ProbLog: a probabilistic programming language for data analysis

CATCH meeting 04.10.13

Angelika Kimmig
angelika.kimmig@cs.kuleuven.be

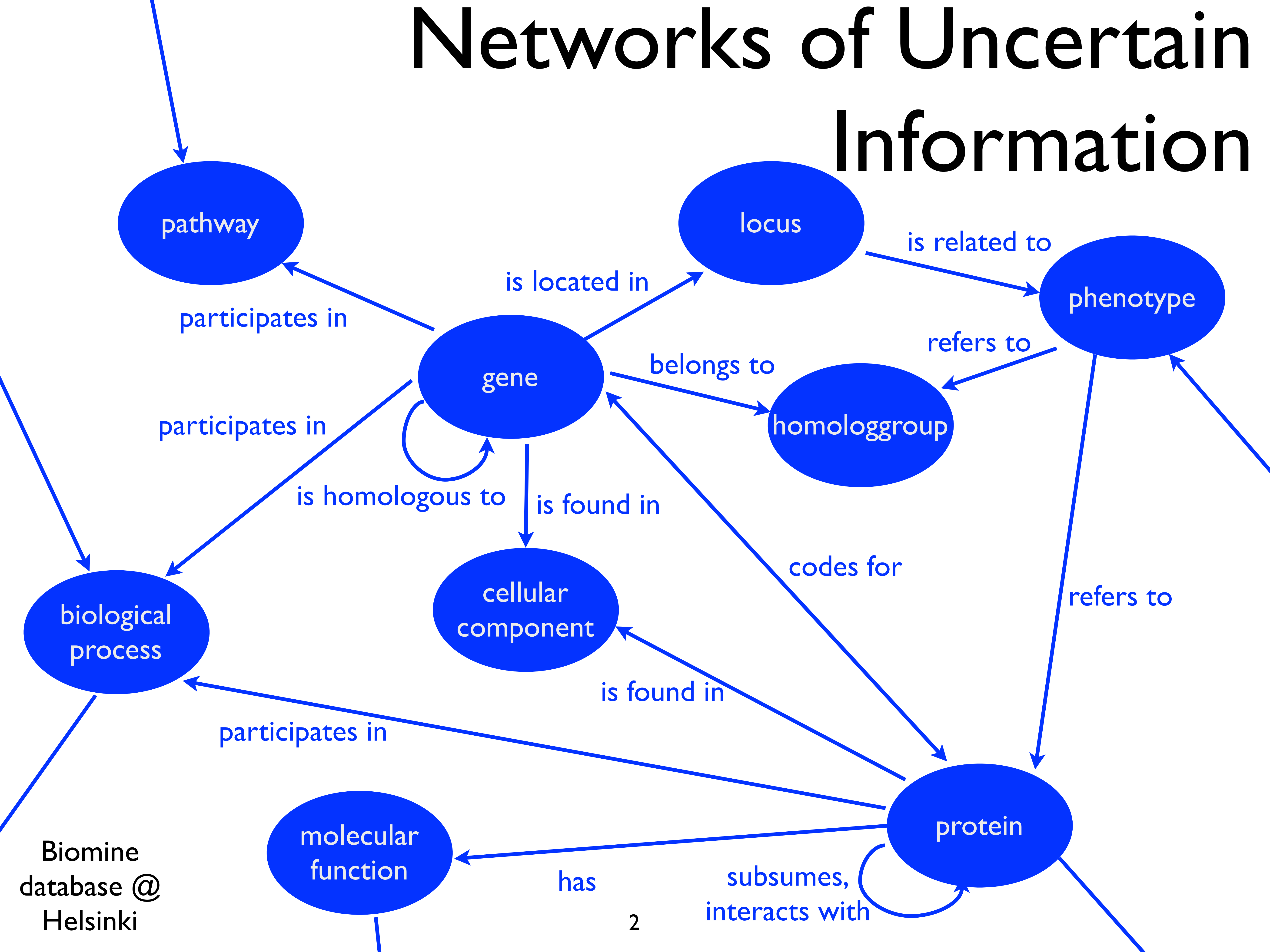
DTAI
MACHINE LEARNING

KU LEUVEN

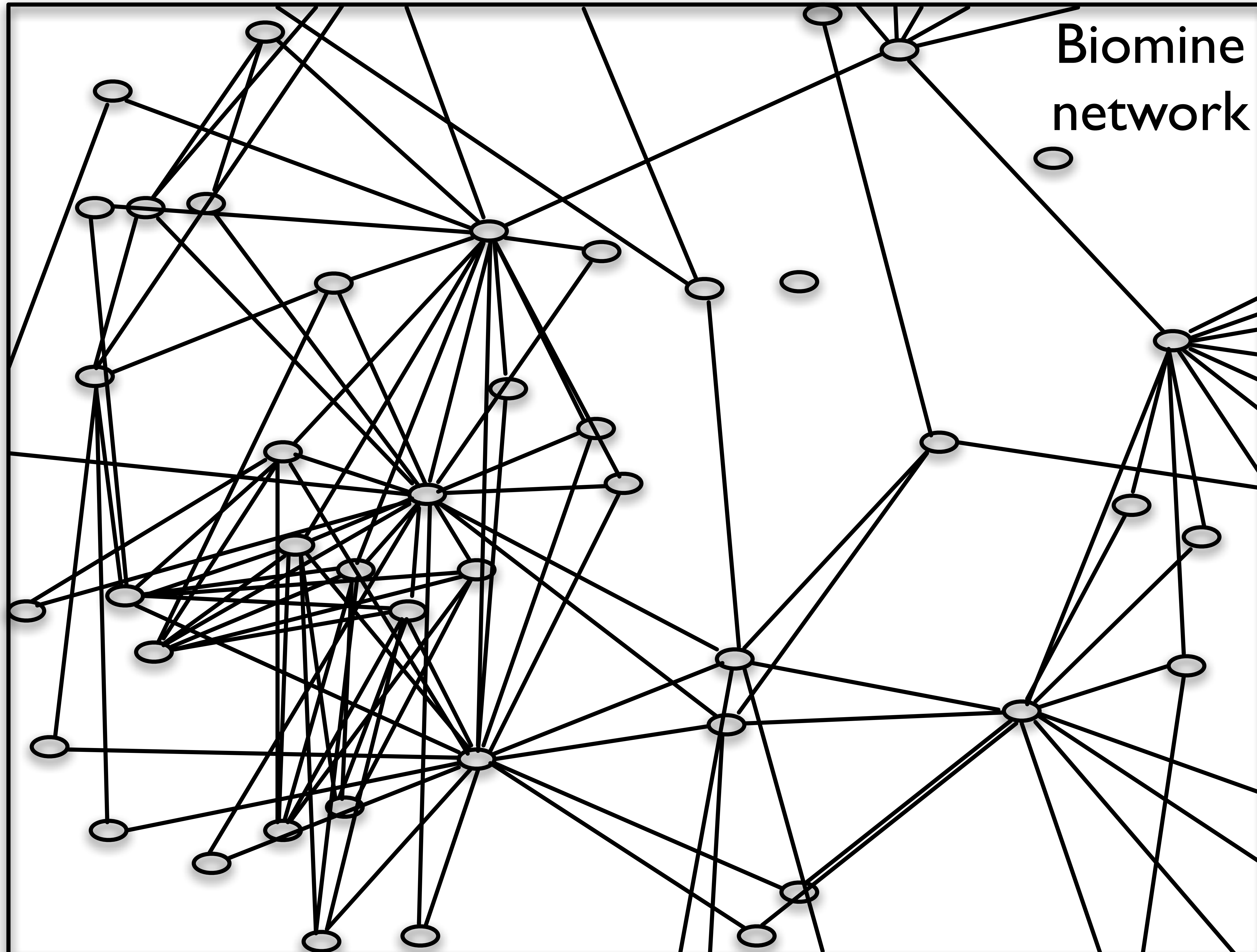
FWO
VLAANDEREN
Fonds Wetenschappelijk Onderzoek
Research Foundation – Flanders

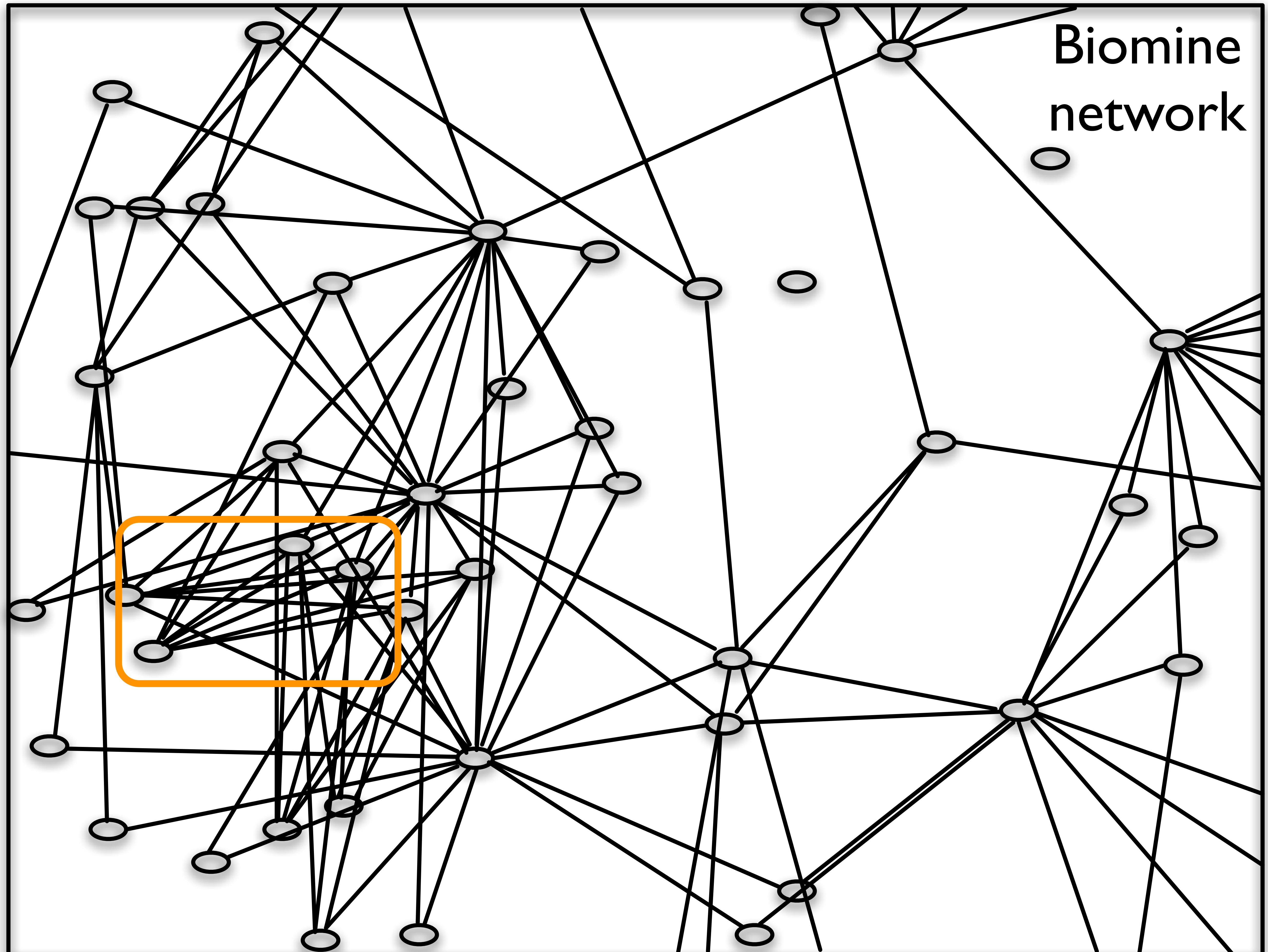
http://dtai.cs.kuleuven.be/problog

Networks of Uncertain Information

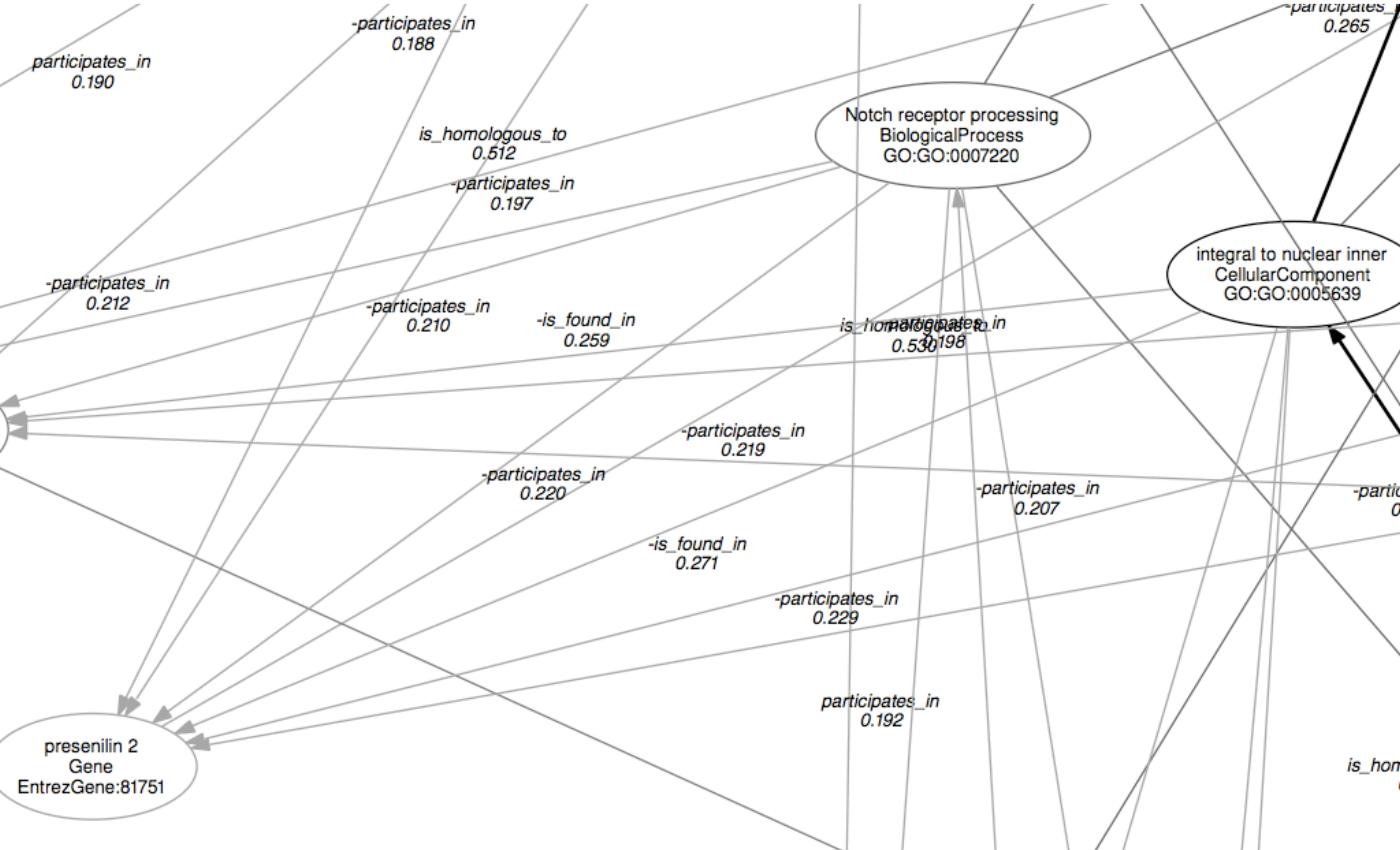


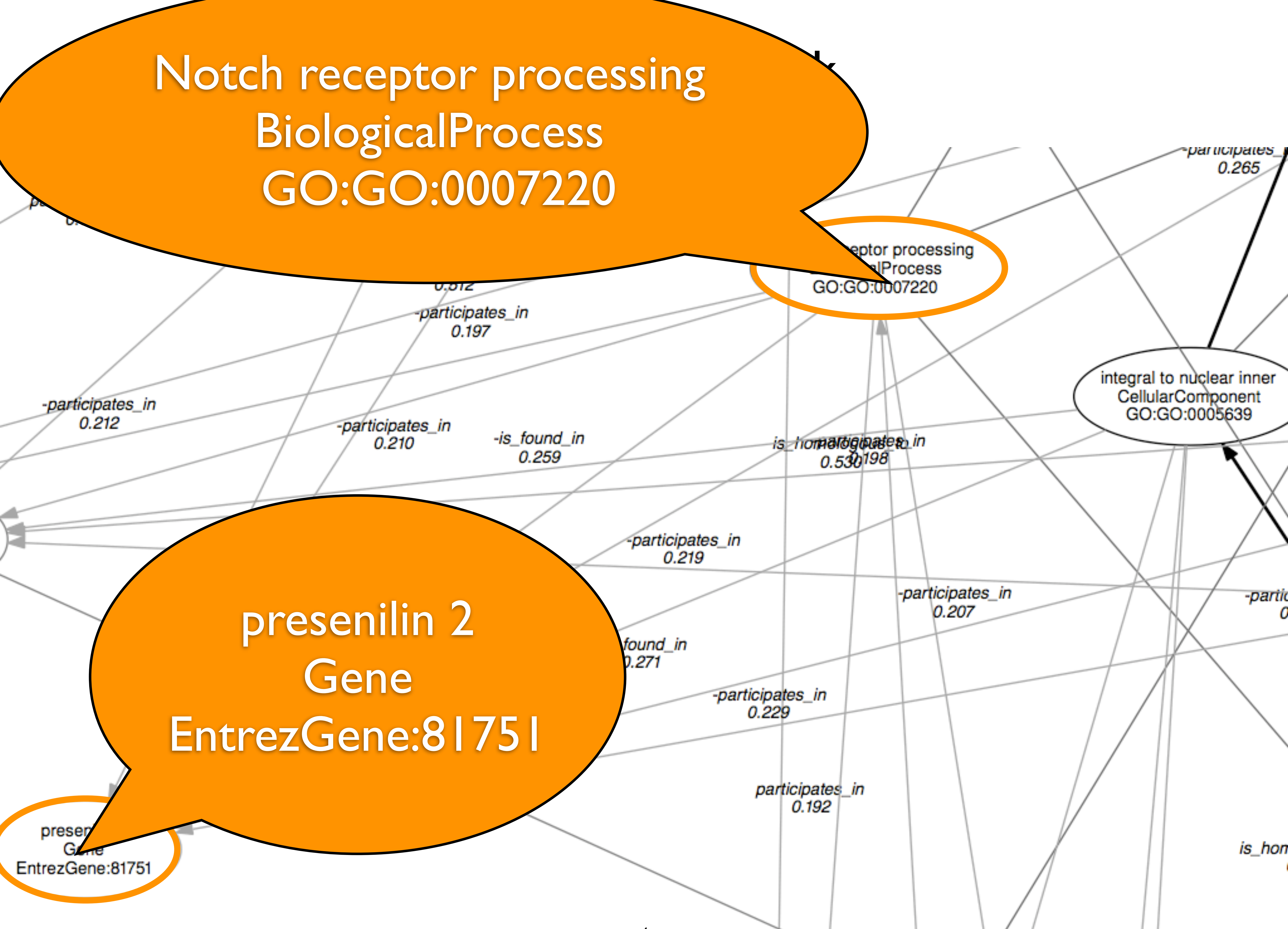
Biomine database @ Helsinki


Biomine
network

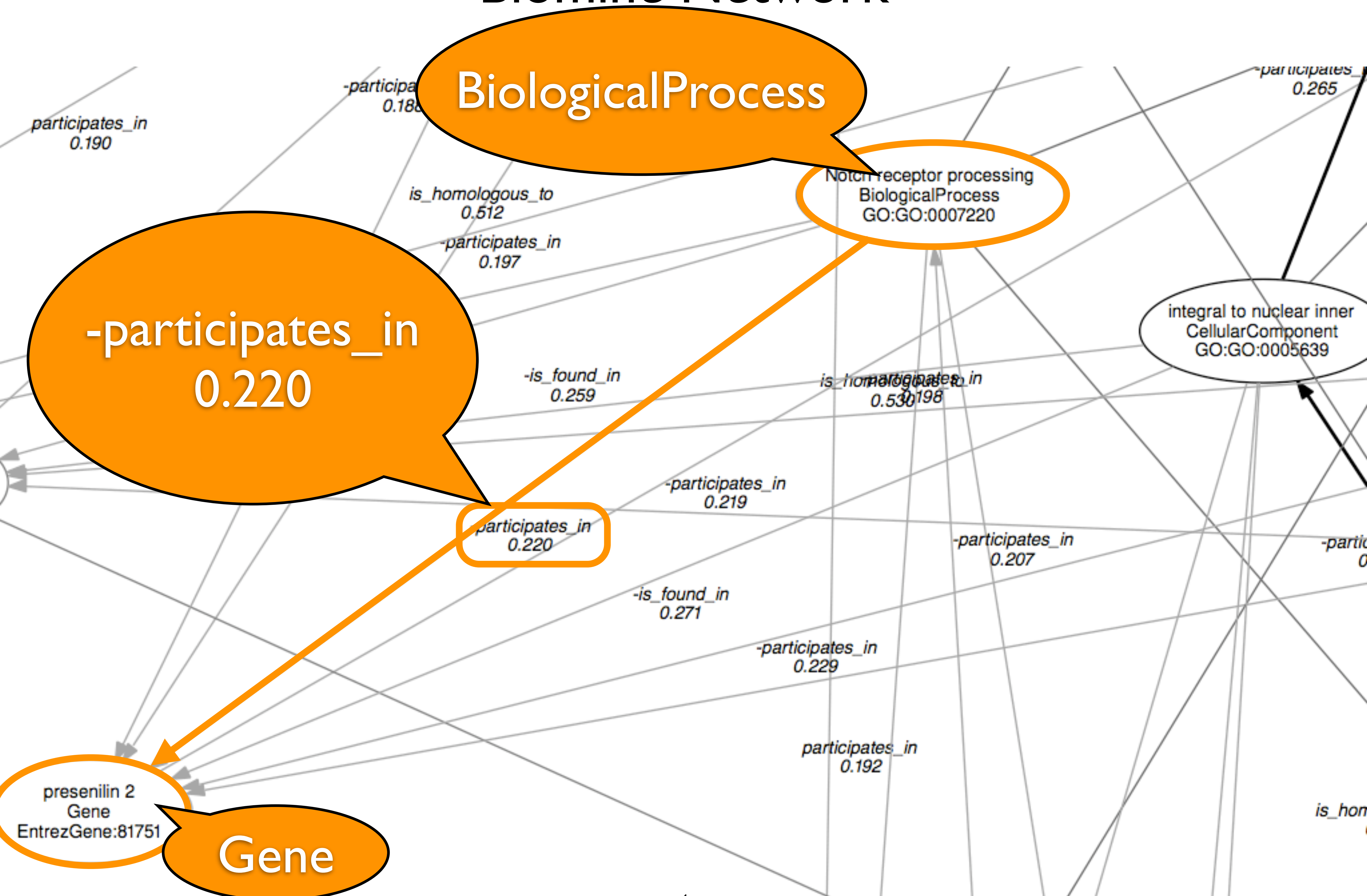

Biomine
network

Biomine Network

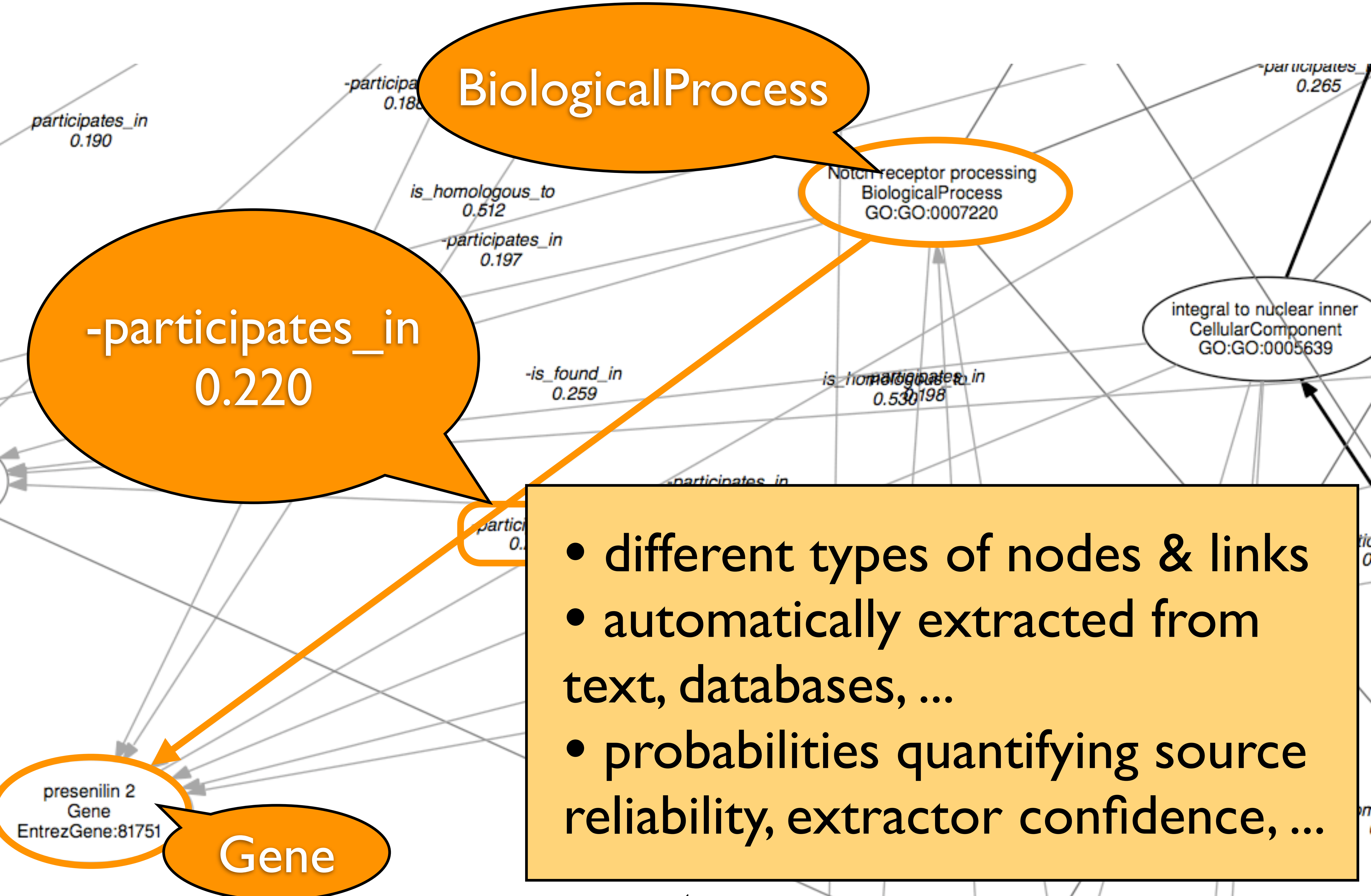



Notch receptor processing
BiologicalProcess
GO:GO:0007220
presenilin 2
Gene
EntrezGene:81751
GO:GO:0007220
integral to nuclear inner
CellularComponent
GO:GO:0005639
-participates_in
0.197
-participates_in
0.212
-participates_in
0.210
-is_found_in
0.259
-participates_in
0.219
-participates_in
0.207
-participates_in
0.229
participates_in
0.192
0.265
EntrezGene:81751

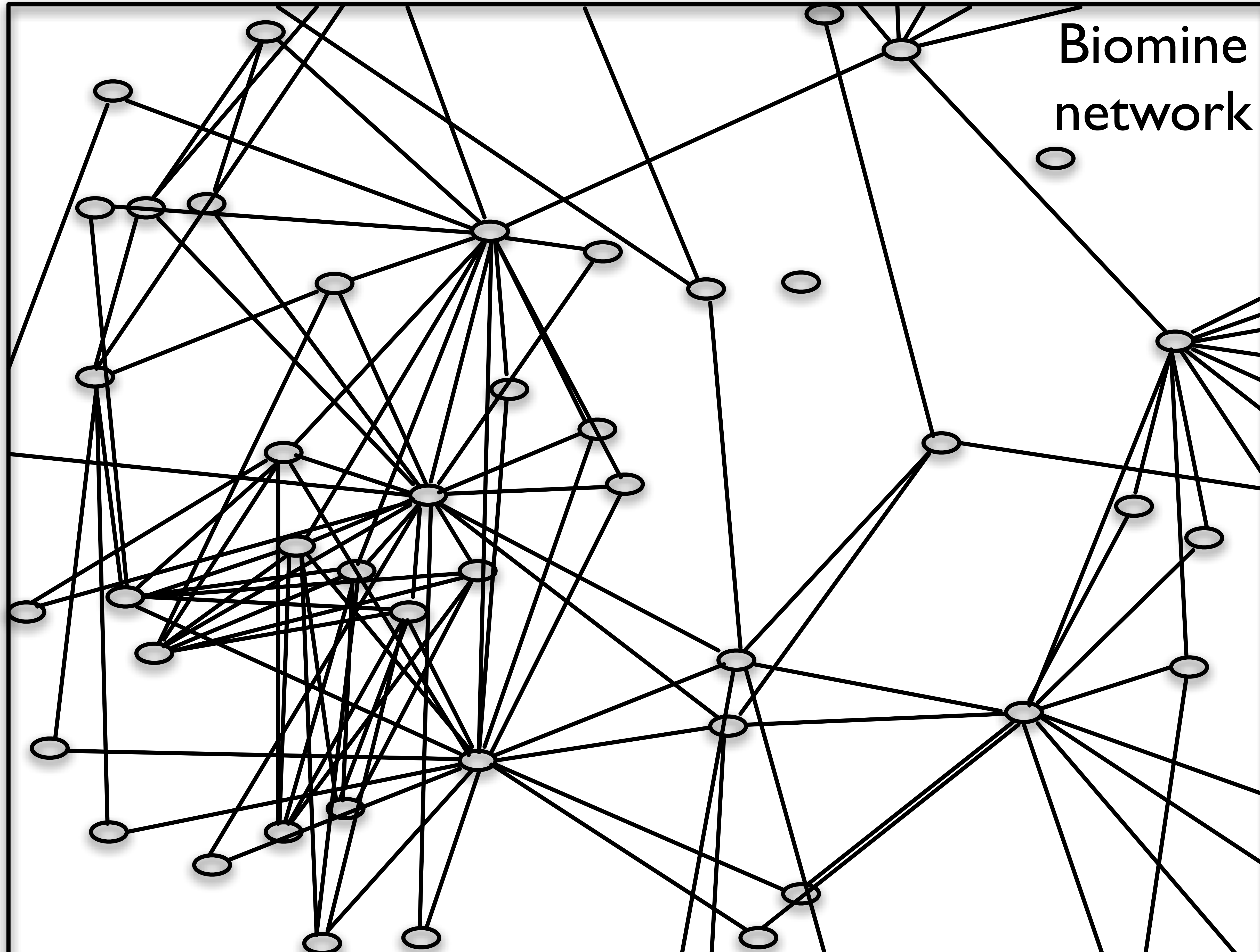
Biomine Network

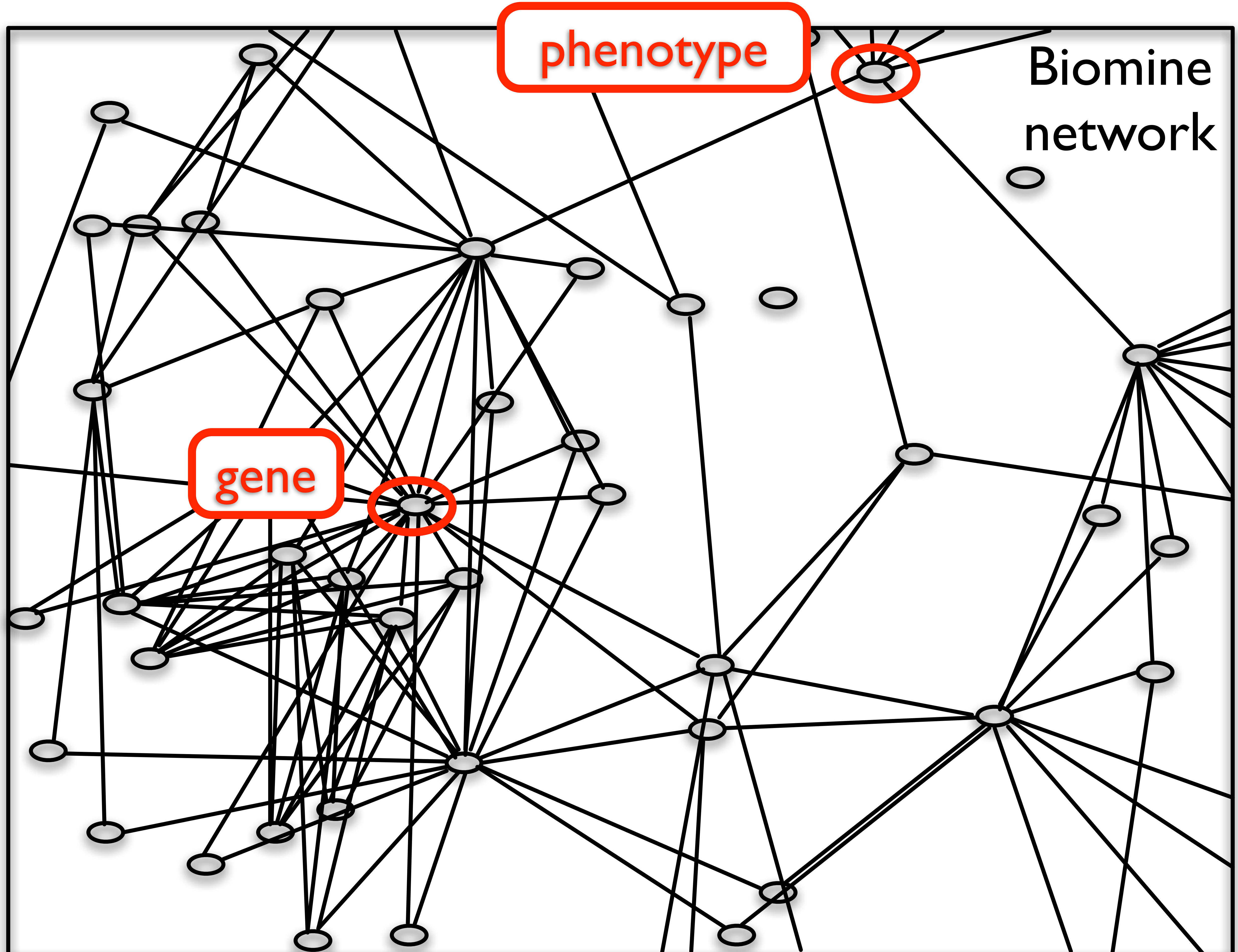


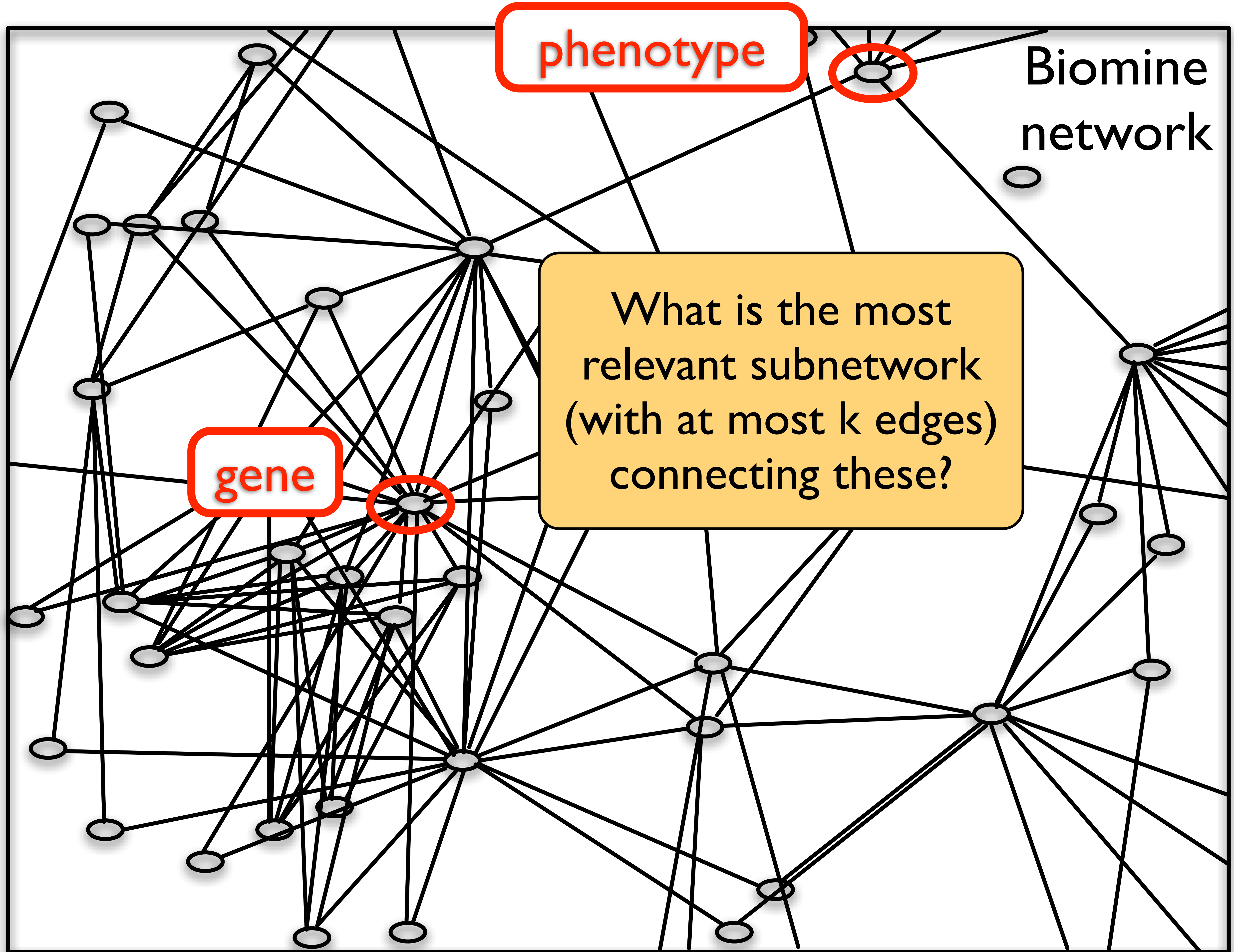
Biomine Network

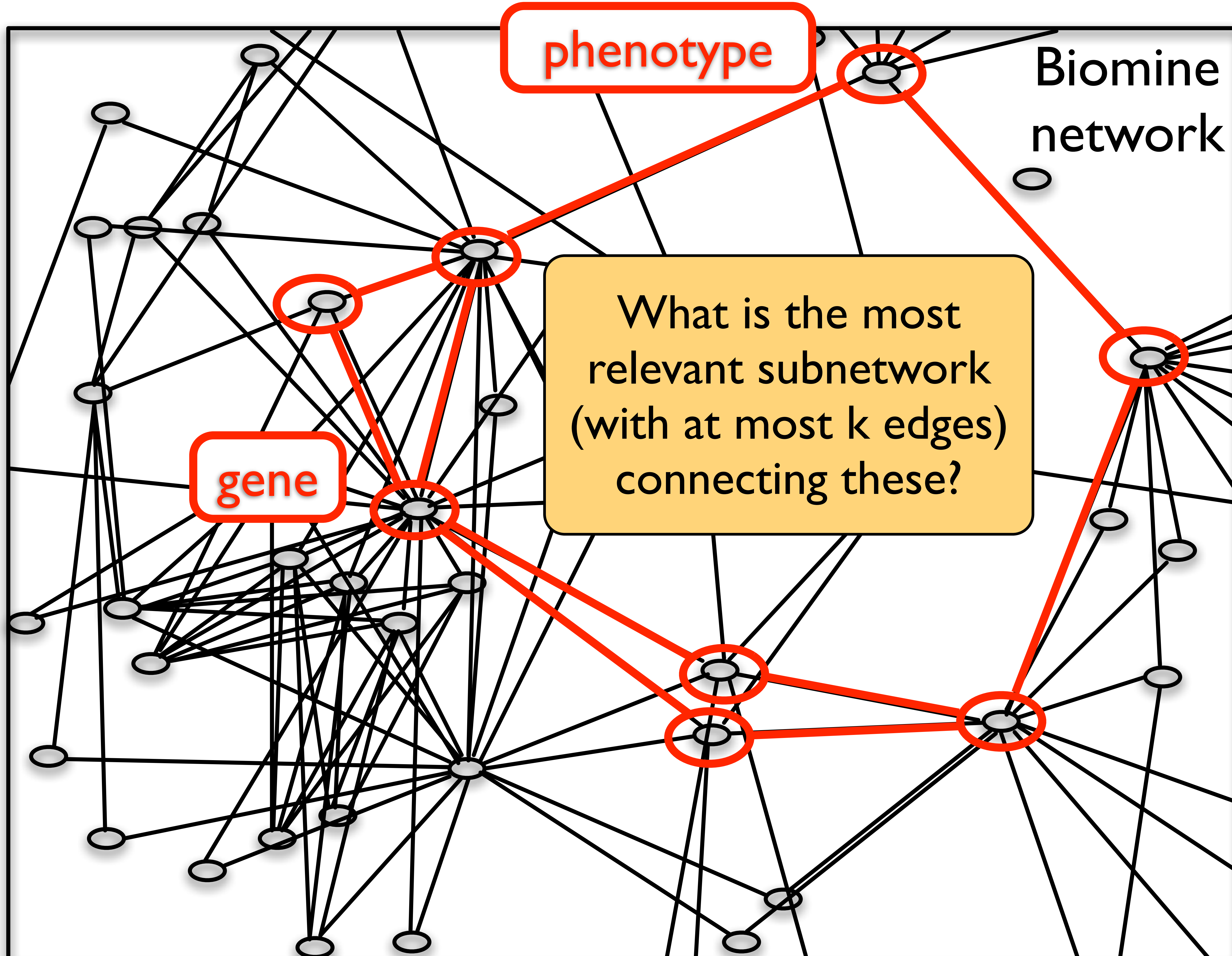


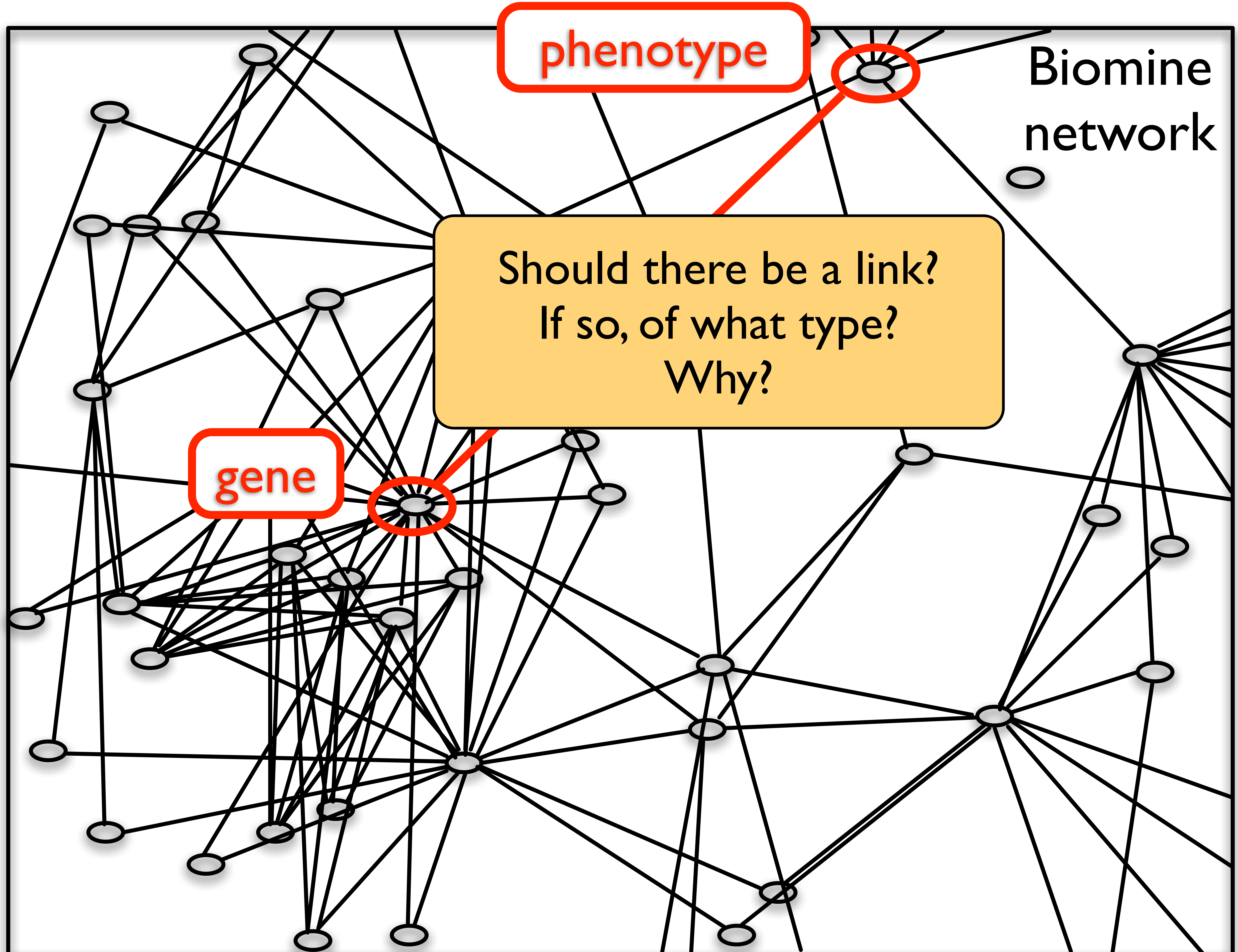
- different types of nodes & links
- automatically extracted from text, databases, ...
- probabilities quantifying source reliability, extractor confidence, ...


Biomine
network

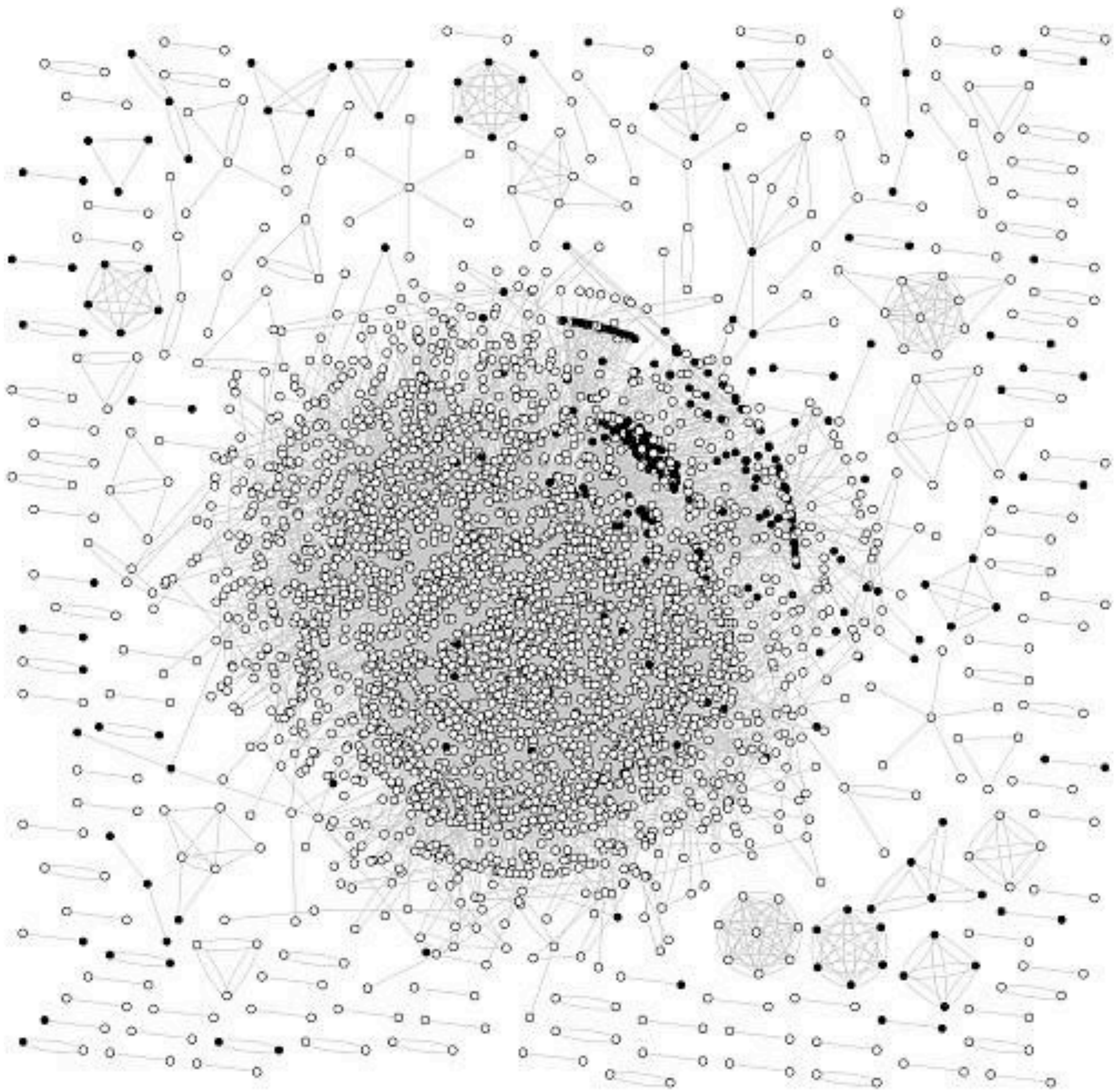

phenotype
Biomine
network
gene


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


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


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

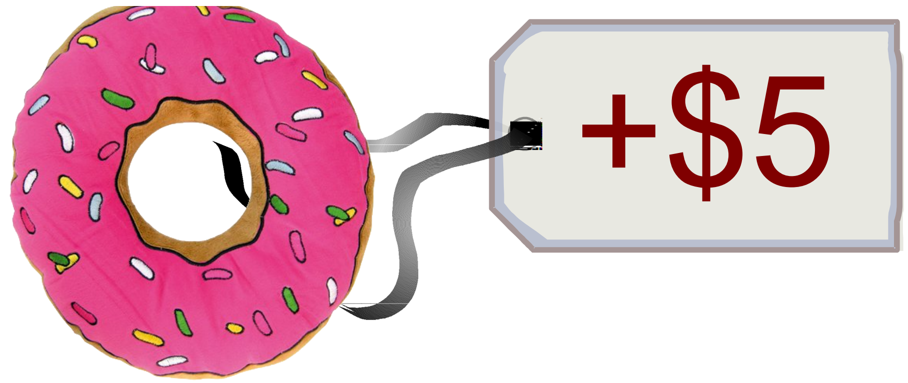
Node Classification



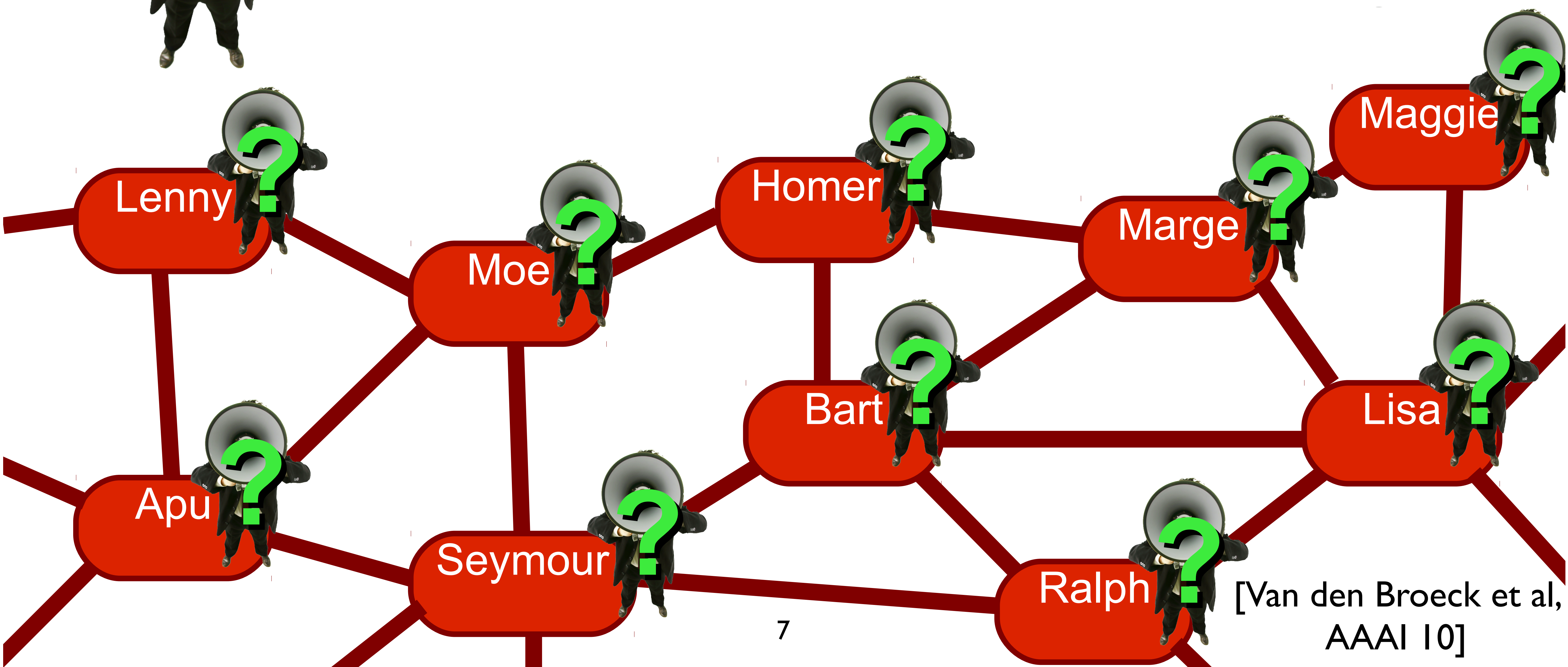
Can we predict the type of a node given information on its neighbors?

e.g., the type of a webpage given its links and the words on the page?

Viral Marketing

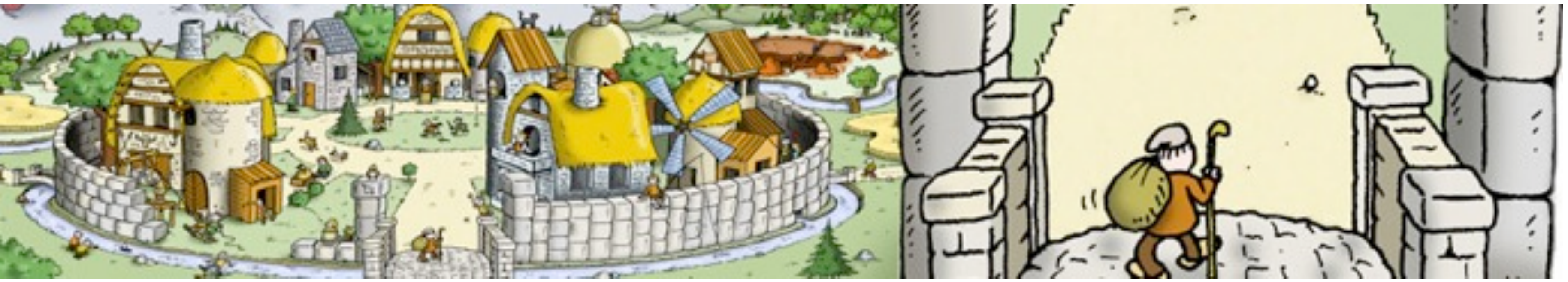


Which advertising strategy maximizes expected profit?



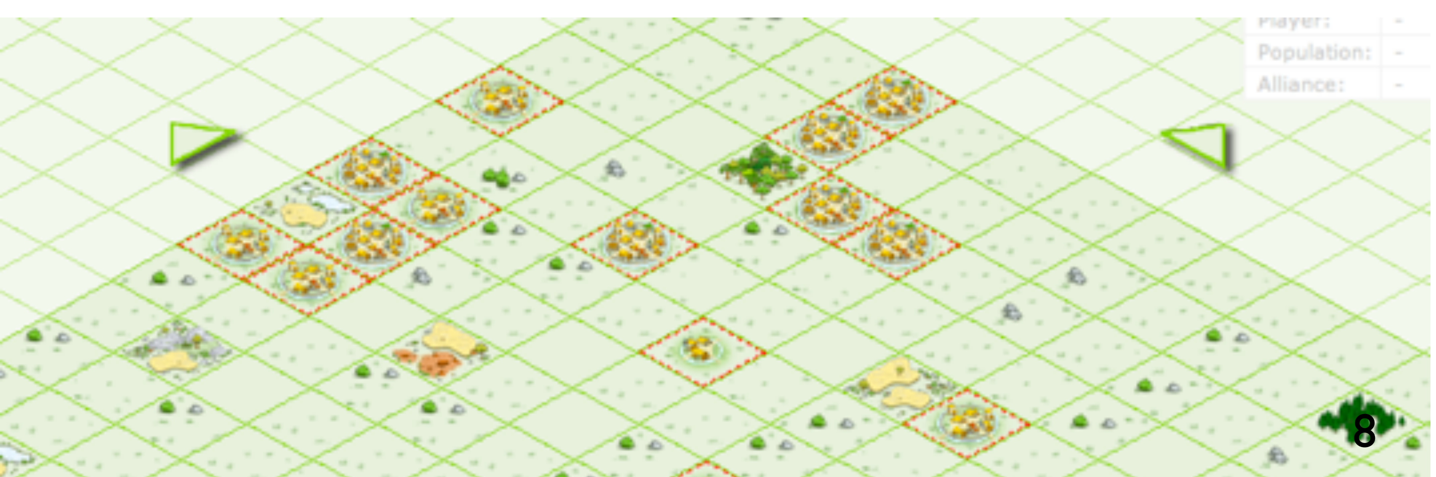
[Van den Broeck et al, AAAI 10]

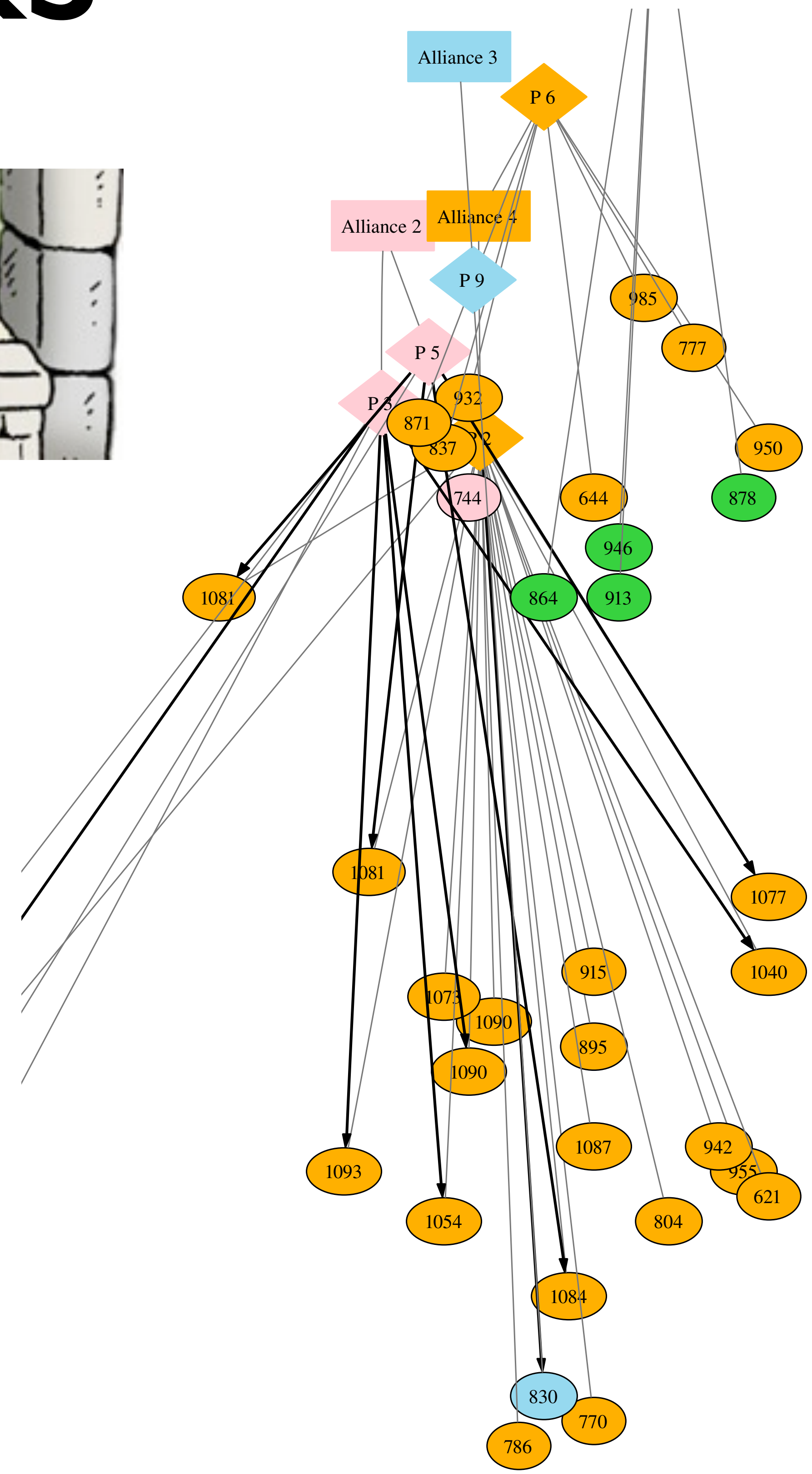
Dynamic networks



Travian: A massively multiplayer real-time strategy game

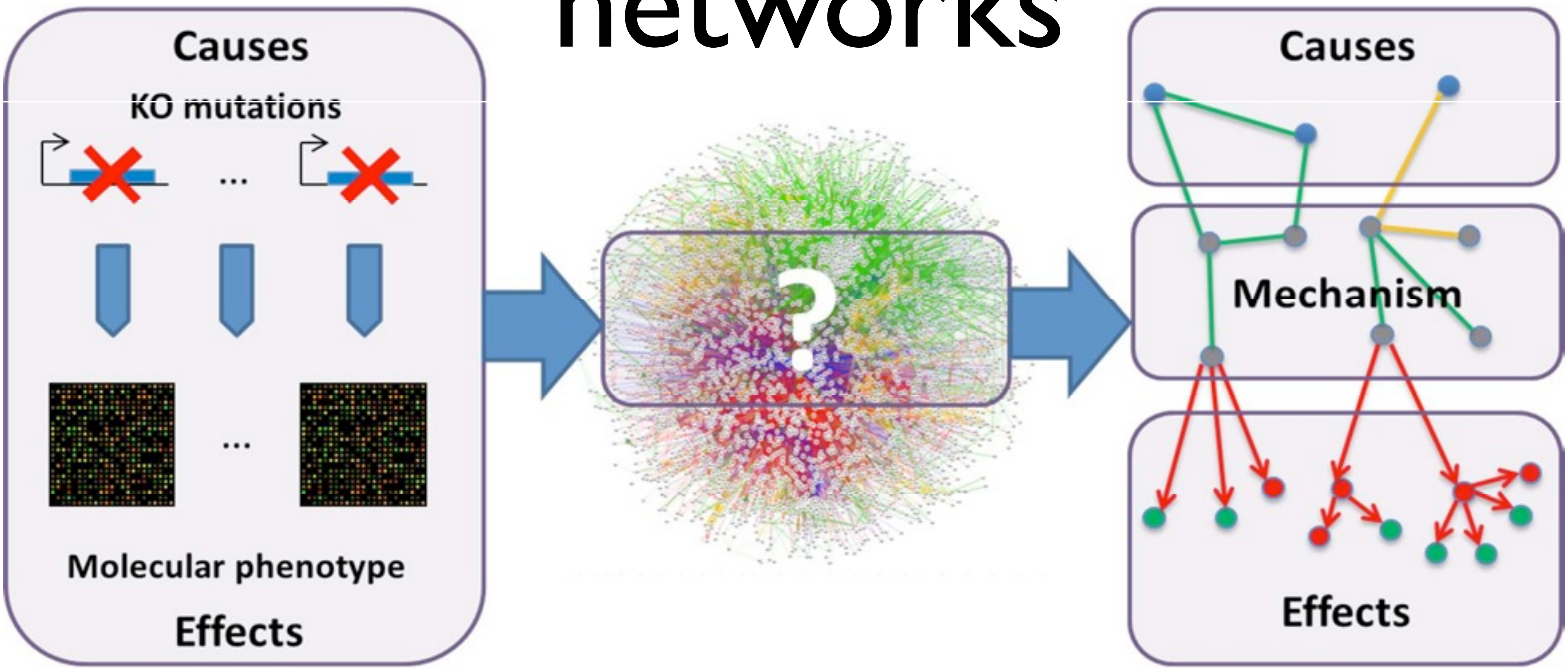
Can we build a model of this world ? Can we use it for playing better ?





[Thon et al, MLJ 11]

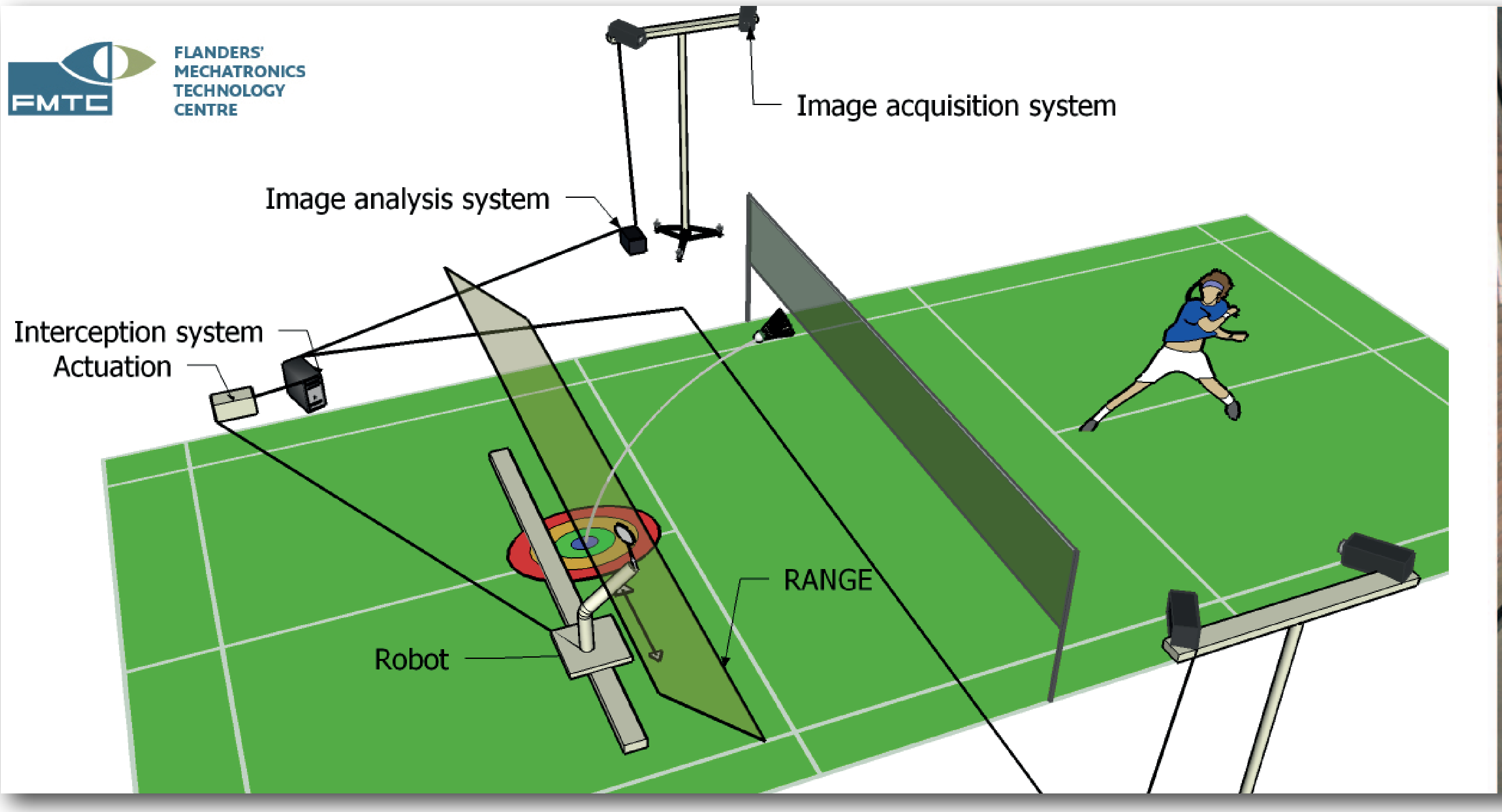
Molecular interaction networks

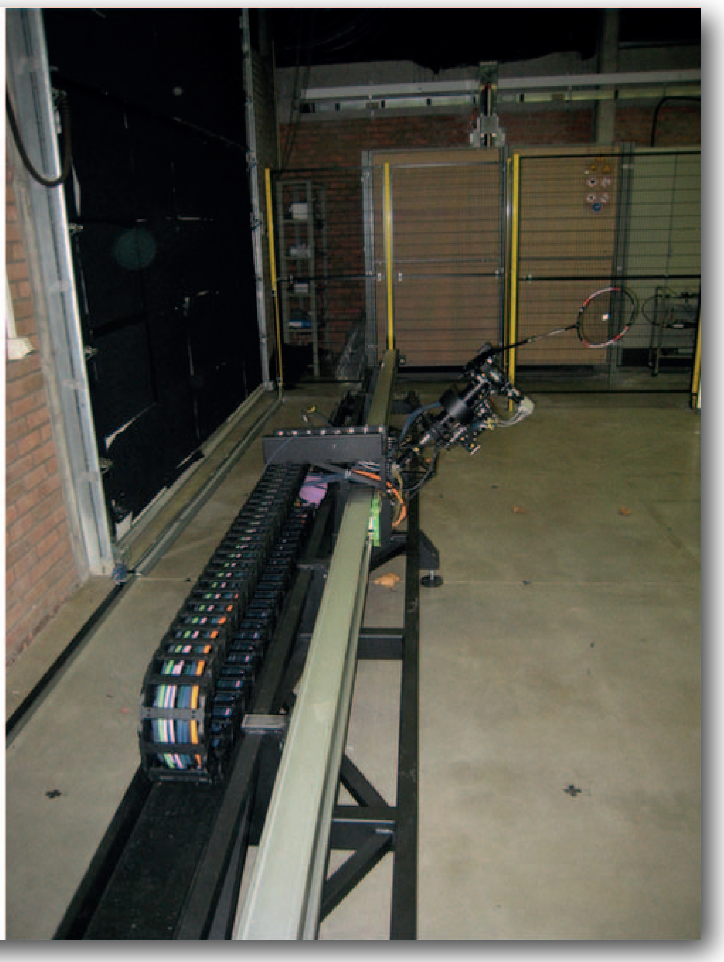


Can we find the mechanism connecting causes to effects?

[De Maeyer at al, Molecular Biosystems 13]

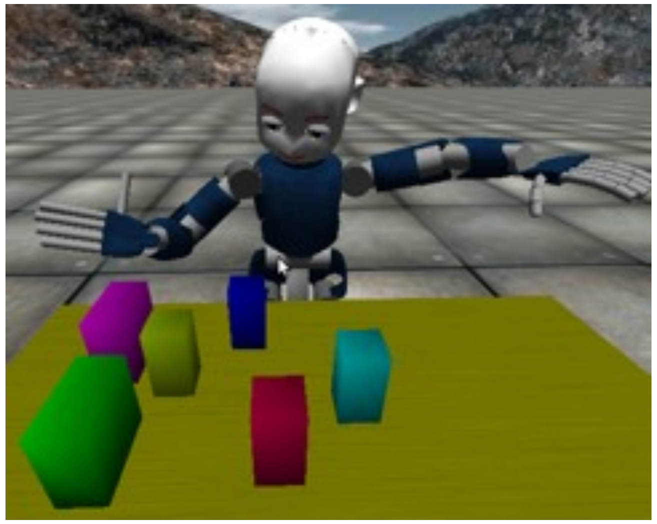
Diagnosing machine failures



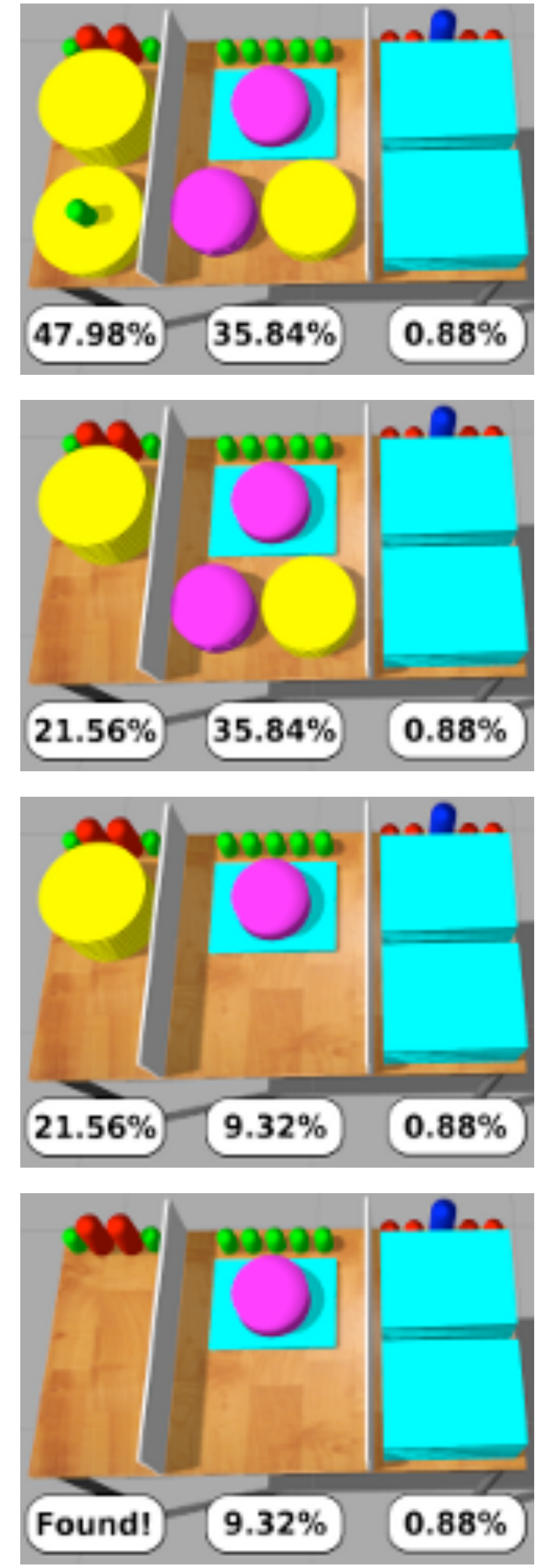


Can we build a model of the robot's working and use it to find causes of failures?

[Schramm, Meert and Driessens]



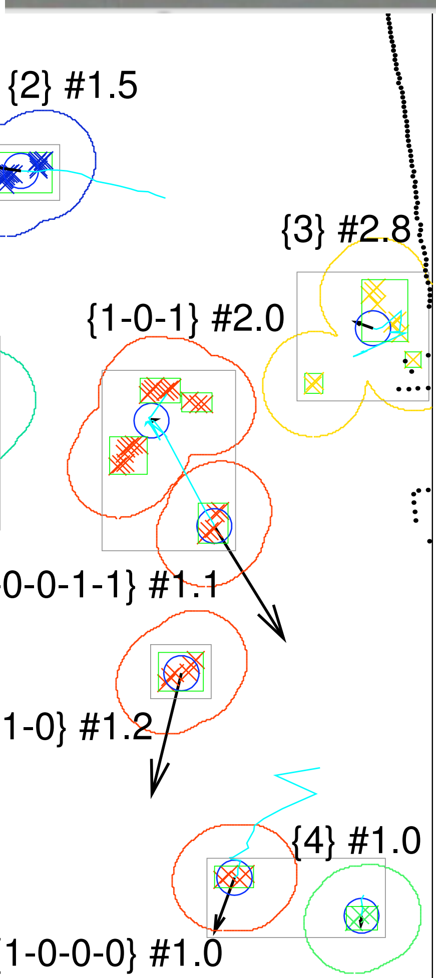
Robotics



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

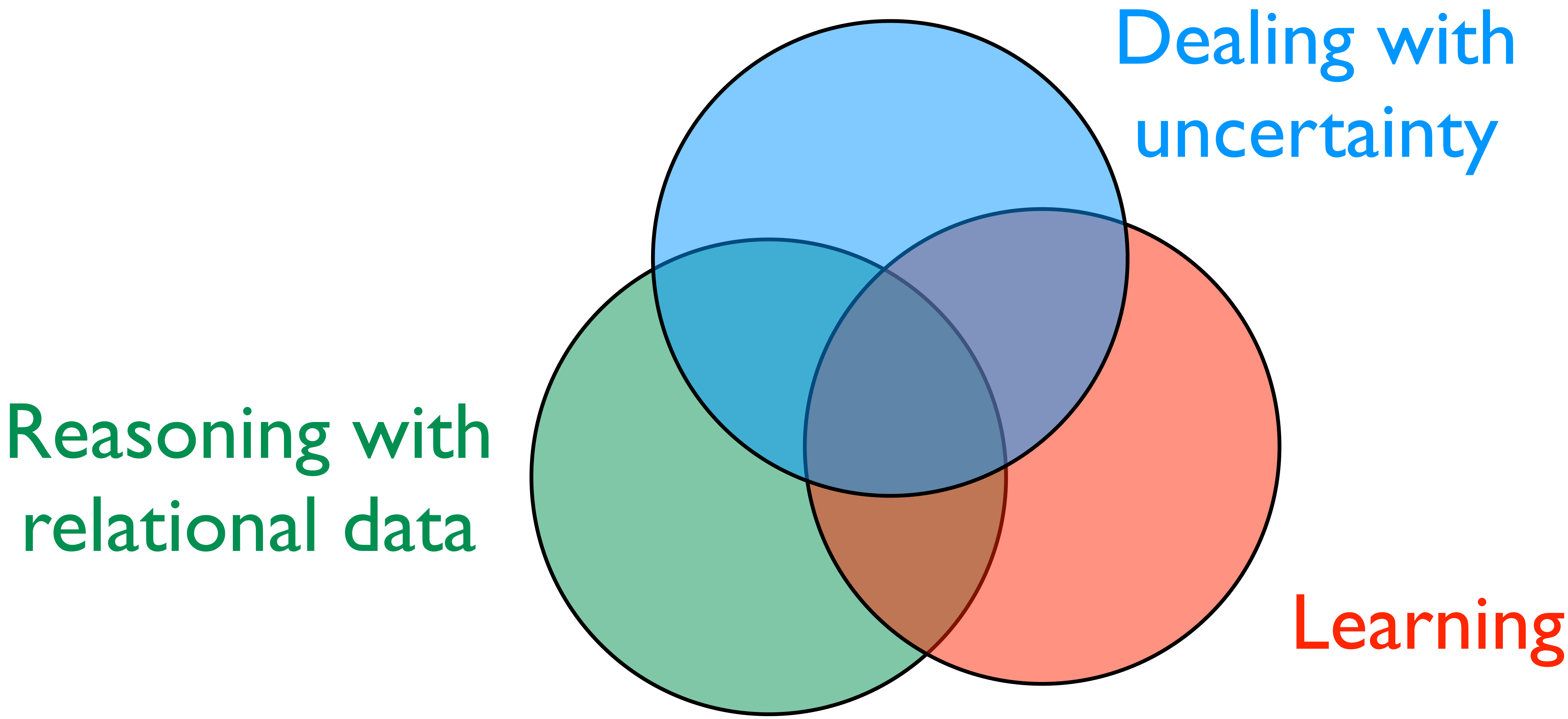
[Moldovan et al]

Analyzing Video Data



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

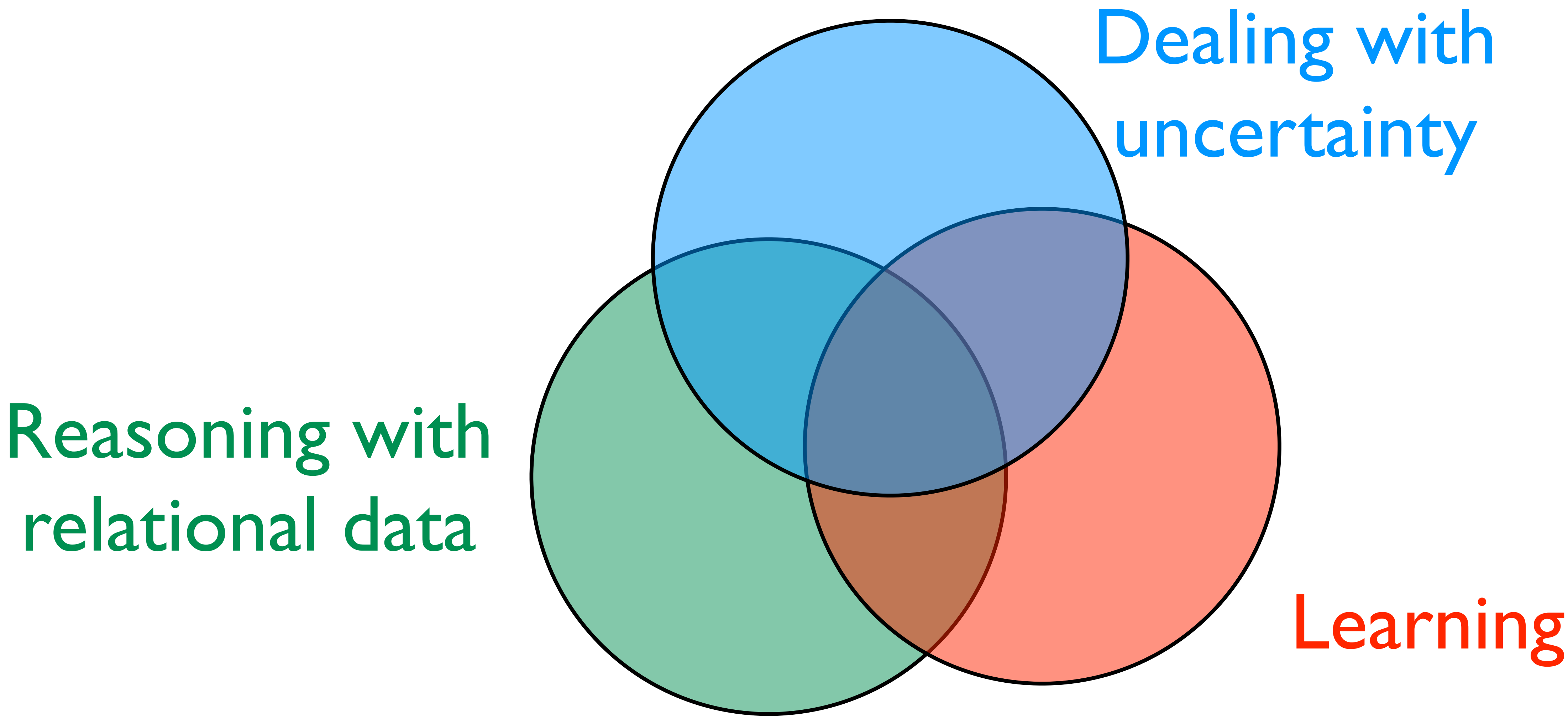
Common theme



Statistical relational learning, probabilistic logic learning, probabilistic programming, ...

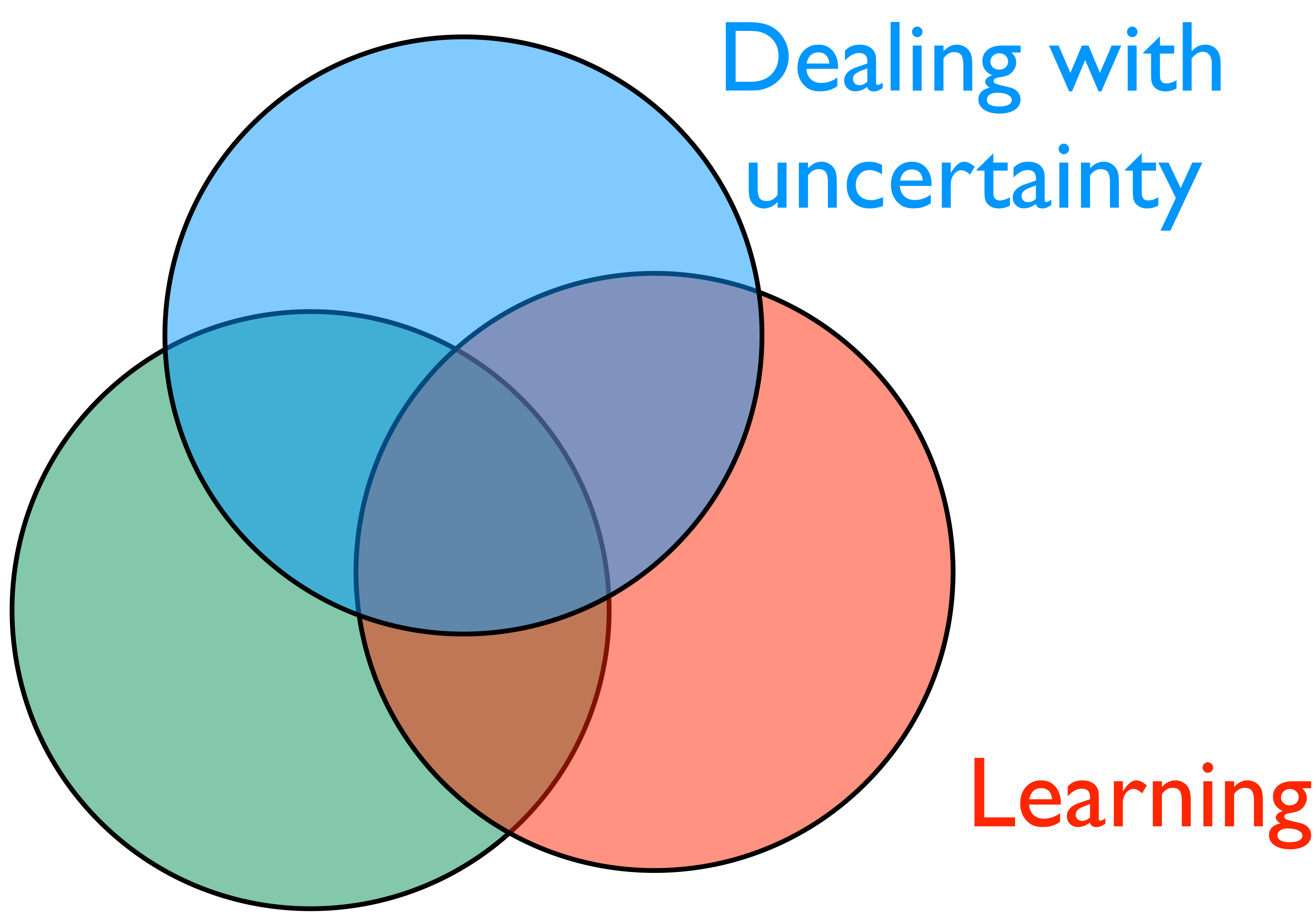
ProbLog

probabilistic Prolog



ProbLog

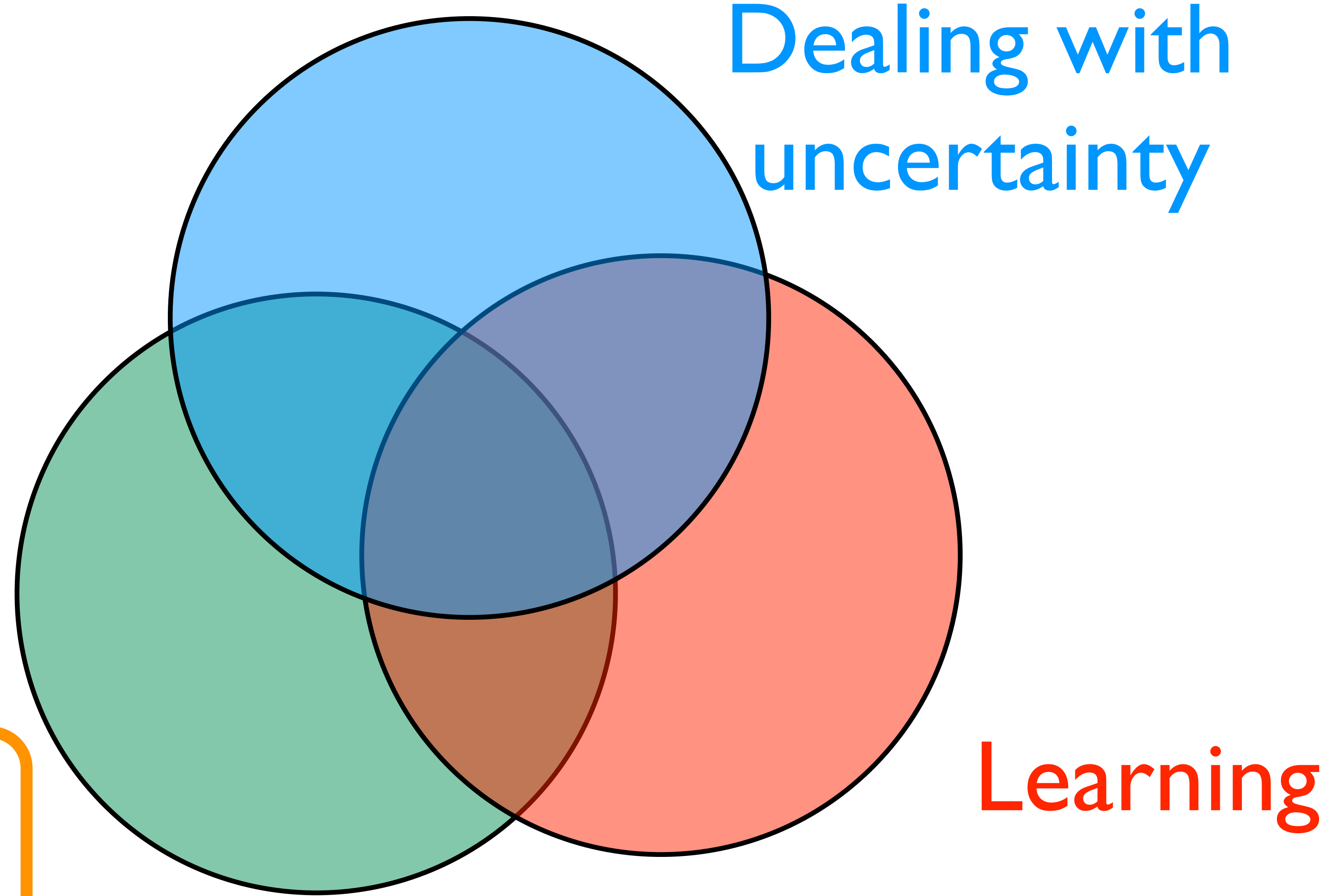
probabilistic Prolog




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

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

ProbLog

probabilistic Prolog



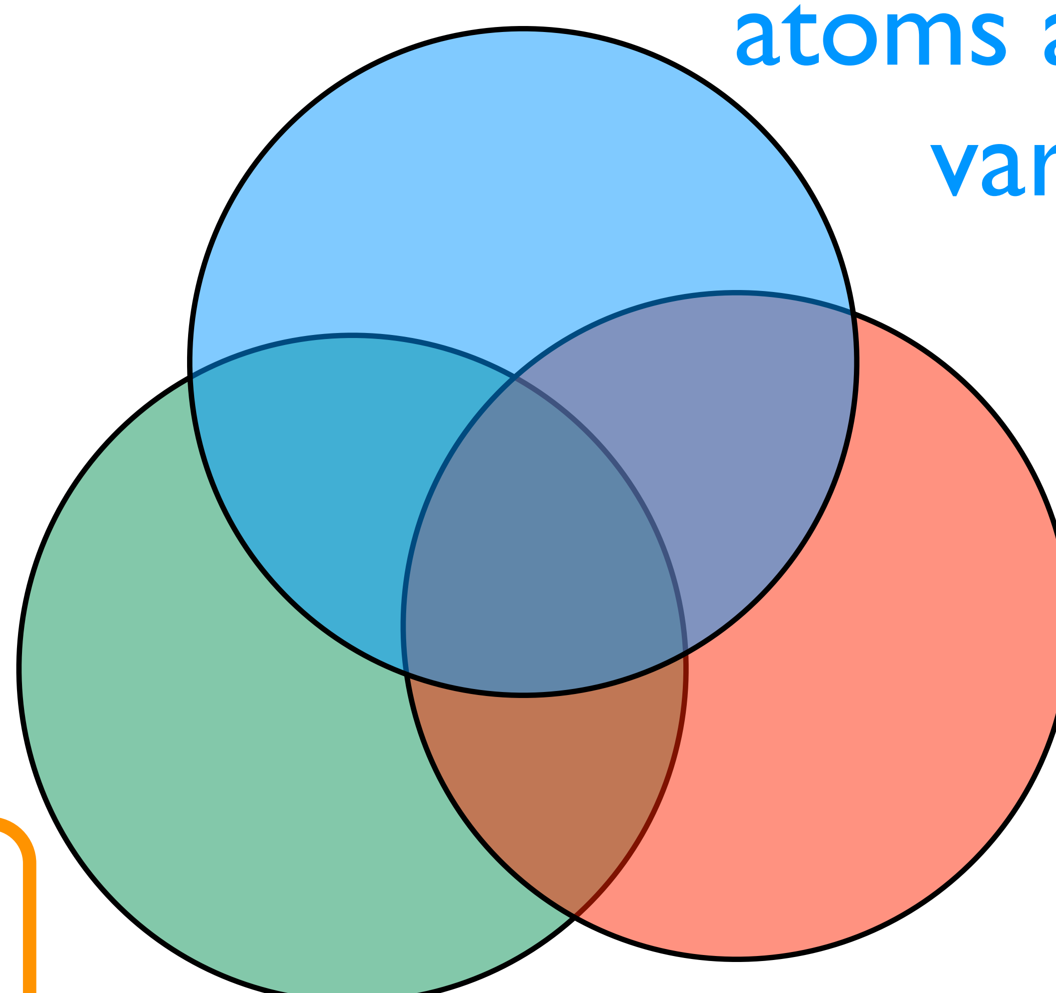

```
stress(ann).
influences(ann,bob).
influences(bob,carl).
```

```
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).
```




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

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

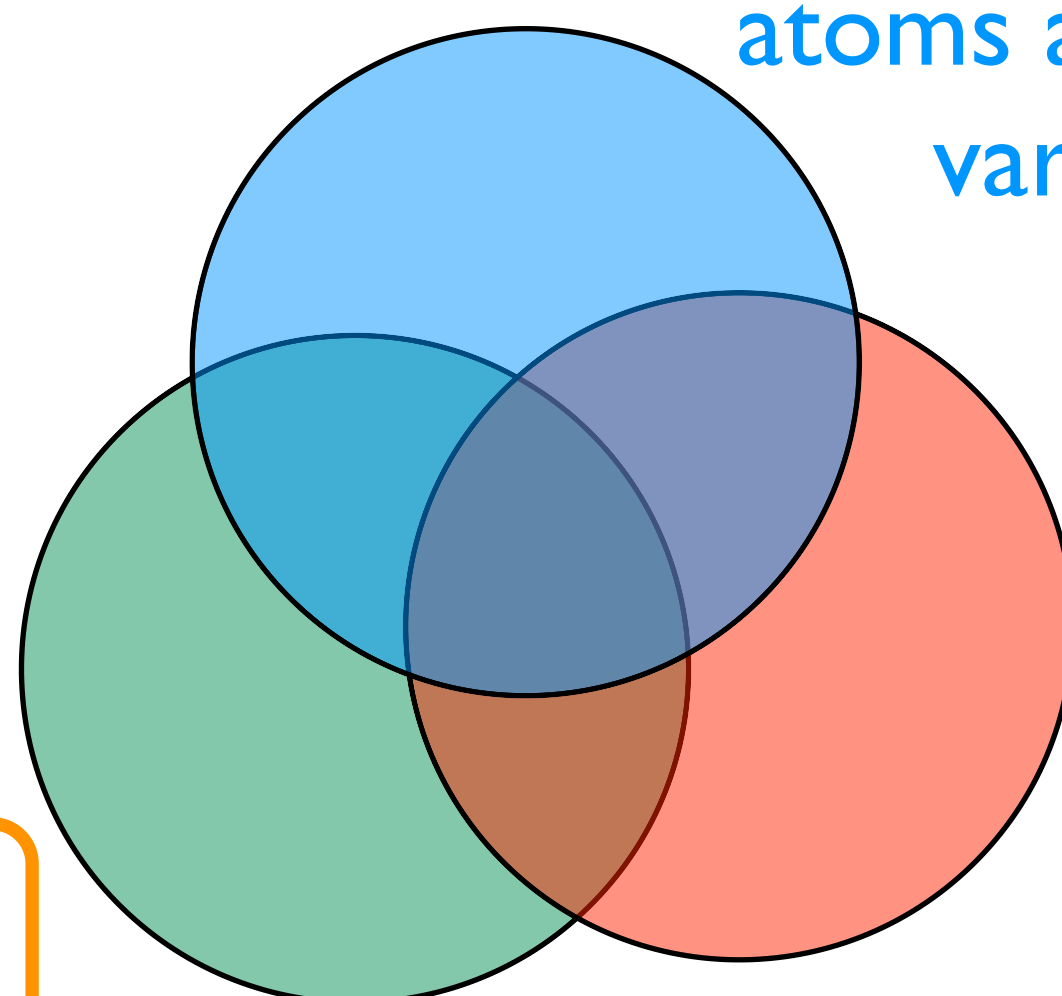
ProbLog

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

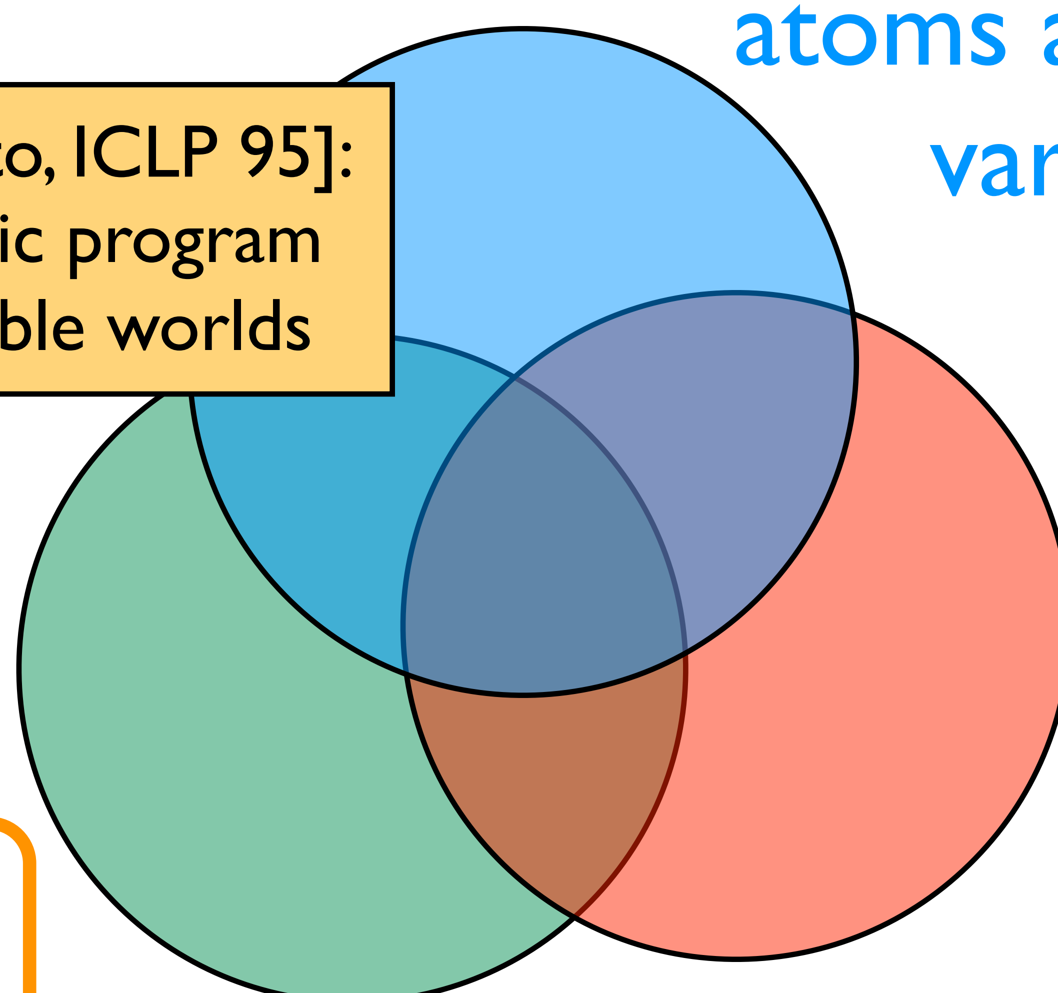
probabilistic Prolog

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

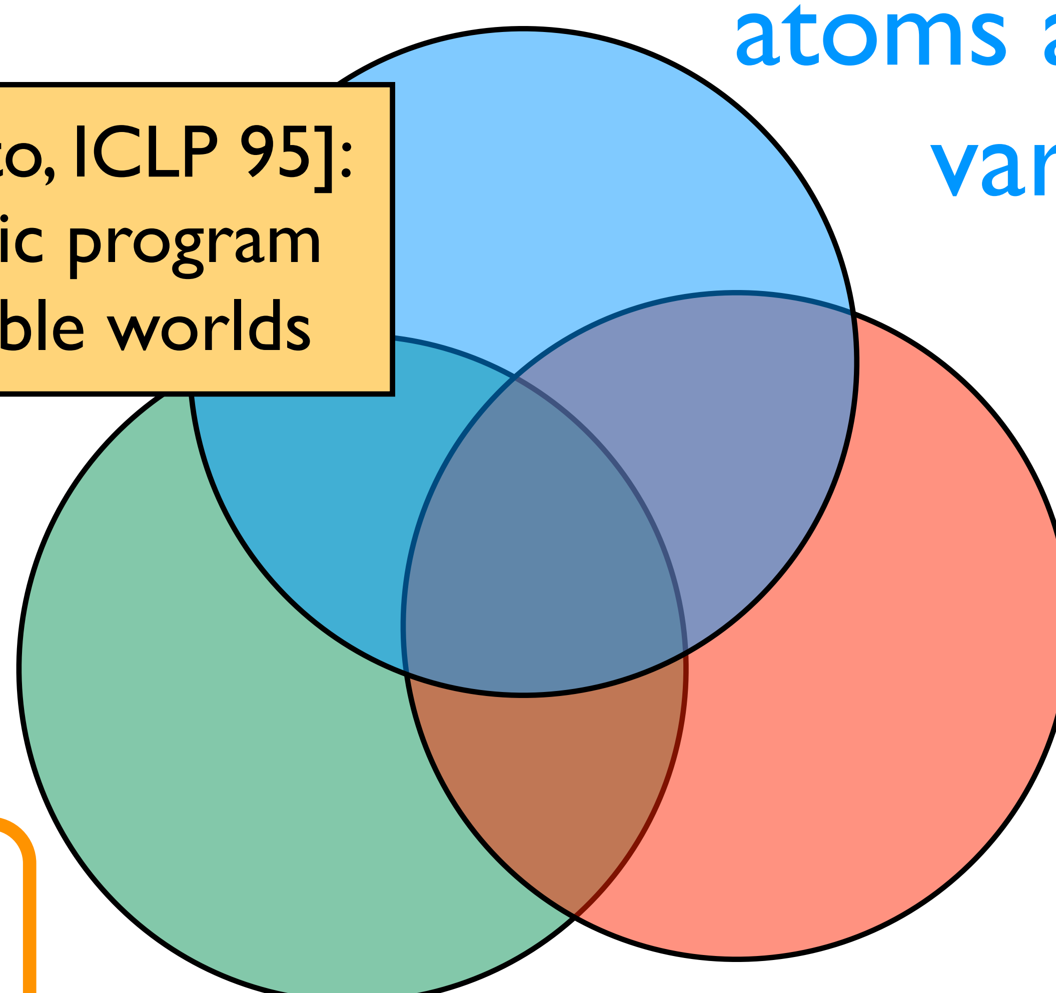
probabilistic Prolog

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

adapted relational learning techniques

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

one world

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

Overview

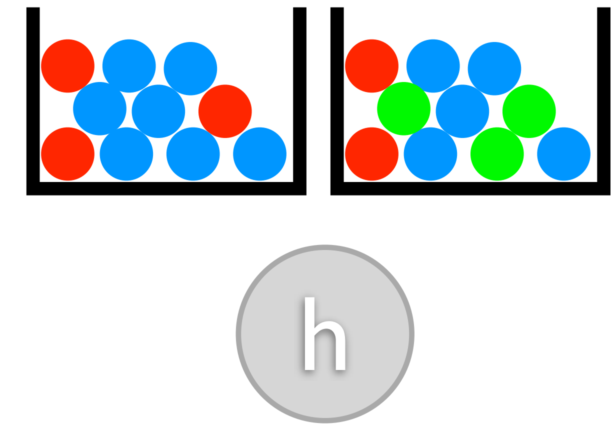
- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

ProbLog by example:

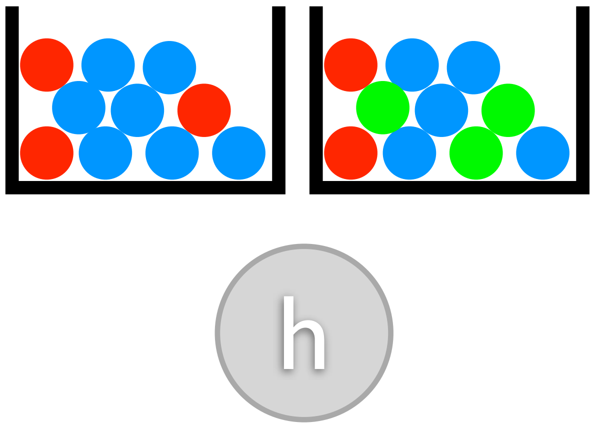
A bit of gambling



- 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



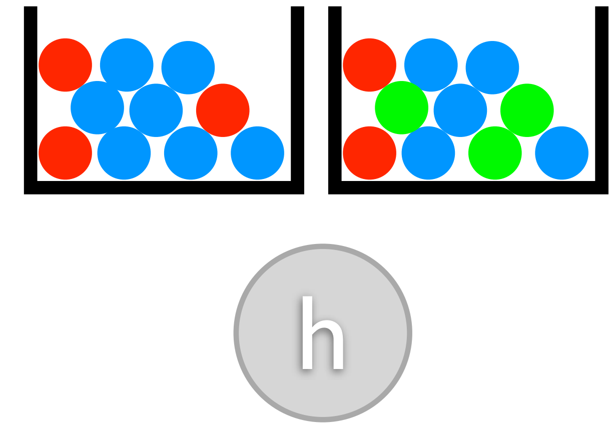
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
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)

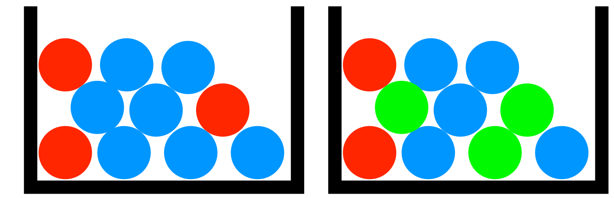
```
0.4 :: heads.
```

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

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

ProbLog by example:

A bit of gambling



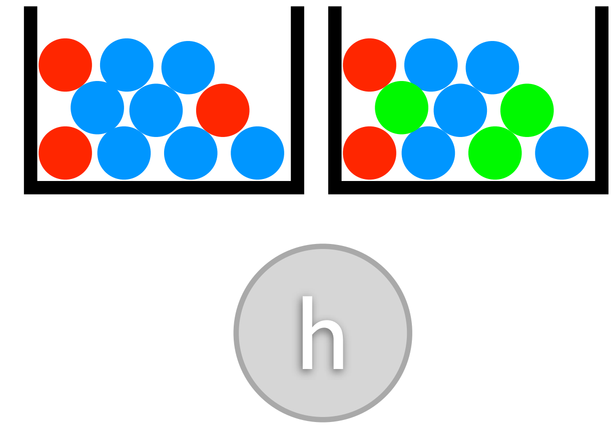
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

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

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



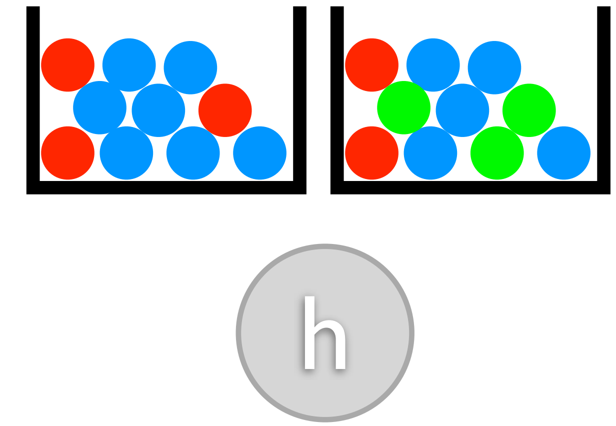
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

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

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

logical rule encoding background knowledge

ProbLog by example:

A bit of gambling



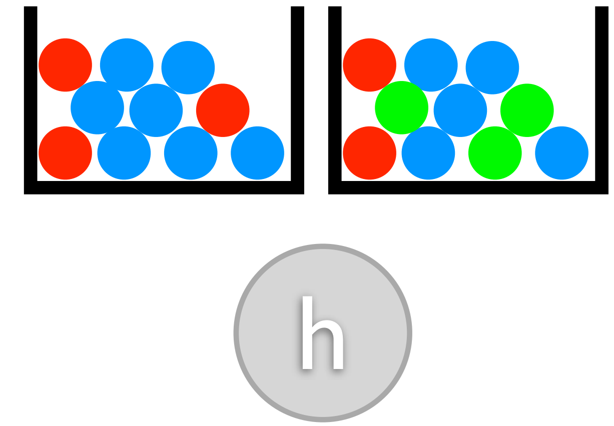
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
0.4 :: heads.

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

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)

probabilistic choices

```
0.4 :: heads.

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

consequences

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

Questions

```
0.4 :: heads.

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

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

- Probability of `win`?
- Probability of `win` given `col(2,green)`?
- Most probable world where `win` is true?

Questions

```
0.4 :: heads.

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

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

marginal probability

- Probability of `win`?

query

- Probability of `win` given `col(2,green)`?
- Most probable world where `win` is true?

Questions

```
0.4 :: heads.

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

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

marginal probability

- Probability of **`win`**?

conditional probability

- Probability of **`win`** given `col(2,green)`?

evidence

- Most probable world where **`win`** is true?

Questions

```
0.4 :: heads.

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

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

marginal probability

- Probability of `win`?

conditional probability

- Probability of `win` given `col(2,green)`?

- Most probable world where `win` is true?

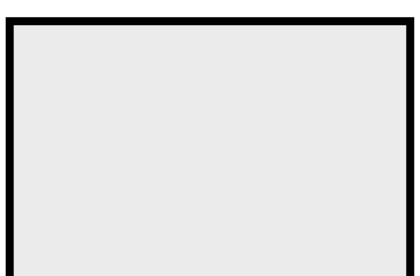
MPE inference

Possible Worlds

```
0.4 :: heads.

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

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

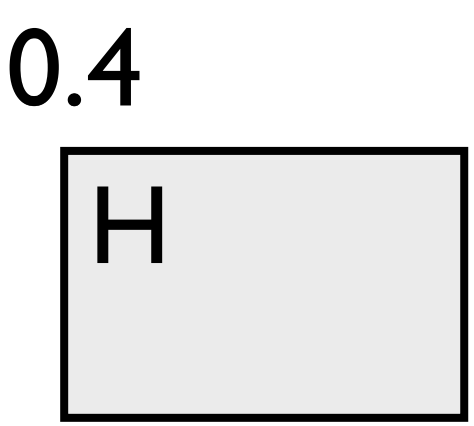
Possible Worlds

```
0.4 :: heads.

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

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



Possible Worlds

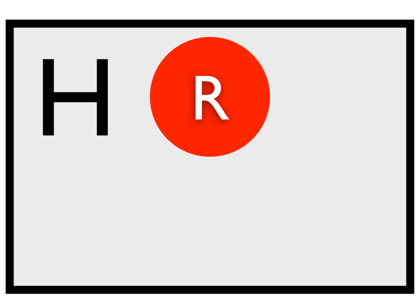
```
0.4 :: heads.

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

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) <- true.
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue) <- true.

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

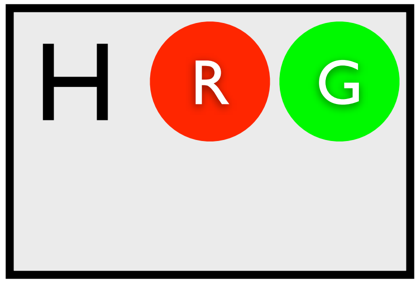
0.4×0.3



Possible Worlds

```
0.4 :: heads.

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

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

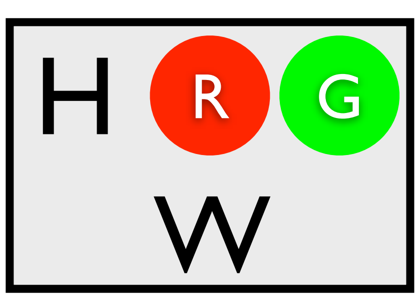
$0.4 \times 0.3 \times 0.3$



Possible Worlds

```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$



Possible Worlds

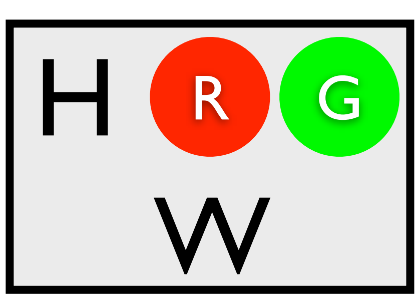
```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$ $(1-0.4)$

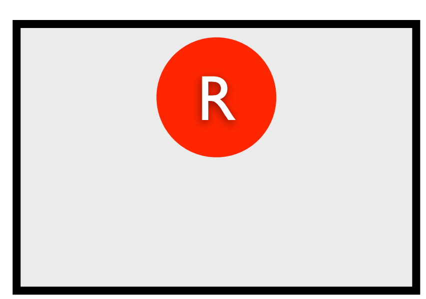
Possible Worlds

```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$



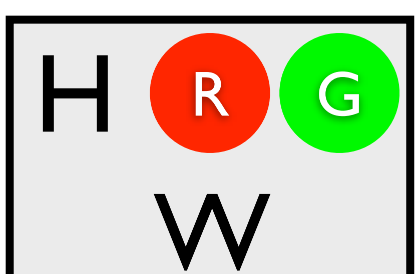
$(1-0.4)\times 0.3$



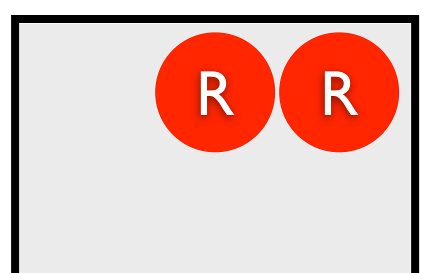
Possible Worlds

```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$



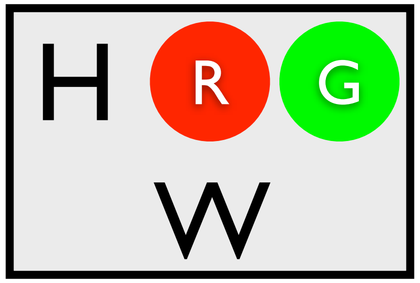
$(1-0.4) \times 0.3 \times 0.2$



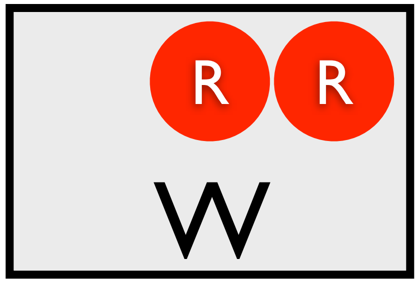
Possible Worlds

```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$



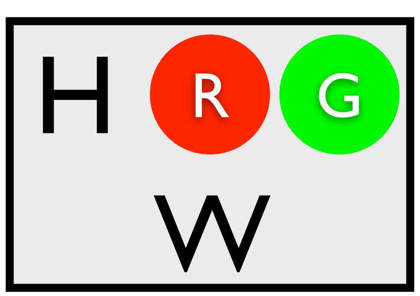
$(1-0.4) \times 0.3 \times 0.2$



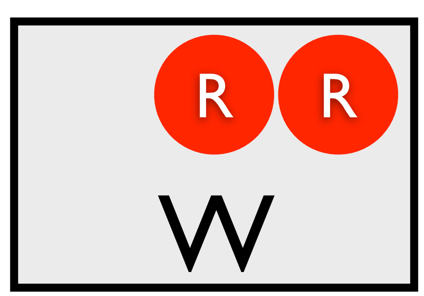
Possible Worlds

```
0.4 :: heads.

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

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

$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$

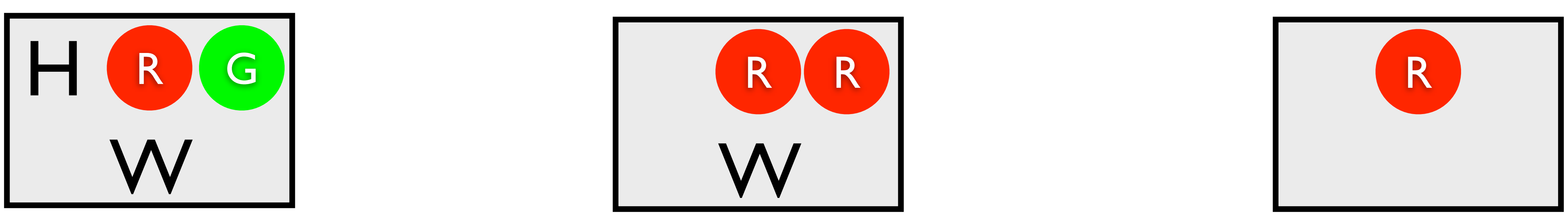


$(1-0.4)$

Possible Worlds

```
0.4 :: heads.

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

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

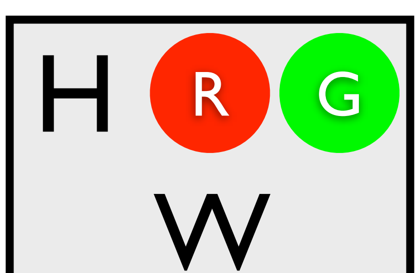
$0.4 \times 0.3 \times 0.3$ $(1-0.4) \times 0.3 \times 0.2$ $(1-0.4) \times 0.3$



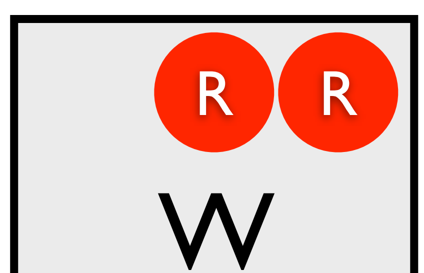
Possible Worlds

```
0.4 :: heads.

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

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

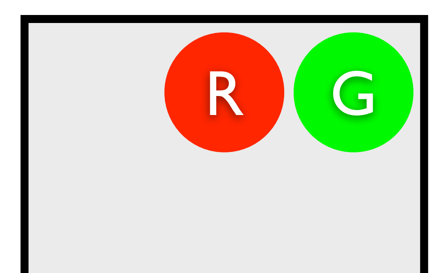
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



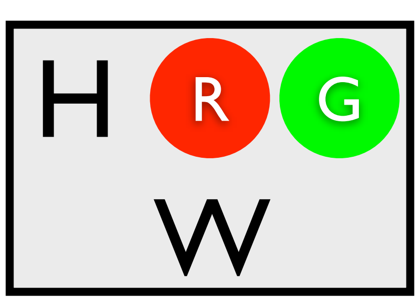
$(1-0.4) \times 0.3 \times 0.3$



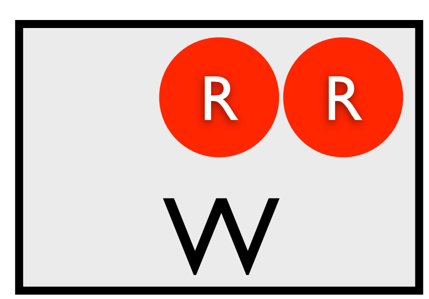
Possible Worlds

```
0.4 :: heads.

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

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

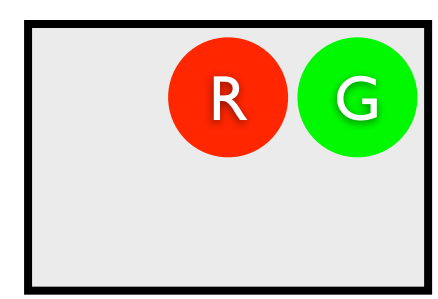
$0.4 \times 0.3 \times 0.3$



$(1-0.4) \times 0.3 \times 0.2$



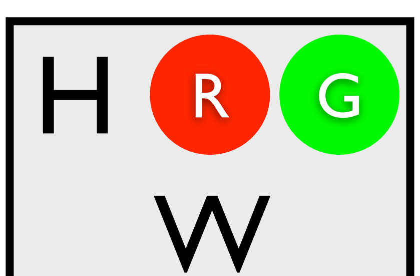
$(1-0.4) \times 0.3 \times 0.3$



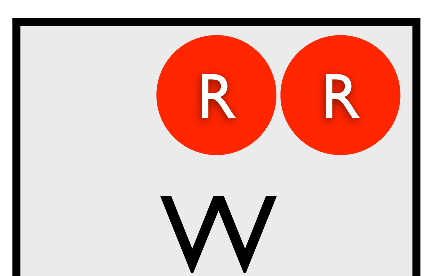
Possible Worlds

```
0.4 :: heads.

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

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

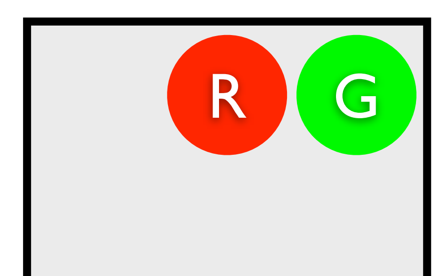
$0.4 \times 0.3 \times 0.3$



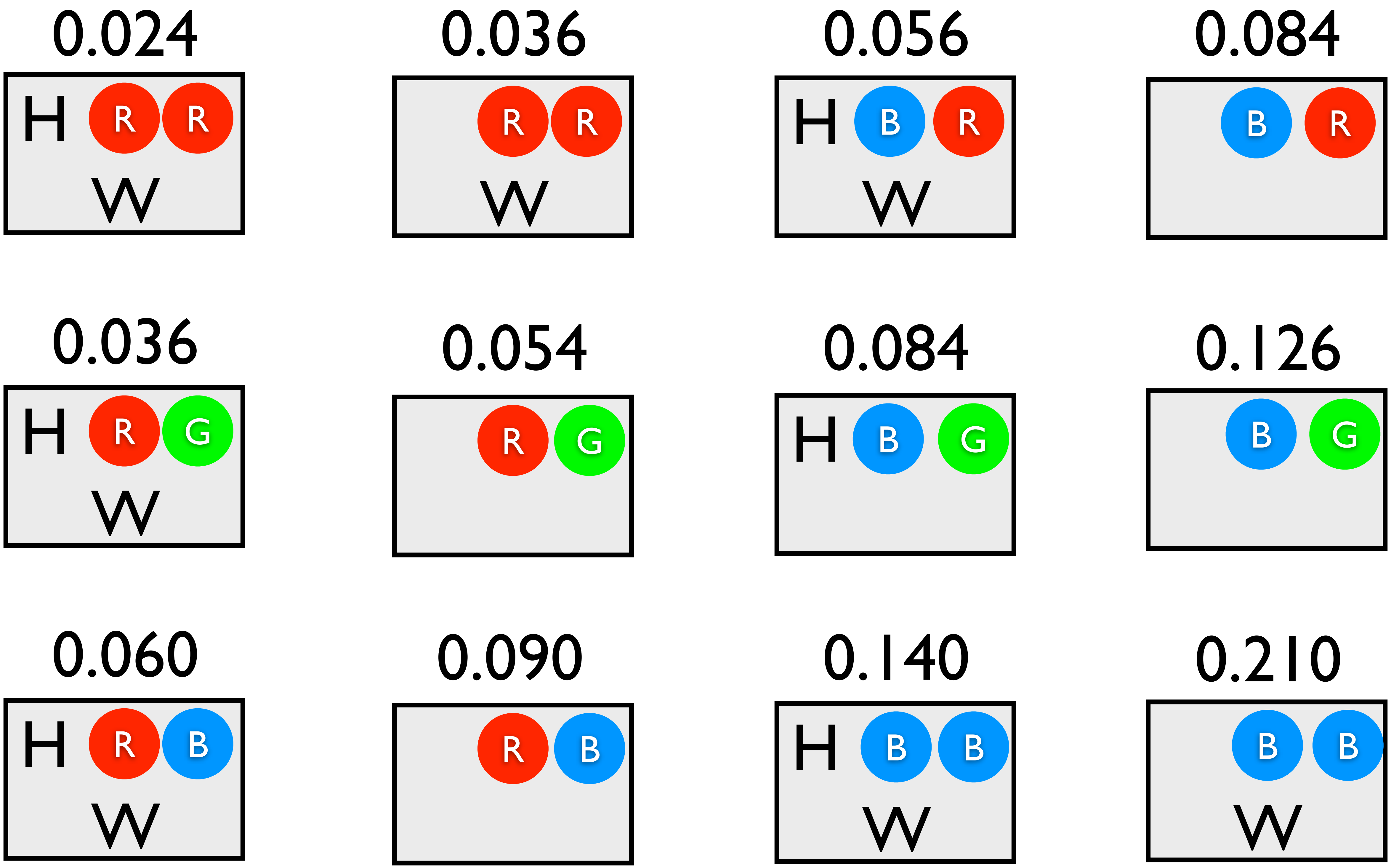
$(1-0.4)\times 0.3 \times 0.2$



$(1-0.4)\times 0.3 \times 0.3$

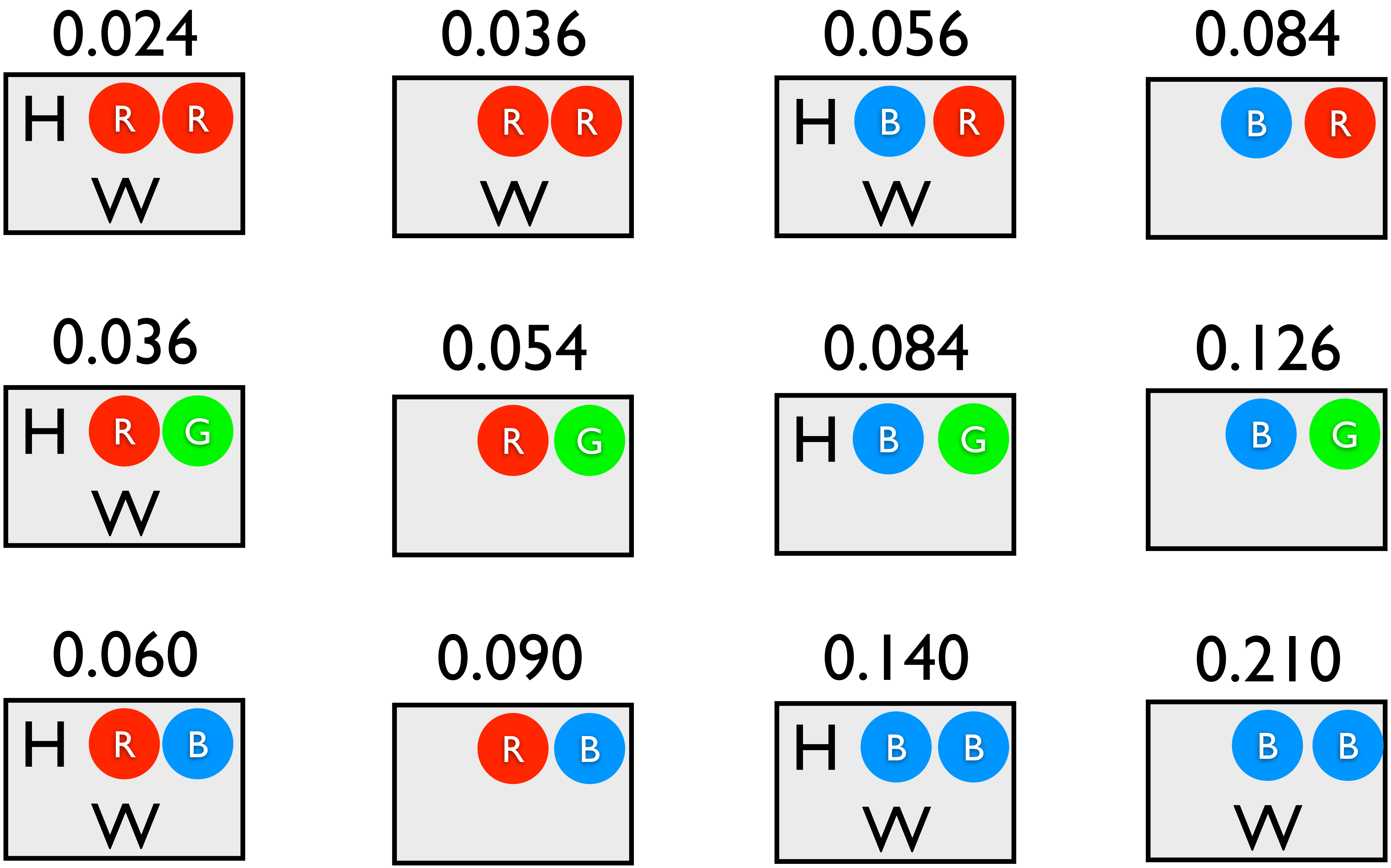


All Possible Worlds



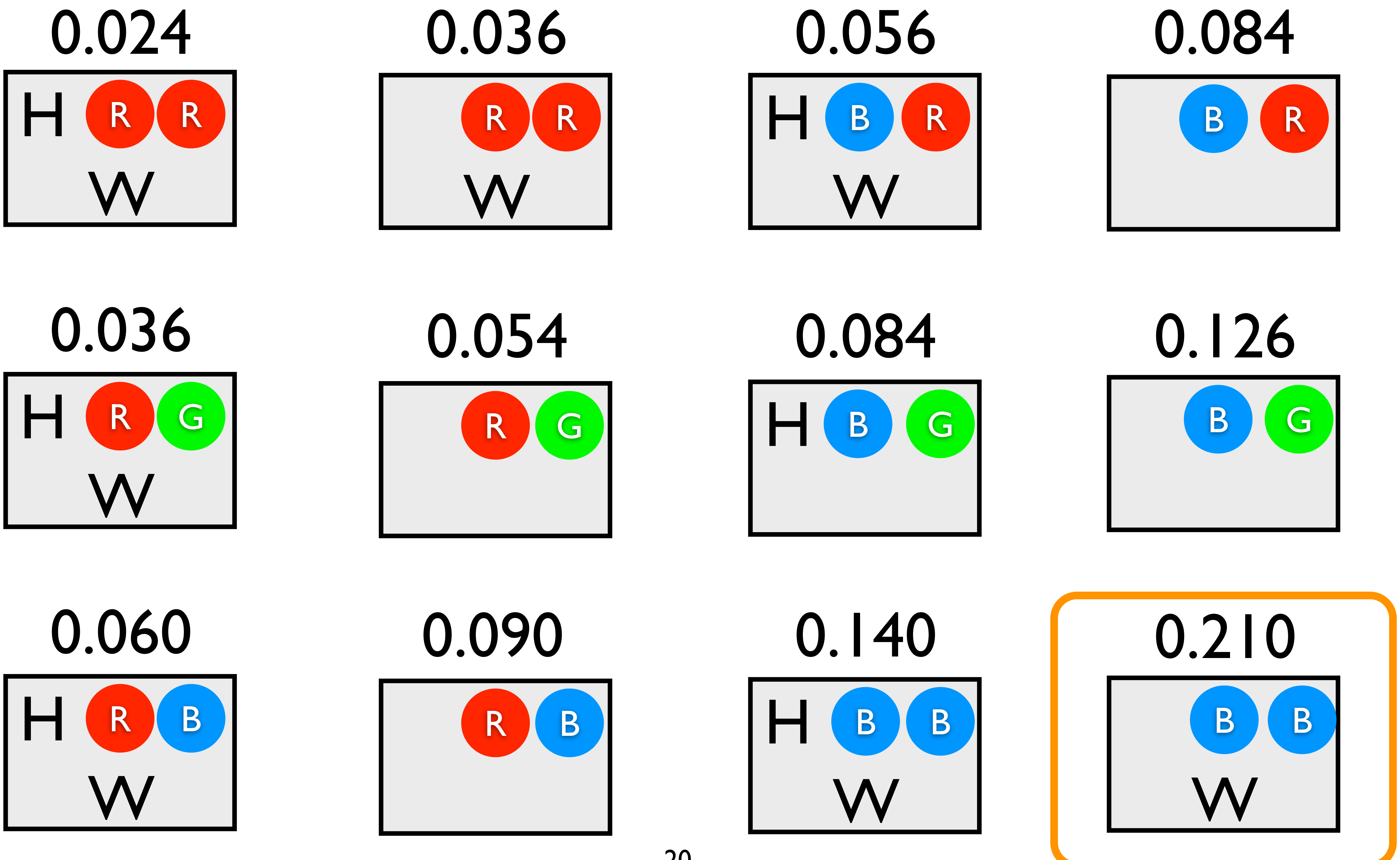
Most likely world where **`win`** is true?

MPE Inference



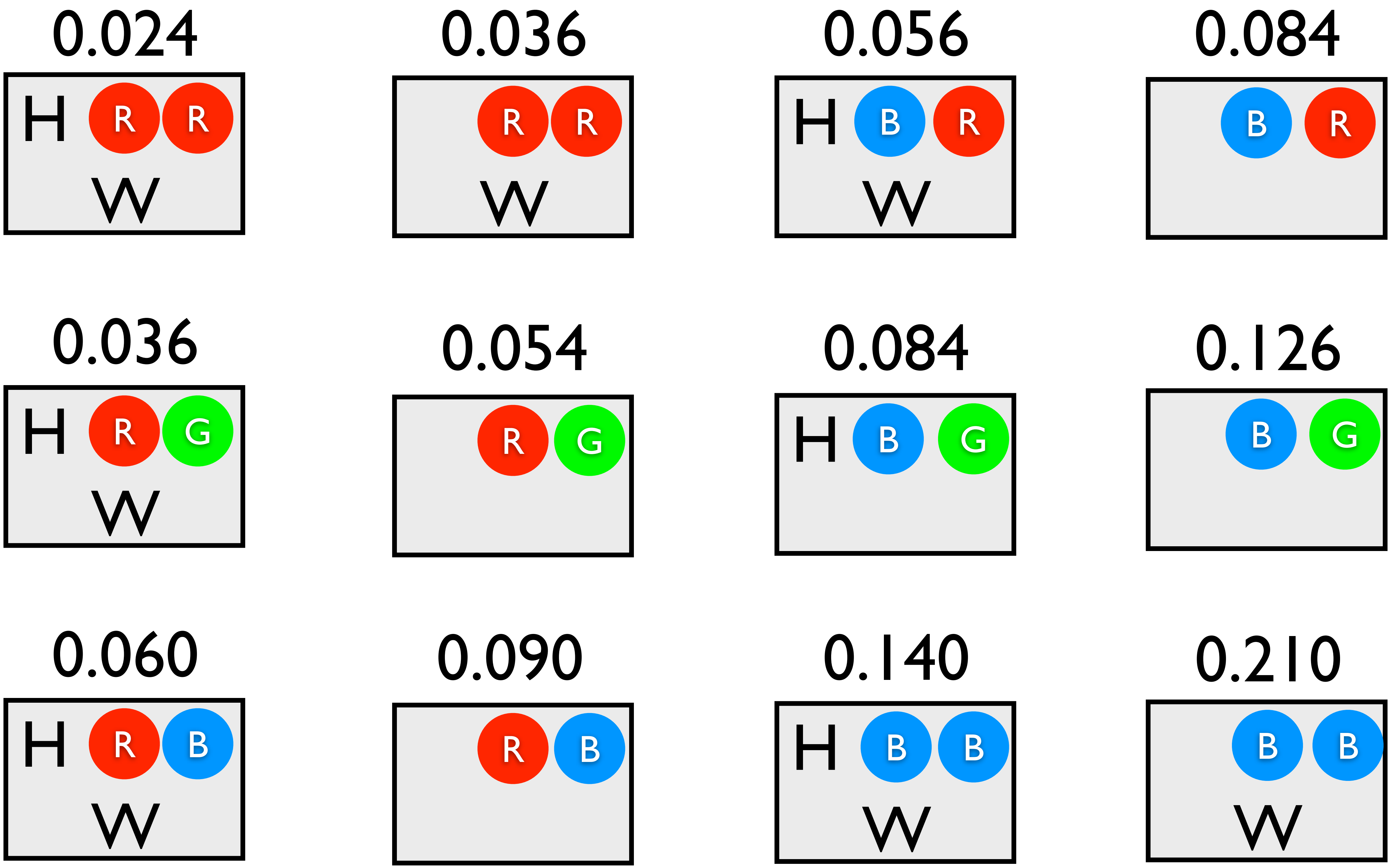
Most likely world where **`win`** is true?

MPE Inference



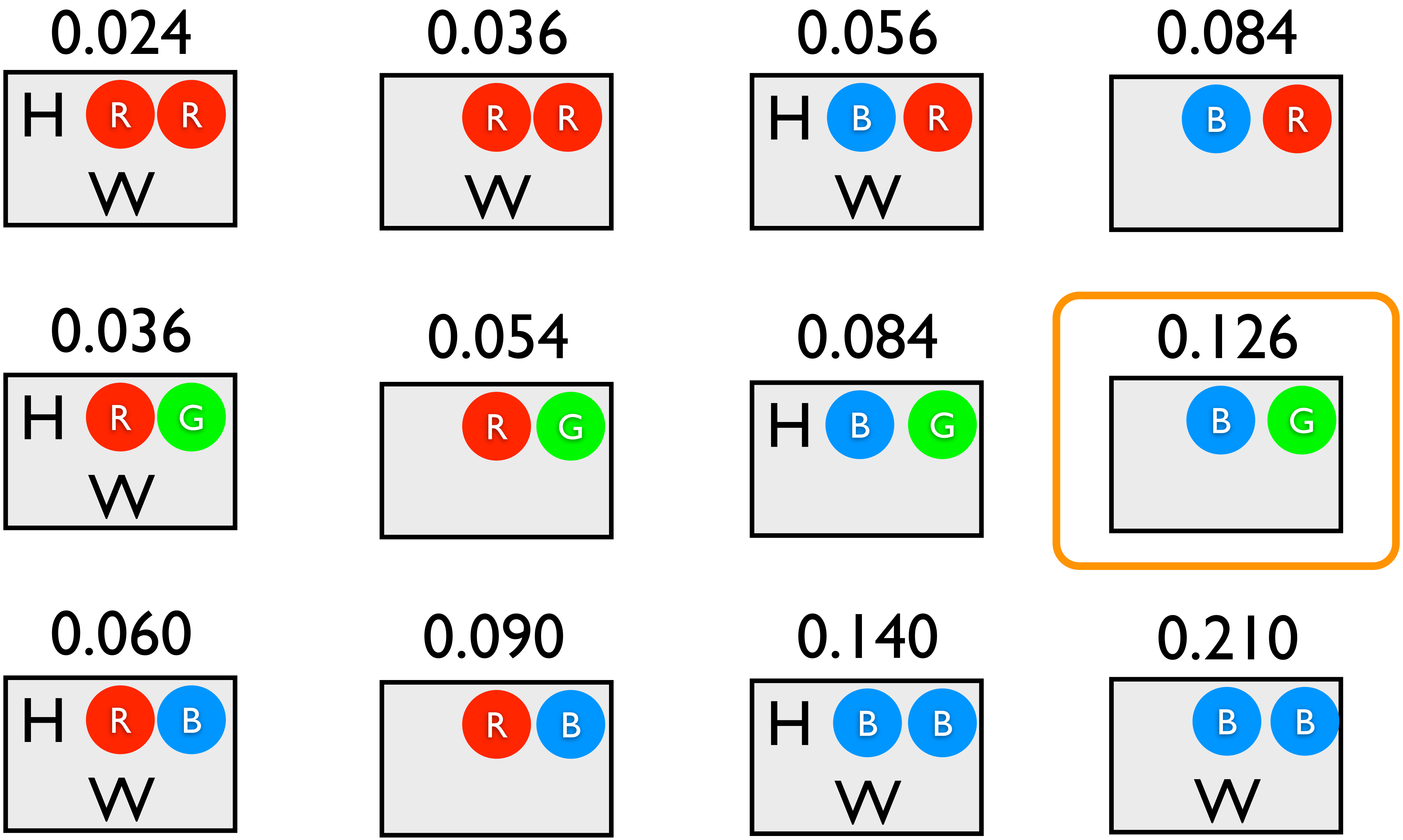
Most likely world where `col(2,blue)` is false?

MPE Inference



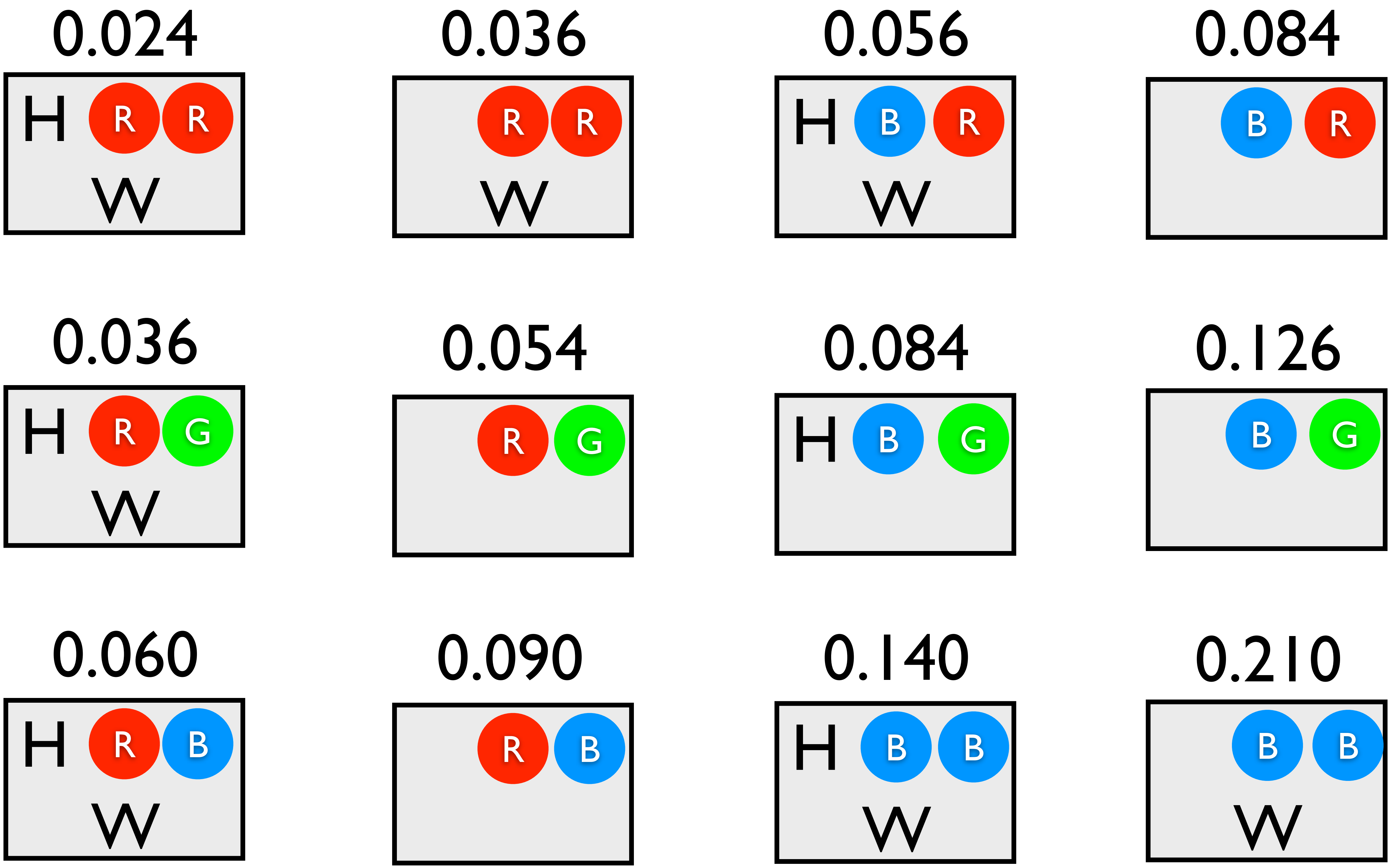
Most likely world where `col(2,blue)` is false?

MPE Inference



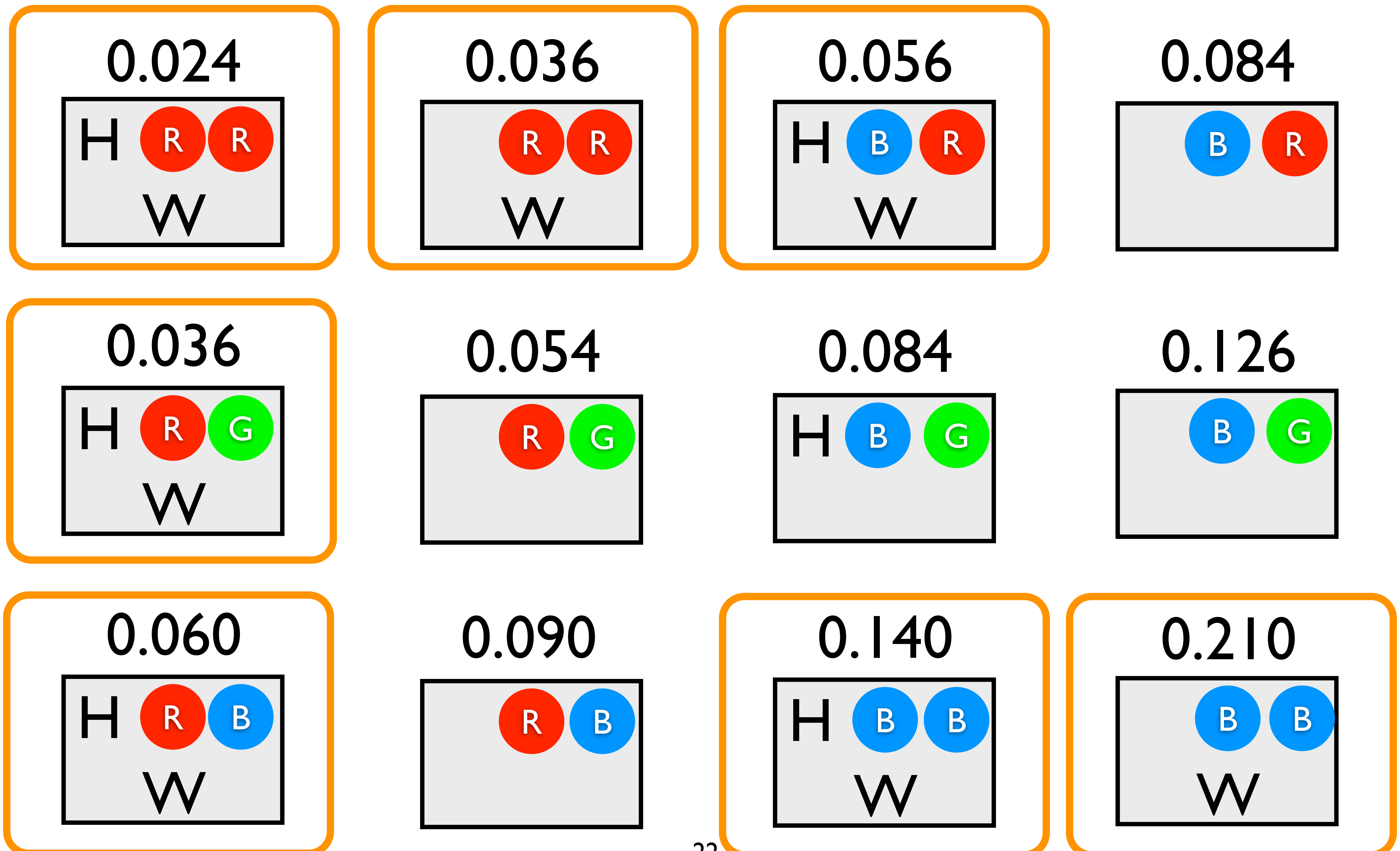
P(win)=?

Marginal Probability



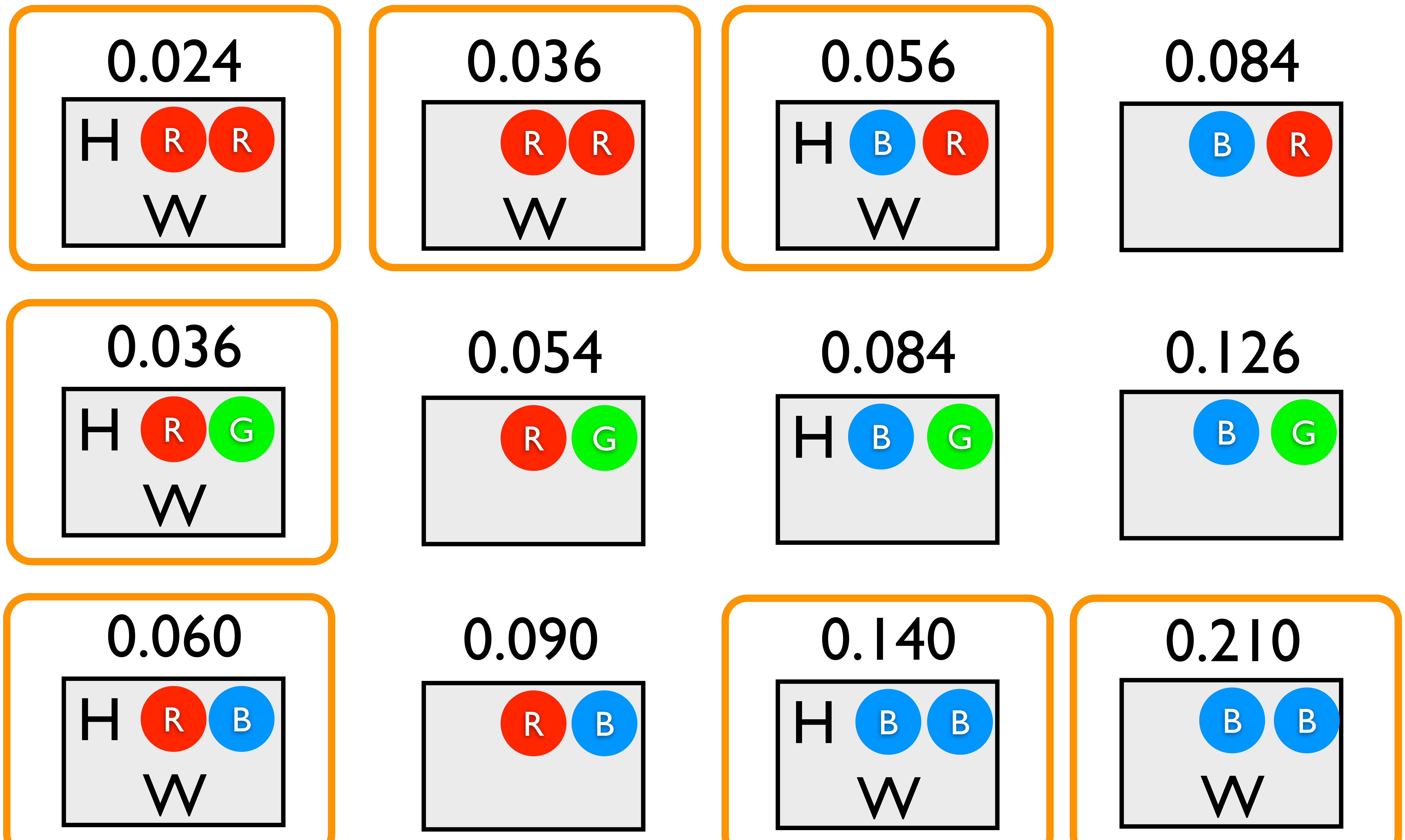
$P(\underline{\text{win}}) = \sum$

Marginal Probability



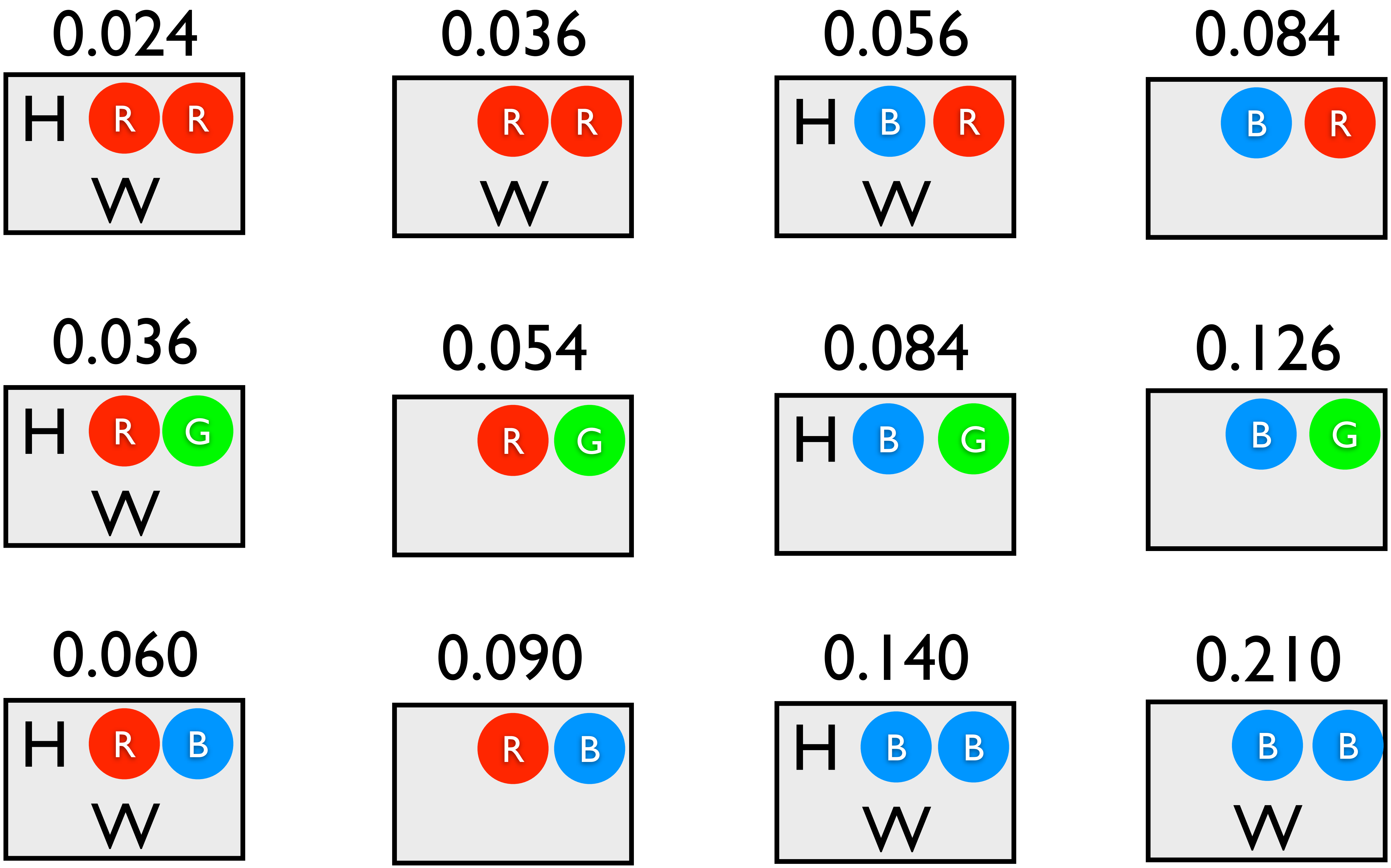
$P(\underline{win}) = \sum = 0.562$

Marginal Probability



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

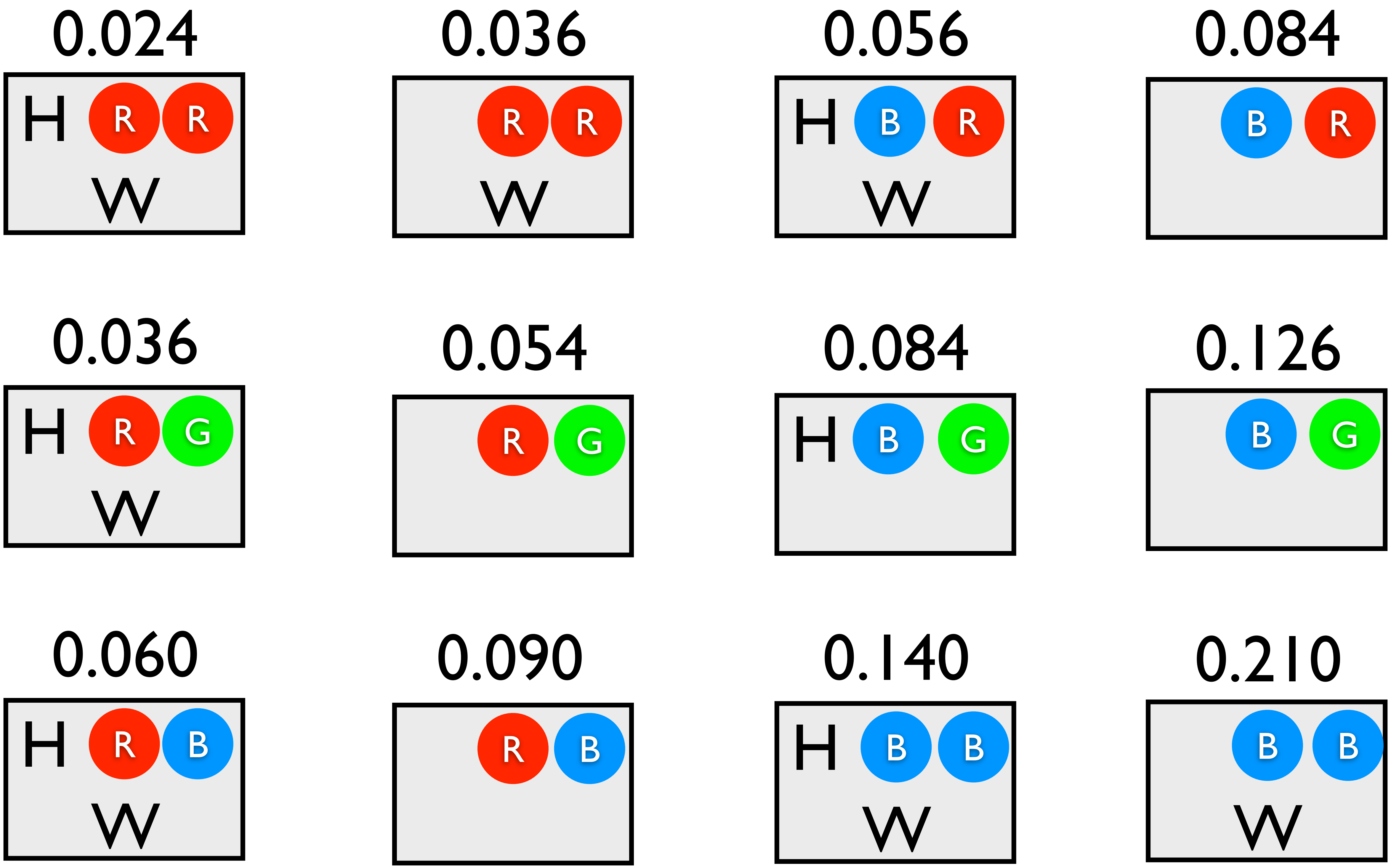
Conditional Probability



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

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

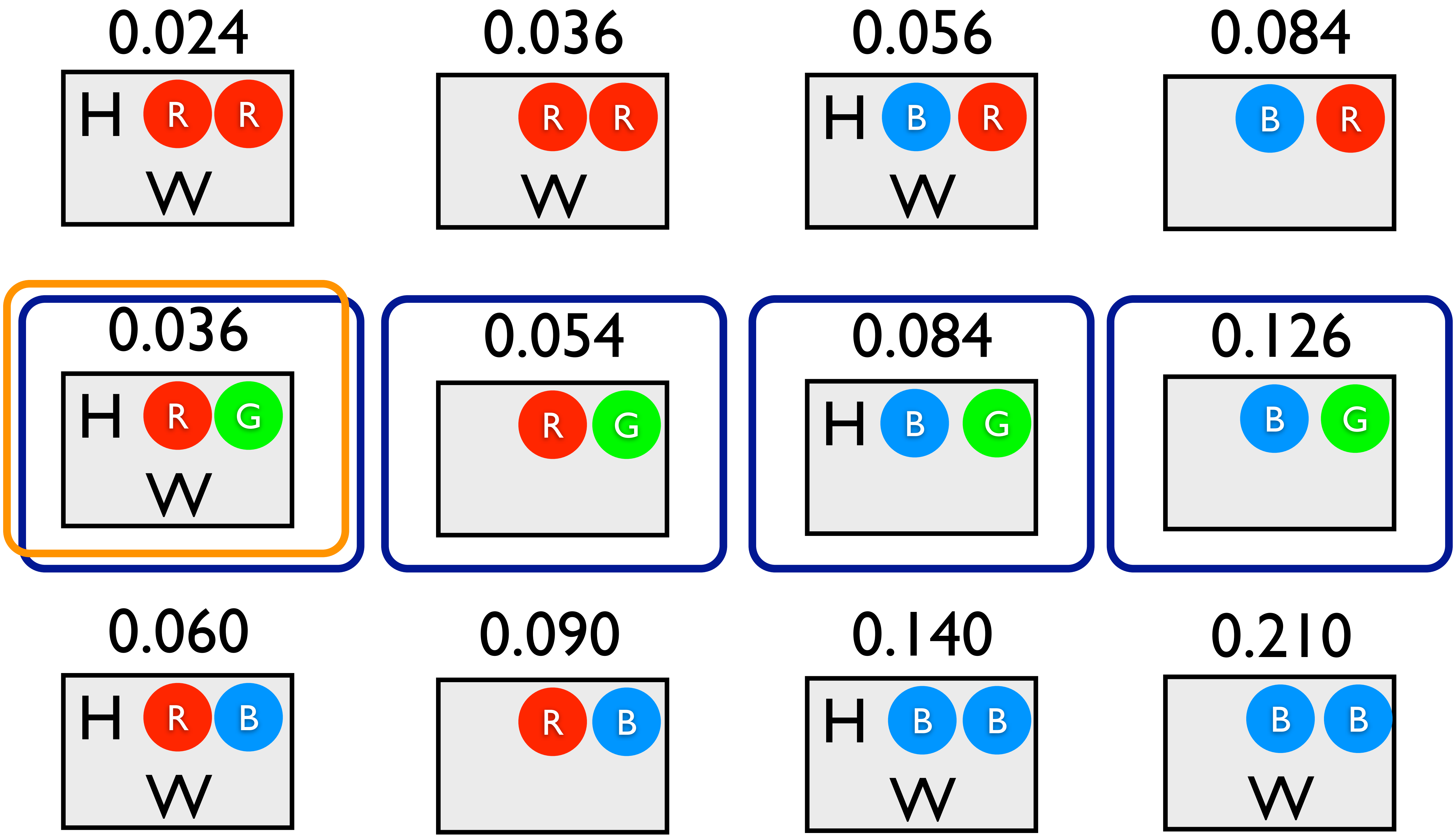
Conditional Probability



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

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

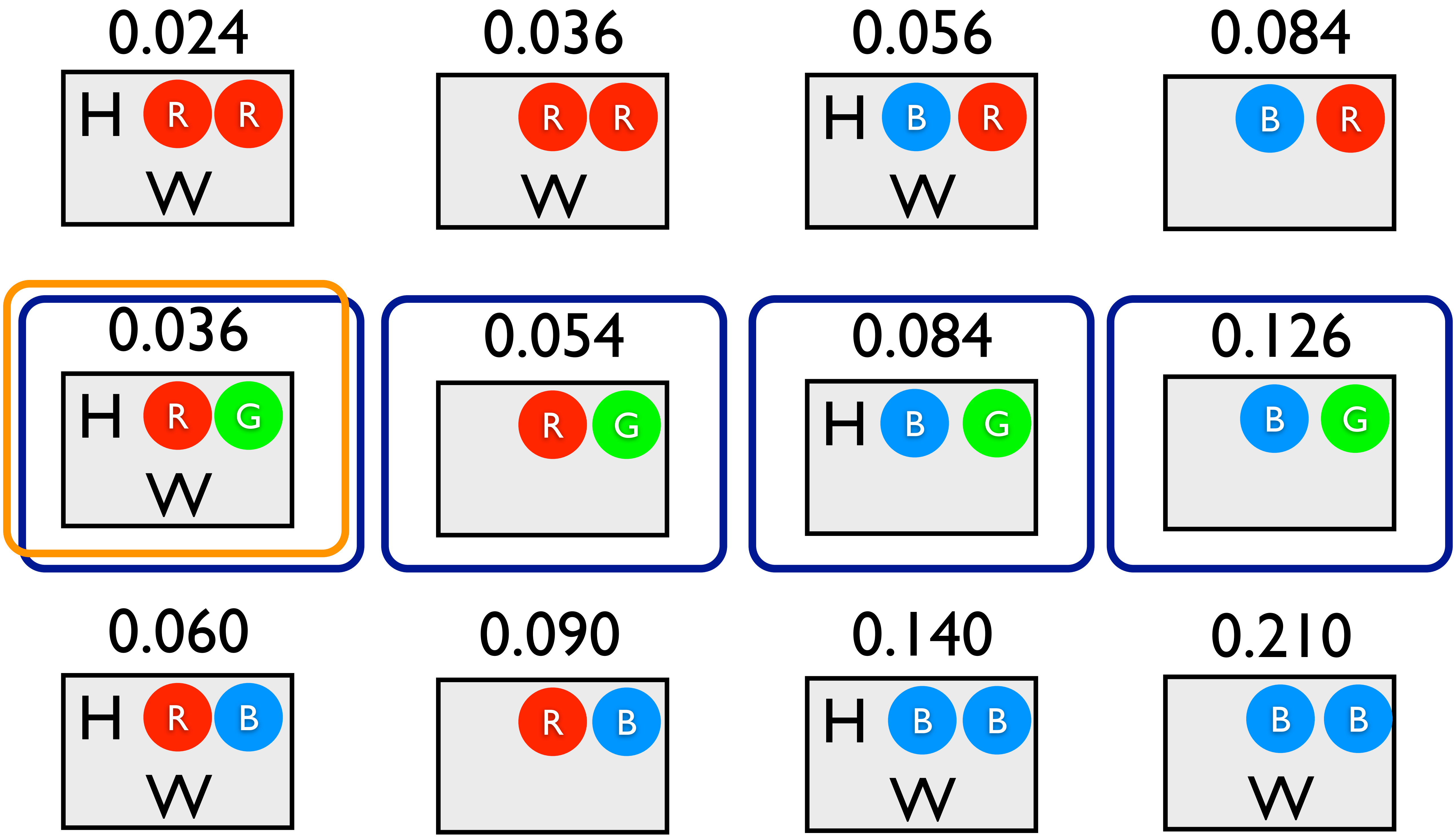
Conditional Probability



Conditional Probability

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

$= 0.036/0.3 = 0.12$

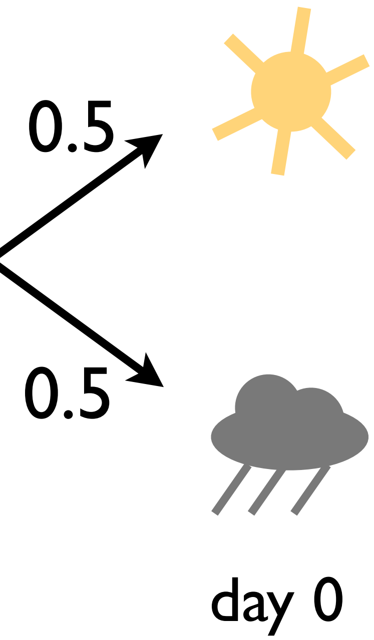


ProbLog by example:

Rain or sun?

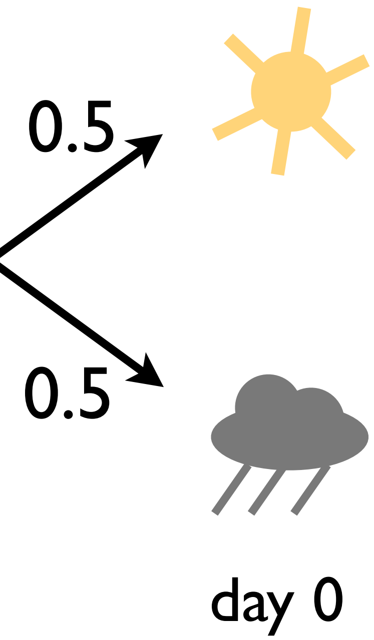
ProbLog by example:

Rain or sun?



ProbLog by example:

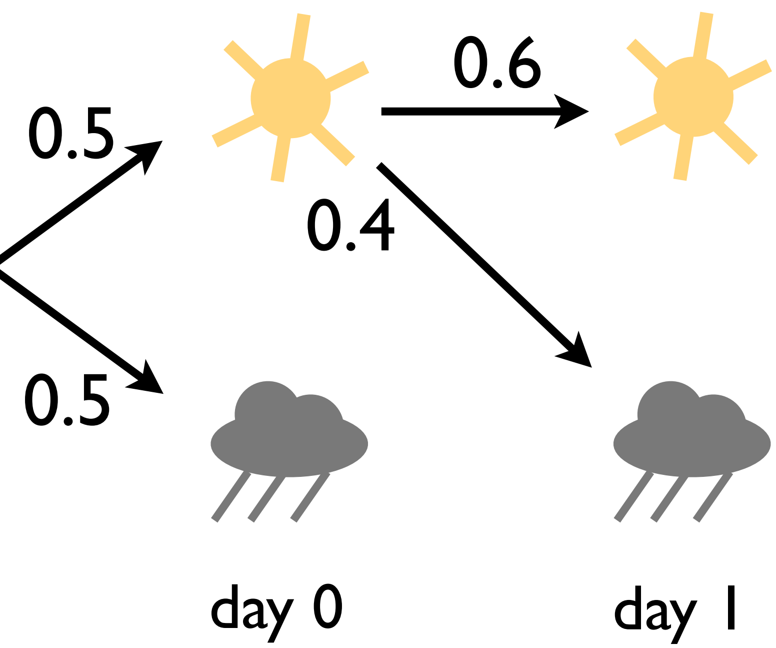
Rain or sun?




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

ProbLog by example:

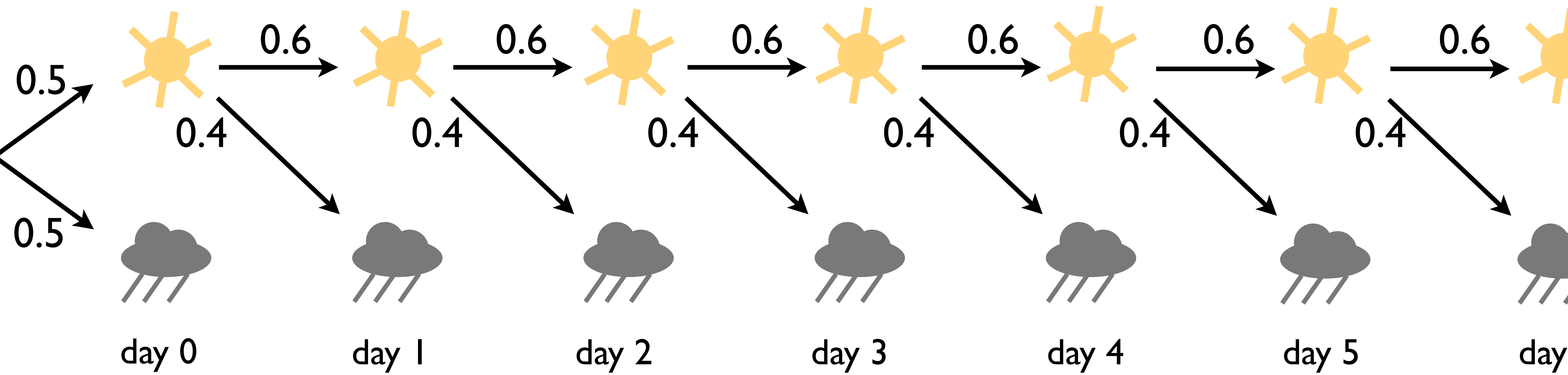
Rain or sun?




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

ProbLog by example:

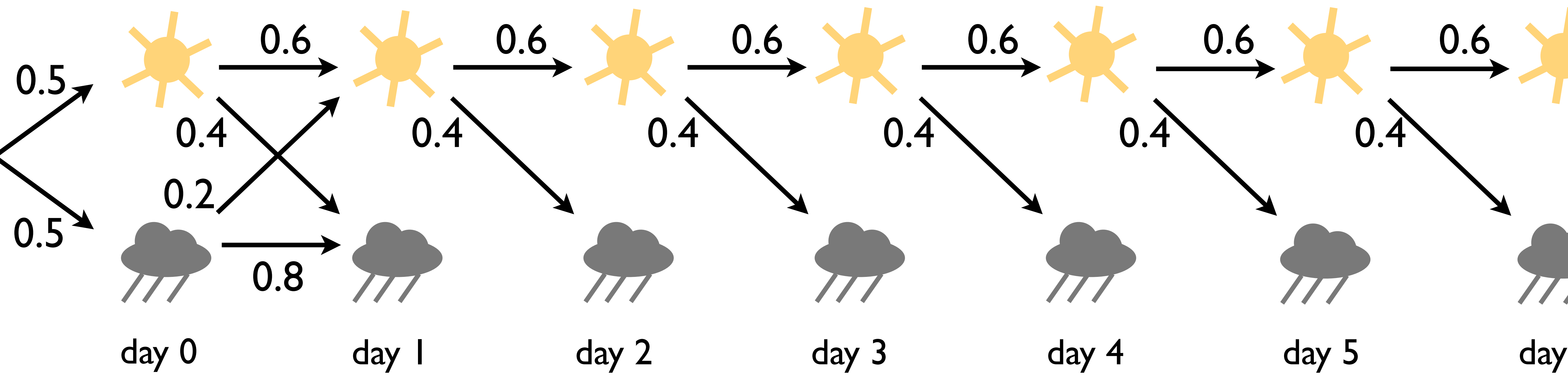
Rain or sun?




```
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:

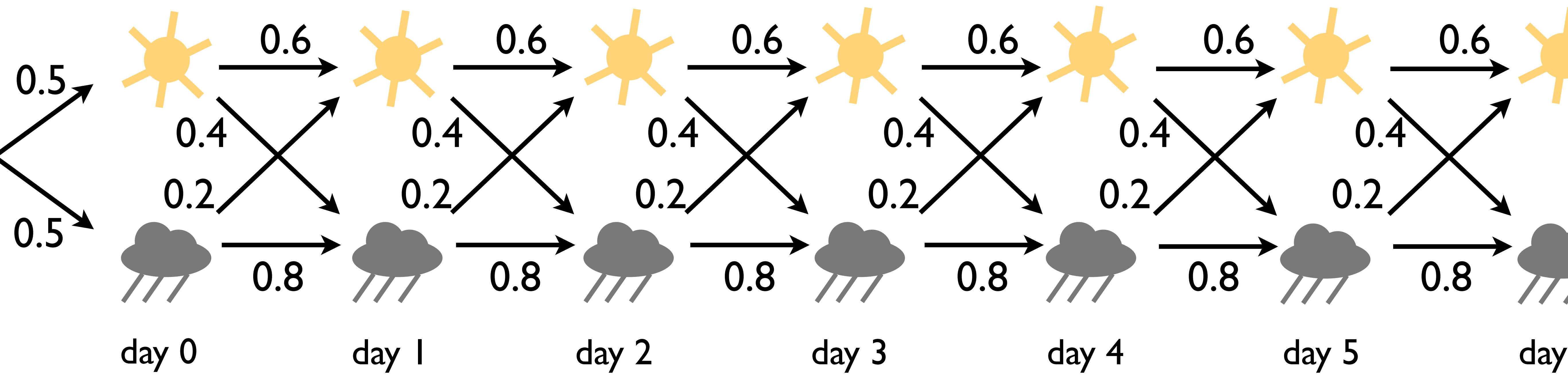
Rain or sun?




```
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).
```

ProbLog by example:

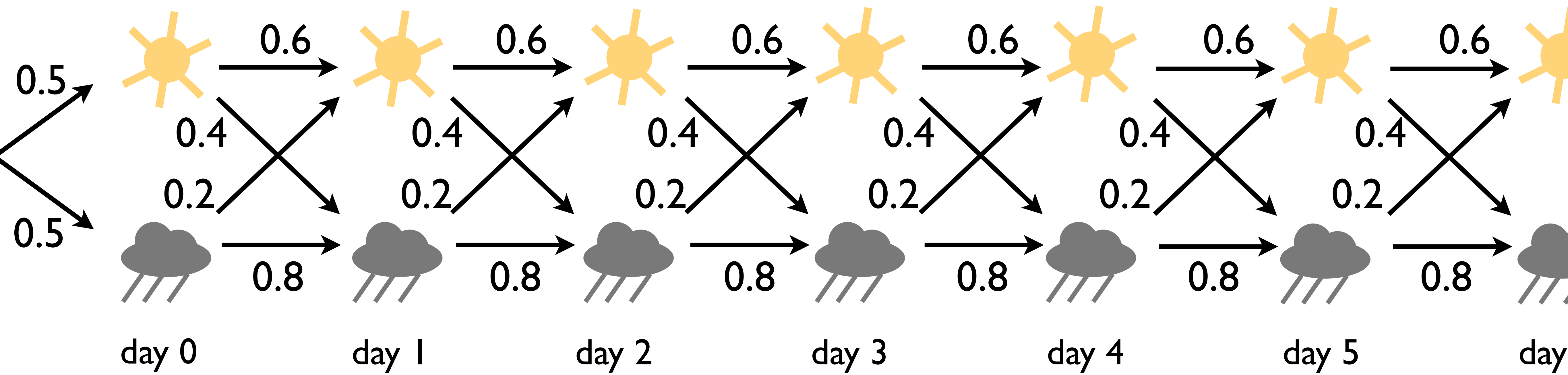
Rain or sun?




```
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?

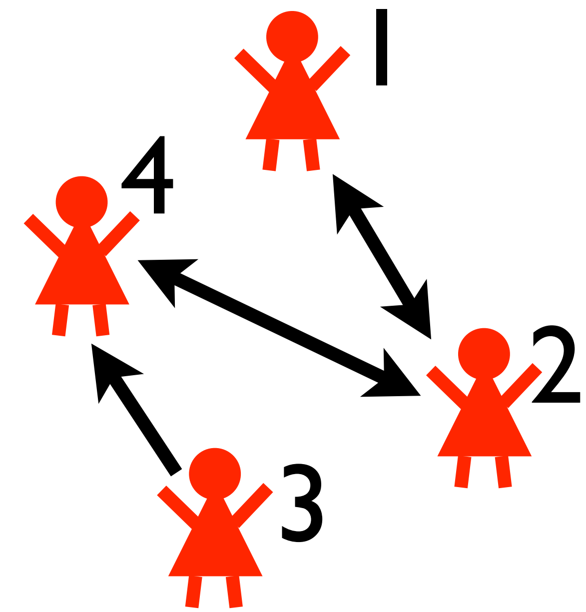



```
0.5::weather(sun,0) ; 0.5::weather(rain,0) <- true.

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 suffice to answer any given ground query

ProbLog by example:

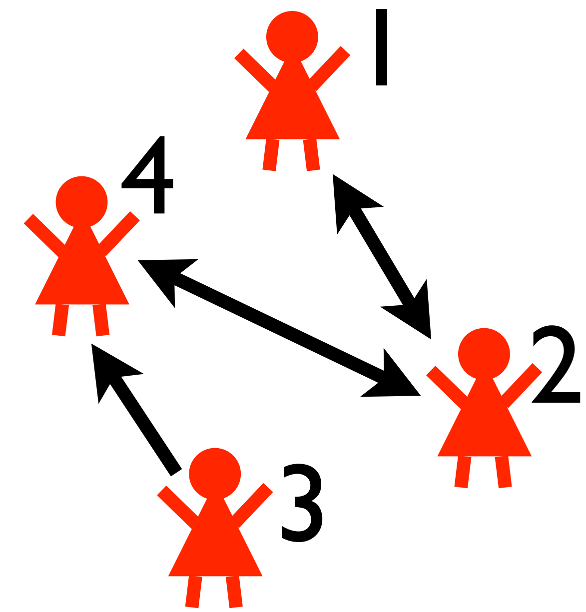
Friends & smokers




```
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

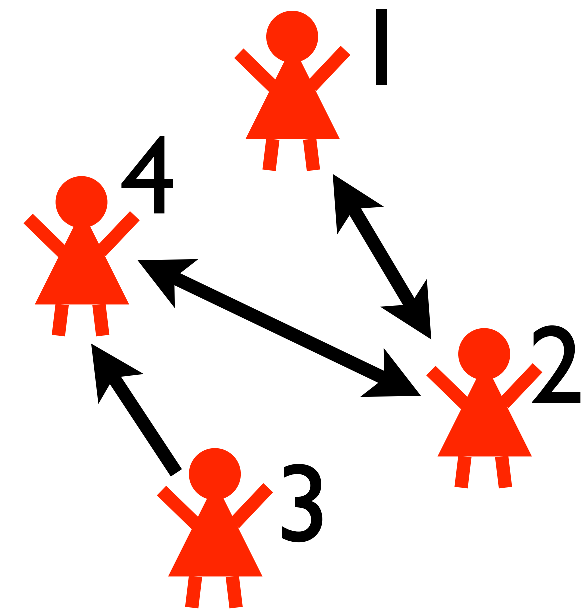
= a probabilistic fact for each grounding

```
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).
```

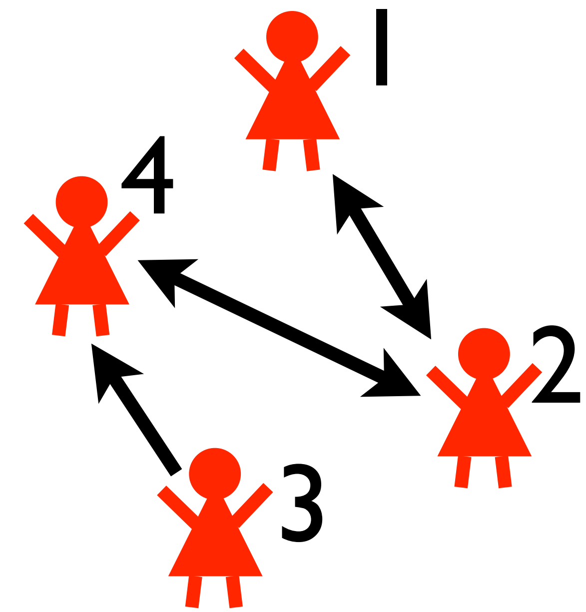
```
0.3::stress(1).
0.3::stress(2).
0.3::stress(3).
0.3::stress(4).
```

```
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

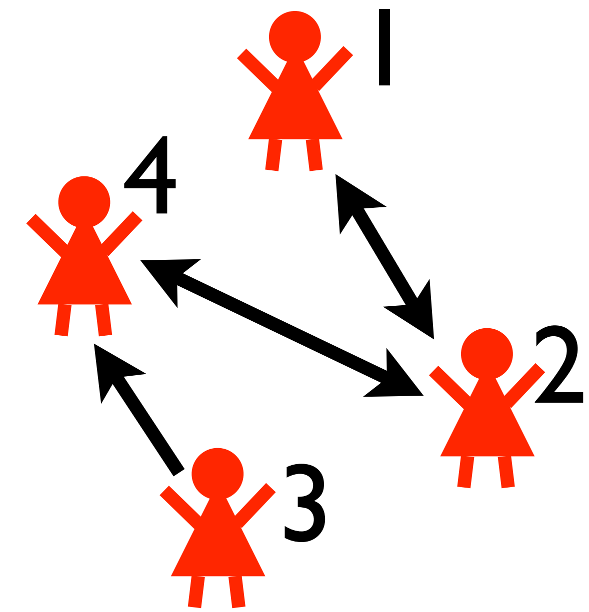



```
0.3::stress(X):- person(X).
0.2::influences(X,Y):-
            person(X), person(Y).

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).
```

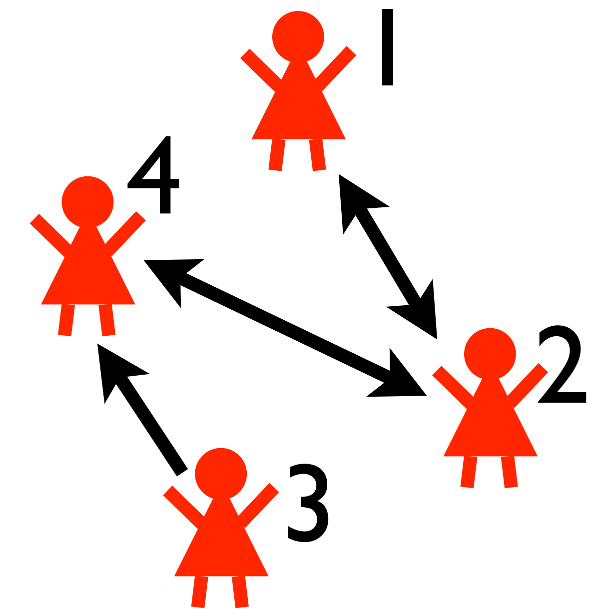
```
person(1).
person(2).
person(3).
person(4).

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).
```

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.
```

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.
```

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).
```

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

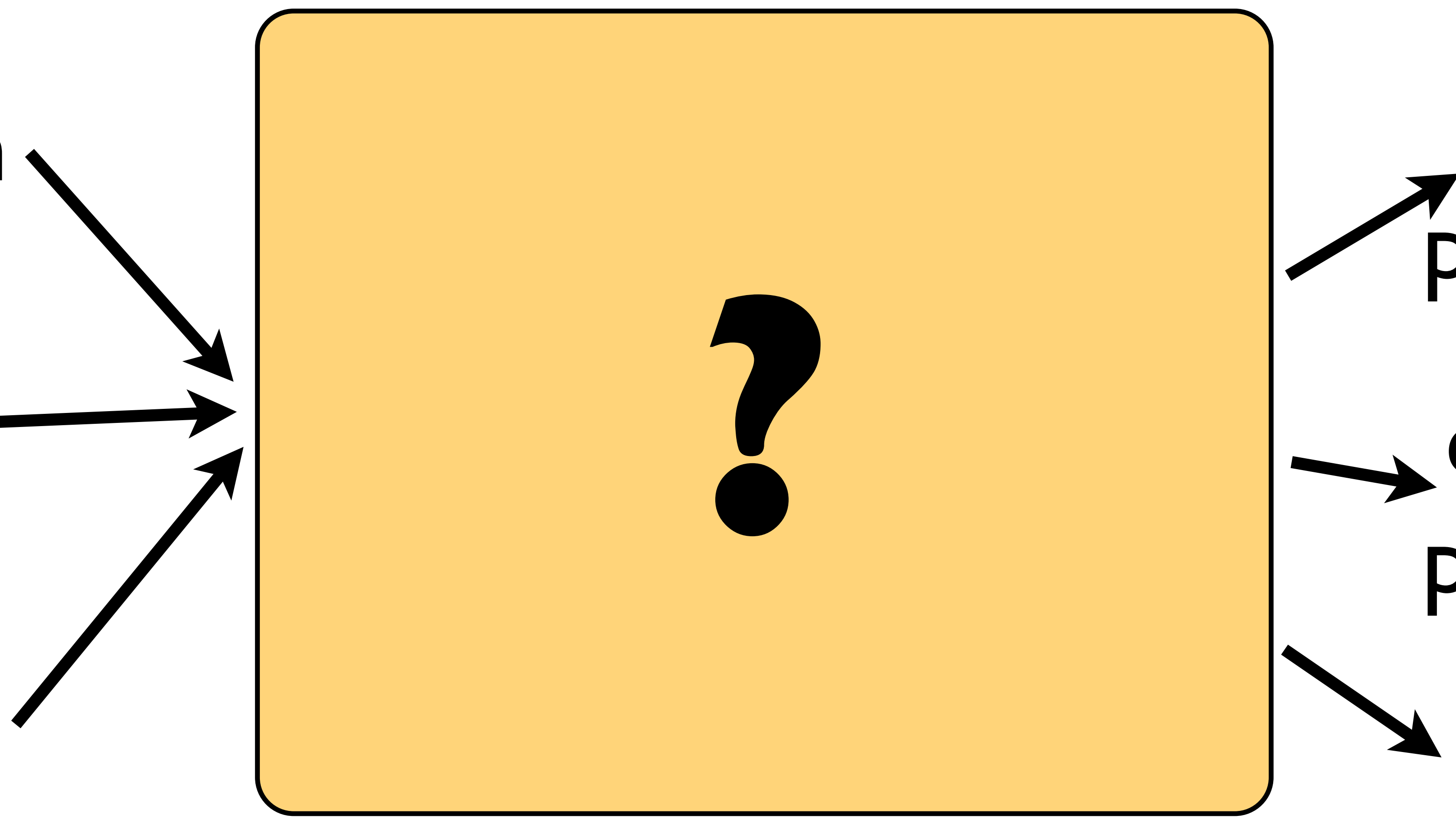
Answering Questions

Given:

program

queries

evidence



Find:

marginal probabilities

conditional probabilities

MPE state

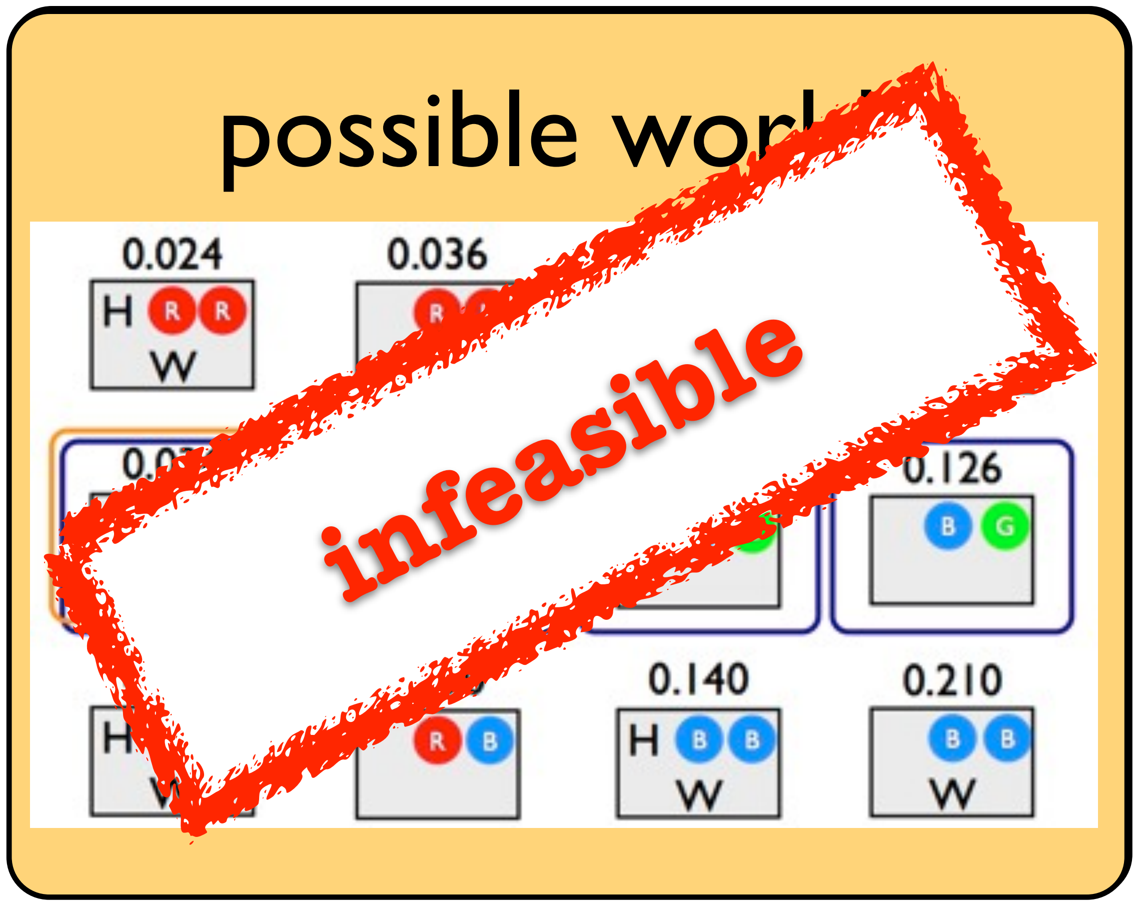
Answering Questions

Given:

program

queries

evidence



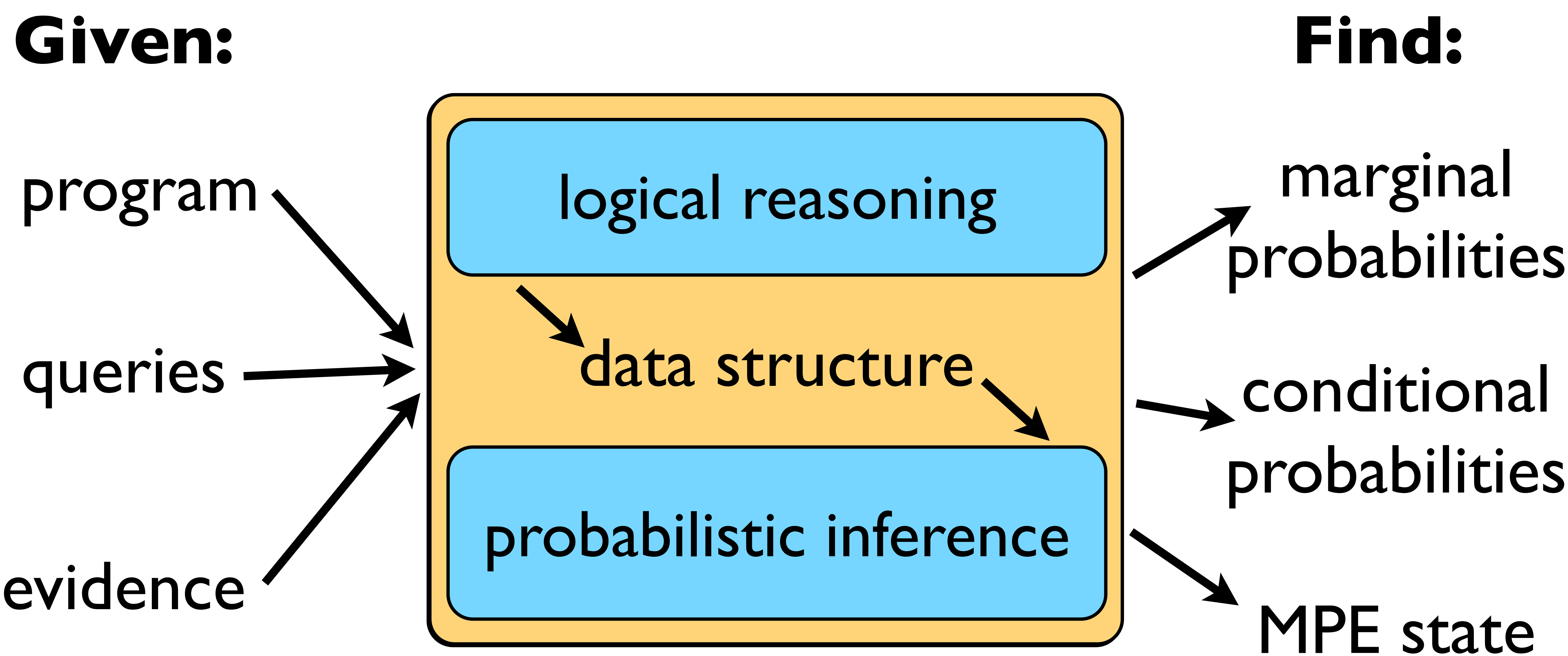
Find:

marginal probabilities

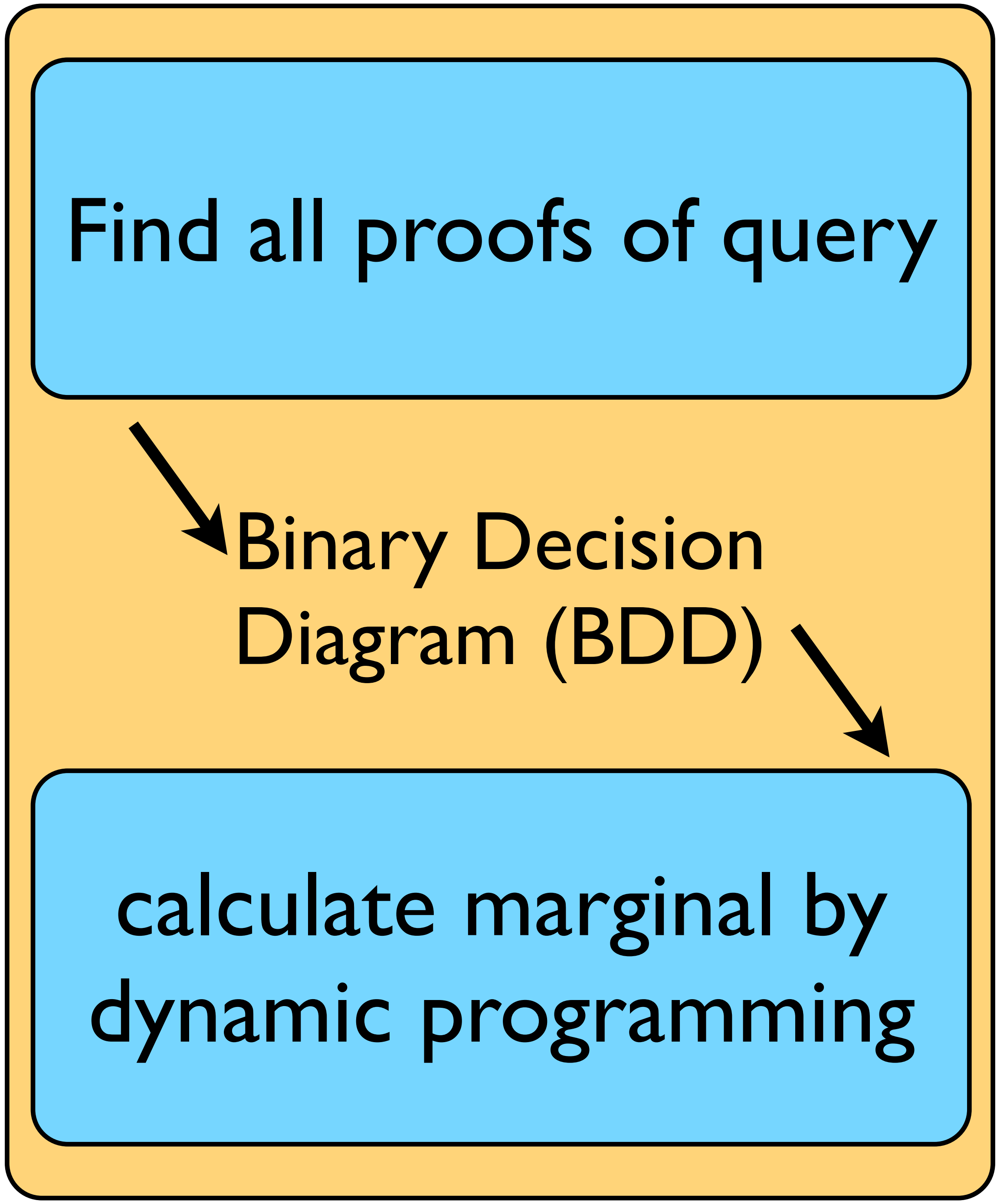
conditional probabilities

MPE state

Answering Questions



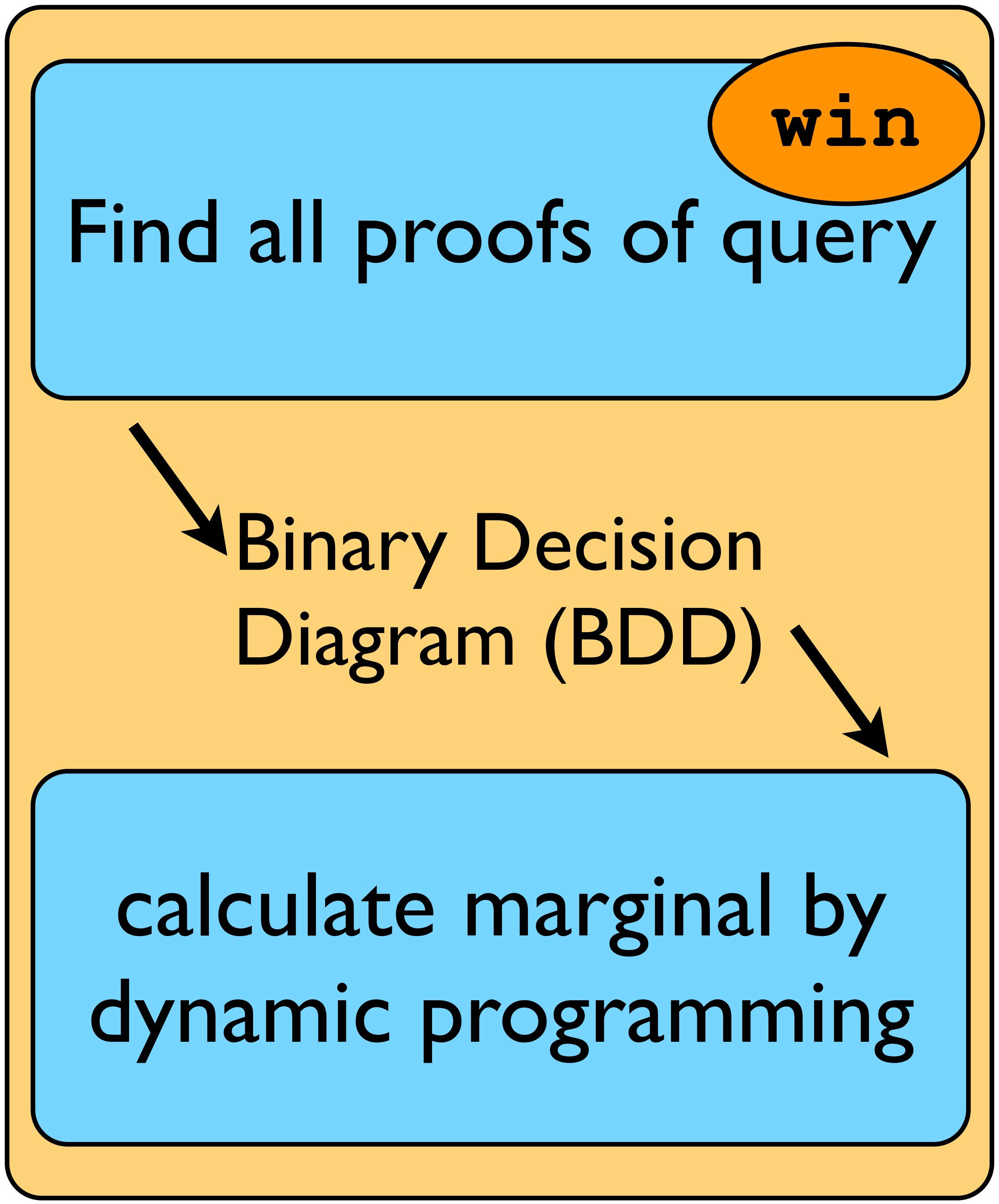
Initial Approach
(ProbLog1)



 [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).
```

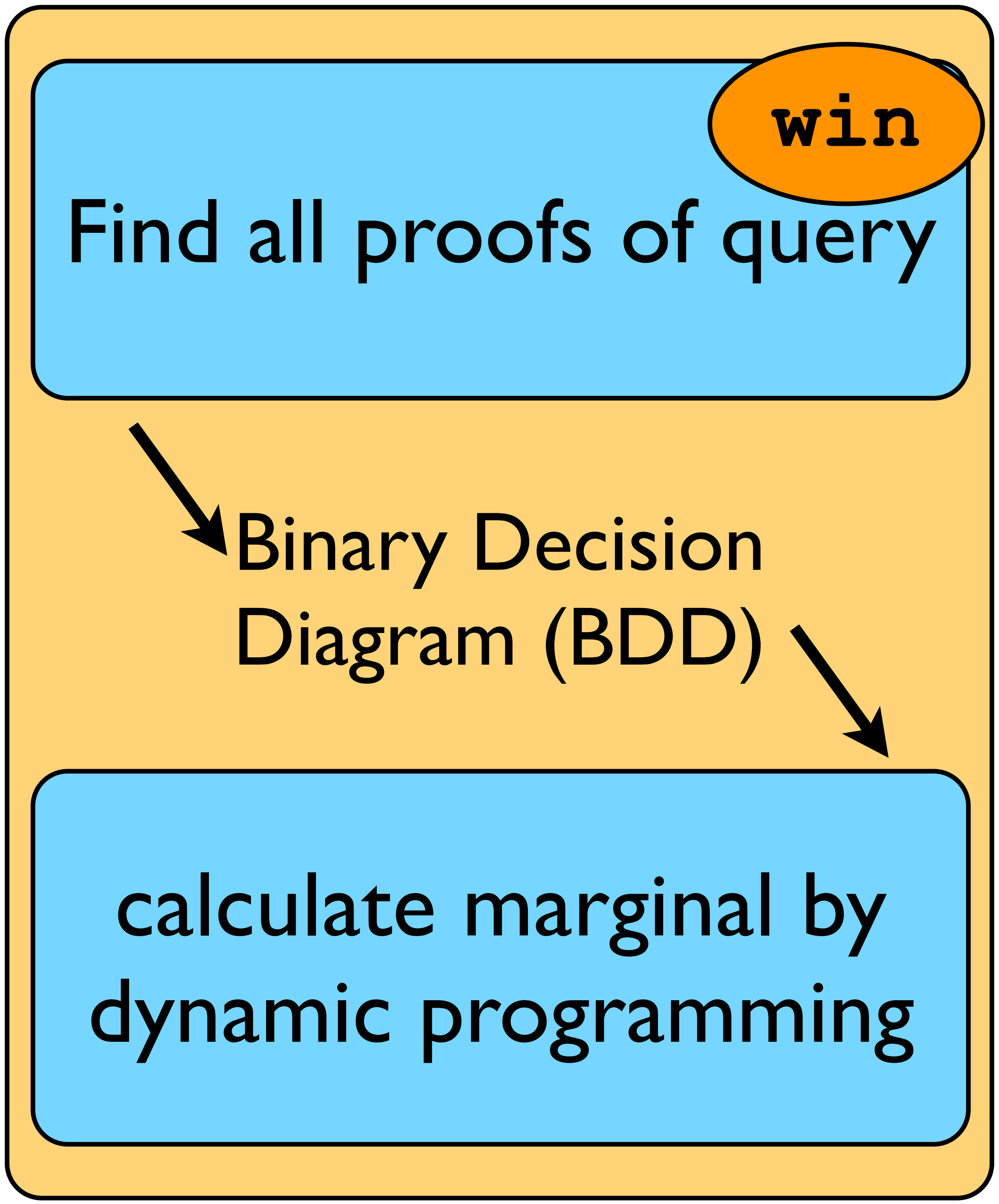


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1)

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

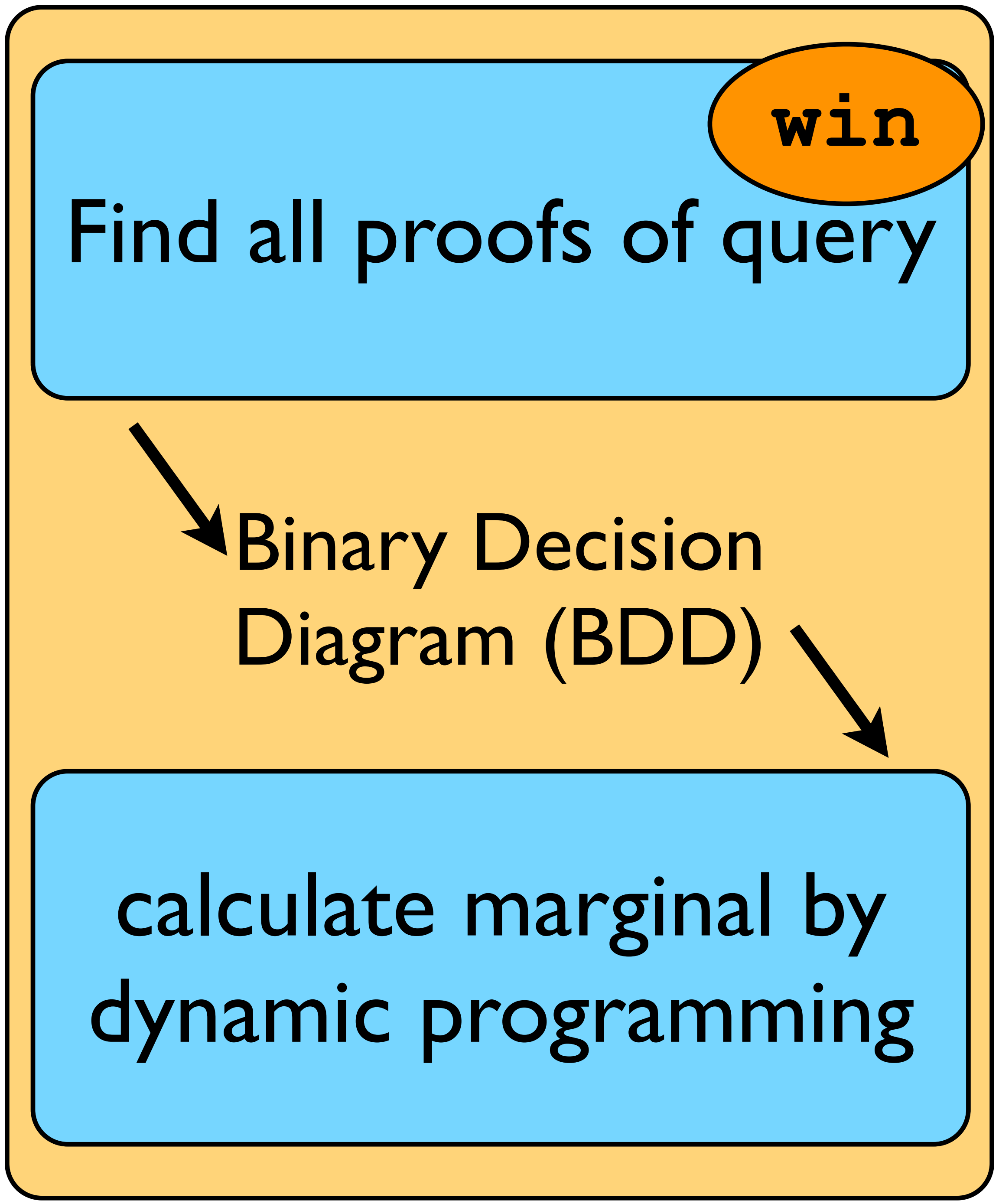



```
heads(1)
heads(2)  & heads(3)
```

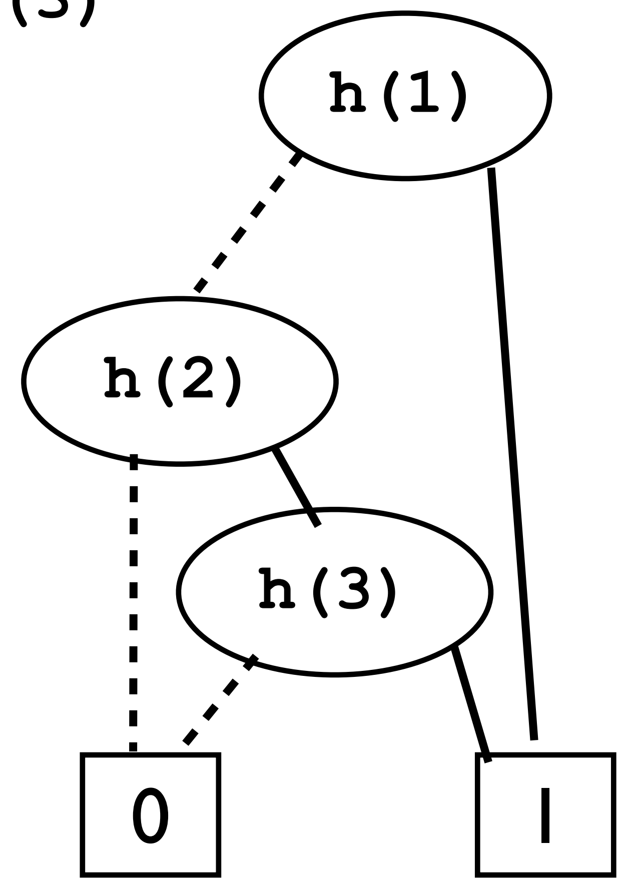

[De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach
(ProbLog1)

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



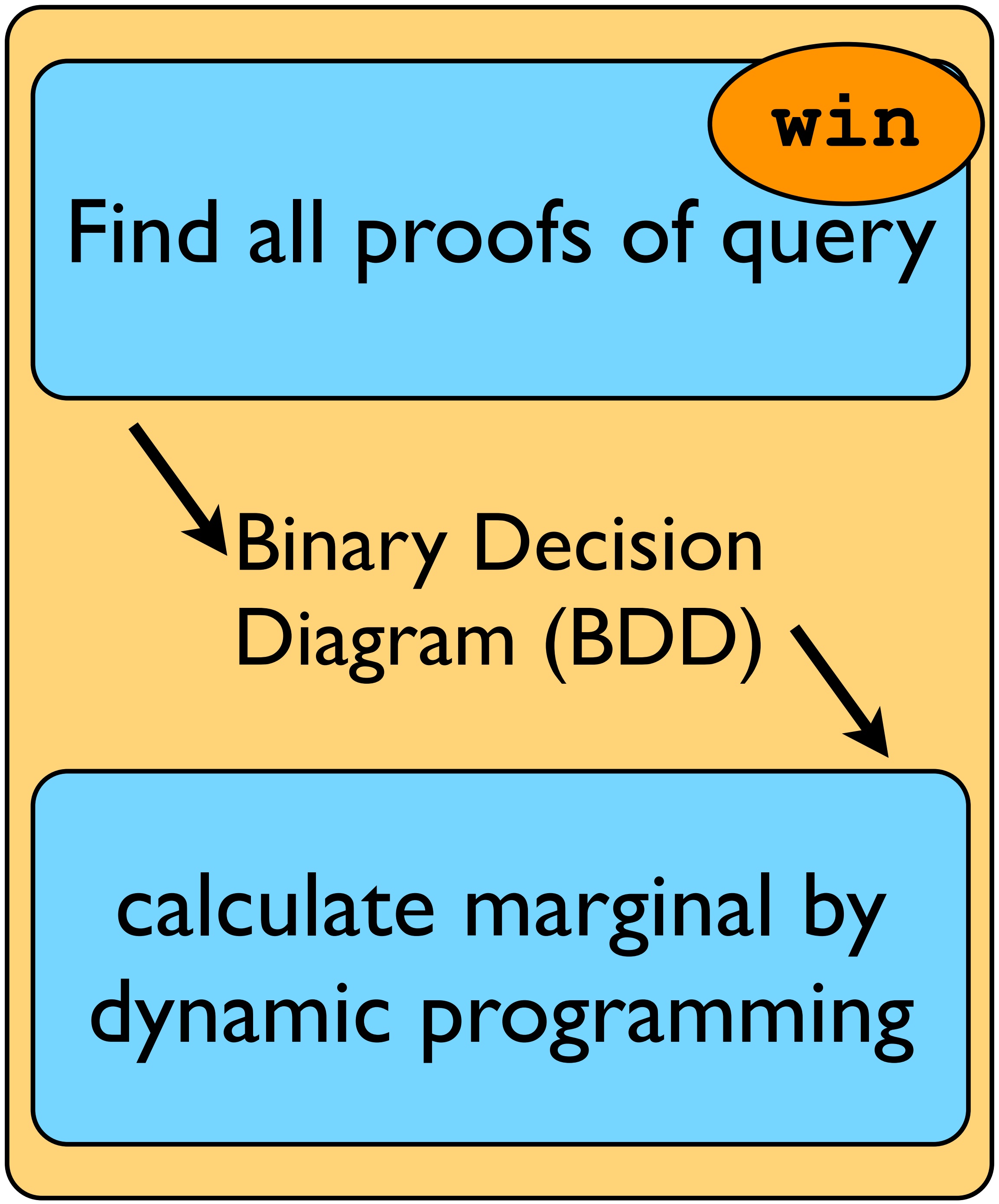

```
heads(1)
heads(2) & heads(3)
```



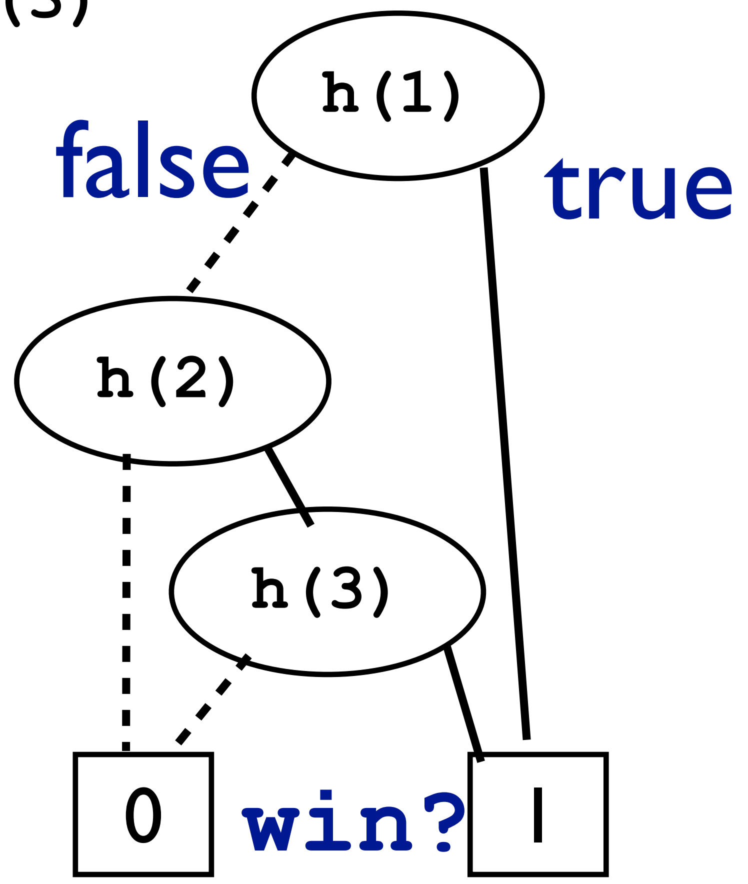
 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

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).
```

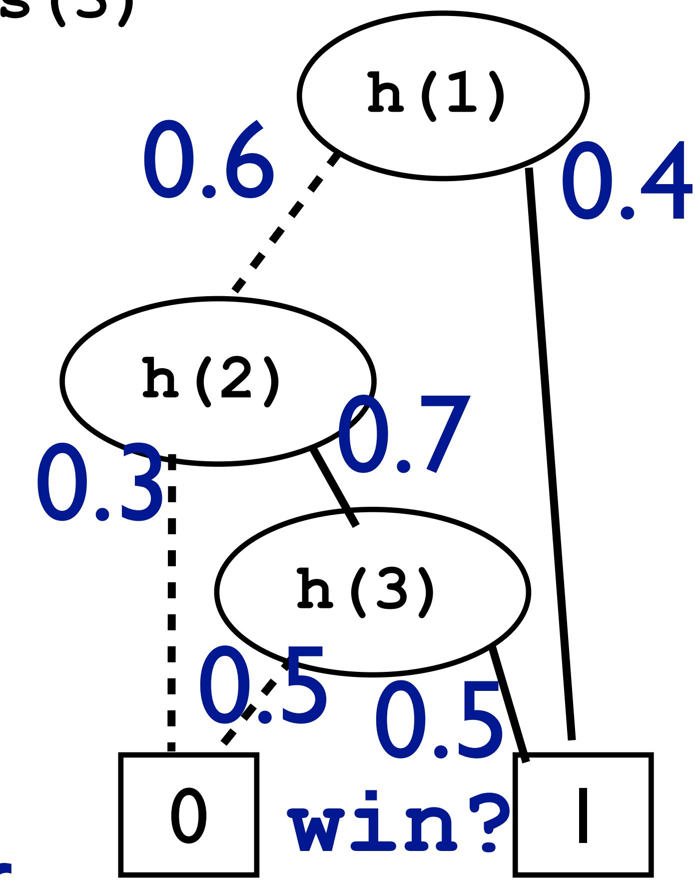
win

Find all proofs of query

→ Binary Decision Diagram (BDD) →

calculate marginal by dynamic programming

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

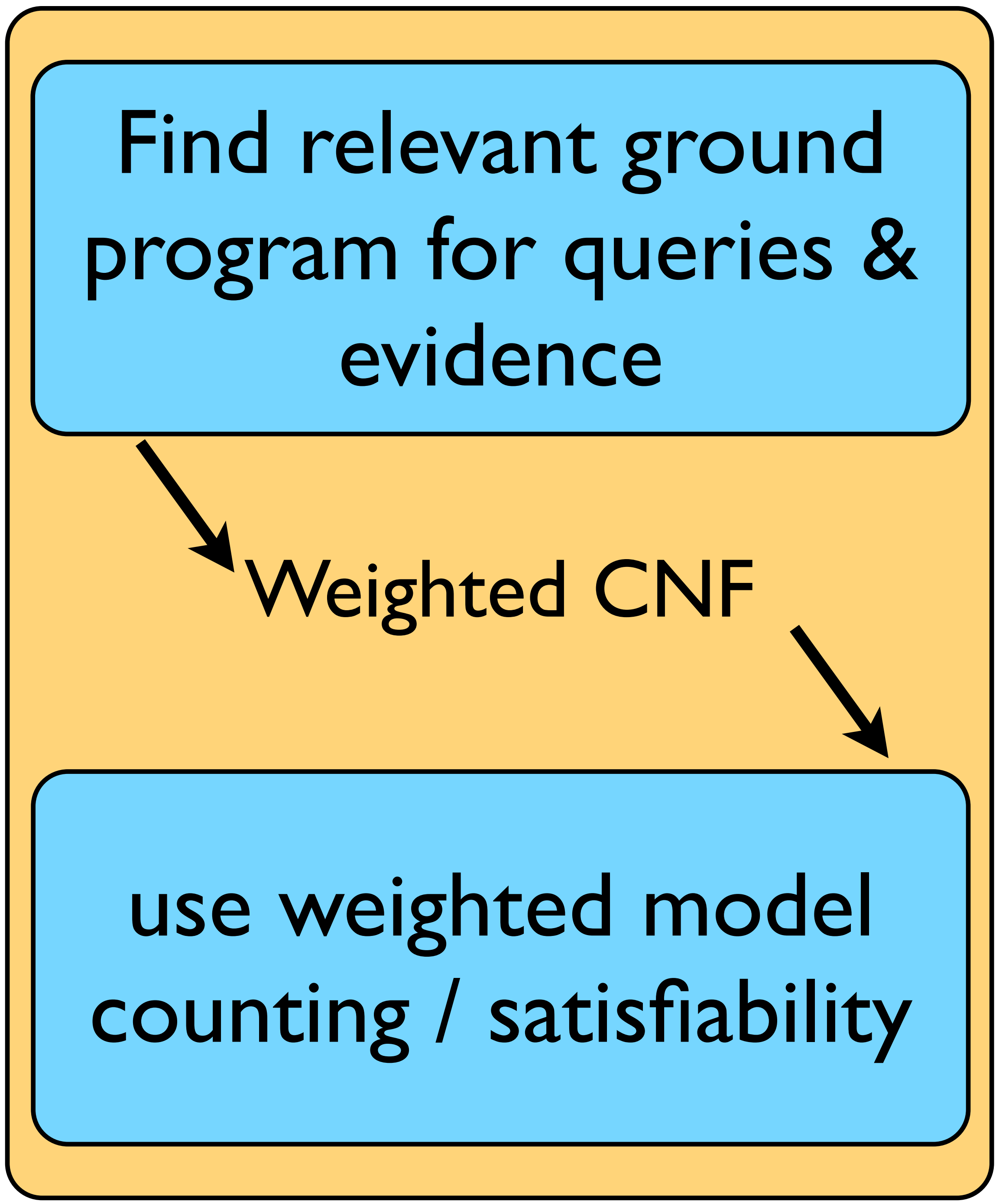


P(**win**) = probability of reaching 1-leaf

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

Current Approach

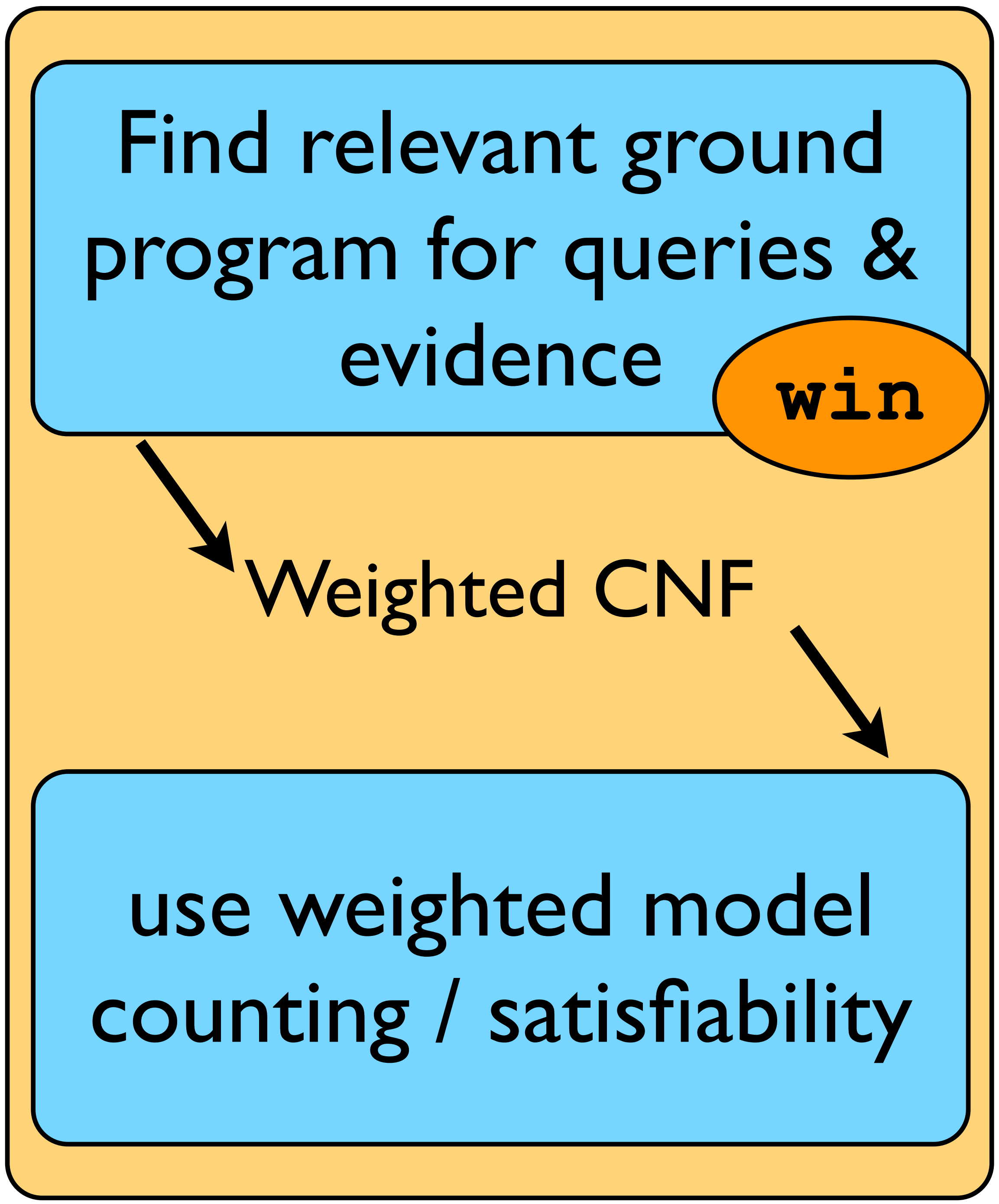
(ProbLog2)



[Fierens et al, TPLP 13]

Current Approach
(ProbLog2)

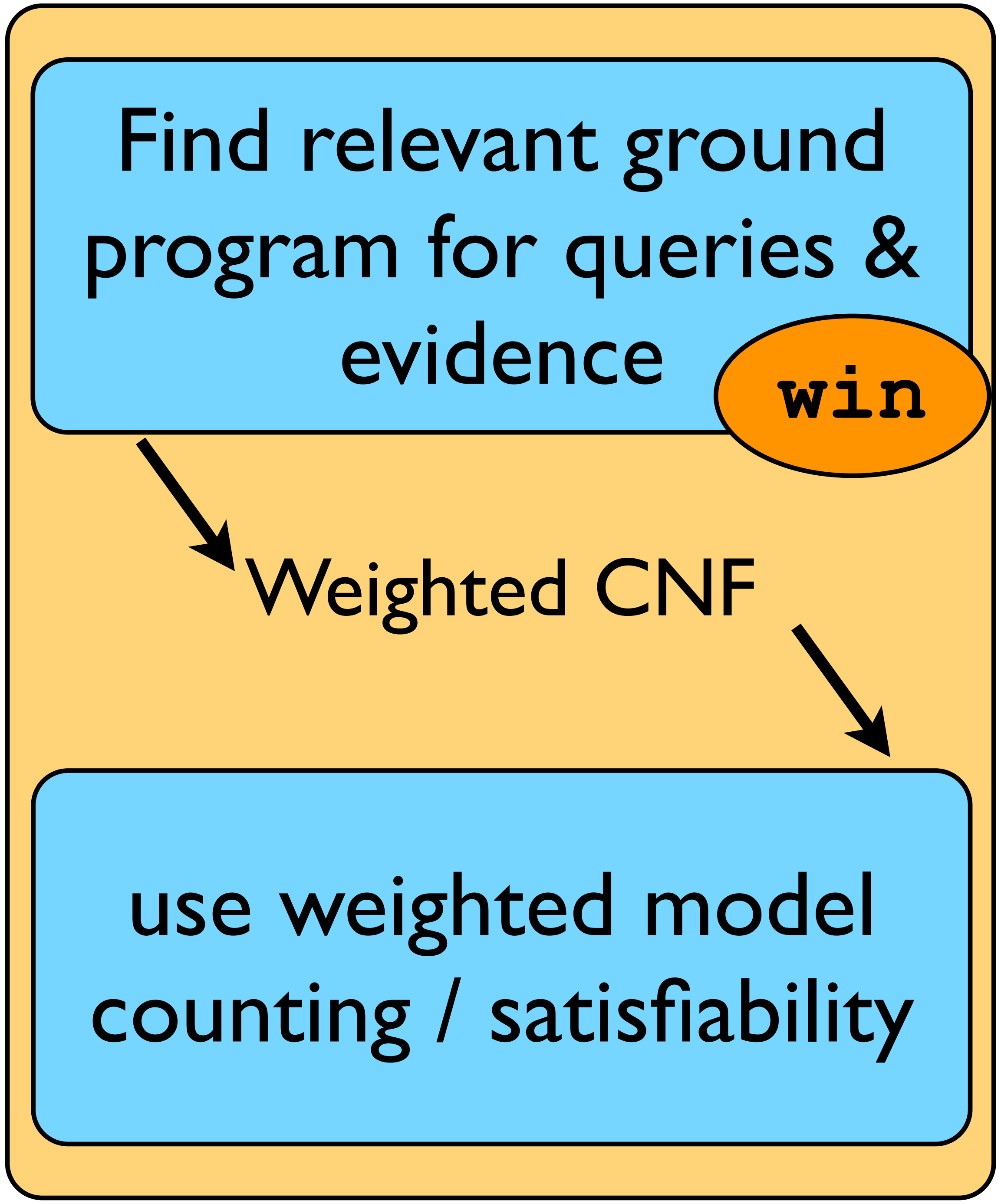
```
0.4::heads(1).
0.7::heads(2).
0.5::heads(3).
win :- heads(1).
win :- heads(2),
             heads(3).
```



[Fierens et al, TPLP 13]

Current Approach
(ProbLog2)

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

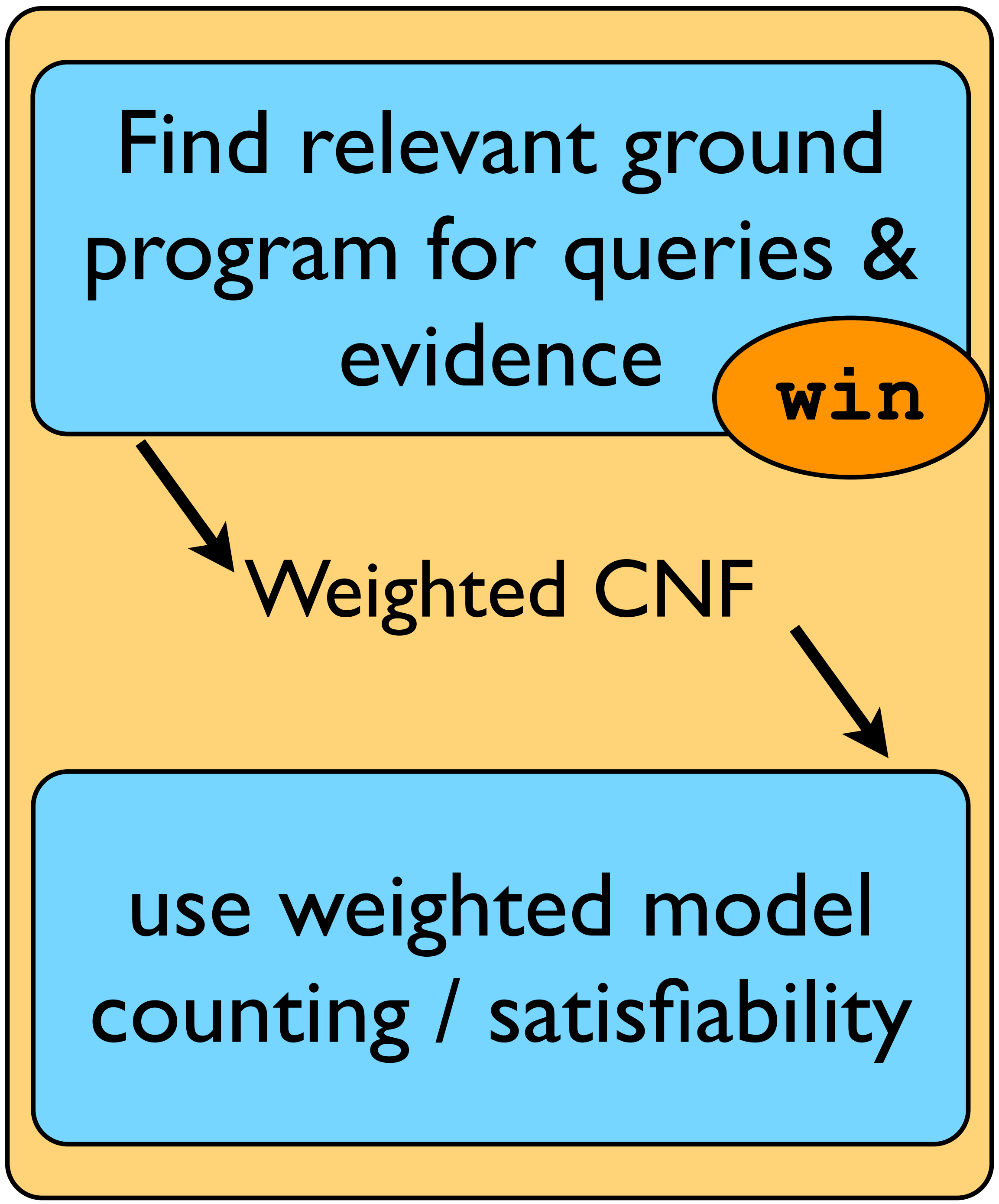



```
win :- heads(1).
win :- heads(2), heads(3).
```


[Fierens et al, TPLP 13]

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).
```

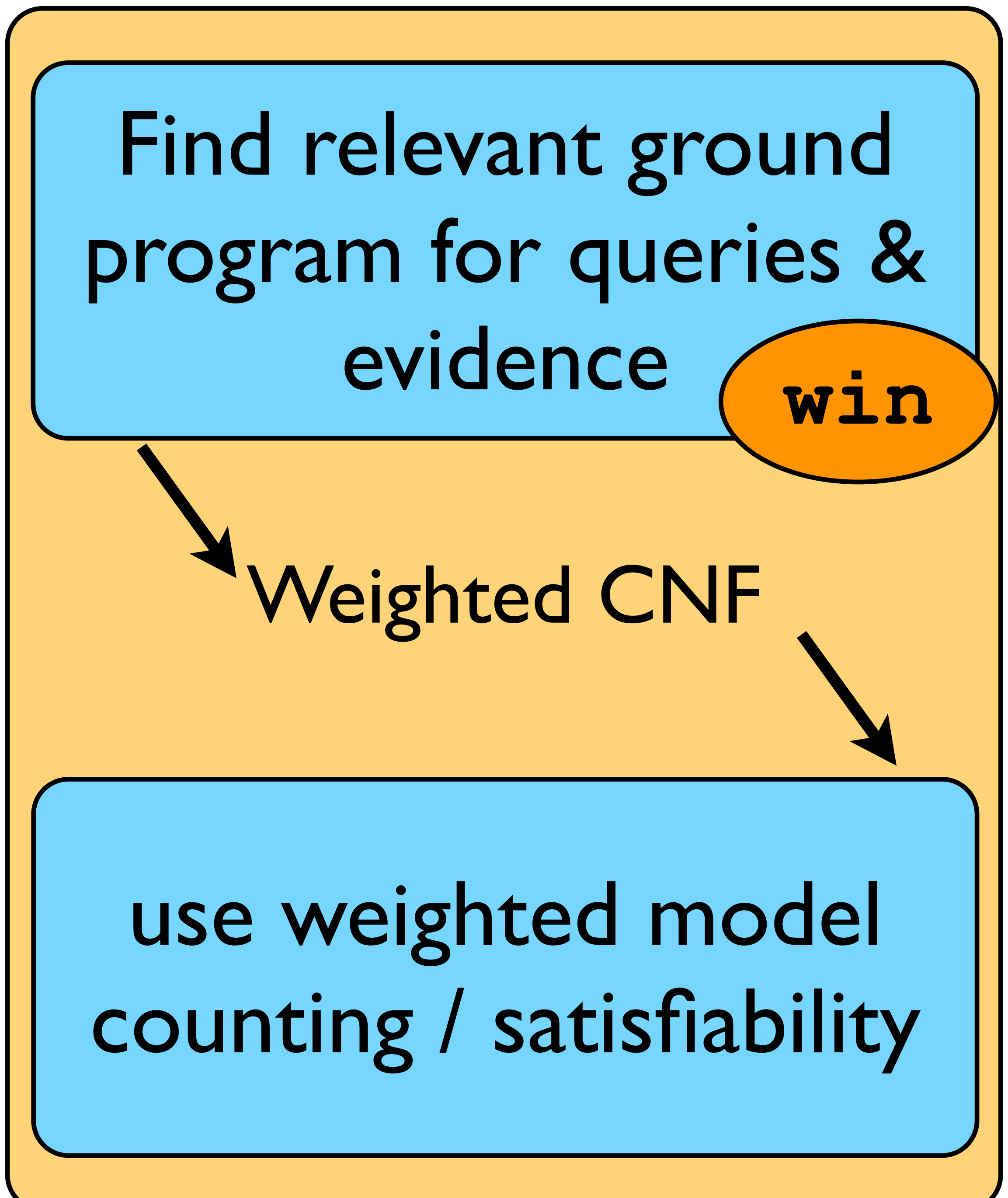
↓

$$\text{win} \leftrightarrow h(1) \lor (h(2) \land h(3))$$

[Fierens et al, TPLP 13]

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).
```

↓

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

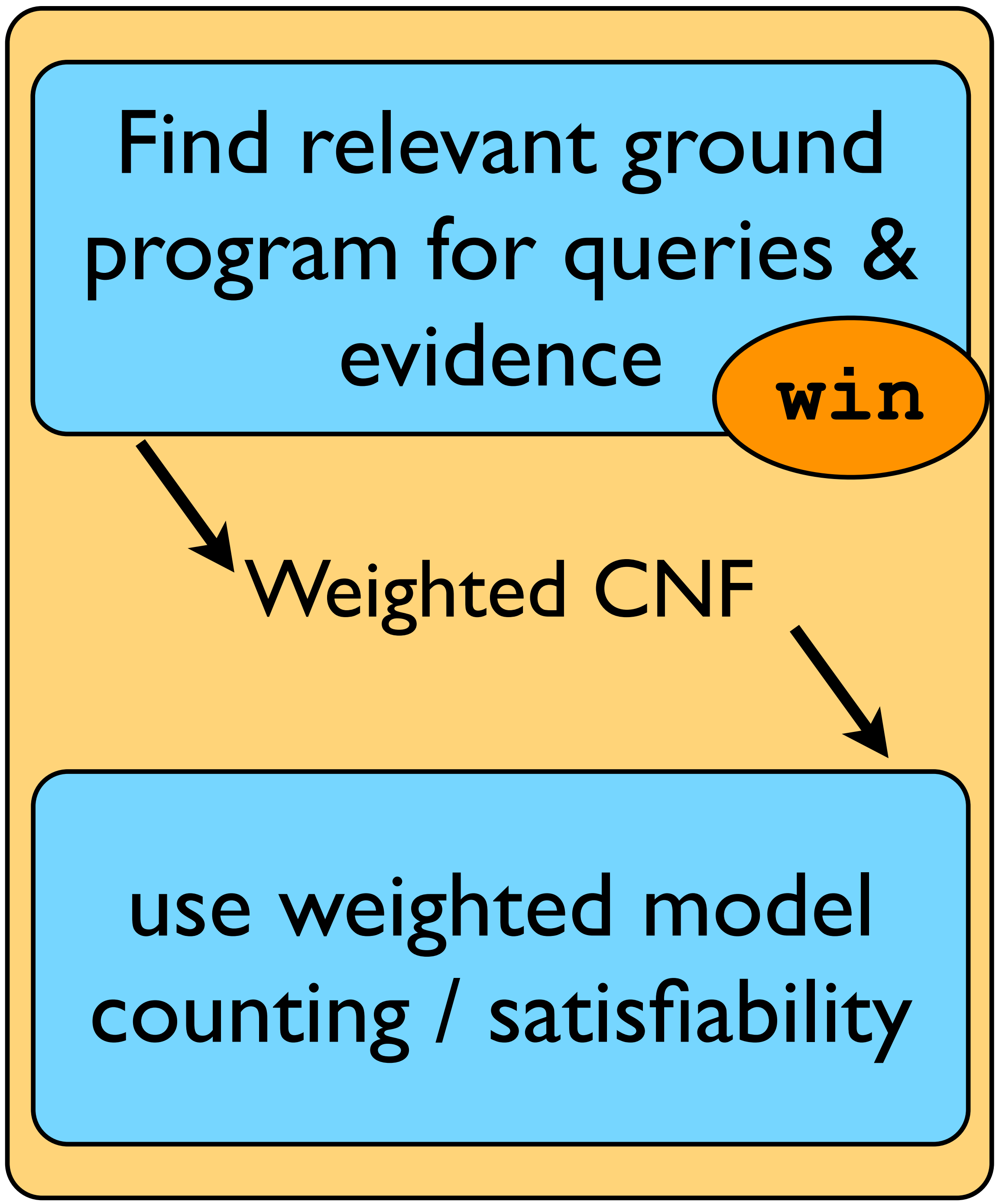
↓

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

[Fierens et al, TPLP 13]

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).
```

↓

$$\text{win} \leftrightarrow h(1) \lor (h(2) \land h(3))$$

↓

$$(\neg\text{win} \lor h(1) \lor h(2)) \land (\neg\text{win} \lor h(1) \lor h(3)) \land (\text{win} \lor \neg h(1)) \land (\text{win} \lor \neg h(2) \lor \neg h(3))$$

$h(1) \to 0.4$ $h(2) \to 0.7$ $h(3) \to 0.5$

$\neg h(1) \to 0.6$ $\neg h(2) \to 0.3$ $\neg h(3) \to 0.5$

[Fierens et al, TPLP 13]

Current Approach

(ProbLog2)

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

Find relevant ground program for queries & evidence **win**

↓

Weighted CNF

↓

use weighted model counting / satisfiability

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

↓

$$win \leftrightarrow h(1) \lor (h(2) \land h(3))$$

↓

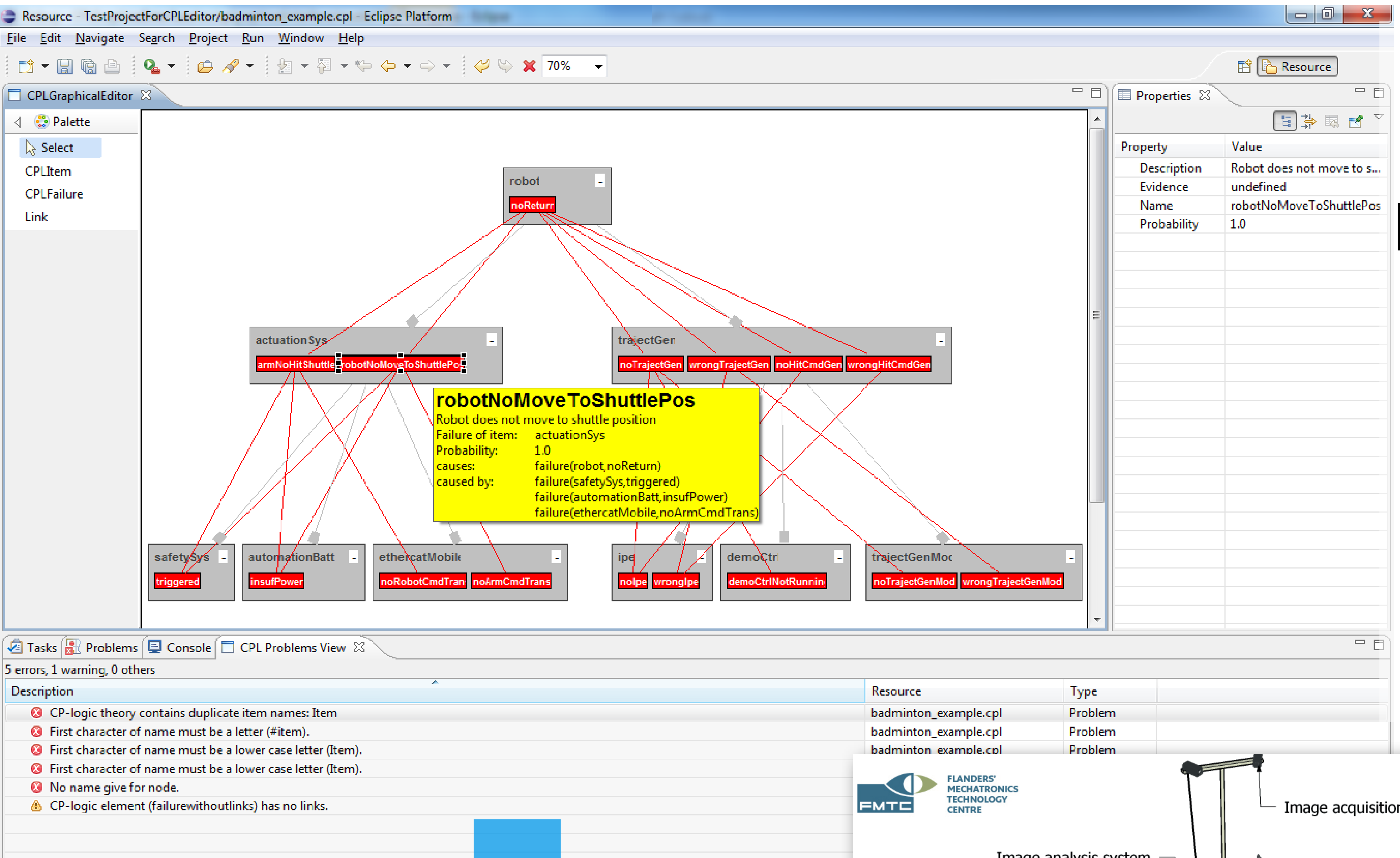
$$(\neg win \lor h(1) \lor h(2)) \land (\neg win \lor h(1) \lor h(3)) \land (win \lor \neg h(1)) \land (win \lor \neg h(2) \lor \neg h(3))$$

use standard tool

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

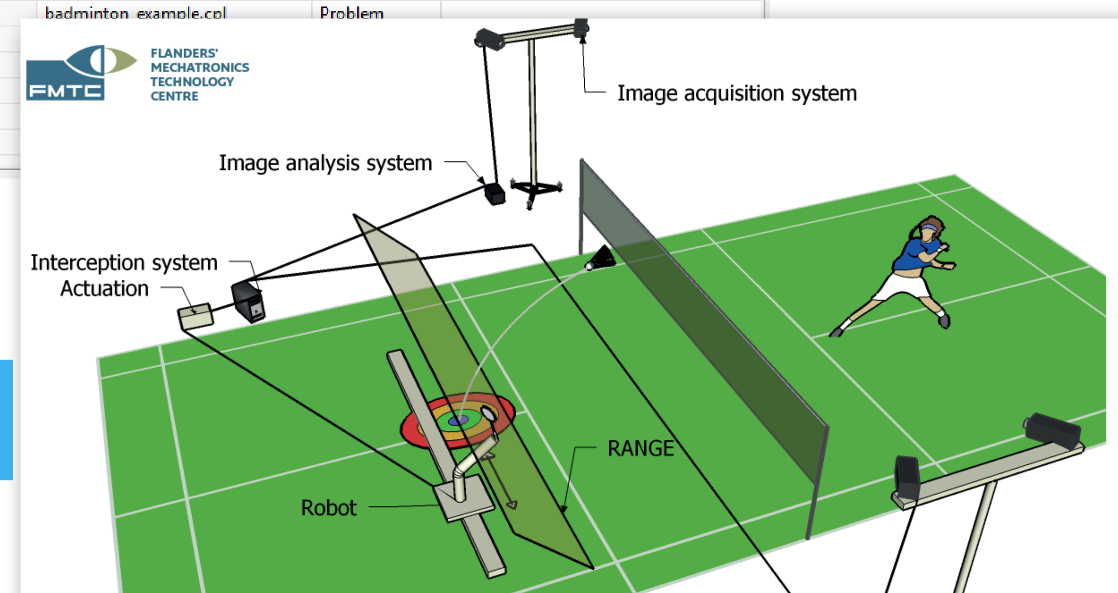
[Fierens et al, TPLP 13]

Diagnostics for Prognostics



Visual representation of a ProbLog diagnostics program

The GUI knows how to interpret certain predicates (e.g. failure, partof)



Find most probable reason of a failure given a set of sensor measurements

[Schramm, Meert & Driessens]

ProbLog for activity recognition from video

CAVIAR-INRIA human activity dataset

28 videos
≈ 26.500 frames

- Separation between low-level events (LLE) and high-level events (HLE)
  - LLE: *walking, running, active, inactive, abrupt*
  - HLE: *meeting, moving, fighting, leaving_object*
- Probabilistic Logic approach: *Event Calculus in ProbLog* (Prob-EC) to infer the high-level events from an **algebra** of low-level events.
- Example:

$$
\begin{gathered}
initiatedAt(fighting(P_1, P_2) = true, T) \leftarrow \\
happensAt(abrupt(P_1), T), \\
holdsAt(close(P_1, P_2, 44) = true, T), \\
not\ happensAt(inactive(P_2), T).
\end{gathered}
$$

[Skarlatidis et al, TPLP 13]

Parameter Learning

e.g., webpage classification model

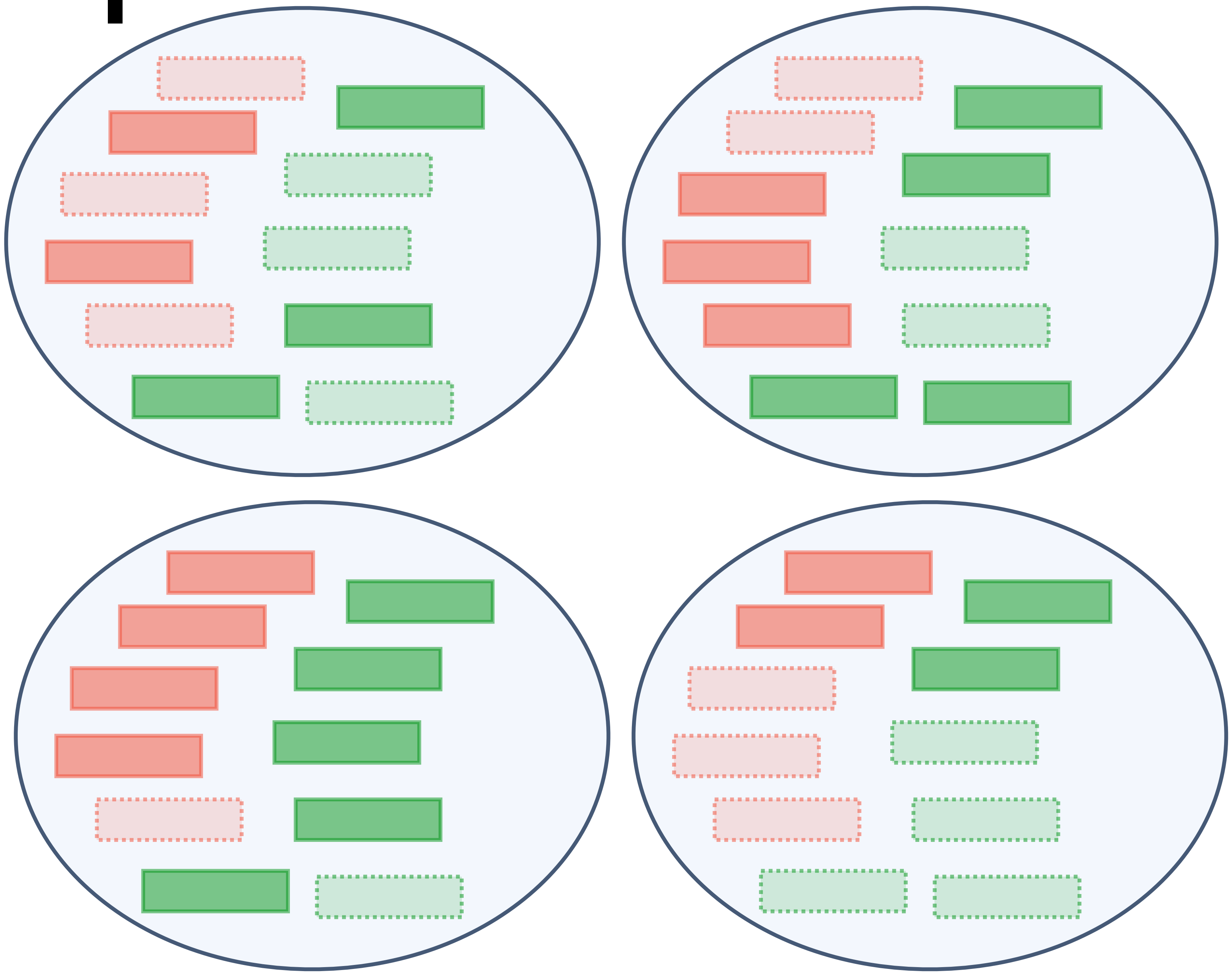
for each ***CLASS1*, *CLASS2*** and each ***WORD***

```
?? :: link_class(Source,Target,CLASS1,CLASS2).
?? :: word_class(WORD,CLASS).
```

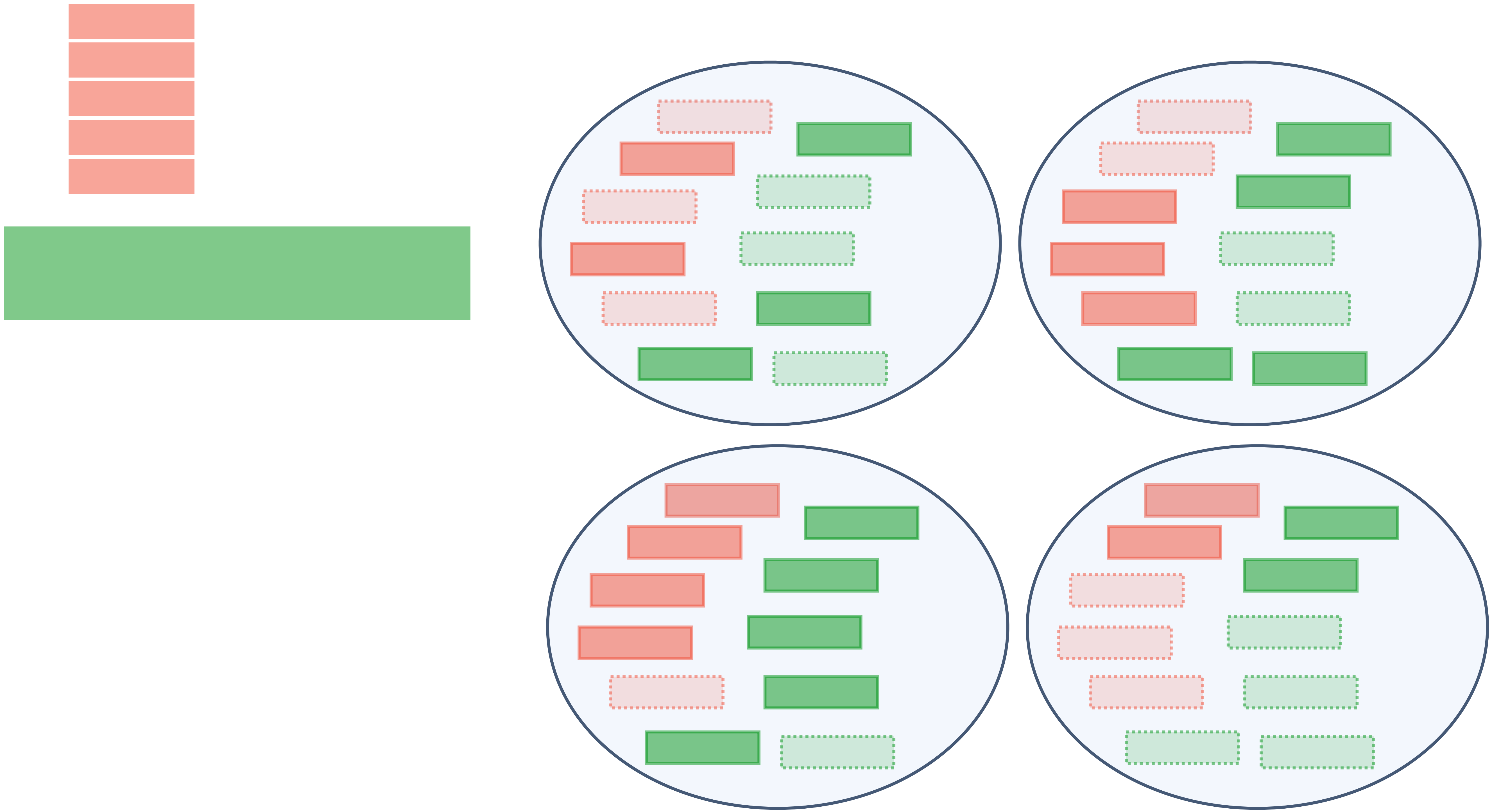
```
class(Page,C)   :-  has_word(Page,W), word_class(W,C).

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

Sampling Interpretations

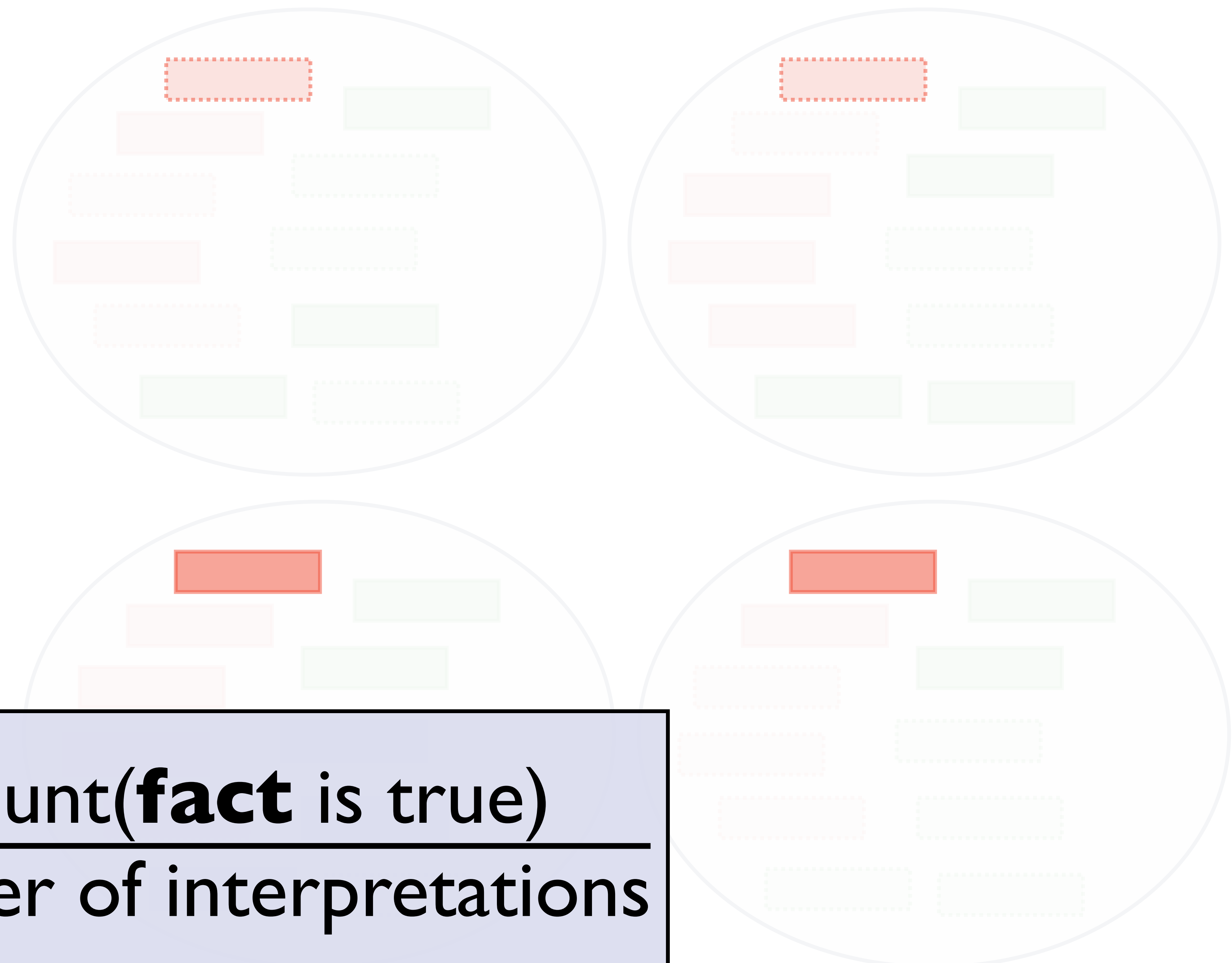
Sampling Interpretations



Parameter Estimation

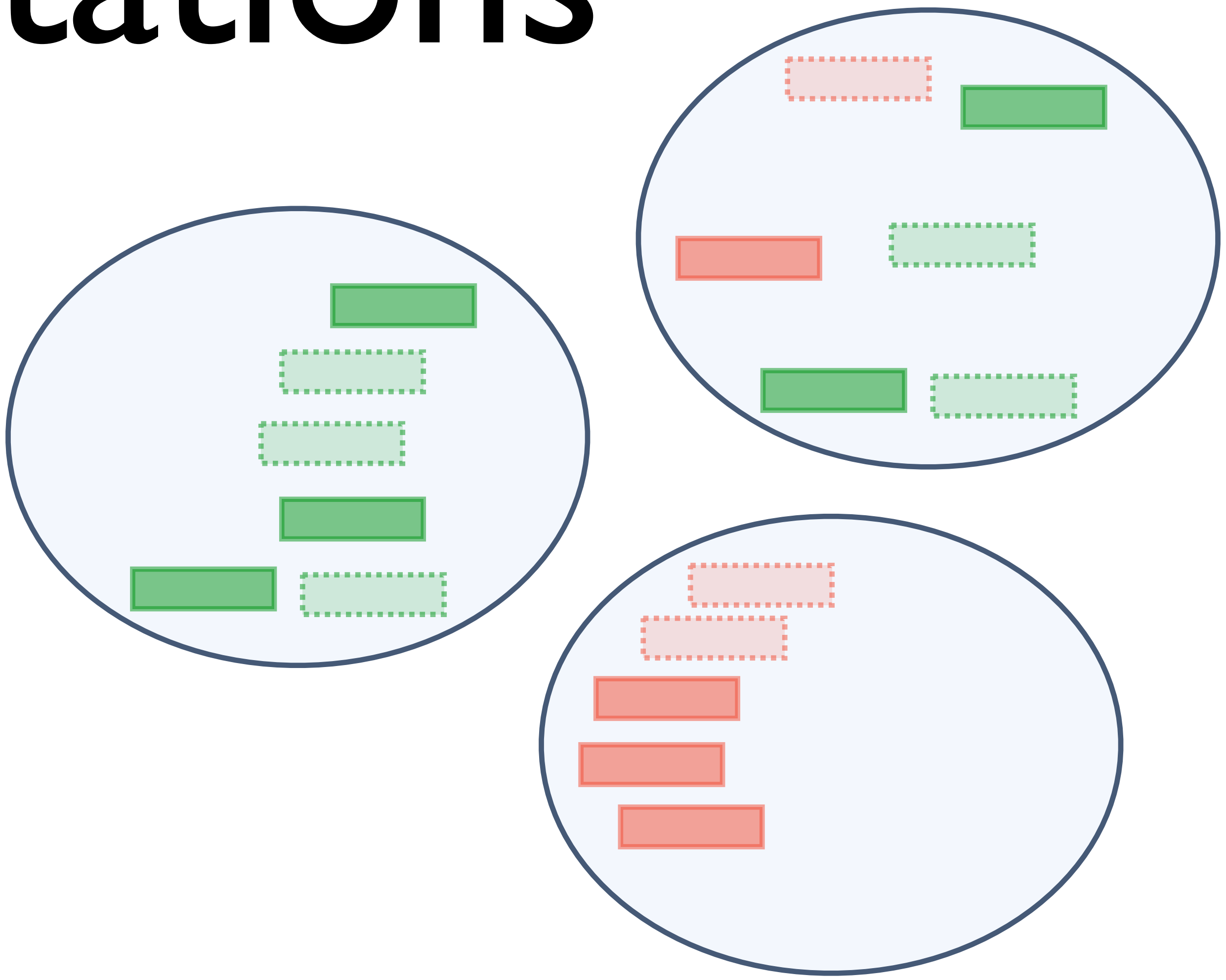


Parameter Estimation



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

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 13]

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

Upgrading relational learning

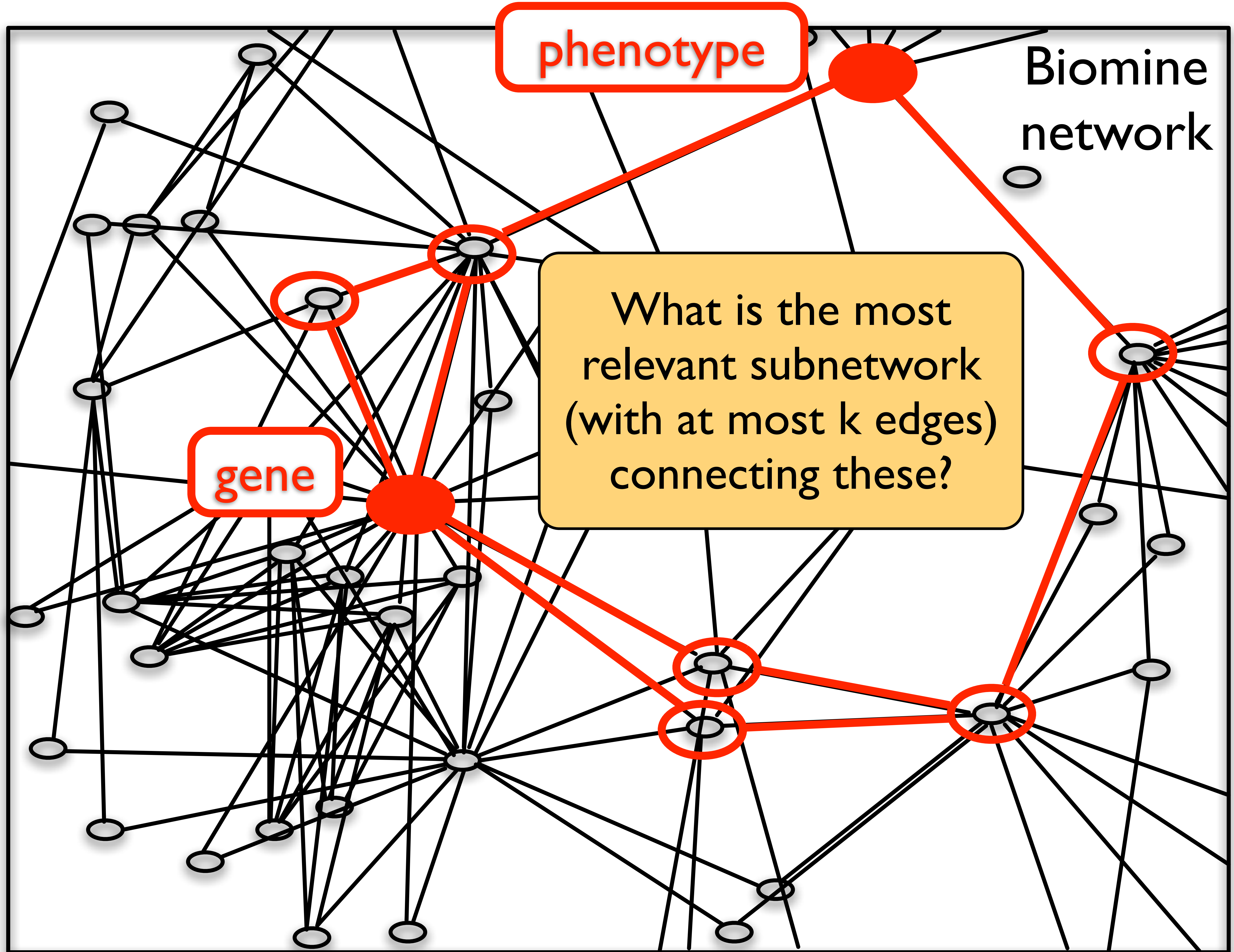
	Prolog
	
	`infl(a,b).`
	`...`
		ProbLog
		
		`0.4::infl(a,b).`
		`...`

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

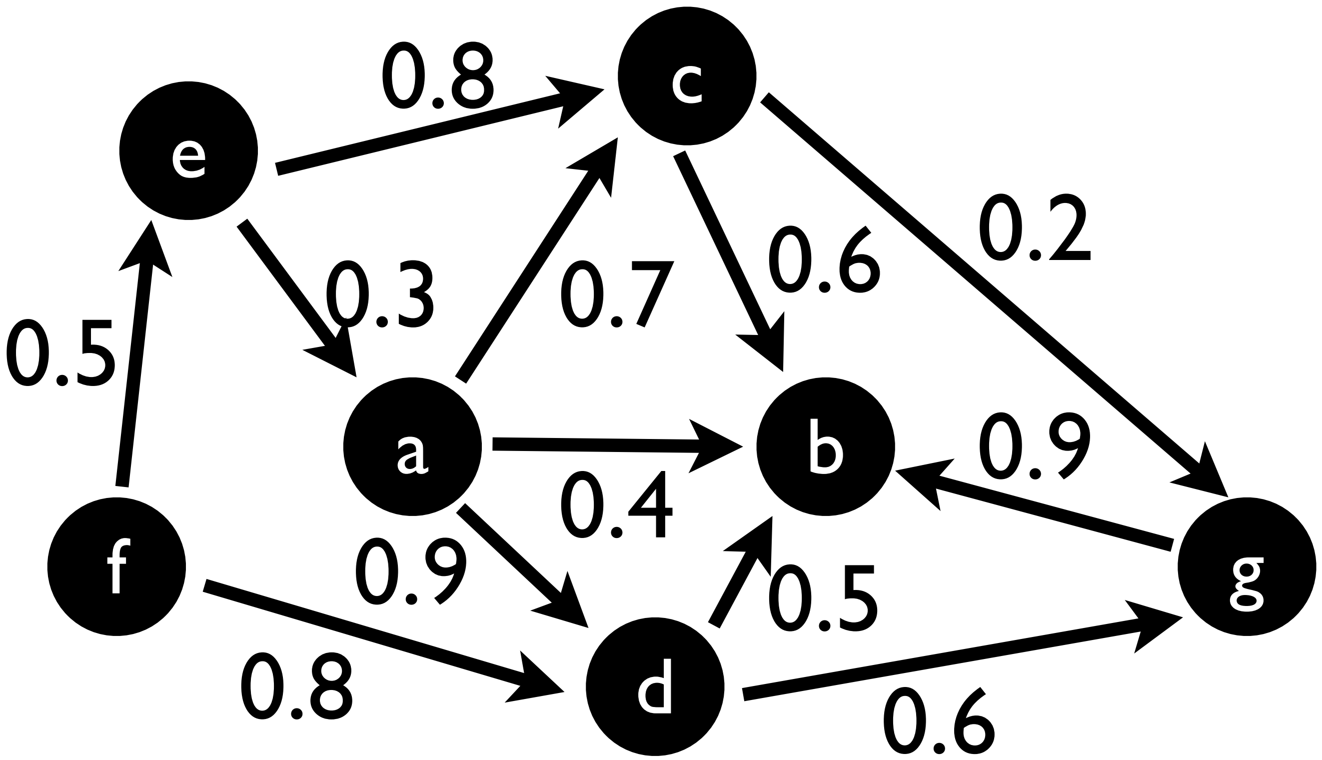
Upgrading relational learning

	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


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

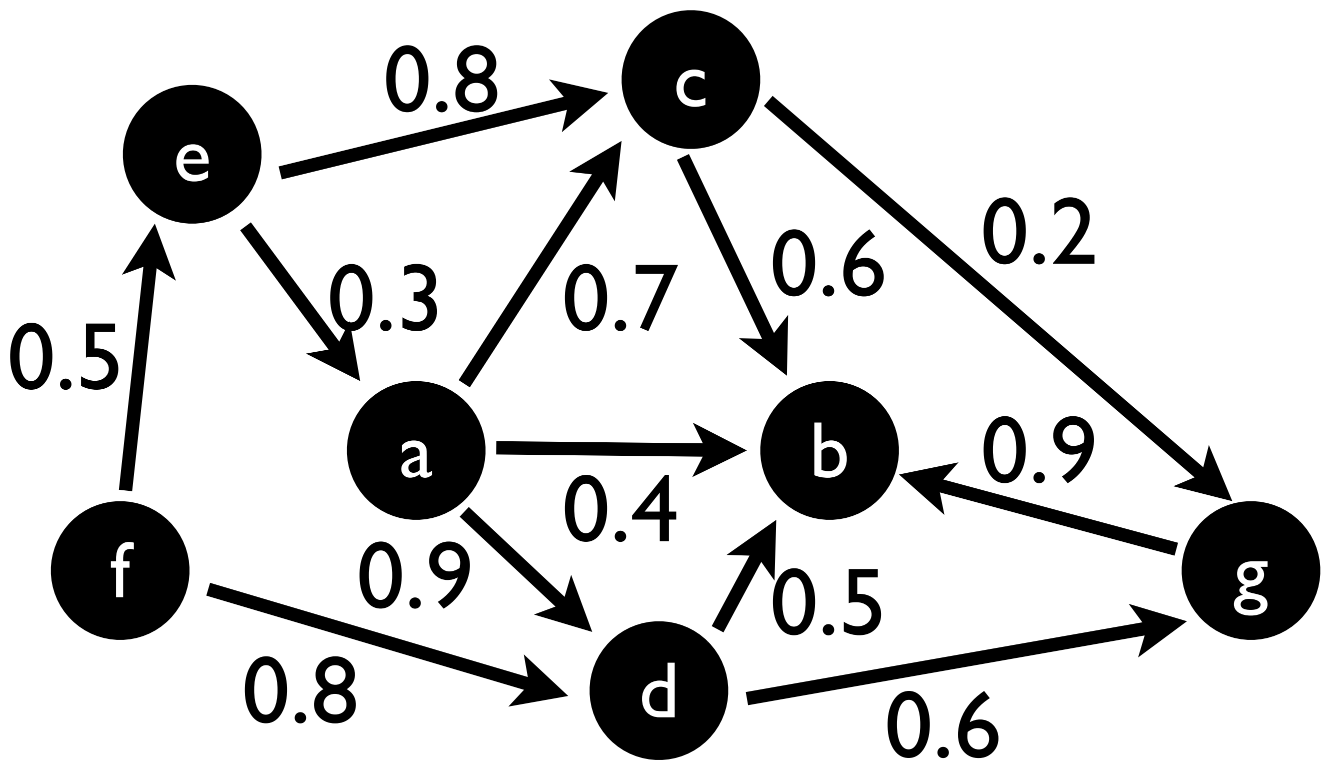
Theory compression




```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

[De Raedt et al, MLJ 08]

Theory compression

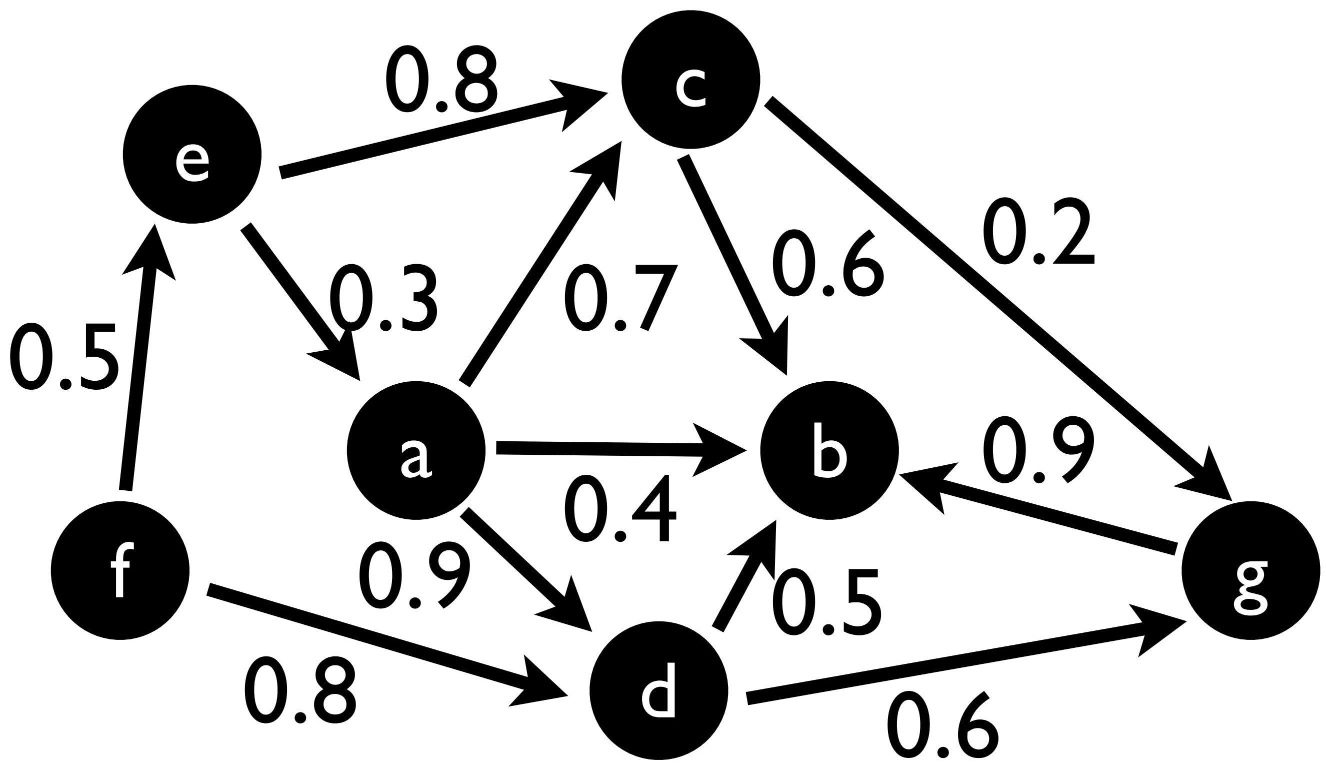



```
path(X,Y)  :- edge(X,Y).
path(X,Y)  :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

[De Raedt et al, MLJ 08]

Theory compression



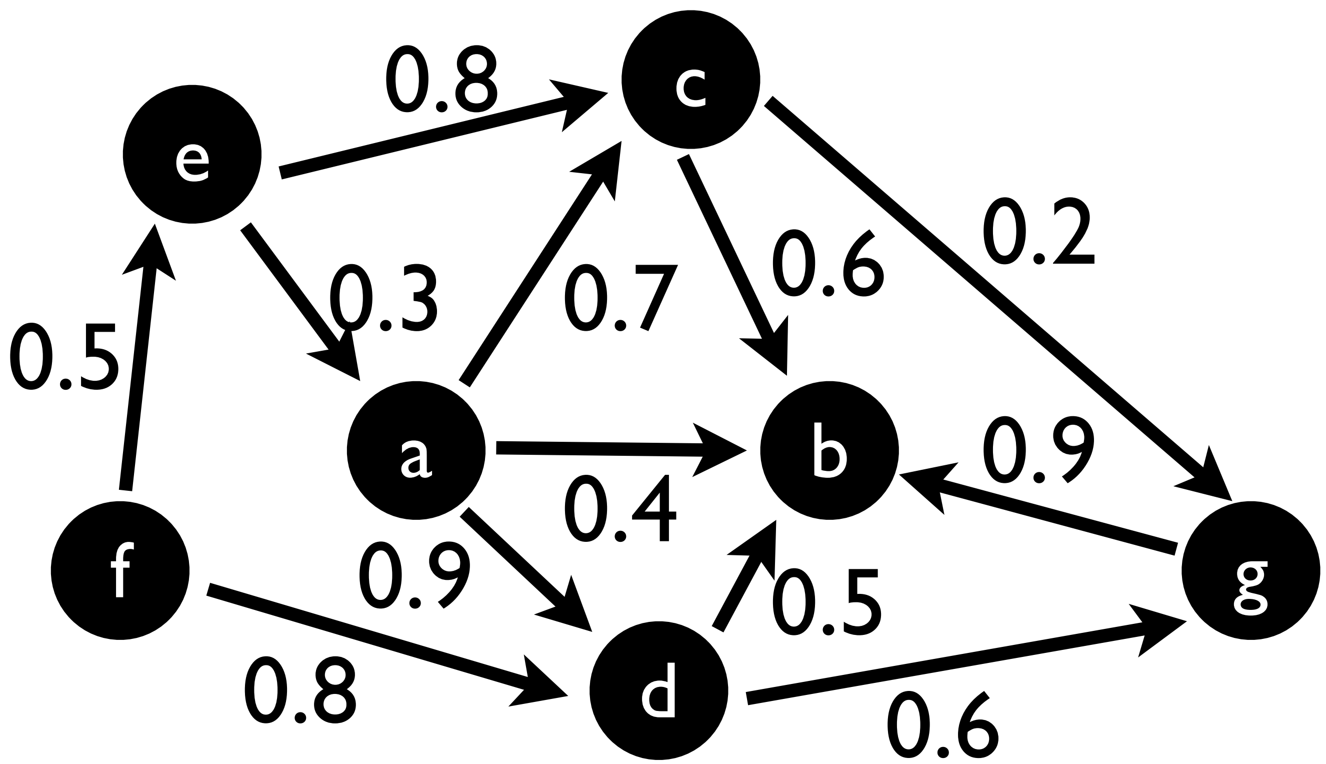

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1-P)$

[De Raedt et al, MLJ 08]

Theory compression



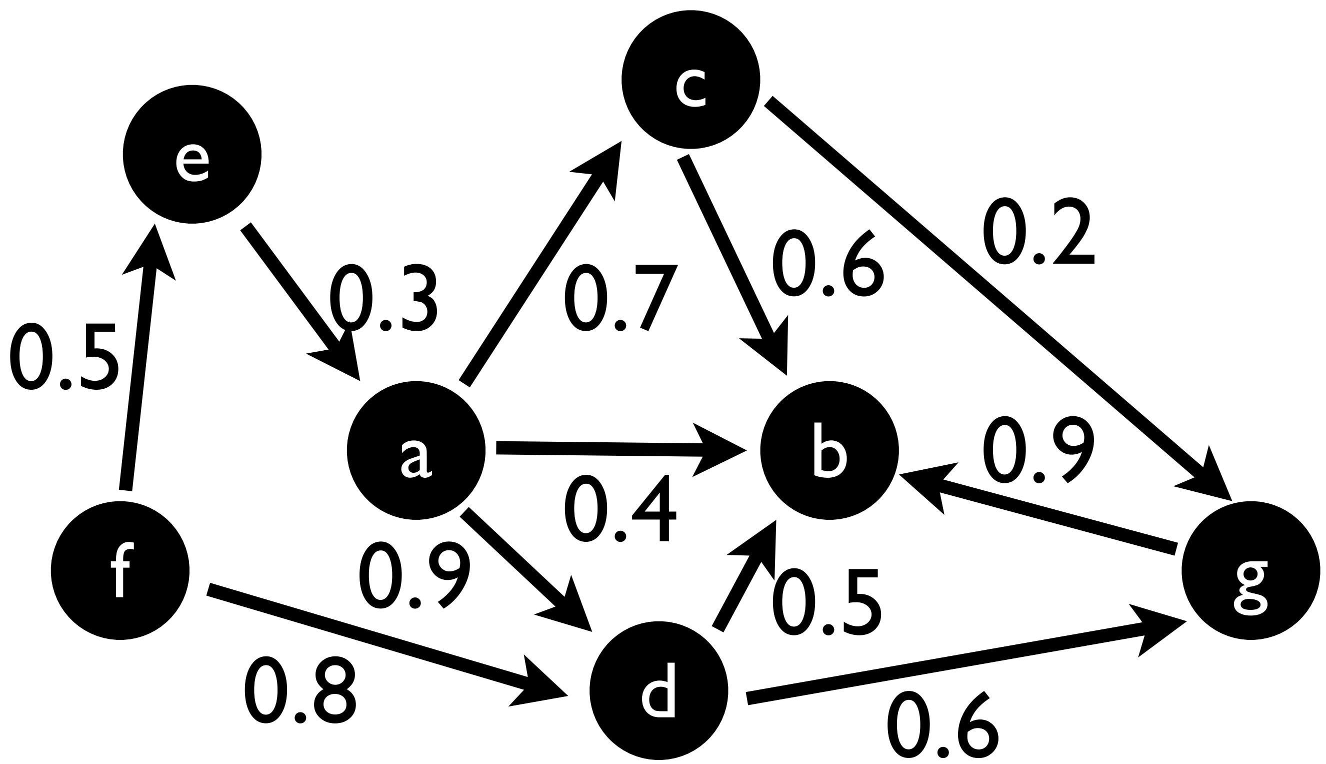

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg}(1-P)$

[De Raedt et al, MLJ 08]

Theory compression



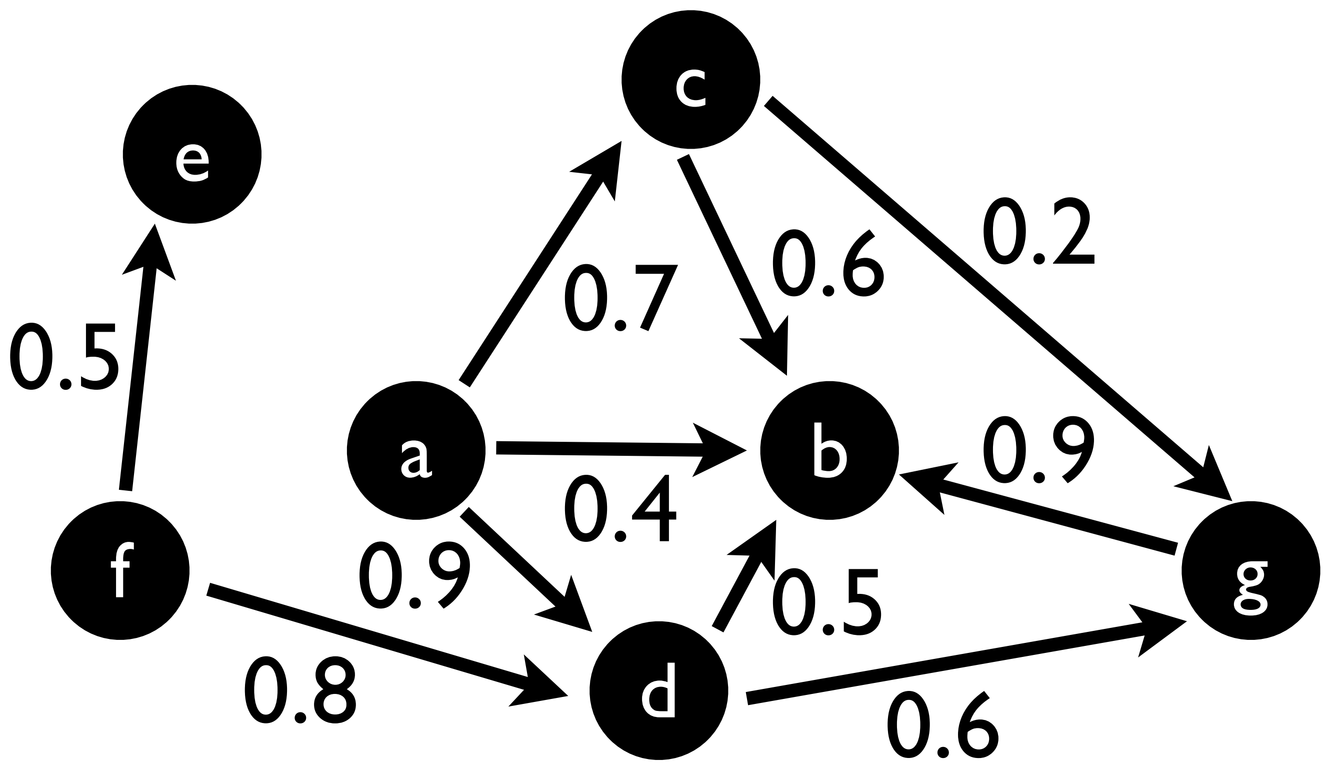

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1-P)$

[De Raedt et al, MLJ 08]

Theory compression



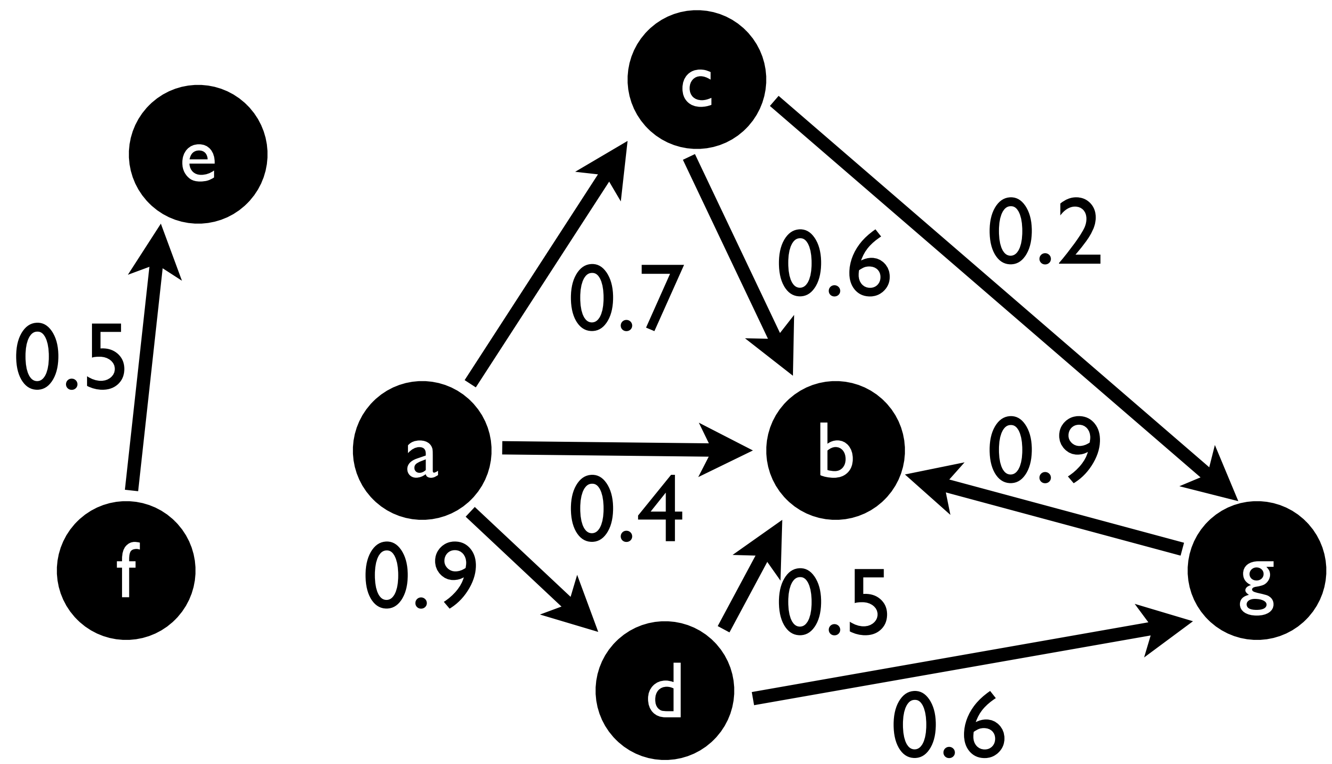

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg}(1-P)$

[De Raedt et al, MLJ 08]

Theory compression



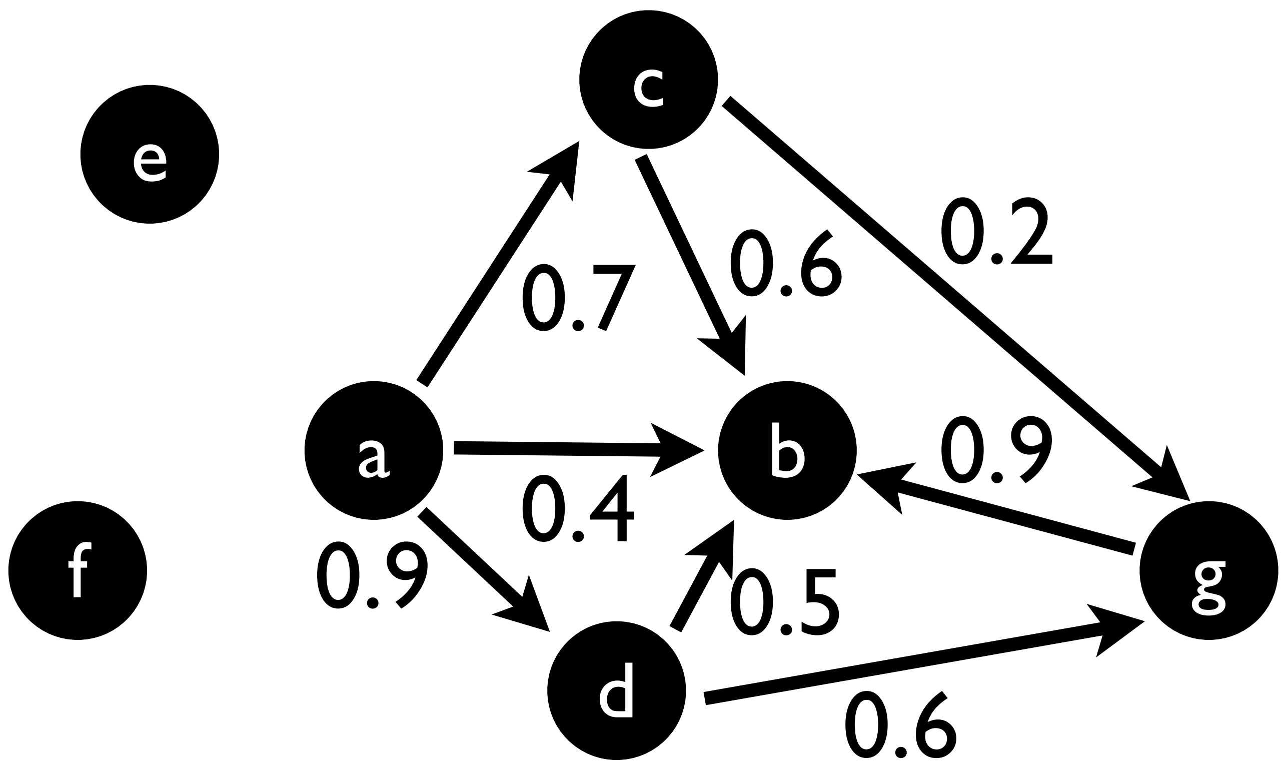

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

[De Raedt et al, MLJ 08]

Theory compression



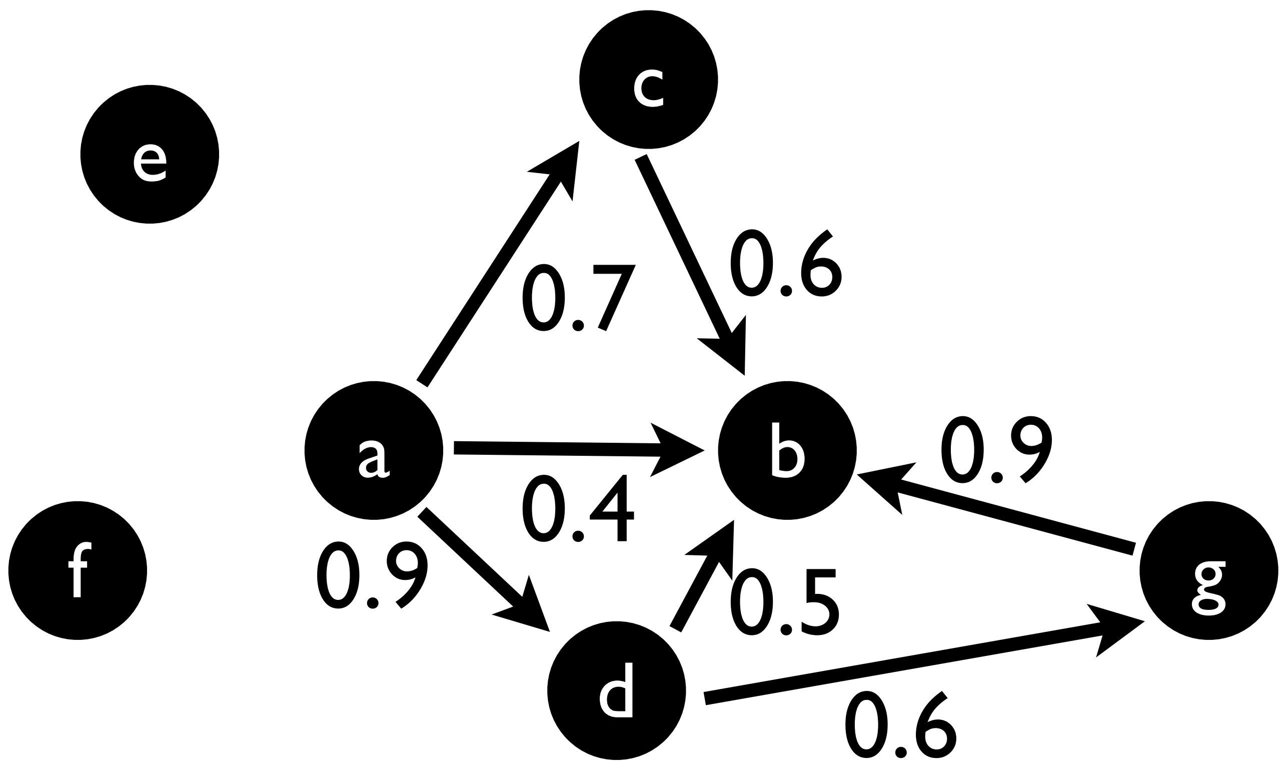

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1 - P)$

[De Raedt et al, MLJ 08]

Theory compression



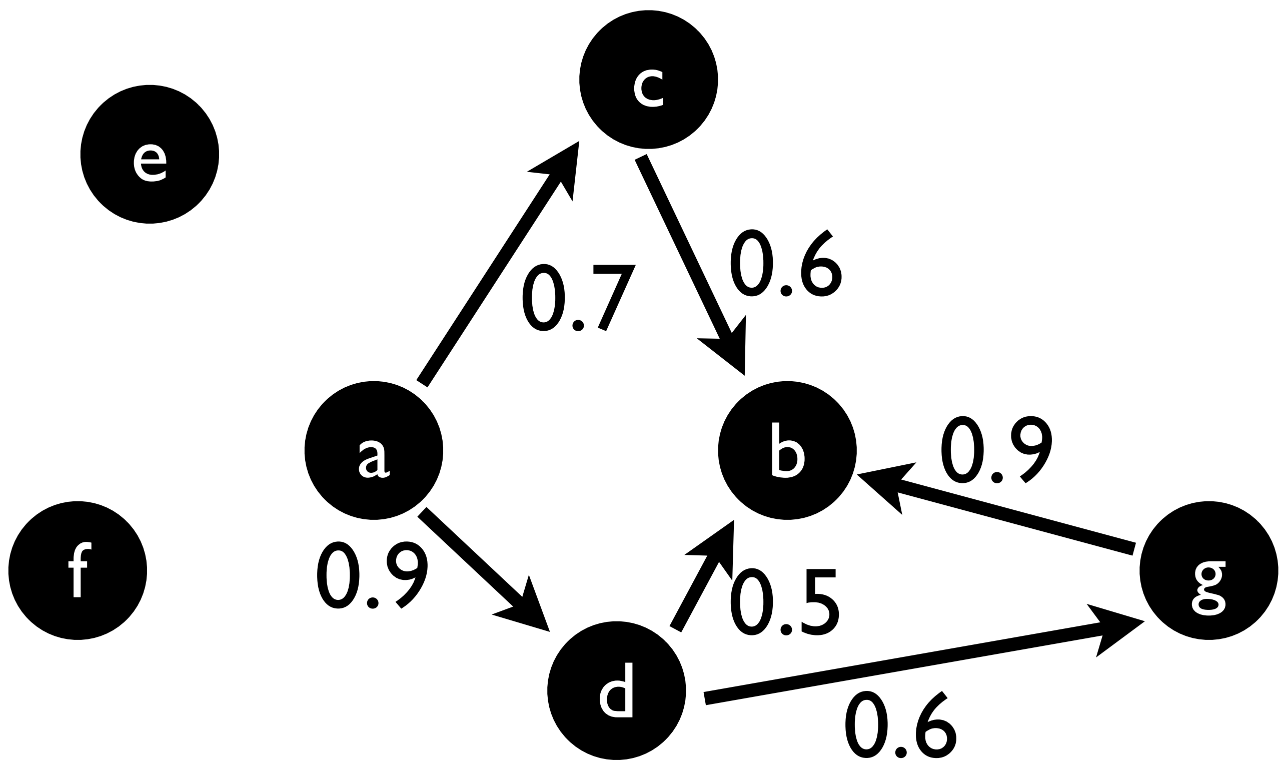

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg} (1-P)$

[De Raedt et al, MLJ 08]

Theory compression



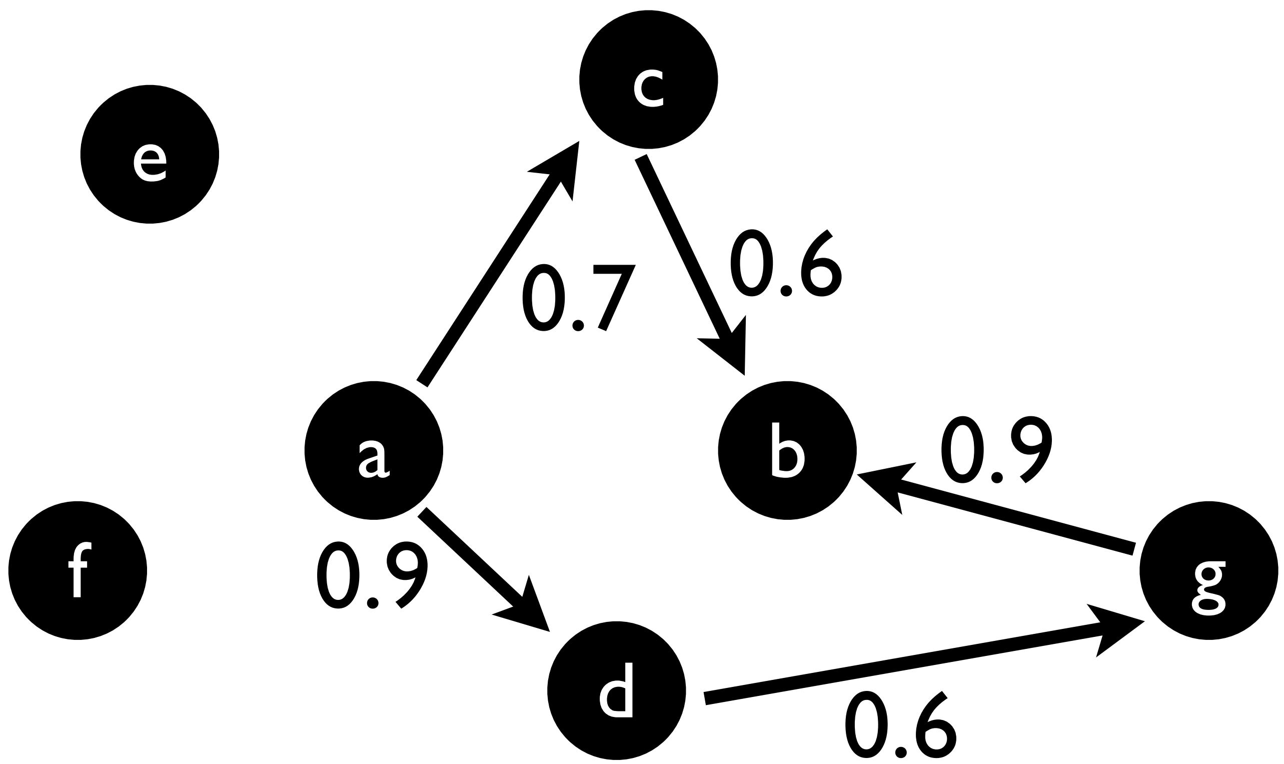

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg}(1-P)$

[De Raedt et al, MLJ 08]

Theory compression

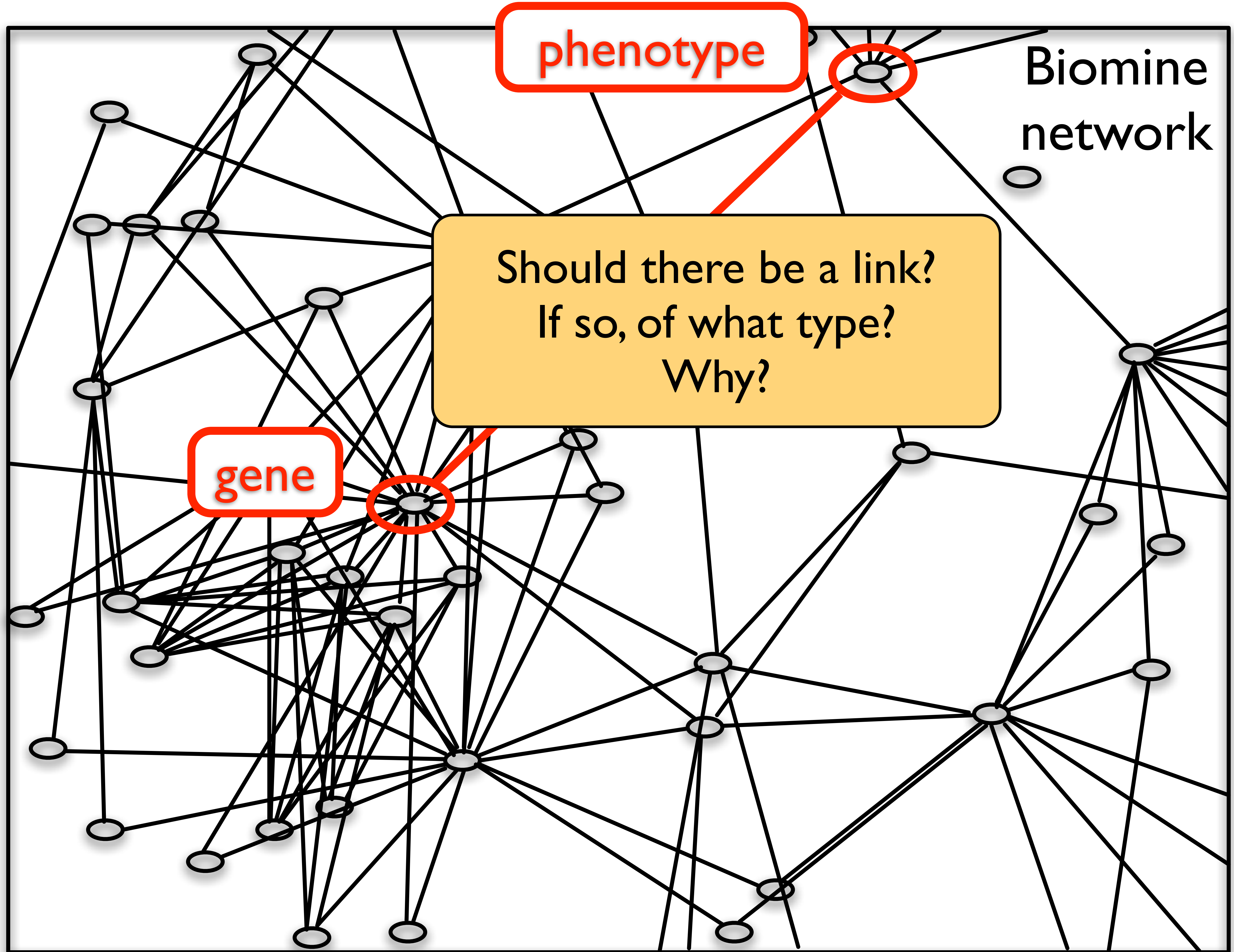



```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z), path(Z,Y).

0.8::edge(e,c).
0.3::edge(e,a).
...
```

best subnetwork of at most 5 edges where **`path(a,b)`** but not **`path(e,g)`**?

- build inference data structures for all examples
- simulate edge deletion by setting probability to 0
- greedily delete the one that maximizes $\prod_{pos} P - \prod_{neg}(1 - P)$

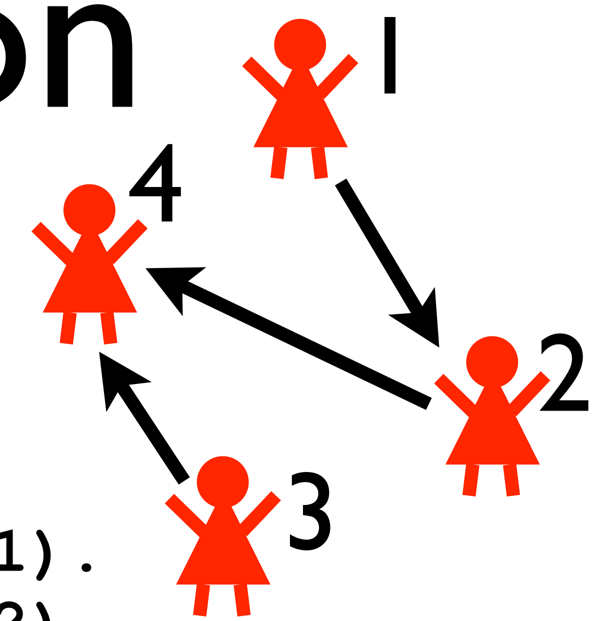
[De Raedt et al, MLJ 08]


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

Learning Patterns

- **Probabilistic Explanation Based Learning**: example + background theory → rule
- **Probabilistic Query Mining**: pos./neg. examples → set of independent rules
- **Probabilistic Rule Learning**: probabilistic examples → (probabilistic) concept definition

Probabilistic explanation based learning

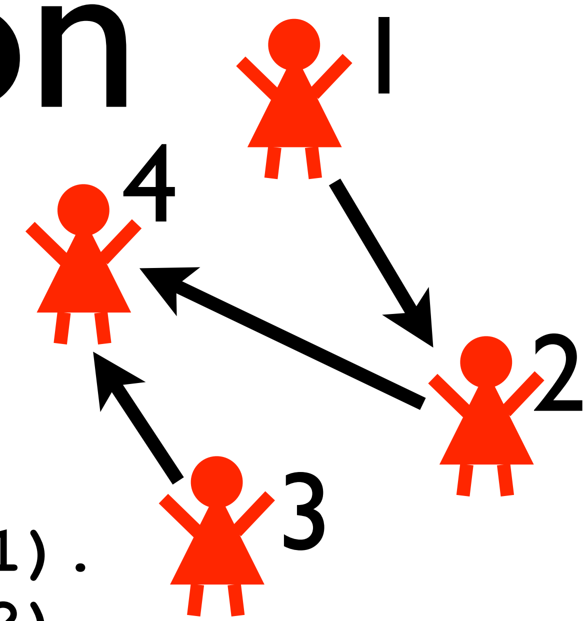



```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```


[Kimmig et al, ECML 07]

Probabilistic explanation based learning



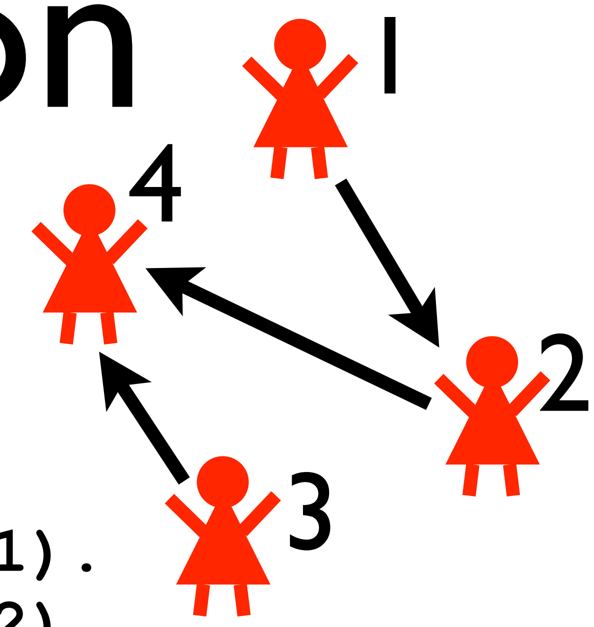

```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```

example `smokes(4)`

[Kimmig et al, ECML 07]

Probabilistic explanation based learning




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

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```

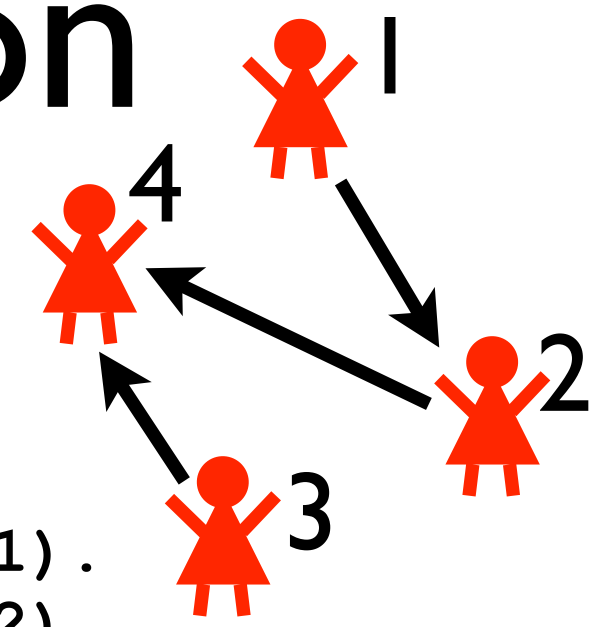
example `smokes(4)`

proofs

```
stress(4)
influences(2,4) & stress(2)
influences(2,4) & influences(1,2) & stress(1)
influences(3,4) & stress(3)
```


[Kimmig et al, ECML 07]

Probabilistic explanation based learning




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

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```

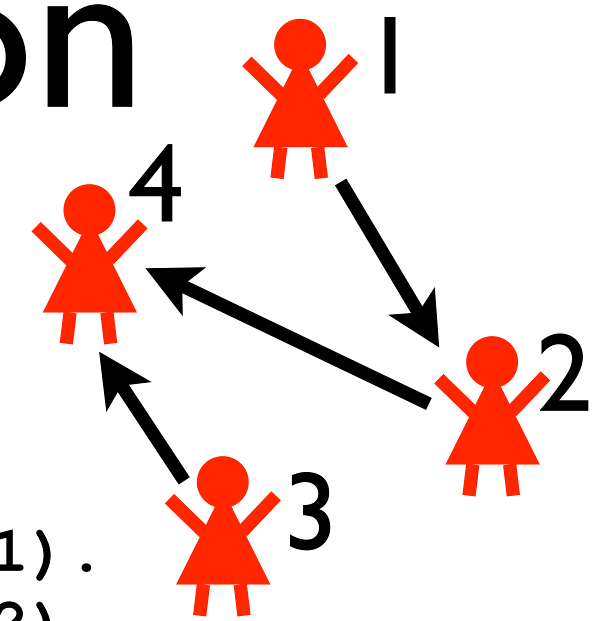
example `smokes(4)`

proofs

0.200	`stress(4)`
0.630	`influences(2,4) & stress(2)`
0.224	`influences(2,4) & influences(1,2) & stress(1)`
0.250	`influences(3,4) & stress(3)`

[Kimmig et al, ECML 07]

Probabilistic explanation based learning




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

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```

example `smokes(4)`

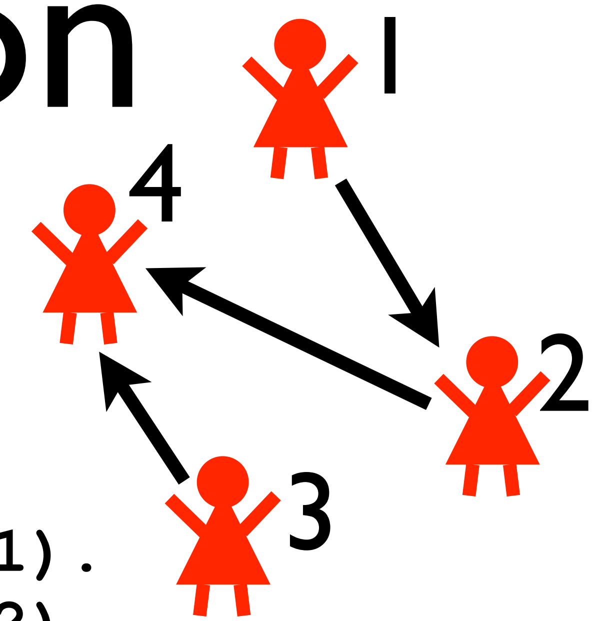
proofs

0.200 `stress(4)`
0.630 `influences(2,4) & stress(2)`
0.224 `influences(2,4) & influences(1,2) & stress(1)`
0.250 `influences(3,4) & stress(3)`

specific explanation `smokes(4) if influences(2,4) & stress(2)`

[Kimmig et al, ECML 07]

Probabilistic explanation based learning




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

```
0.4::stress(1).
0.9::stress(2).
0.5::stress(3).
0.2::stress(4).

0.8::influences(1,2).
0.7::influences(2,4).
0.5::influences(3,4).
```

example `smokes(4)`

proofs

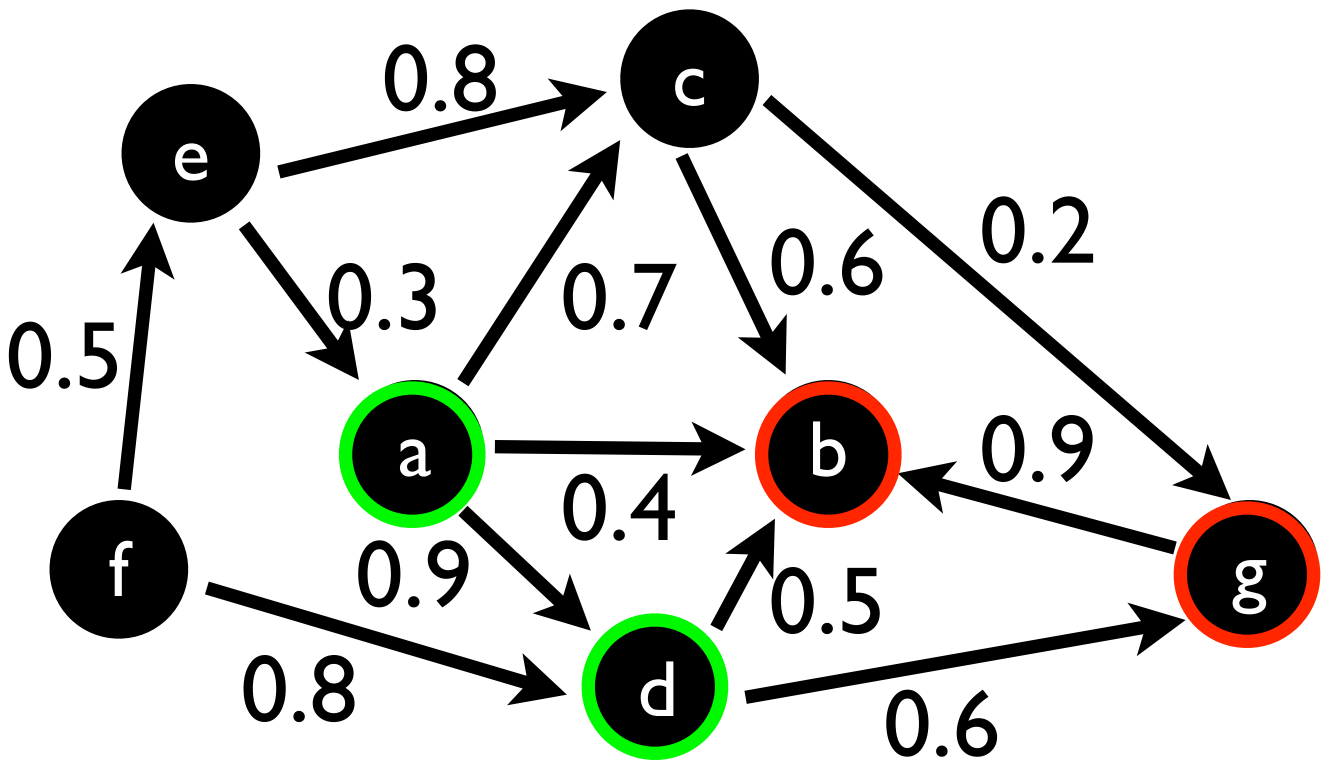
0.200 `stress(4)`
0.630 `influences(2,4) & stress(2)`
0.224 `influences(2,4) & influences(1,2) & stress(1)`
0.250 `influences(3,4) & stress(3)`

specific explanation `smokes(4) if influences(2,4) & stress(2)`

general explanation `smokes(A) if influences(B,A) & stress(B)`

[Kimmig et al, ECML 07]

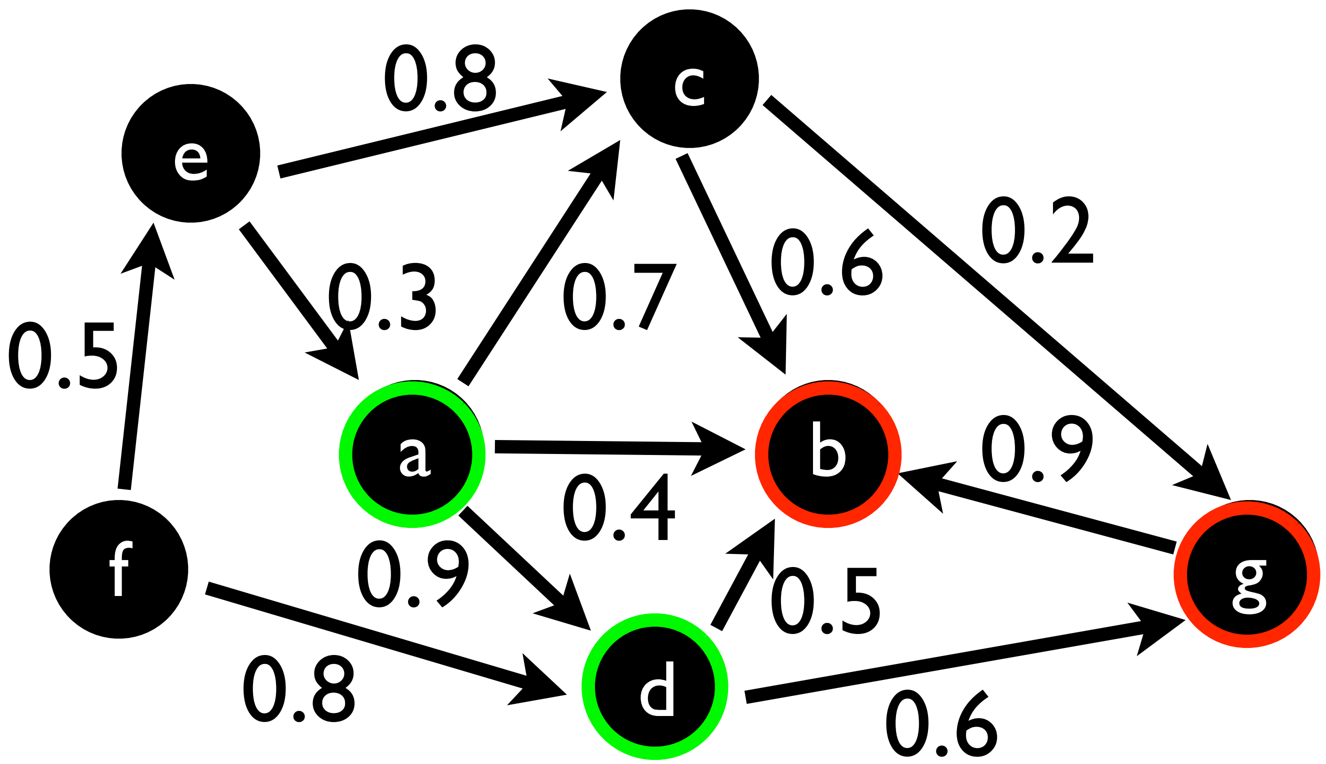
Probabilistic query mining




```
pos(a).    not pos(b).
pos(d).    not pos(g).
```


[Kimmig and De Raedt, IJCAI 09]

Probabilistic query mining

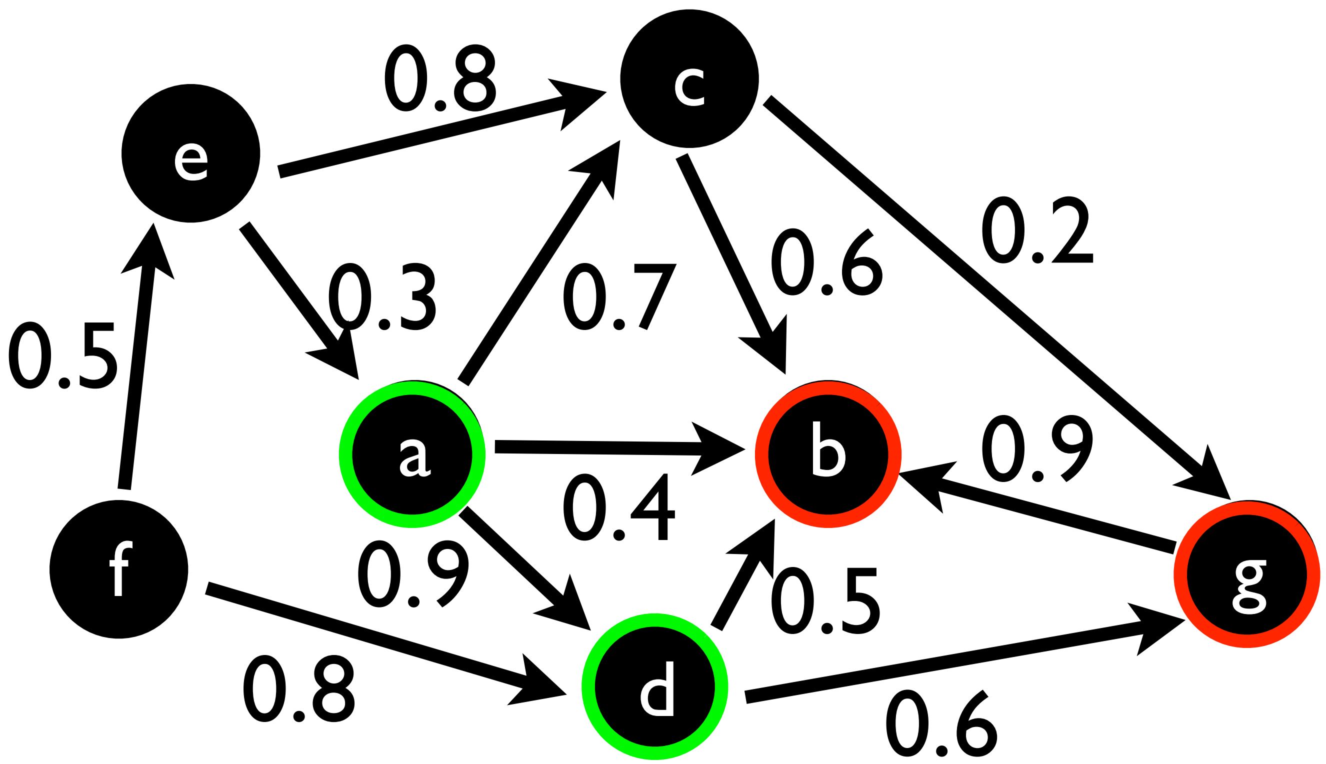



```
pos(a).    not pos(b).
pos(d).    not pos(g).
```

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

[Kimmig and De Raedt, IJCAI 09]

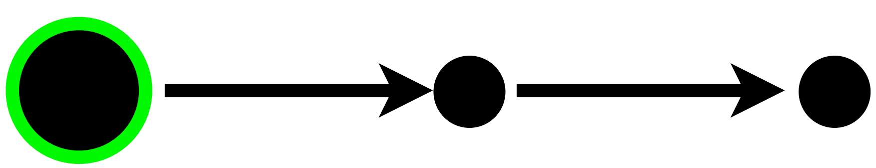
Probabilistic query mining




```
pos(a).    not pos(b).
pos(d).    not pos(g).
```

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

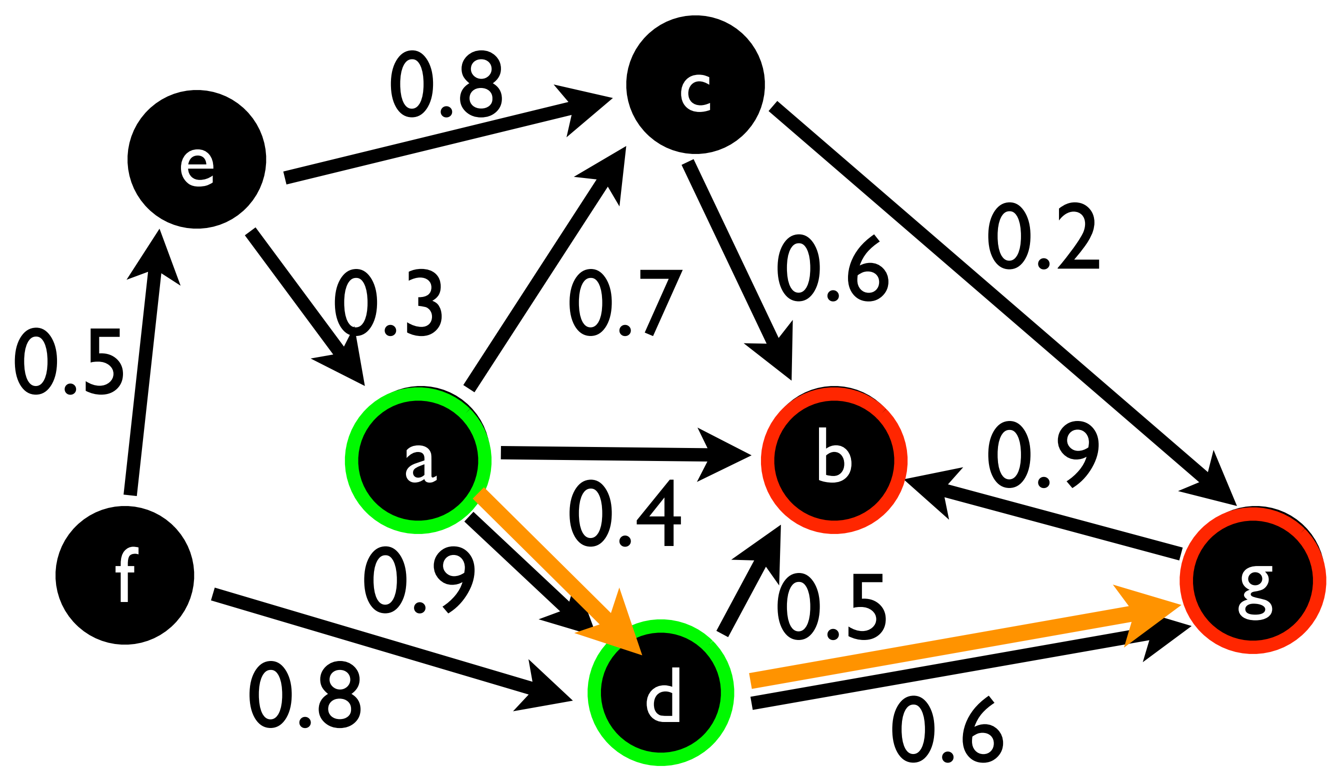
```
pos(X) :- edge(X,Y),edge(Y,Z).
```



1.08

[Kimmig and De Raedt, IJCAI 09]

Probabilistic query mining

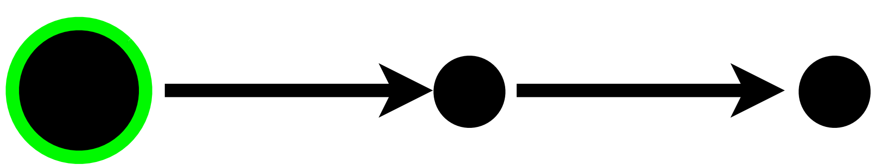


`pos(a).` `not pos(b).`

`pos(d).` `not pos(g).`

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

`pos(X) :- edge(X,Y),edge(Y,Z).`



1.08

a-d-g 0.54

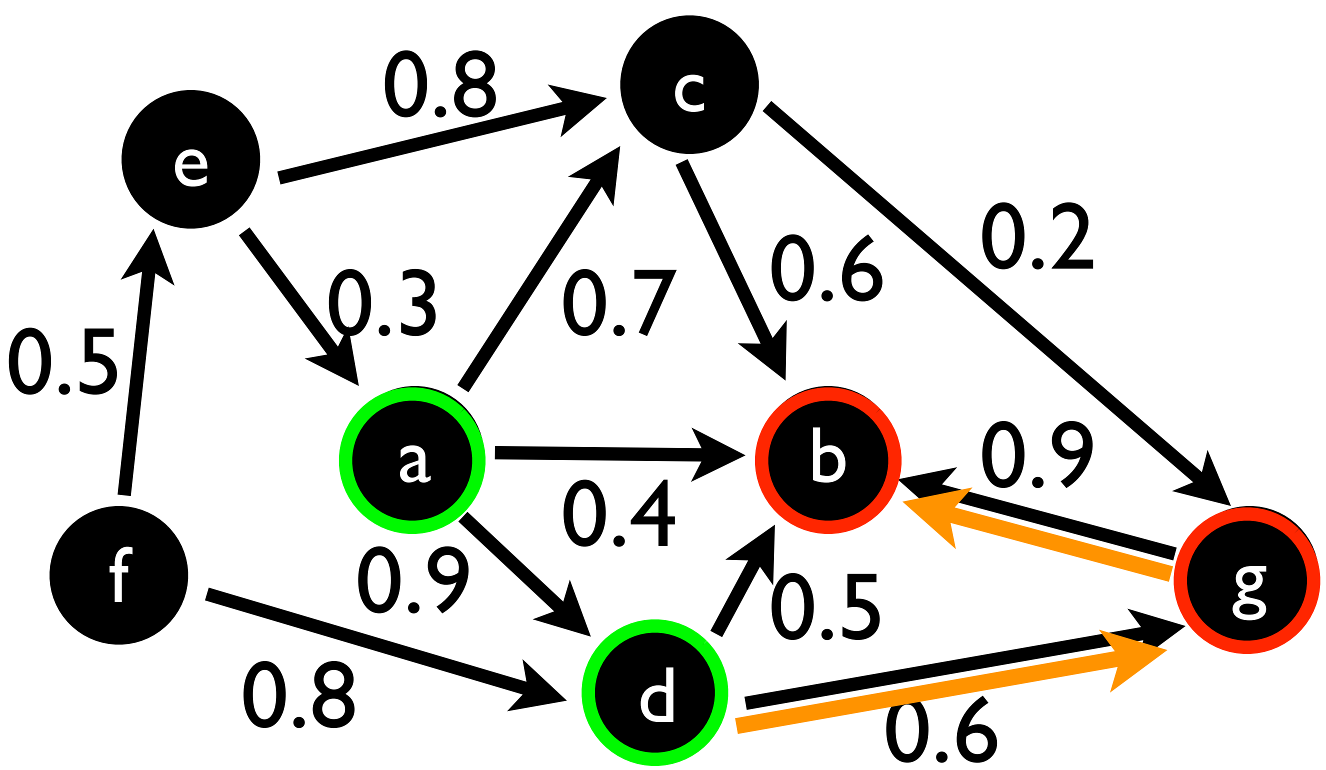
d-g-b 0.54

b-

g-

[Kimmig and De Raedt, IJCAI 09]

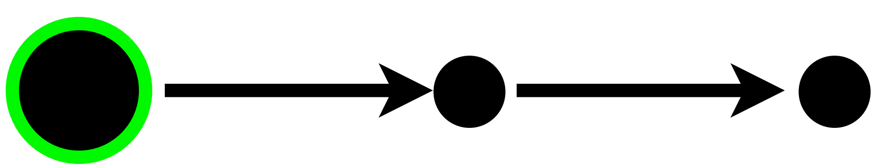
Probabilistic query mining




```
pos(a).    not pos(b).
pos(d).    not pos(g).
```

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

```
pos(X) :- edge(X,Y),edge(Y,Z).
```



1.08

a-d-g 0.54

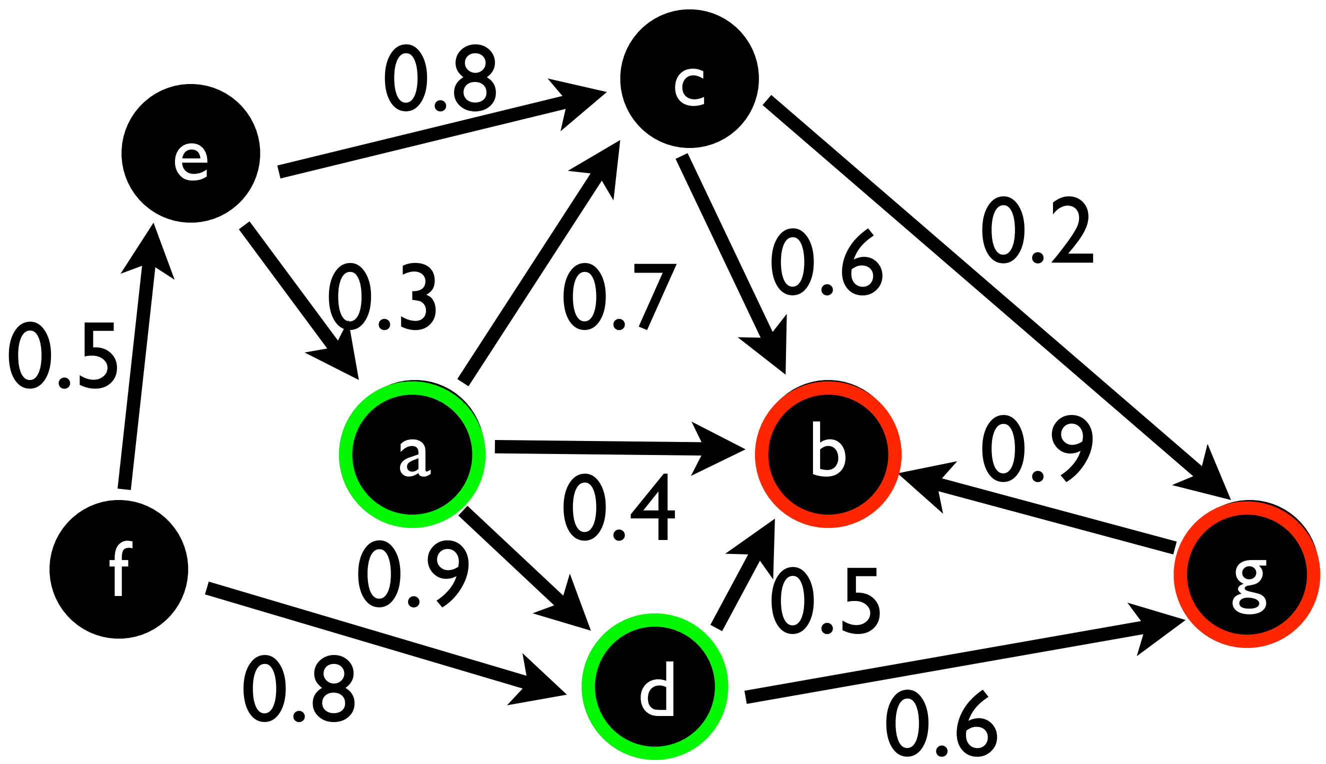
d-g-b 0.54

b-

g-

[Kimmig and De Raedt, IJCAI 09]

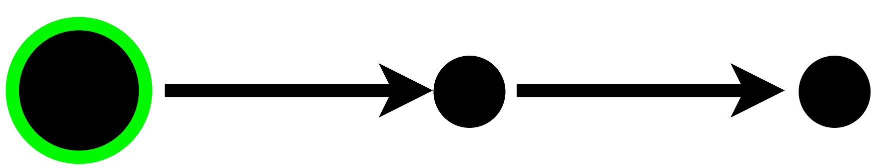
Probabilistic query mining




```
pos(a).    not pos(b).
pos(d).    not pos(g).
```

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

```
pos(X) :- edge(X,Y),edge(Y,Z).
```



1.08

a-d-g 0.54

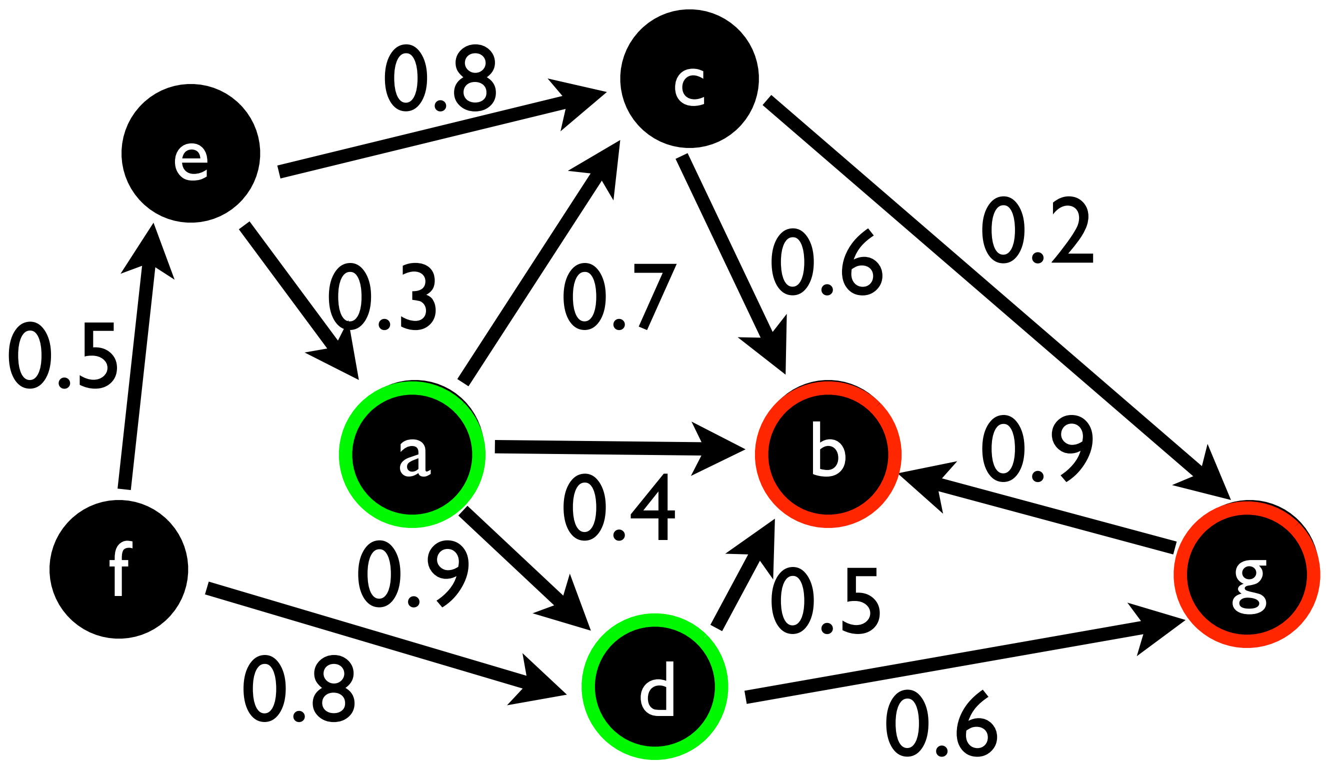
d-g-b 0.54

b-

g-

[Kimmig and De Raedt, IJCAI 09]

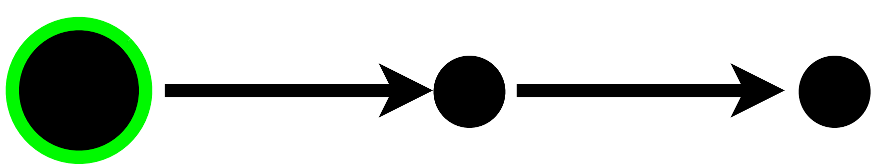
Probabilistic query mining




```
pos(a).    not pos(b).
pos(d).    not pos(g).
```

subgraph queries that maximize $\sum_{pos} P - \sum_{neg} P$?

```
pos(X) :- edge(X,Y),edge(Y,Z).
```



1.08

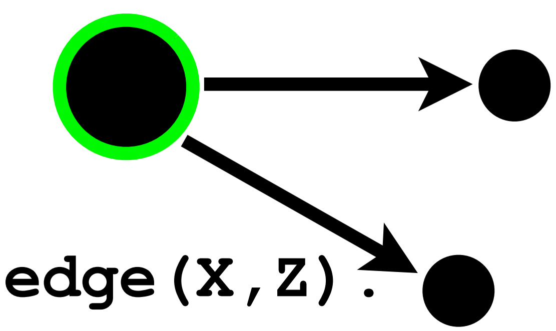
a-d-g 0.54

d-g-b 0.54

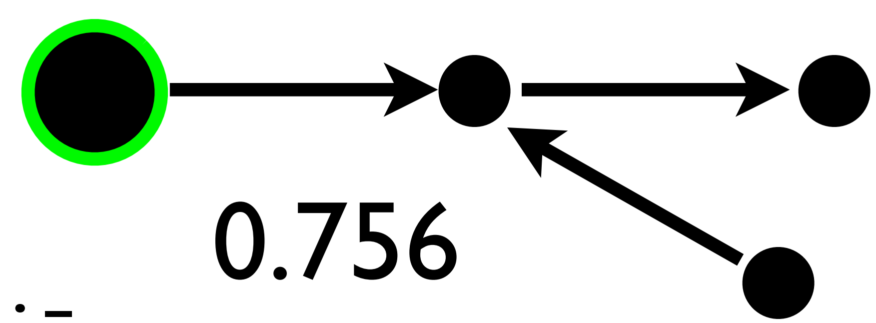
b-

g-

```
pos(X) :-
edge(X,Y),edge(X,Z).
```



0.93



0.756

```
pos(X) :-
edge(X,Y),edge(Y,Z),edge(W,Y).
```


[Kimmig and De Raedt, IJCAI 09]

ProbFOIL

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

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

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]

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]

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]

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]

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).
        ...
```

Prob2FOIL

- Learning **probabilistic rules** from probabilistic facts

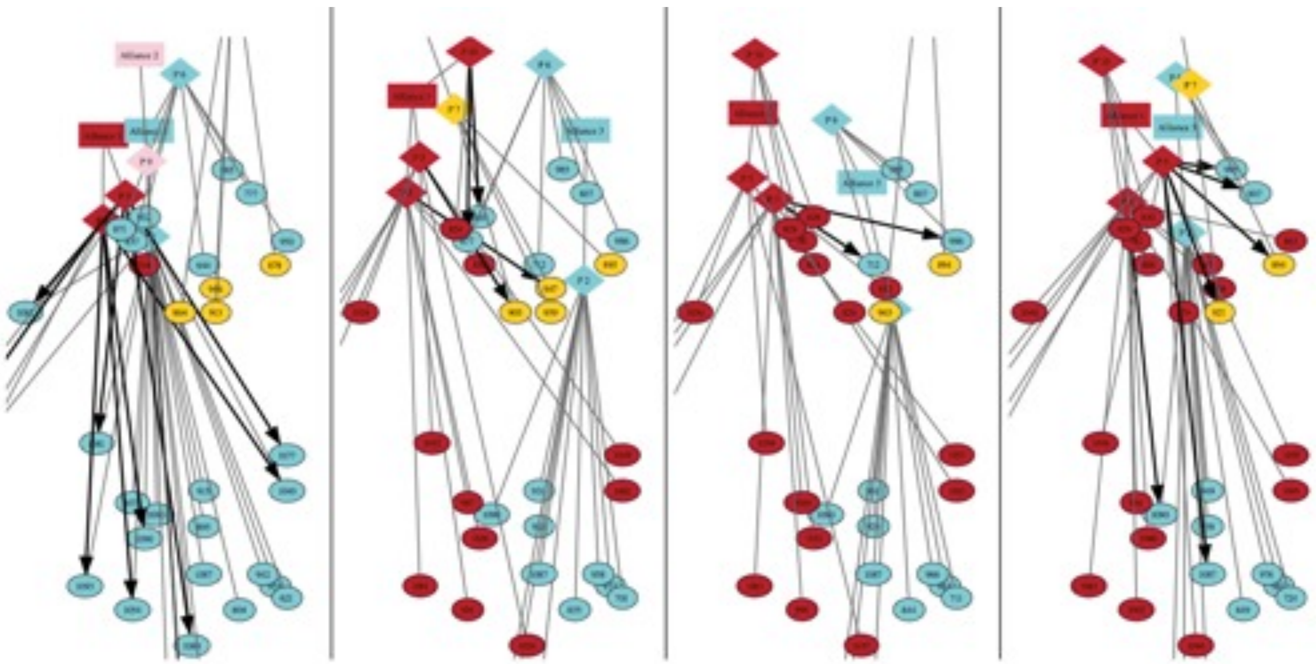
```
 0.4987::rain(1).
0.3591::windok(1).
0.4534::sunshine(1).
0.3257::surfing(1).
 0.7391::rain(2).
0.6022::windok(2).
0.9837::sunshine(2).
0.2592::surfing(2).
 0.2898::rain(3).
0.7423::windok(3).
0.2275::sunshine(3).
0.5688::surfing(3).
       ...
```

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

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

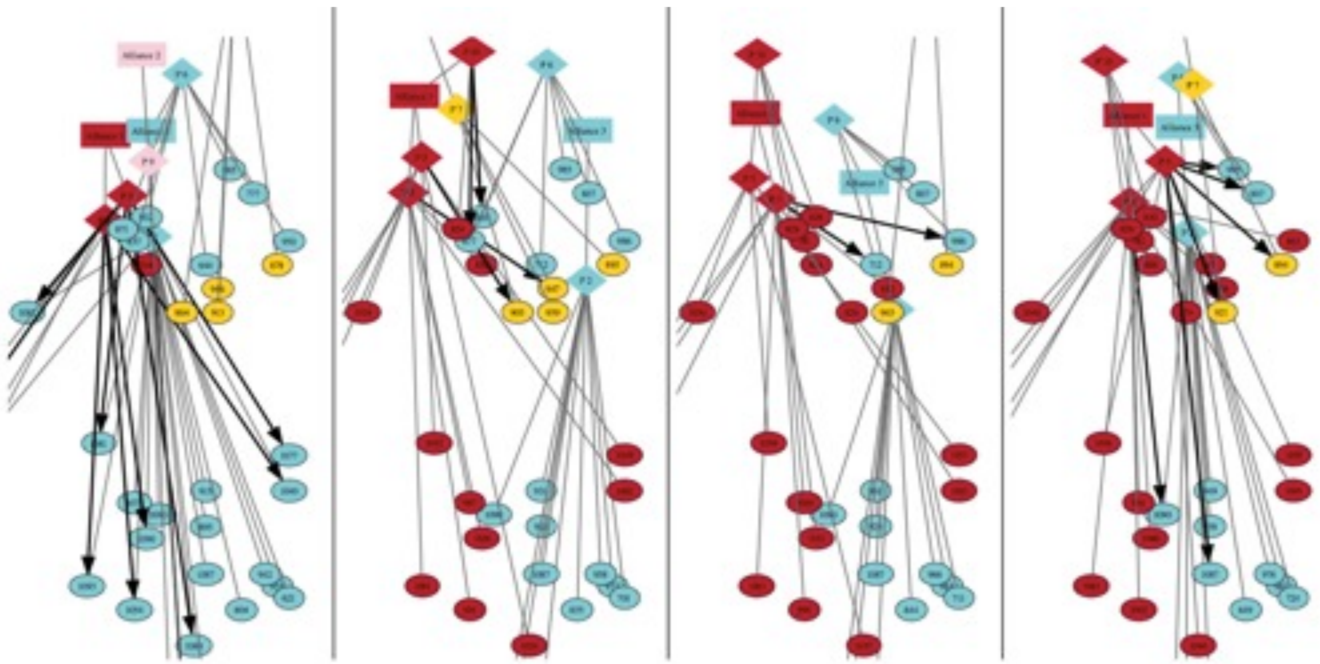
Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



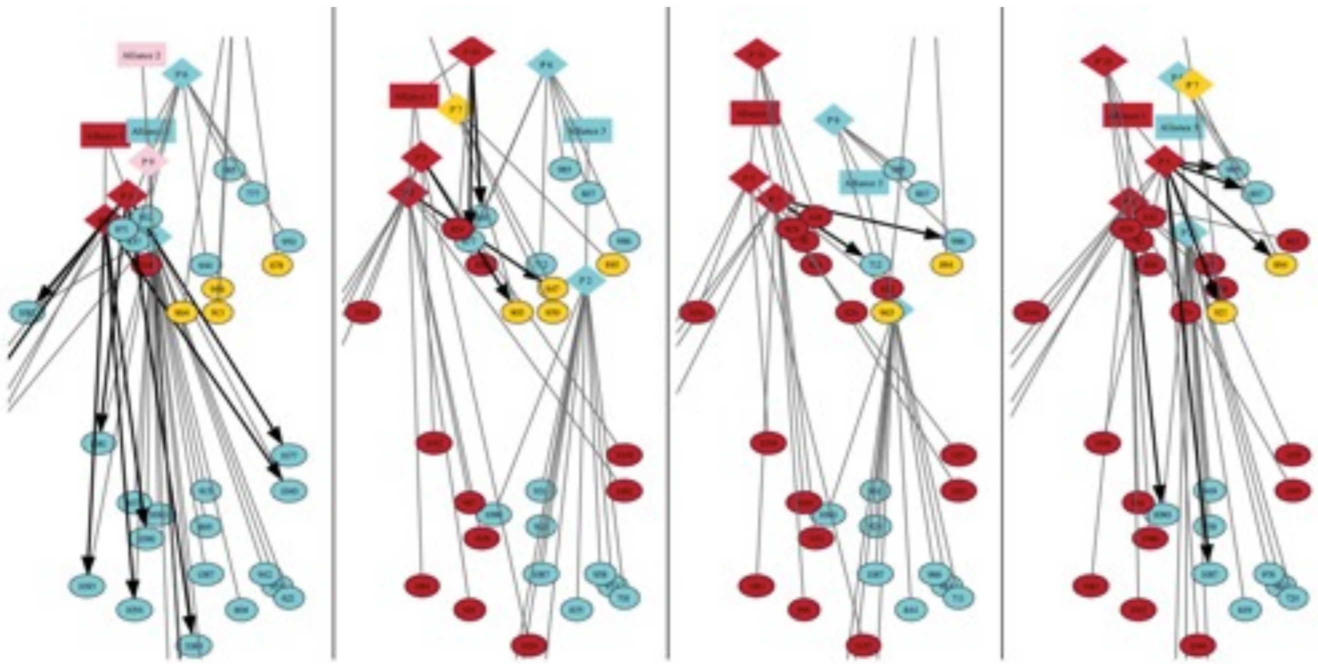
how does the world change over time?

```
0.4::conquest(Attacker,C); 0.6::nil <-
        city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

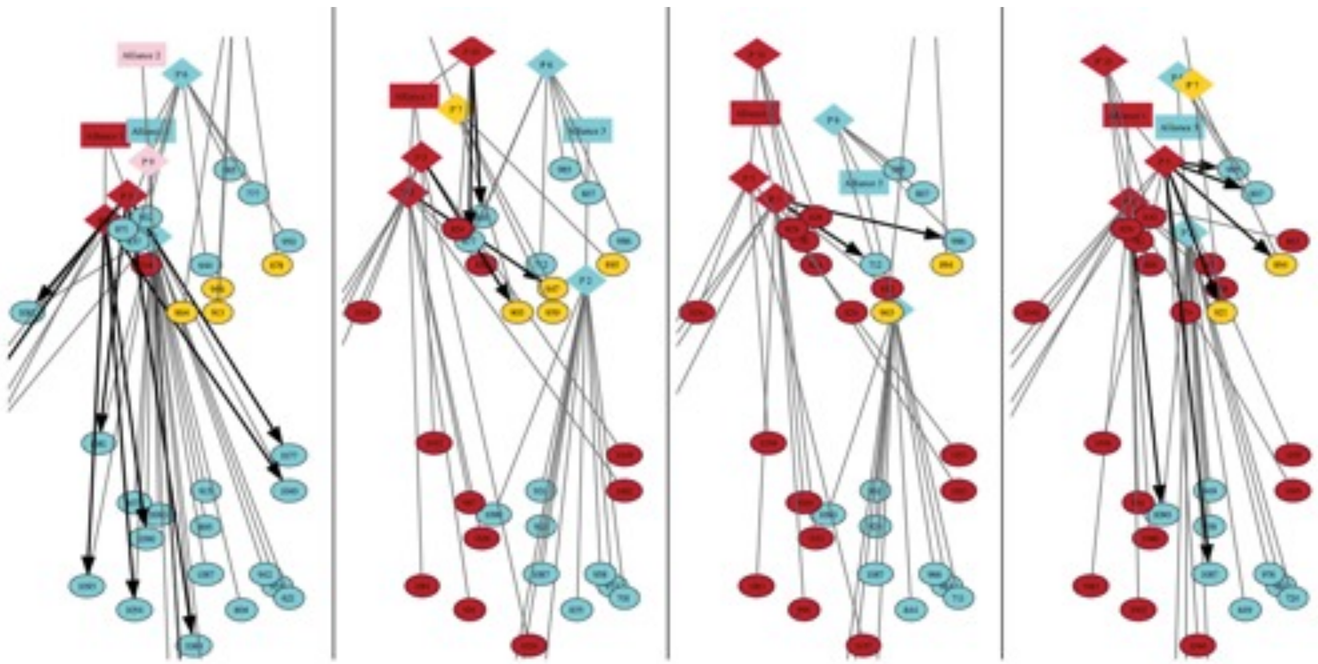
one of the **effects** holds at time T+1

```
0.4::conquest(Attacker,C); 0.6::nil <-
                    city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

one of the **effects** holds at time T+1

```
0.4::conquest(Attacker,C); 0.6::nil <-
              city(C,Owner),city(C2,Attacker),close(C,C2).
```

if **cause** holds at time T

[Thon et al, MLJ 11]

Distributional Clauses (DC)

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

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

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]

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]

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]

Distributional Clauses (DC)

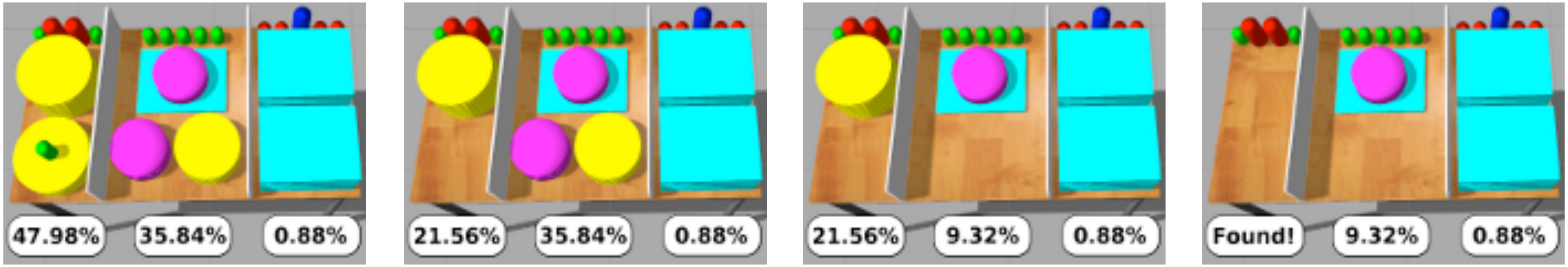
- Discrete- and continuous-valued random variables
- Inference: particle filter

```
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]

Occluded Object Search

- DC model of objects and their spatial arrangement
- different types of objects suitable for different tasks
- shelves with objects of different shape and size
- given a task, find an object to perform that task

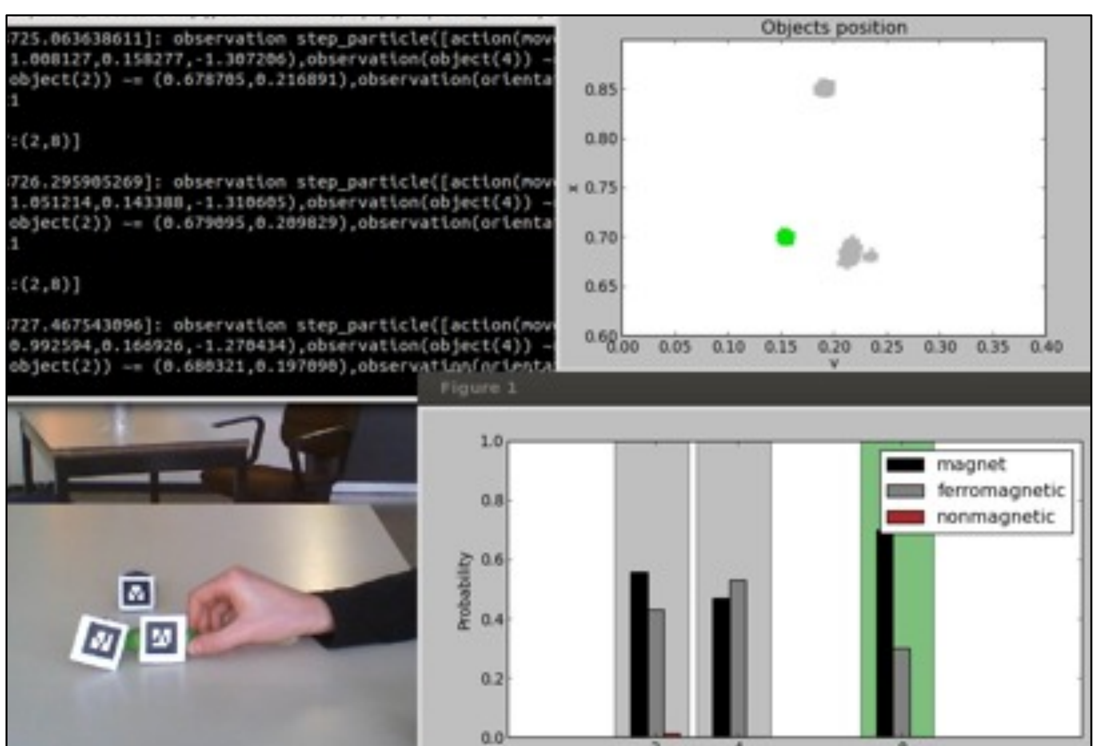


[Moldovan et al]

Relational State Estimation over Time

Magnetism scenario

- object tracking
- category estimation from interactions

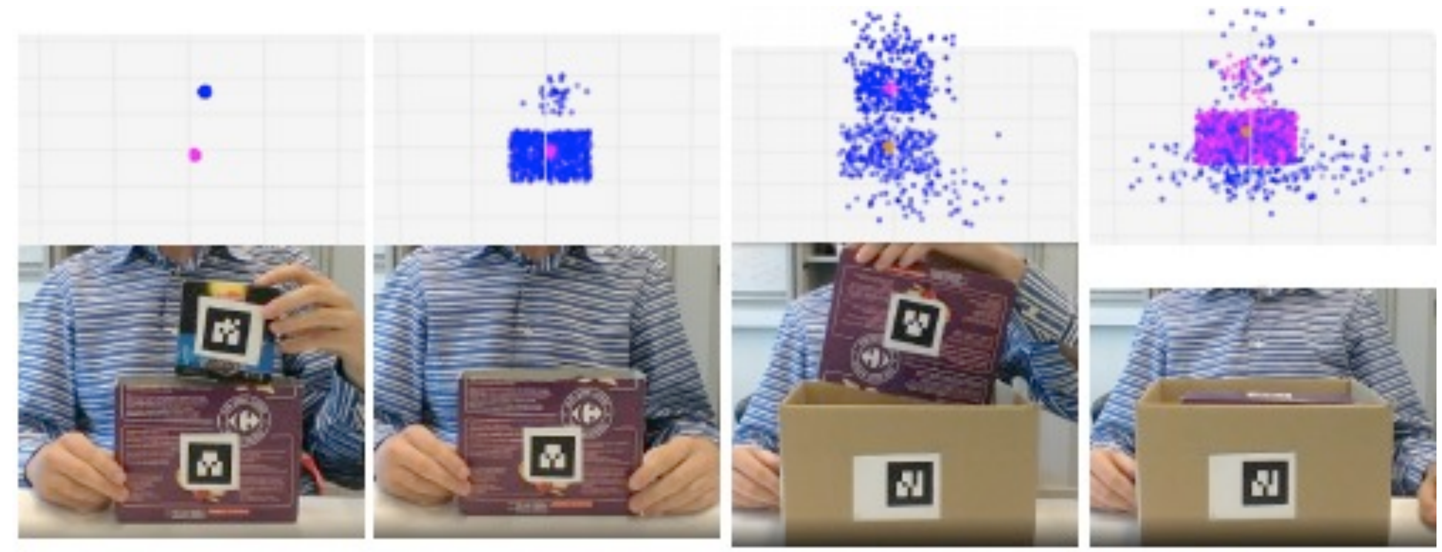


[Nitti et al, IEEE/RSJ 13]

Relational State Estimation over Time

Magnetism scenario

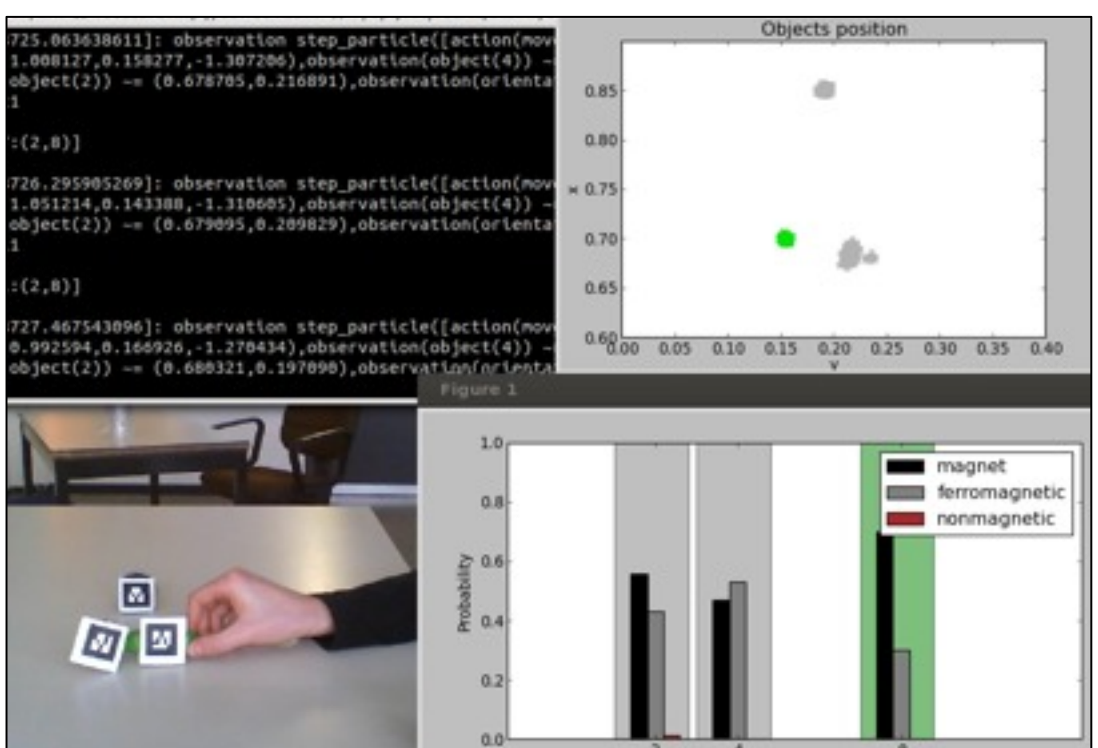
- object tracking
- category estimation from interactions



(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

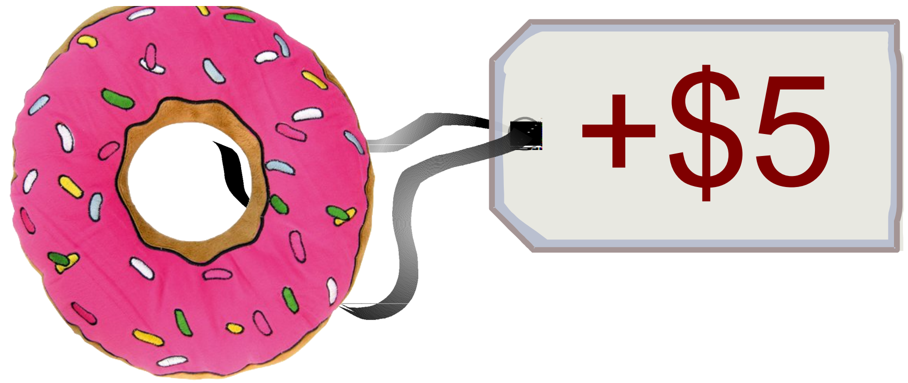


[Nitti et al, IEEE/RSJ 13]

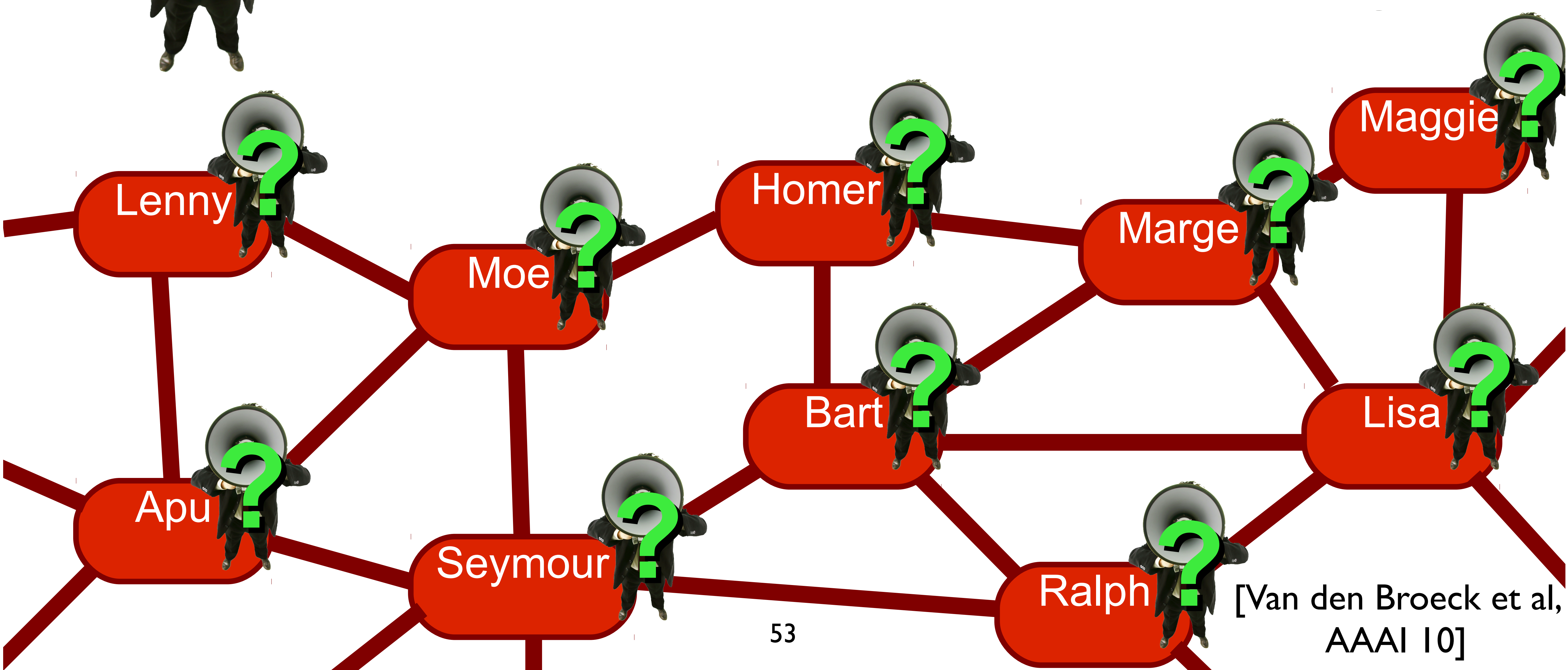
Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

Viral Marketing

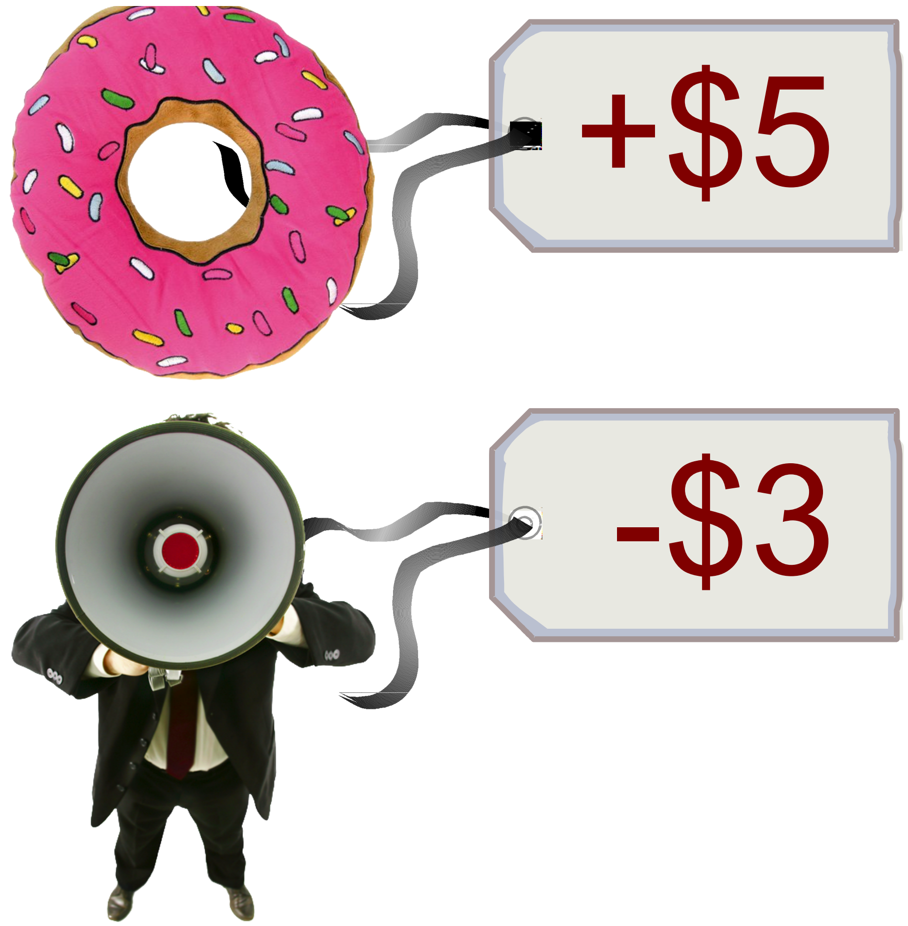


Which advertising strategy maximizes expected profit?

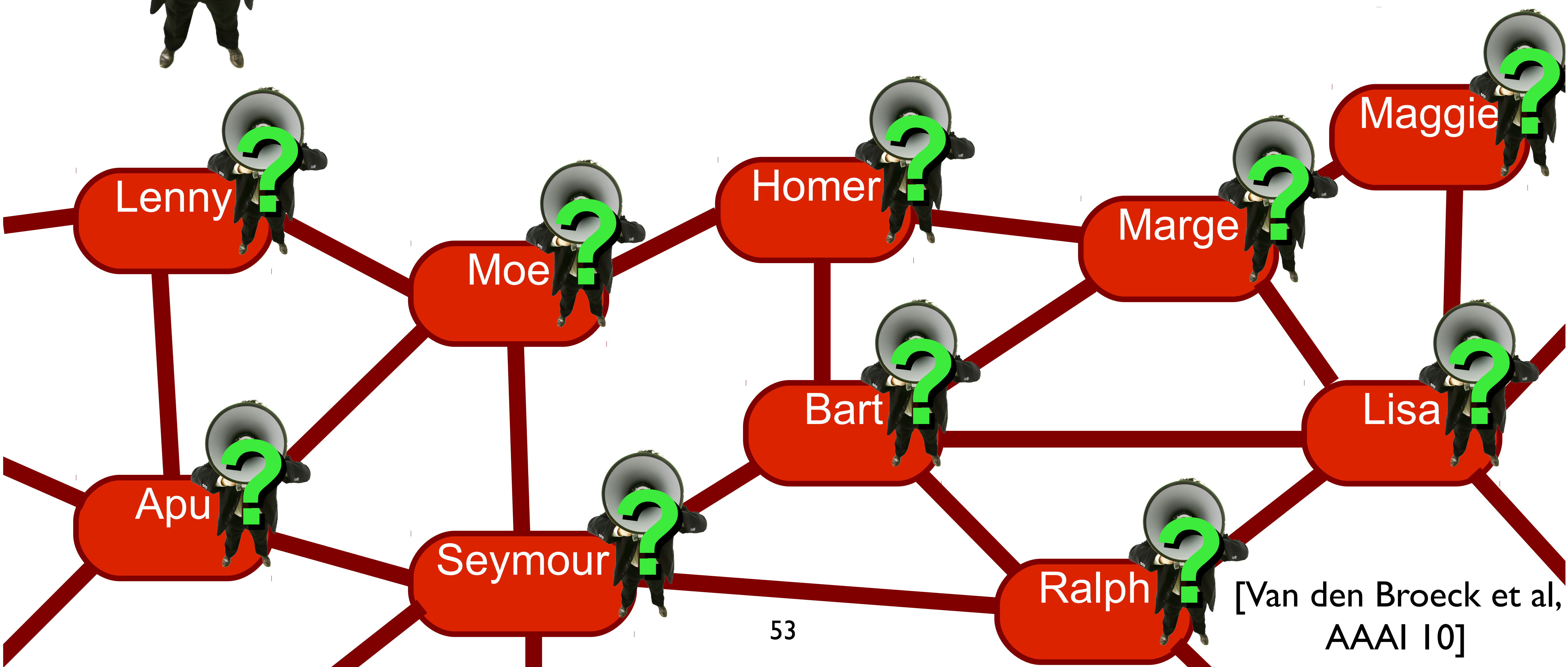


[Van den Broeck et al, AAAI 10]

Viral Marketing

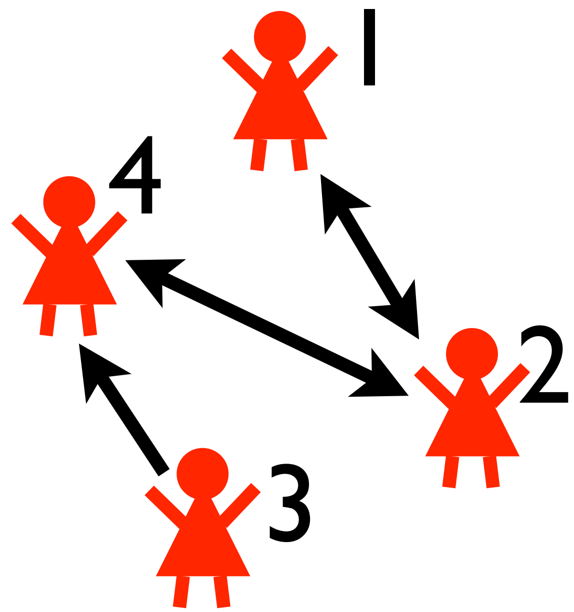


decide truth values of some atoms



[Van den Broeck et al, AAAI 10]

DTProbLog

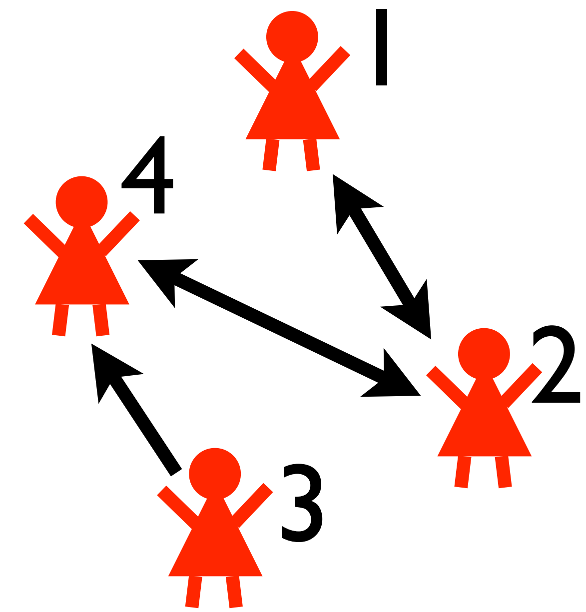



```
person(1).
person(2).
person(3).
person(4).

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?



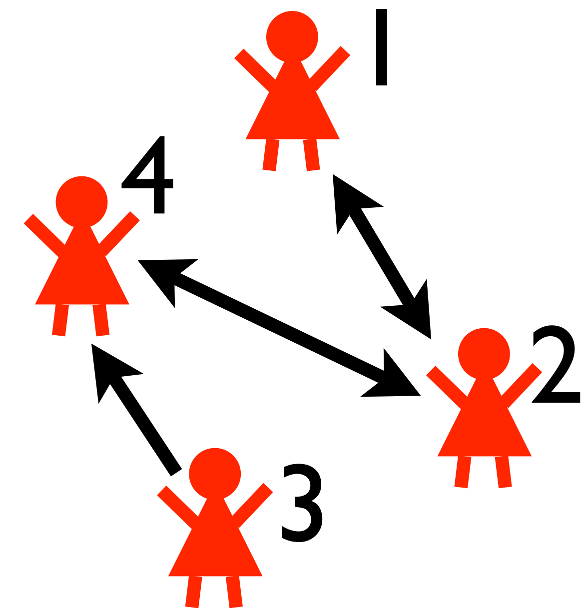

```
person(1).
person(2).
person(3).
person(4).

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

DTProbLog

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

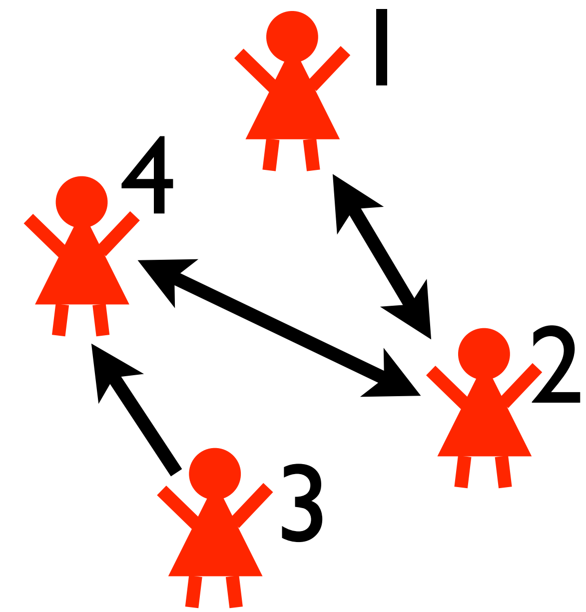
```
0.3 :: buy_trust(X,Y) :- friend(X,Y).
0.2 :: buy_marketing(P) :- person(P).

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



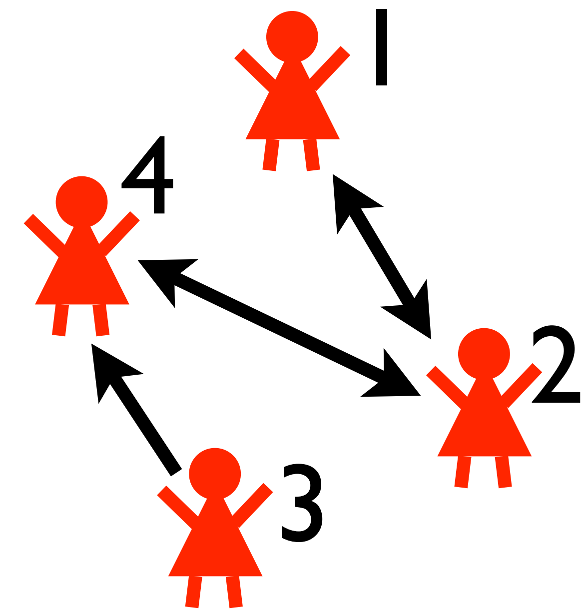

```
? :: marketed(P) :- person(P).

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).
```

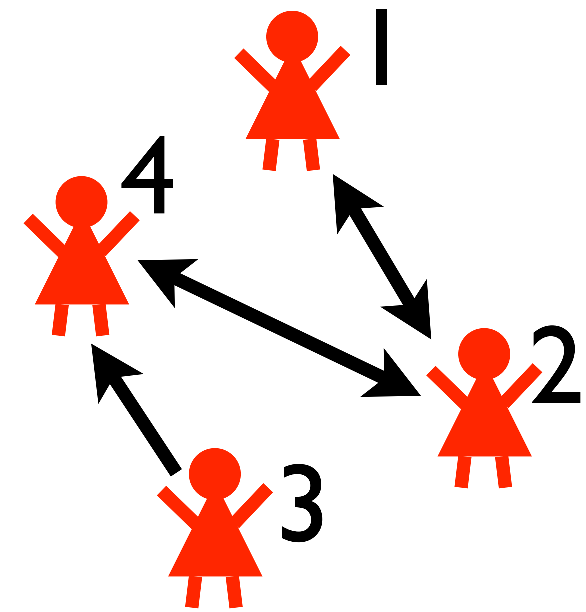
DTProbLog

```
? :: 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).
```

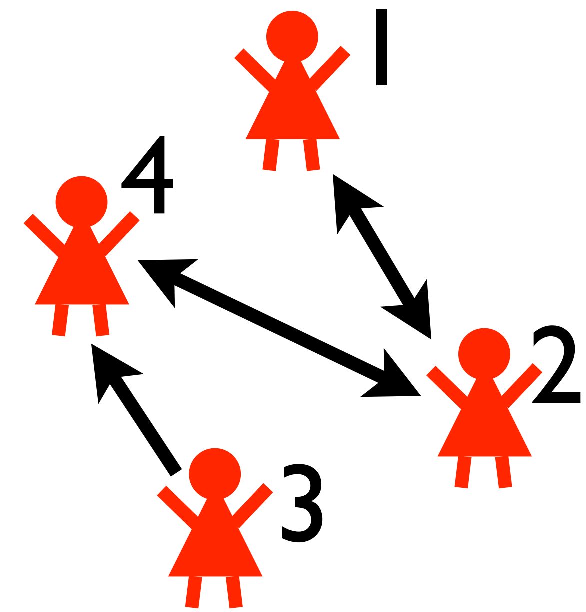
DTProbLog




```
? :: 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

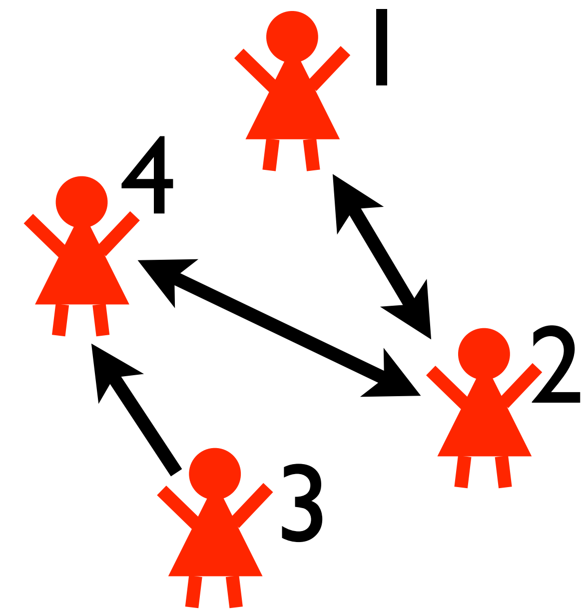



```
? :: marketed(P) :- person(P).

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

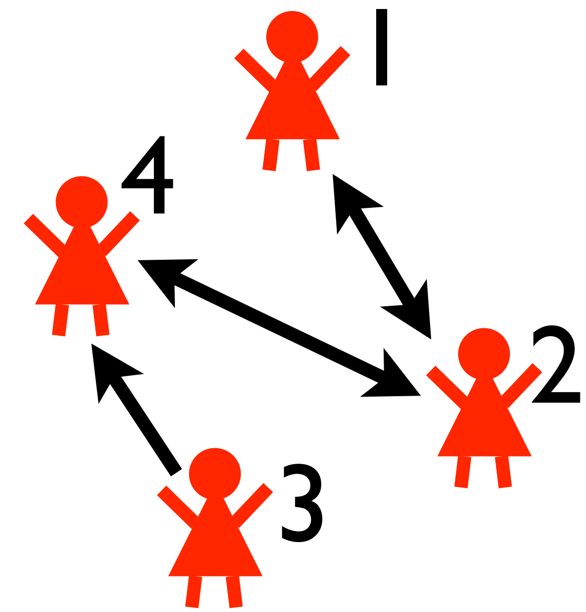



```
? :: marketed(P) :- person(P).

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

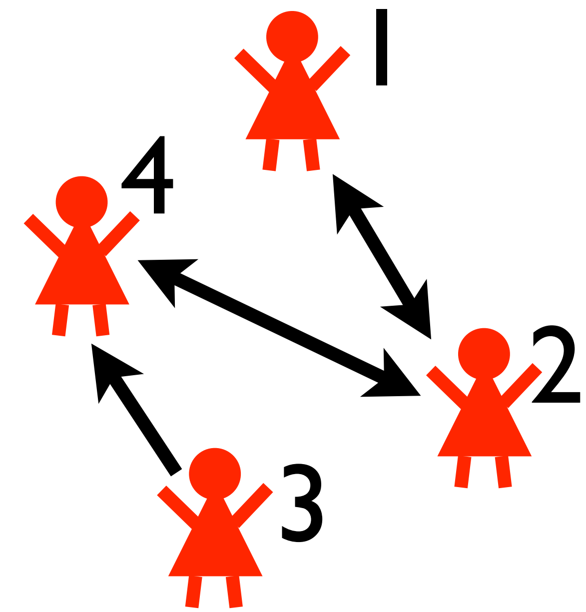



```
? :: marketed(P) :- person(P).

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).
```

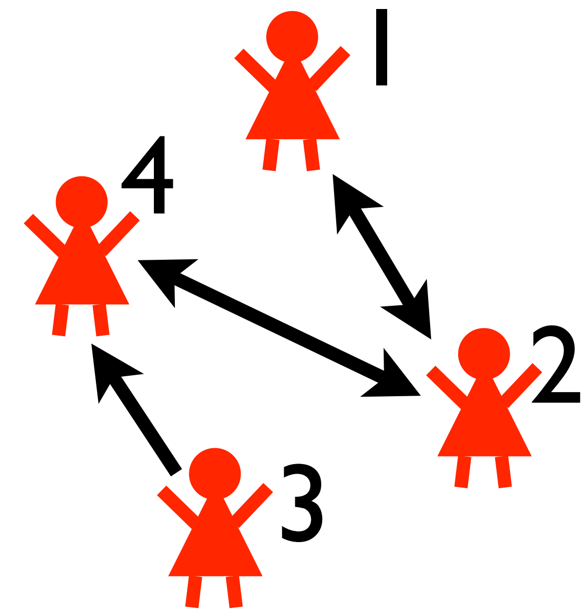
```
person(1).
person(2).
person(3).
person(4).

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

utility = −3 + −3 + 5 + 5 = 4

probability = 0.0032

```
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).
```

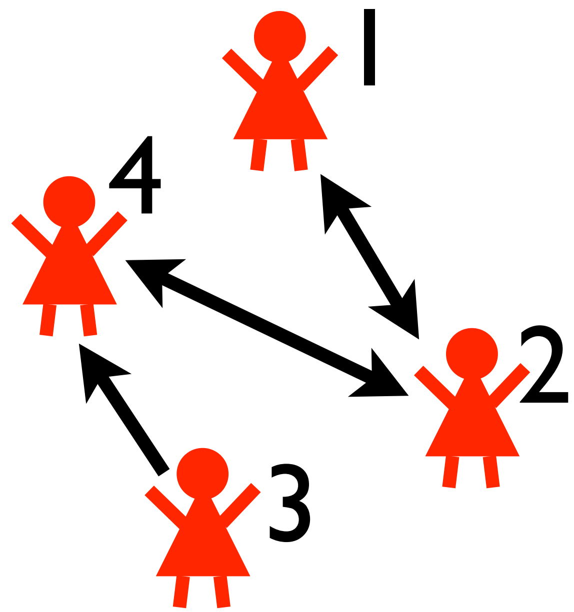
utility = −3 + −3 + 5 + 5 = 4

probability = 0.0032

```
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

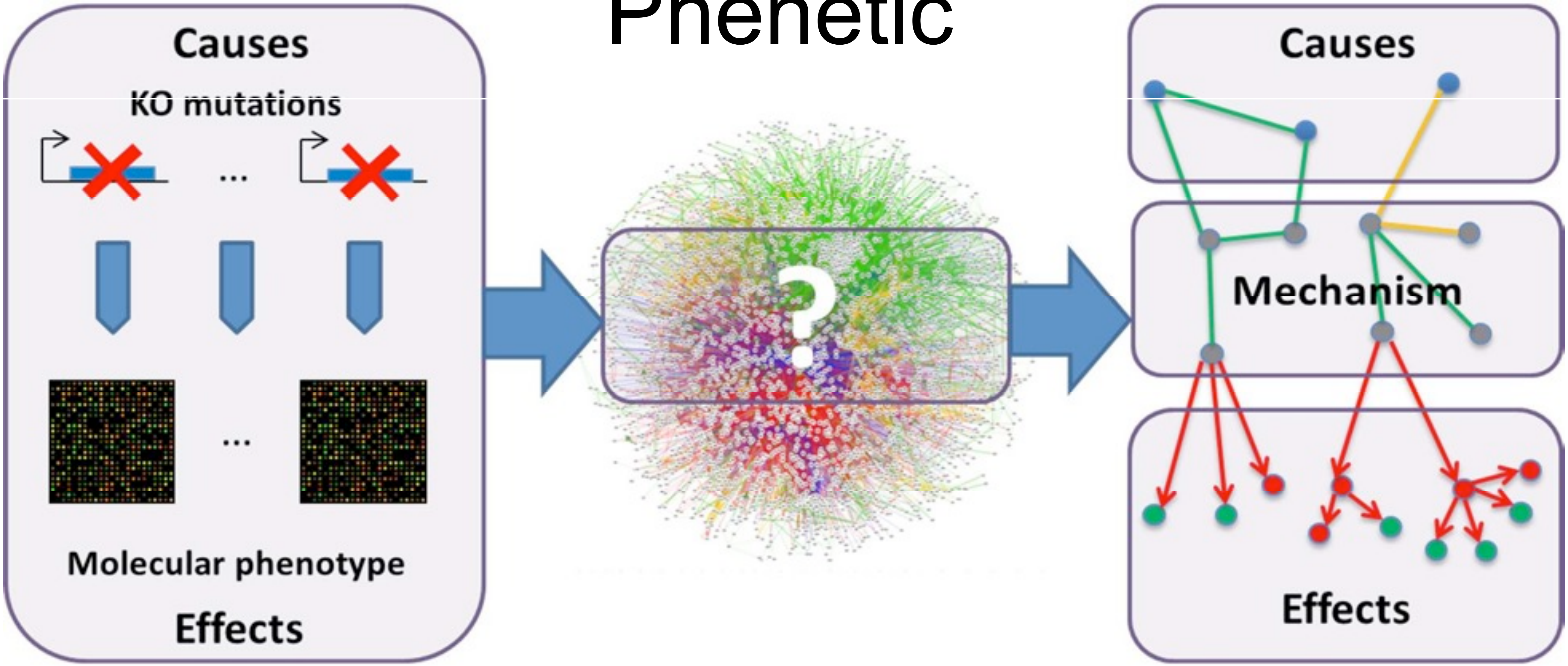



```
? :: marketed(P) :- person(P).

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

Phenetic



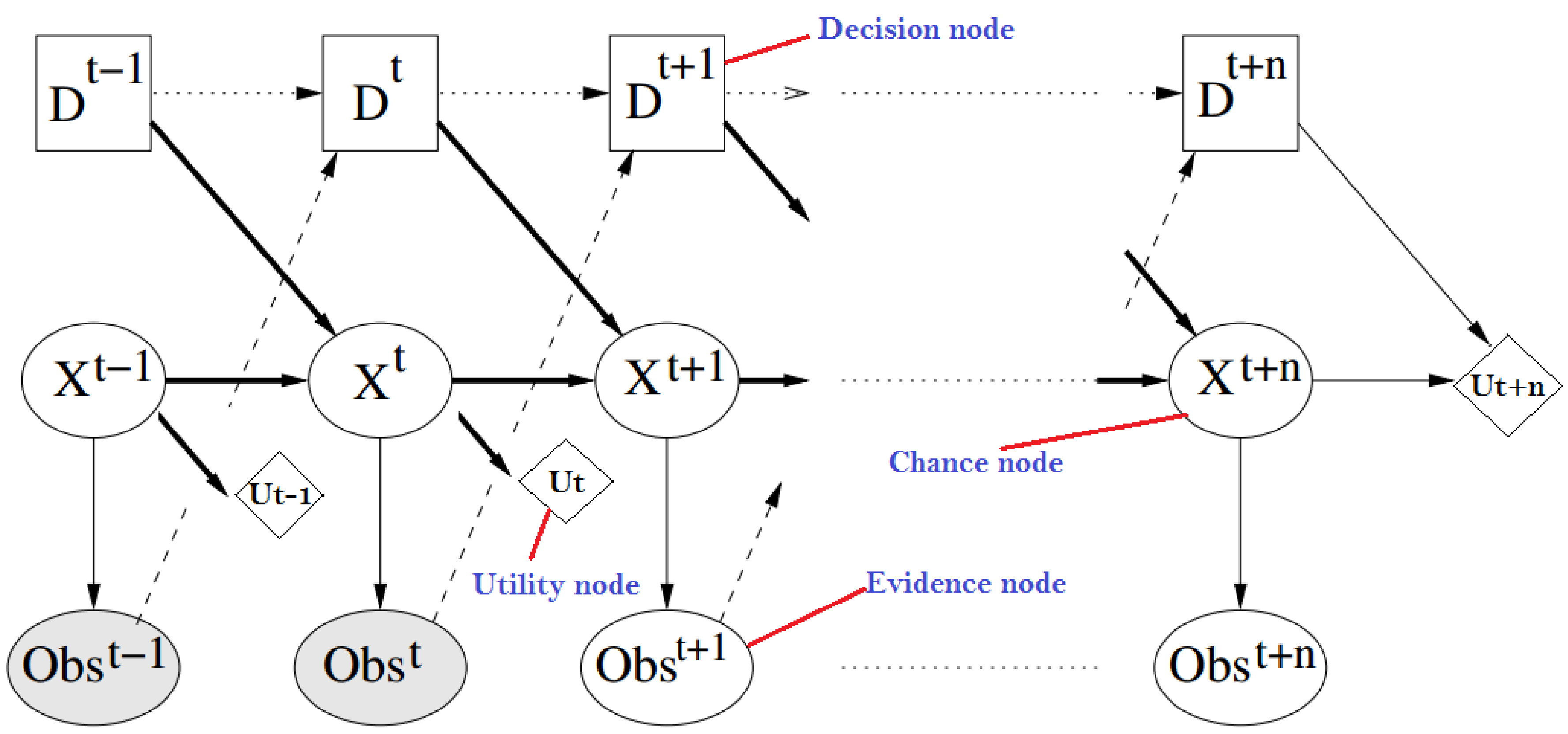
- Causes: Mutations
  - All related to similar phenotype
- Effects: Differentially expressed genes
- 27 000 cause effect pairs

- Interaction network:
  - 3063 nodes
    - Genes
    - Proteins
  - 16794 edges
    - Molecular interactions
    - Uncertain

- Goal: connect causes to effects through common subnetwork
  - = Find mechanism
- Techniques:
  - DTProbLog
  - Approximate inference

[De Maeyer et al., Molecular Biosystems 13]

Dynamic Decision Network



Use non-functional requirements and quality of context to make decisions based on the outcome of previous decisions

Pick-a-caregiver scenario:
Decide which caregiver to call in case of an emergency by optimising over the probability distribution over her availability and locality and the non-functional requirements evaluated on previous decisions.

[Naqvi, Preuveneers, Meert & Berbers]

Overview

- ProbLog Basics
  - ProbLog by example
  - Inference
  - Parameter Learning
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

cProbLog: constraints on possible worlds

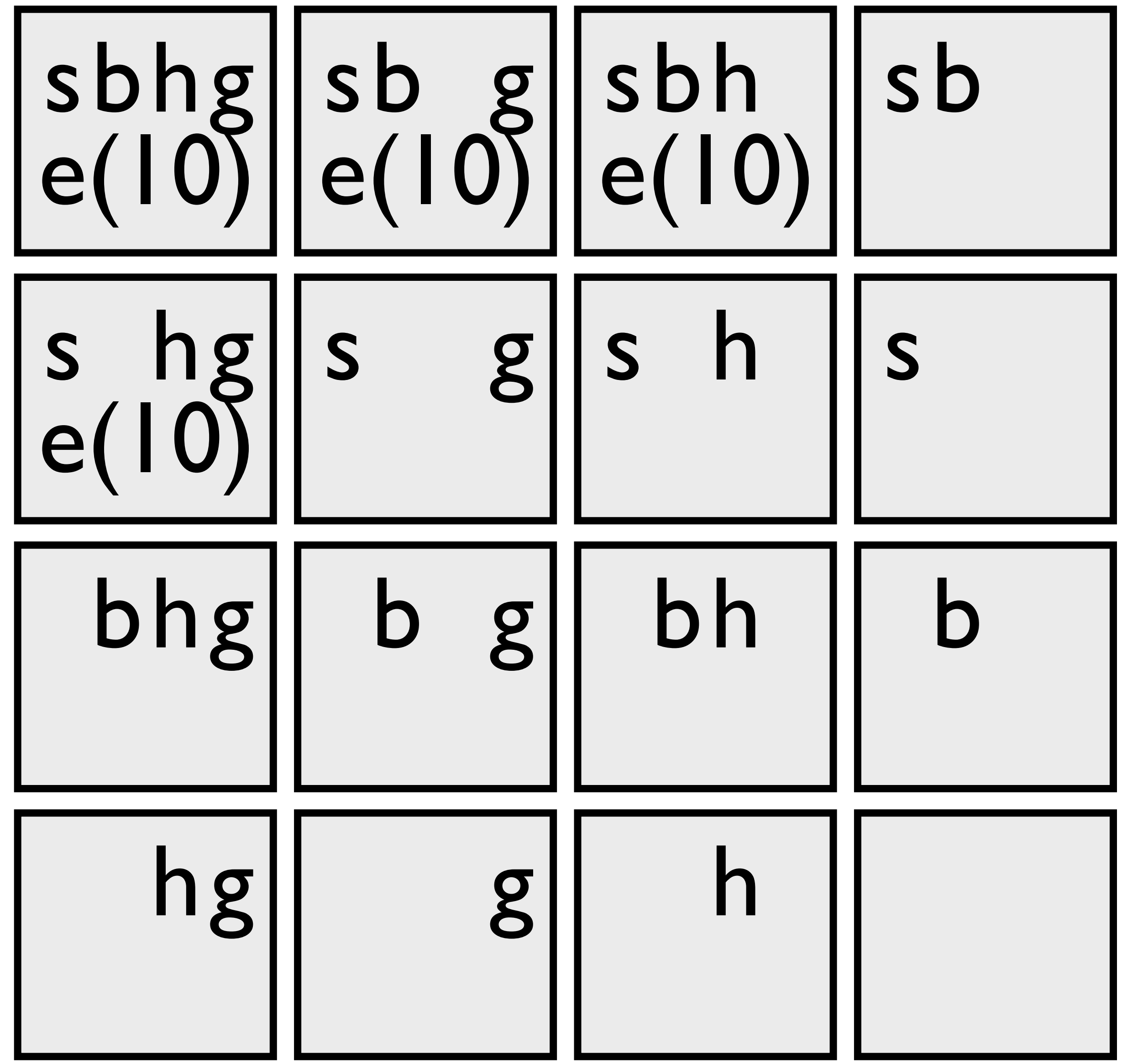
```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).

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).

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

excess(Limit) :- ...

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

distribution over **all** possible worlds



[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	

constraints as first-order logic formulas

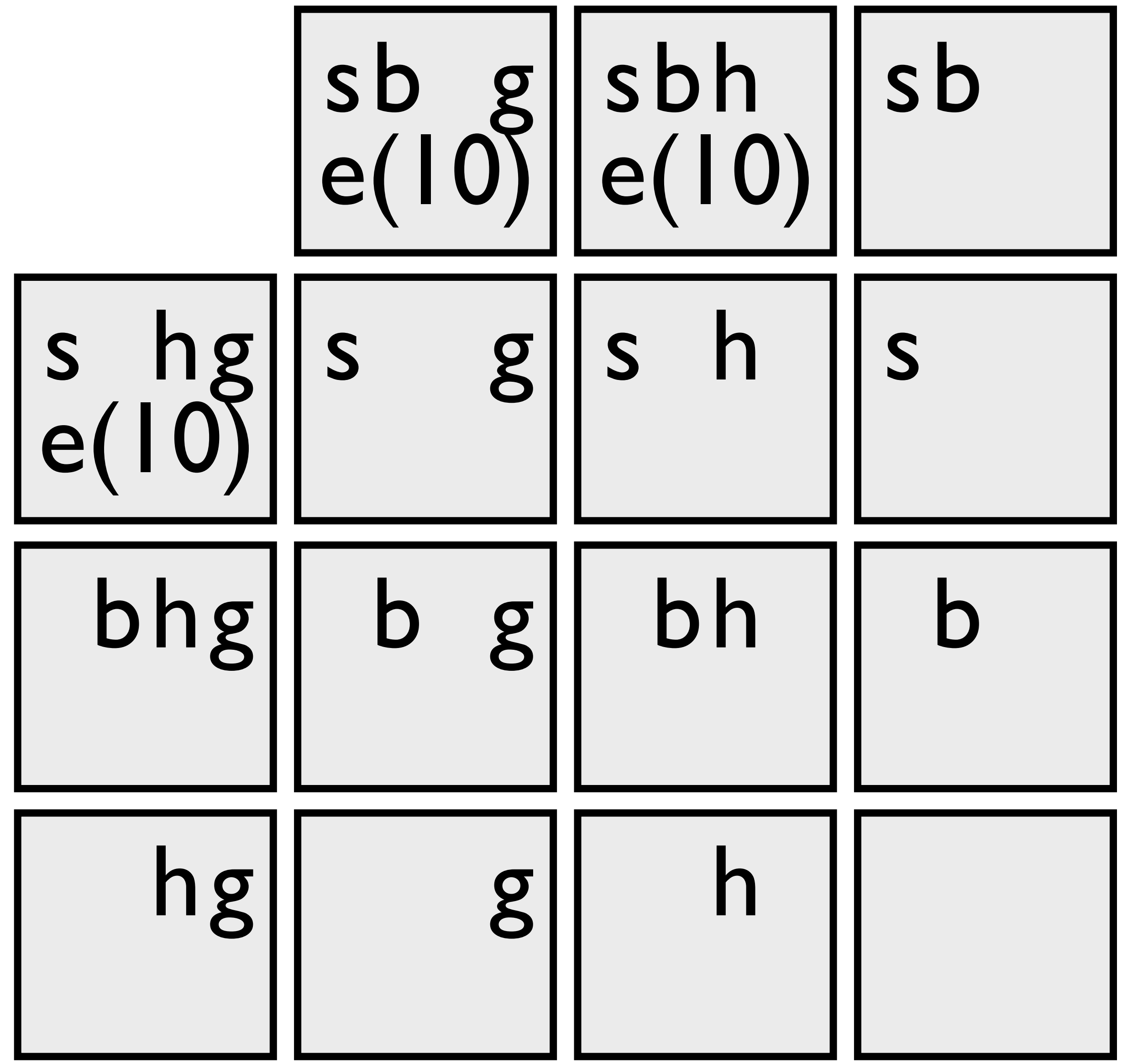
[Fierens et al, PP 12; Shterionov et al]

cPobLog: 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

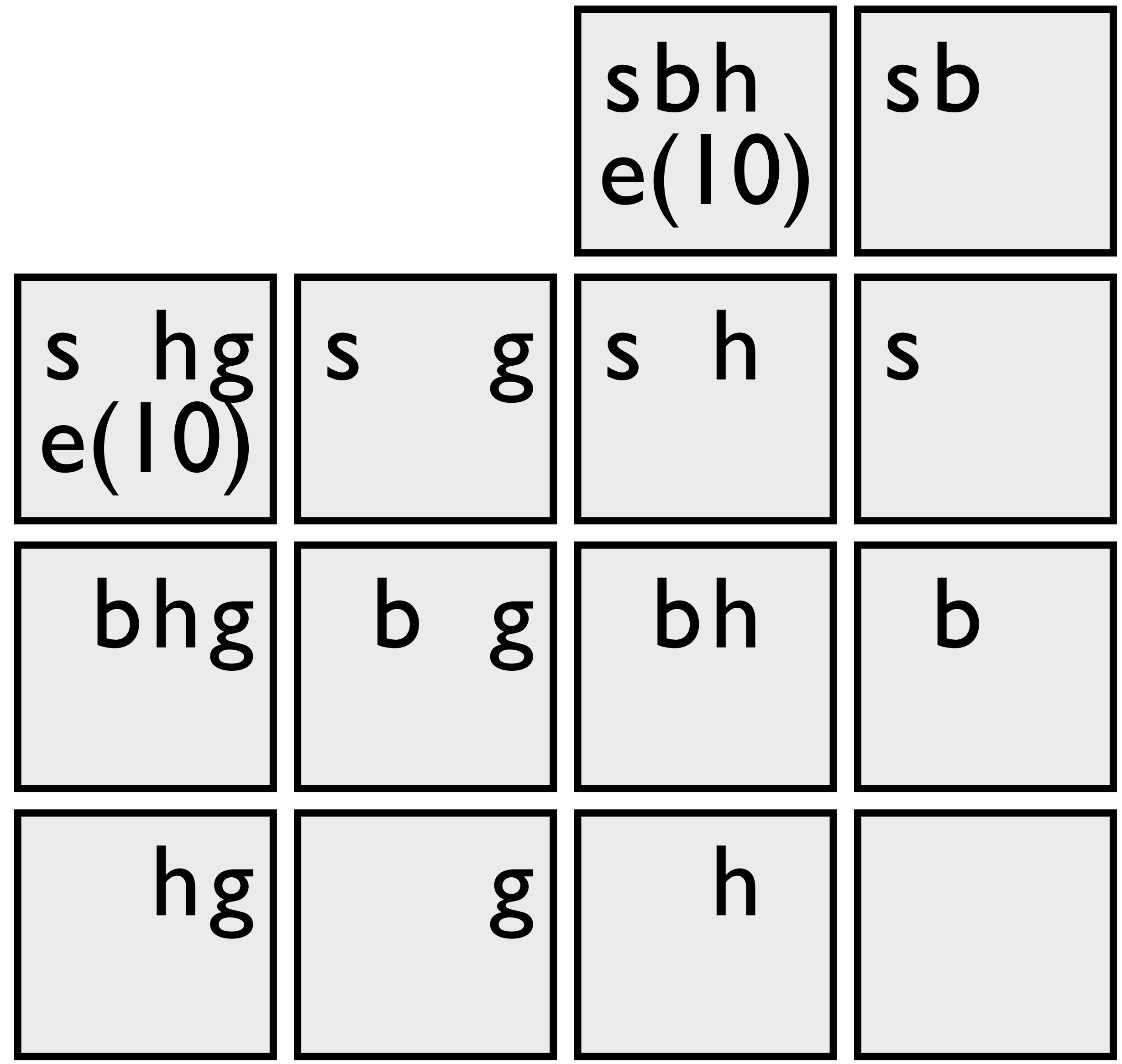


[Fierens et al, PP 12; Shterionov et al]

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).
```



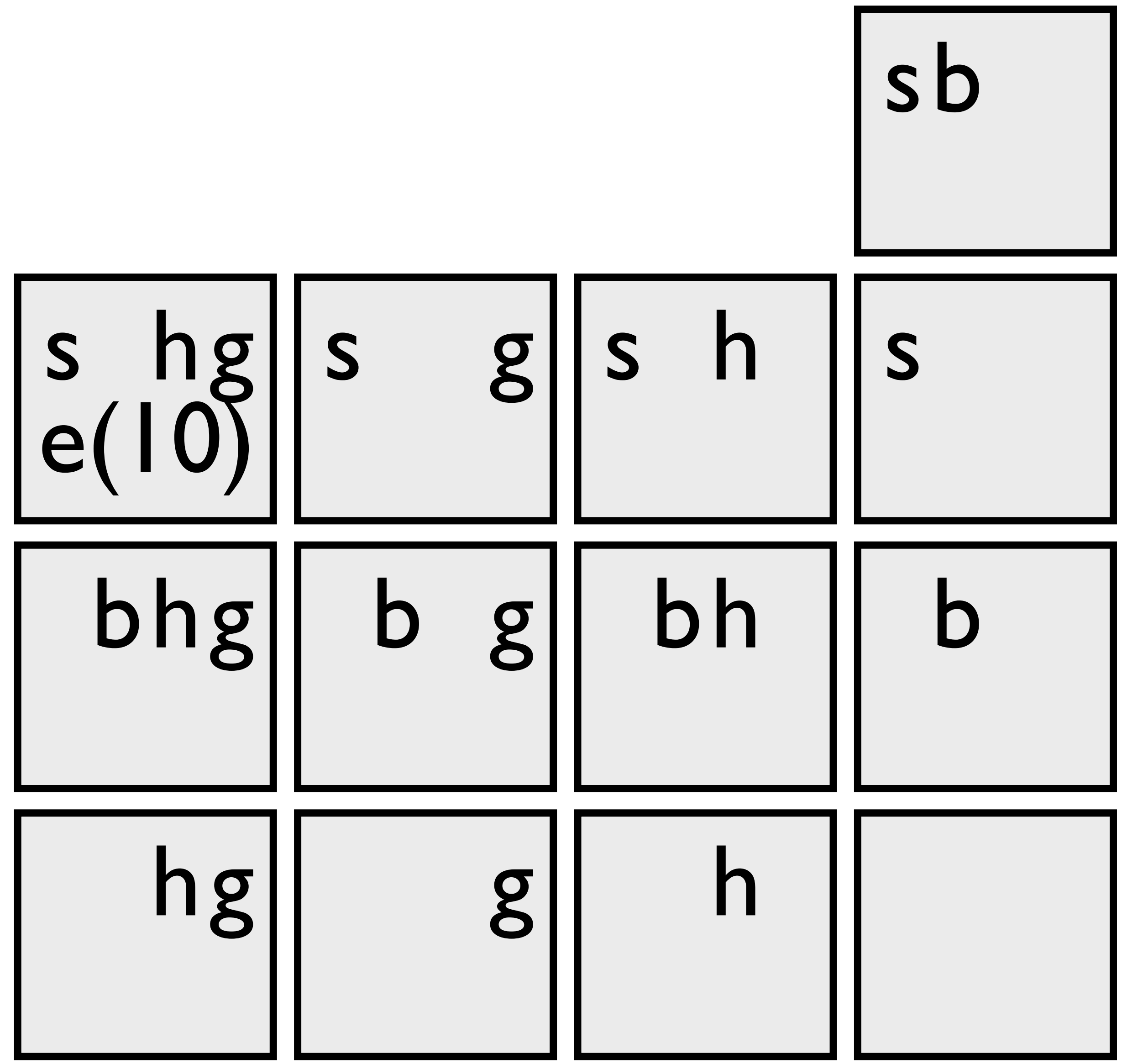
constraints as first-order logic formulas

[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

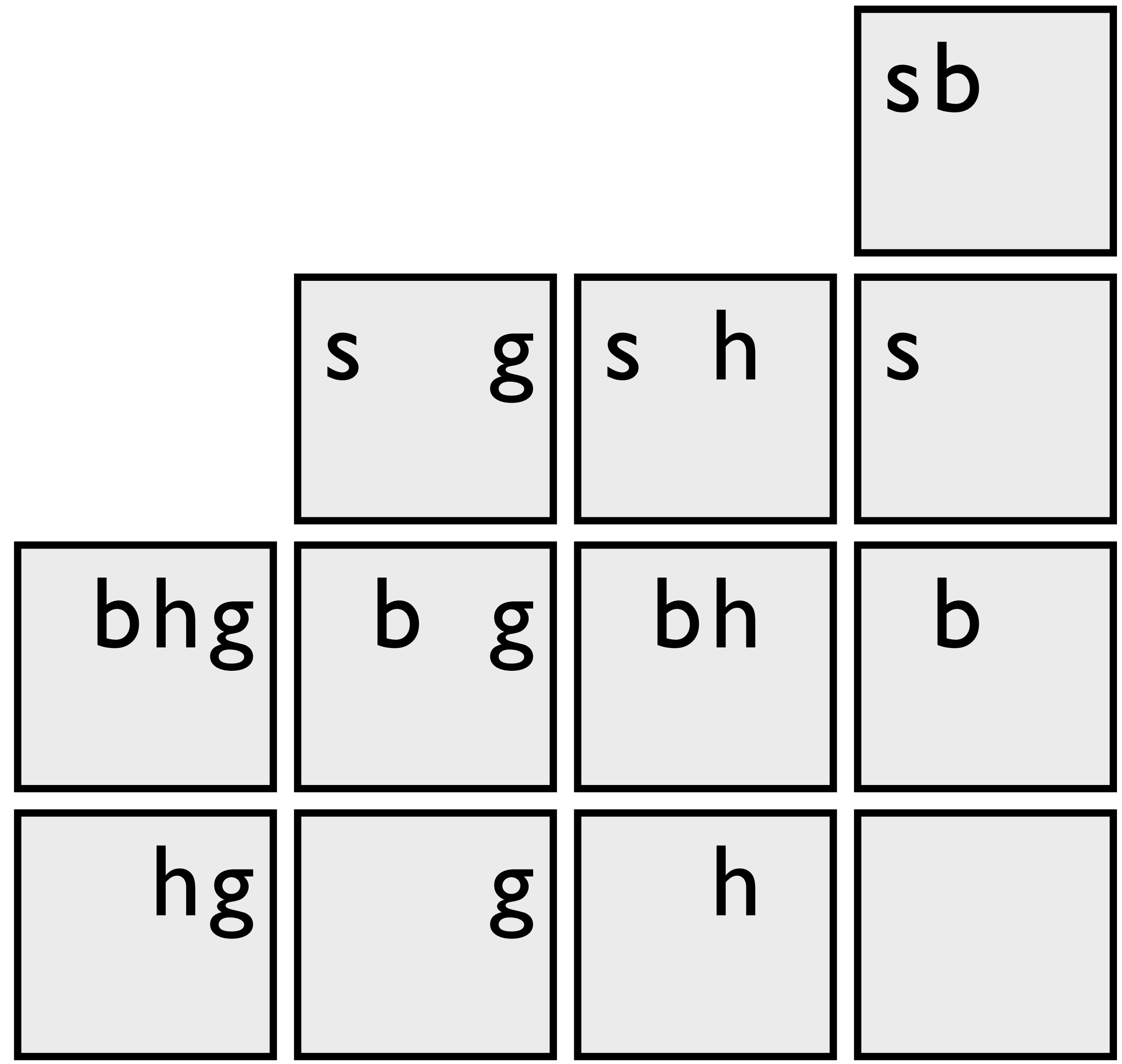


[Fierens et al, PP 12; Shterionov et al]

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

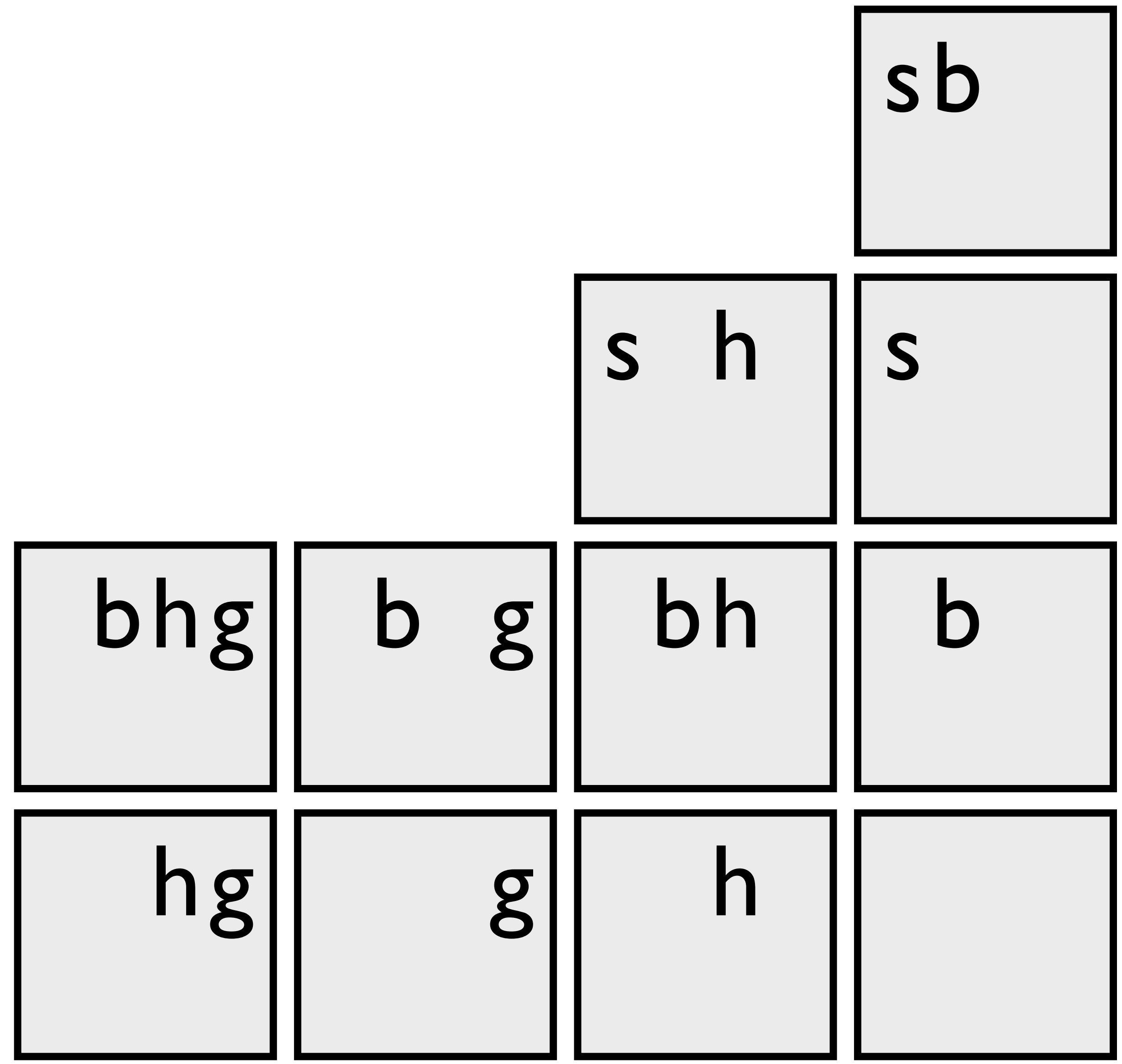


[Fierens et al, PP 12; Shterionov et al]

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

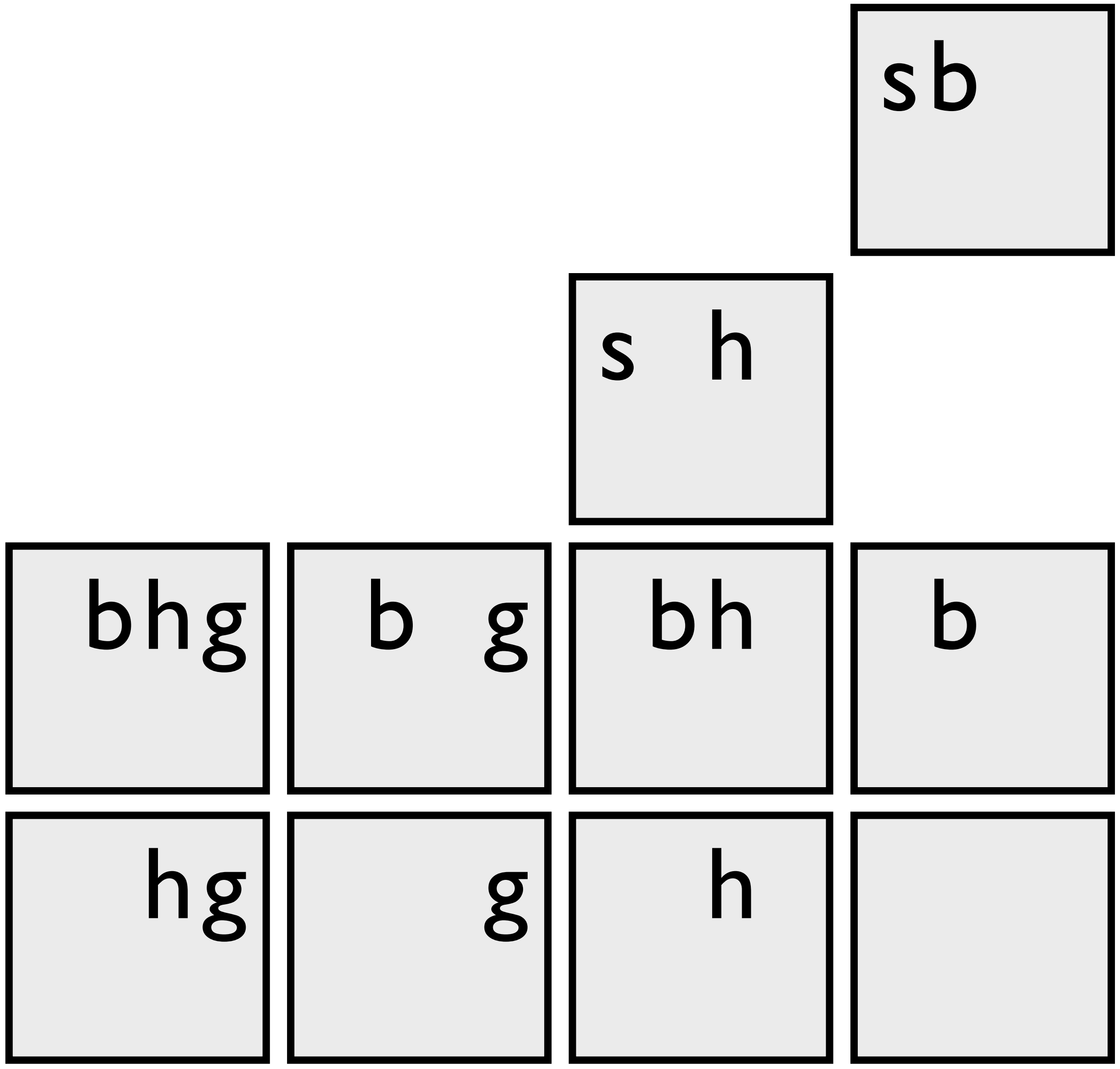


[Fierens et al, PP 12; Shterionov et al]

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).
```

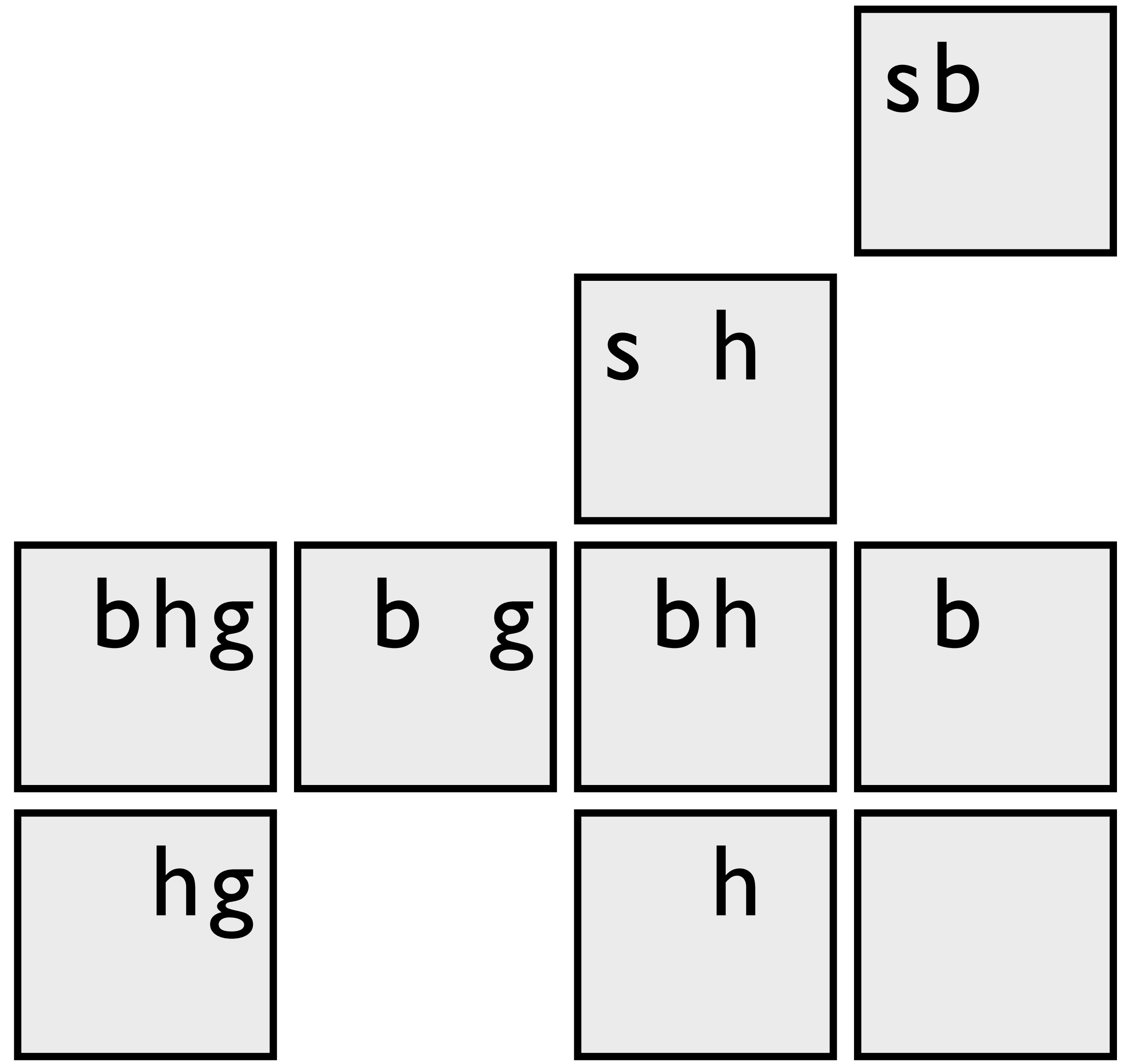
constraints as first-order logic formulas



[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).
```



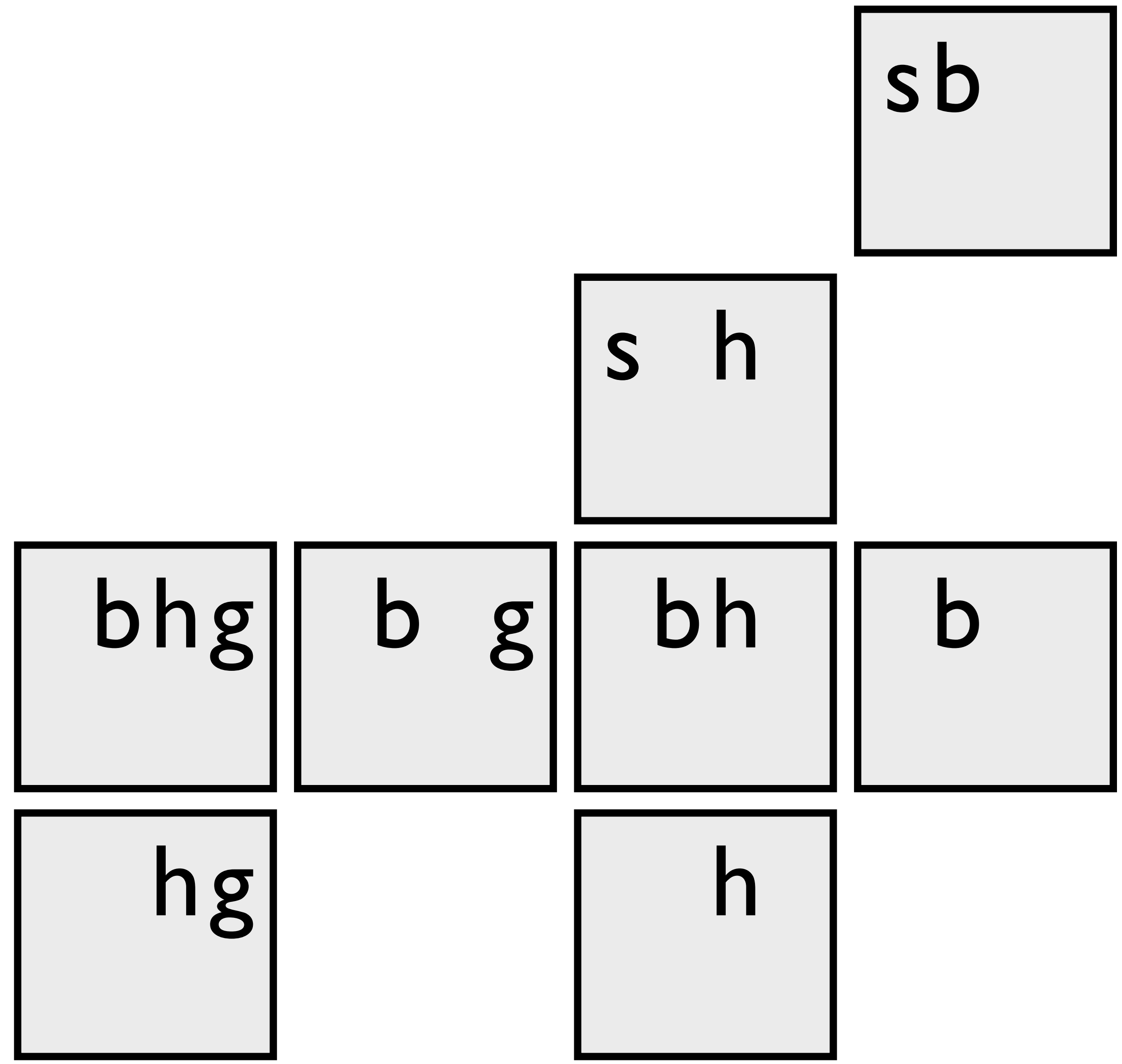
constraints as first-order logic formulas

[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

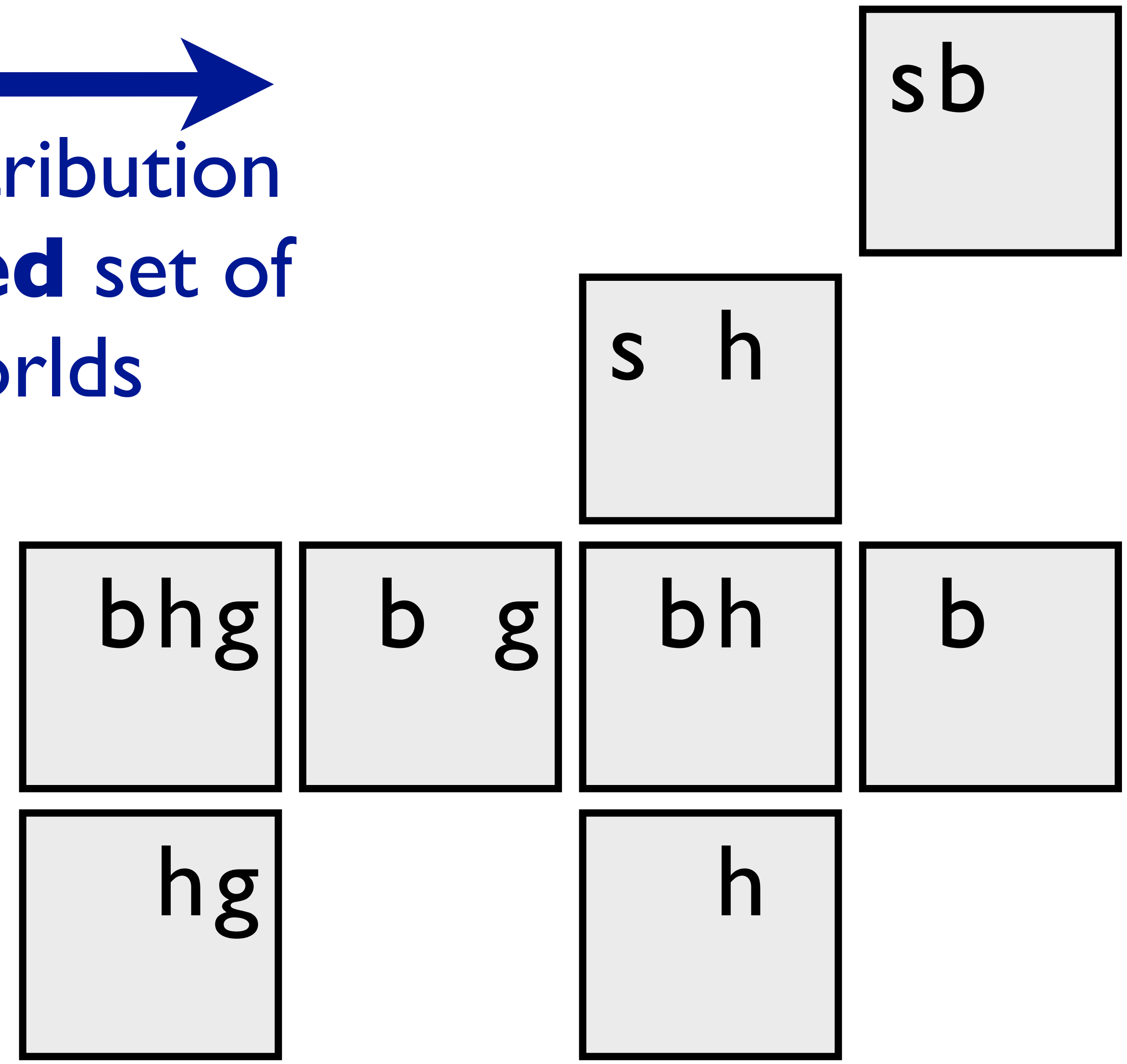


[Fierens et al, PP 12; Shterionov et al]

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).
```

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



constraints as first-order logic formulas

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

Summary

- ProbLog Basics
  - ProbLog = probabilistic choices + logical consequences
  - Inference: MPE, marginals, conditional probabilities
  - Parameter learning from (partial) interpretations
- Selected Topics
  - Upgrading relational learning
  - Dynamics under uncertainty
  - Continuous-valued random variables
  - Decision making
  - Constraints

Getting started

- http://dtai.cs.kuleuven.be/problog
  - interactive tutorial
  - online interface for inference and parameter estimation
  - offline version for download

Maurice Bruynooghe
Bart Demoen
Luc De Raedt
Anton Dries
Daan Fierens
Jason Filippou
Bernd Gutmann
Manfred Jaeger
Gerda Janssens
Kristian Kersting
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

Maurice Bruynooghe
Bart Demoen
Luc De Raedt
Anton Dries
Daan Fierens
Jason Filippou
Bernd Gutmann
Manfred Jaeger
Gerda Janssens
Kristian Kersting
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

Thanks!

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