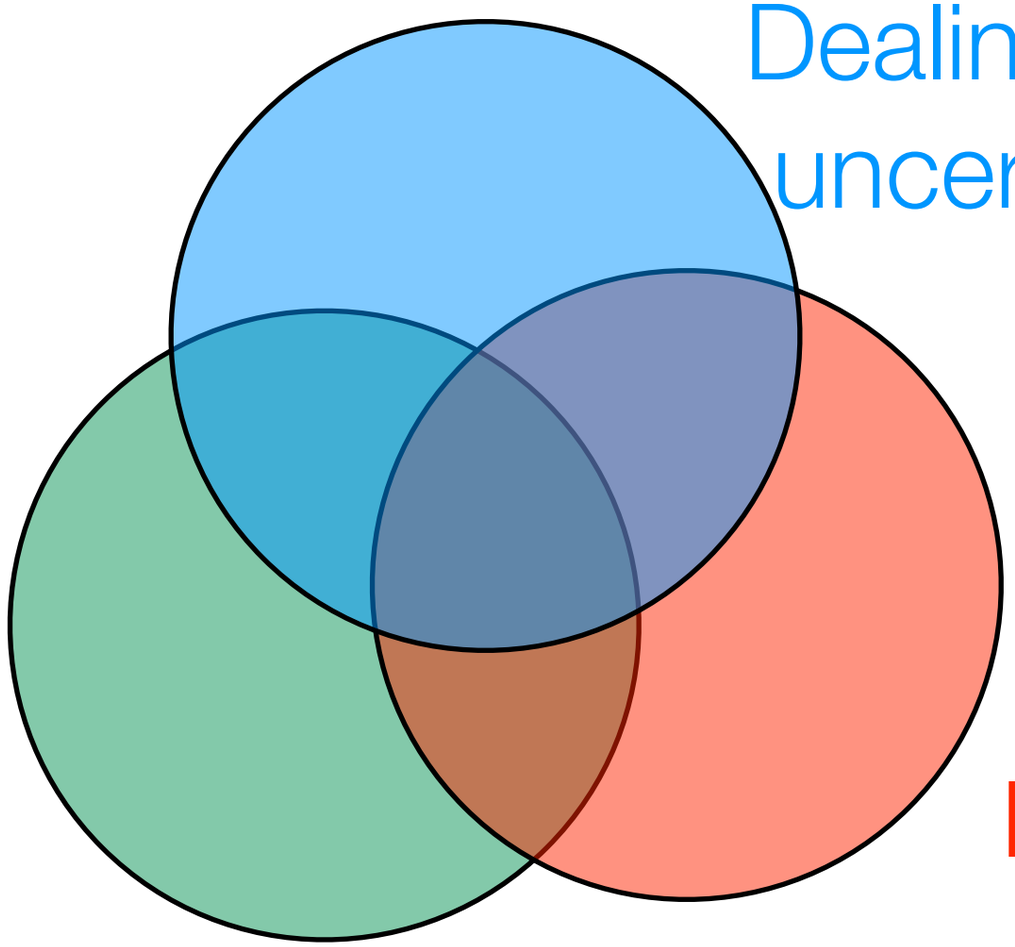
ProbLog

probabilistic Prolog



ProbLog

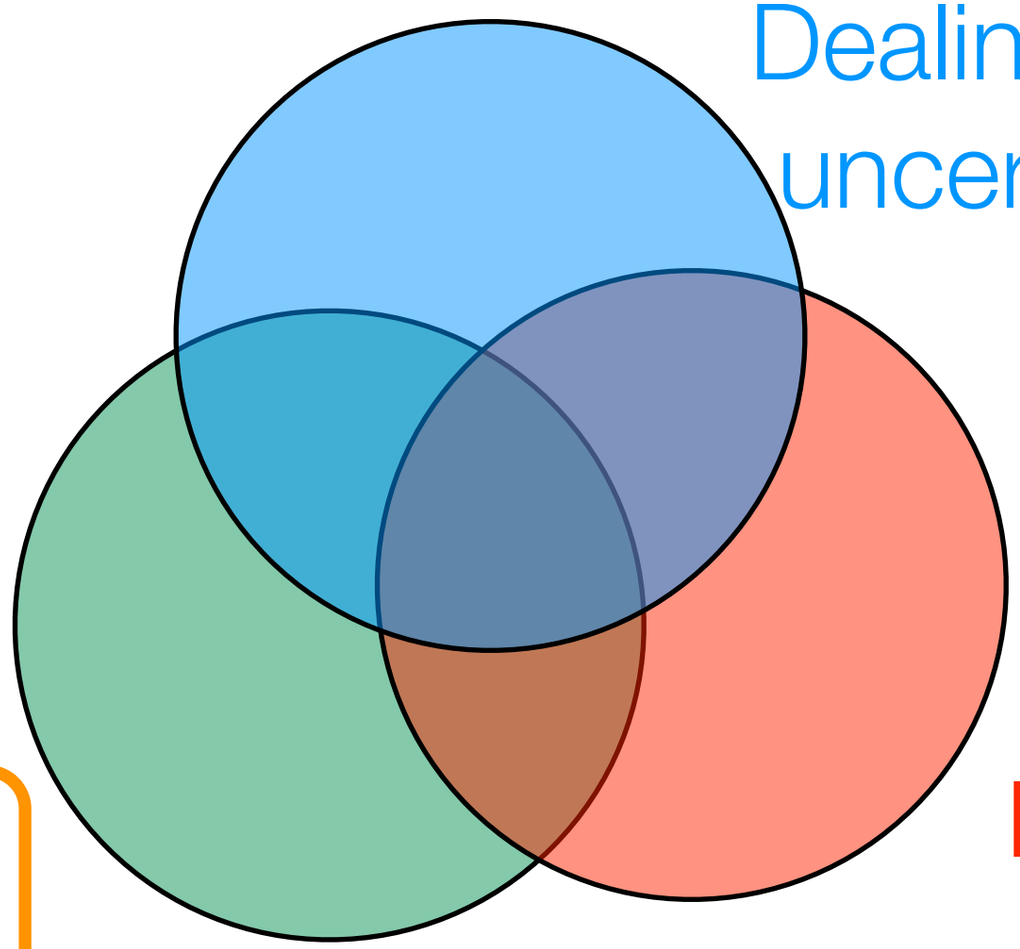
probabilistic Prolog




```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

ProbLog

probabilistic Prolog




```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

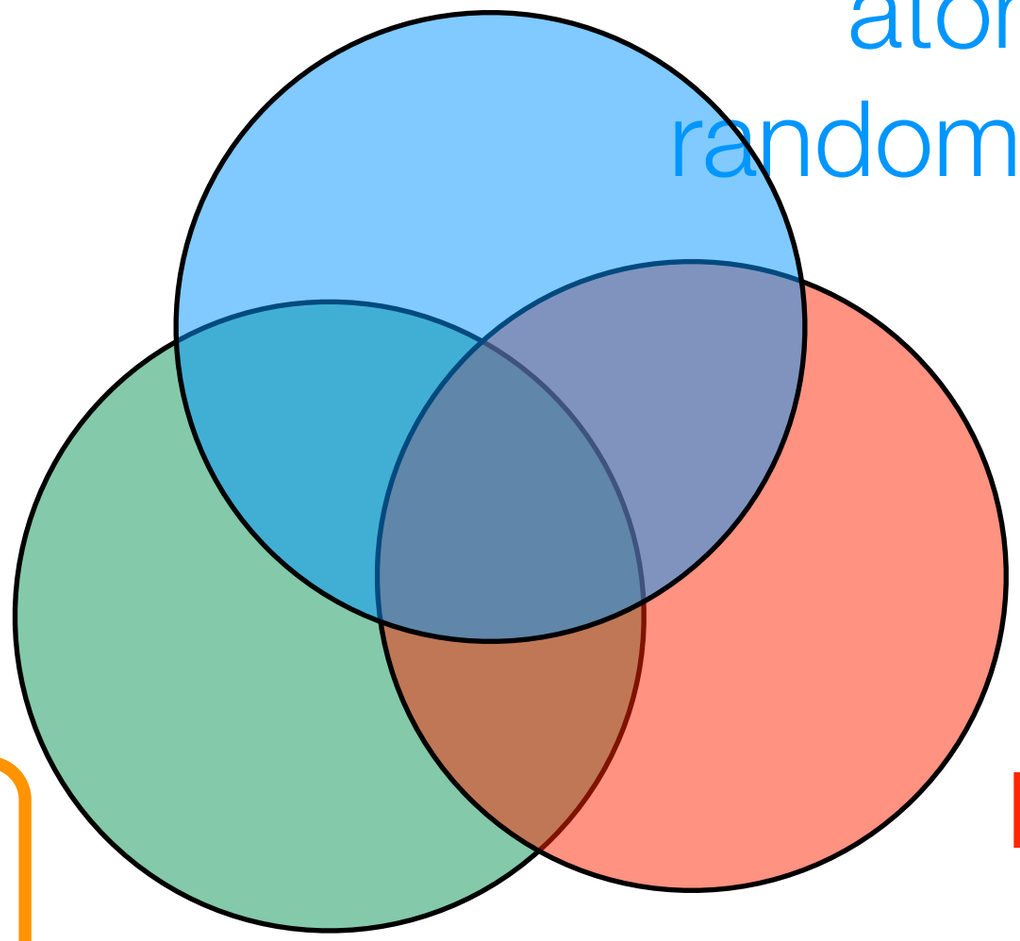
```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

ProbLog

probabilistic Prolog

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as
random variables



Prolog / logic
programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

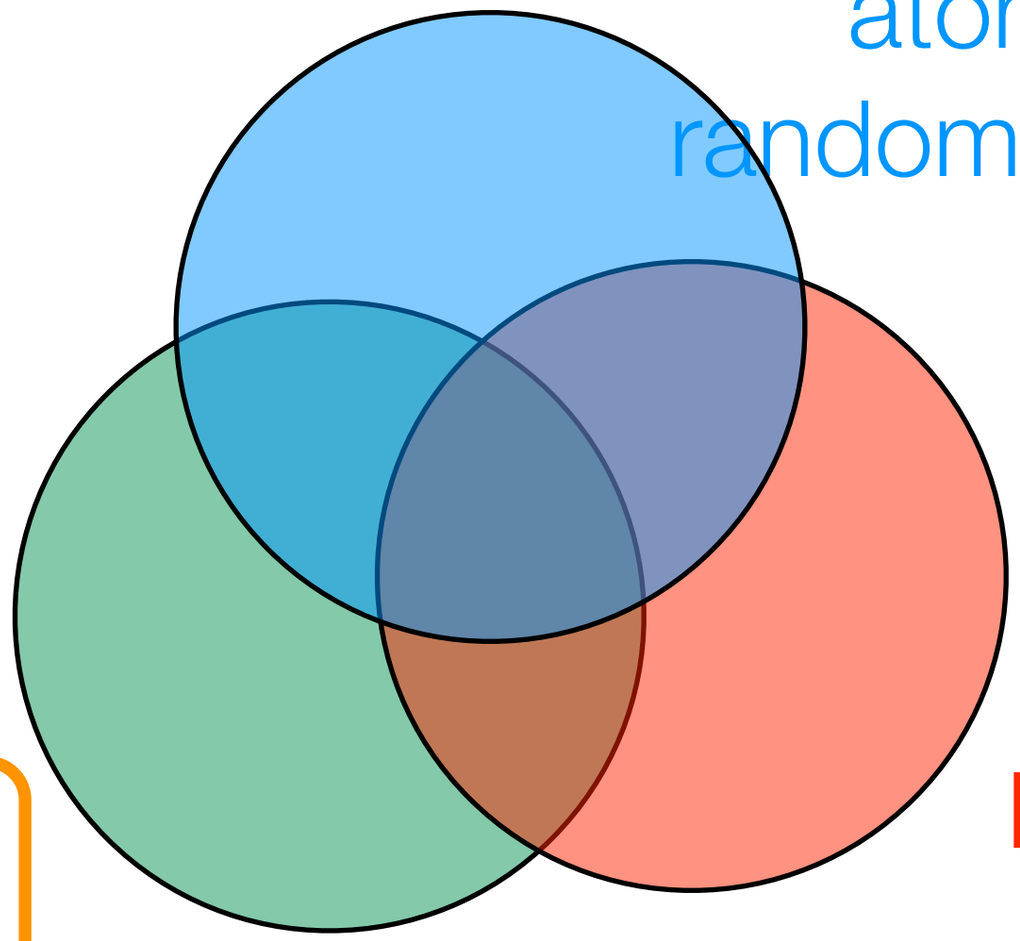
ProbLog

probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as
random variables



Prolog / logic
programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

ProbLog

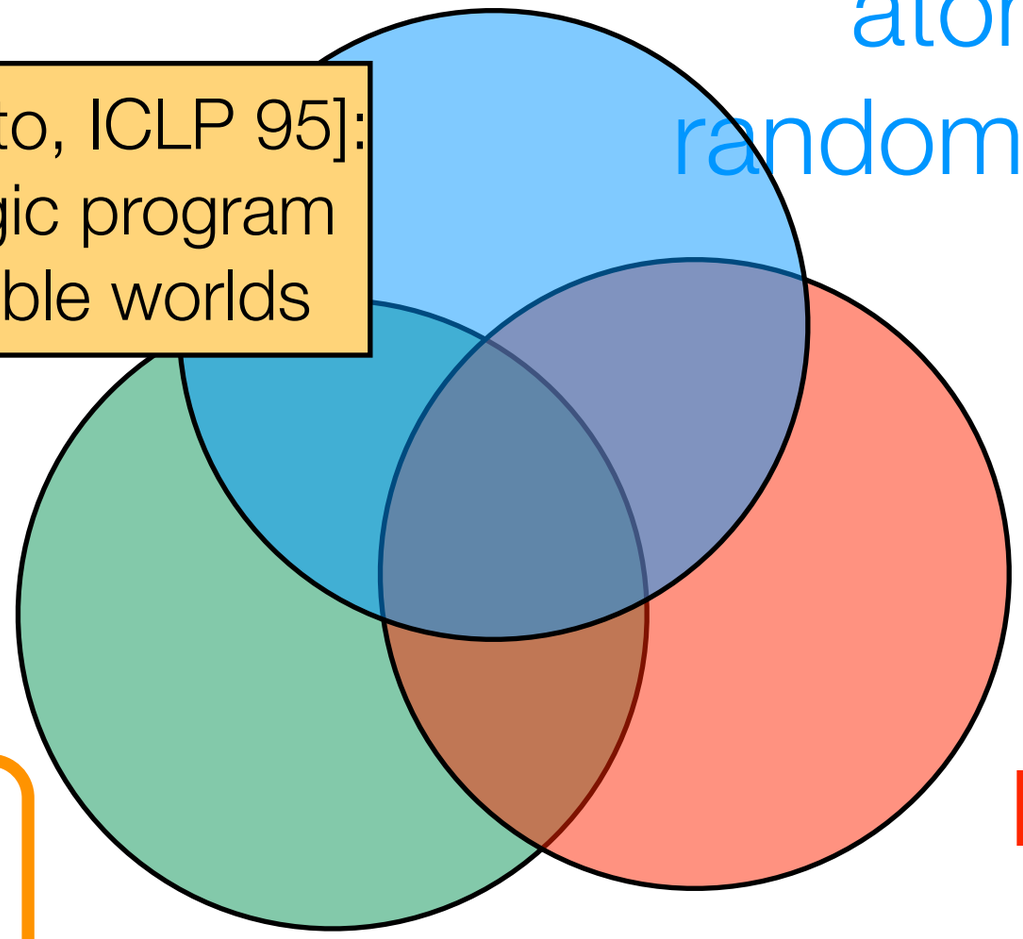
probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Prolog / logic programming

Learning

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

ProbLog

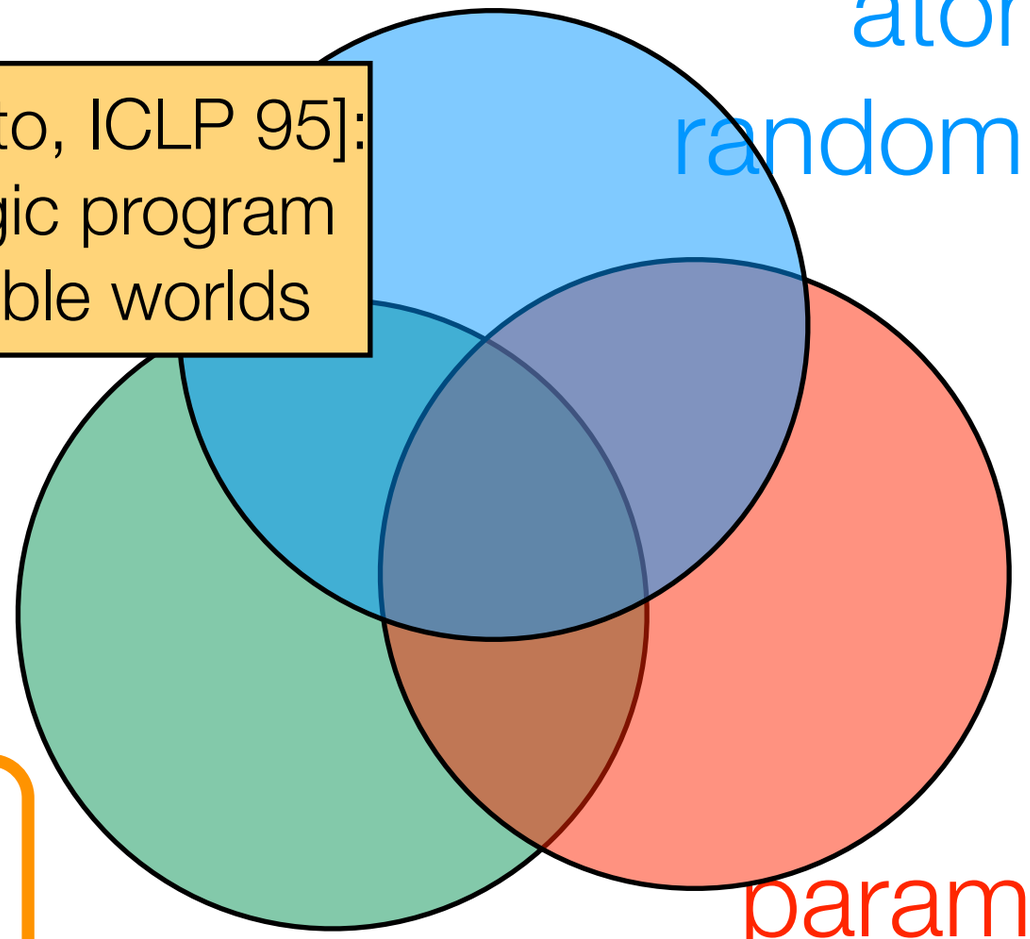
probabilistic Prolog

several possible worlds

```
0.8::stress(ann).
0.6::influences(ann,bob).
0.2::influences(bob,carl).
```

atoms as random variables

Distribution Semantics [Sato, ICLP 95]:
probabilistic choices + logic program
→ distribution over possible worlds



Prolog / logic programming

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

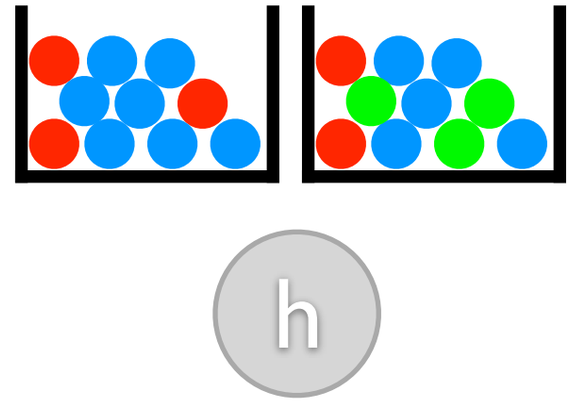
one world

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

parameter learning, adapted relational learning techniques

ProbLog by example:

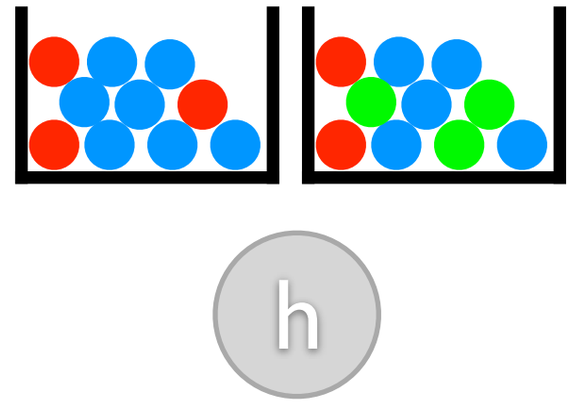
A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

ProbLog by example:

A bit of gambling



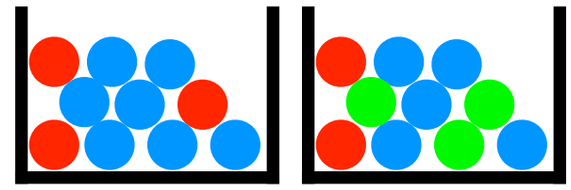
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.
```

probabilistic fact: heads is true with probability 0.4 (and false with 0.6)

ProbLog by example:

A bit of gambling



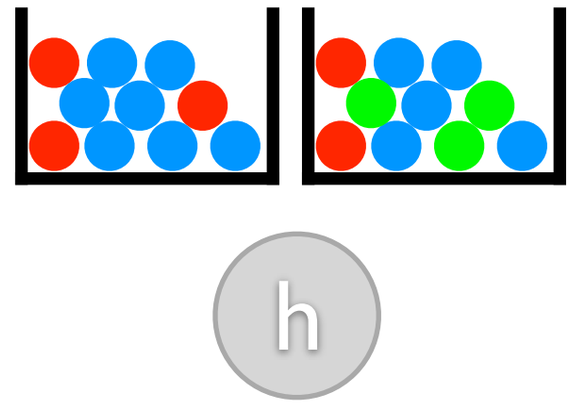
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
```

annotated disjunction: first ball is red with probability 0.3 and blue with 0.7

ProbLog by example:

A bit of gambling



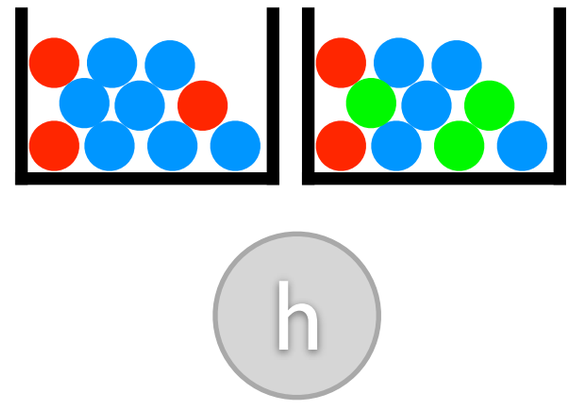
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).
```

annotated disjunction: second ball is red with probability 0.2, green with 0.3, and blue with 0.5

ProbLog by example:

A bit of gambling



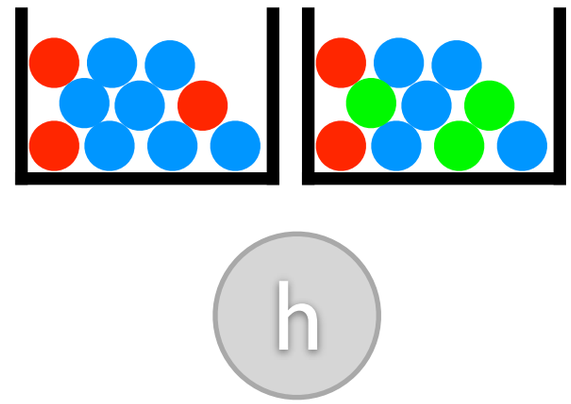
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).

win :- heads, col(_,red).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



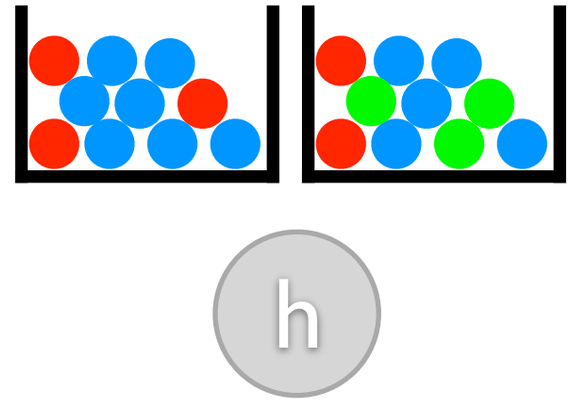
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green);
                   0.5 :: col(2,blue).
```

probabilistic choices

```
win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

consequences

Questions

```
0.4::heads.
0.3::col(1,red); 0.7::col(1,blue).
0.2::col(2,red); 0.3::col(2,green); 0.5::col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

- Probability of win? *Marginal probability*
  query
- Probability of win given col(2,green)? *Conditional probability*
  evidence
- Most probable world where win is true? *MPE inference*

KU LEUVEN

Pythonic representation

ProbLog

```
0.4::heads.
0.3::col(1,red); 0.7::col(1,blue).
0.2::col(2,red); 0.3::col(2,green); 0.5::col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

Python

How would you implement memchoice?

```
def heads():
      return memchoice(0.4)
```

Stochastic variables

```
colors = {
      1: {‘red’: 0.3, ‘blue’: 0.7}
      2: {‘red’: 0.2, ‘green’: 0.3, ‘blue’: 0.5}
}

def color(u):
      return memchoice(colors[u])

def win():
      return heads() and any([color[u] == ‘red’ for u in colors.keys()]) or \
            color(1) == color(2)
```

KU LEUVEN

Possible Worlds

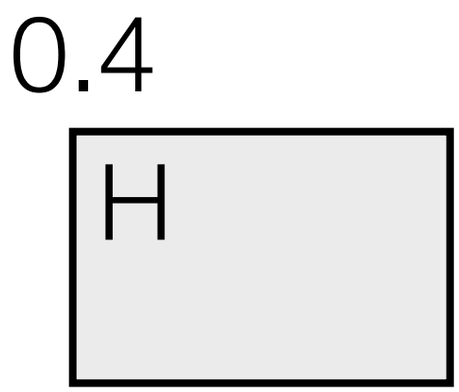
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

Possible Worlds

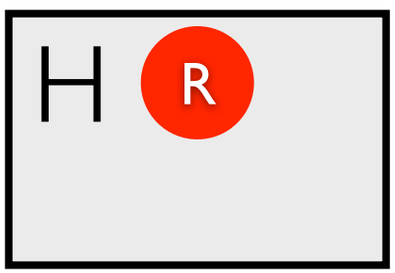
```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

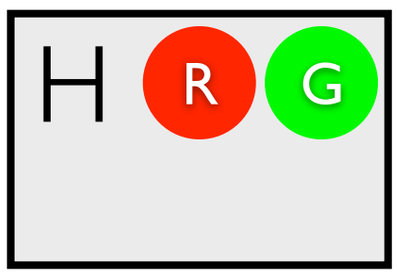
0.4×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

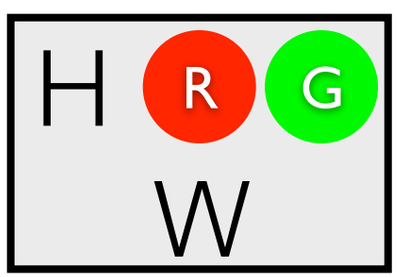
0.4×0.3×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

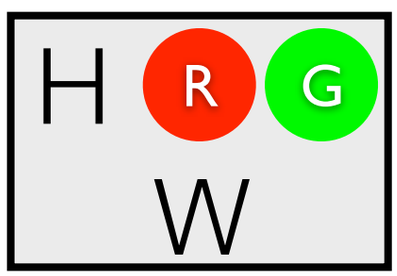
0.4×0.3×0.3



Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

0.4×0.3×0.3

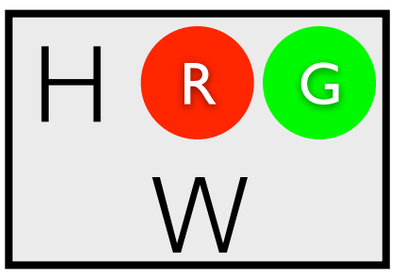


(1–0.4)

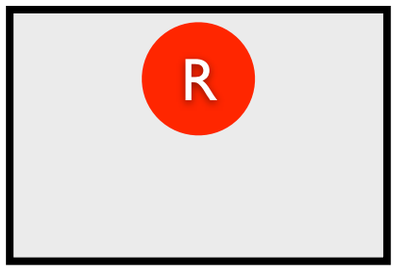
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4×0.3×0.3



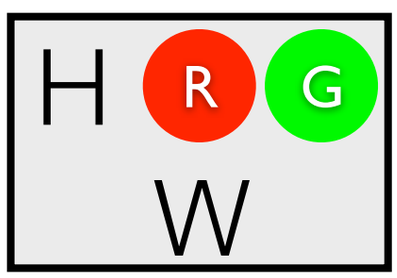
(1–0.4)×0.3



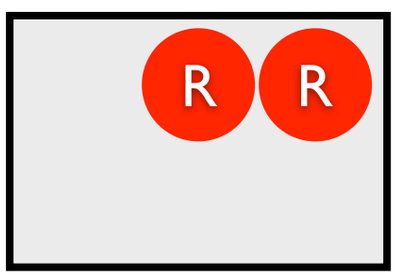
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4×0.3×0.3



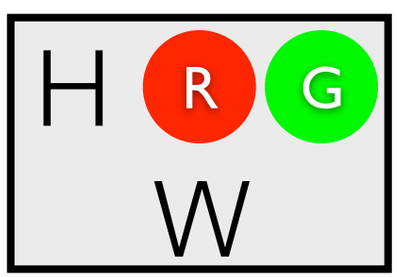
(1–0.4)×0.3×0.2



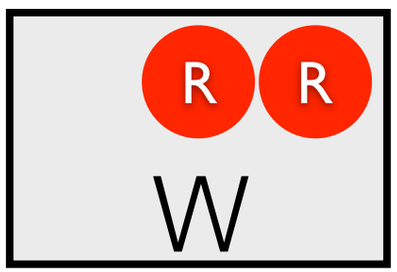
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

0.4×0.3×0.3



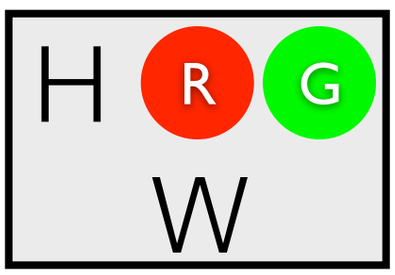
(1–0.4)×0.3×0.2



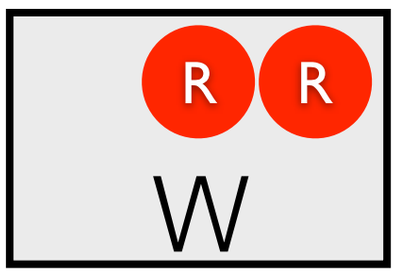
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

0.4×0.3×0.3



(1–0.4)×0.3×0.2

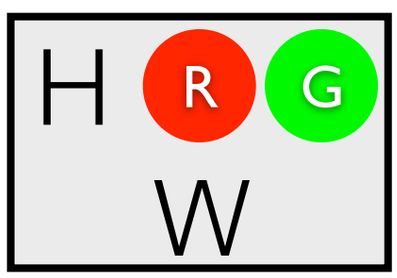


(1–0.4)

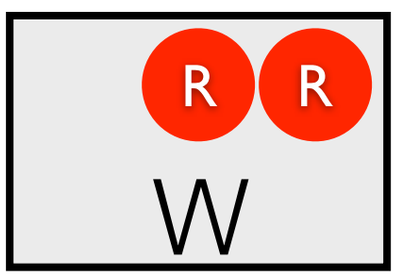
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

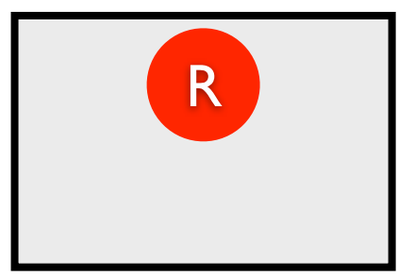
0.4×0.3×0.3



(1–0.4)×0.3×0.2



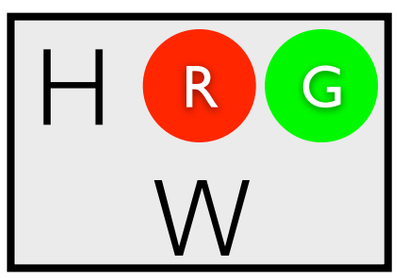
(1–0.4)×0.3



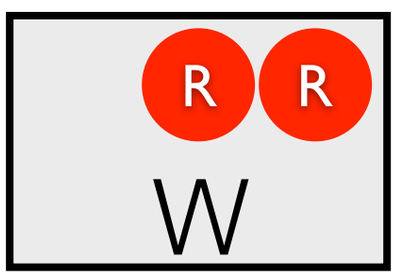
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

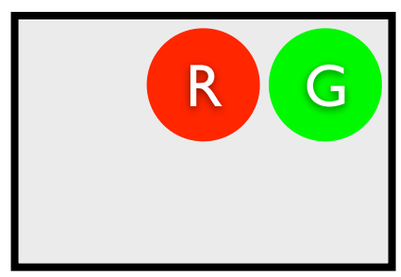
0.4×0.3×0.3



(1−0.4)×0.3×0.2



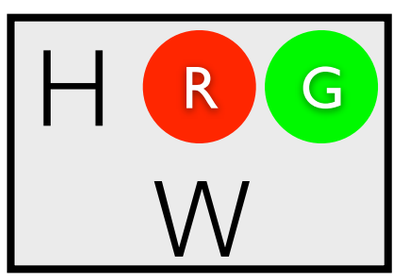
(1−0.4)×0.3×0.3



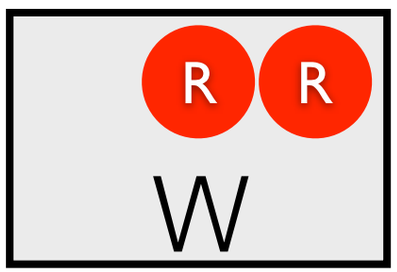
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col(_,red).
win :- col(1,C), col(2,C).
```

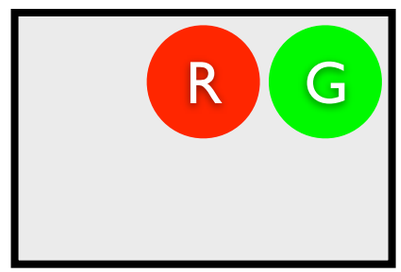
0.4×0.3×0.3



(1–0.4)×0.3×0.2



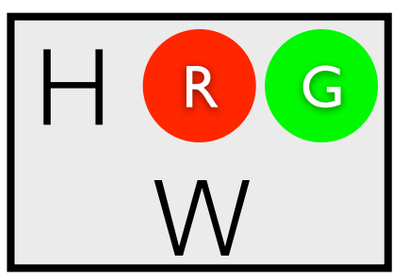
(1–0.4)×0.3×0.3



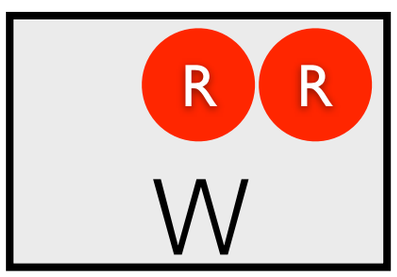
Possible Worlds

```
0.4 :: heads.

0.3 :: col(1,red); 0.7 :: col(1,blue)..
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

win :- heads, col( ,red).
win :- col(1,C), col(2,C).
```

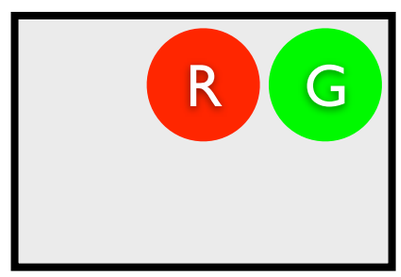
0.4×0.3×0.3



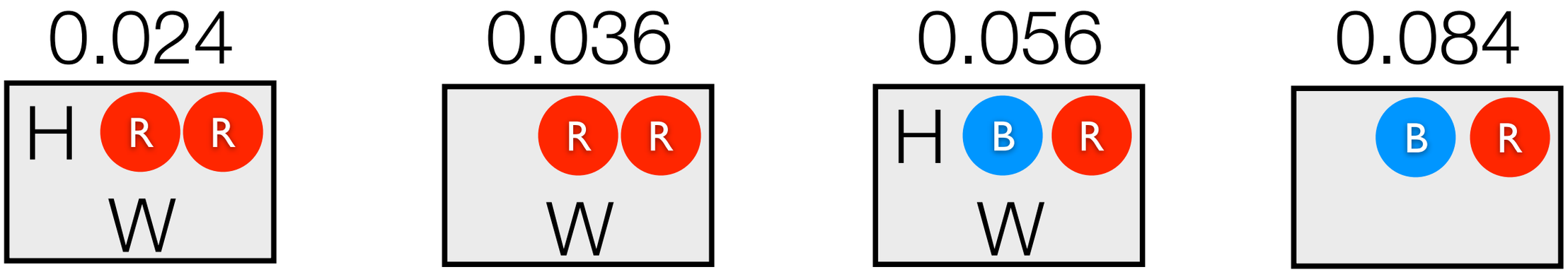
(1–0.4)×0.3×0.2



(1–0.4)×0.3×0.3

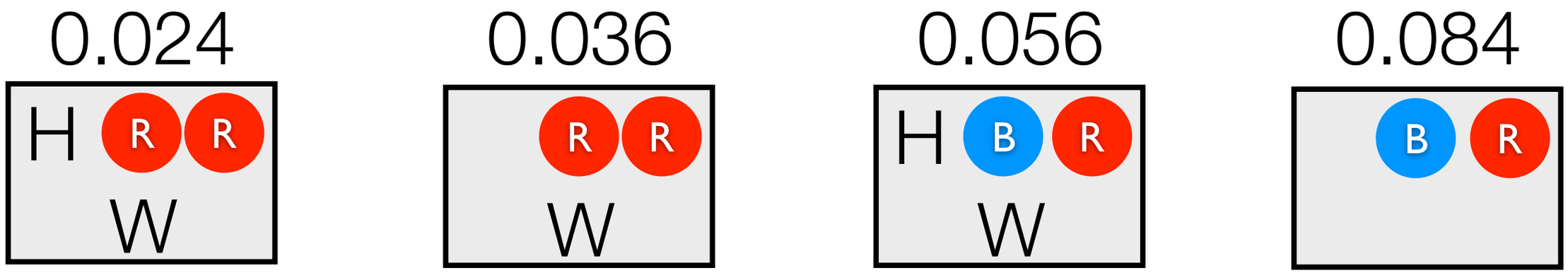


All Possible Worlds



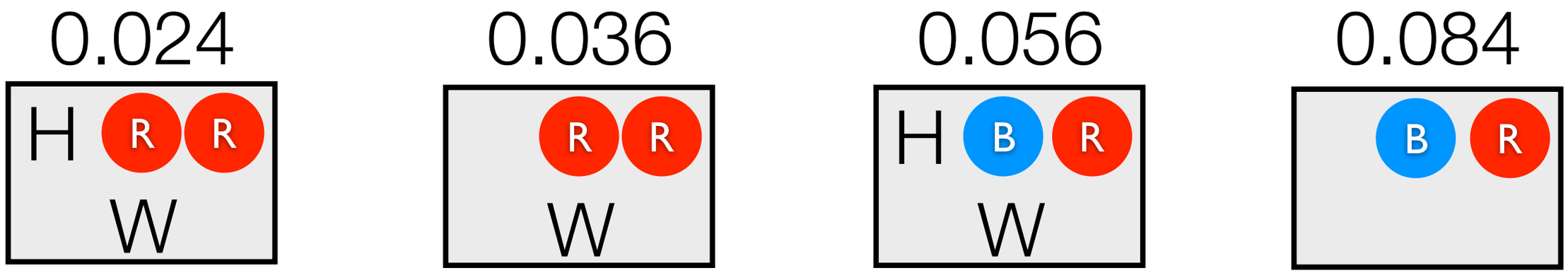
Most likely world where **win** is

MPE Inference



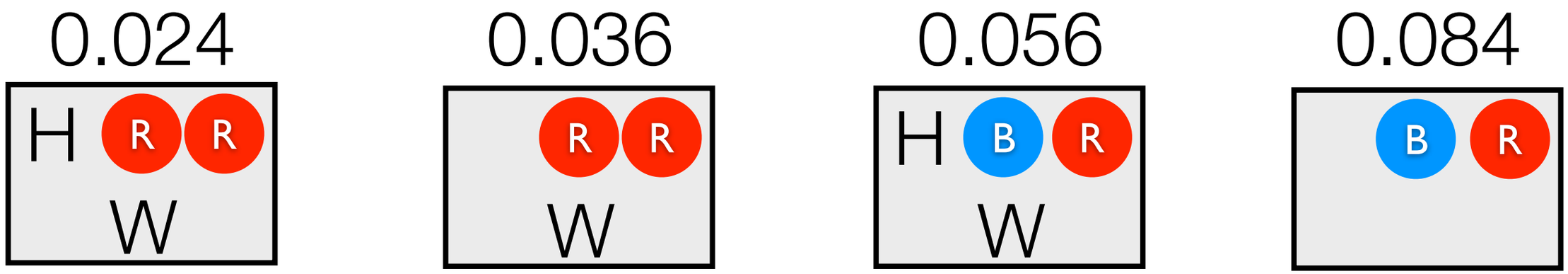
Most likely world where **win** is

MPE Inference



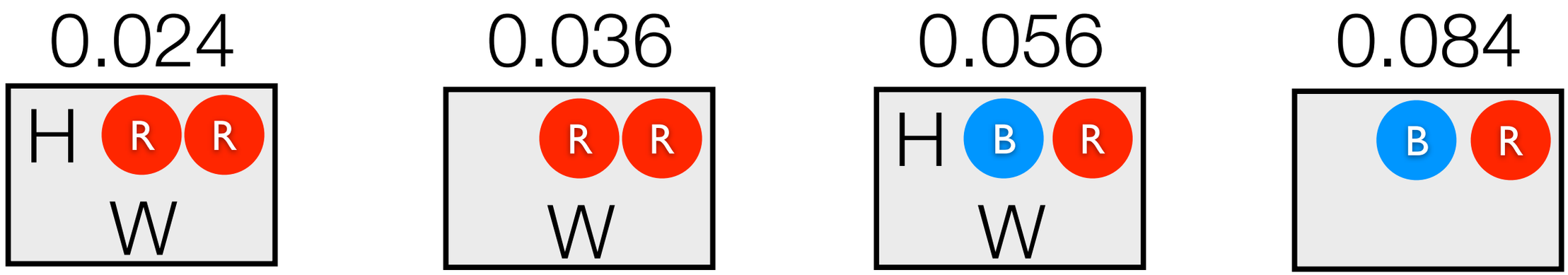
Most likely world where **`col(2,blue)`** is false?

MPE Inference



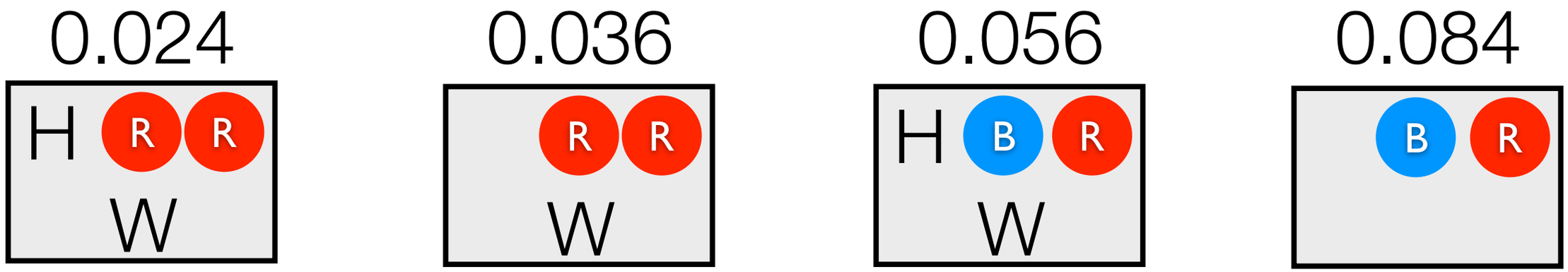
Most likely world where **`col(2,blue)`** is false?

MPE Inference



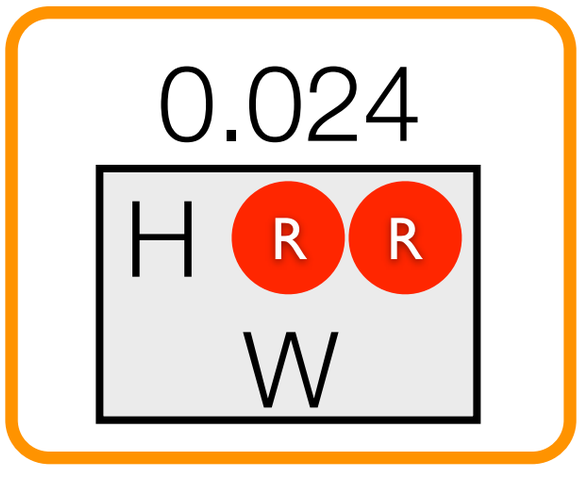
P(win)= ?

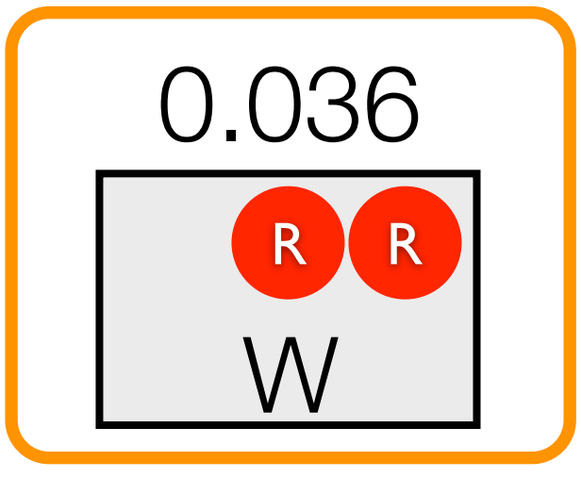
Marginal Probability

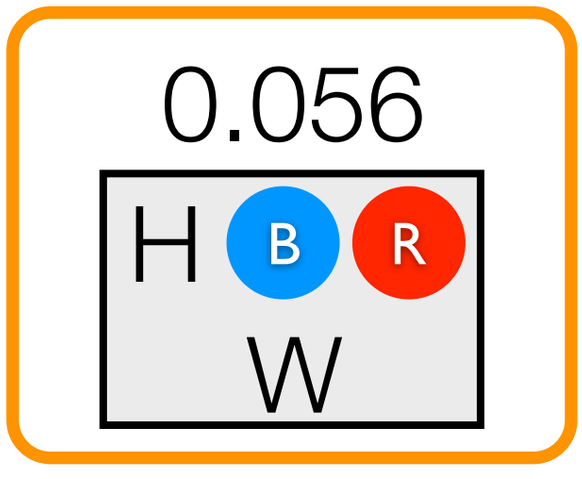


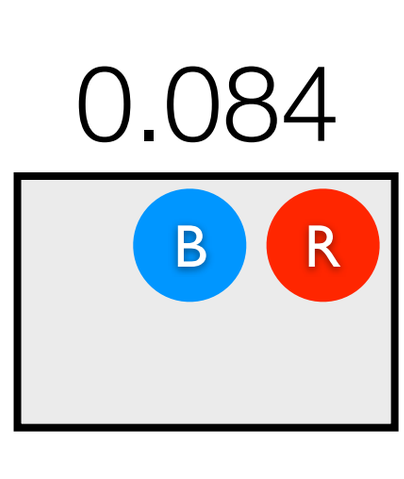
P(win)= Σ

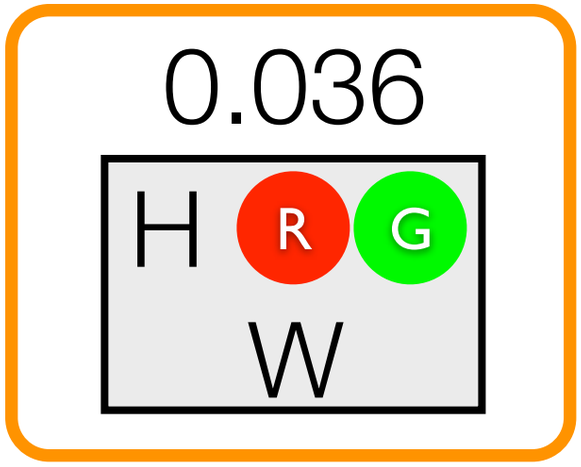
Marginal Probability

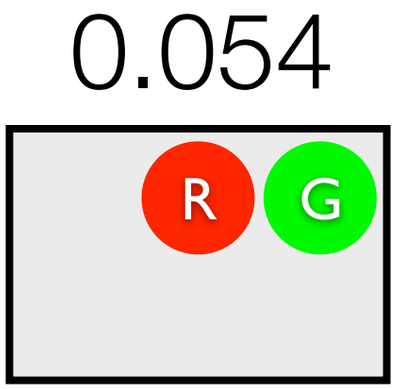


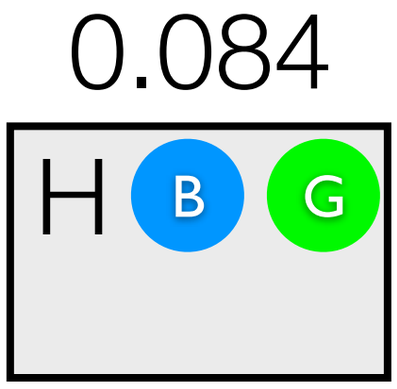


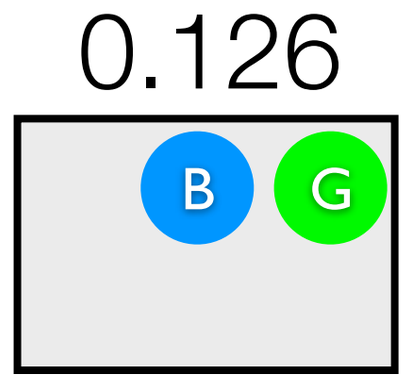


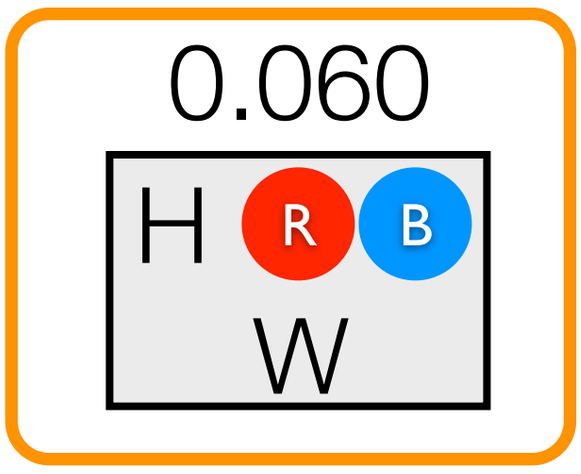


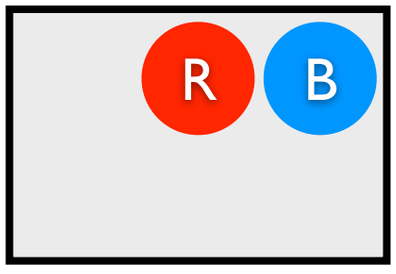


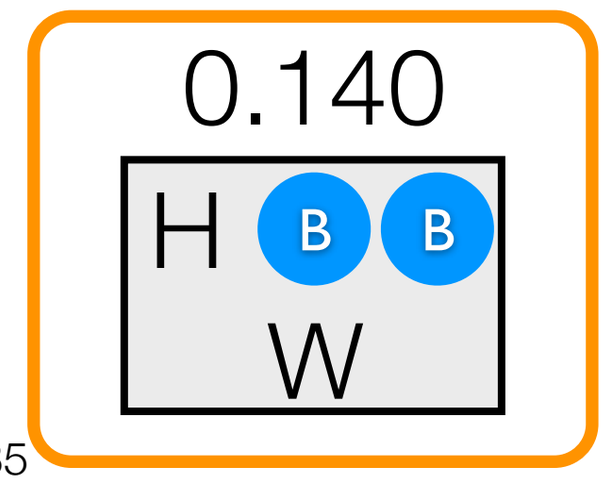


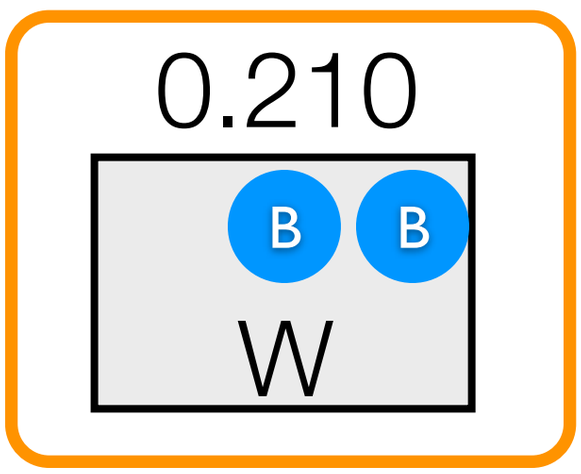






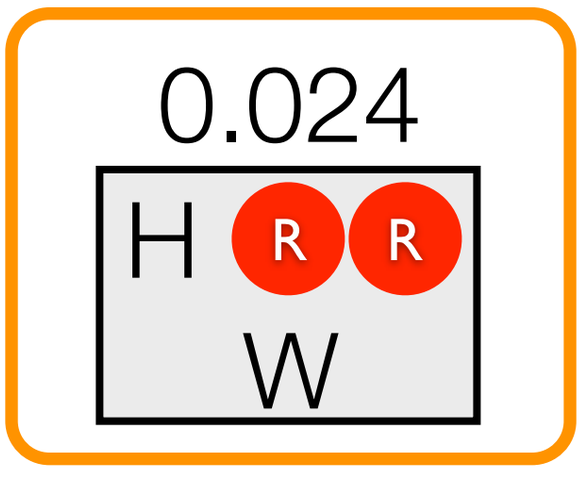


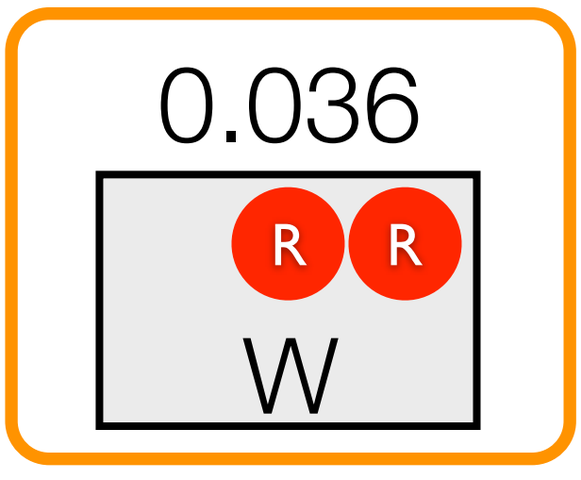


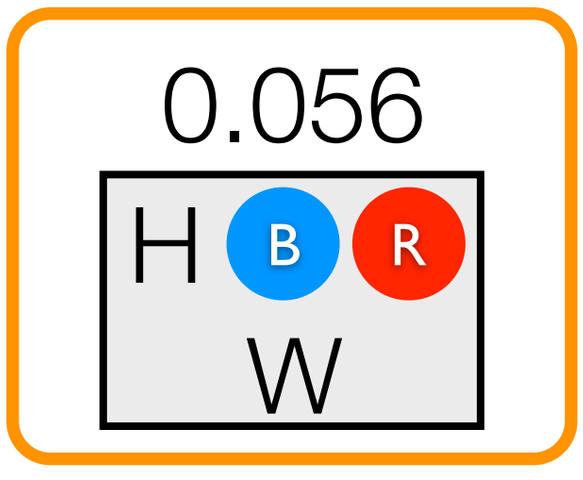


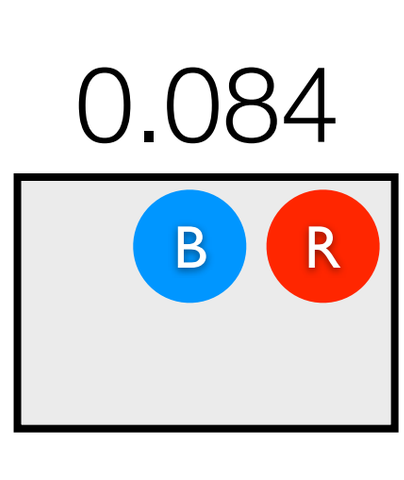
P(win)= $\sum$ =0.562

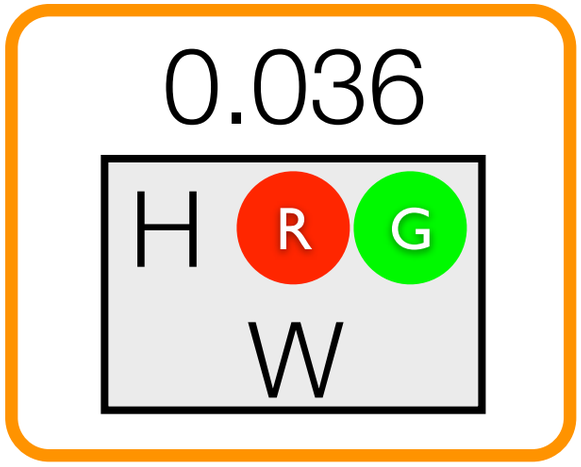
Marginal Probability

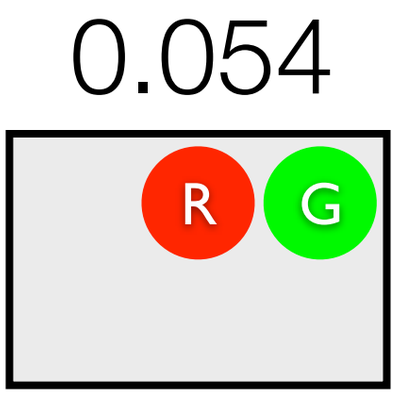


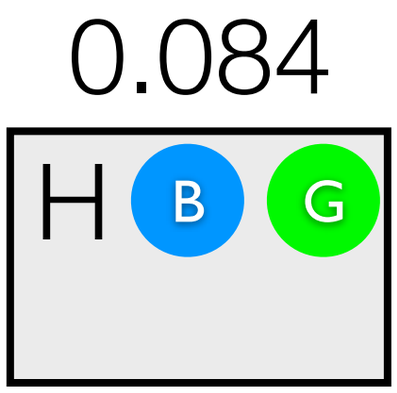


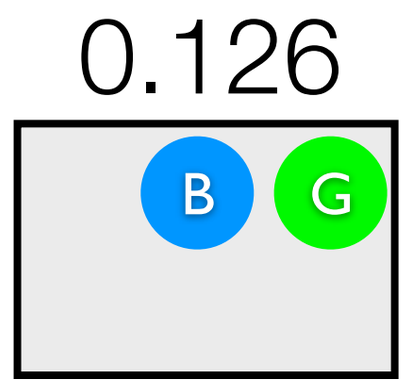


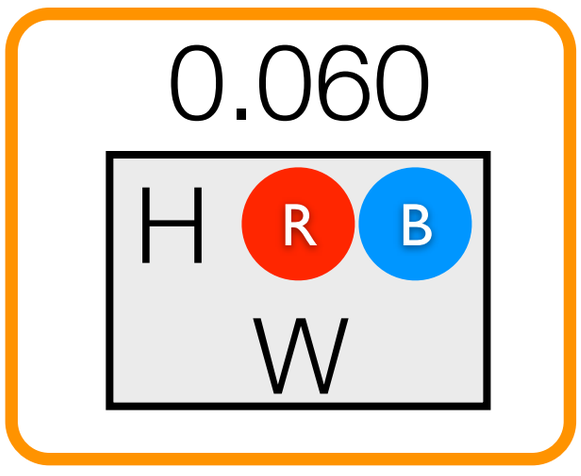


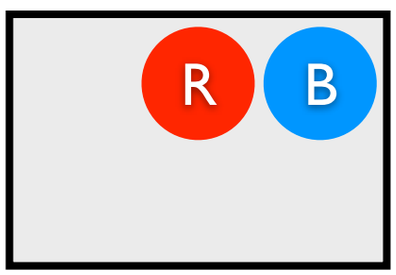


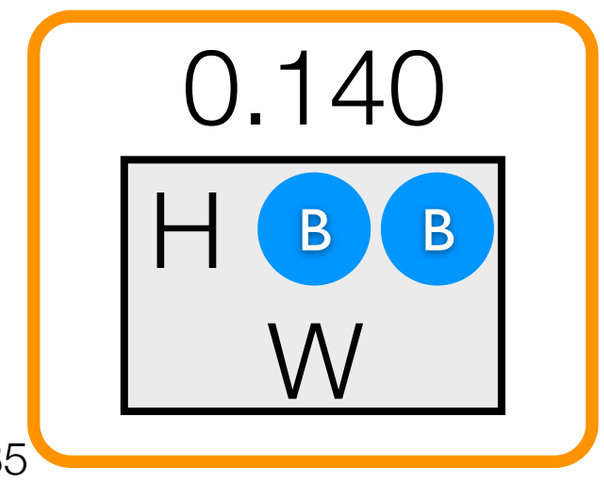


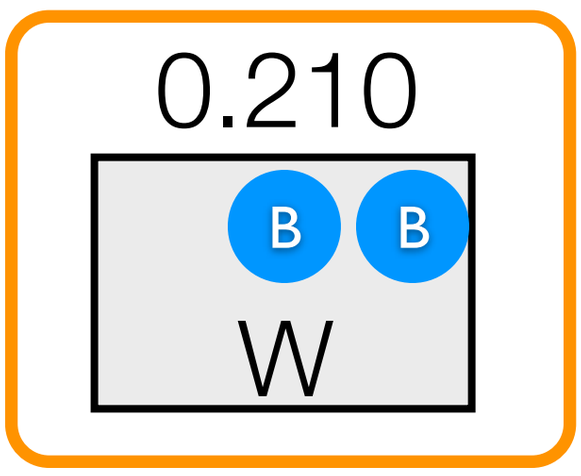








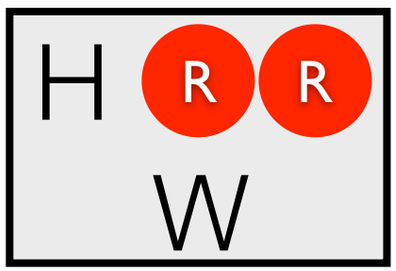




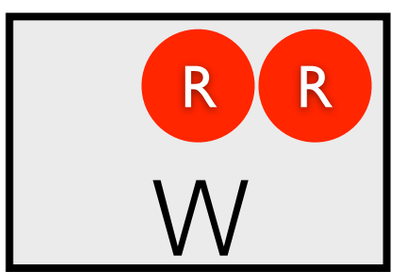
P(win|col(2,green))=?

Conditional Probability

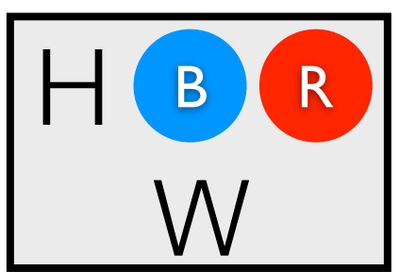
0.024



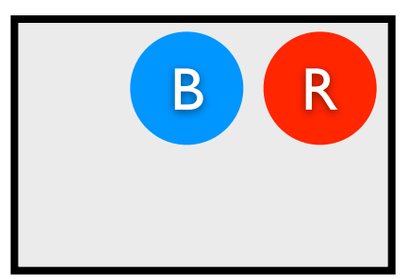
0.036



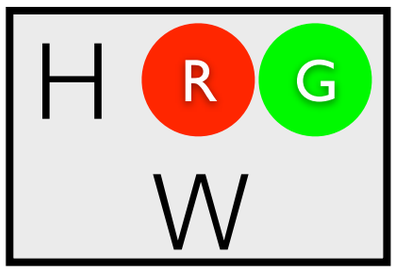
0.056



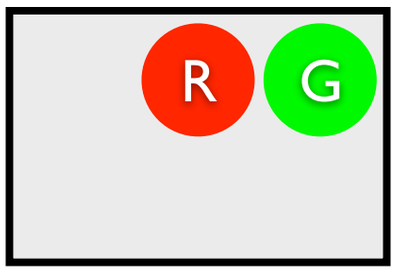
0.084



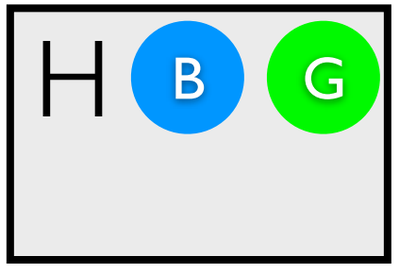
0.036



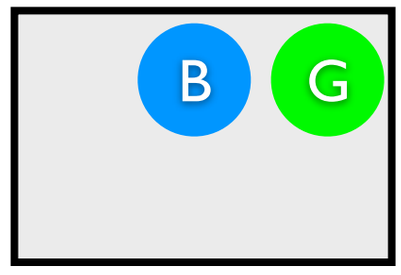
0.054



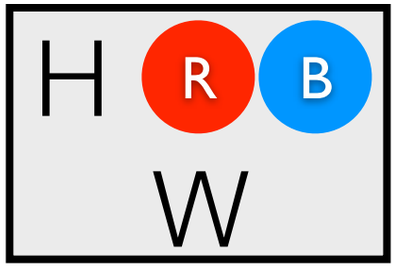
0.084



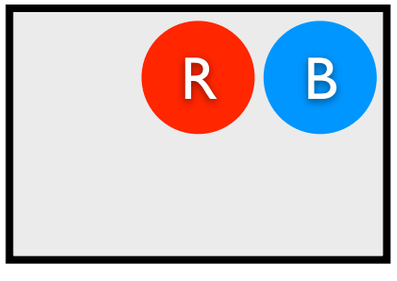
0.126



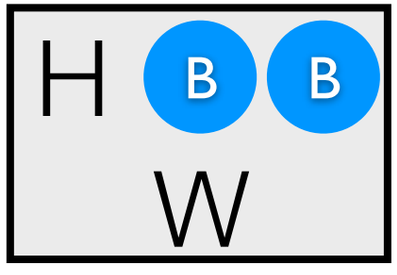
0.060



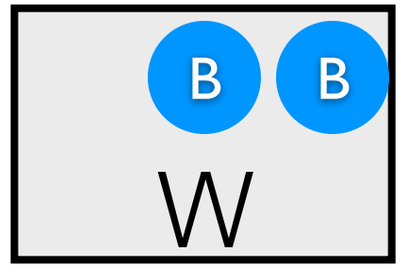
0.090



0.140



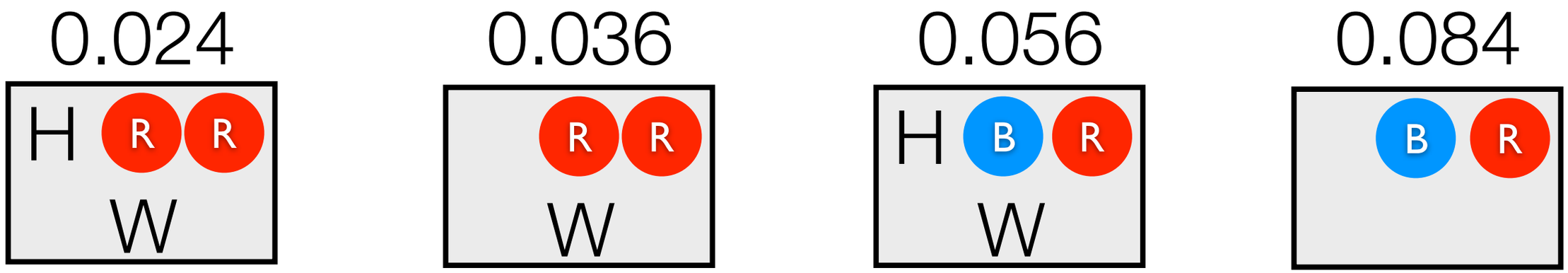
0.210



$$P(win|col(2,green)) = \sum / \sum$$

$$= P(win \wedge col(2,green)) / P(col(2,green))$$

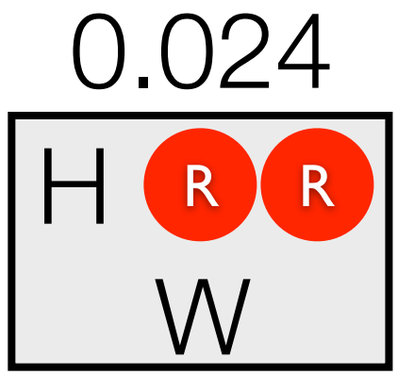
Conditional Probability



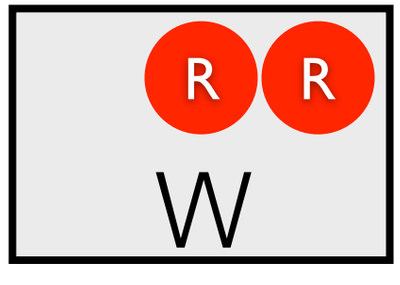
$$P(win|col(2,green)) = \sum / \sum$$

$$= P(win \wedge col(2,green)) / P(col(2,green))$$

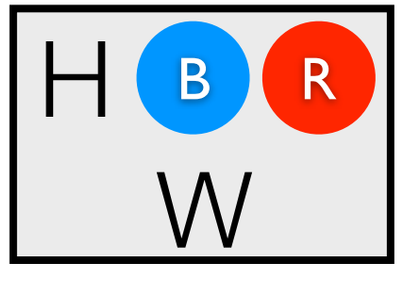
Conditional Probability



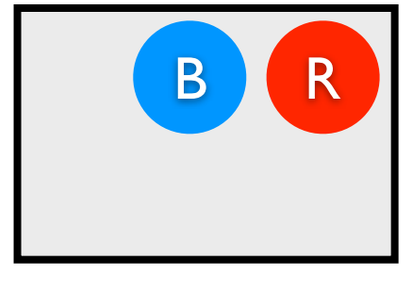
0.024



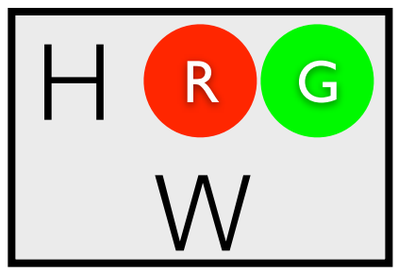
0.036



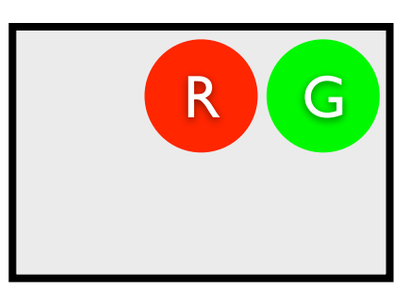
0.056



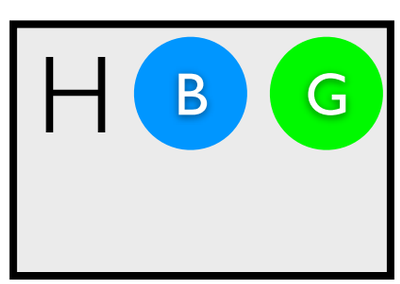
0.084



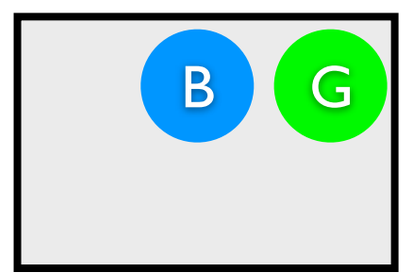
0.036



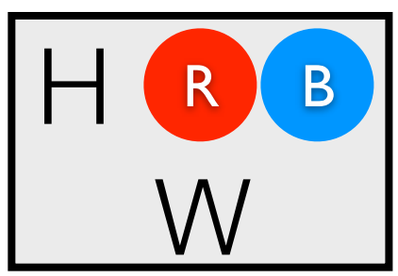
0.054



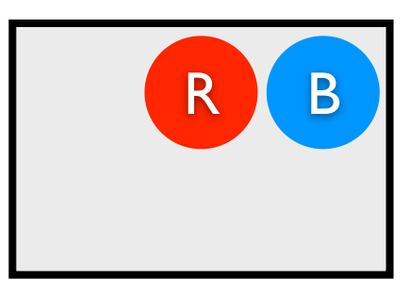
0.084



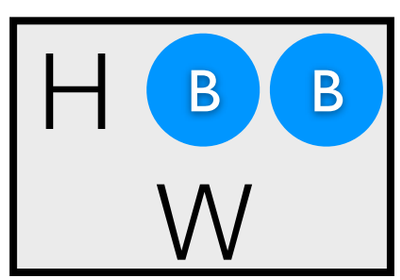
0.126



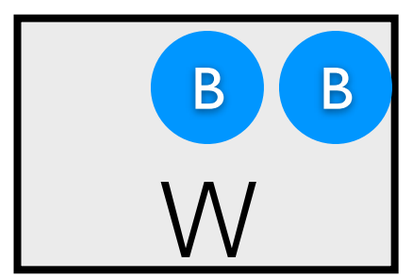
0.060



0.090



0.140

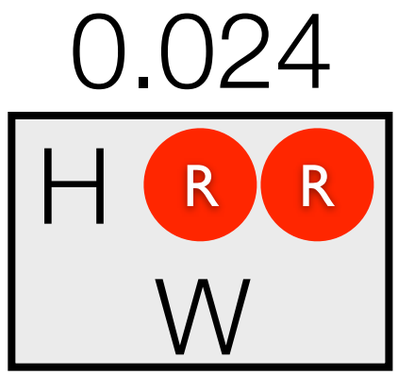


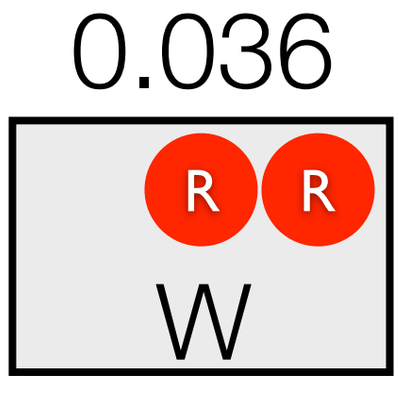
0.210

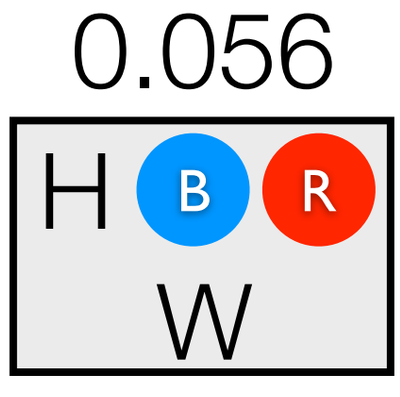
$$P(win|col(2,green)) = \sum / \sum$$

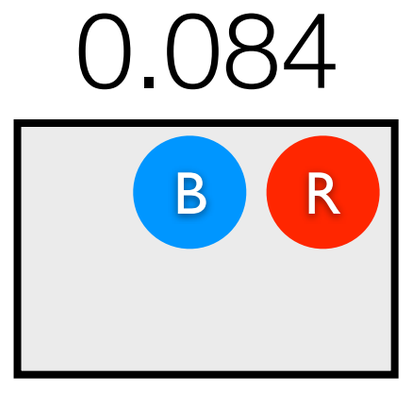
$$= 0.036/0.3 = 0.12$$

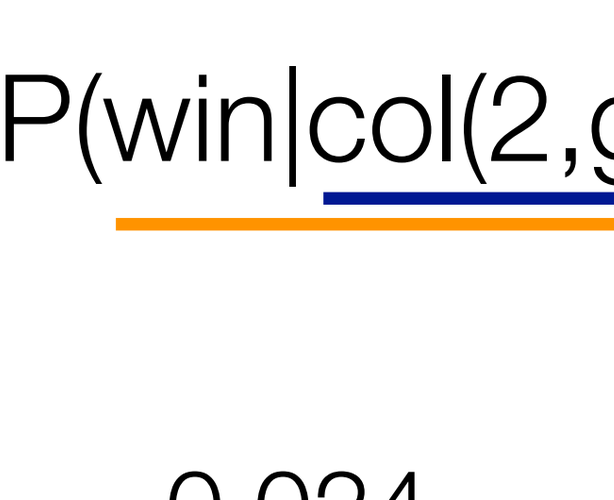
Conditional Probability

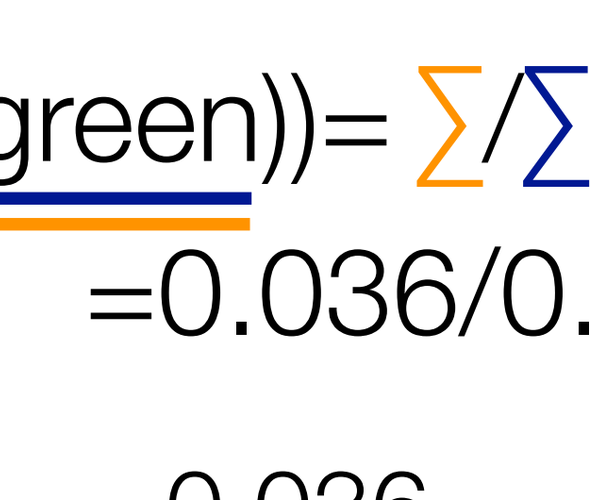


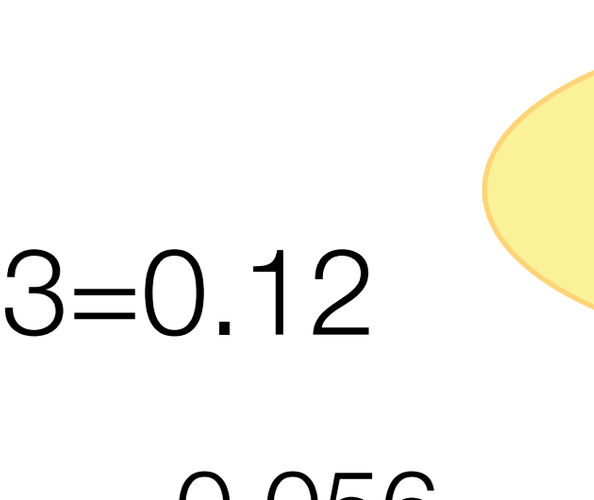


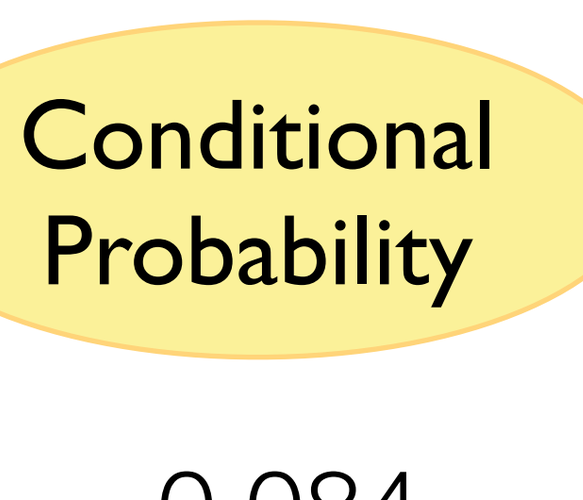


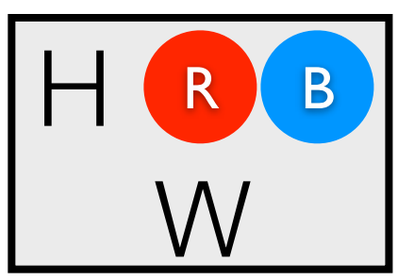


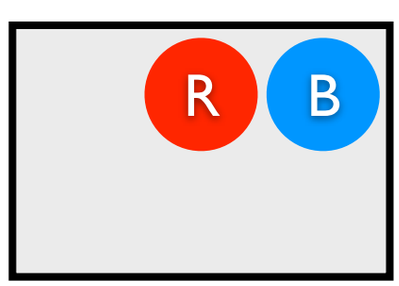


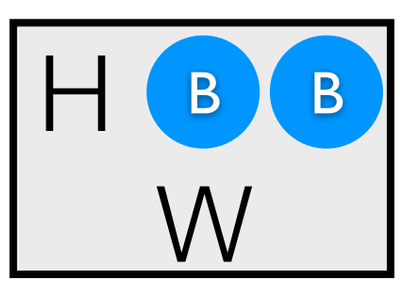


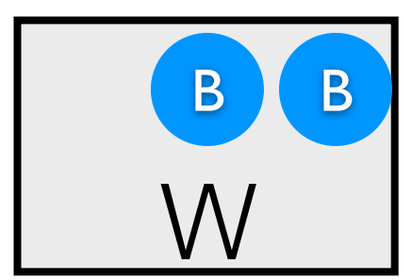












Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

subset of probabilistic facts

Prolog rules

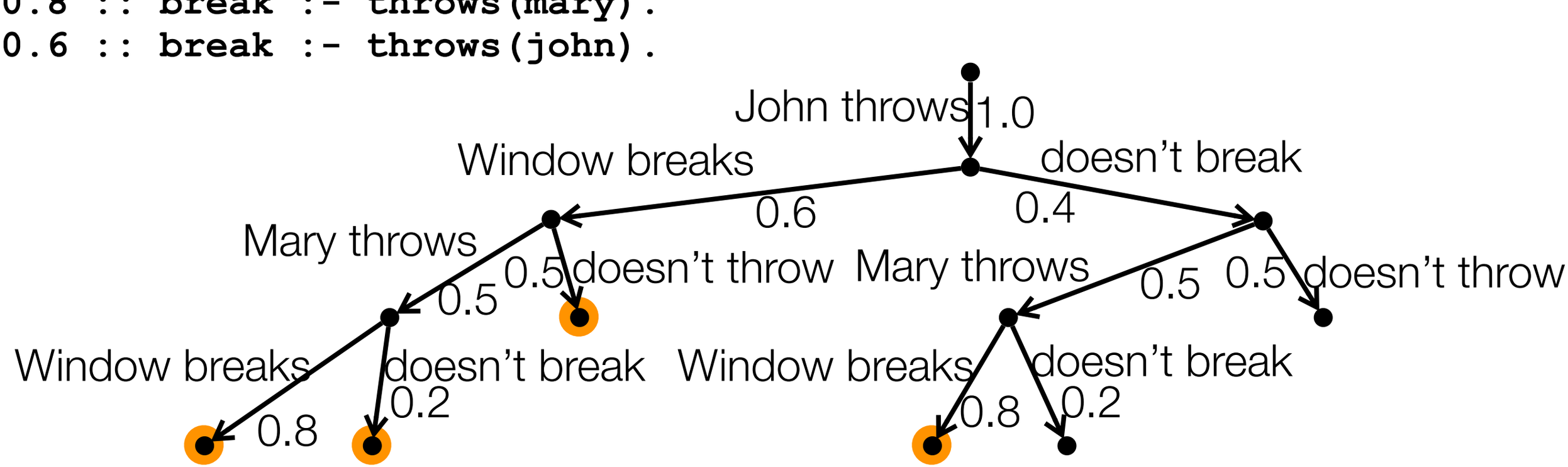
probability of possible world

[Vennekens et al, ICLP 04]

Alternative view: CP-Logic

```
throws(john).
0.5::throws(mary).

0.8 :: break :- throws(mary).
0.6 :: break :- throws(john).
```

probabilistic causal laws



$$P(break)=0.6\times0.5\times0.8+0.6\times0.5\times0.2+0.6\times0.5+0.4\times0.5\times0.8$$

ProbLog Examples

Check out the tutorial: https://dtai.cs.kuleuven.be/problog/tutorial.html

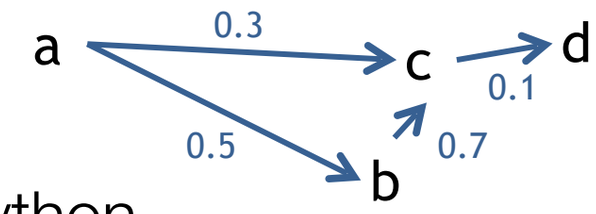
```
$ pip install problog
```

Probabilistic Graph

Discrete probabilistic reachability program

Logic Programming

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z),
             path(Z,Y).
P::edge(X,Y) :- …random vars…
```



Functional Program (Scala-like)

```
def path(start,end,visited=List())={
    if(start == end)
      return true
    if(visited.contains(start))
      return false
    return start.neighbors.exists{
      path(_,end,(visited+start))
    }
}
nodeA.neighbors = pr(…random vars…)
nodeB.neighbors = pr(…random vars…)
```

Python

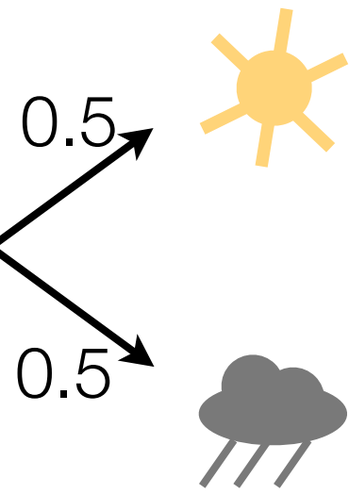
```
edges = {
    'a': [(0.9,'b'), (0.3,'c')], 'b': [(0.4,'c')], 'c': [(0.5,'d')]
}

def path(start, end, visited=()):
    if start == end: return True
    if start in visited: return False
    return any((path(nxt,end,visited+(start,)) for pr,nxt in \
        edges[start] if memchoice((pr,(start,nxt)))))
```

ProbLog by example:

Rain or sun?

ProbLog by example:

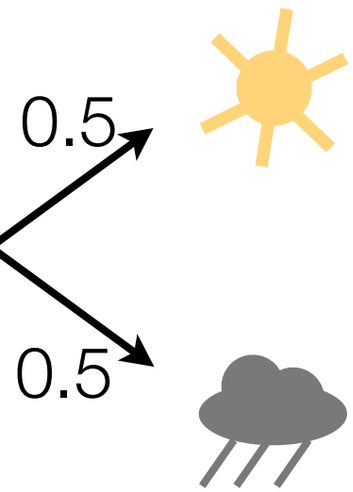
Rain or sun?



day 0

ProbLog by example:

Rain or sun?

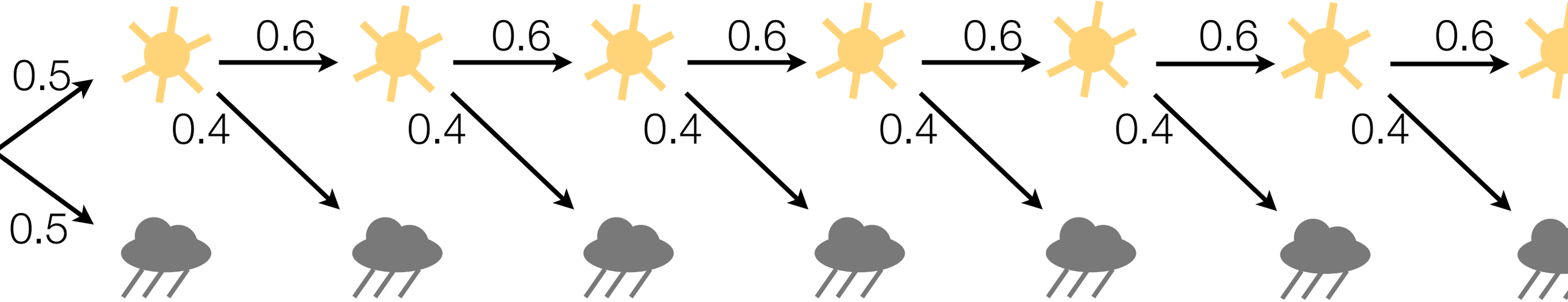


day 0

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.
```

ProbLog by example:

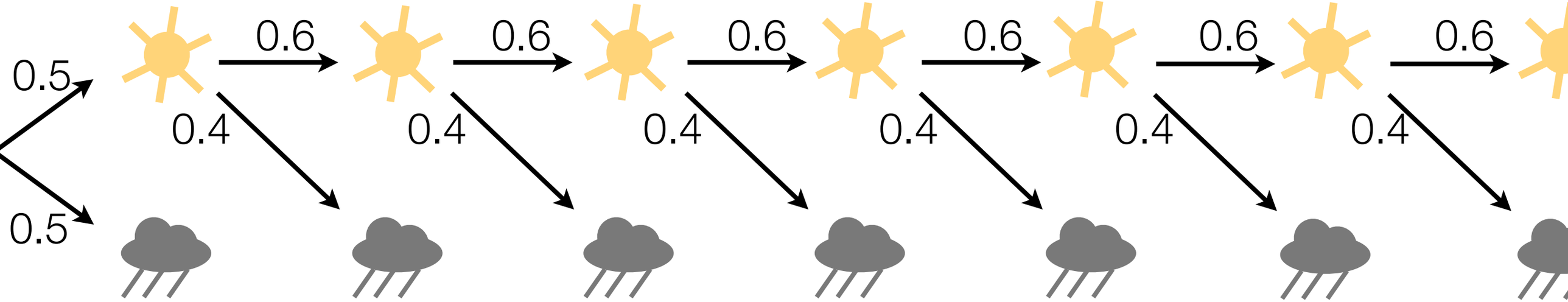
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.
```

ProbLog by example:

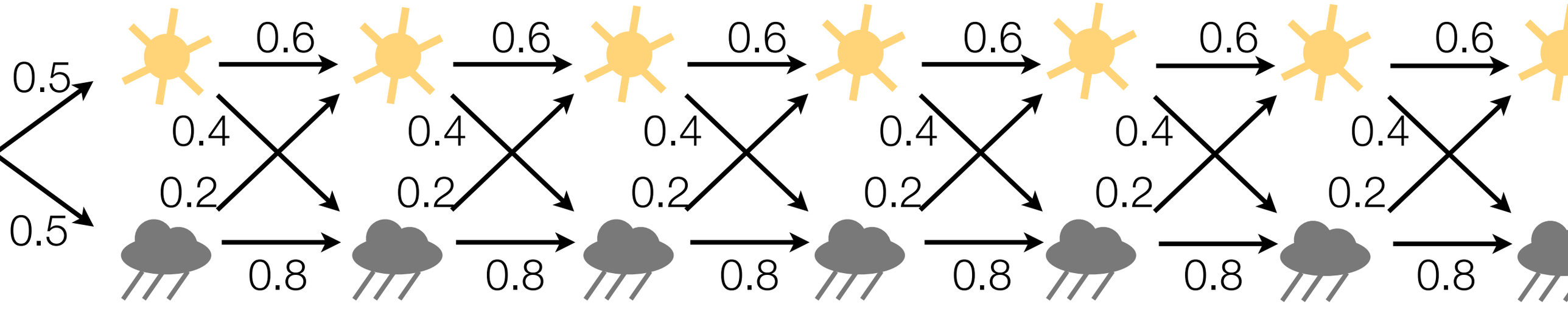
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.
```

ProbLog by example:

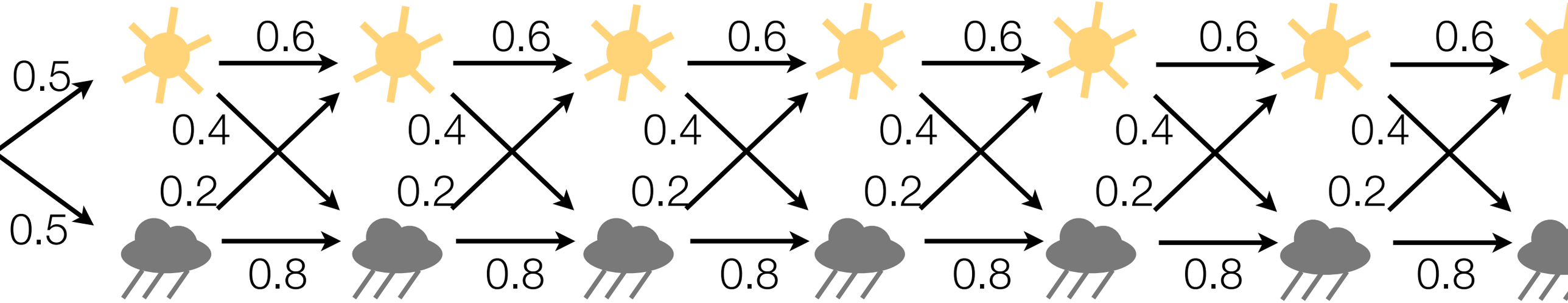
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.
```

ProbLog by example:

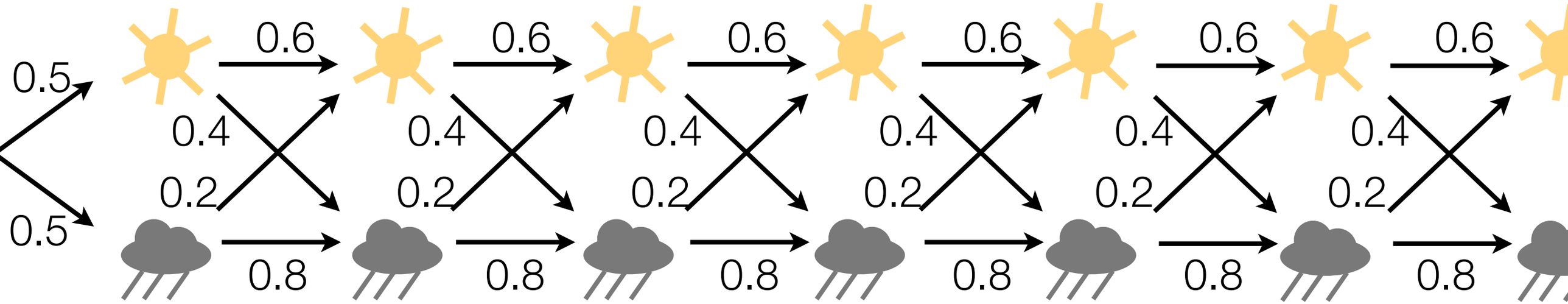
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(sun,Tprev).
```

ProbLog by example:

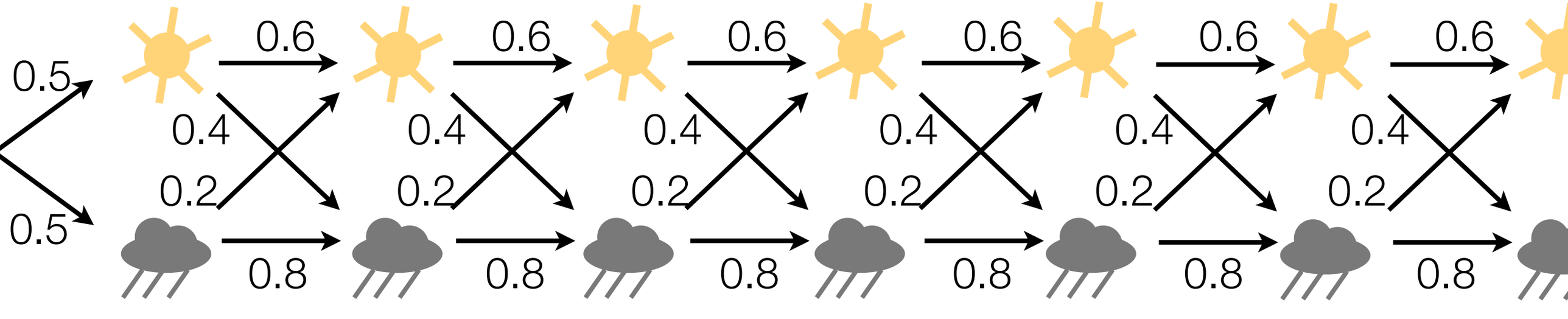
Rain or sun?




```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(rain,Tprev).
```

ProbLog by example:

Rain or sun?

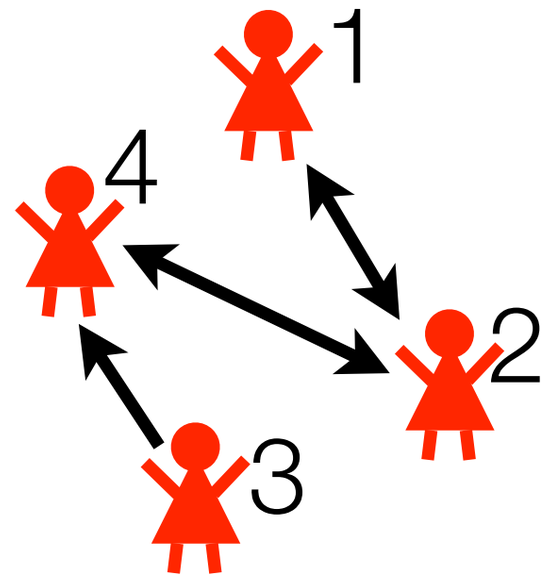



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(rain,Tprev).
```

infinite possible worlds! BUT: finitely many partial worlds suffice to answer any given ground query

ProbLog by example:

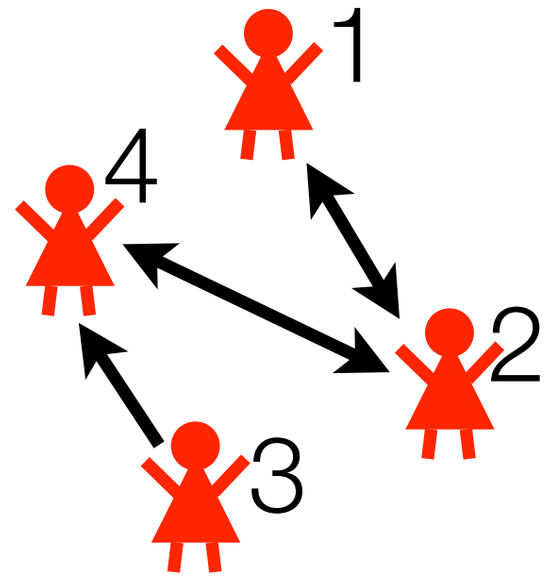
Friends & smokers




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers



typed probabilistic facts

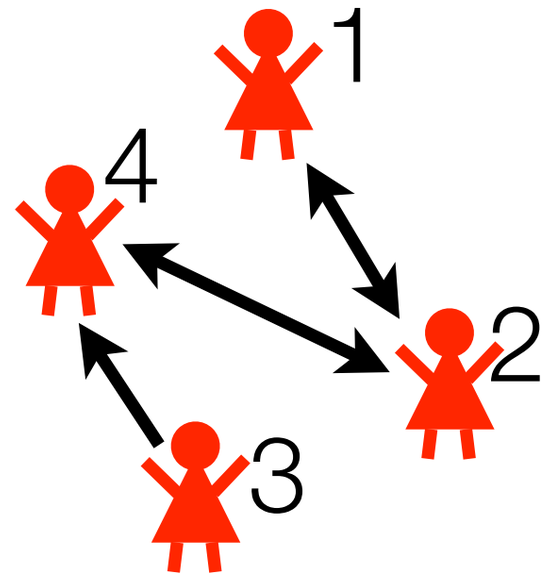
= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers



typed probabilistic facts

= a probabilistic fact for each grounding

```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).
```

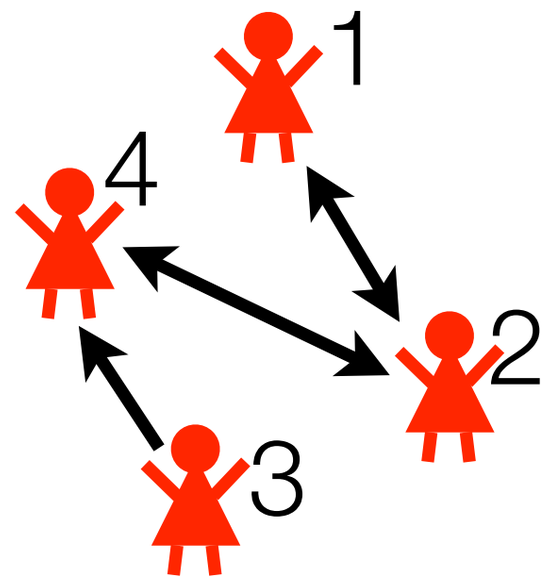
```
0.3::stress(1).
0.3::stress(2).
0.3::stress(3).
0.3::stress(4).
```

```
0.2::influences(1,1).
0.2::influences(1,2).
0.2::influences(1,3).
0.2::influences(1,4).
0.2::influences(2,1).
...
0.2::influences(4,2).
0.2::influences(4,3).
0.2::influences(4,4).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers

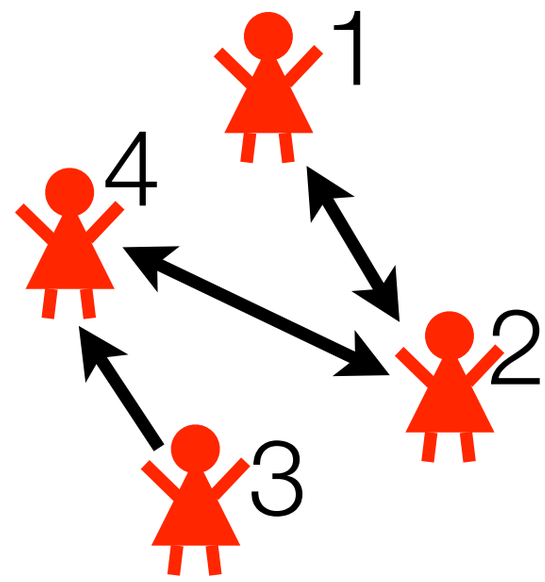



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers

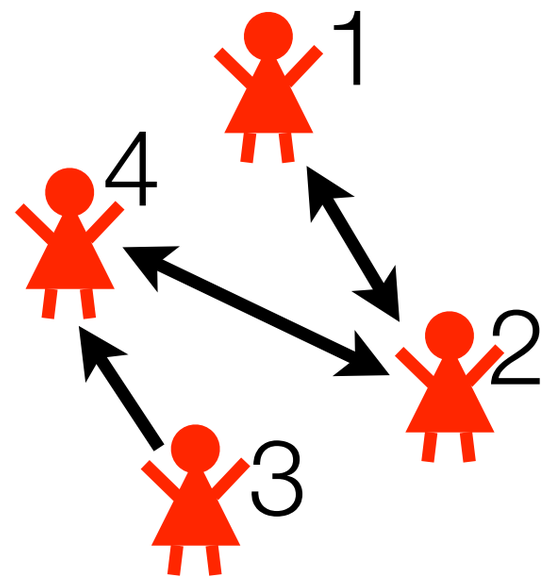



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers

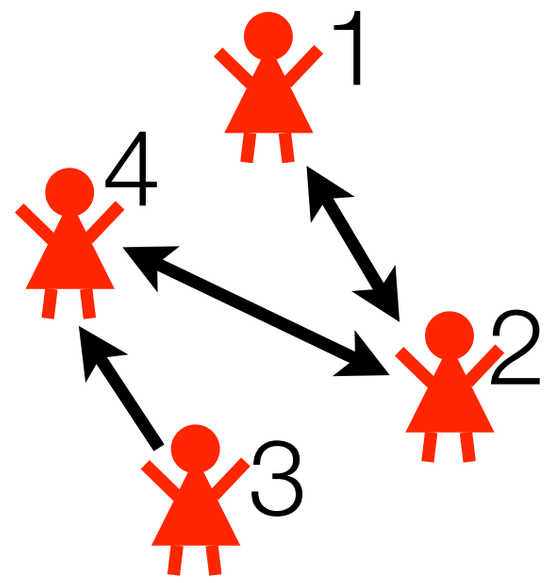



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Friends & smokers




```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
                person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) <- smokes(X).
```

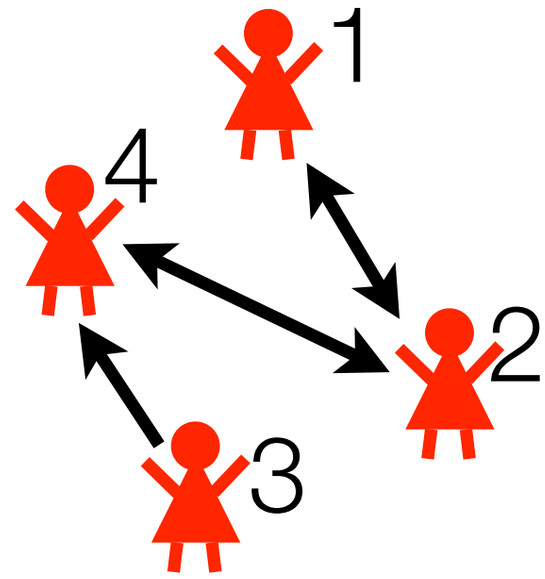
```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

annotated disjunction with implicit head atom:

with probability 0.6, nothing happens

ProbLog by example:

Friends & smokers




```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
              person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :-
     friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) <- smokes(X).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.

exce                                                    ],Limit).

exce
exce
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

flexible probability:

computed from the weight of the item

ProbLog by example:

Limited Luggage

```
weight(skis,6).                1/6::pack(skis).
weight(boots,4).               1/4::pack(boots).
weight(helmet,3).              1/3::pack(helmet).
weight(gloves,2).              1/2::pack(gloves).

P::pack(Item) :- weight(Item,Weight), P is 1.0/Weight.

exce                                                ],Limit).

exce
exce
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

flexible probability:

computed from the weight of the item

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

list of all items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

pack first item, decrease limit by its weight, and continue with rest of items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

do **not** pack first item, continue with rest of items

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

no items left: did we add too much?

ProbLog by example:

Limited Luggage

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :- weight(Item,Weight),  P is 1.0/Weight.

excess(Limit) :- excess([skis,boots,helmet,gloves],Limit).

excess([],Limit) :- Limit<0.
excess([I|R],Limit) :-
   pack(I), weight(I,W), L is Limit-W, excess(R,L).
excess([I|R],Limit) :-
   \+pack(I), excess(R,Limit).
```

ProbLog

- **probabilistic choices** + their **consequences**
- probability distribution over **possible worlds**
- how to efficiently answer **questions**?
  - most probable world (MPE inference)
  - probability of query (computing marginals)
  - probability of query given evidence

Summary: ProbLog Syntax

- input database: ground facts
- probabilistic facts

```
person(bob).

0.5::stress(bob).
```

- flexible probabilities
- annotated disjunctions
- Prolog clauses

```
P::pack(Item) :- weight(Item,W),
                 P is 1.0/W.

0.4::asthma(X) :- smokes(X).

0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

smokes(X) :- influences(Y,X), smokes(Y).

excess([I|R],Limit) :- \+pack(I), excess(R,Limit).
```

Answering Questions

Given:

program

queries

evidence

?

Find:

marginal probabilities

conditional probabilities

MPE state

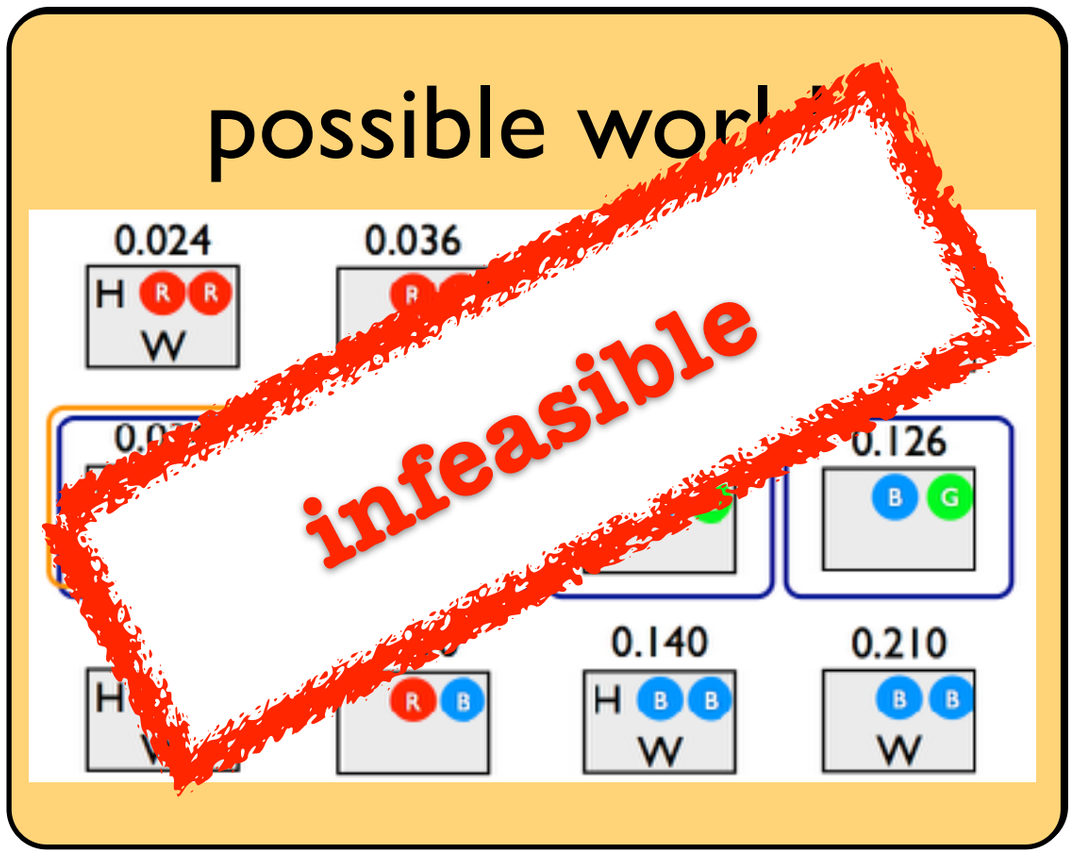
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

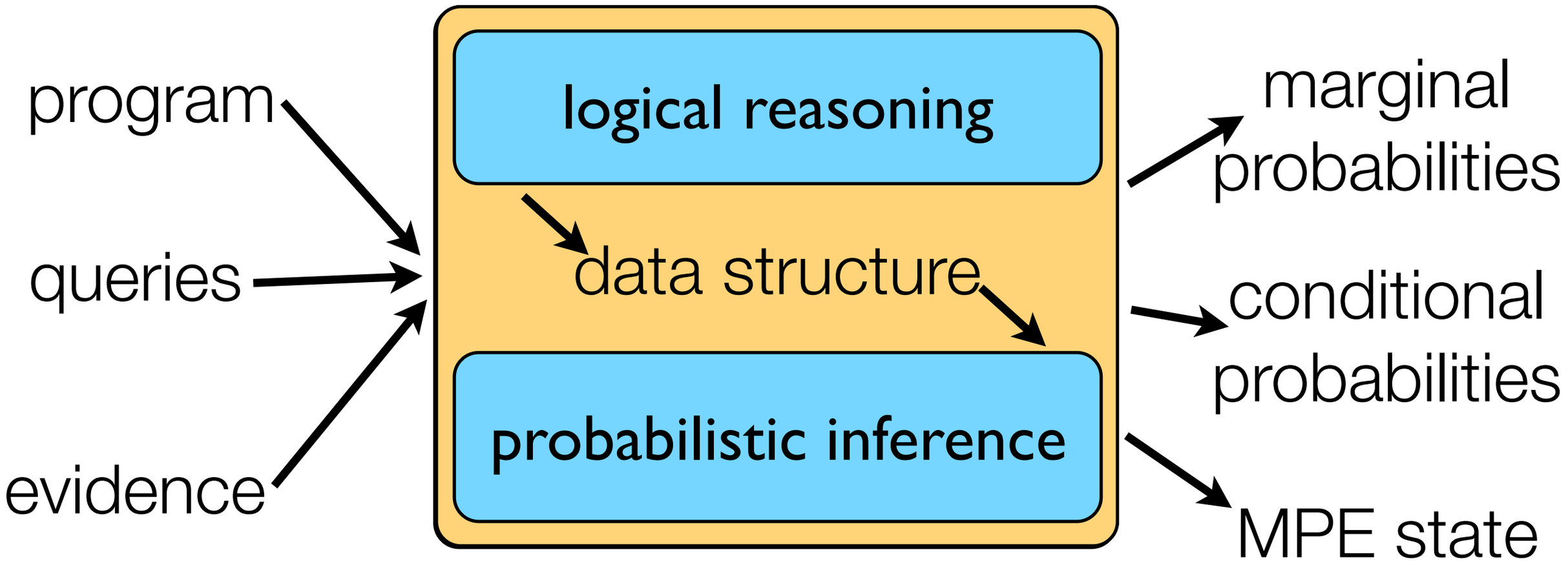
conditional probabilities

MPE state

Answering Questions

Given:

Find:

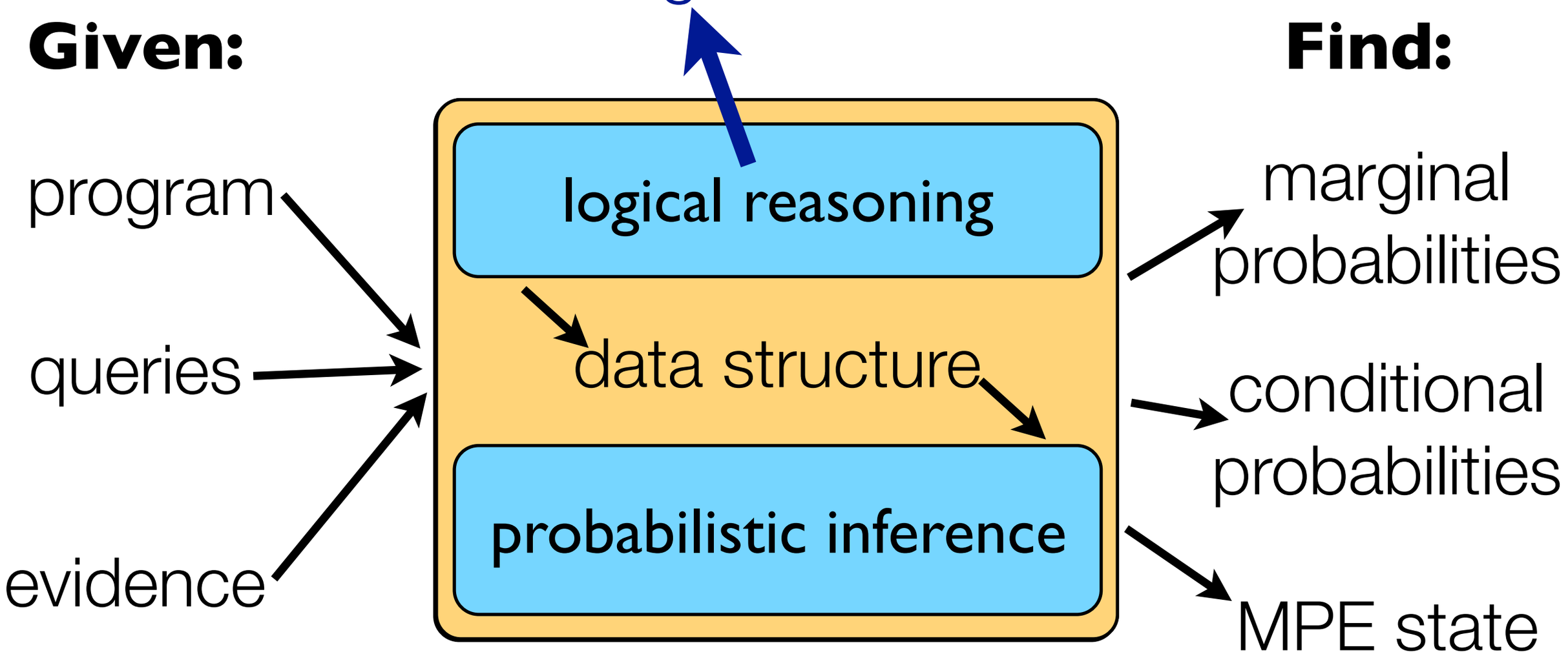


Answering Questions

1. using proofs
2. using models

Given:

Find:



Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

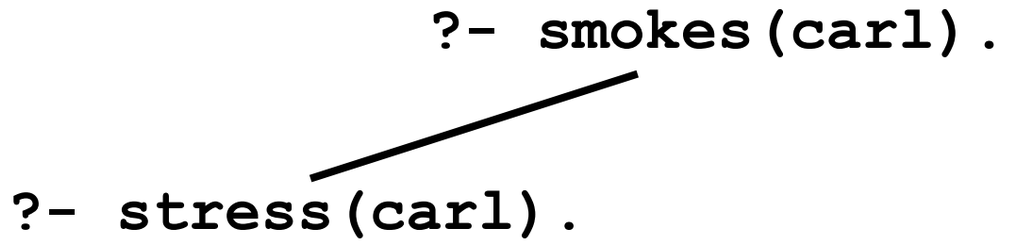
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

```
?- smokes(carl).
```

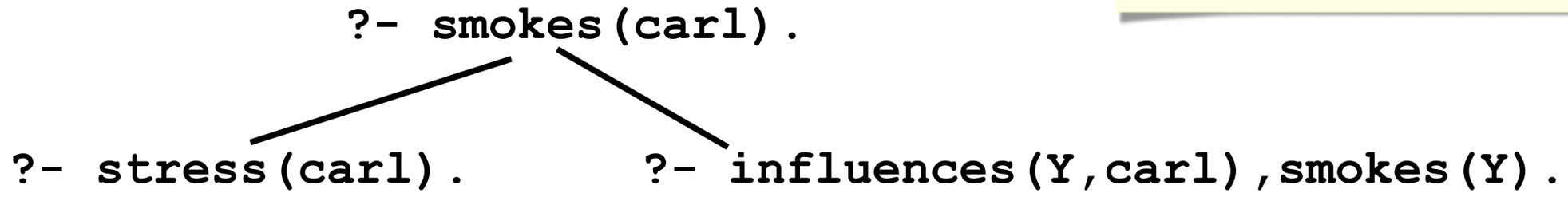
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



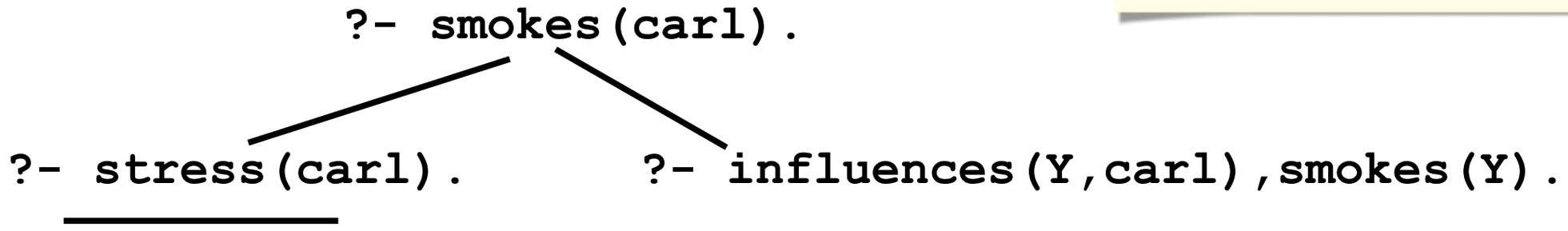
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



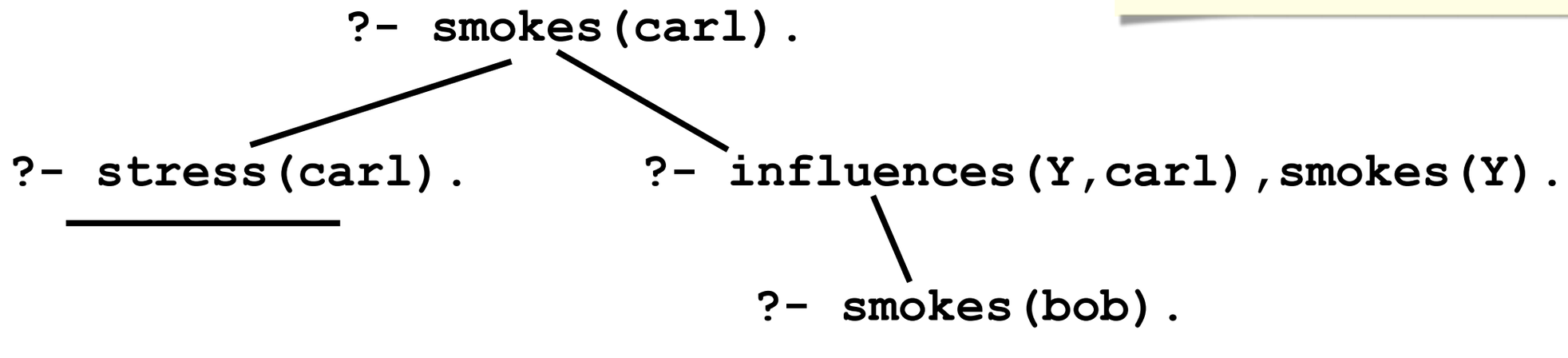
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



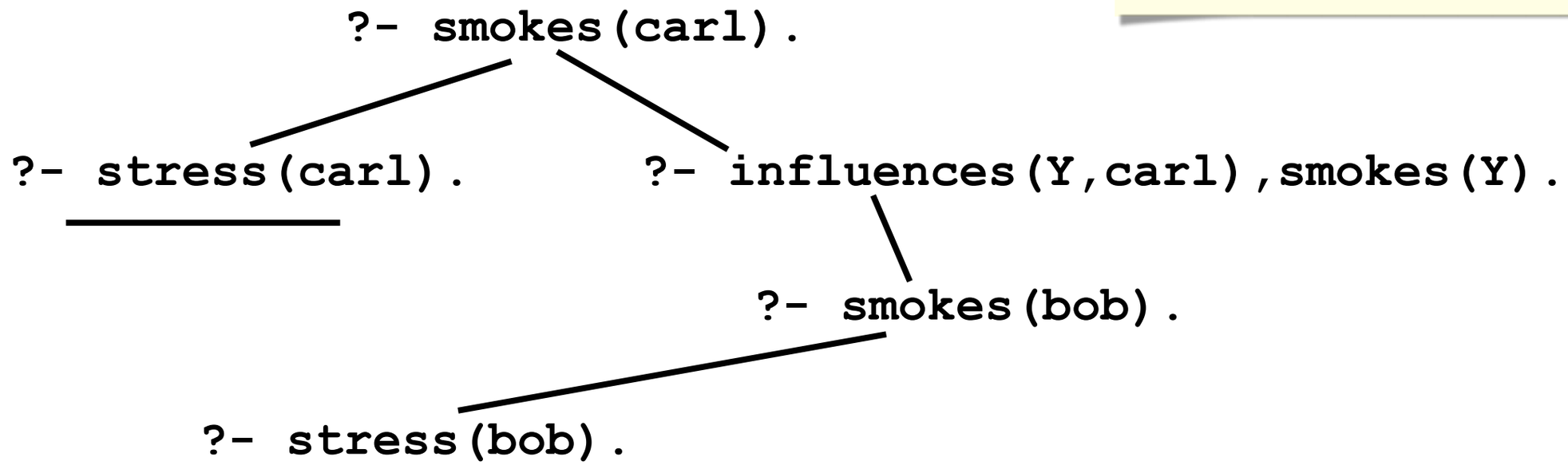
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



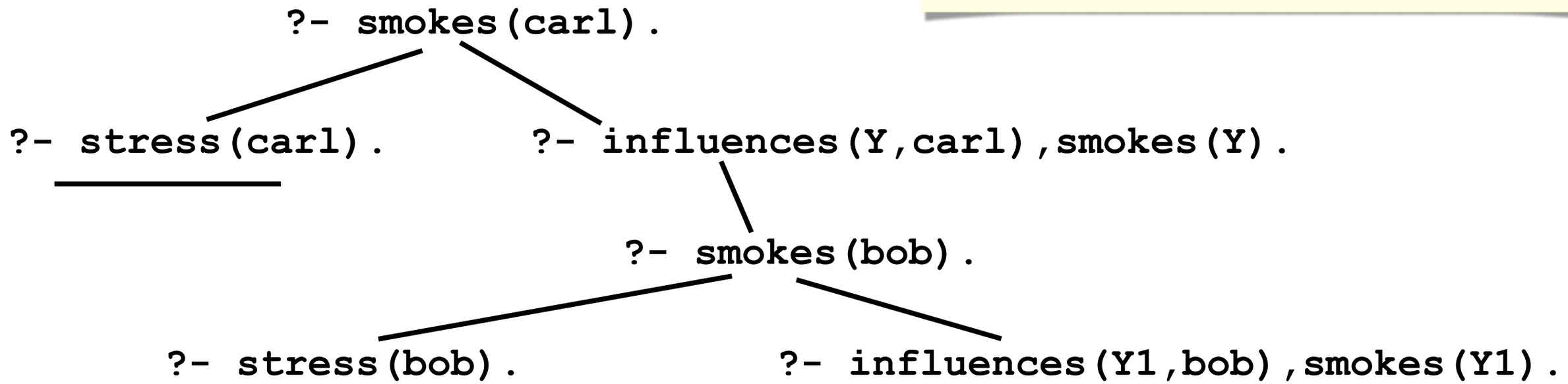
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



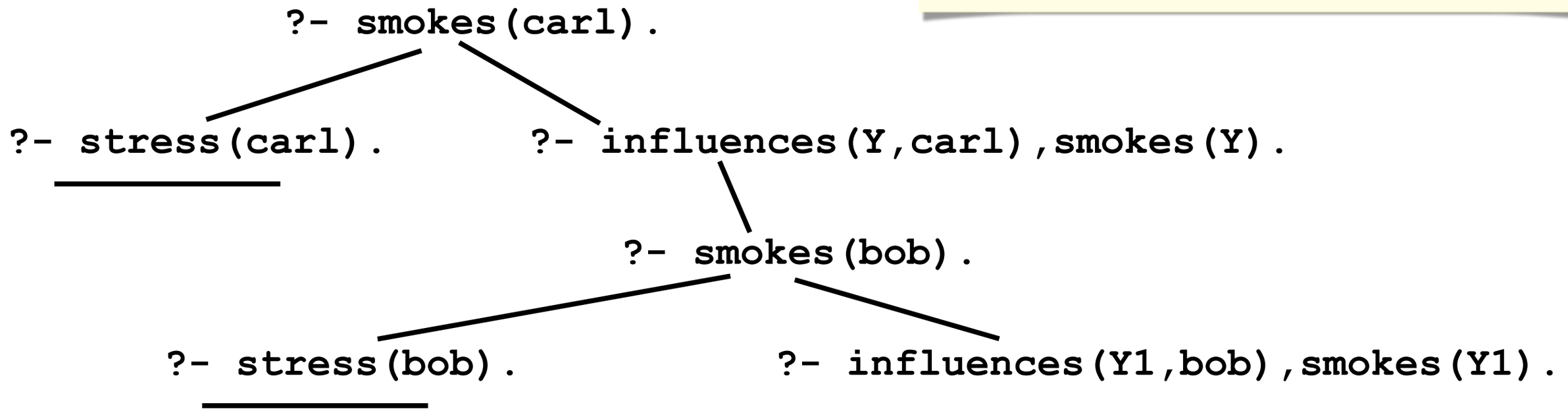
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



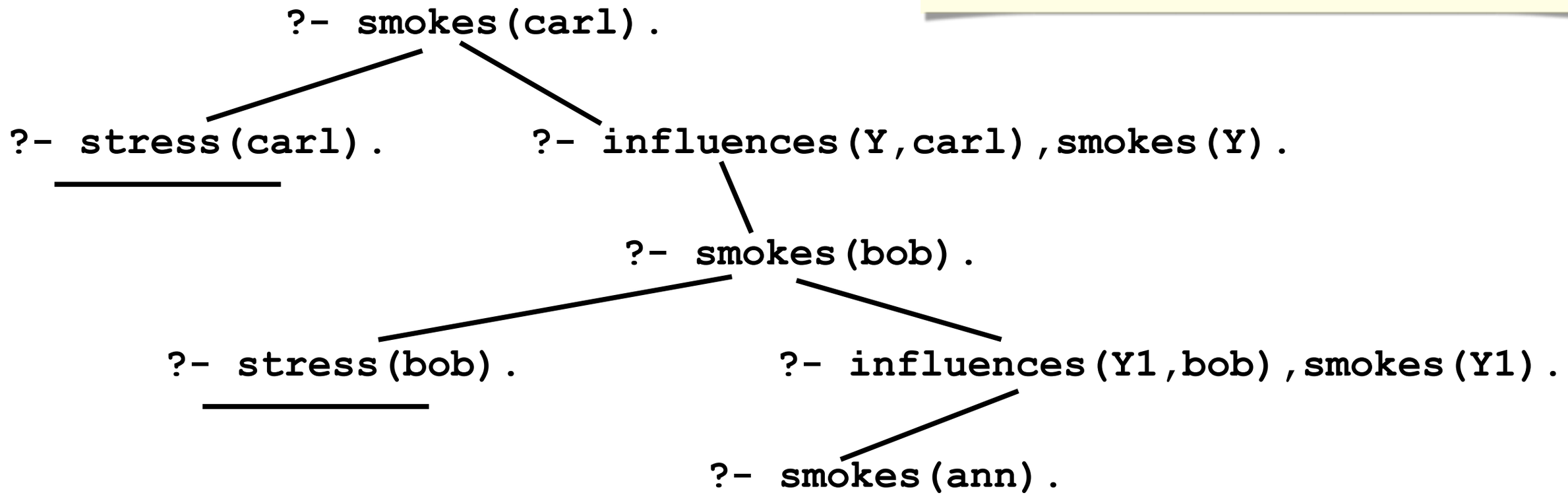
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



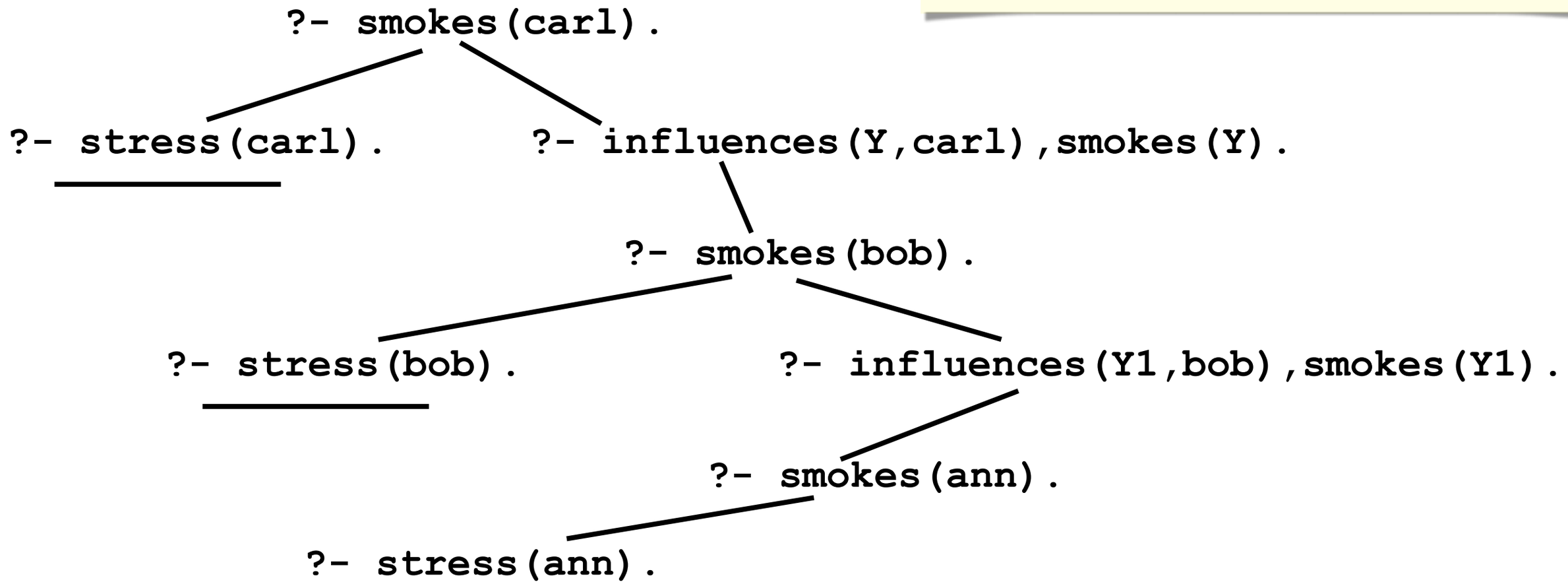
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



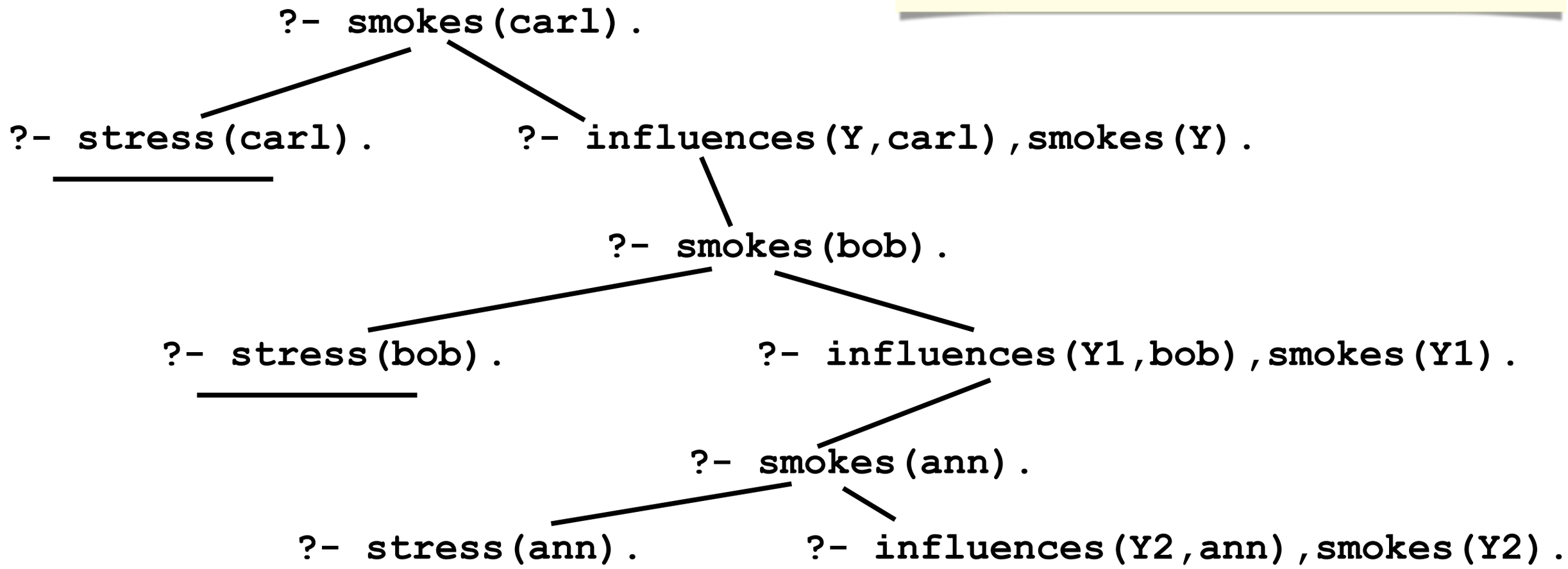
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



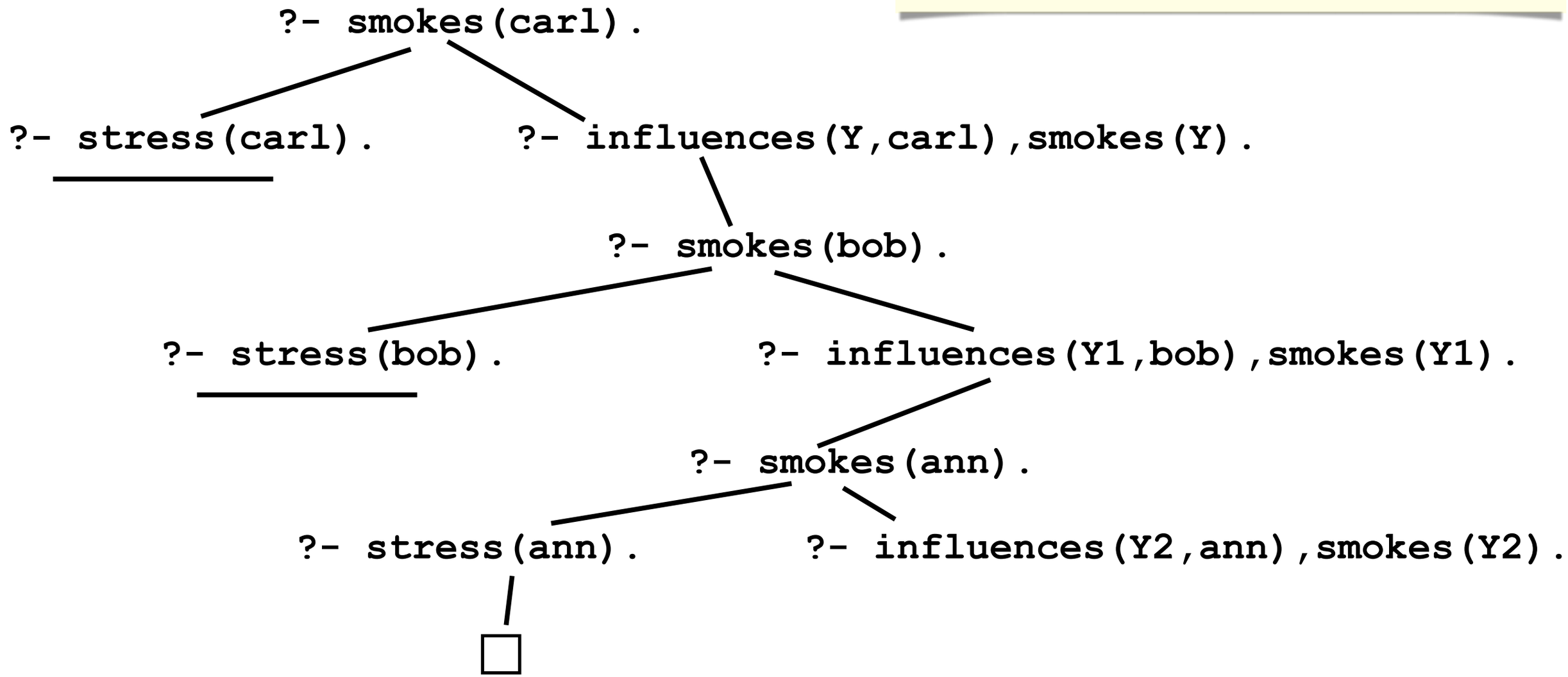
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



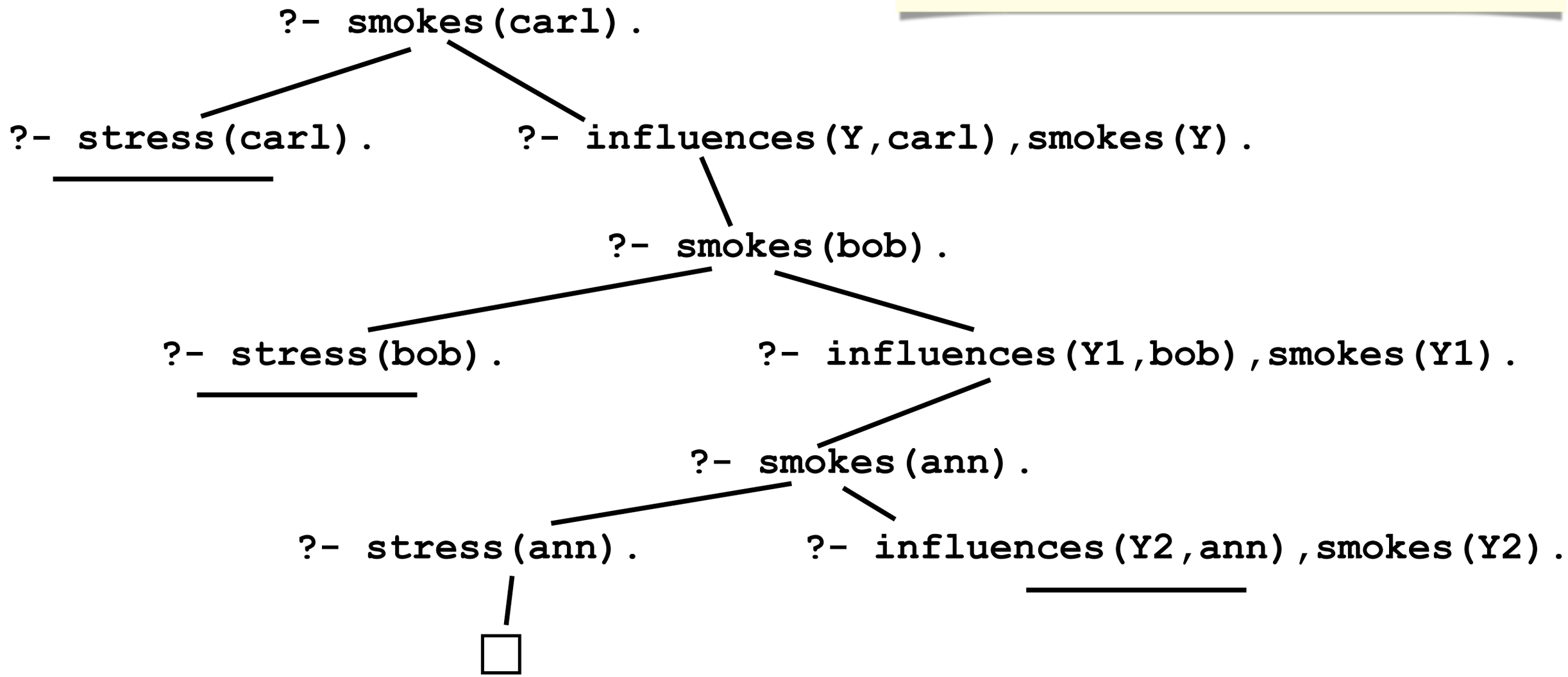
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



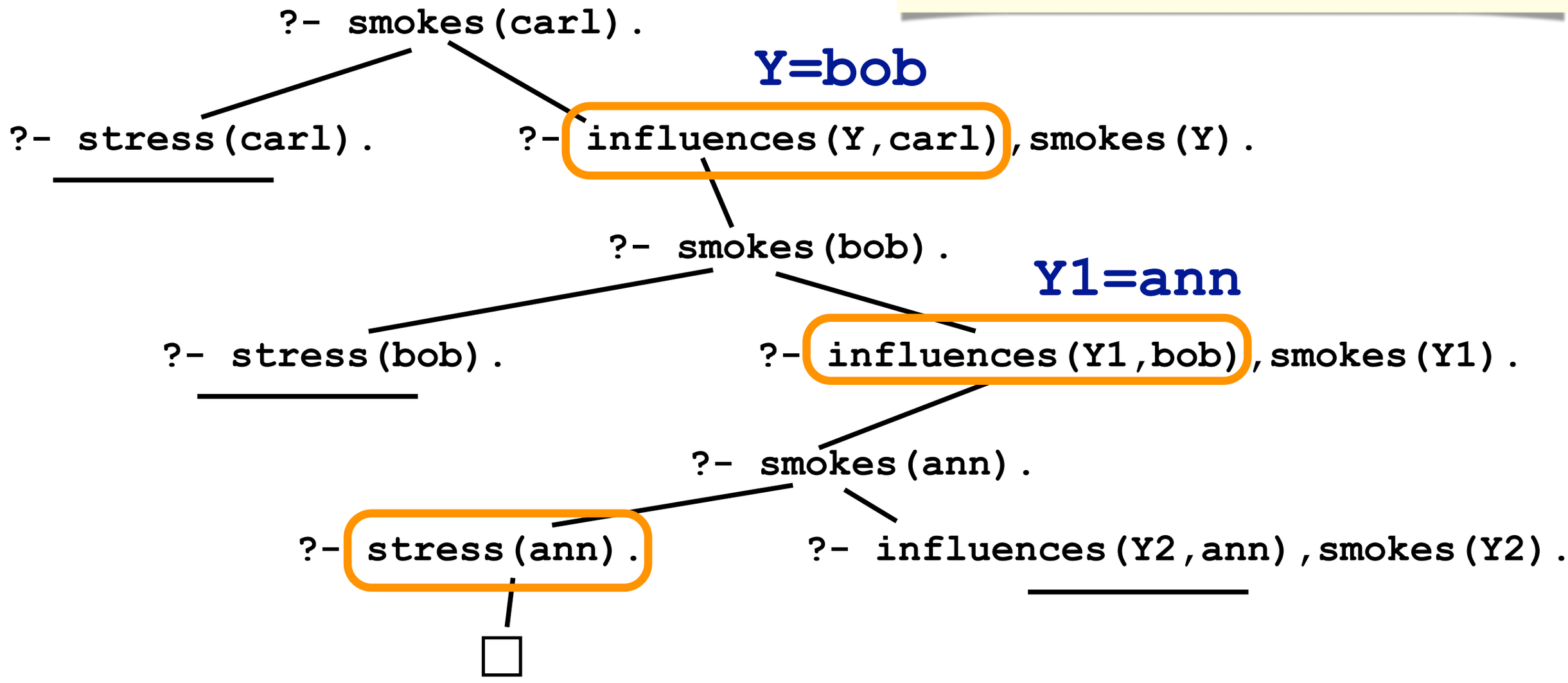
Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



Logical Reasoning: Proofs in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X), smokes(Y).
```

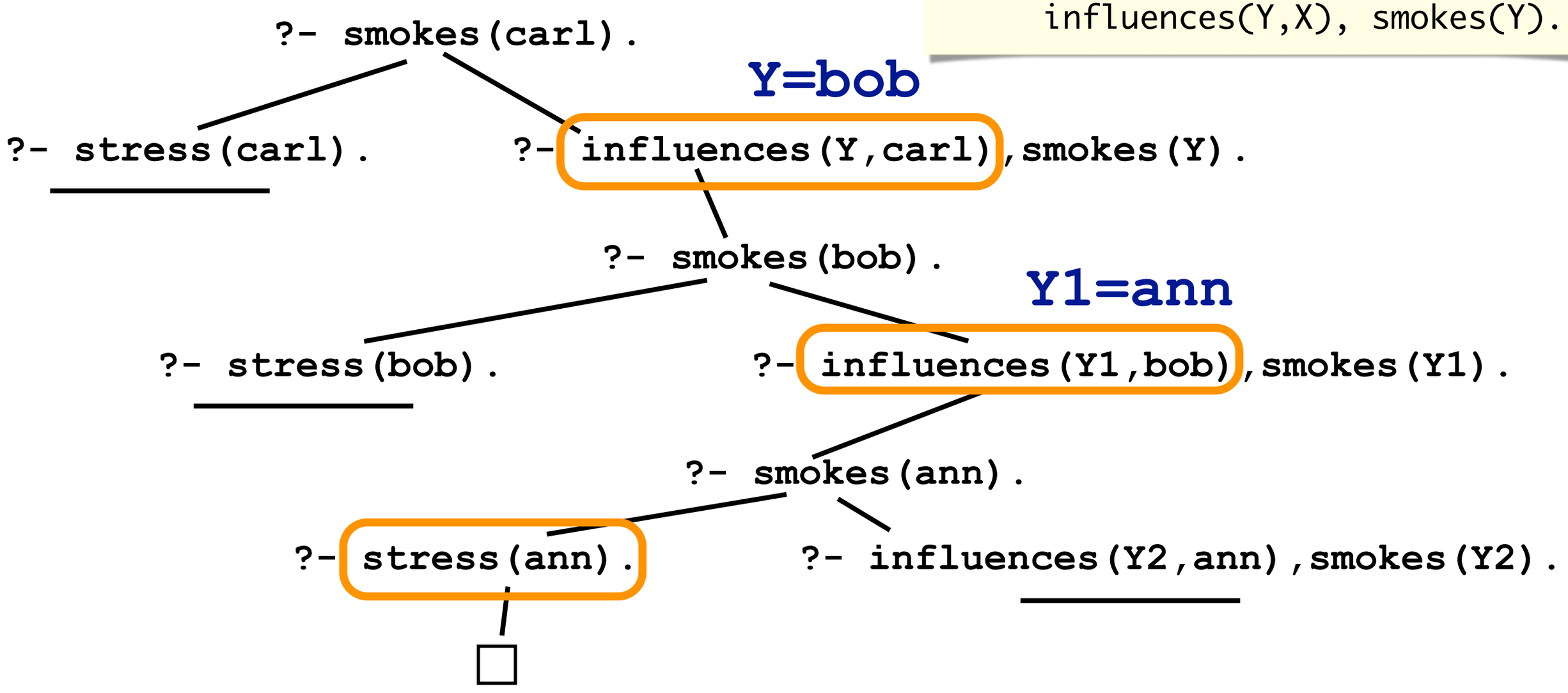


proof = facts used in successful derivation:

influences(bob,carl) &influences(ann,bob) &stress(ann)

Proofs in ProbLog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

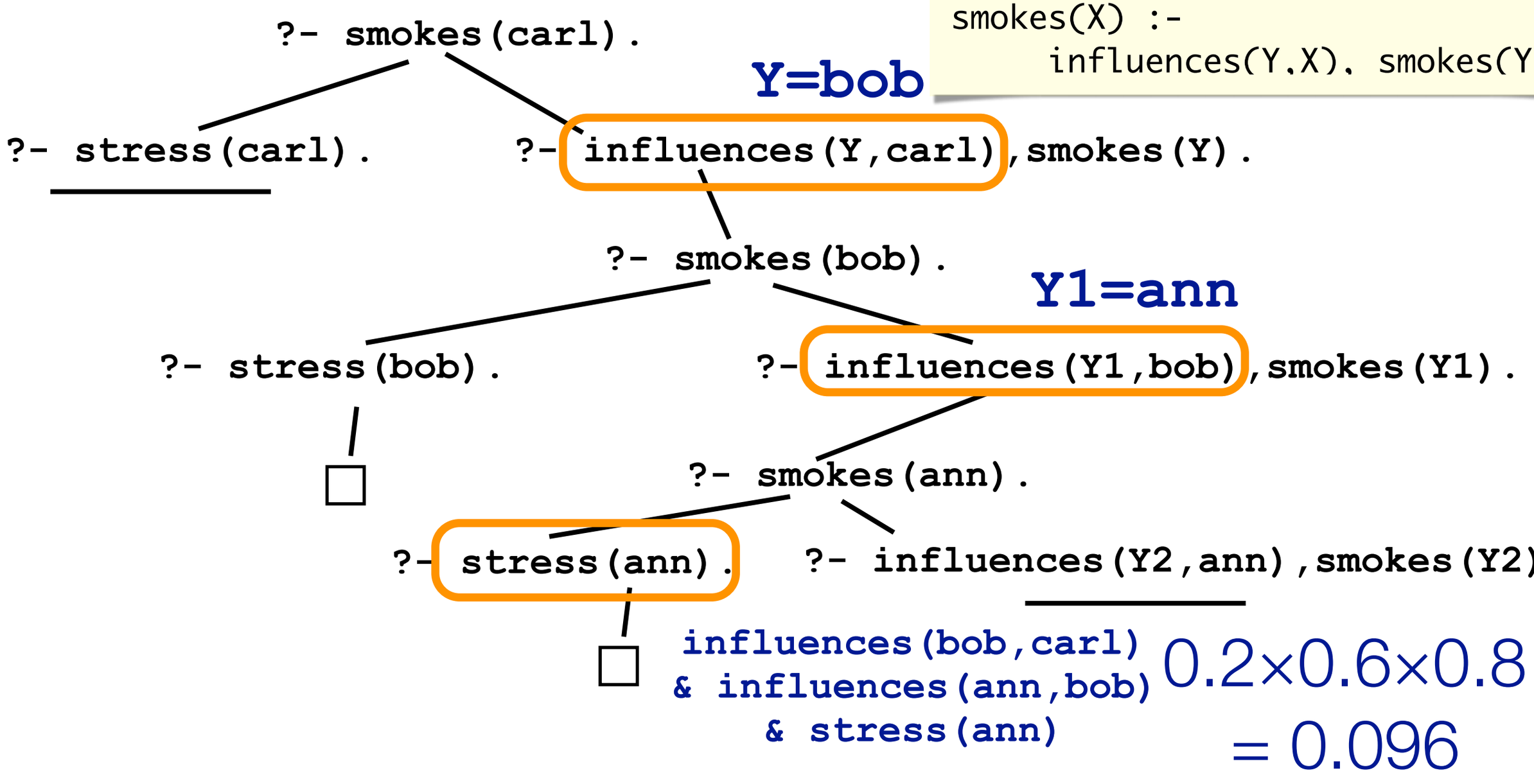


influences(bob,carl)&influences(ann,bob)&stress(ann)

probability of proof = 0.2 × 0.6 × 0.8 = 0.096

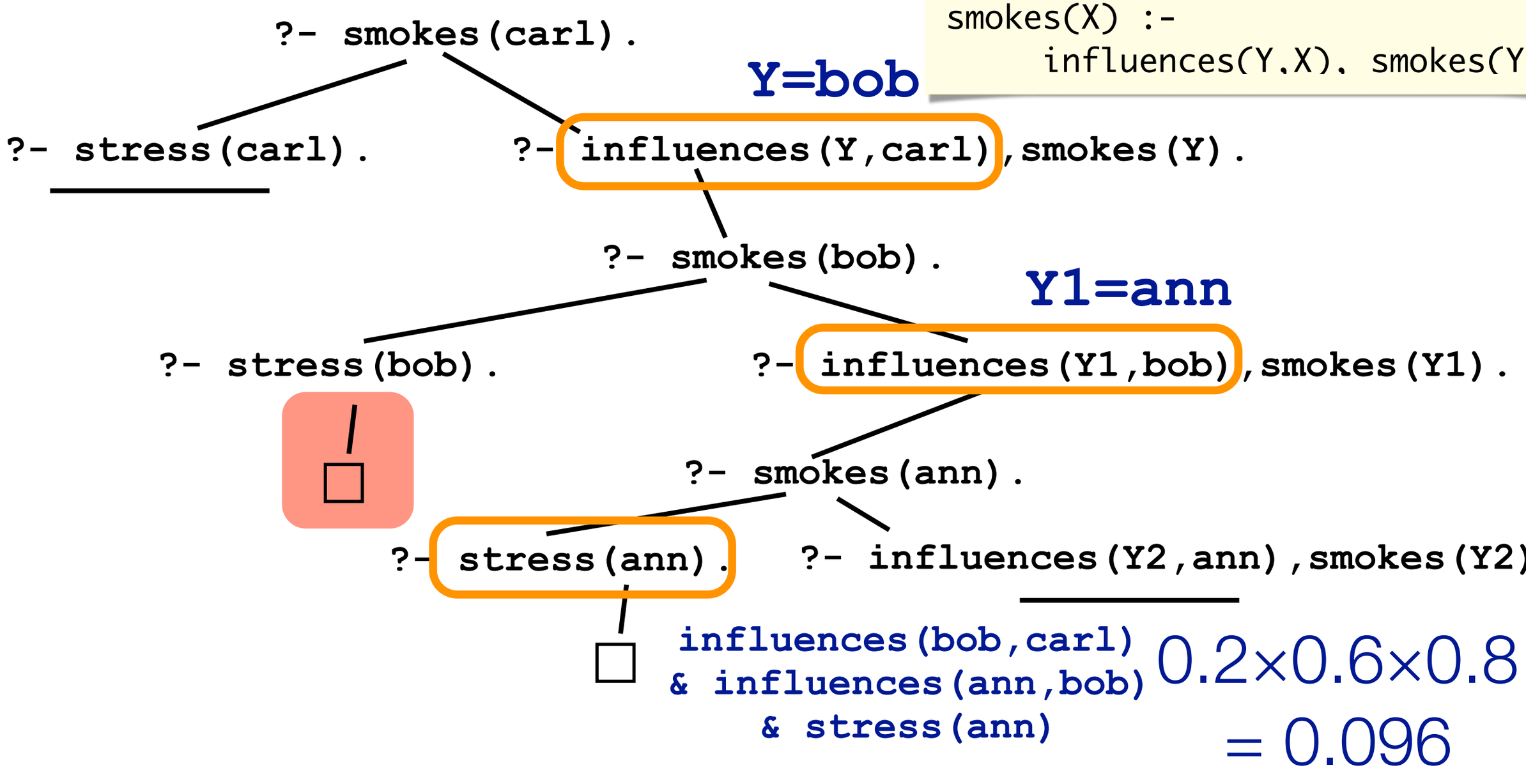
Proofs in ProbLog

```
stress(ann).
stress(bob).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



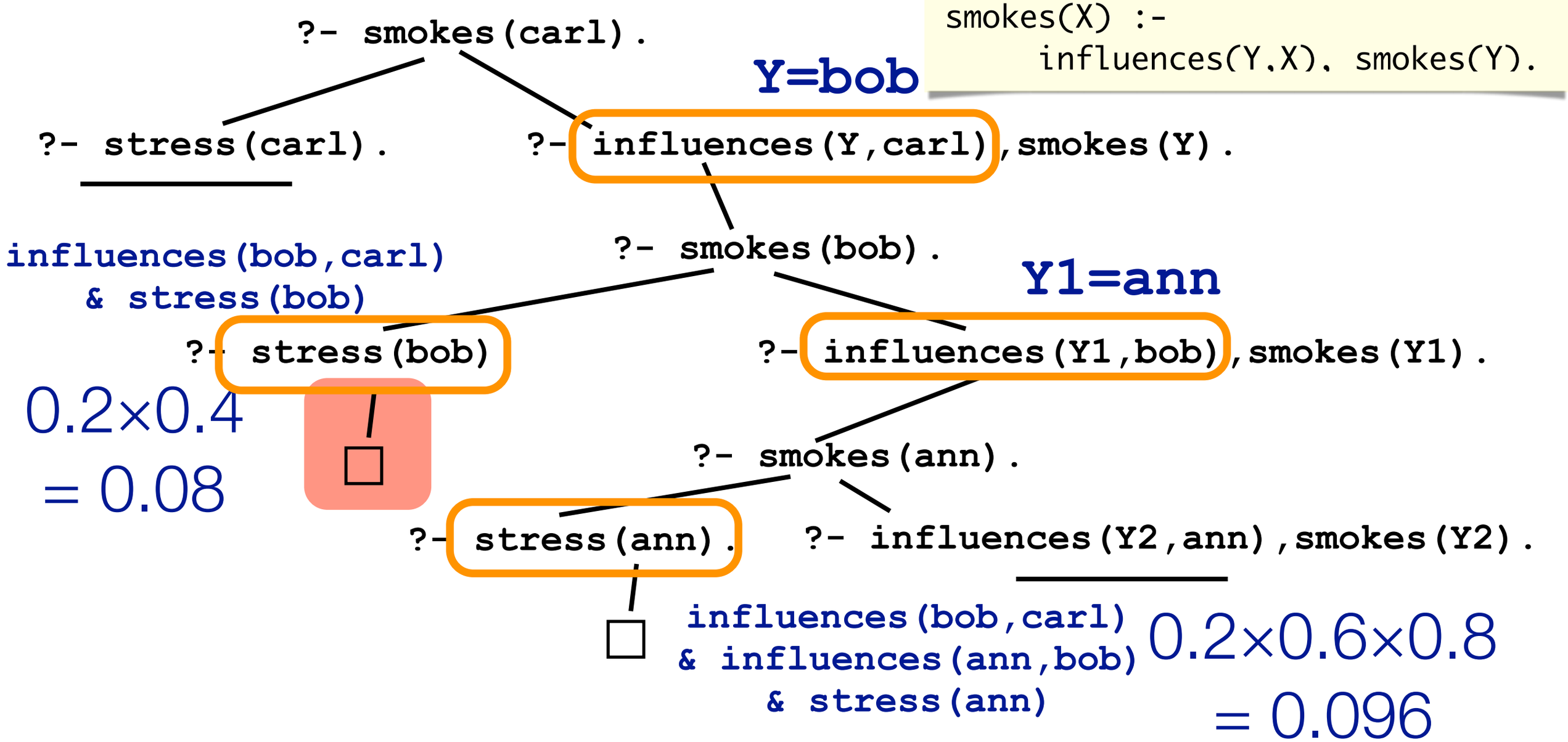
Proofs in ProbLog

```
stress(ann).
stress(bob).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



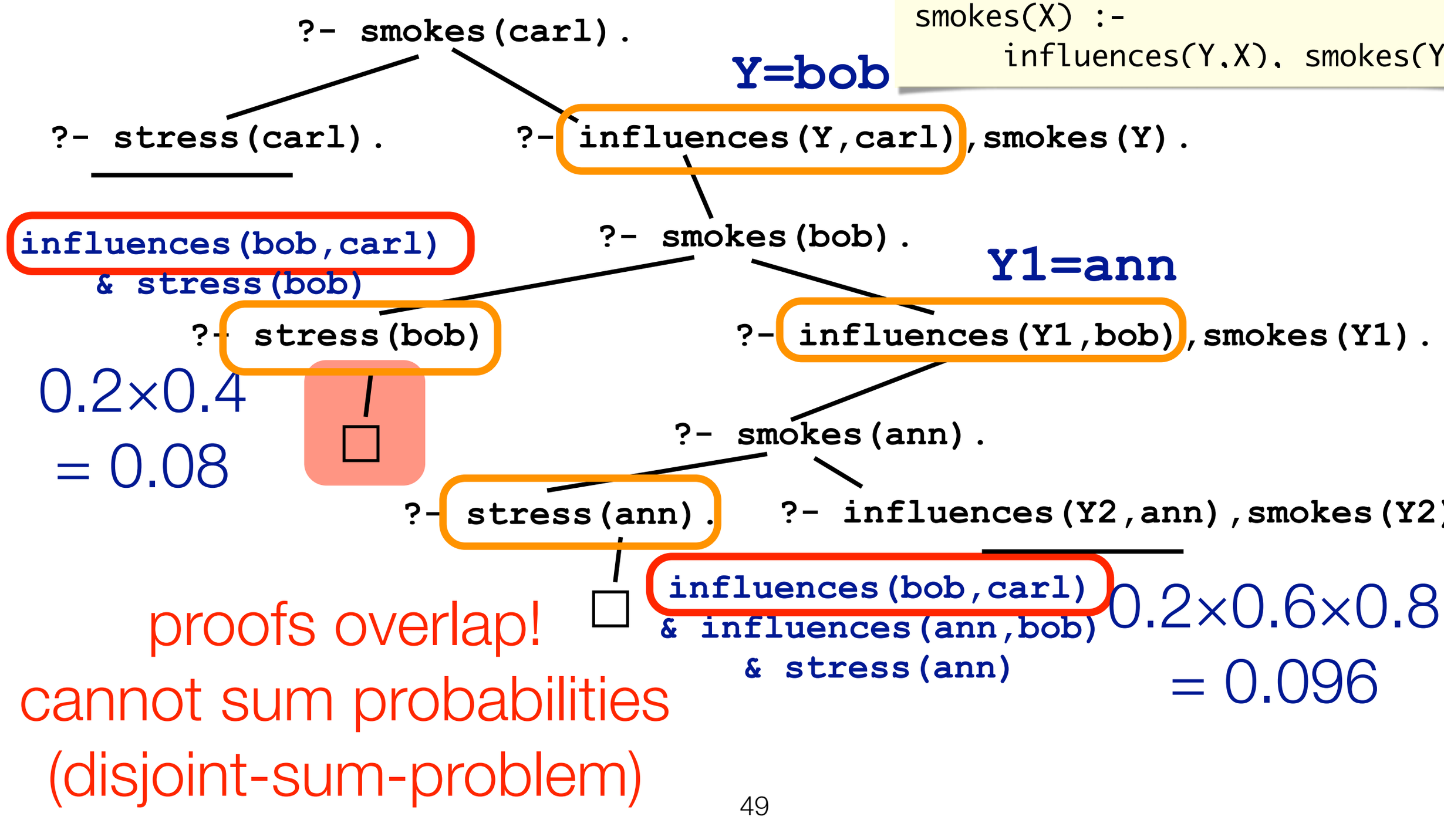
Proofs in ProbLog

```
stress(ann).
stress(bob).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



Proofs in ProbLog

```
stress(ann).
stress(bob).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```



Disjoint-Sum-Problem

possible worlds

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)

infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...
```

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)

infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)

infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)

infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)

...
```

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)
```

... **influences(bob,carl) & stress(bob)**

Disjoint-Sum-Problem

possible worlds

influences(bob,carl) &
influences(ann,bob) & stress(ann)

```
infl(bob,carl) &   infl(ann,bob) &   st(ann) & \+st(bob)
infl(bob,carl) &   infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) &   st(ann) &   st(bob)
infl(bob,carl) &   infl(ann,bob) & \+st(ann) &   st(bob)
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) &   st(bob)
```

... **influences(bob,carl) & stress(bob)**

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

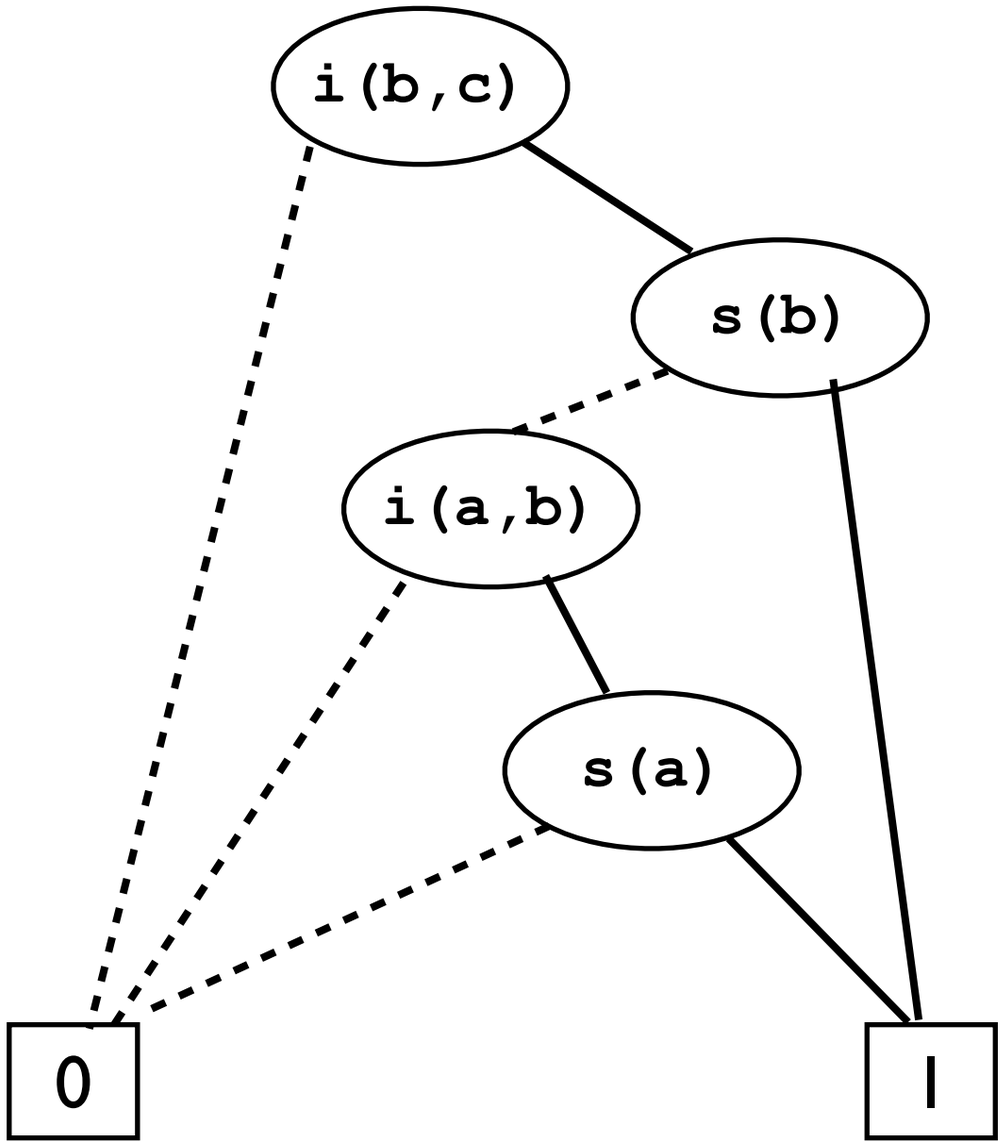
influences(bob,carl) &
influences(ann,bob) & stress(ann)

infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)	0.0576
infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)	0.0384
infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)	0.0256
infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)	0.0096
infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)	0.0064
... **influences(bob,carl) & stress(bob)**	∑ = 0.1376

sum of proof probabilities: 0.096+0.08 = 0.1760

Binary Decision Diagrams
[Bryant 86]

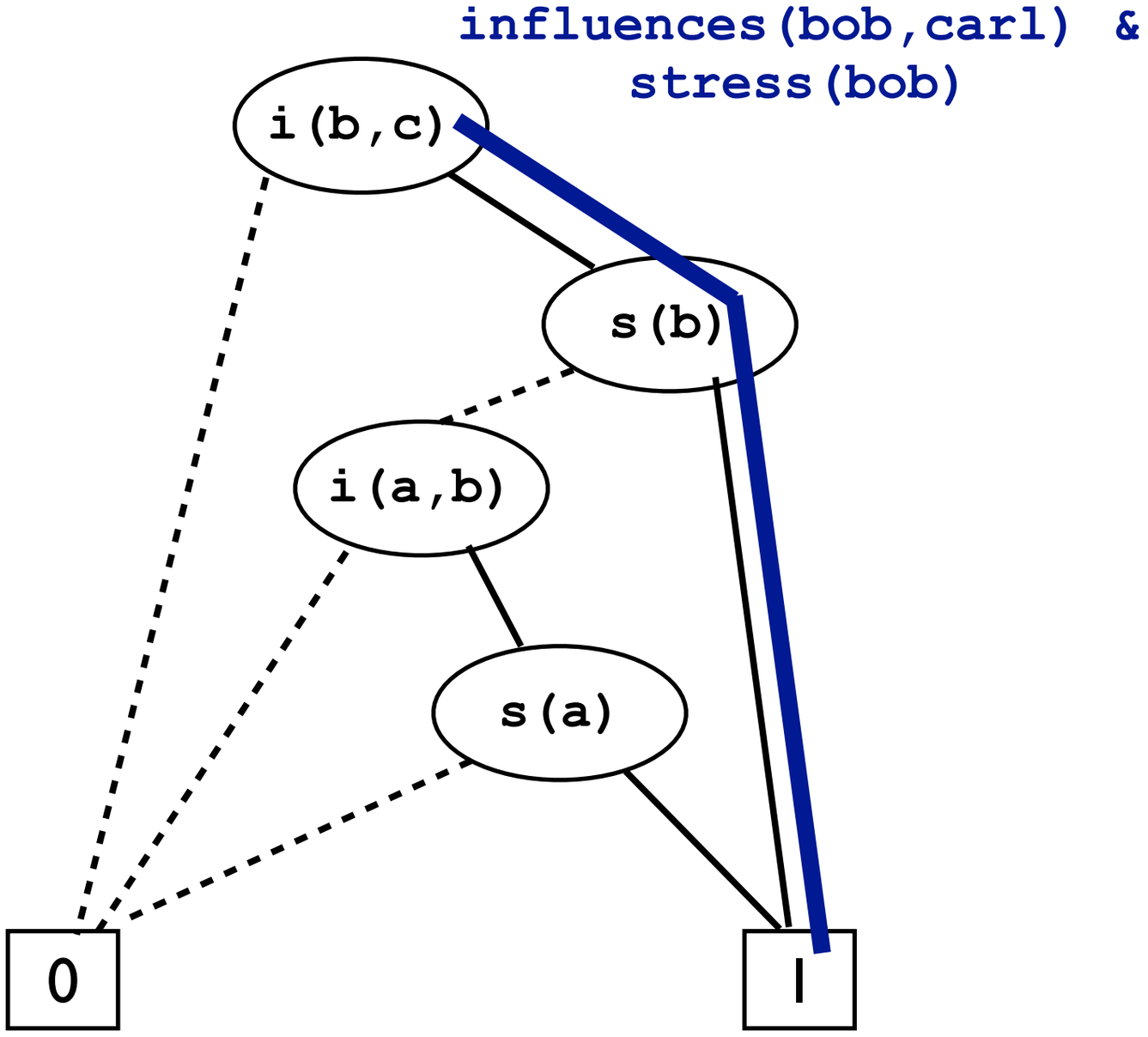
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

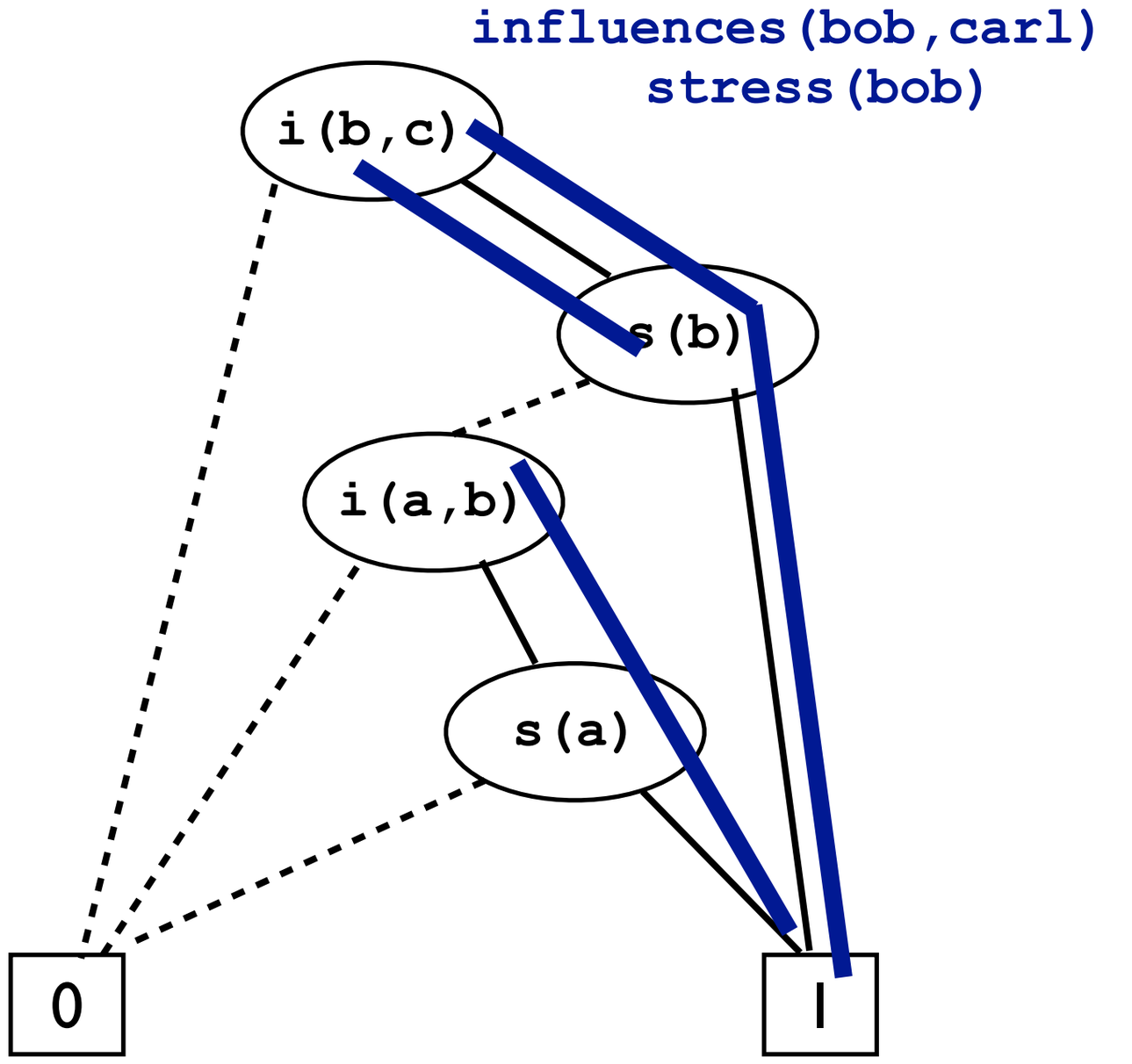
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

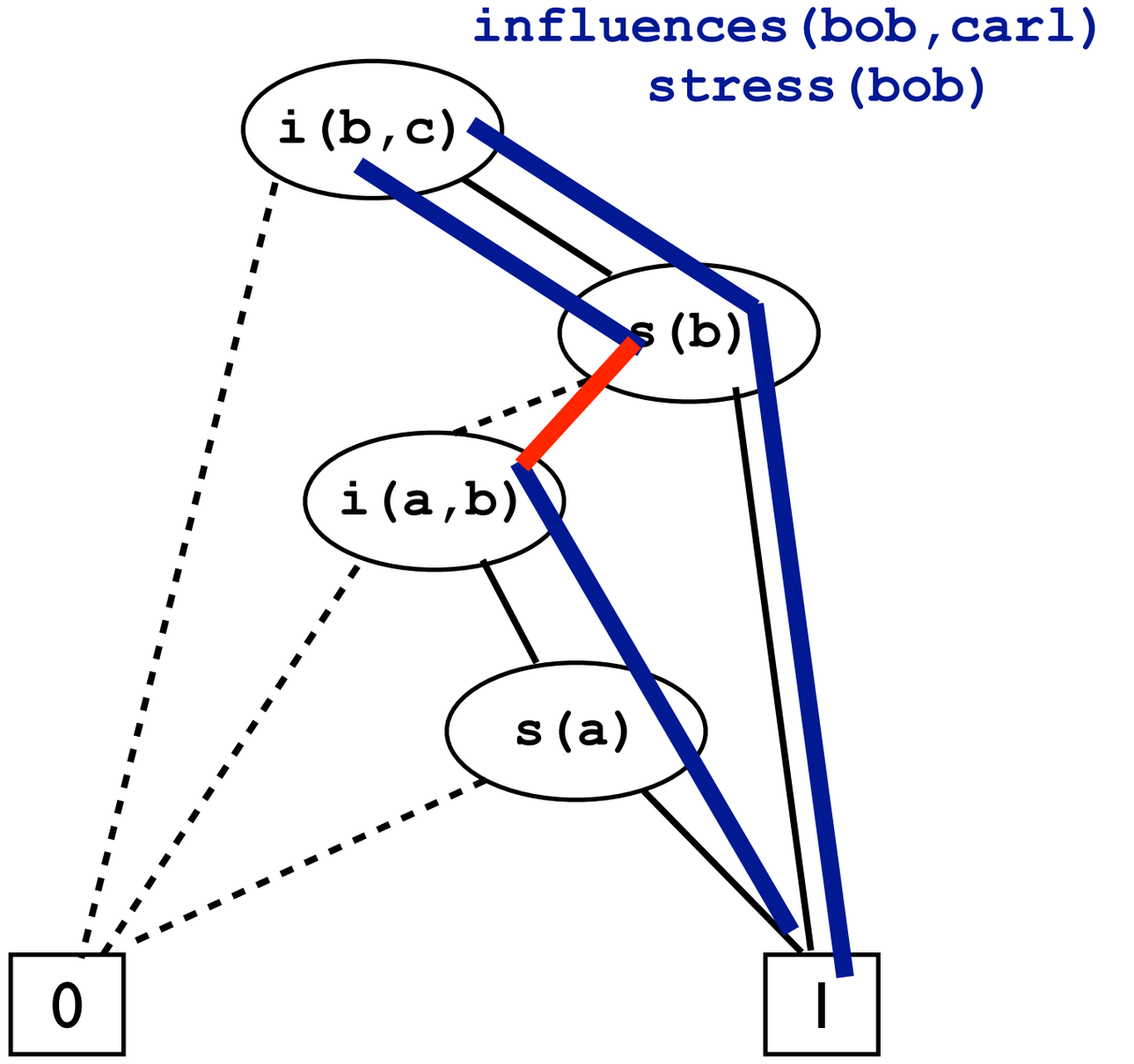
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



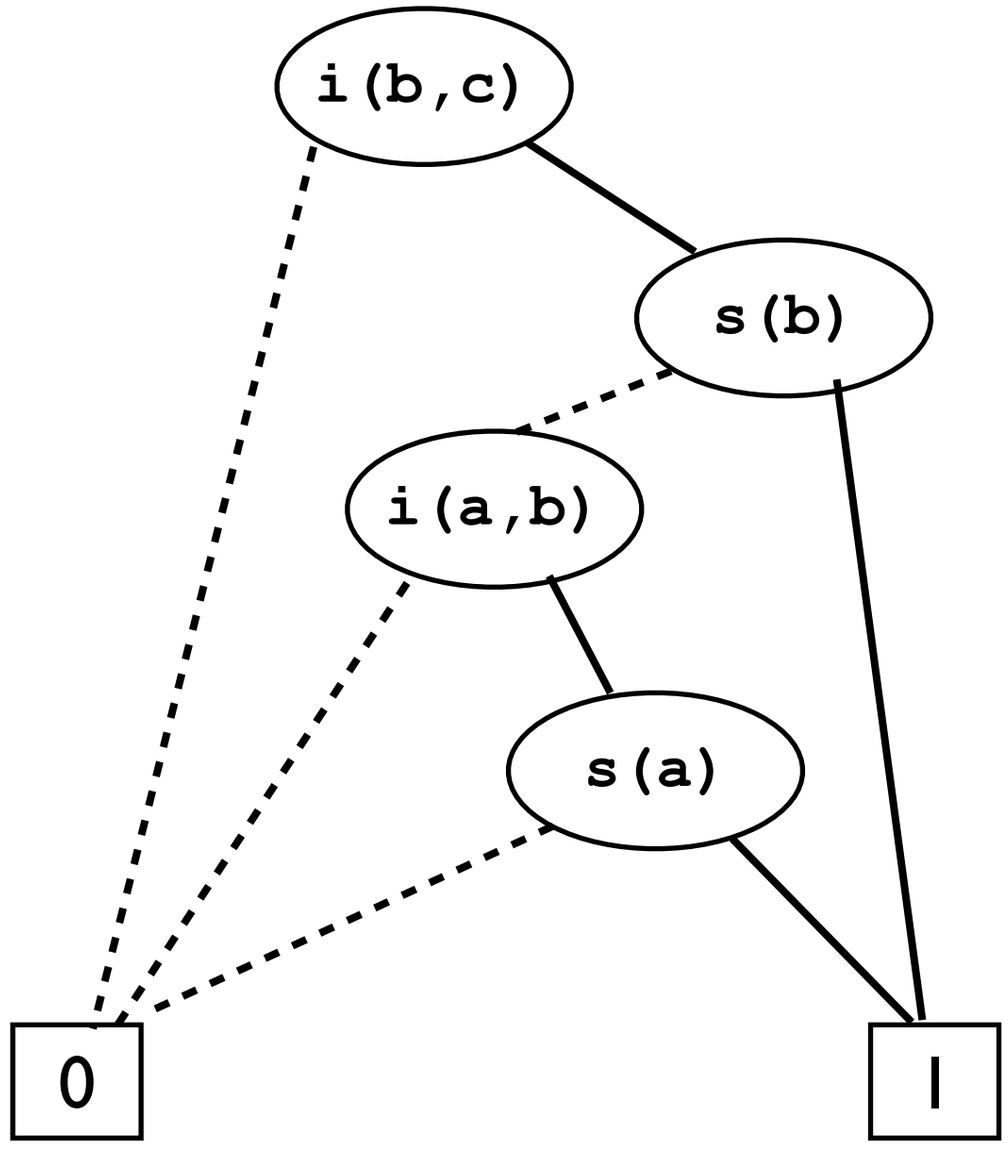
Binary Decision Diagrams

[Bryant 86]

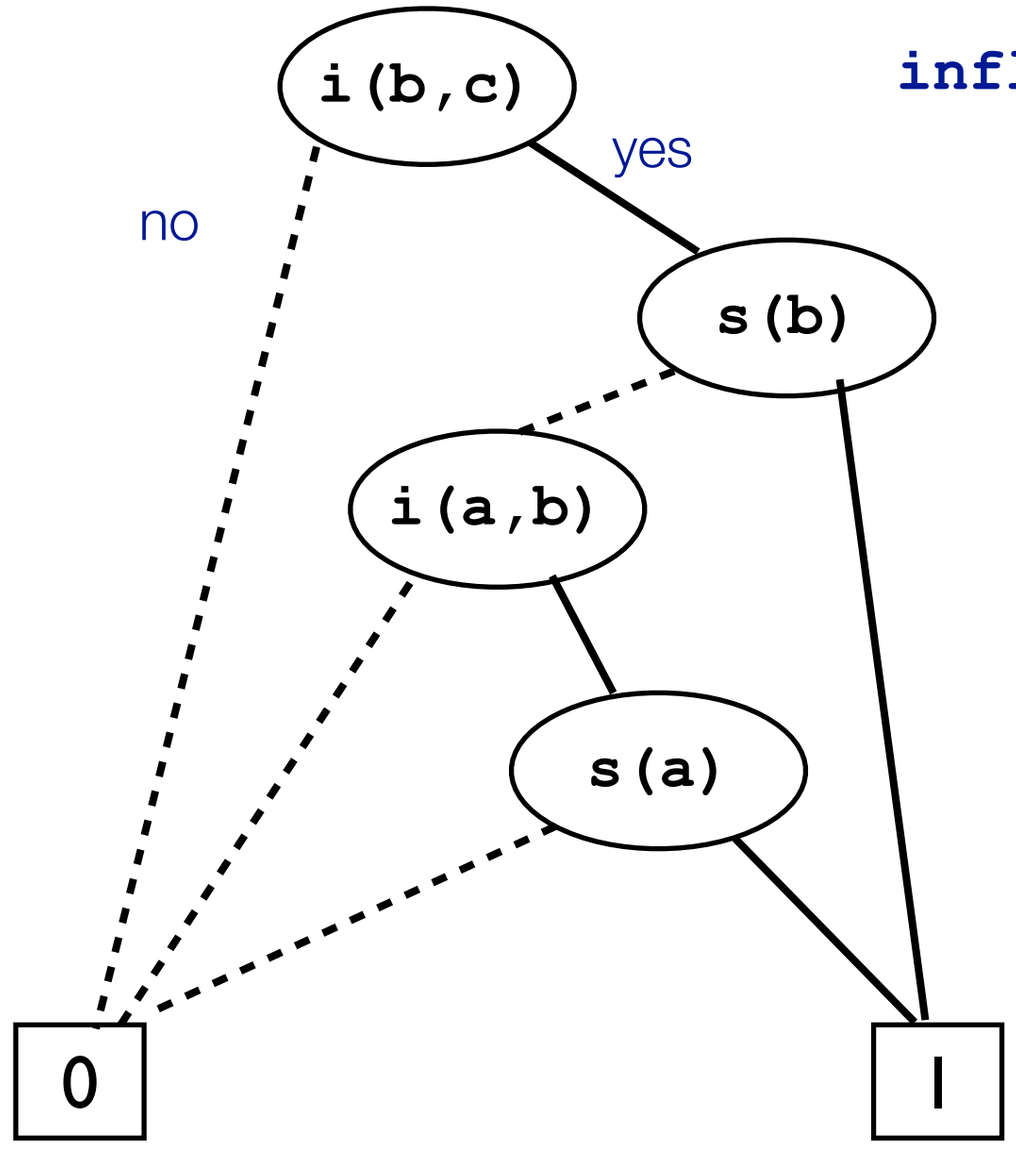
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

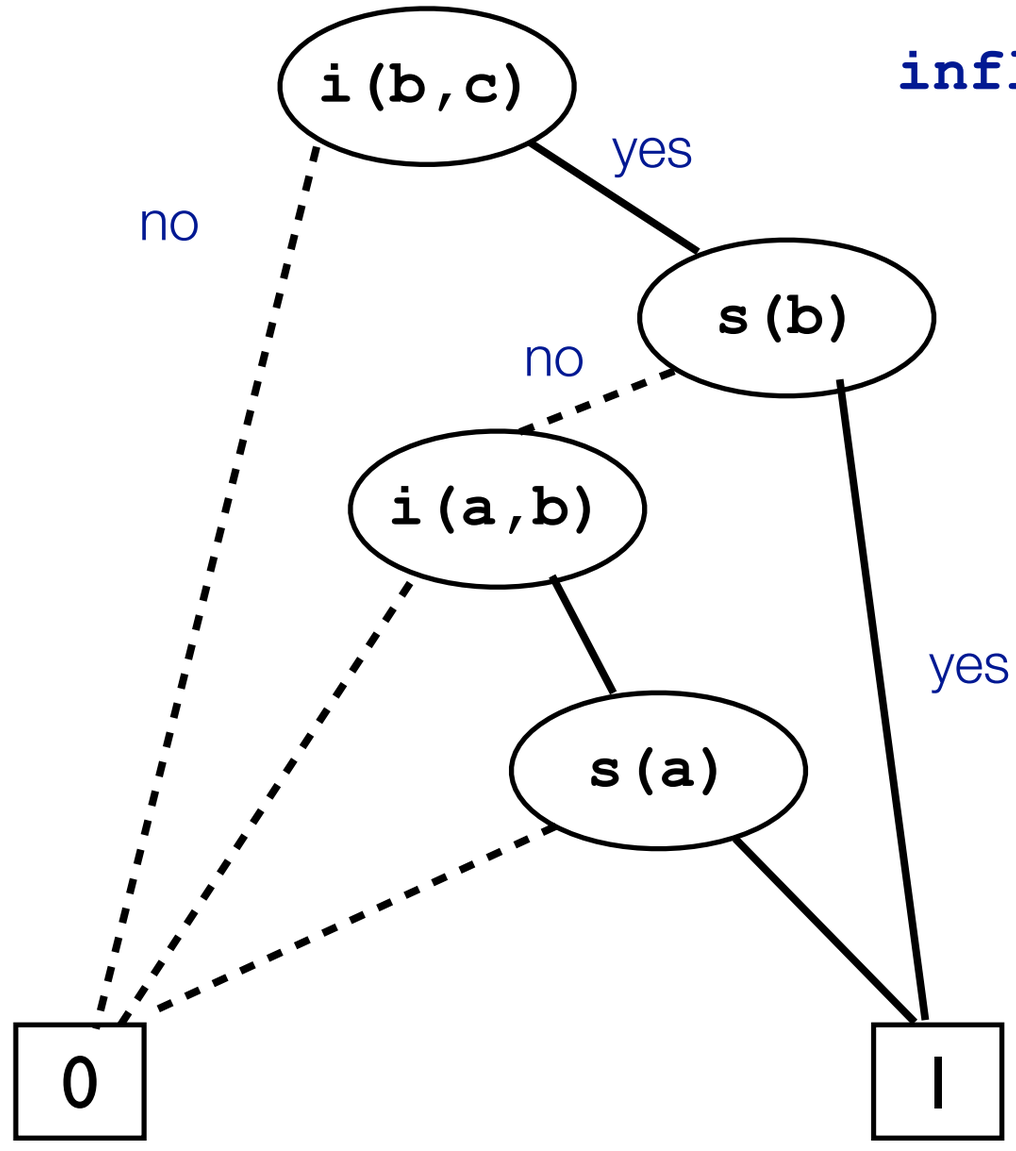


Binary Decision Diagrams



influences(bob,carl)?

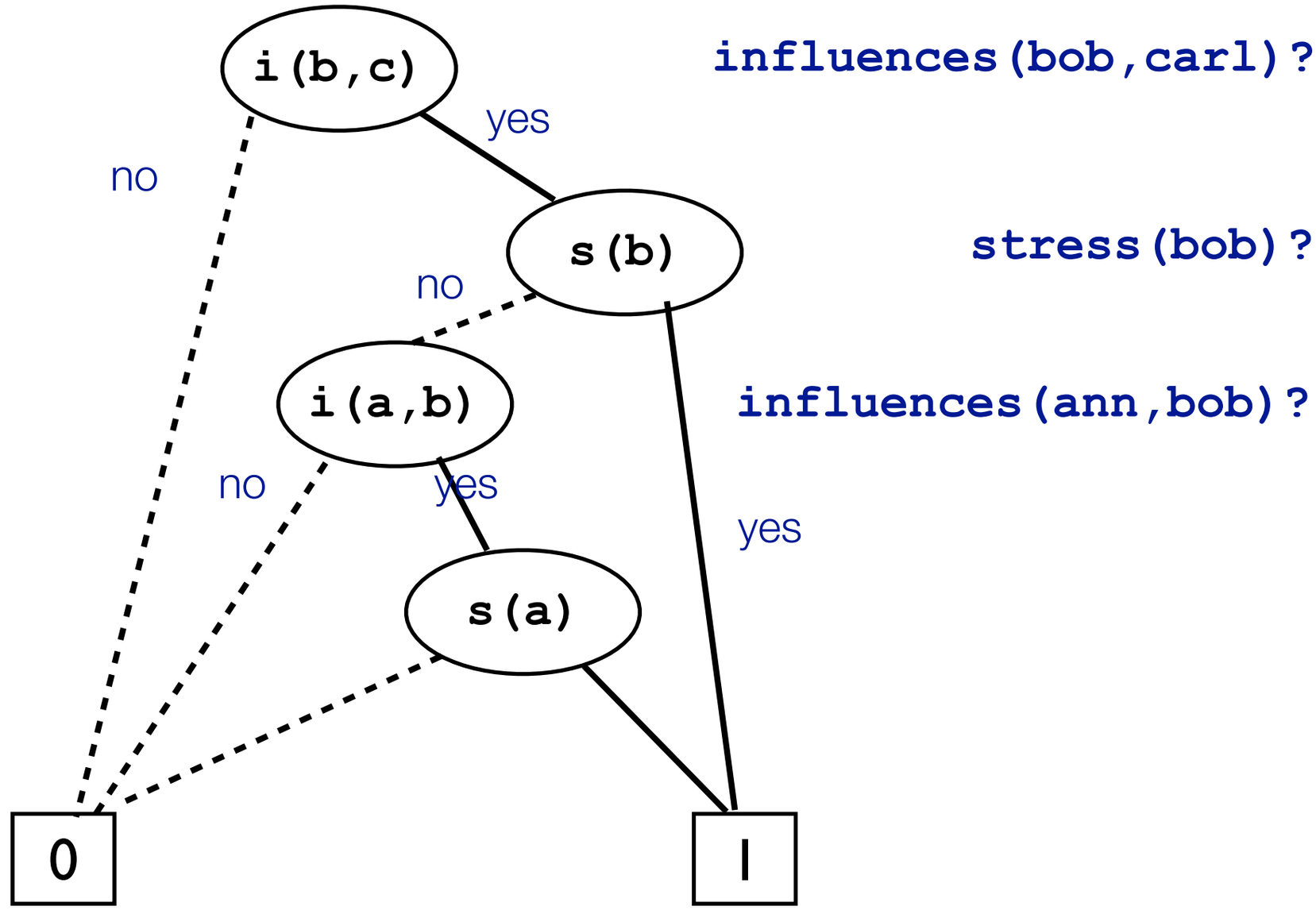
Binary Decision Diagrams



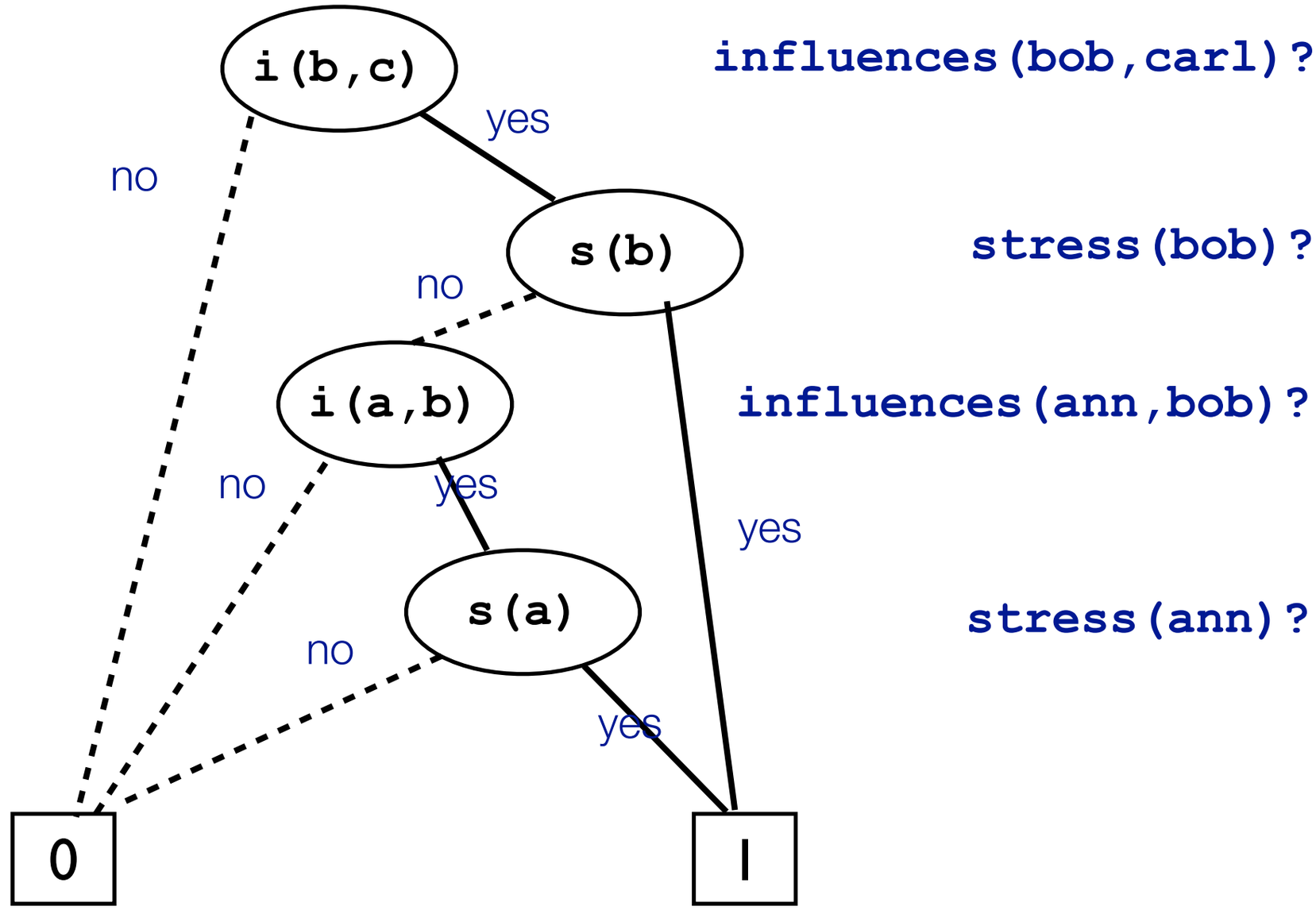
influences(bob,carl)?

stress(bob)?

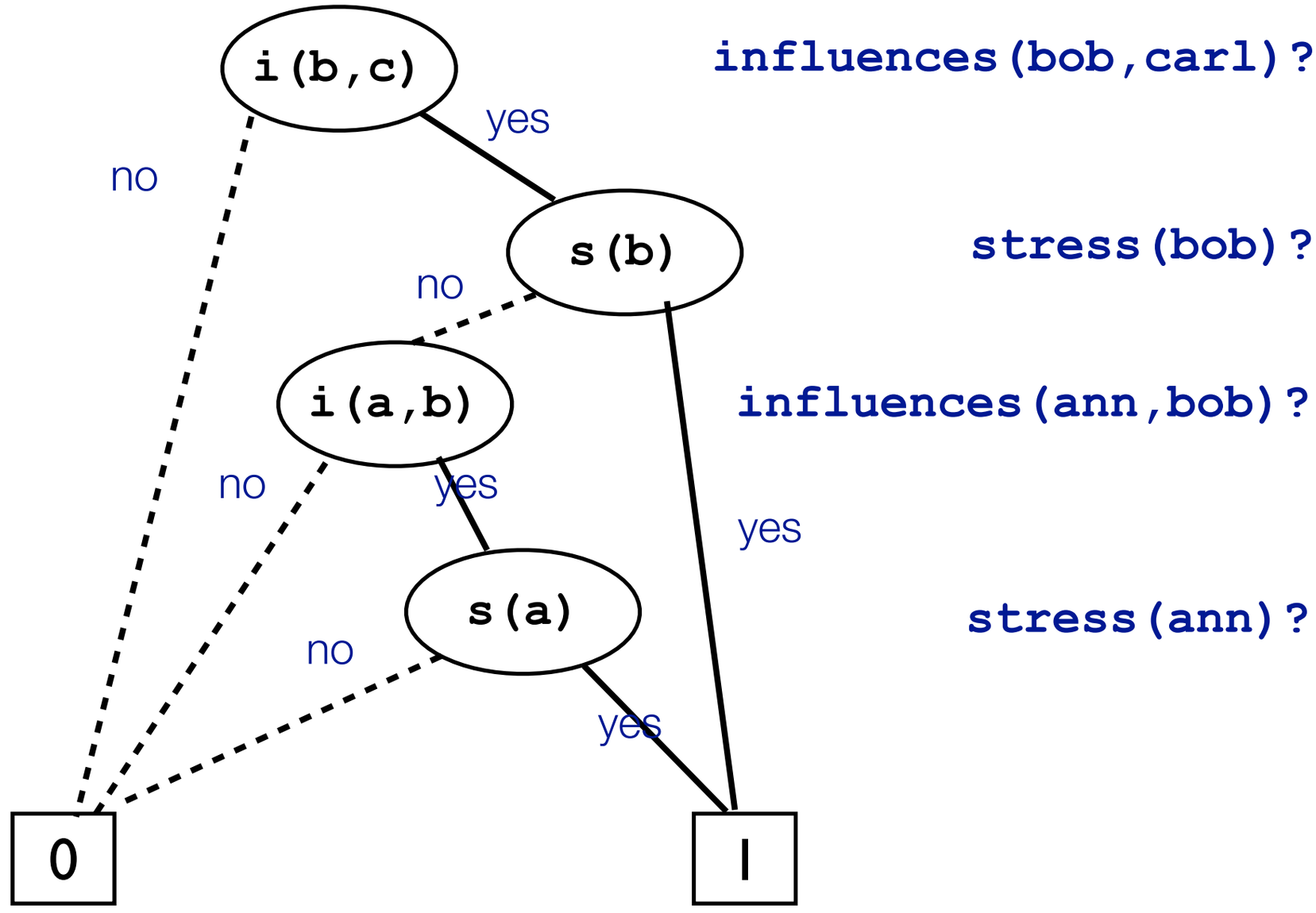
Binary Decision Diagrams



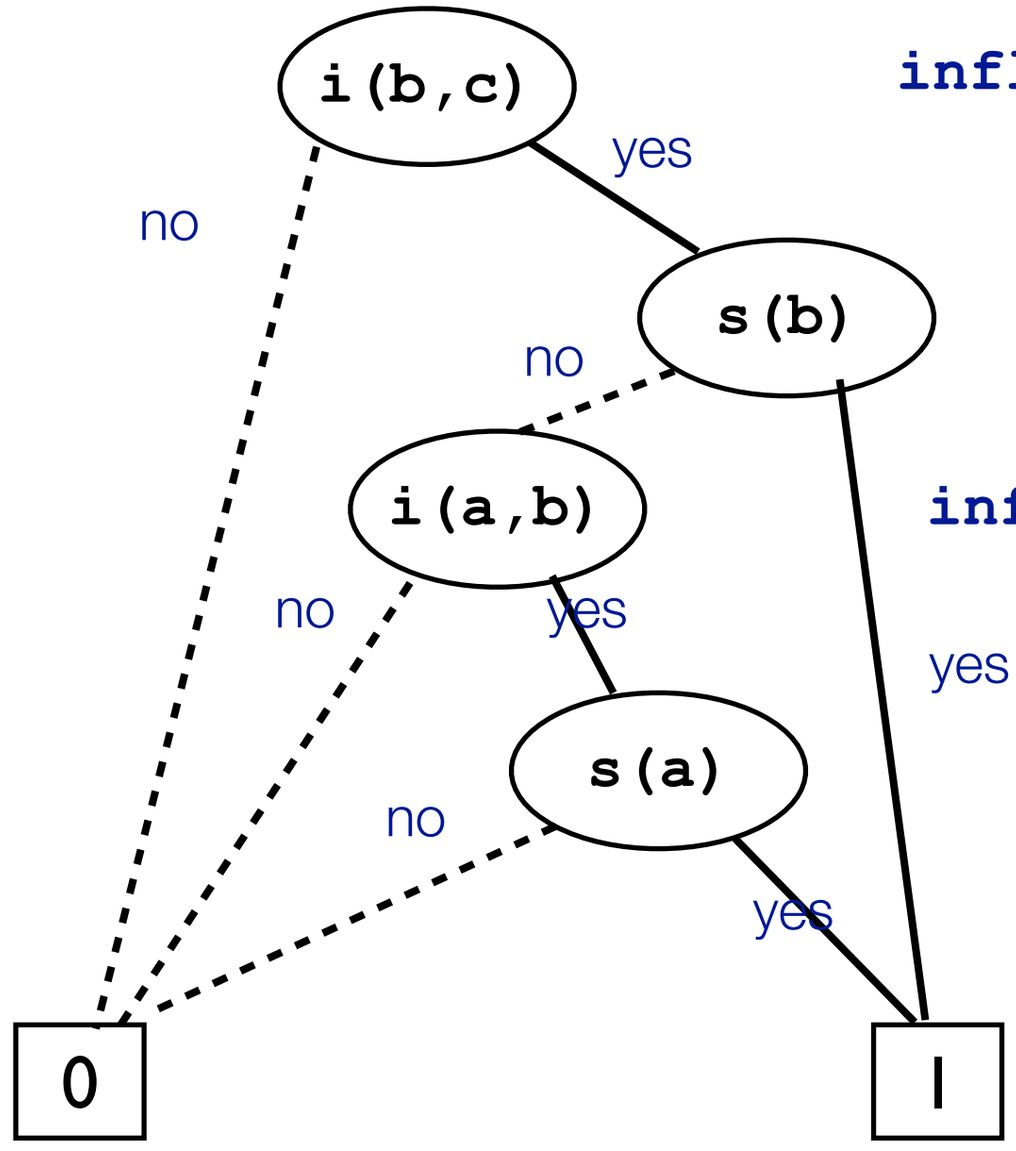
Binary Decision Diagrams



Binary Decision Diagrams



Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

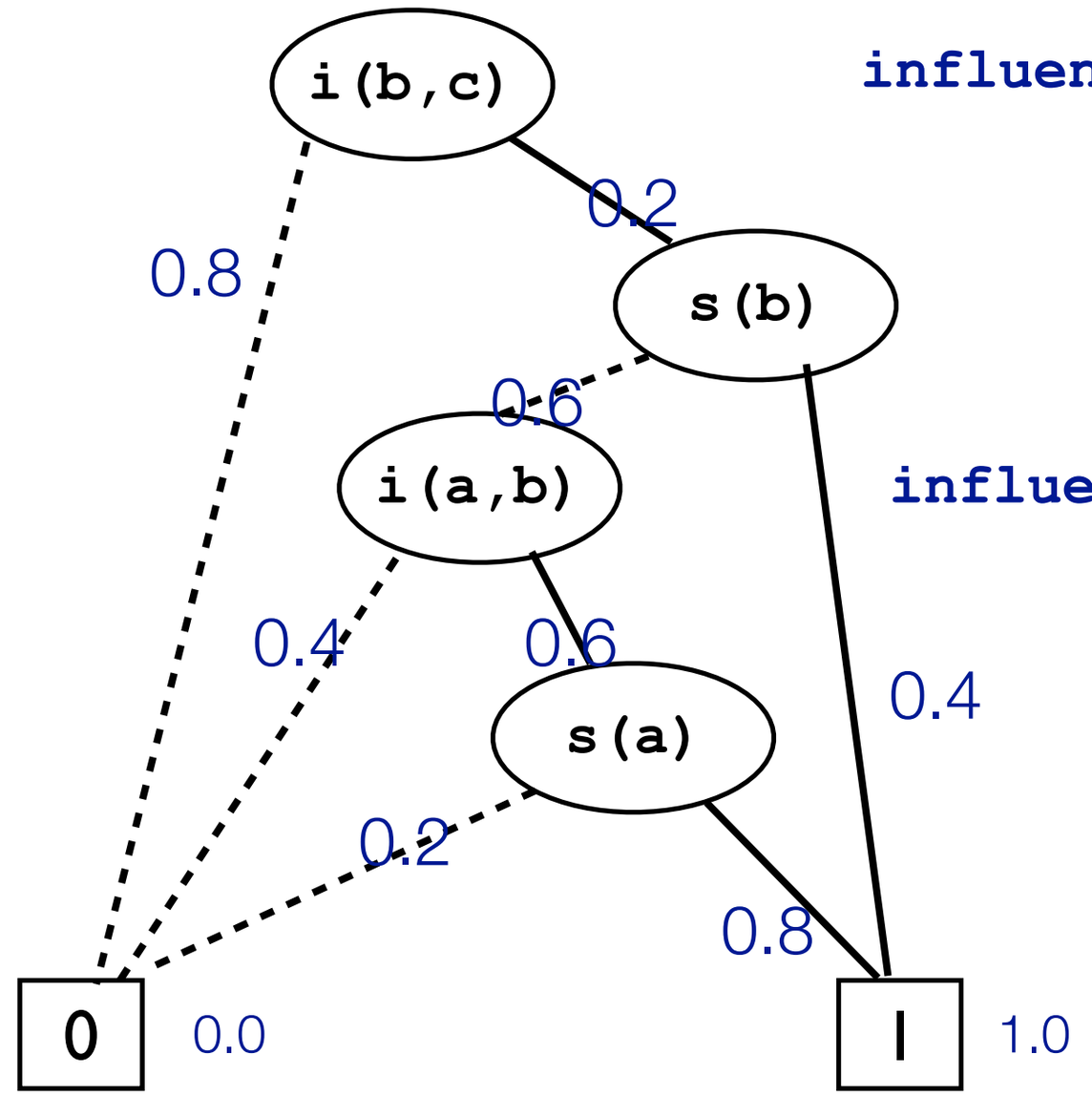
stress(ann)?

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

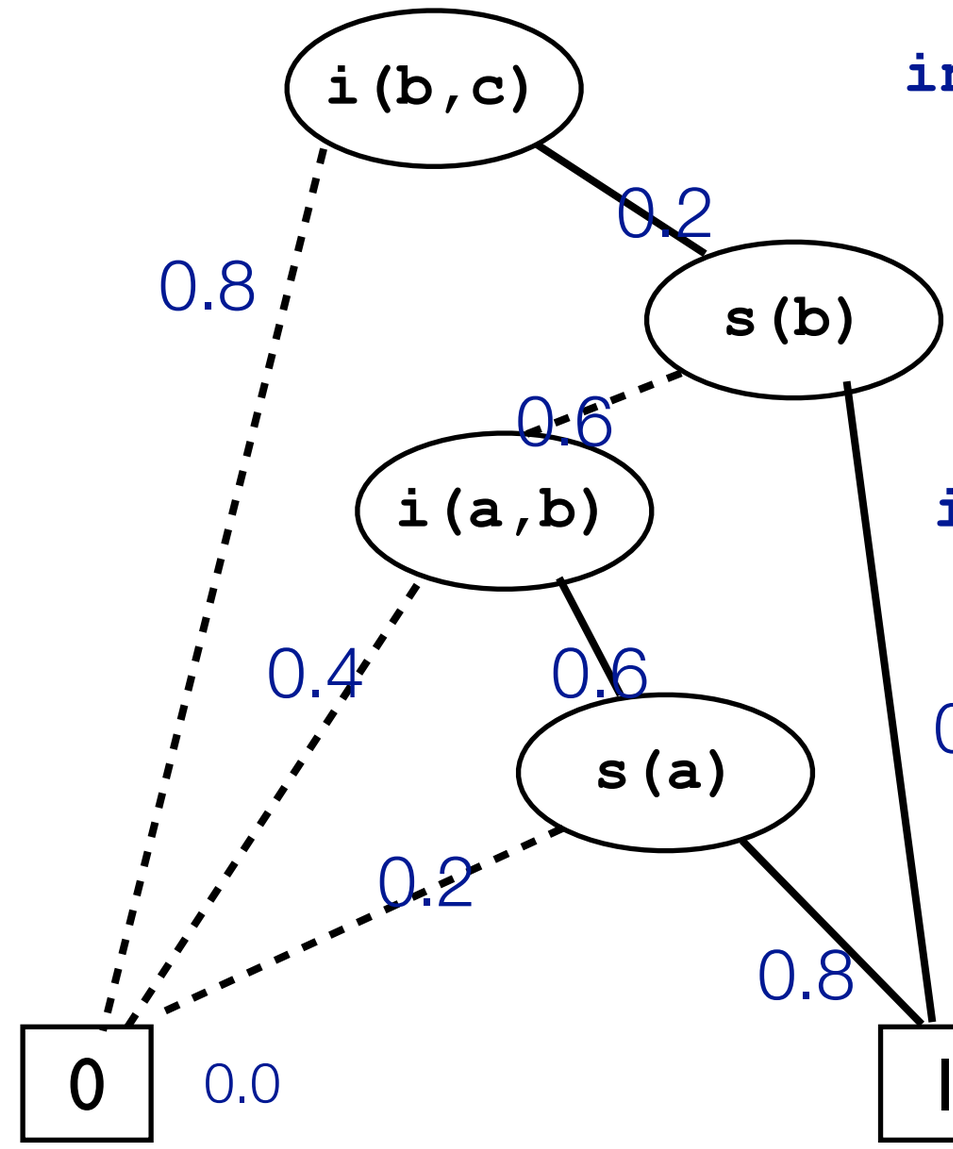
stress(ann)?

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

0.4

stress(ann)?

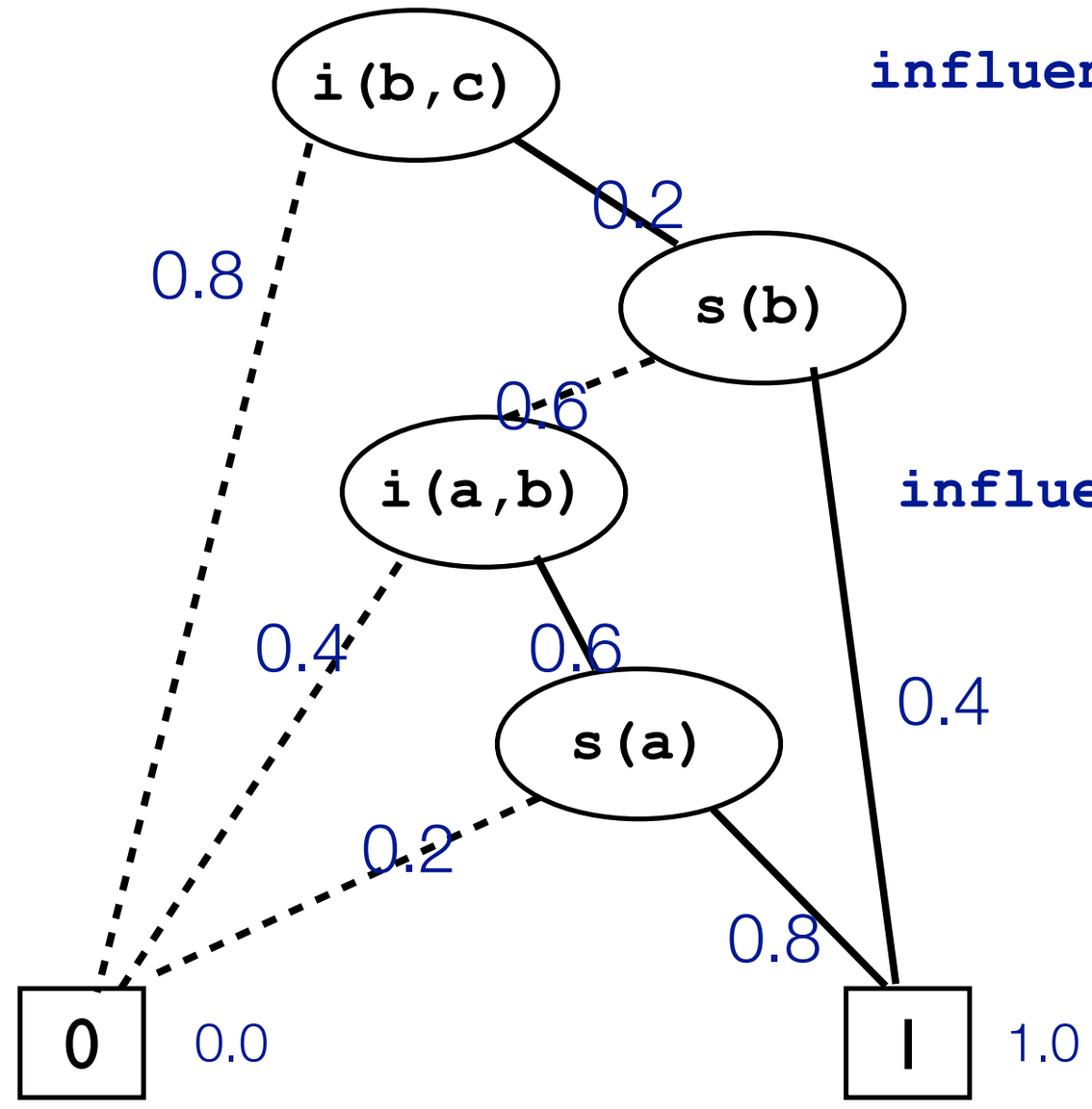
0.2×0.0+0.8×1.0=0.8

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

stress(ann)?

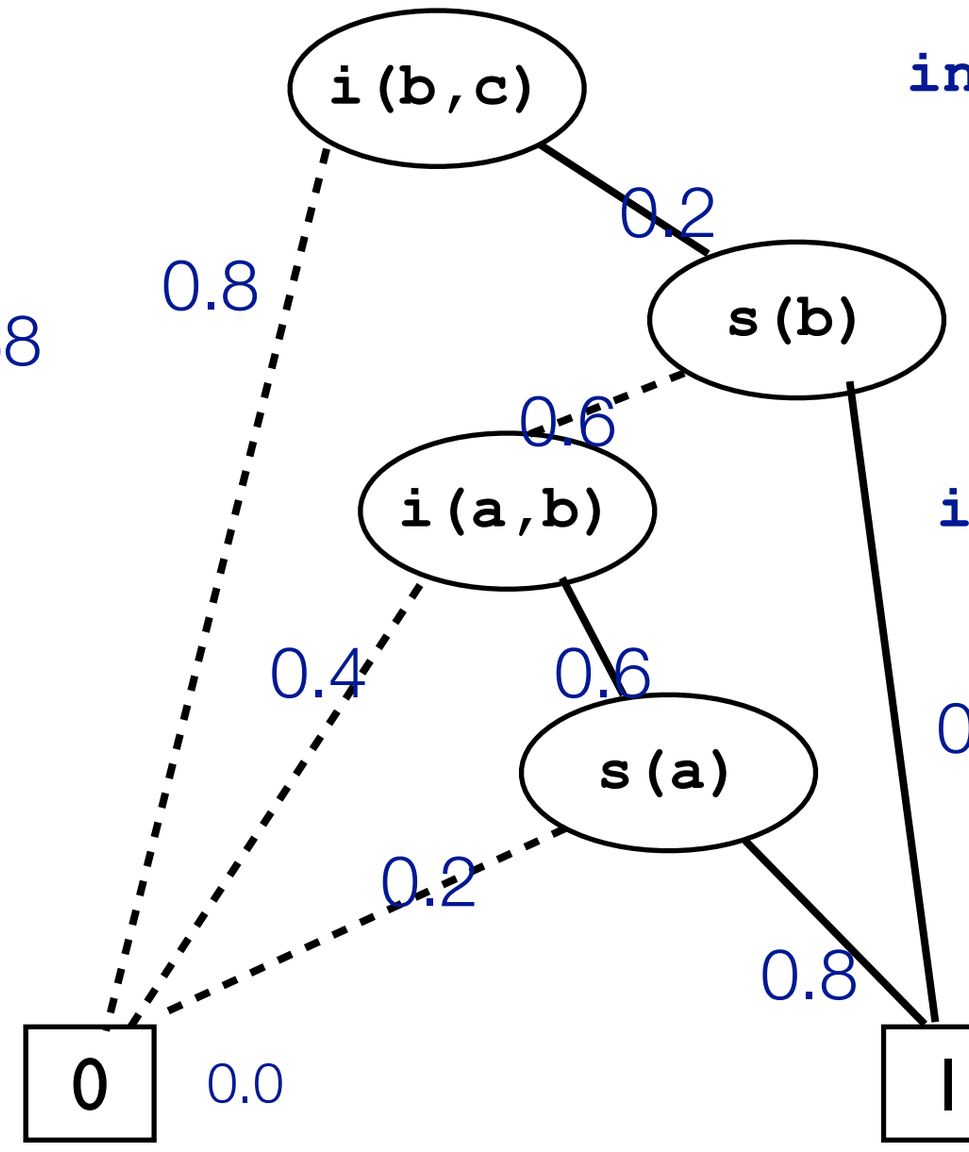
0.4×0.0+0.6×0.8=0.48

0.2×0.0+0.8×1.0=0.8

probability of **smokes(c)**?

smokes(c) = i(b,c)&s(b) v i(b,c)&i(a,b)&s(a)

Binary Decision Diagrams



influences(bob,carl)?

0.6×0.48+0.4×1.0=0.688

stress(bob)?

0.4×0.0+0.6×0.8=0.48

influences(ann,bob)?

0.2×0.0+0.8×1.0=0.8

stress(ann)?

probability of

smokes(c)?

smokes(c) = i(b,c)&s(b) v
i(b,c)&i(a,b)&s(a)

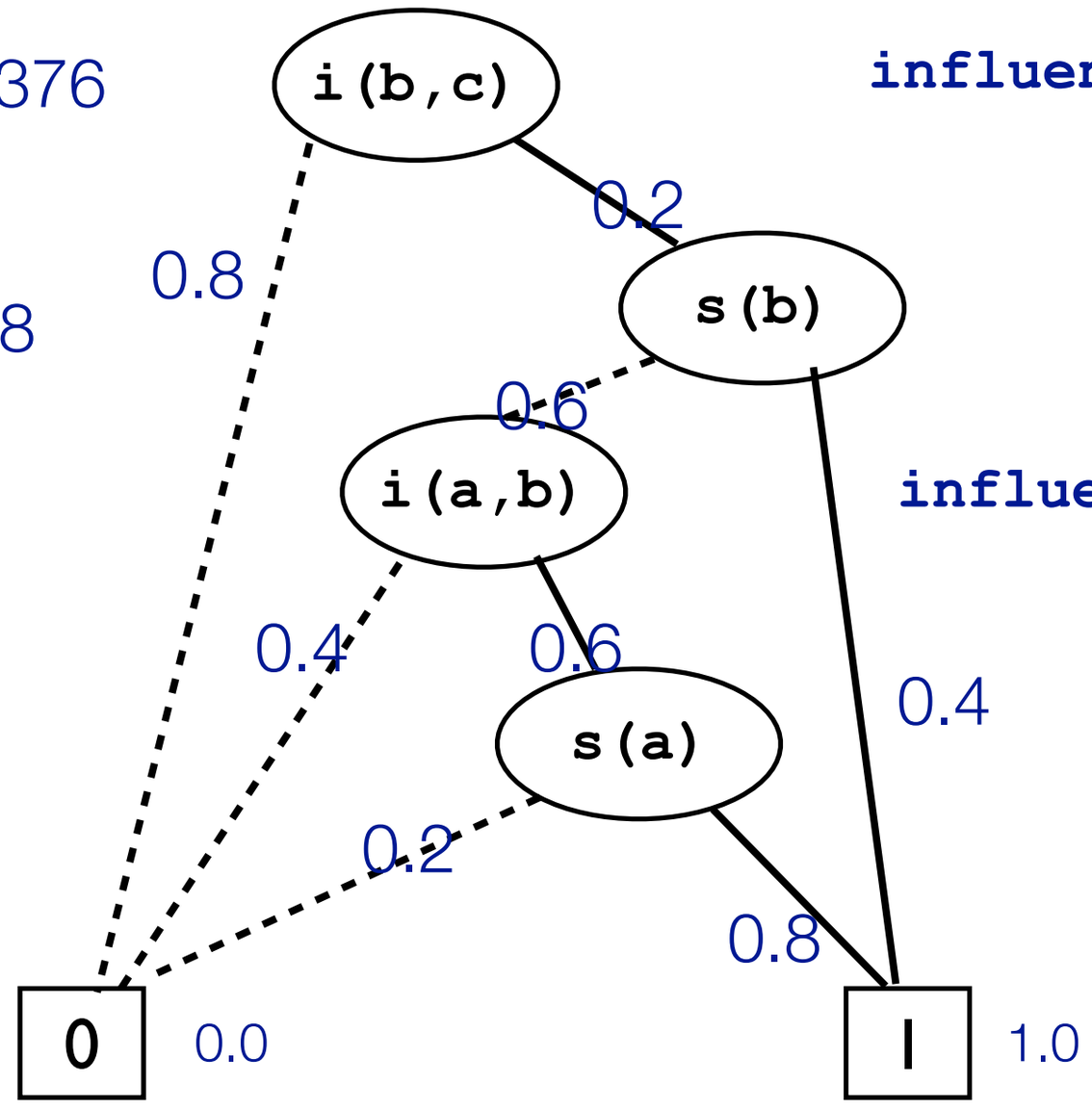
Binary Decision Diagrams

0.8×0.0+0.2×0.688=0.1376

0.6×0.48+0.4×1.0=0.688

0.4×0.0+0.6×0.8=0.48

0.2×0.0+0.8×1.0=0.8



influences(bob,carl)?

stress(bob)?

influences(ann,bob)?

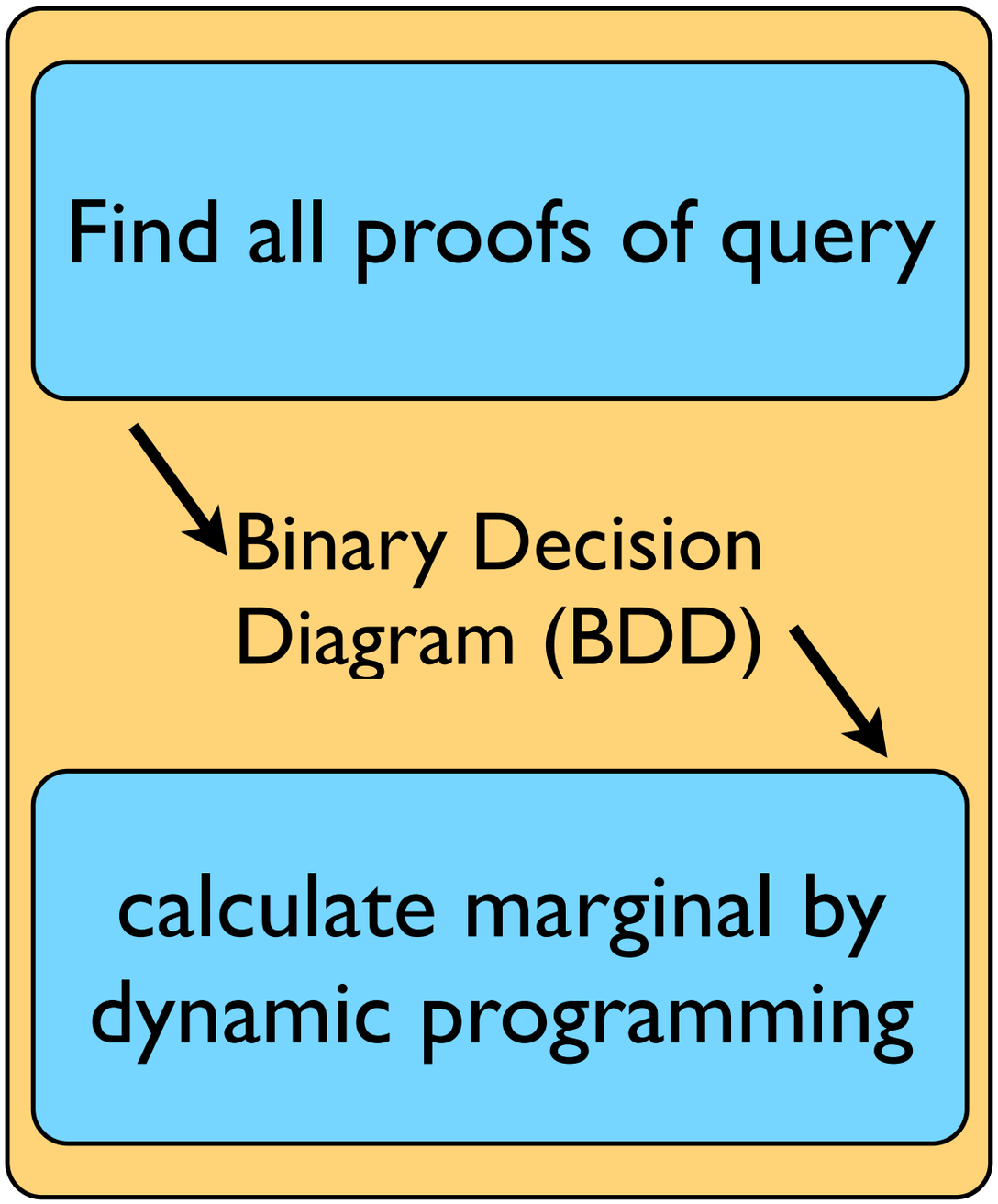
stress(ann)?

probability of **smokes(c)**?

smokes(c) = i(b,c)&s(b) v i(b,c)&i(a,b)&s(a)

Initial Approach

(ProbLog1)

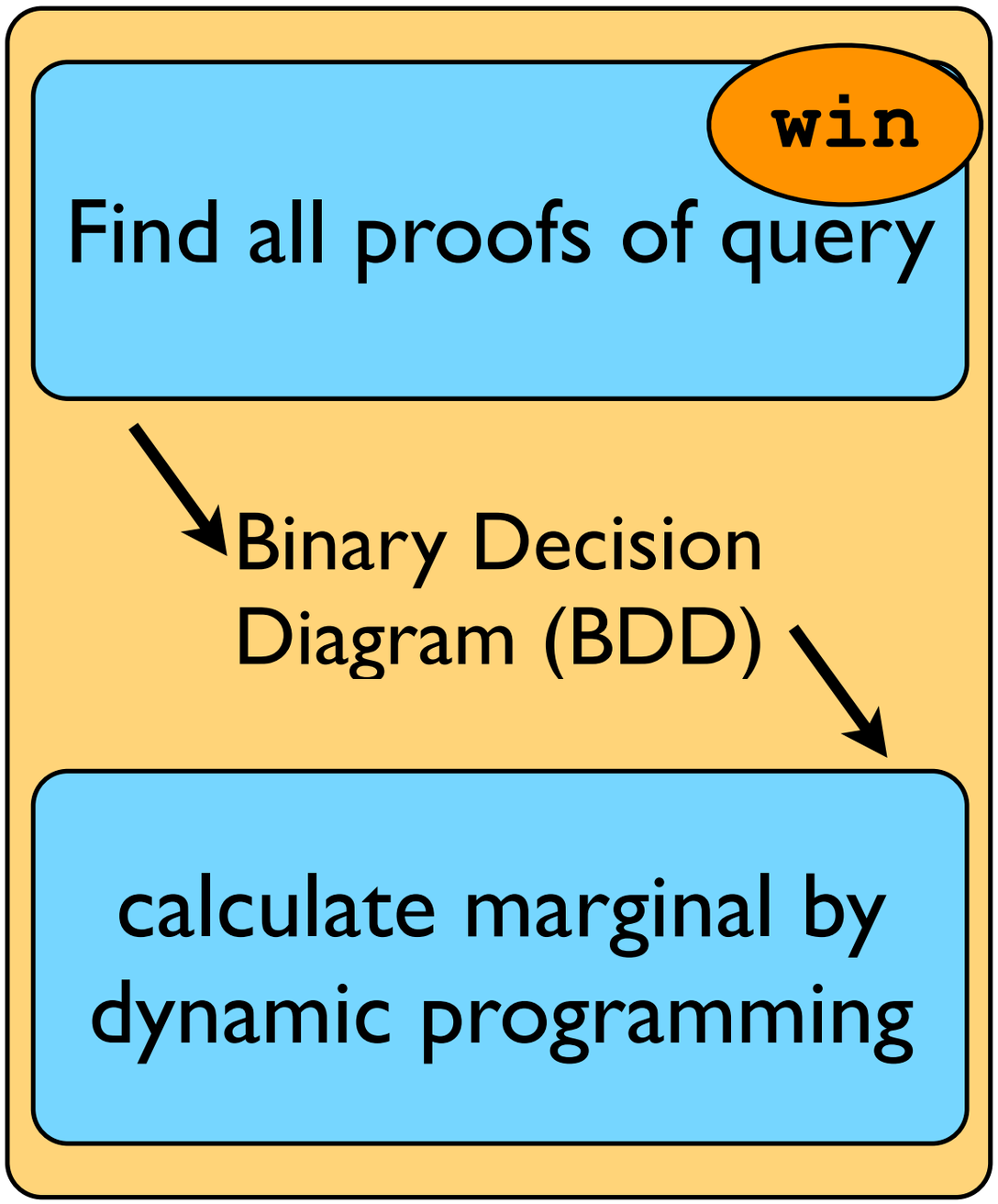


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```

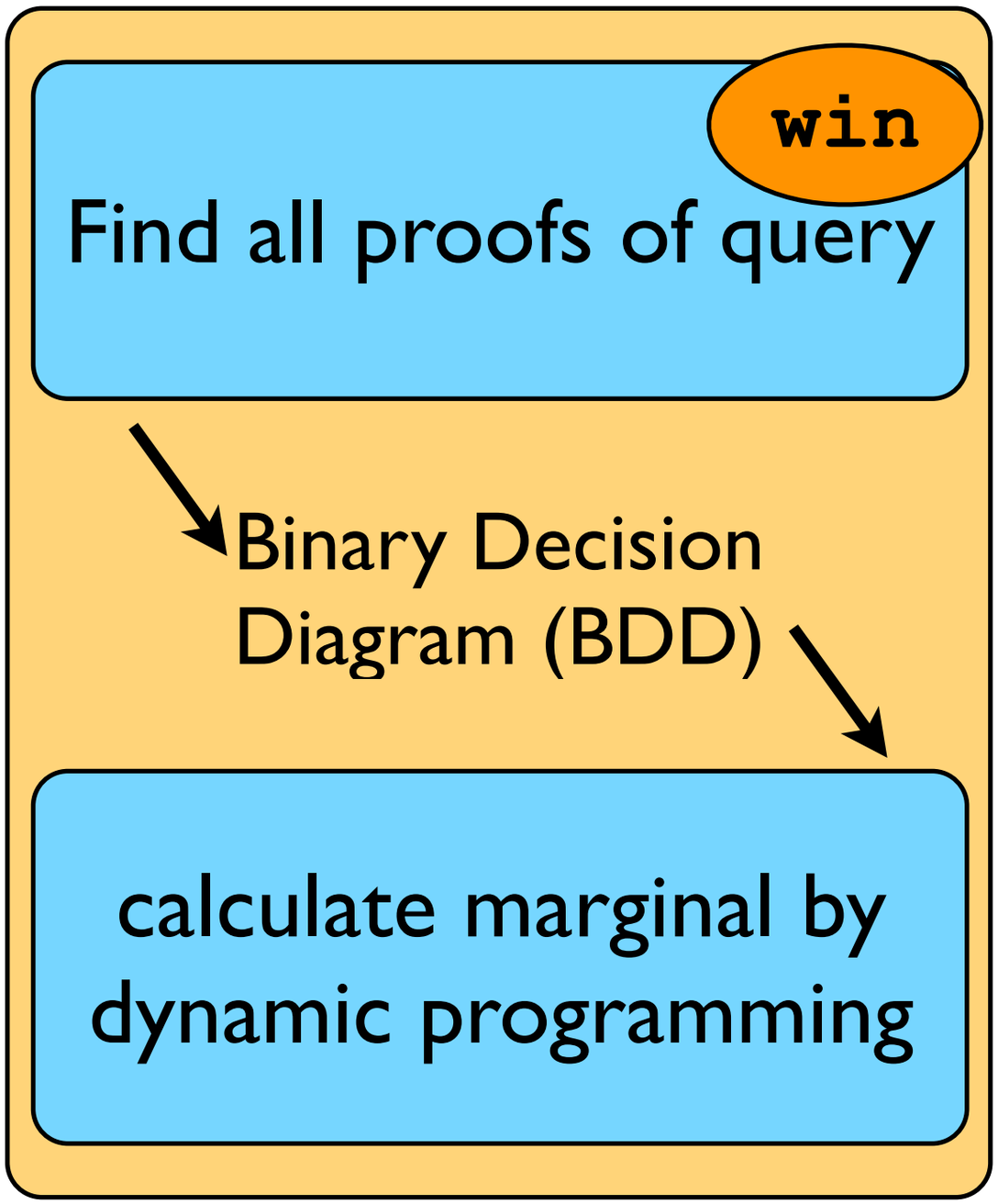


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```



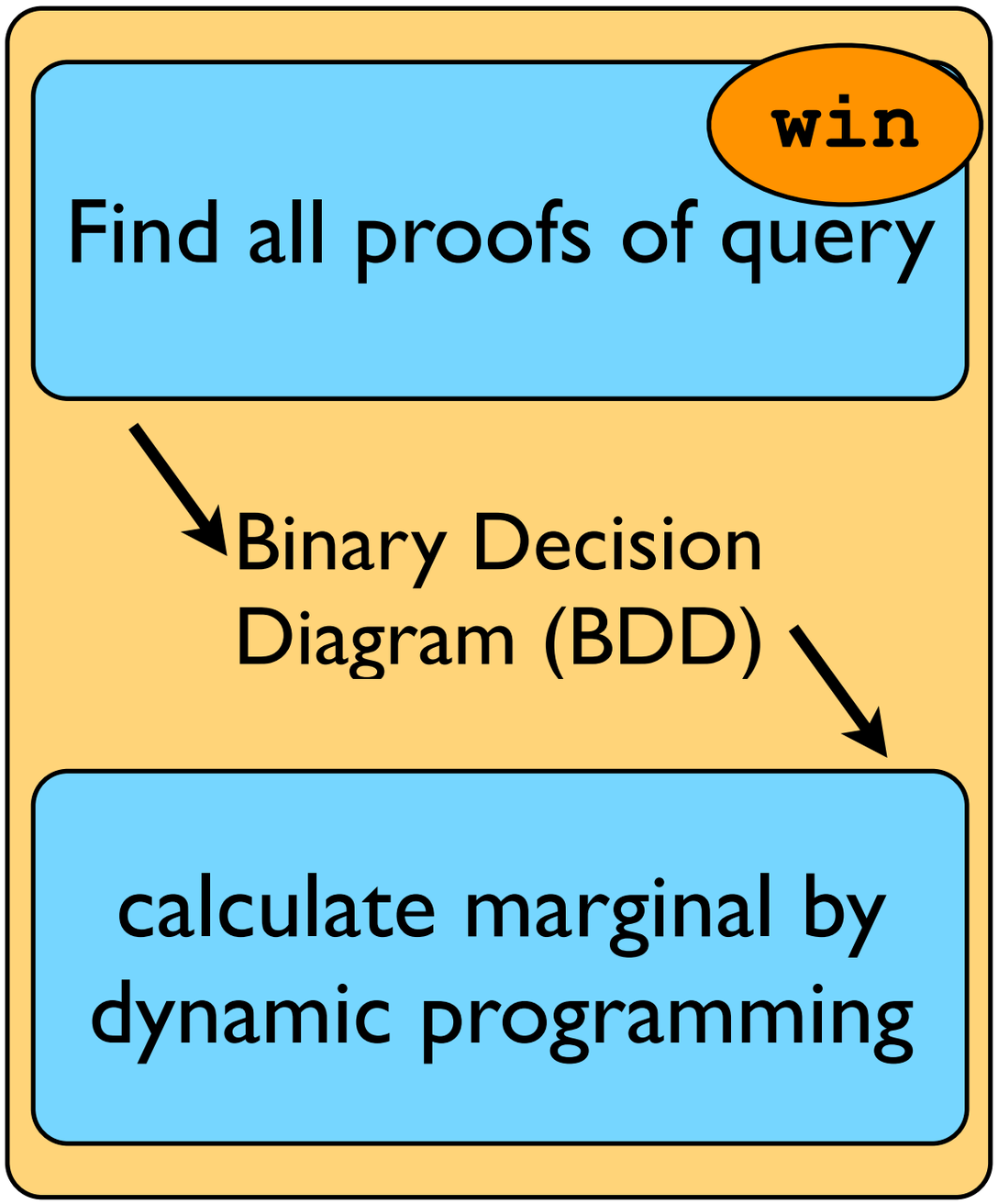

```
heads(1)
heads(2)  & heads(3)
```

[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

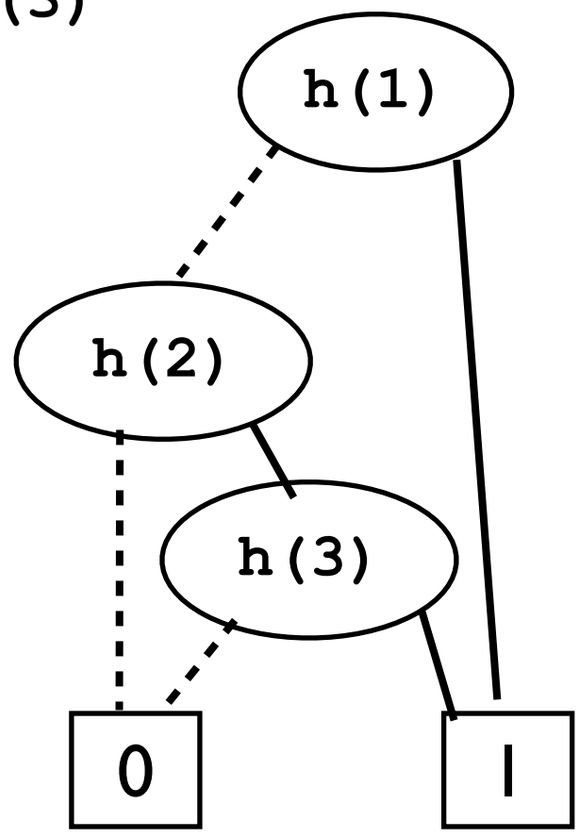
Initial Approach

(ProbLog1)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  &  heads(3)
```

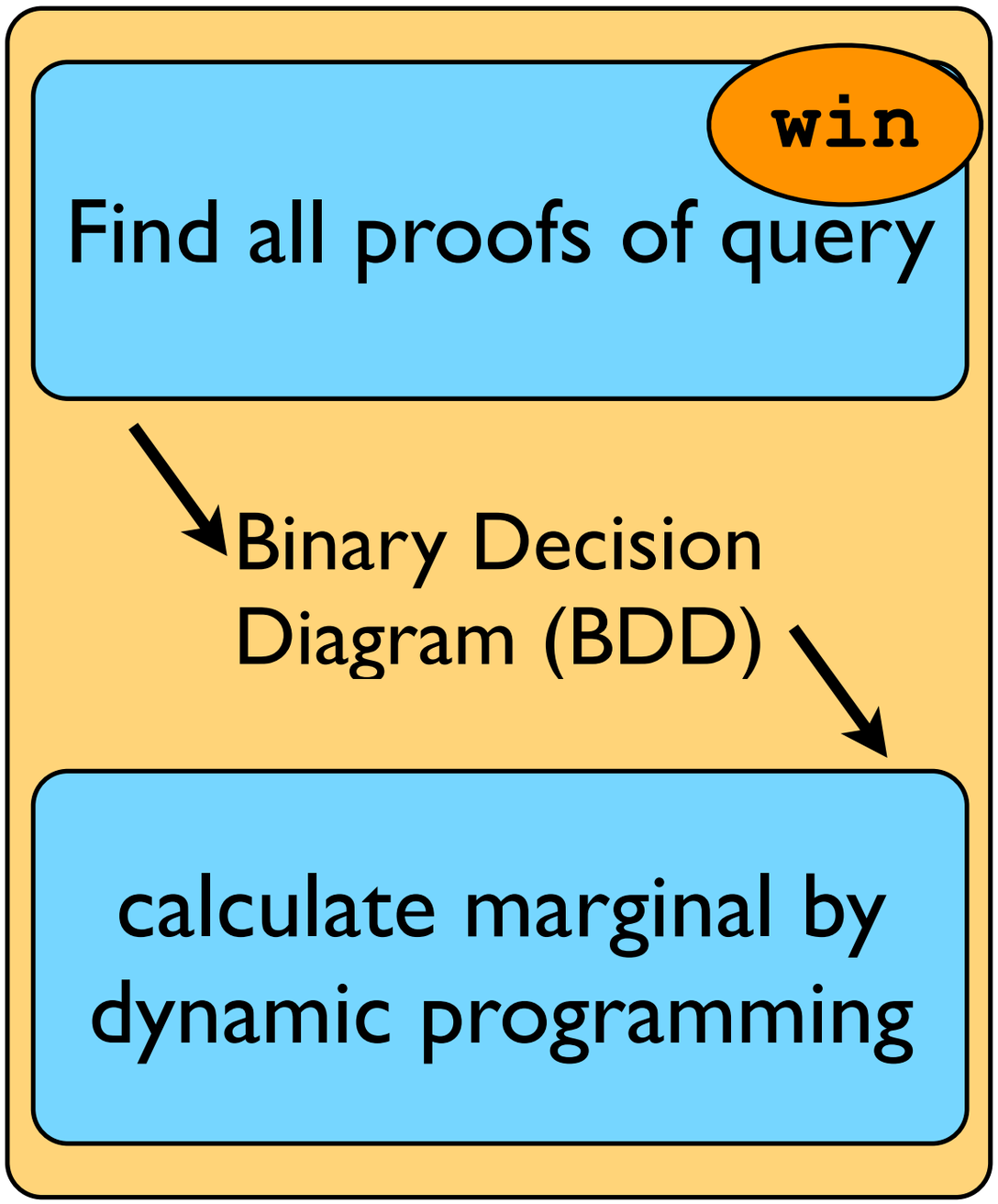


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

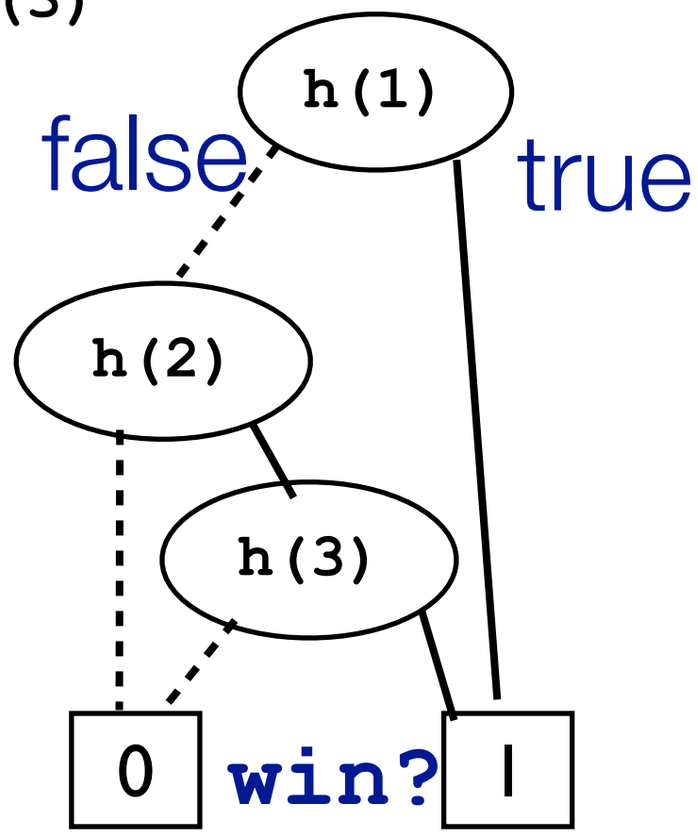
Initial Approach

(ProbLog1)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```

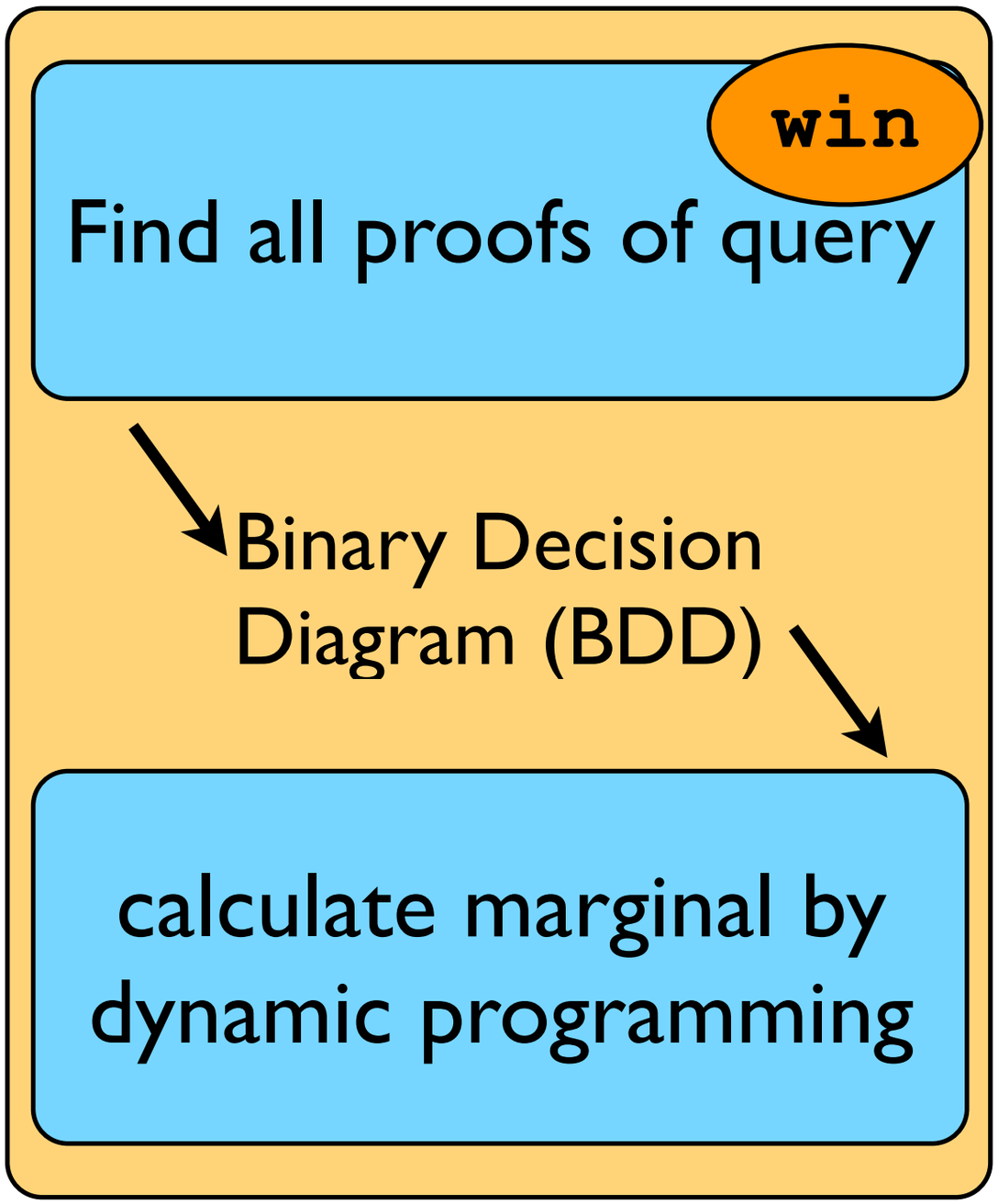


[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

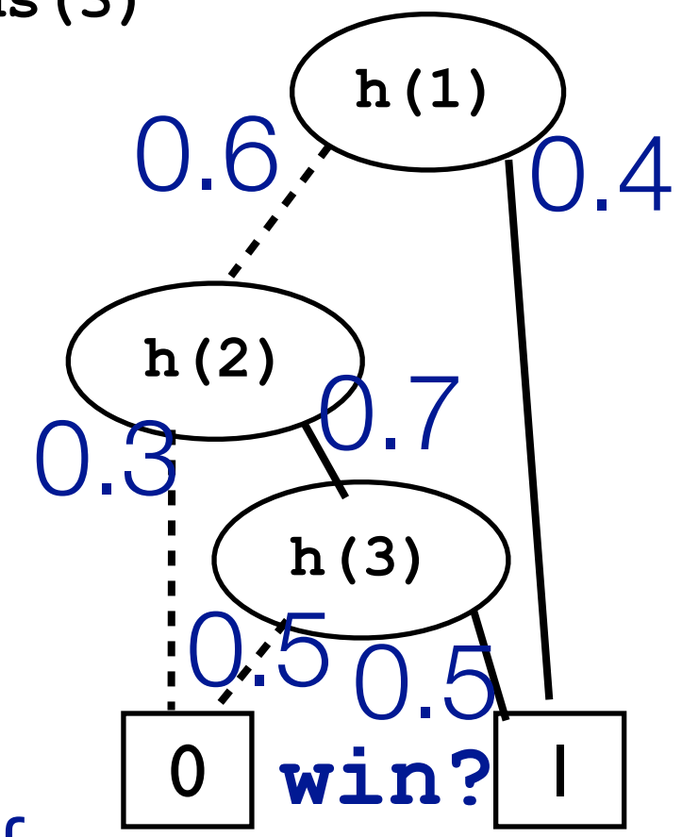
Initial Approach

(ProbLog1)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),heads(3).
```




```
heads(1)
heads(2)  & heads(3)
```



P(**win**) = probability of reaching 1-leaf

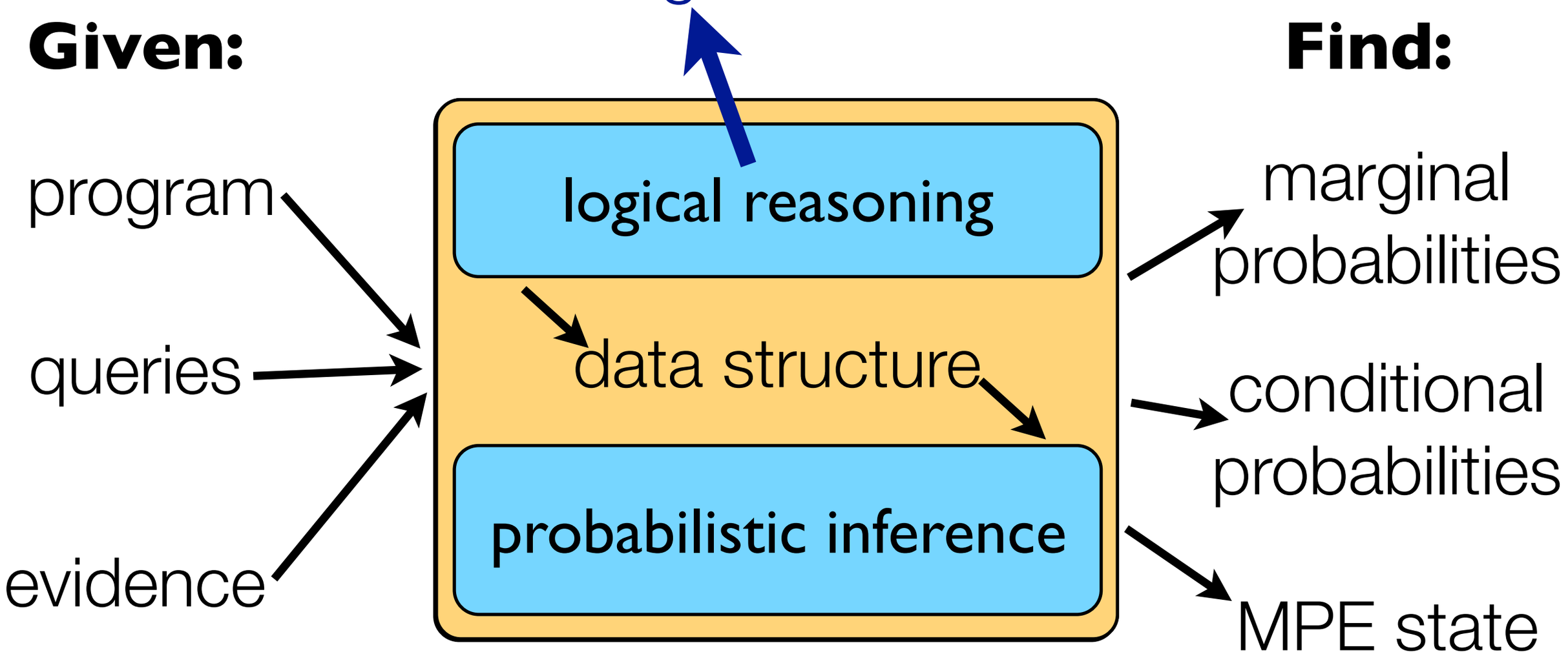
[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Answering Questions

1. using proofs
2. using models

Given:

Find:



Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

```
smokes(ann).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

```
smokes(ann).
smokes(bob).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

Logical Reasoning: Models in Prolog

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

```
?- smokes(carl).
```

- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
stress(ann).
influences(ann,bob).
influences(bob,carl).

smokes(ann).
smokes(bob).
smokes(carl).
```

- **ProbLog**: each possible world is a model, probability of query is sum over models where query is true

→ weighted model counting

Weighted Model Counting

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,

w(f) = p

w(not f) = 1–p

Weight

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)$$

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

$$WMC(\phi) = \sum_{I_V \models \phi} \prod_{l \in I} w(l)$$

interpretations (truth value assignments) of propositional variables

possible worlds

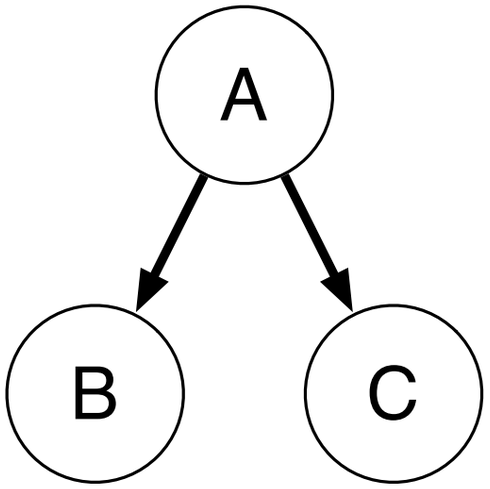
weight of literal

for p::f,

w(f) = p

w(not f) = 1–p

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

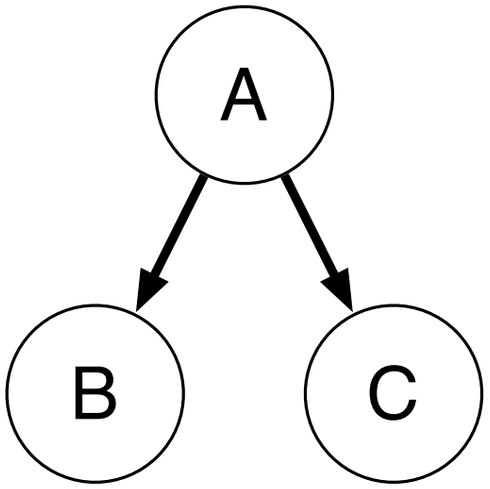
Variable A:	$\lambda_{a_1} \vee \lambda_{a_2}$	$\neg\lambda_{a_1} \vee \neg\lambda_{a_2}$
Variable B:	$\lambda_{b_1} \vee \lambda_{b_2}$	$\neg\lambda_{b_1} \vee \neg\lambda_{b_2}$
Variable C:	$\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}$	$\neg\lambda_{c_1} \vee \neg\lambda_{c_2}$
		$\neg\lambda_{c_1} \vee \neg\lambda_{c_3}$
		$\neg\lambda_{c_2} \vee \neg\lambda_{c_3}$
CPT 1:	$\lambda_{a_1} \Leftrightarrow \theta_{a_1}$	$\lambda_{a_2} \Leftrightarrow \theta_{a_2}$
CPT 2:	$\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_1}$	$\lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_1}$	$\lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_2}$
CPT 3:	$\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3\|a_2}$

$W(\theta_{a_1}) = 0.1$	$W(\theta_{a_2}) = 0.9$	
$W(\theta_{b_1\|a_1}) = 0.1$	$W(\theta_{b_2\|a_1}) = 0.9$	
$W(\theta_{b_1\|a_2}) = 0.2$	$W(\theta_{b_2\|a_2}) = 0.8$	
$W(\theta_{c_1\|a_1}) = 0.1$	$W(\theta_{c_2\|a_1}) = 0.2$	$W(\theta_{c_3\|a_1}) = 0.7$
$W(\theta_{c_1\|a_2}) = 0.01$	$W(\theta_{c_2\|a_2}) = 0.09$	$W(\theta_{c_3\|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

Pr(A|B=1,C=1) ~ Pr(A=1,B=1,C=1) + Pr(A=2,B=1,C=1)
= 0.1·0.1·0.1 + 0.9·0.2·0.01
2 models

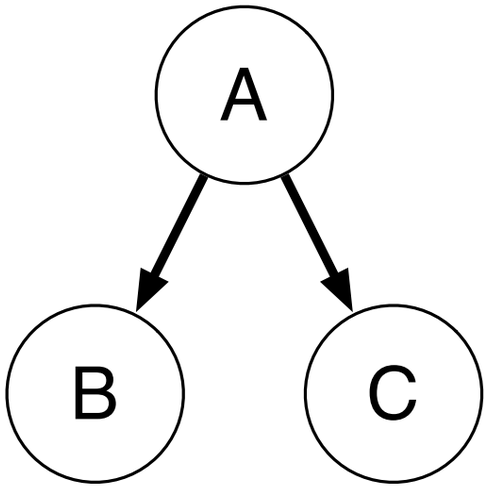
Variable A:	$\lambda_{a_1} \vee \lambda_{a_2}$	$\neg\lambda_{a_1} \vee \neg\lambda_{a_2}$
Variable B:	$\lambda_{b_1} \vee \lambda_{b_2}$	$\neg\lambda_{b_1} \vee \neg\lambda_{b_2}$
Variable C:	$\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}$	$\neg\lambda_{c_1} \vee \neg\lambda_{c_2}$
		$\neg\lambda_{c_1} \vee \neg\lambda_{c_3}$
		$\neg\lambda_{c_2} \vee \neg\lambda_{c_3}$
CPT 1:	$\lambda_{a_1} \Leftrightarrow \theta_{a_1}$	$\lambda_{a_2} \Leftrightarrow \theta_{a_2}$
CPT 2:	$\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_1}$	$\lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_1}$	$\lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2\|a_2}$
CPT 3:	$\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2\|a_2}$
	$\lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3\|a_1}$	$\lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3\|a_2}$

$W(\theta_{a_1}) = 0.1$	$W(\theta_{a_2}) = 0.9$	
$W(\theta_{b_1\|a_1}) = 0.1$	$W(\theta_{b_2\|a_1}) = 0.9$	
$W(\theta_{b_1\|a_2}) = 0.2$	$W(\theta_{b_2\|a_2}) = 0.8$	
$W(\theta_{c_1\|a_1}) = 0.1$	$W(\theta_{c_2\|a_1}) = 0.2$	$W(\theta_{c_3\|a_1}) = 0.7$
$W(\theta_{c_1\|a_2}) = 0.01$	$W(\theta_{c_2\|a_2}) = 0.09$	$W(\theta_{c_3\|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

Pr(A|B=1,C=1) ~ Pr(A=1,B=1,C=1) + Pr(A=2,B=1,C=1)
= 0.1·0.1·0.1 + 0.9·0.2·0.01
2 models

Variable A: $\lambda_{a_1} \vee \lambda_{a_2}$ $\quad \neg\lambda_{a_1} \vee \neg\lambda_{a_2}$

Variable B: $\lambda_{b_1} \vee \lambda_{b_2}$ $\quad \neg\lambda_{b_1} \vee \neg\lambda_{b_2}$

Variable C: $\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}$ $\quad \neg\lambda_{c_1} \vee \neg\lambda_{c_2}$ $\quad \neg\lambda_{c_1} \vee \neg\lambda_{c_3}$ $\quad \neg\lambda_{c_2} \vee \neg\lambda_{c_3}$

CPT 1: $\lambda_{a_1} \Leftrightarrow \theta_{a_1}$ $\quad \lambda_{a_2} \Leftrightarrow \theta_{a_2}$

CPT 2: $\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_2}$

CPT 3: $\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_2}$

$W(\theta_{a_1}) = 0.1$ $\quad W(\theta_{a_2}) = 0.9$

$W(\theta_{b_1|a_1}) = 0.1$ $\quad W(\theta_{b_2|a_1}) = 0.9$

$W(\theta_{b_1|a_2}) = 0.2$ $\quad W(\theta_{b_2|a_2}) = 0.8$

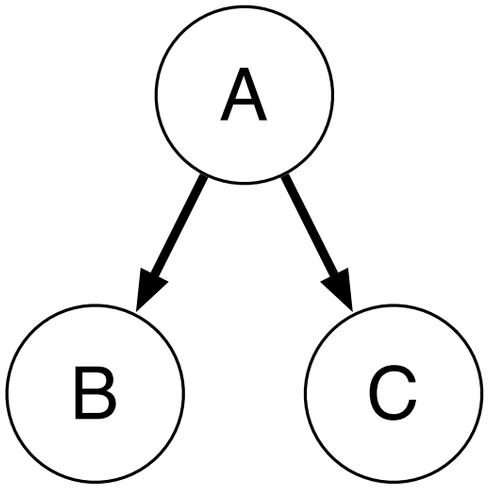
$W(\theta_{c_1|a_1}) = 0.1$ $\quad W(\theta_{c_2|a_1}) = 0.2$ $\quad W(\theta_{c_3|a_1}) = 0.7$

$W(\theta_{c_1|a_2}) = 0.01$ $\quad W(\theta_{c_2|a_2}) = 0.09$ $\quad W(\theta_{c_3|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

Pr(A|B=1,C=1) ~ Pr(A=1,B=1,C=1) + Pr(A=2,B=1,C=1)
= 0.1·0.1·0.1 + 0.9·0.2·0.01
2 models

Variable A: $\lambda_{a_1} \vee \lambda_{a_2}$ $\quad \neg\lambda_{a_1} \vee \neg\lambda_{a_2}$

Variable B: $\lambda_{b_1} \vee \cancel{\lambda_{b_2}}$ $\quad \cancel{\neg\lambda_{b_1}} \vee \neg\lambda_{b_2}$

Variable C: $\lambda_{c_1} \vee \cancel{\lambda_{c_2}} \vee \cancel{\lambda_{c_3}}$ $\quad \cancel{\neg\lambda_{c_1}} \vee \neg\lambda_{c_2}$

$\cancel{\neg\lambda_{c_1}} \vee \neg\lambda_{c_3}$

$\neg\lambda_{c_2} \vee \neg\lambda_{c_3}$

CPT 1: $\lambda_{a_1} \Leftrightarrow \theta_{a_1}$ $\quad \lambda_{a_2} \Leftrightarrow \theta_{a_2}$

CPT 2: $\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_2}$

CPT 3: $\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_2}$

$W(\theta_{a_1}) = 0.1$ $\quad W(\theta_{a_2}) = 0.9$

$W(\theta_{b_1|a_1}) = 0.1$ $\quad W(\theta_{b_2|a_1}) = 0.9$

$W(\theta_{b_1|a_2}) = 0.2$ $\quad W(\theta_{b_2|a_2}) = 0.8$

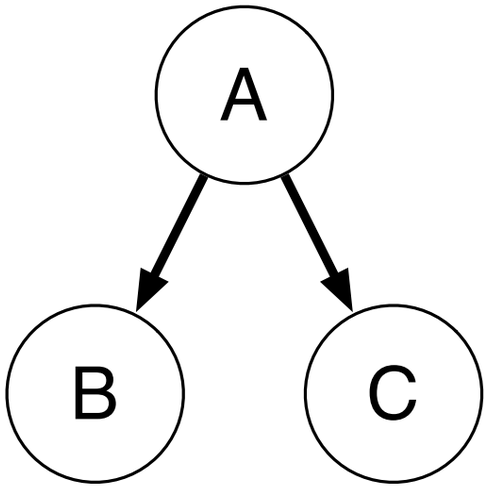
$W(\theta_{c_1|a_1}) = 0.1$ $\quad W(\theta_{c_2|a_1}) = 0.2$ $\quad W(\theta_{c_3|a_1}) = 0.7$

$W(\theta_{c_1|a_2}) = 0.01$ $\quad W(\theta_{c_2|a_2}) = 0.09$ $\quad W(\theta_{c_3|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

Pr(A|B=1,C=1) ~ Pr(A=1,B=1,C=1) + Pr(A=2,B=1,C=1)
= 0.1·0.1·0.1 + 0.9·0.2·0.01
2 models

Variable A: $\lambda_{a_1} \vee \lambda_{a_2}$ $\quad \neg\lambda_{a_1} \vee \neg\lambda_{a_2}$

Variable B: $\lambda_{b_1} \vee \cancel{\lambda_{b_2}}$ $\quad \cancel{\neg\lambda_{b_1}} \vee \neg\lambda_{b_2}$

Variable C: $\lambda_{c_1} \vee \cancel{\lambda_{c_2}} \vee \cancel{\lambda_{c_3}}$ $\quad \cancel{\neg\lambda_{c_1}} \vee \neg\lambda_{c_2}$

$\cancel{\neg\lambda_{c_1}} \vee \neg\lambda_{c_3}$

$\neg\lambda_{c_2} \vee \neg\lambda_{c_3}$

CPT 1: $\lambda_{a_1} \Leftrightarrow \theta_{a_1}$ $\quad \lambda_{a_2} \Leftrightarrow \theta_{a_2}$

CPT 2: $\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_2}$

$\lambda_{a_1} \wedge \cancel{\lambda_{b_2}} \Leftrightarrow \theta_{b_2|a_1}$ $\quad \cancel{\lambda_{a_2}} \wedge \cancel{\lambda_{b_2}} \Leftrightarrow \theta_{b_2|a_2}$

CPT 3: $\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_2}$

$\lambda_{a_1} \wedge \cancel{\lambda_{c_2}} \Leftrightarrow \theta_{c_2|a_1}$ $\quad \lambda_{a_2} \wedge \cancel{\lambda_{c_2}} \Leftrightarrow \theta_{c_2|a_2}$

$\lambda_{a_1} \wedge \cancel{\lambda_{c_3}} \Leftrightarrow \theta_{c_3|a_1}$ $\quad \lambda_{a_2} \wedge \cancel{\lambda_{c_3}} \Leftrightarrow \theta_{c_3|a_2}$

$W(\theta_{a_1}) = 0.1$ $\quad W(\theta_{a_2}) = 0.9$

$W(\theta_{b_1|a_1}) = 0.1$ $\quad W(\theta_{b_2|a_1}) = 0.9$

$W(\theta_{b_1|a_2}) = 0.2$ $\quad W(\theta_{b_2|a_2}) = 0.8$

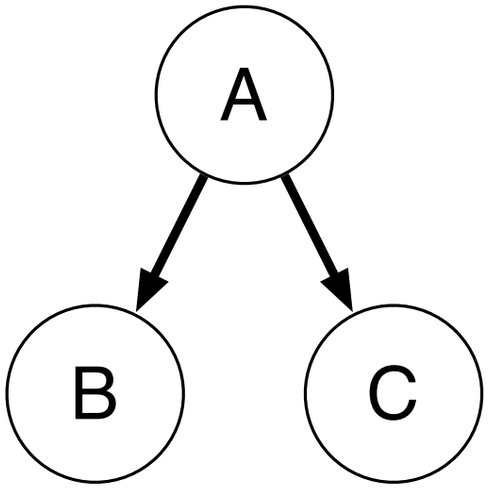
$W(\theta_{c_1|a_1}) = 0.1$ $\quad W(\theta_{c_2|a_1}) = 0.2$ $\quad W(\theta_{c_3|a_1}) = 0.7$

$W(\theta_{c_1|a_2}) = 0.01$ $\quad W(\theta_{c_2|a_2}) = 0.09$ $\quad W(\theta_{c_3|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

Bayesian net to WMC



A	Pr
a_1	0.1
a_2	0.9

A	B	Pr
a_1	b_1	0.1
a_1	b_2	0.9
a_2	b_1	0.2
a_2	b_2	0.8

A	C	Pr
a_1	c_1	0.1
a_1	c_2	0.2
a_1	c_3	0.7
a_2	c_1	0.01
a_2	c_2	0.09
a_2	c_3	0.9

Pr(A|B=1,C=1) ~ Pr(A=1,B=1,C=1) + Pr(A=2,B=1,C=1)
= 0.1·0.1·0.1 + 0.9·0.2·0.01
2 models

Variable A: $\lambda_{a_1} \vee \lambda_{a_2}$ $\quad \neg\lambda_{a_1} \vee \neg\lambda_{a_2}$

Variable B: $\lambda_{b_1} \vee \lambda_{b_2}$ $\quad \neg\lambda_{b_1} \vee \neg\lambda_{b_2}$

Variable C: $\lambda_{c_1} \vee \lambda_{c_2} \vee \lambda_{c_3}$ $\quad \neg\lambda_{c_1} \vee \neg\lambda_{c_2}$ $\quad \neg\lambda_{c_1} \vee \neg\lambda_{c_3}$ $\quad \neg\lambda_{c_2} \vee \neg\lambda_{c_3}$

CPT 1: $\lambda_{a_1} \Leftrightarrow \theta_{a_1}$ $\quad \lambda_{a_2} \Leftrightarrow \theta_{a_2}$

CPT 2: $\lambda_{a_1} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_1} \Leftrightarrow \theta_{b_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{b_2} \Leftrightarrow \theta_{b_2|a_2}$

CPT 3: $\lambda_{a_1} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_1} \Leftrightarrow \theta_{c_1|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_2} \Leftrightarrow \theta_{c_2|a_2}$

$\lambda_{a_1} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_1}$ $\quad \lambda_{a_2} \wedge \lambda_{c_3} \Leftrightarrow \theta_{c_3|a_2}$

$W(\theta_{a_1}) = 0.1$ $\quad W(\theta_{a_2}) = 0.9$

$W(\theta_{b_1|a_1}) = 0.1$ $\quad W(\theta_{b_2|a_1}) = 0.9$

$W(\theta_{b_1|a_2}) = 0.2$ $\quad W(\theta_{b_2|a_2}) = 0.8$

$W(\theta_{c_1|a_1}) = 0.1$ $\quad W(\theta_{c_2|a_1}) = 0.2$ $\quad W(\theta_{c_3|a_1}) = 0.7$

$W(\theta_{c_1|a_2}) = 0.01$ $\quad W(\theta_{c_2|a_2}) = 0.09$ $\quad W(\theta_{c_3|a_2}) = 0.9$

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. Artificial Intelligence, 172(6):772–799, 2008.

KU LEUVEN

ProbLog →CNF

```
?- smokes(carl).
```

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
      influences(Y,X),
      smokes(Y).
```

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

ProbLog →CNF

```
?- smokes(carl).
```

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

- Convert to propositional logic formula

ProbLog →CNF

```
?- smokes(carl).
```

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob)  :- stress(bob).
smokes(bob)  :- influences(ann,bob),smokes(ann).
smokes(ann)  :- stress(ann).
```

- Convert to propositional logic formula

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

ProbLog →CNF

```
0.8::stress(ann).
0.4::stress(bob).
0.6::influences(ann,bob).
0.2::influences(bob,carl).

smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X),
     smokes(Y).
```

```
?- smokes(carl).
```

- Find relevant ground rules by backward reasoning

```
smokes(carl) :- influences(bob,carl),smokes(bob).
smokes(bob) :- stress(bob).
smokes(bob) :- influences(ann,bob),smokes(ann).
smokes(ann) :- stress(ann).
```

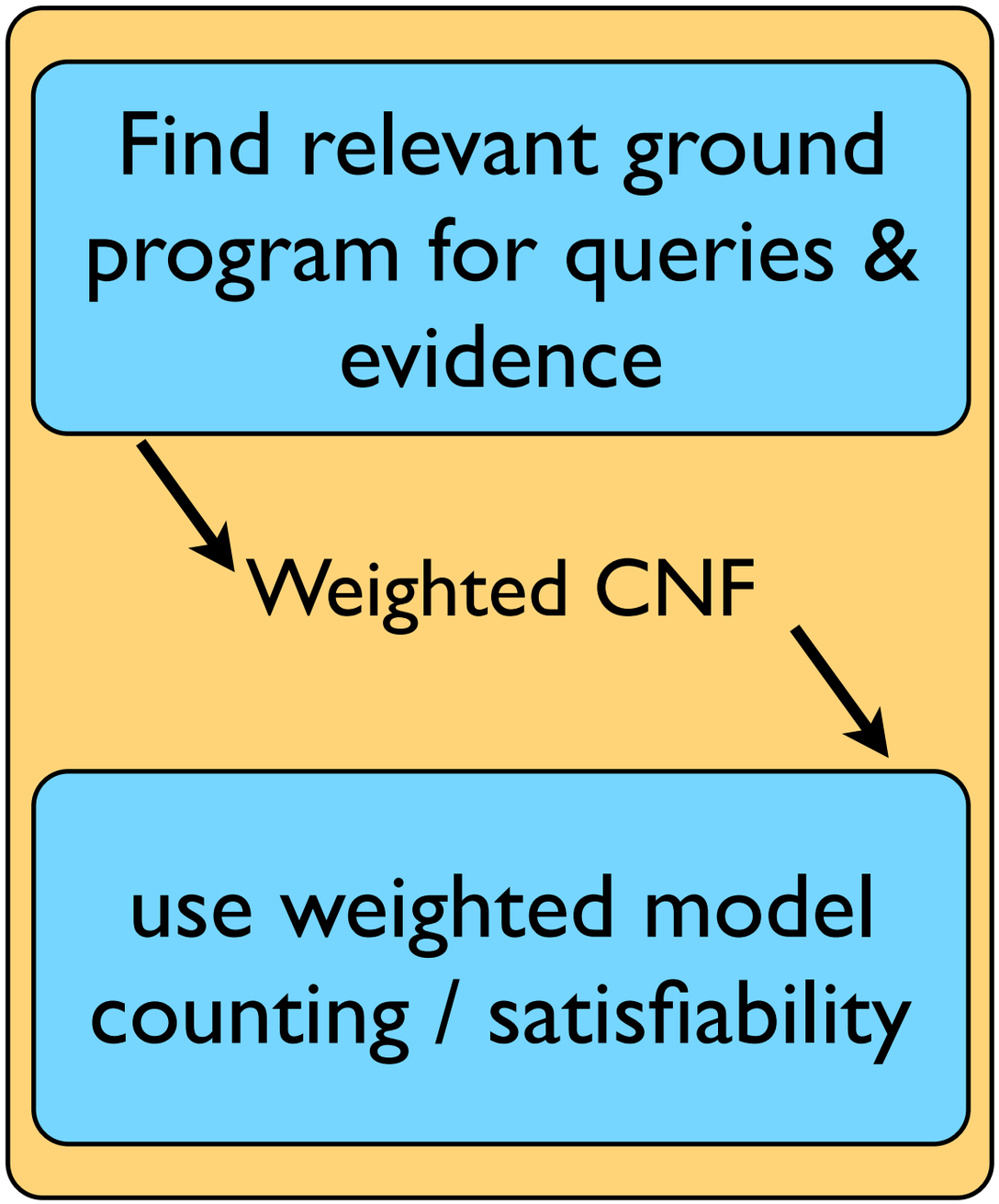
- Convert to propositional logic formula

$$sm(c) \leftrightarrow (i(b,c) \wedge sm(b))$$
$$\wedge\ sm(b) \leftrightarrow (st(b) \vee (i(a,b) \wedge sm(a)))$$
$$\wedge\ sm(a) \leftrightarrow st(a)$$

- Rewrite in CNF (as usual)

Current Approach

(ProbLog2)

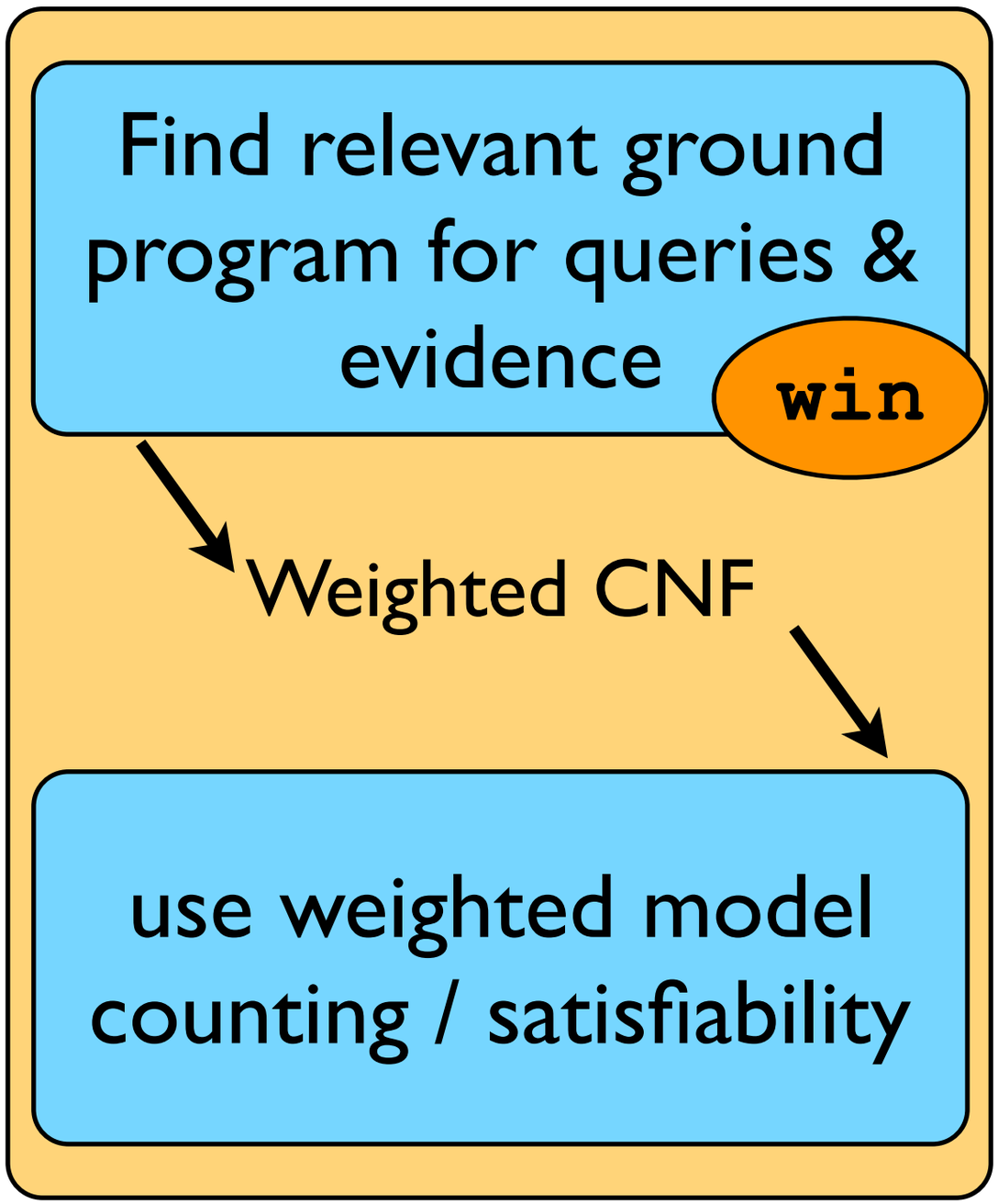


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```

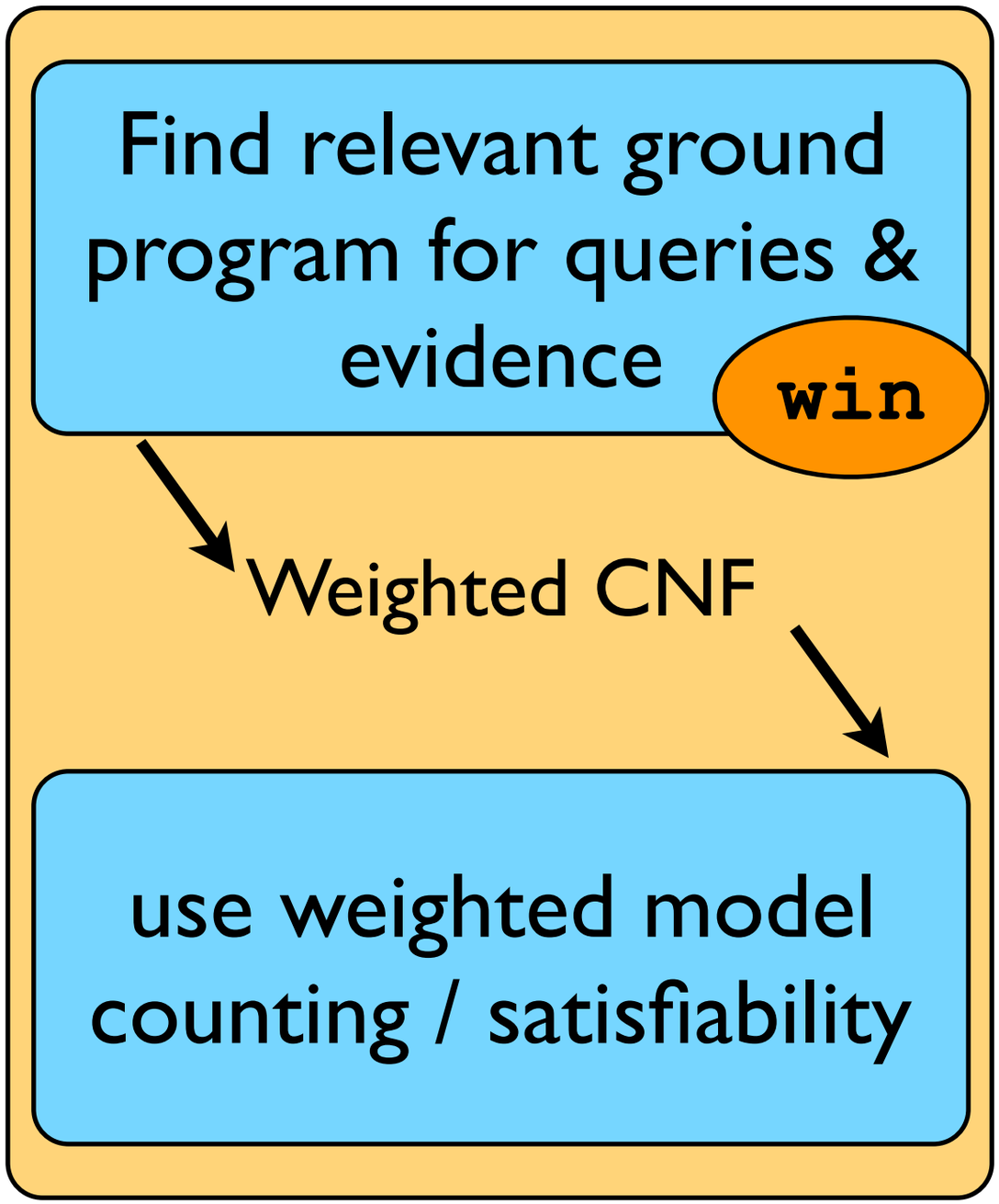


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```



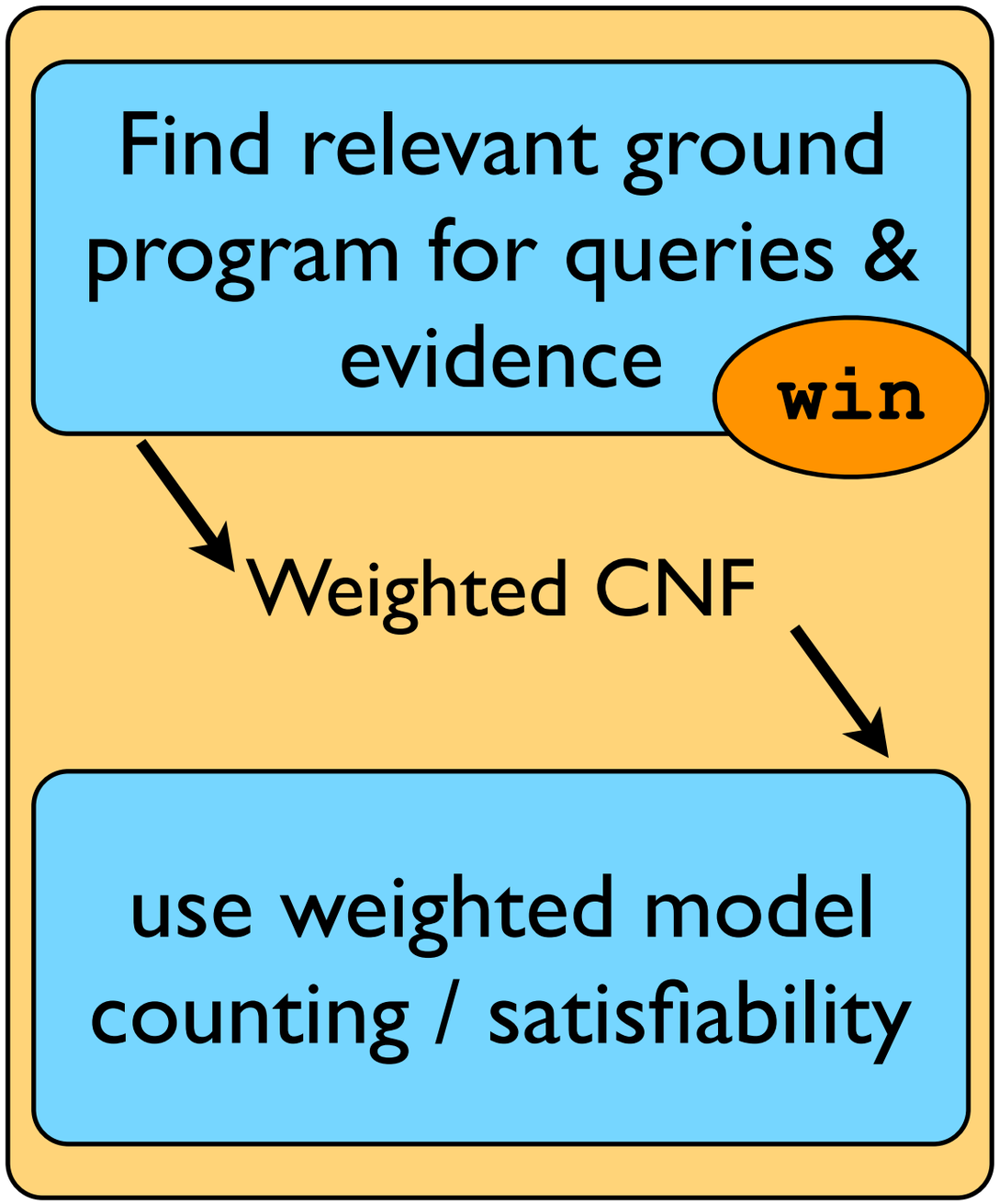

```
win :- heads(1).
win :- heads(2), heads(3).
```


[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

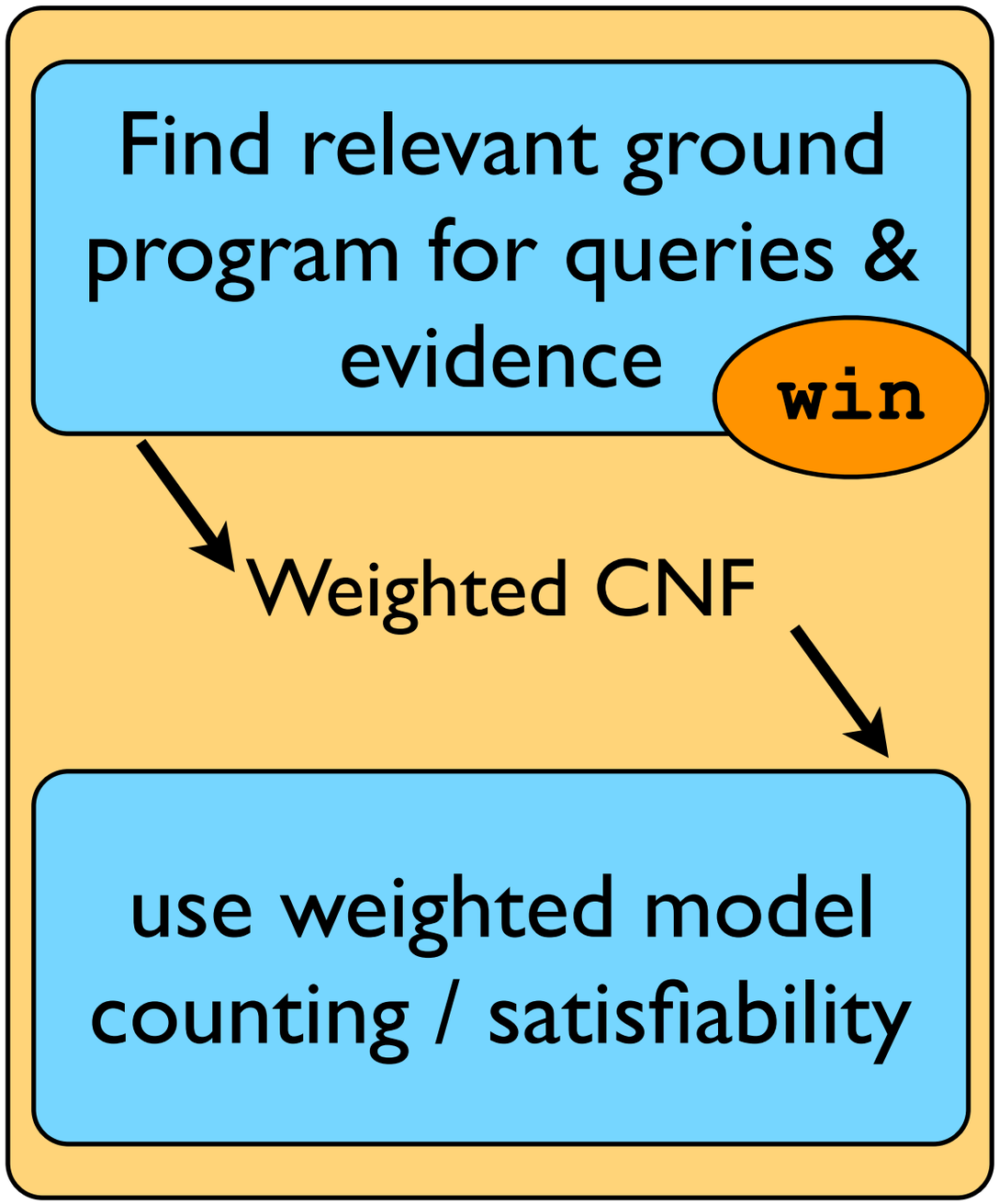
win ↔ h(1) ∨ (h(2) ∧ h(3))

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

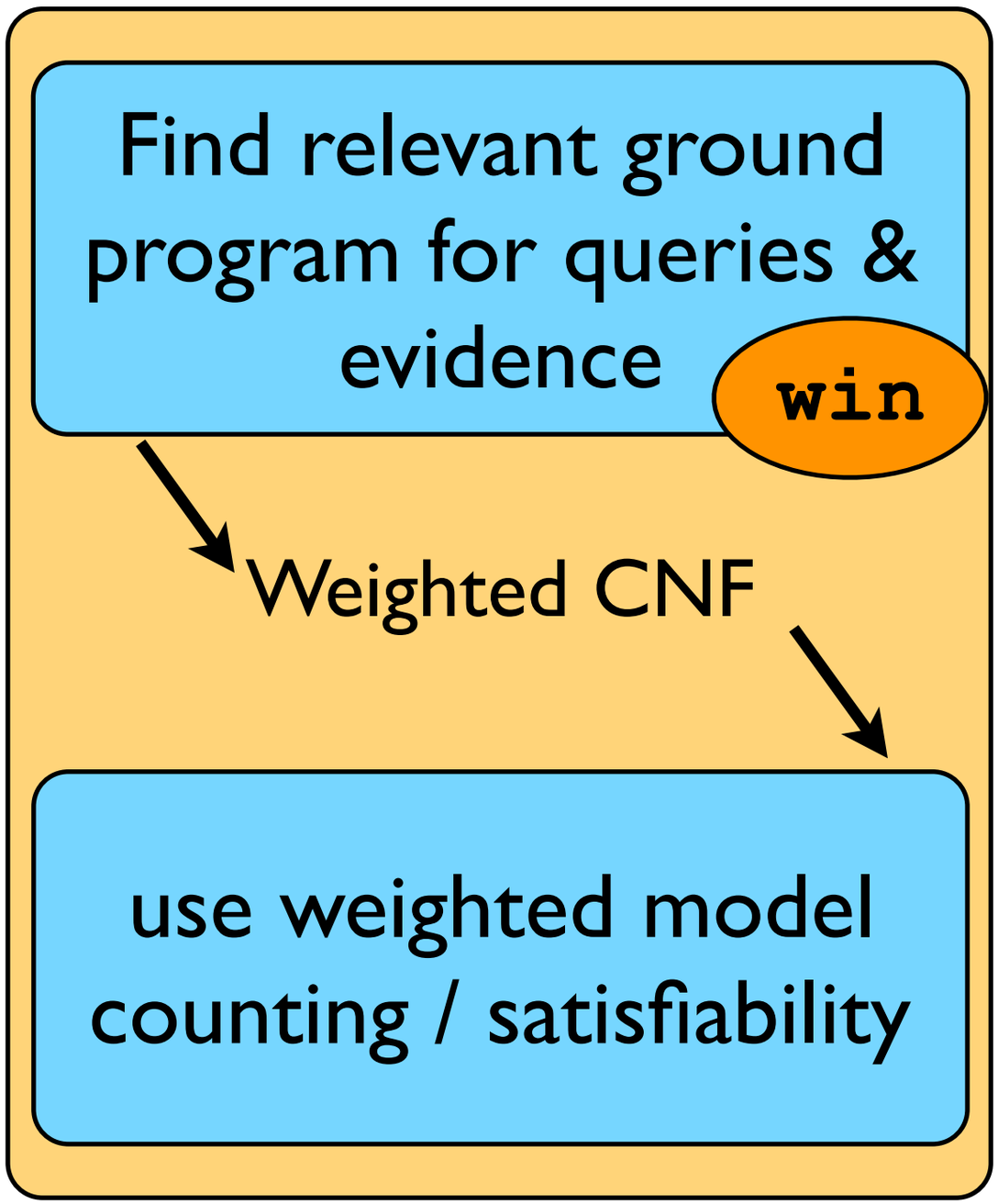
$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

$$\text{win} \leftrightarrow h(1) \vee (h(2) \wedge h(3))$$

$$(\neg\text{win} \vee h(1) \vee h(2)) \wedge (\neg\text{win} \vee h(1) \vee h(3)) \wedge (\text{win} \vee \neg h(1)) \wedge (\text{win} \vee \neg h(2) \vee \neg h(3))$$

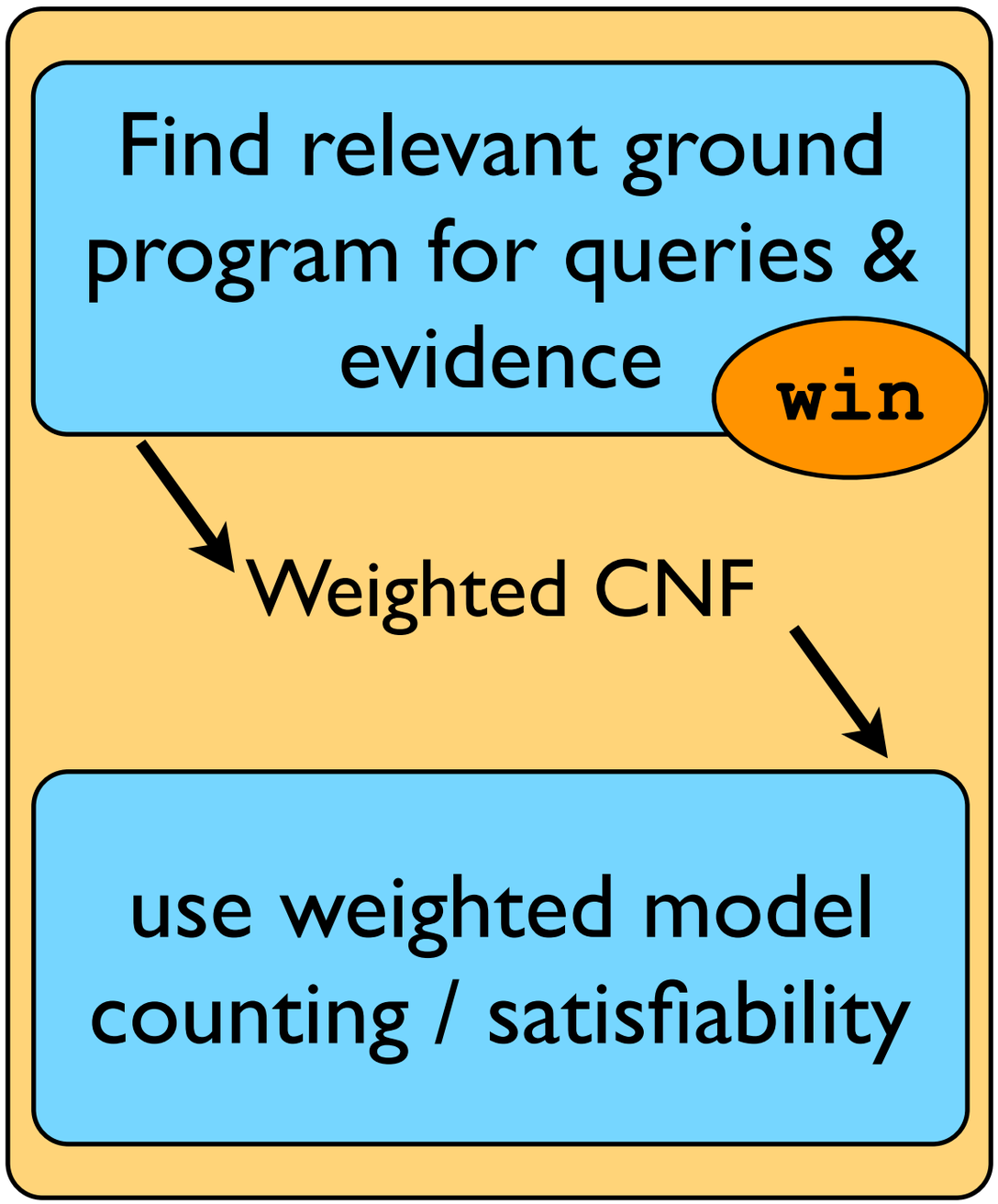
$h(1) \rightarrow 0.4$	$h(2) \rightarrow 0.7$	$h(3) \rightarrow 0.5$
$\neg h(1) \rightarrow 0.6$	$\neg h(2) \rightarrow 0.3$	$\neg h(3) \rightarrow 0.5$

[Fierens et al, TPLP 14]

Current Approach

(ProbLog2)

```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
       heads(3).
```




```
win :- heads(1).
win :- heads(2), heads(3).
```

win ↔ h(1) ∨ (h(2) ∧ h(3))

(¬win ∨ h(1) ∨ h(2))
∧ (¬win ∨ h(1) ∨ h(3))
∧ (win ∨ ¬h(1))
∧ (win ∨ ¬h(2) ∨ ¬h(3))

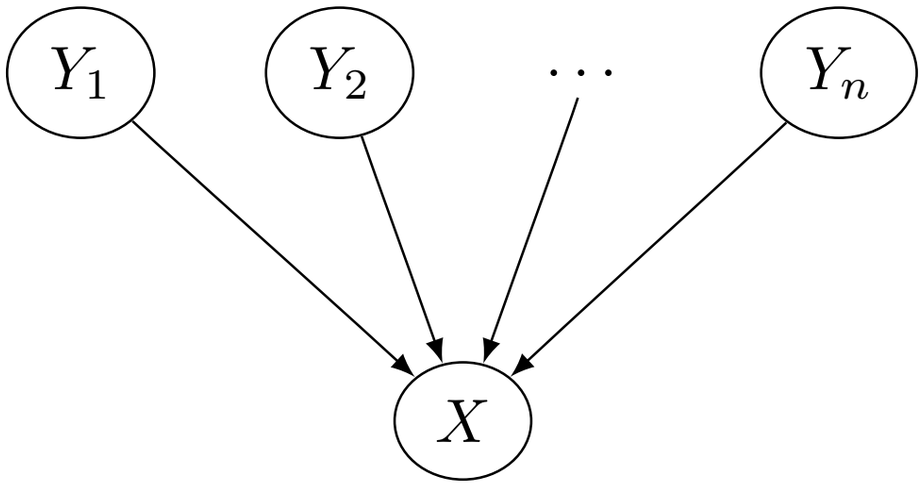
use standard tool

h(1) → 0.4	h(2) → 0.7	h(3) → 0.5
¬h(1) → 0.6	¬h(2) → 0.3	¬h(3) → 0.5

[Fierens et al, TPLP 14]

Example: Encoding

Bayesian Network



Probabilistic Logic

```
0.4::y(1).
0.3::y(2).
…
0.5::y(n).
0.8::x :- y(N).
```

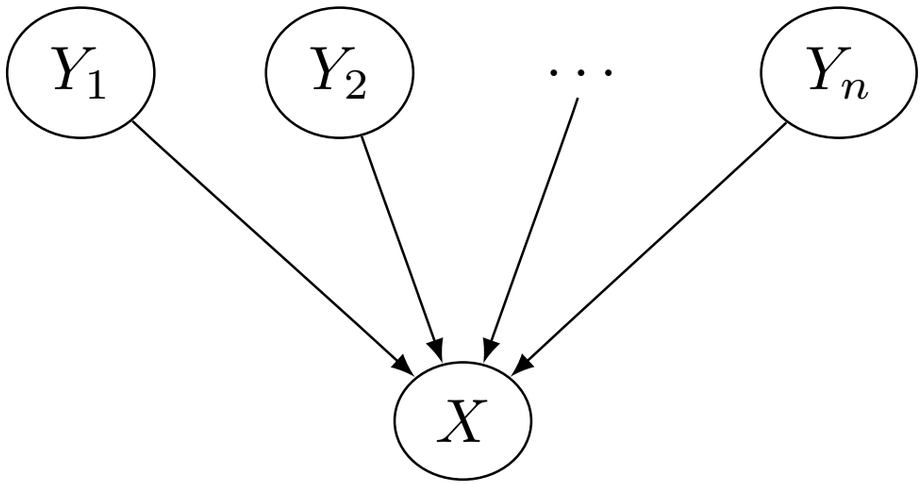
Propositional Logic

```
y(1) ⇔ p(1,1)
y(2) ⇔ p(1,2)
…
y(n) ⇔ p(1,n)
x ⇔ (y(1) ∧ p(2,1)) ∨ …
      ∨ (y(n) ∧ p(2,n))
```

```
w(p(1,1)) = 0.4
w(p(1,2)) = 0.3
…
w(p(1,n)) = 0.5
w(p(2,1)) = 0.8
…
w(p(2,n)) = 0.8
```

Example: Encoding

Bayesian Network



Probabilistic Logic

```
0.4::y(1).
0.3::y(2).
…
0.5::y(n).
0.8::x :- y(N).
```

Propositional Logic

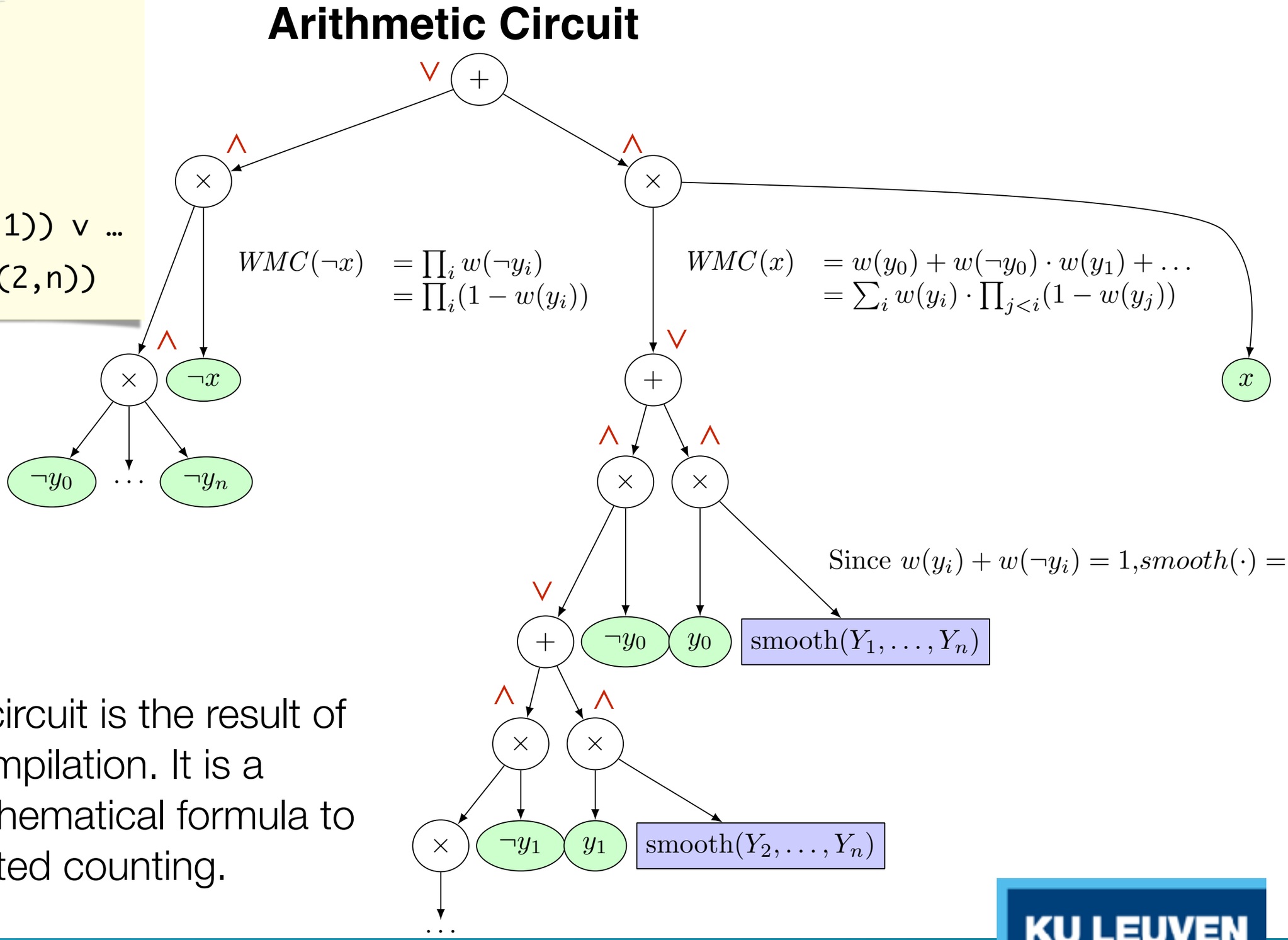
```
y(1) ⇔ p(1,1)
y(2) ⇔ p(1,2)
…
y(n) ⇔ p(1,n)
x ⇔ (y(1) ∧ p(2,1)) ∨ …
      ∨ (y(n) ∧ p(2,n))
```

```
w(p(1,1)) = 0.4
w(p(1,2)) = 0.3
…
w(p(1,n)) = 0.5
w(p(2,1)) = 0.8
…
w(p(2,n)) = 0.8
```

Compact representation of full CPT

Example: Weighted Model Counting

```
y(1) ⇔ p(1,1)
y(2) ⇔ p(1,2)

…
y(n) ⇔ p(1,n)
x ⇔ (y(1) ∧ p(2,1)) ∨ …
      ∨ (y(n) ∧ p(2,n))
```

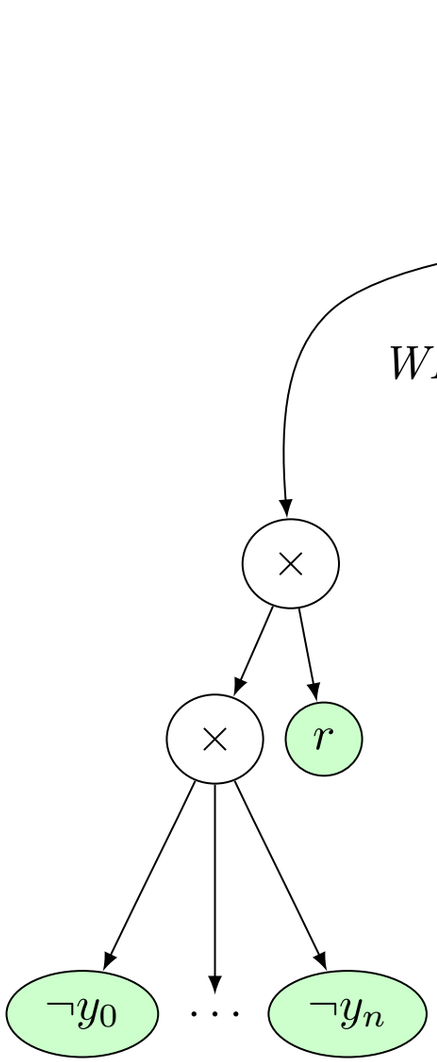


An arithmetic circuit is the result of knowledge compilation. It is a (compact) mathematical formula to perform weighted counting.

Example: Alternative Encoding

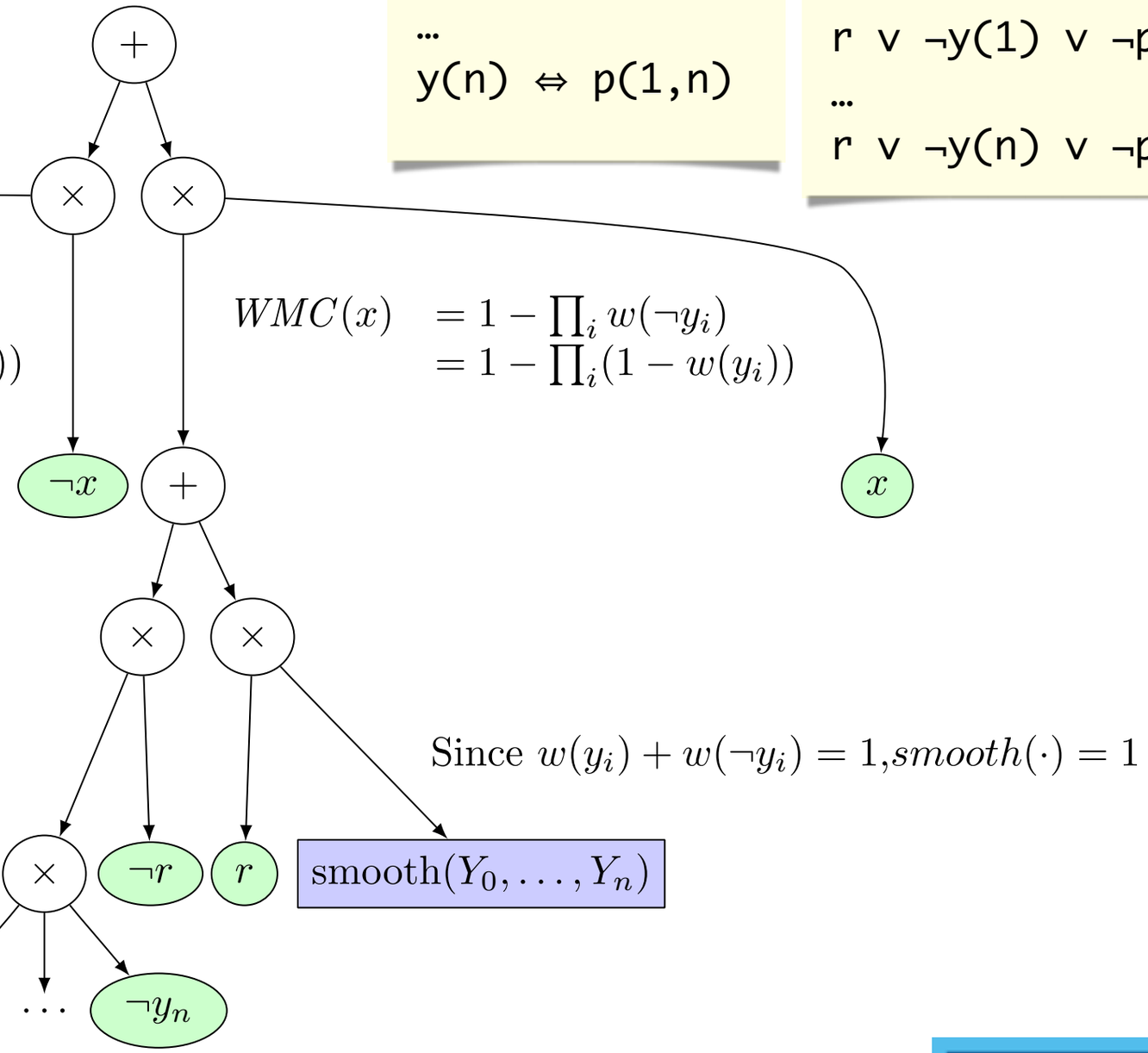
```
y(1) ⇔ p(1,1)
y(2) ⇔ p(1,2)
…
y(n) ⇔ p(1,n)
```

```
x ⇔ z
z v ¬y(1) v ¬p(2,1)
…
z v ¬y(n) v ¬p(2,n)
r v z
r v ¬y(1) v ¬p(2,1)
…
r v ¬y(n) v ¬p(2,n)
```



$$WMC(\neg x) = \prod_i w(\neg y_i) = \prod_i (1 - w(y_i))$$

$$WMC(x) = 1 - \prod_i w(\neg y_i) = 1 - \prod_i (1 - w(y_i))$$



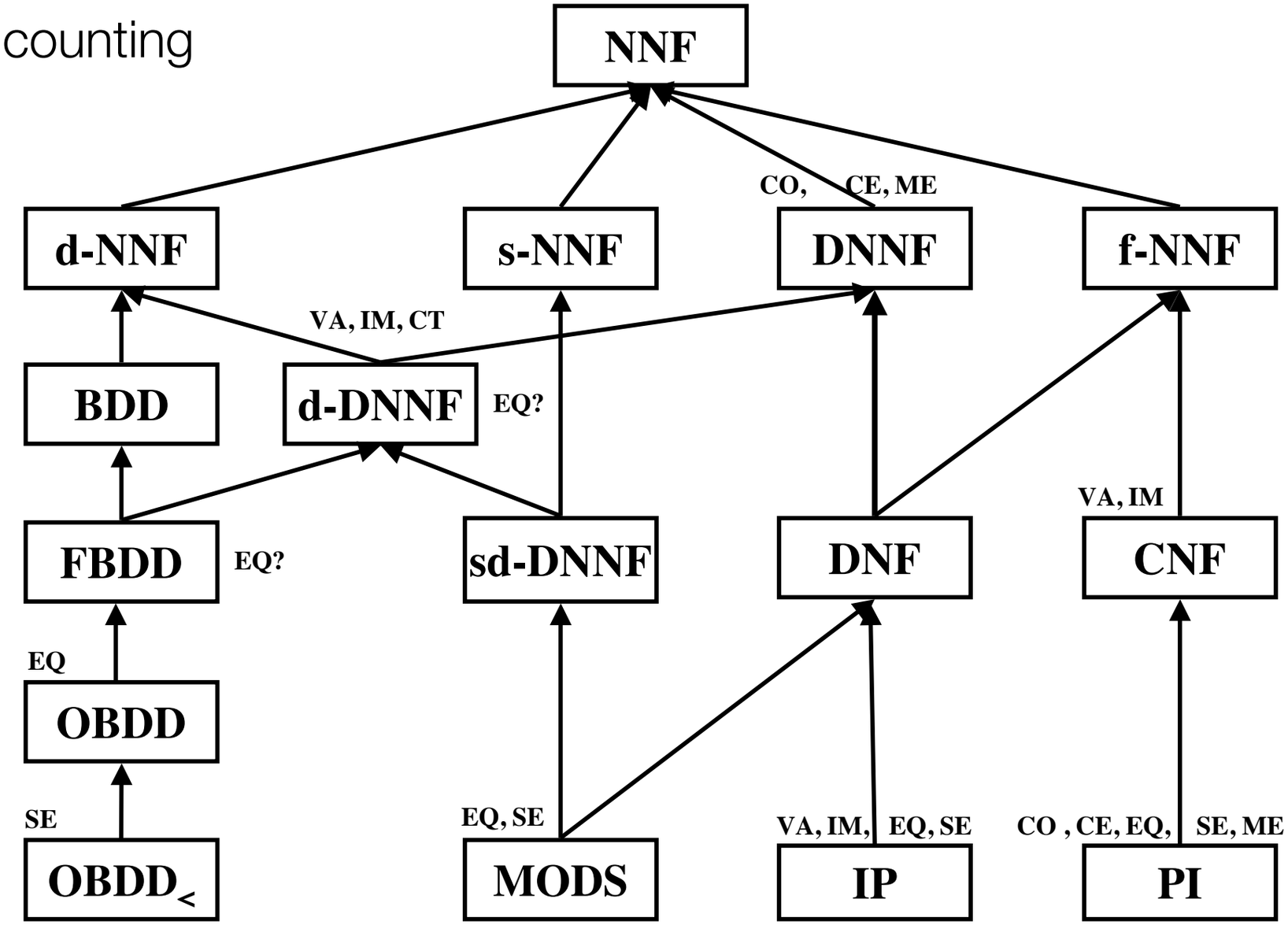
Since $w(y_i) + w(\neg y_i) = 1, smooth(\cdot) = 1$

G. Van den Broeck, W. Meert, and A. Darwiche. Skolemization for weighted first-order model counting.KR , 2014.
W. Meert, J. Vlasselaer, and G. Van den Broeck. A relaxed tseitin transformation for weighted model counting. StarAI, 2016.

KU LEUVEN

Knowledge Compilation

Translate a propositional formula into a compact representation that allows for tractable weighted model counting



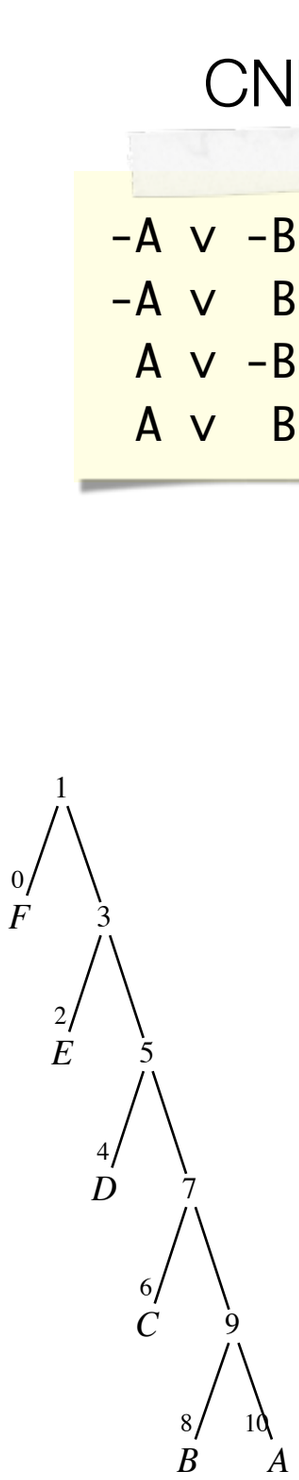
A. Darwiche and P. Marquis. A knowledge compilation map. Journal of Artificial Intelligence Research, 17:229–264, 2002.

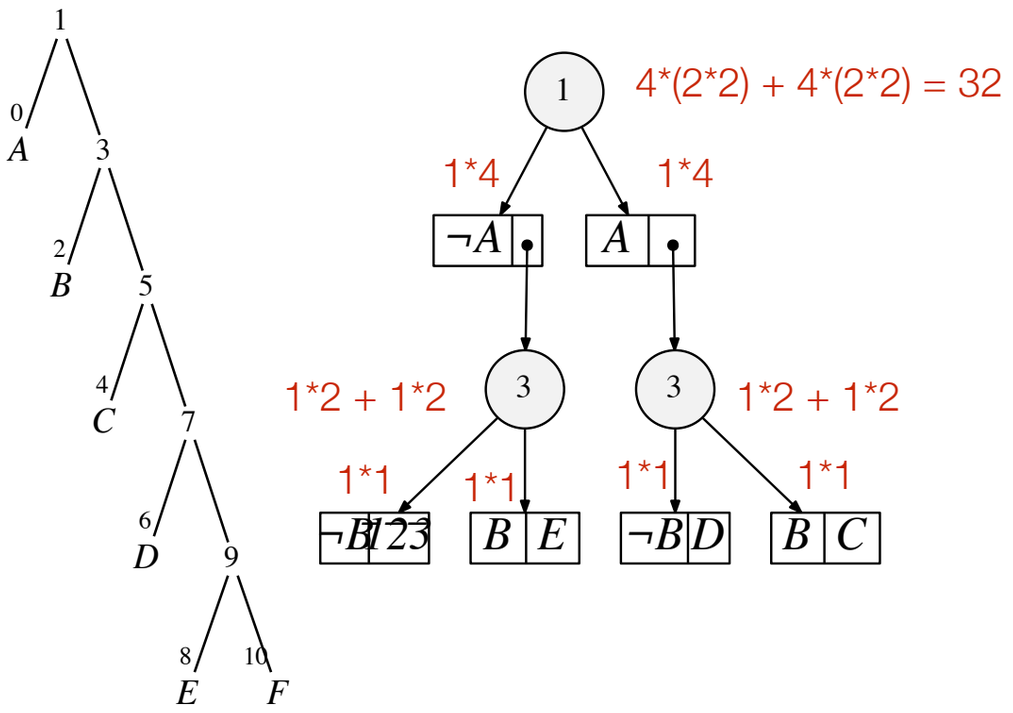
KU LEUVEN

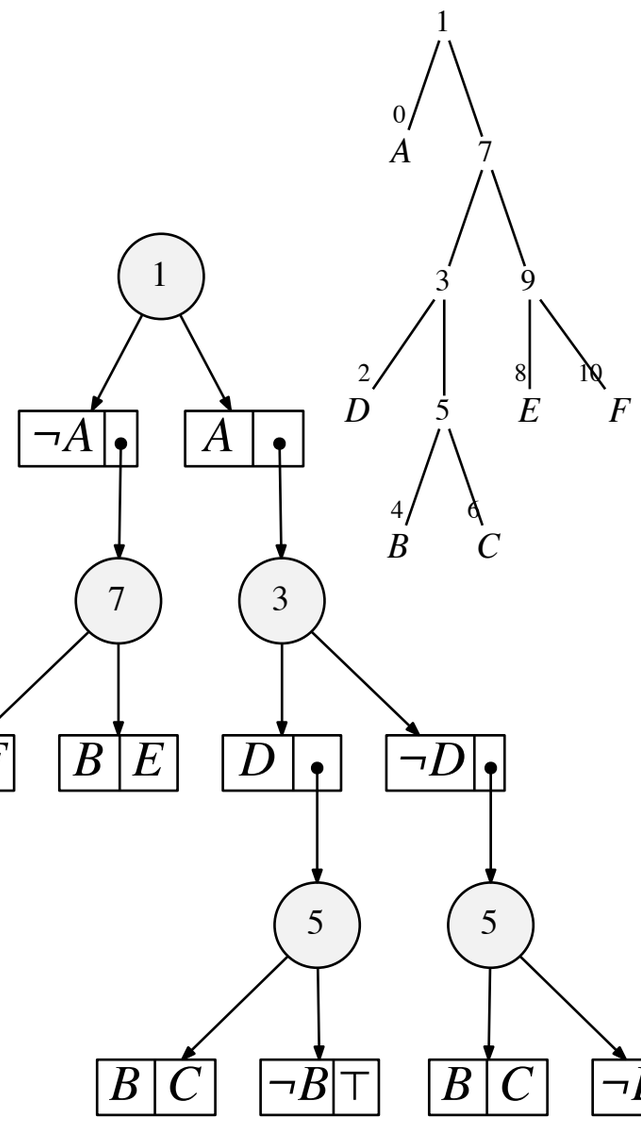
Example: Effect of Ordering (vtree)

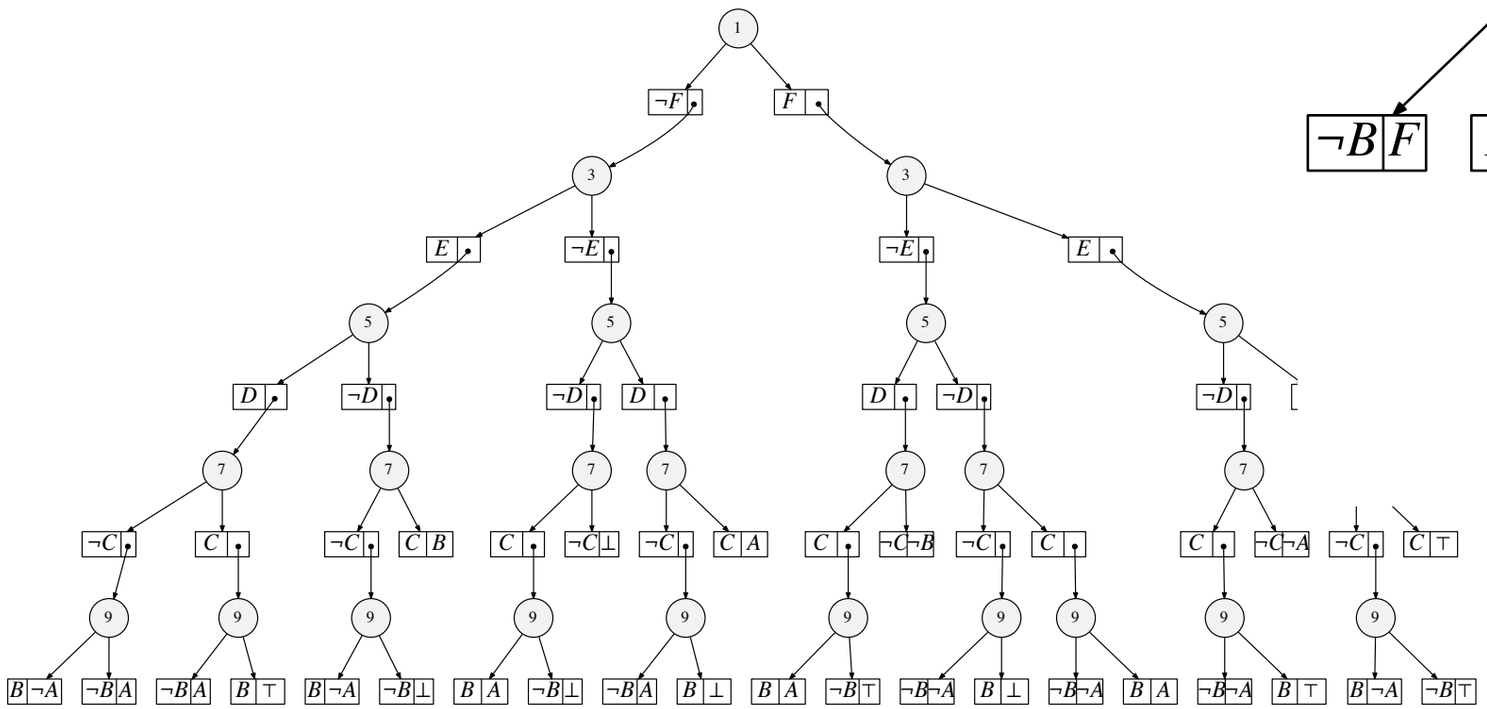
CNF

```
-A v -B v C
-A v  B v D
 A v -B v E
 A v  B v F
```







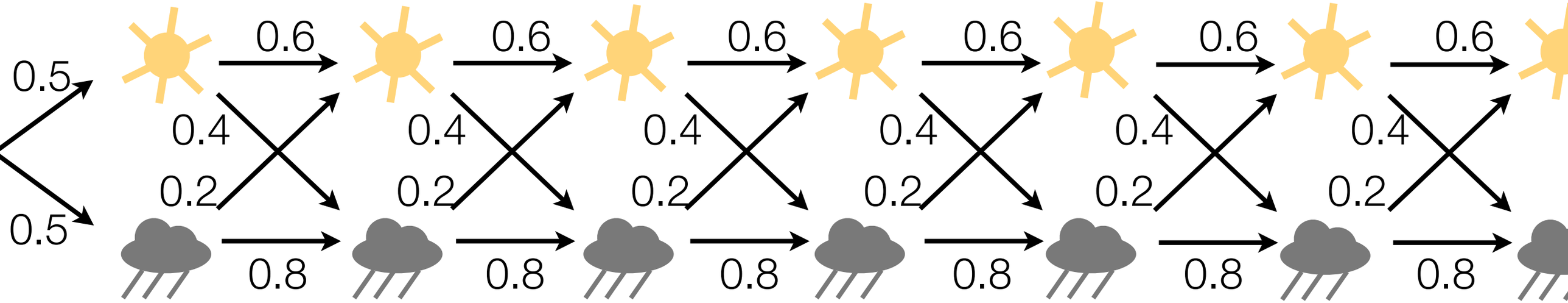


ProbLog Inference

- reduction to propositional formula
- addresses disjoint-sum-problem
- **but**: not all probabilistic logic programs face this problem! e.g., weather

ProbLog by example:

Rain or sun?



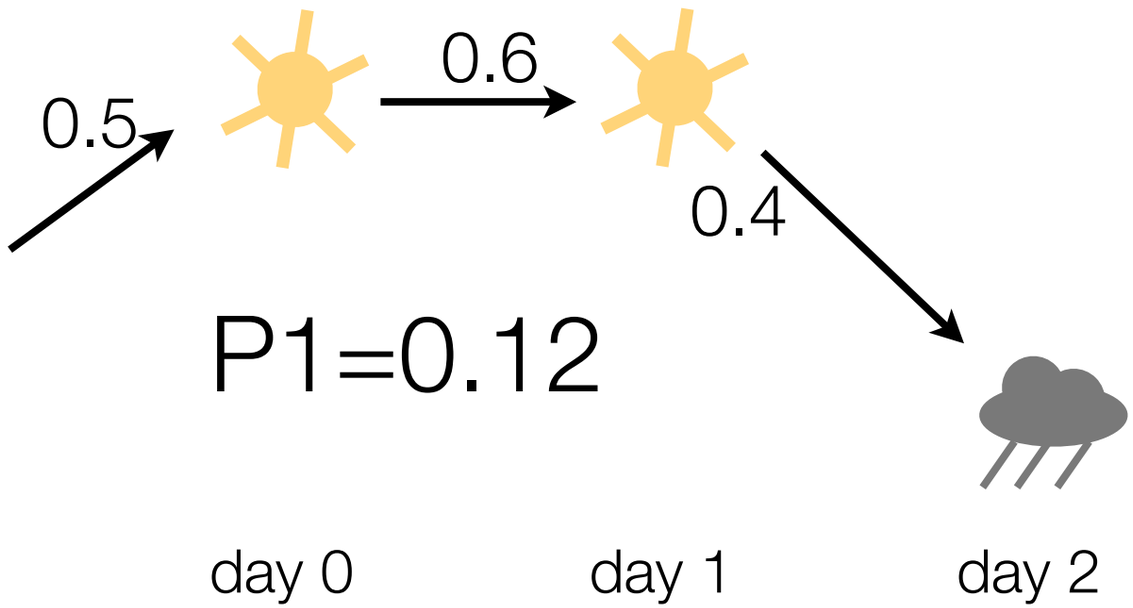

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(sun,Tprev).
0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    <- T>0, Tprev is T-1, weather(rain,Tprev).
```

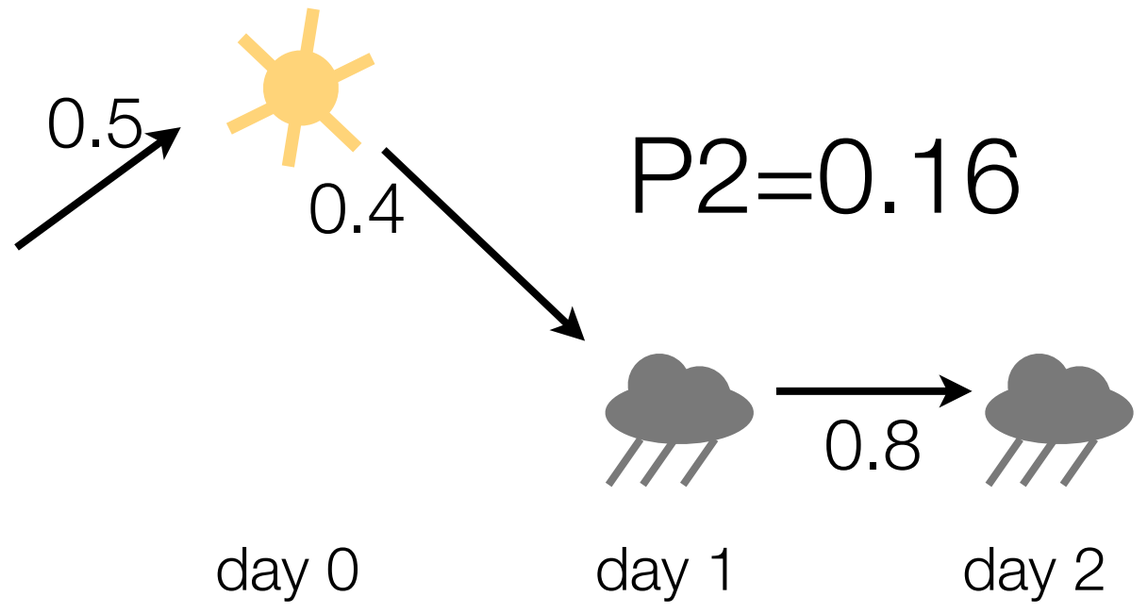
in every possible world, at most one proof for a ground query → summing proof probabilities is correct!

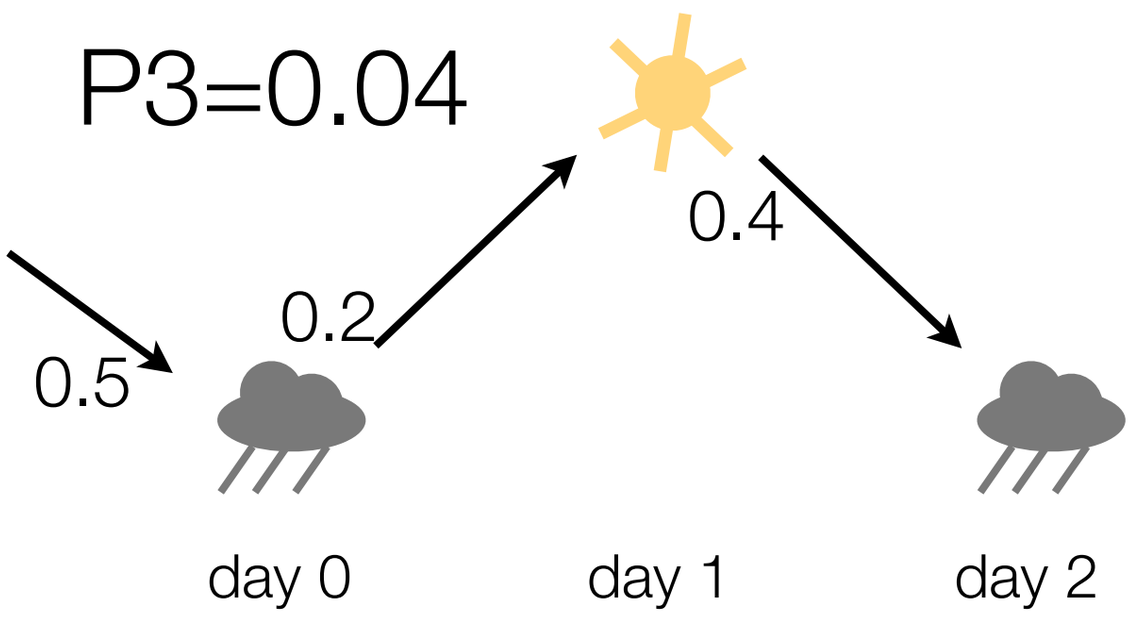
Possible worlds

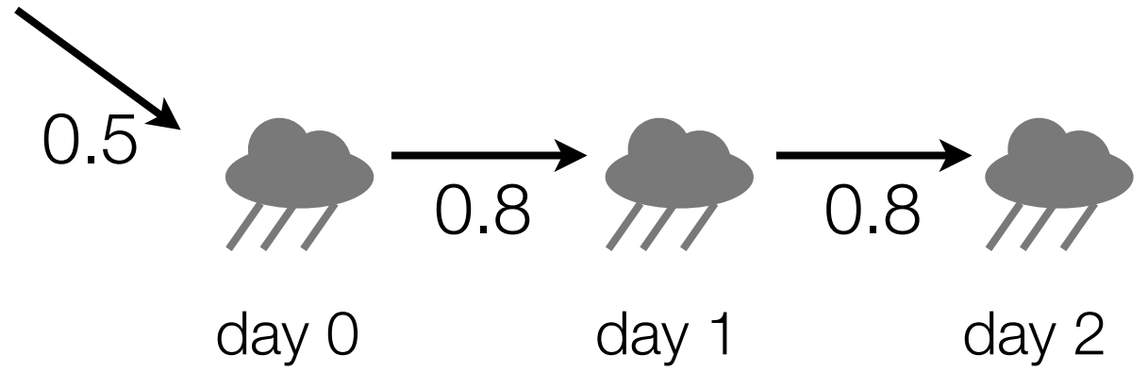
```
?- weather(rain,2).
```

P=P1+P2+P3+P4









Mutually Exclusive Rules:

no two rules apply simultaneously

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                :- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                :- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                :- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                :- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                :- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                :- T>0, Tprev is T-1, weather(rain,Tprev).
```

Mutually Exclusive Rules:

no two rules apply simultaneously

first rule for day 0, others for later days

day 0: either sun or rain

```
0.5::weather(sun,0) ; 0.5::weather(rain,0) :- true.

0.6::weather(sun,T) ; 0.4::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(sun,Tprev).

0.2::weather(sun,T) ; 0.8::weather(rain,T)
                    :- T>0, Tprev is T-1, weather(rain,Tprev).
```

rules for T>0 cover mutually exclusive cases on previous day

Learning

- Parameter learning
- Upgrading relational learning

Parameter Learning

e.g., webpage classification model

for each ***CLASS1, CLASS2*** and each ***WORD***

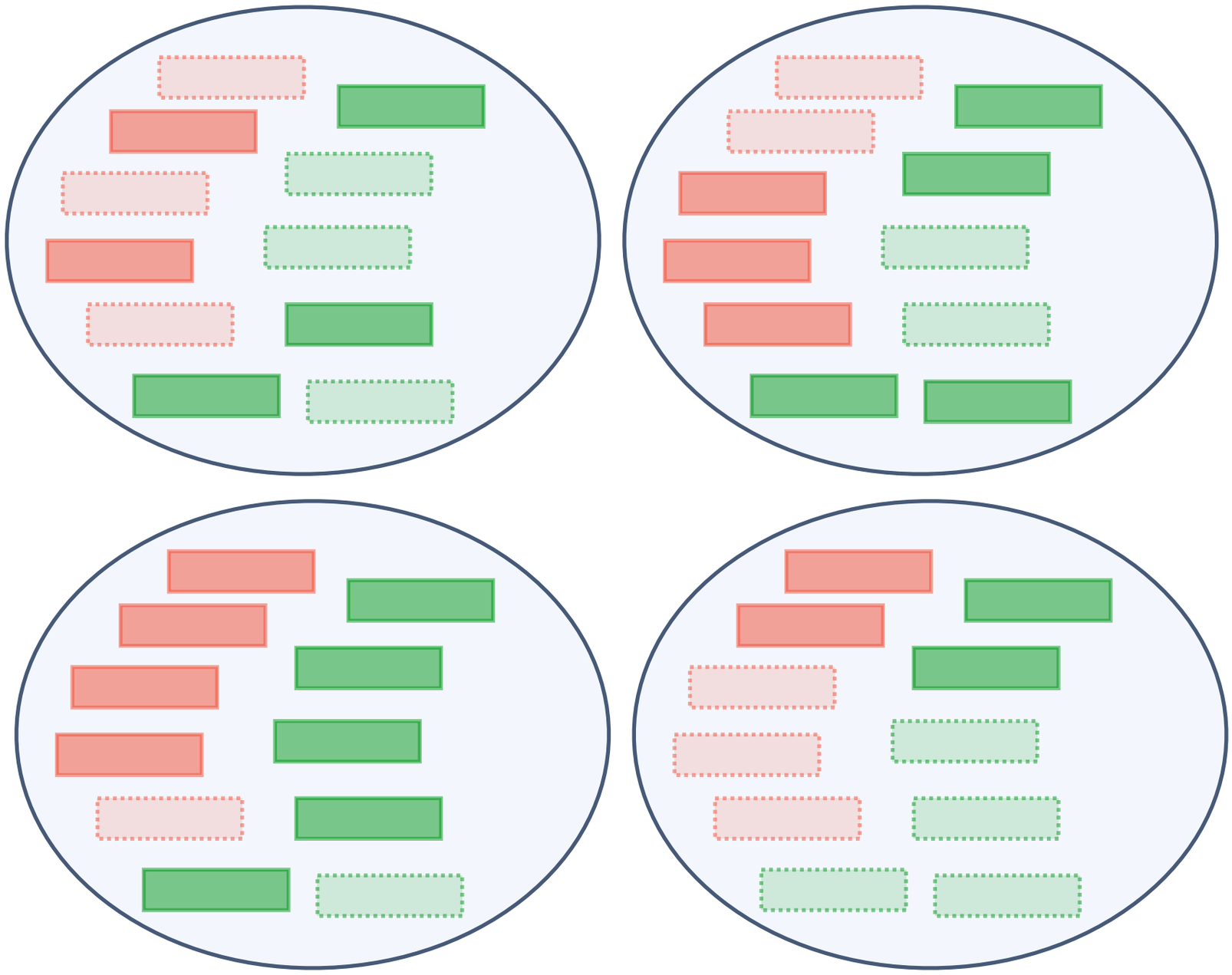
?? :: link_class(Source,Target,***CLASS1***,***CLASS2***).

?? :: word_class(***WORD***,***CLASS***).

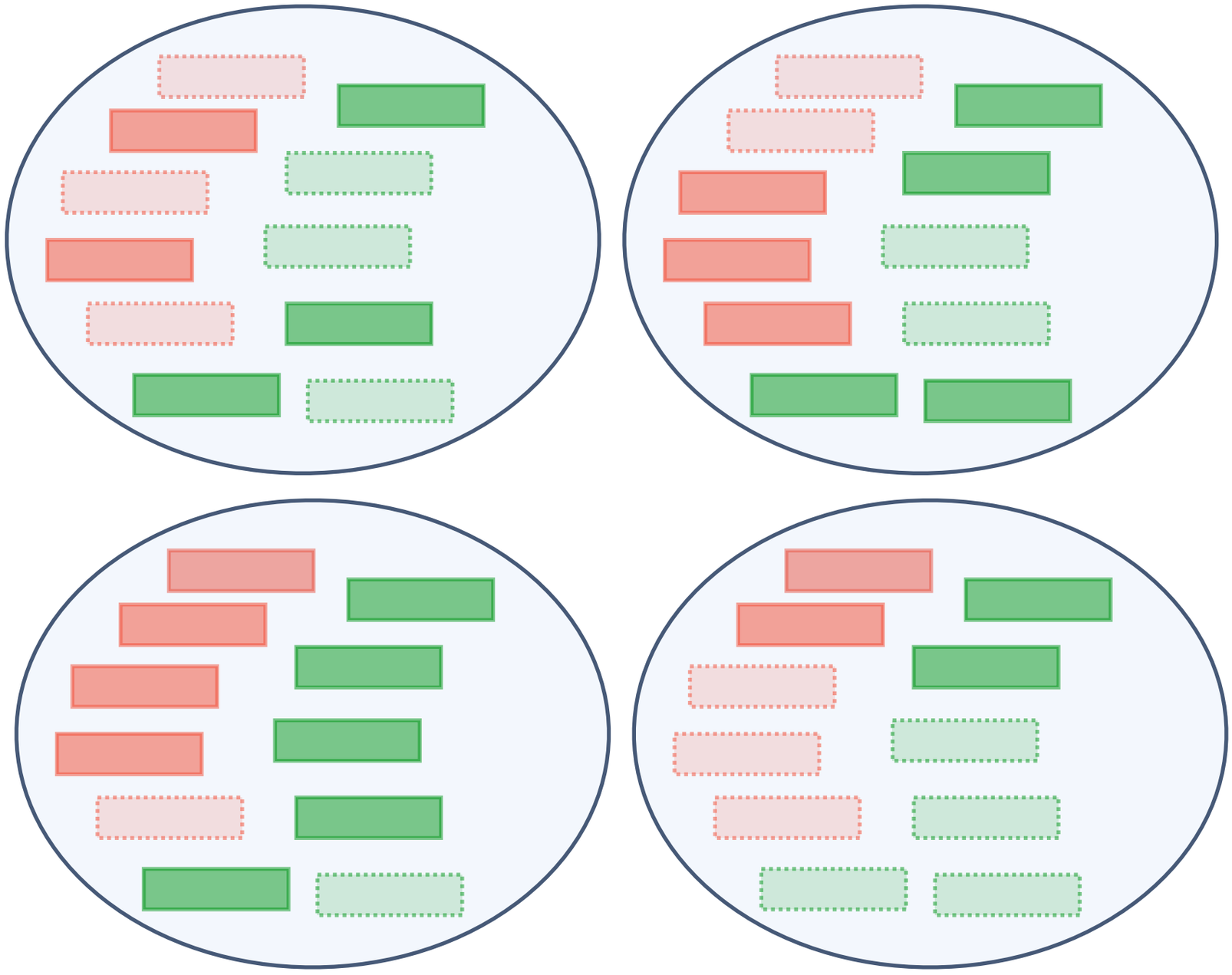
```
class(Page,C)  :-  has_word(Page,W), word_class(W,C).

class(Page,C)      :-  links_to(OtherPage,Page),
    class(OtherPage,OtherClass),
                       link_class(OtherPage,Page,OtherClass,C).
```

Sampling Interpretations

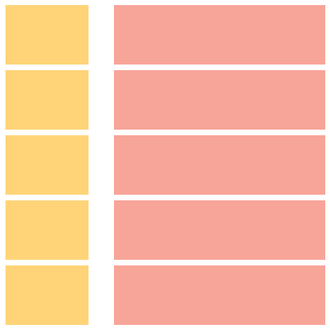
Sampling Interpretations

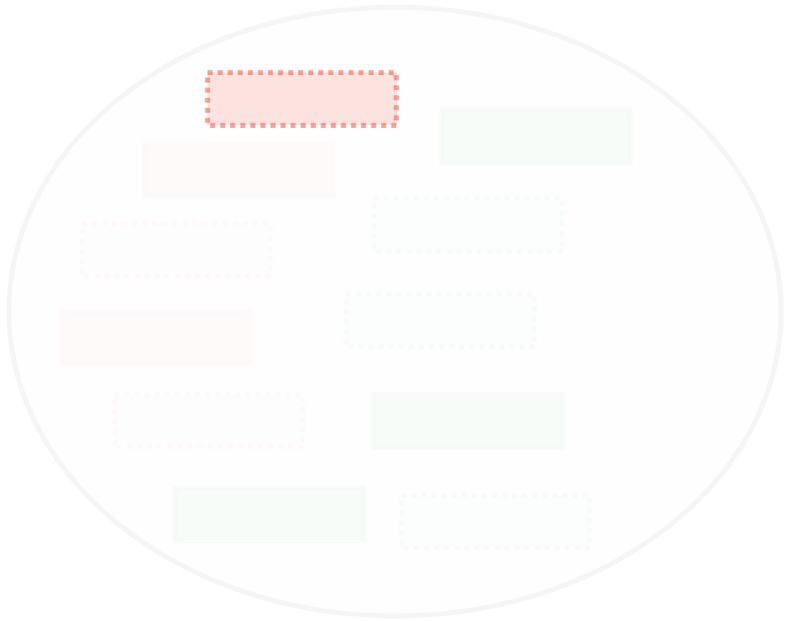


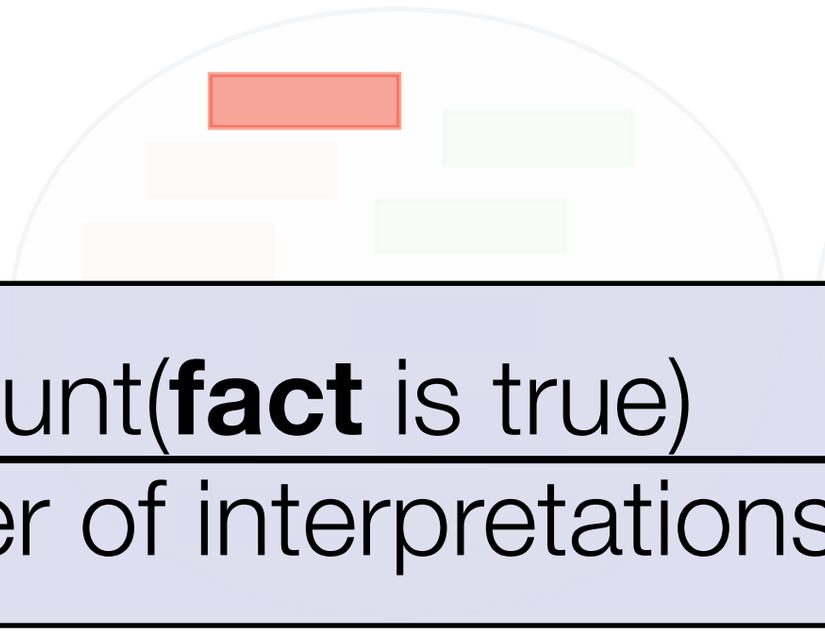
Parameter Estimation



Parameter Estimation

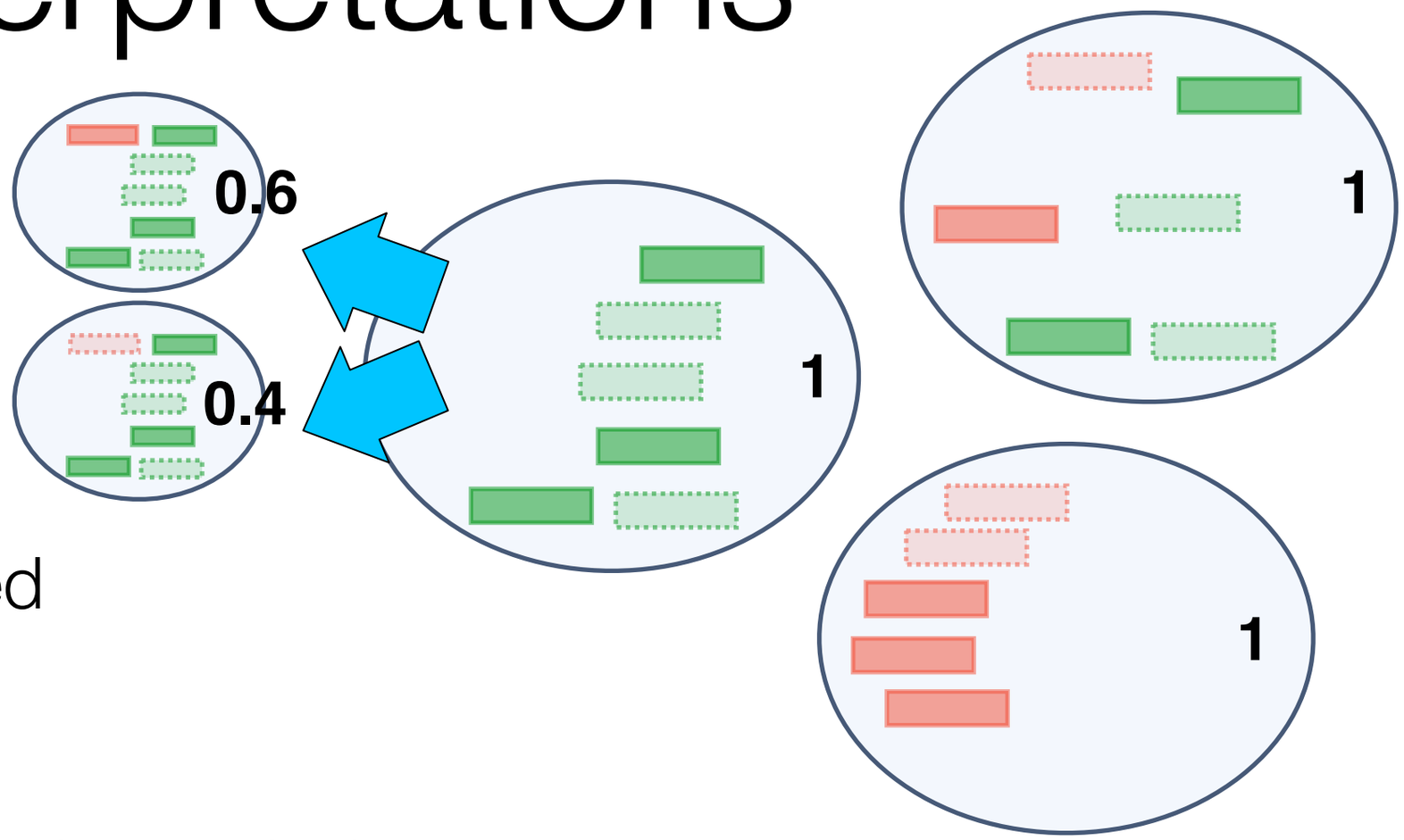






$$p(\textbf{fact}) = \frac{\text{count}(\textbf{fact} \text{ is true})}{\text{Number of interpretations}}$$

Learning from partial interpretations



- Not all facts observed
- Soft-EM
- use **expected count** instead of **count**
- **P(Q |E) — conditional queries !**

[Gutmann et al, ECML 11; Fierens et al, TPLP 14

Upgrading relational learning

	Prolog
	
	`infl(a,b).`
	`...`
		ProbLog
		
		`0.4::infl(a,b).`
		`...`

Reasoning	query true? yes/no	query true? with probability P

Upgrading relational learning

	Prolog
	
	`infl(a,b).`
	`...`
		ProbLog
		
		`0.4::infl(a,b).`
		`...`

Reasoning	query true? yes/no	query true? with probability P
Machine Learning	example covered? yes/no	example covered? with probability P

Structure Learning: ProbFOIL

Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.7::father(piet, wim).
0.9::father(bart, pieter).
0.6::father(tom, greet).
mother(emma,piet).
mother(emma,greet).
mother(greet,pieter).
grandmother(emma,ilse).
0.7::grandmother(emma,wim).
....
```

[De Raedt and Thon, ILP 10; Dries et al]

Structure Learning: ProbFOIL

Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.7::father(piet, wim).
0.9::father(bart, pieter).
0.6::father(tom, greet).
mother(emma,piet).
mother(emma,greet).
mother(greet,pieter).
grandmother(emma,ilse).
0.7::grandmother(emma,wim).
....
```

```
grandmother(X,Y) :-
    mother(X,Z),
    father(Z,Y).
```

[De Raedt and Thon, ILP 10; Dries et al]

Structure Learning: ProbFOIL

Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.4987::rain(1).
0.3591::windok(1).
0.4534::sunshine(1).
0.3257::surfing(1).
0.7391::rain(2).
0.6022::windok(2).
0.9837::sunshine(2).
0.2592::surfing(2).
0.2898::rain(3).
0.7423::windok(3).
0.2275::sunshine(3).
0.5688::surfing(3)....
```

[De Raedt and Thon, ILP 10; Dries et al]

Structure Learning: ProbFOIL

Upgrading FOIL to learn a set of rules from **probabilistic** facts

```
0.4987::rain(1).
0.3591::windok(1).
0.4534::sunshine(1).
0.3257::surfing(1).
0.7391::rain(2).
0.6022::windok(2).
0.9837::sunshine(2).
0.2592::surfing(2).
0.2898::rain(3).
0.7423::windok(3).
0.2275::sunshine(3).
0.5688::surfing(3)....
```

```
surfing(X) :- \+ rain(X), windok(X).
surfing(X) :- \+ rain(X), sunshine(X).
```

[De Raedt and Thon, ILP 10; Dries et al]

Prob2FOIL

- Learning **probabilistic rules** from probabilistic facts

```
0.4987::rain(1).
0.3591::windok(1).
0.4534::sunshine(1).
0.3257::surfing(1).
0.7391::rain(2).
0.6022::windok(2).
0.9837::sunshine(2).
0.2592::surfing(2).
0.2898::rain(3).
0.7423::windok(3).
0.2275::sunshine(3).
0.5688::surfing(3).
...
```

```
$ pip install probfoil
```

Prob2FOIL

- Learning **probabilistic rules** from probabilistic facts

```
0.4987::rain(1).
0.3591::windok(1).
0.4534::sunshine(1).
0.3257::surfing(1).
0.7391::rain(2).
0.6022::windok(2).
0.9837::sunshine(2).
0.2592::surfing(2).
0.2898::rain(3).
0.7423::windok(3).
0.2275::sunshine(3).
0.5688::surfing(3).
...
```

```
0.7023::surfing(A)  :- \+rain(A).
0.01243::surfing(A) :- true.
```

```
$ pip install probfoil
```

Language Extensions and Variants

- Dynamics under uncertainty
- Continuous-valued random variables
- Decision making
- Constraint

Language Extensions and Variants

not included in ProbLog system

variants

Causal Probabilistic Time-Logic *dynamics*

Distributional Clauses *continuous distributions*

ProbLog

extensions

utilities and decisions Decision-Theoretic ProbLog

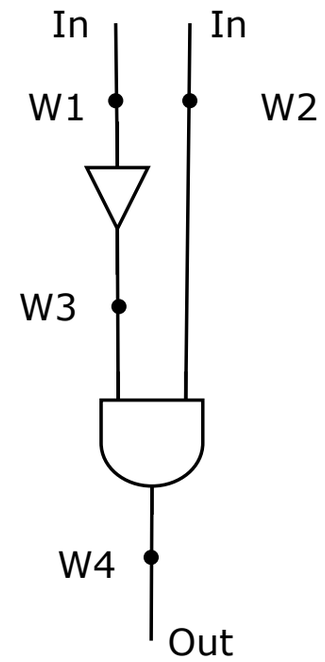
constraints ProbLog with constraints

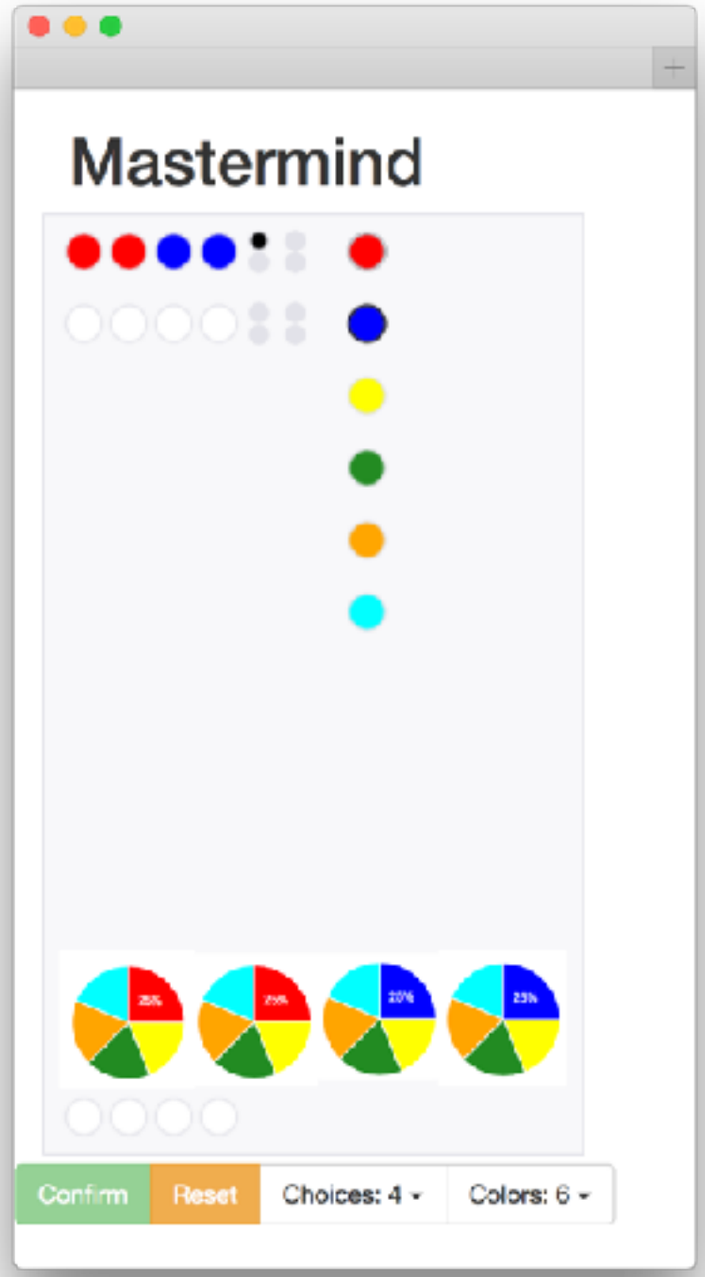
Dynamic ProbLog

Efficient Probabilistic Inference for Dynamic Relational Models

Jonas Vlasselaer, Wannes Meert, Guy Van den Broeck and Luc De Raedt

```
wire(1). wire(2).
wire(3). wire(4).
in(1). in(2). out(4).
gate(a,not,[1],3).
gate(b,and,[3,2],4).

0.990::healthy(G,0) :- gate(G,_,_,_).
0.990::healthy(G,T) :- T>0,    healthy(G,T-1).
0.001::healthy(G,T) :- T>0, \+healthy(G,T-1).

0.5::high(W,T) :- in(W).
0.5::high(W,T) :- gate(G,_,_,W), \+healthy(G,T).
high(W,T) :- gate(G,not,[I],W), healthy(G,T),
             \+high(I,T).
high(W,T) :- gate(G,and,[I,J],W), healthy(G,T),
             high(I,T), high(J,T).

input(W,T) :- in(W), high(W,T).
output(W,T) :- out(W), high(W,T).
```





Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

random variable with Gaussian distribution

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
    ≃length(OBot) ≥ ≃length(OTop),
    ≃width(OBot) ≥ ≃width(OTop).
```

comparing values of random variables

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                        :- obj(Obj), on(Obj,O2), type(O2,plate).
```

random variable with discrete distribution

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables
- Inference: particle filter

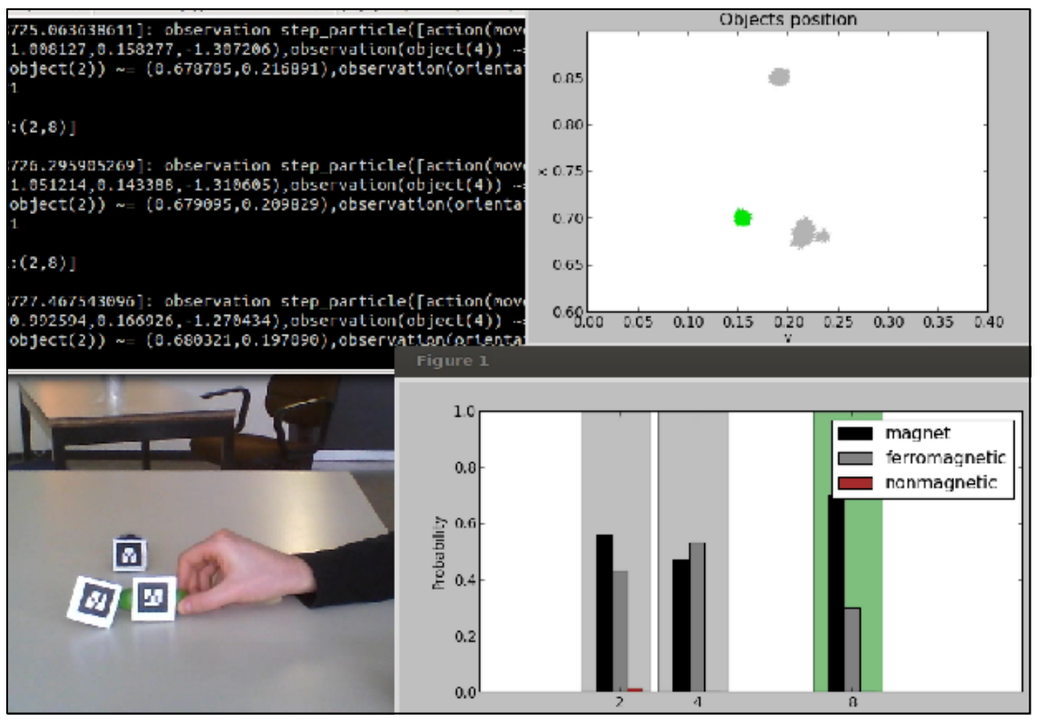
```
length(Obj) ~ gaussian(6.0,0.45) :- type(Obj,glass).
stackable(OBot,OTop) :-
      ≃length(OBot) ≥ ≃length(OTop),
      ≃width(OBot) ≥ ≃width(OTop).
ontype(Obj,plate) ~ finite([0 : glass, 0.0024 : cup,
                            0 : pitcher, 0.8676 : plate,
                            0.0284 : bowl, 0 : serving,
                            0.1016 : none])
                    :- obj(Obj), on(Obj,O2), type(O2,plate).
```

[Gutmann et al, TPLP 11; Nitti et al, IROS 13]

Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions

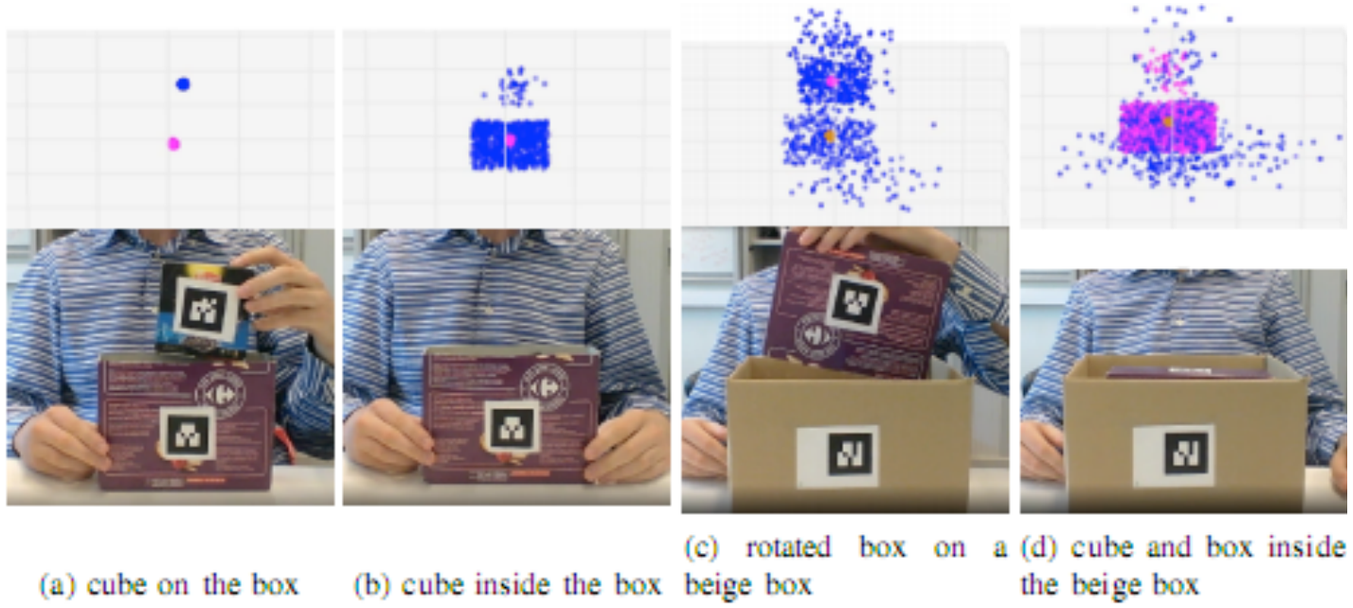


[Nitti et al, IROS 13]

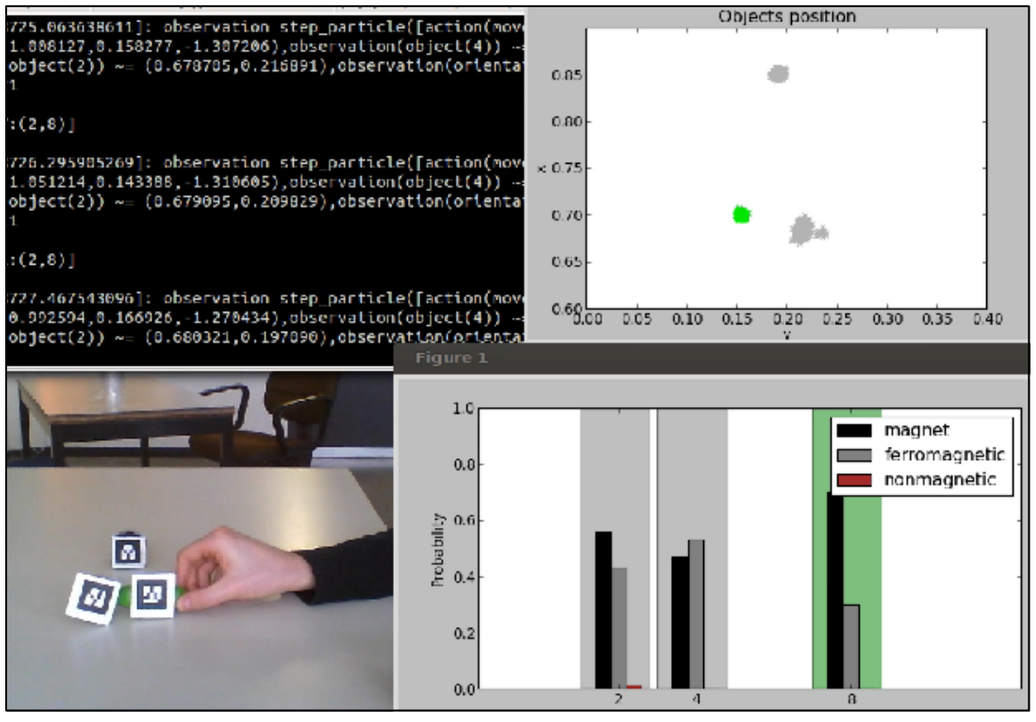
Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions



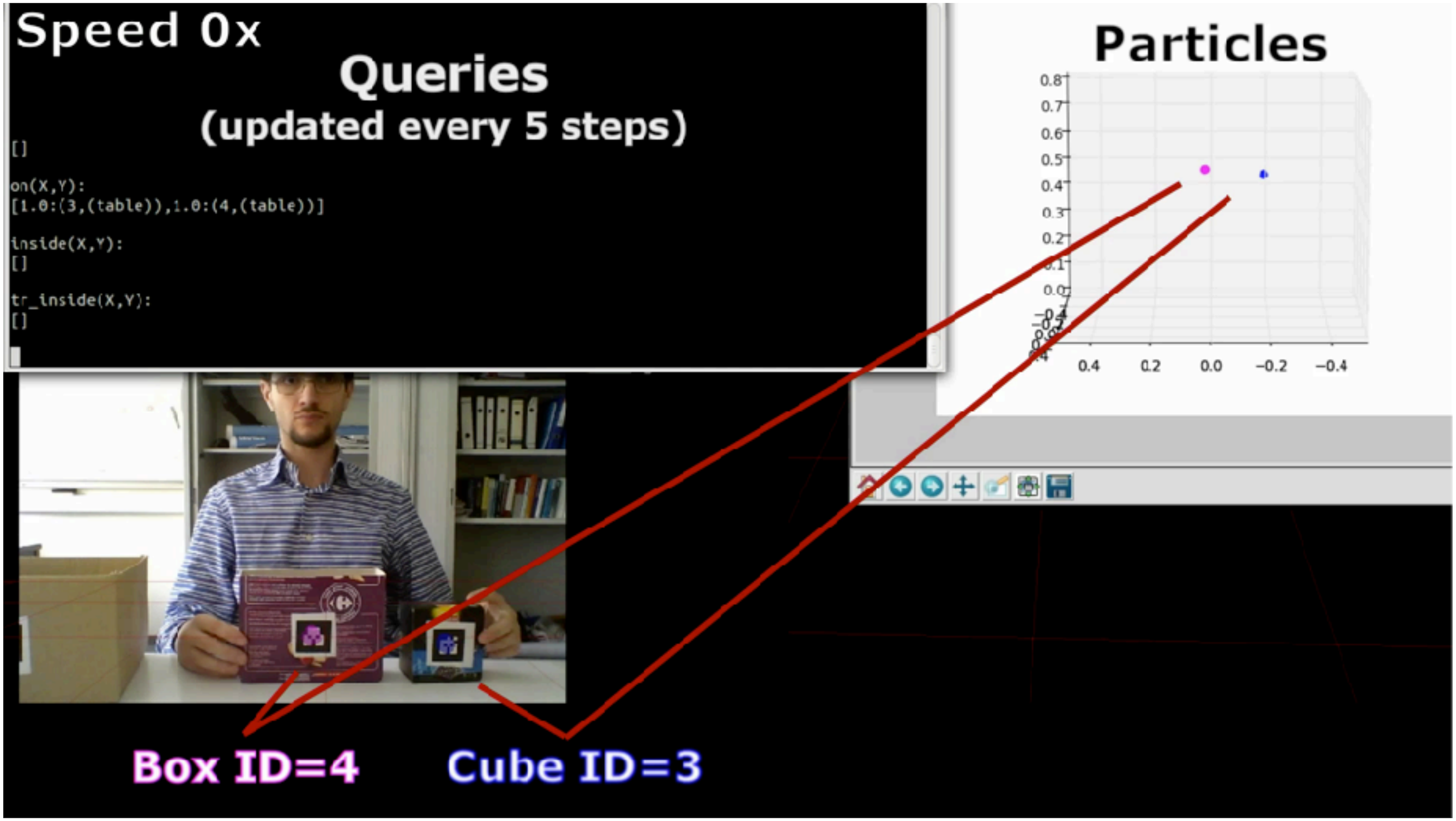
(a) cube on the box (b) cube inside the box (c) rotated box on a beige box (d) cube and box inside the beige box



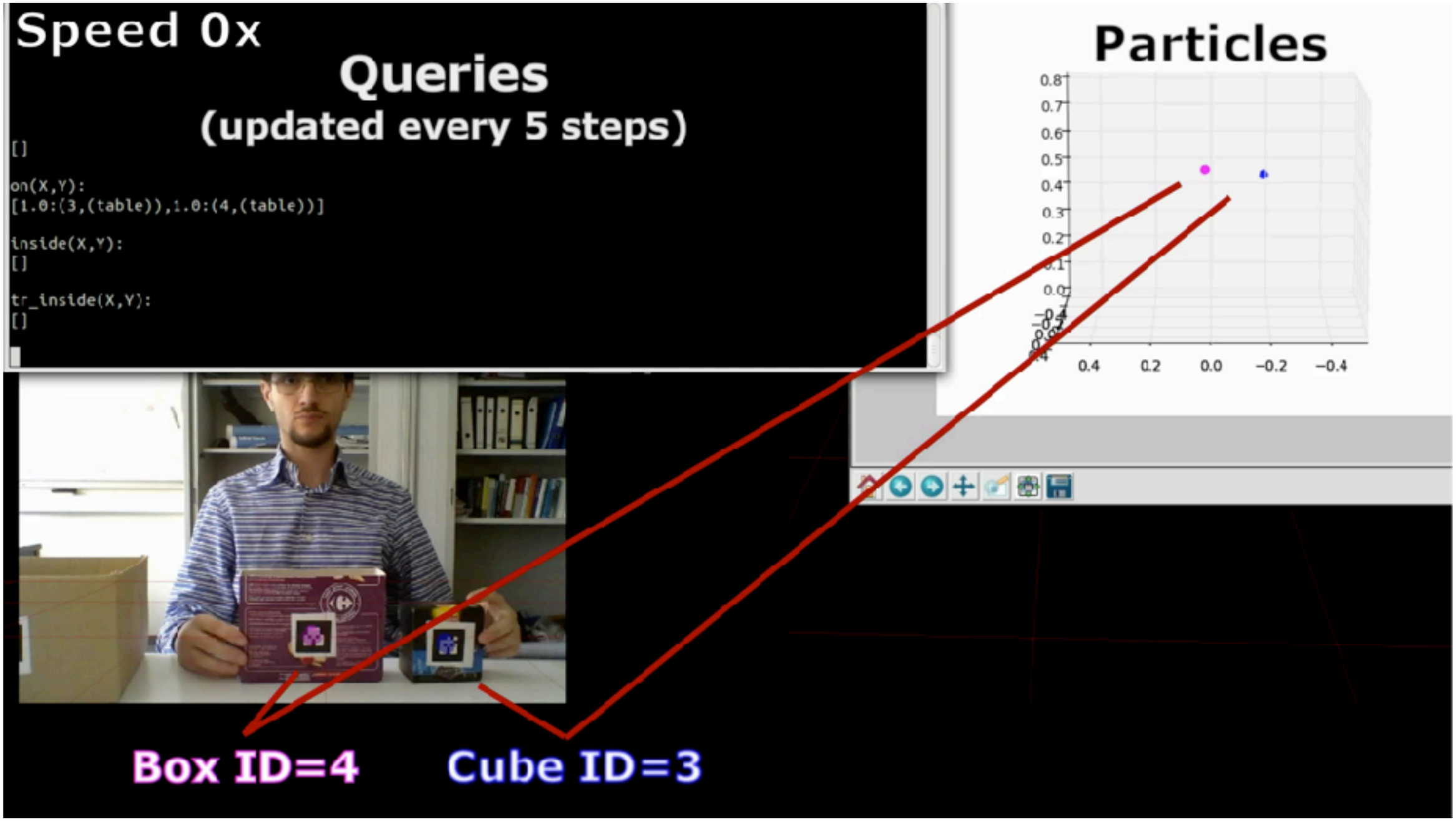
Box scenario

- object tracking even when invisible
- estimate spatial relations

[Nitti et al, IROS 13]


Speed 0x
Queries
(updated every 5 steps)
[]
on(X,Y):
[1.0:(3,(table)),1.0:(4,(table))]
inside(X,Y):
[]
tr_inside(X,Y):
[]
Particles
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.4
0.2
0.0
−0.2
−0.4
Box ID=4
Cube ID=3

IROS 13

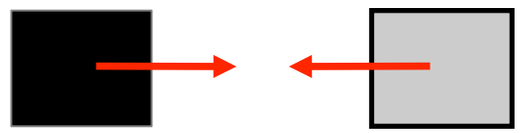

Speed 0x
Queries
(updated every 5 steps)
[]
on(X,Y):
[1.0:(3,(table)),1.0:(4,(table))]
inside(X,Y):
[]
tr_inside(X,Y):
[]
Particles
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.4
0.2
0.0
−0.2
−0.4
Box ID=4
Cube ID=3

IROS 13

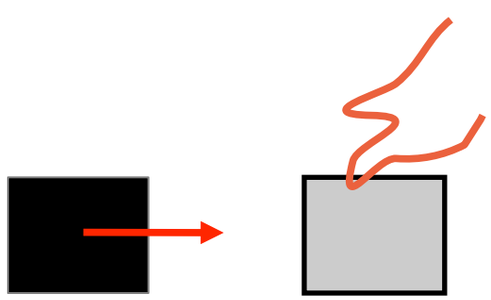
Magnetic scenario

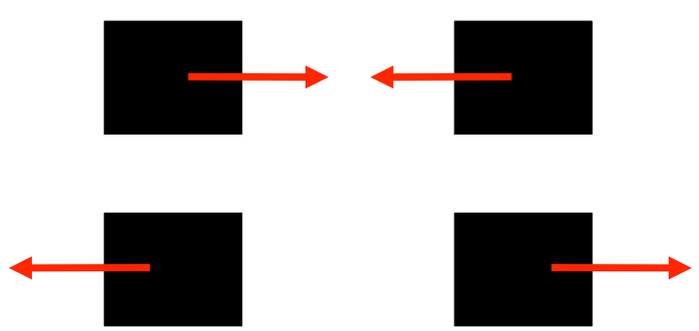
- 3 object types: magnetic, ferromagnetic, nonmagnetic

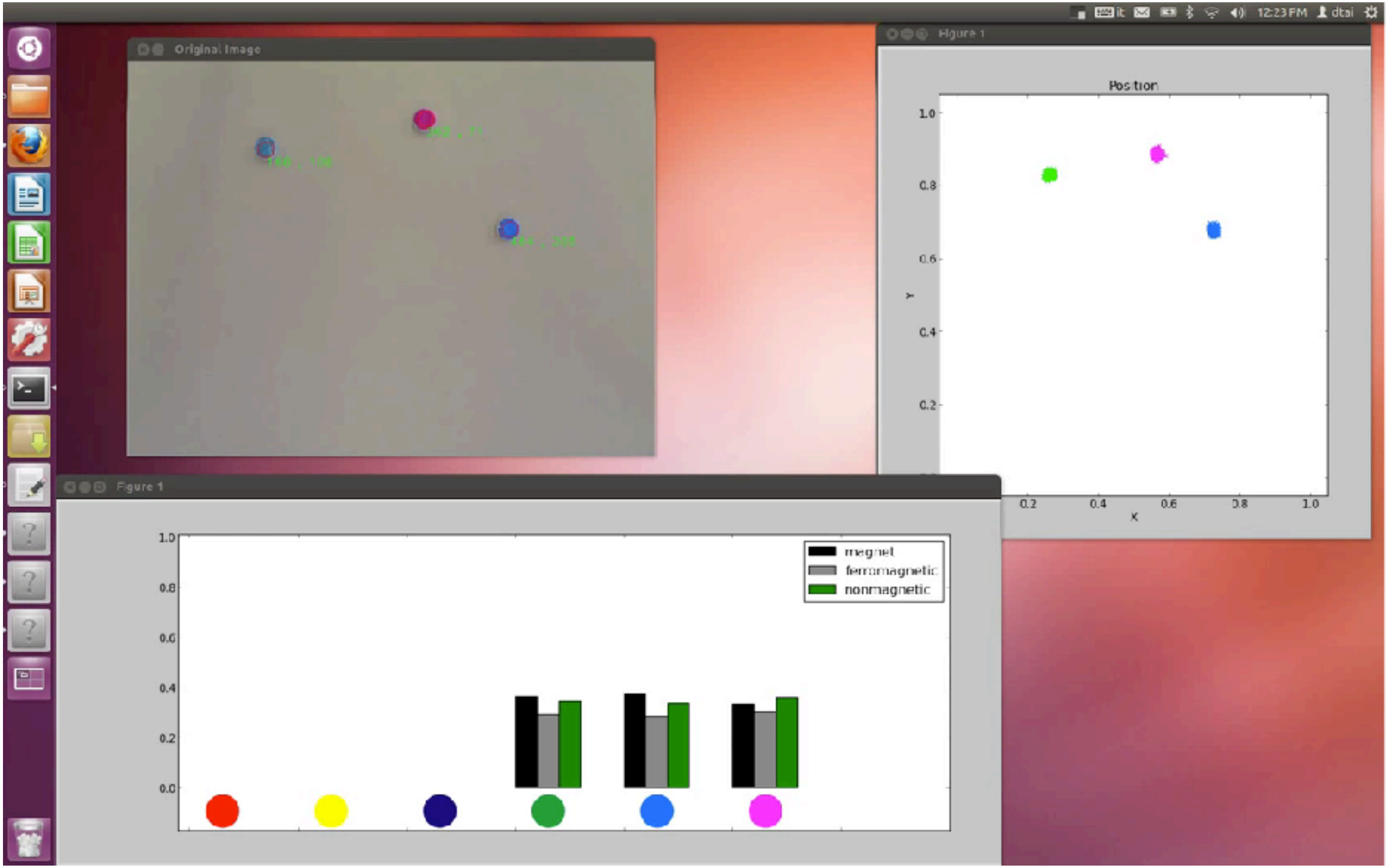
- Nonmagnetic objects do not interact
- A magnet and a ferromagnetic object attract each other

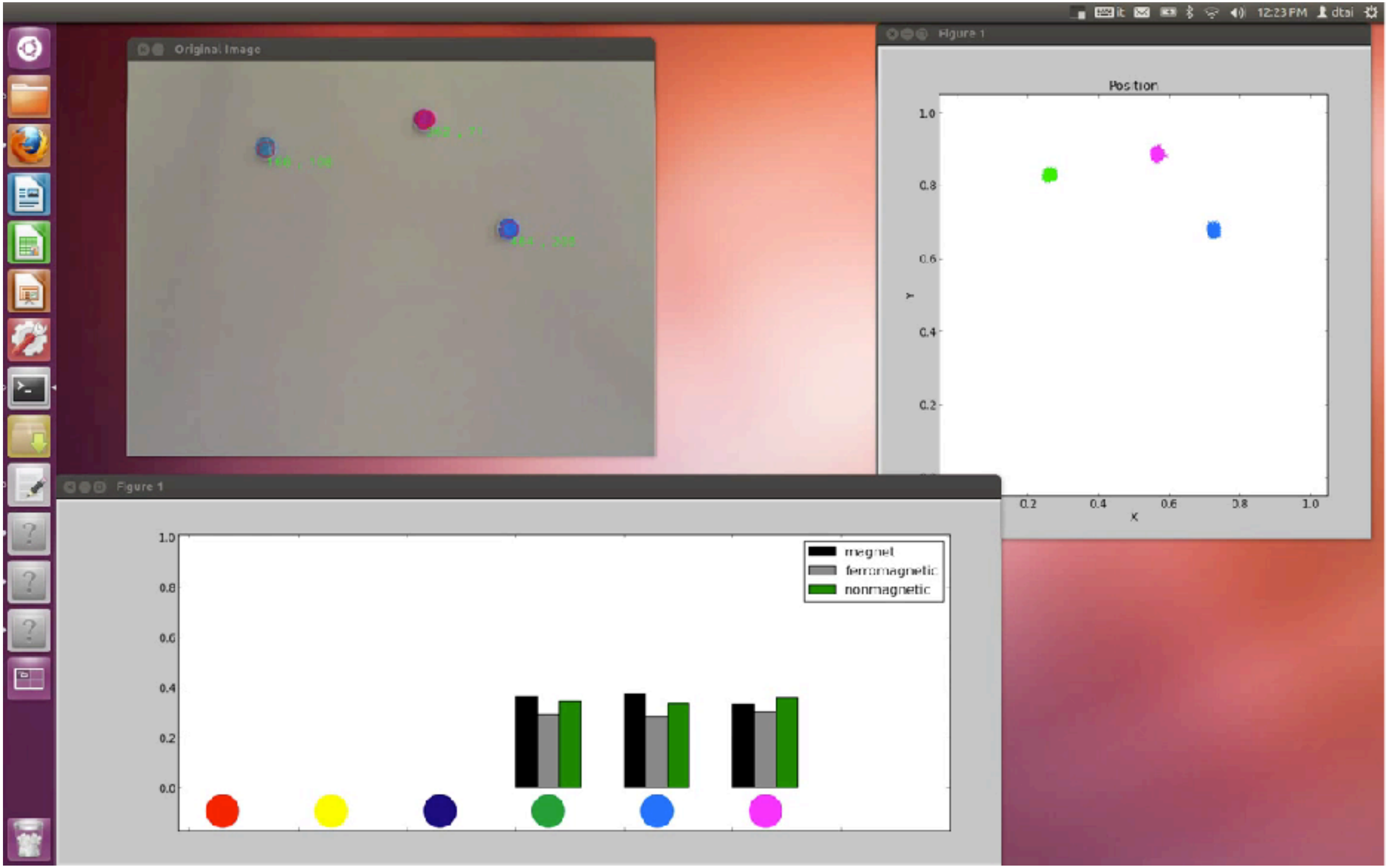


- Magnetic force that depends on the distance
- If an object is held magnetic force is compensated.

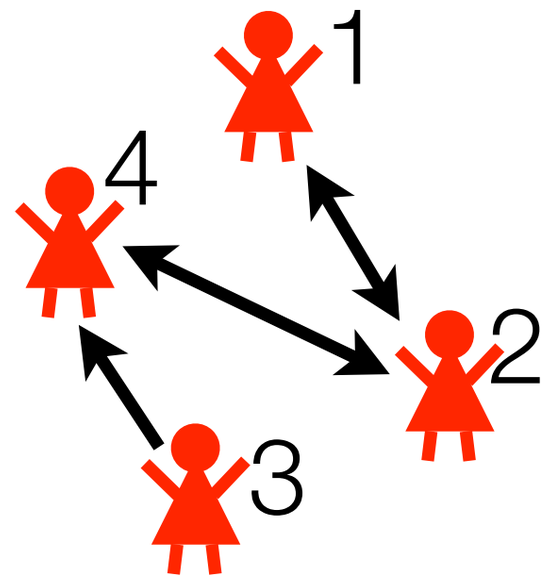





Original Image
Figure 1
Position
1.0
0.8
0.6
0.4
0.2
y
0.2
0.4
0.6
0.8
1.0
x
Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
magnet
ferromagnetic
nonmagnetic


Original Image
Figure 1
Position
1.0
0.8
0.6
0.4
0.2
y
0.2
0.4
0.6
0.8
1.0
x
Figure 1
1.0
0.8
0.6
0.4
0.2
0.0
magnet
ferromagnetic
nonmagnetic

DTProbLog

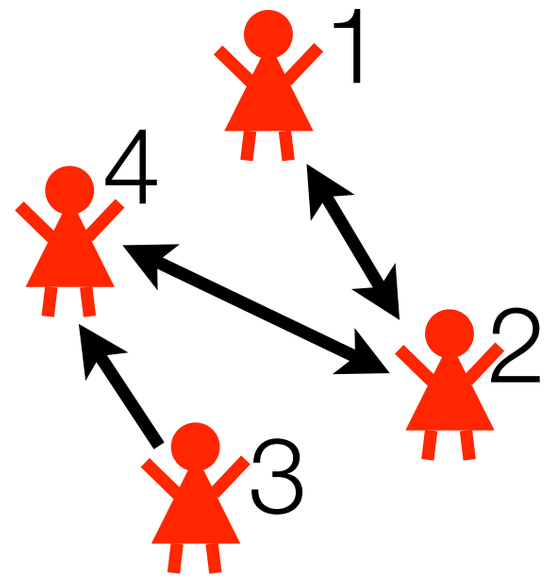



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).
```

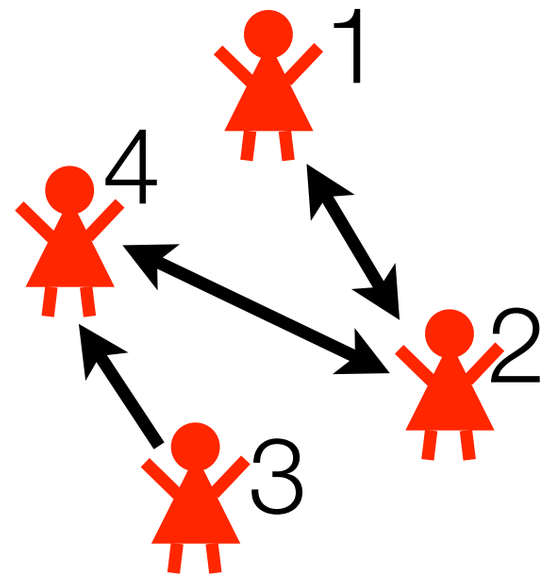
decision fact: true or false?




```
person(1).
person(2).
person(3).
person(4).
```

```
friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog




```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).
```

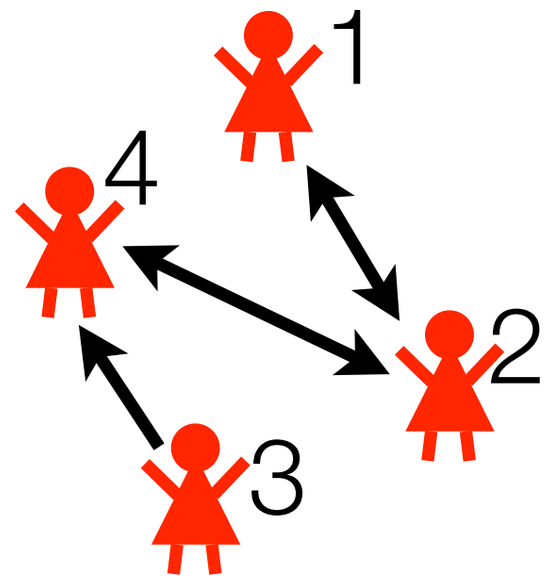
probabilistic facts
+ logical rules

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

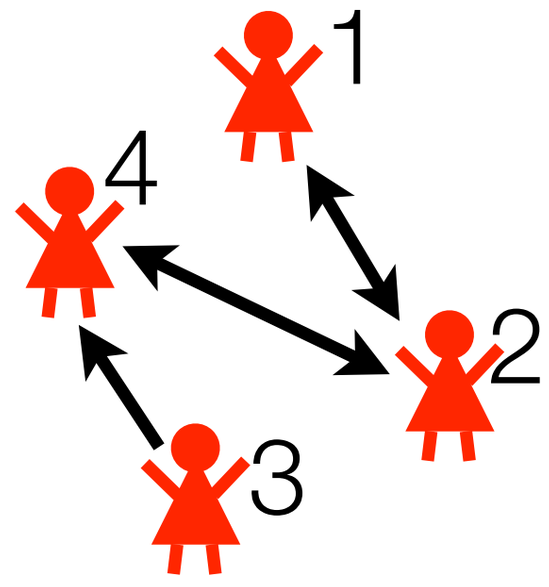
utility facts: cost/reward if true




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog

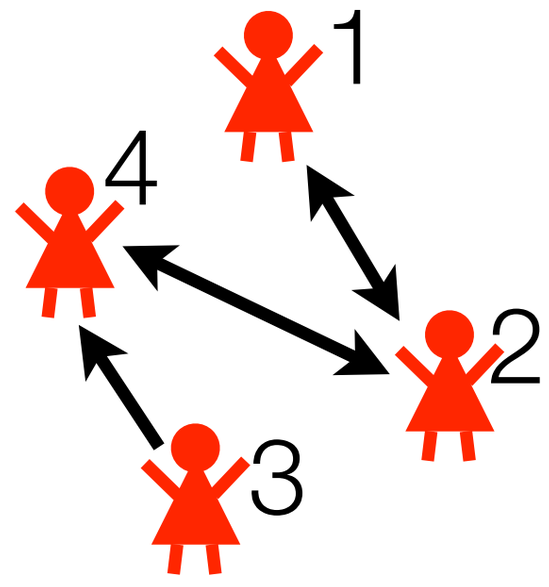
```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

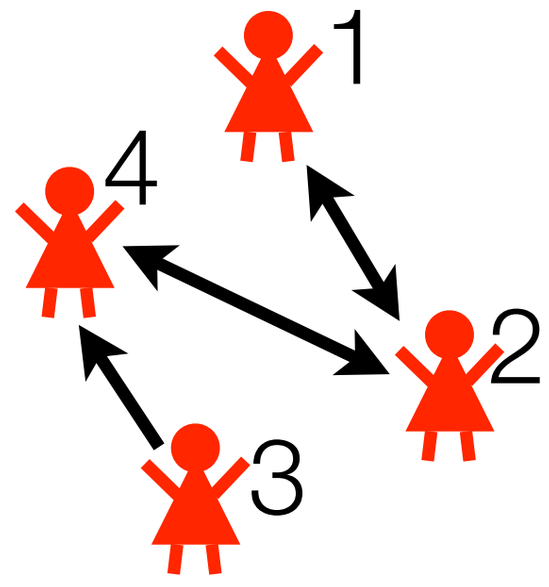
DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

DTProbLog



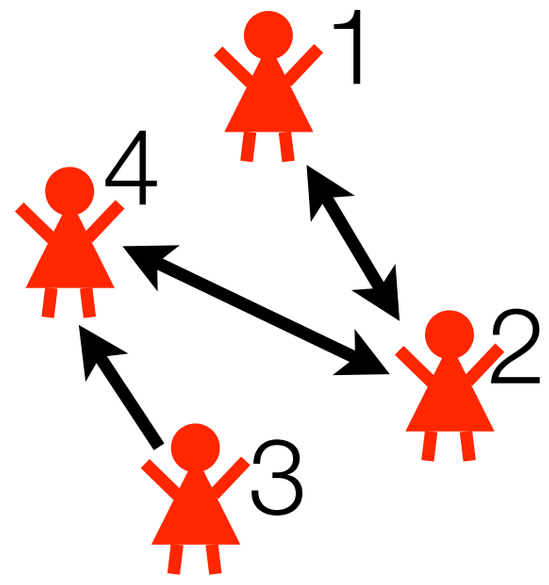

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

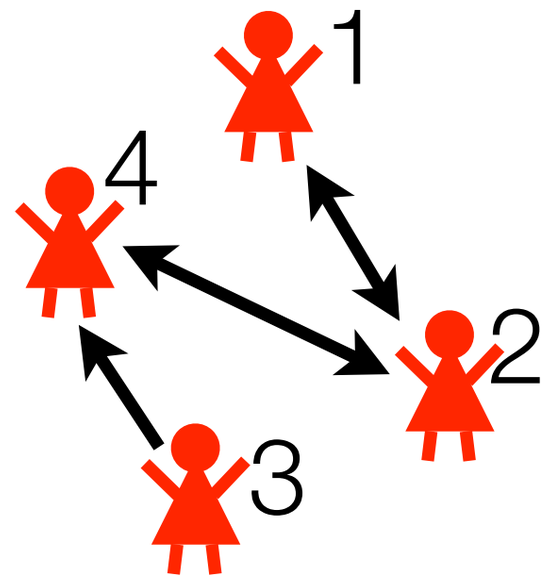



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)          bm(1)
```

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```

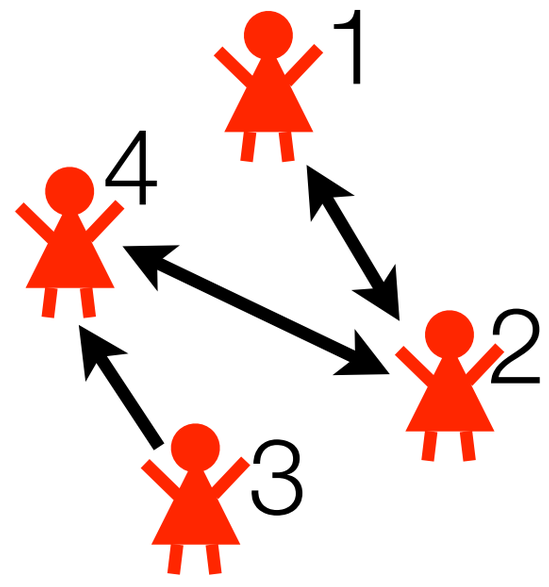



```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

```
marketed(1)        marketed(3)

    bt(2,1)   bt(2,4)        bm(1)

buys(1)    buys(2)
```

DTProbLog

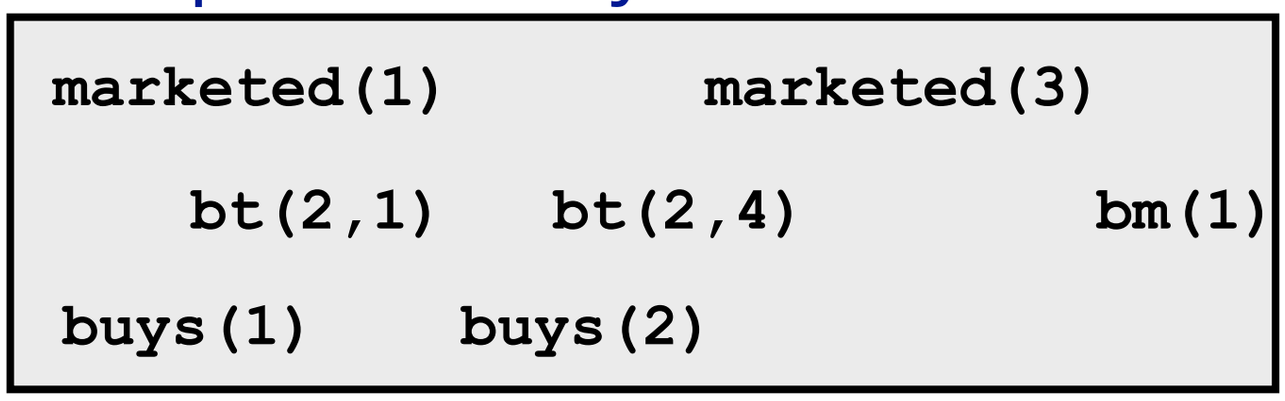
```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

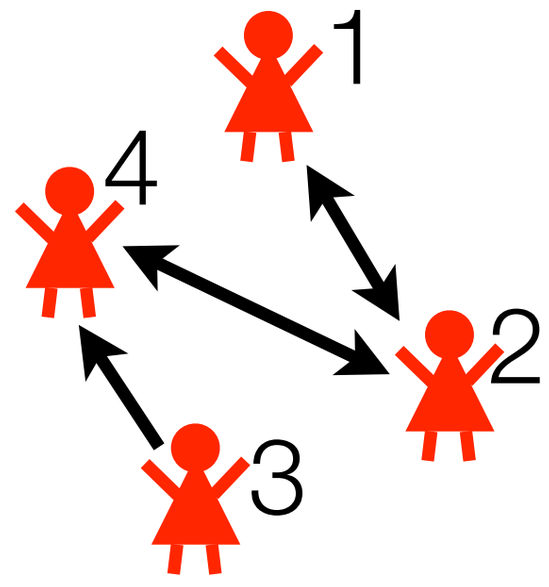
utility = –3 + –3 + 5 + 5 = 4

probability = 0.0032



DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

utility = –3 + –3 + 5 + 5 = 4

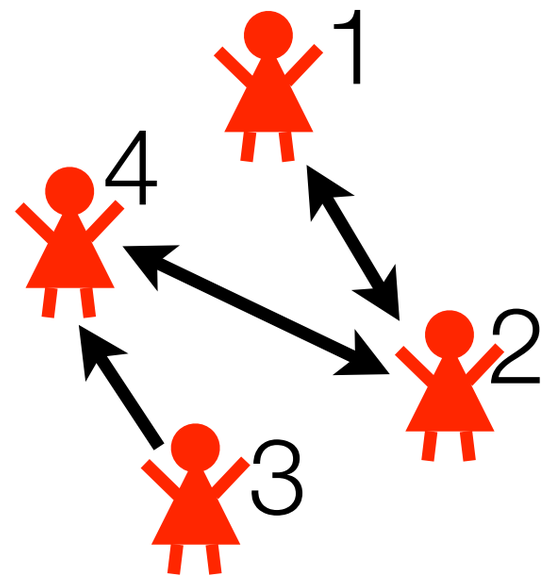
probability = 0.0032

```
marketed(1)        marketed(3)

   bt(2,1)   bt(2,4)          bm(1)

buys(1)    buys(2)
```

world contributes 0.0032×4 to expected utility of strategy

DTProbLog

```
? :: marketed(P) :- person(P).

0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

buys(X) :- friend(X,Y), buys(Y), buy_trust(X,Y).
buys(X) :- marketed(X), buy_marketing(X).

buys(P) => 5 :- person(P).
marketed(P) => -3 :- person(P).
```




```
person(1).
person(2).
person(3).
person(4).

friend(1,2).
friend(2,1).
friend(2,4).
friend(3,4).
friend(4,2).
```

task: find strategy that maximizes expected utility

solution: using ProbLog technology

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```


[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

distribution over **all** possible worlds

```
P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...

not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

sbhg e(10)	sb g e(10)	sbh e(10)	sb
s hg e(10)	s g	s h	s
bhg	b g	bh	b
hg	g	h	

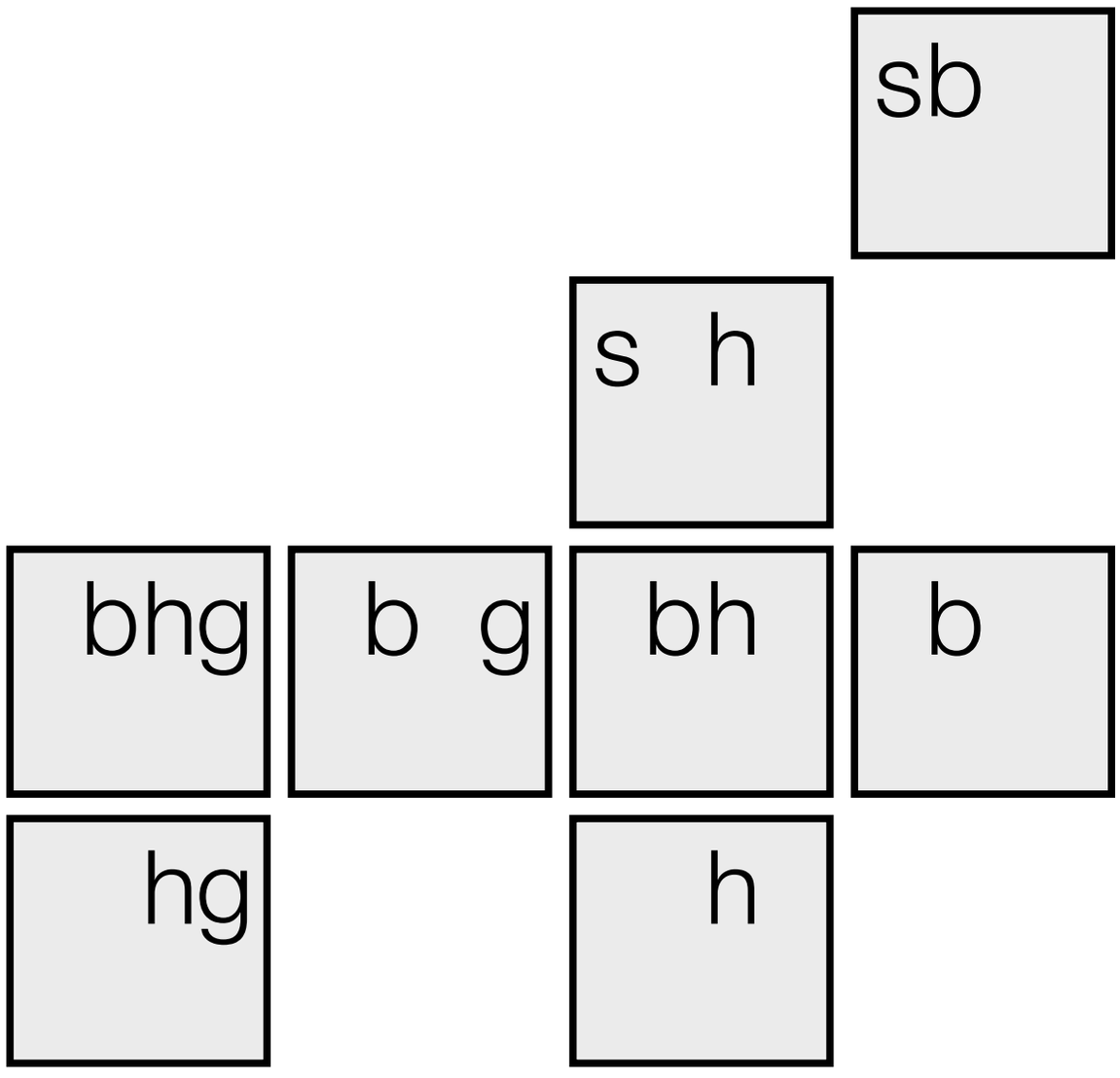
[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas



[Fierens et al, PP 12; Shterionov et al]

cProbLog: constraints on possible worlds

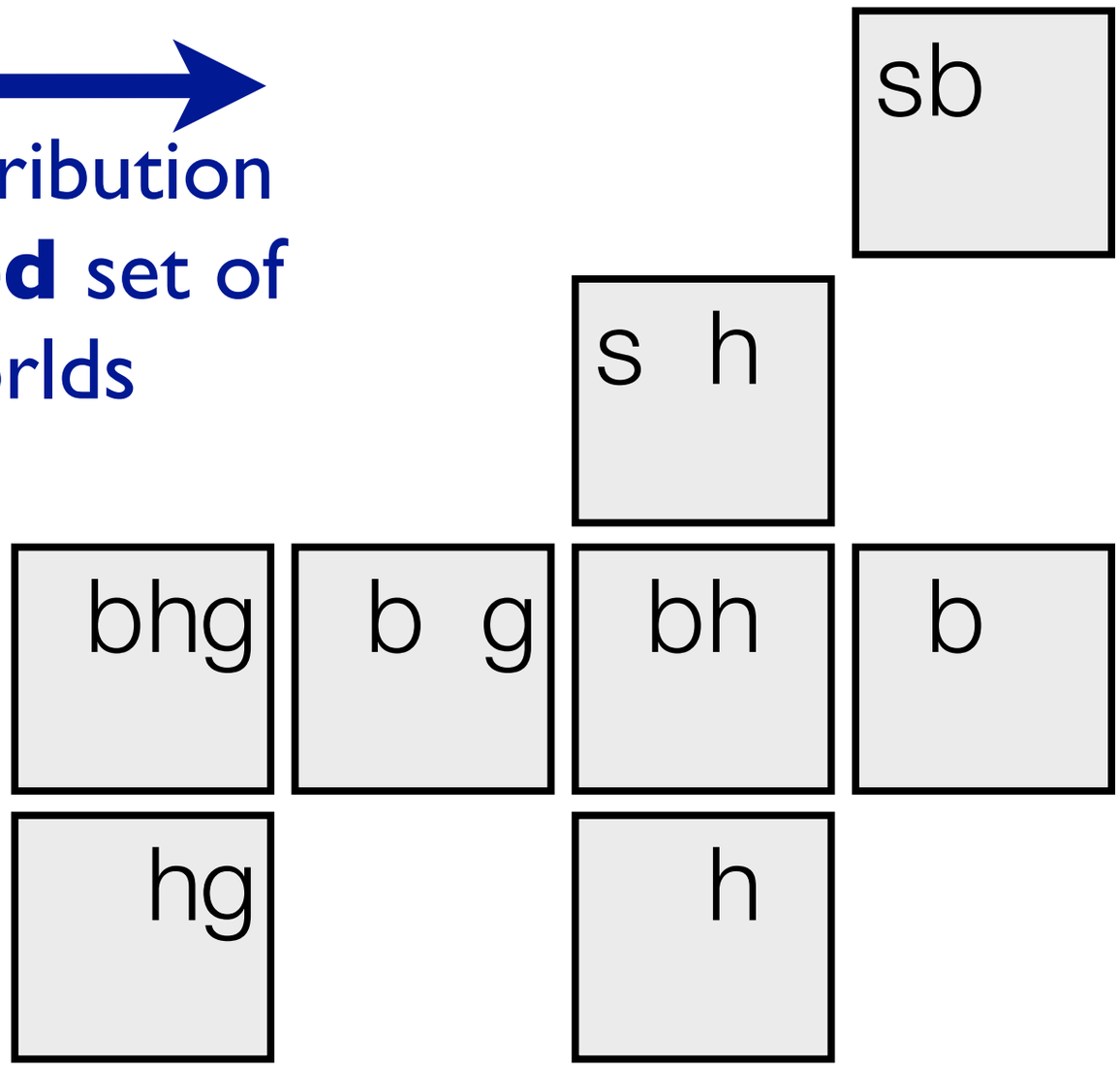
```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
```

```
P::pack(Item) :-
  weight(Item,Weight),
  P is 1.0/Weight.

excess(Limit) :- ...
```

```
not excess(10).
pack(helmet) v pack(boots).
```

constraints as first-order logic formulas

normalized distribution over **restricted** set of possible worlds



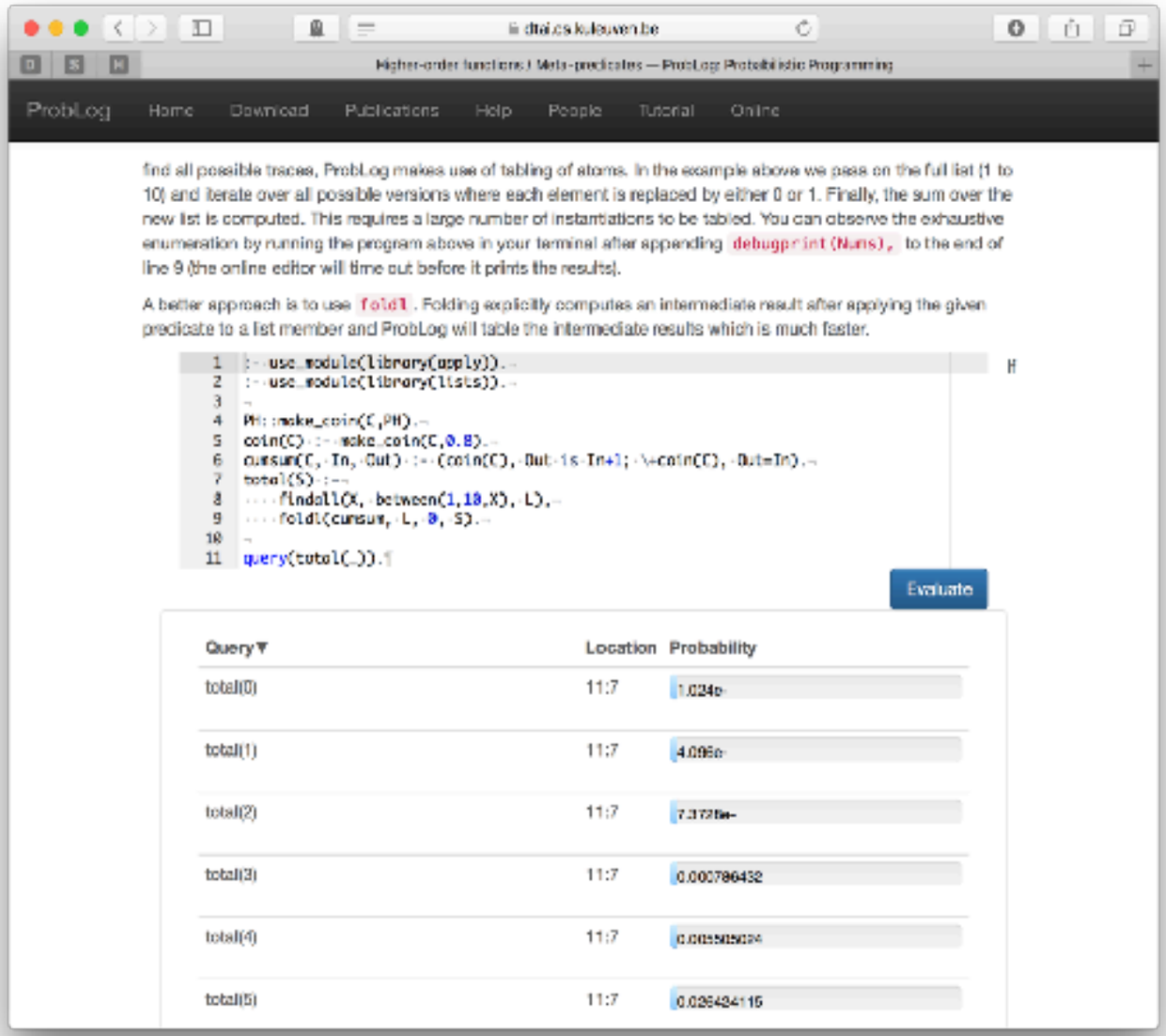
[Fierens et al, PP 12; Shterionov et al]

Summary

Getting started

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

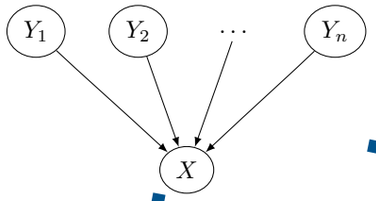
- Interactive tutorial
- Online interface for inference and parameter estimation
- Offline version for download (or `pip install problog`)



Assignment

Write your own state-of-the-art inference engine based on recent advances in probabilistic graphical models

Bayesian net



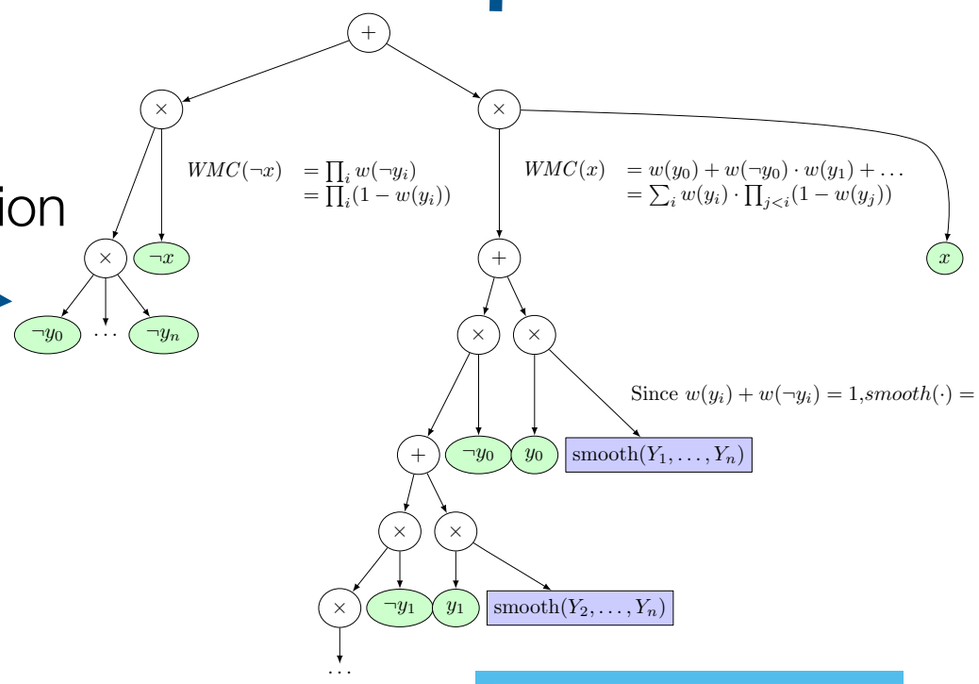
logic encoding

```
x ⇔ y(1) v y(2) v … v y(n)
y(1)
```

CNF

```
x v ¬y(1)
x v ¬y(2)
…
x v ¬y(n)
¬x v y(1) v y(2) v … v y(n)
y(1)
```

Knowledge Compilation



Model Counting

Pr(x | y(1))

ProbLog program

```
0.3::x :- y(1).
0.4::x :- y(2).
…
0.6::x :- y(n).

evidence(y(1)).
query(x).
```

KU LEUVEN

Next class: 7 December

Topics:

? Interactive session about assignment
? More on knowledge compilation
? Probabilistic Databases
? Lifted inference
? Relation to Church
? …

Acknowledgements

Maurice Bruynooghe
Bart Demoen
Luc De Raedt
Anton Dries
Daan Fierens
Jason Filippou
Bernd Gutmann
Manfred Jaeger
Gerda Janssens
Kristian Kersting
Angelika Kimmig
Theofrastos Mantadelis
Wannes Meert
Bogdan Moldovan
Siegfried Nijssen
Davide Nitti
Joris Renkens
Kate Revoredo
Ricardo Rocha
Vitor Santos Costa
Dimitar Shterionov
Ingo Thon
Hannu Toivonen
Guy Van den Broeck
Jonas Vlasselaer

KU LEUVEN