Probabilistic Programming

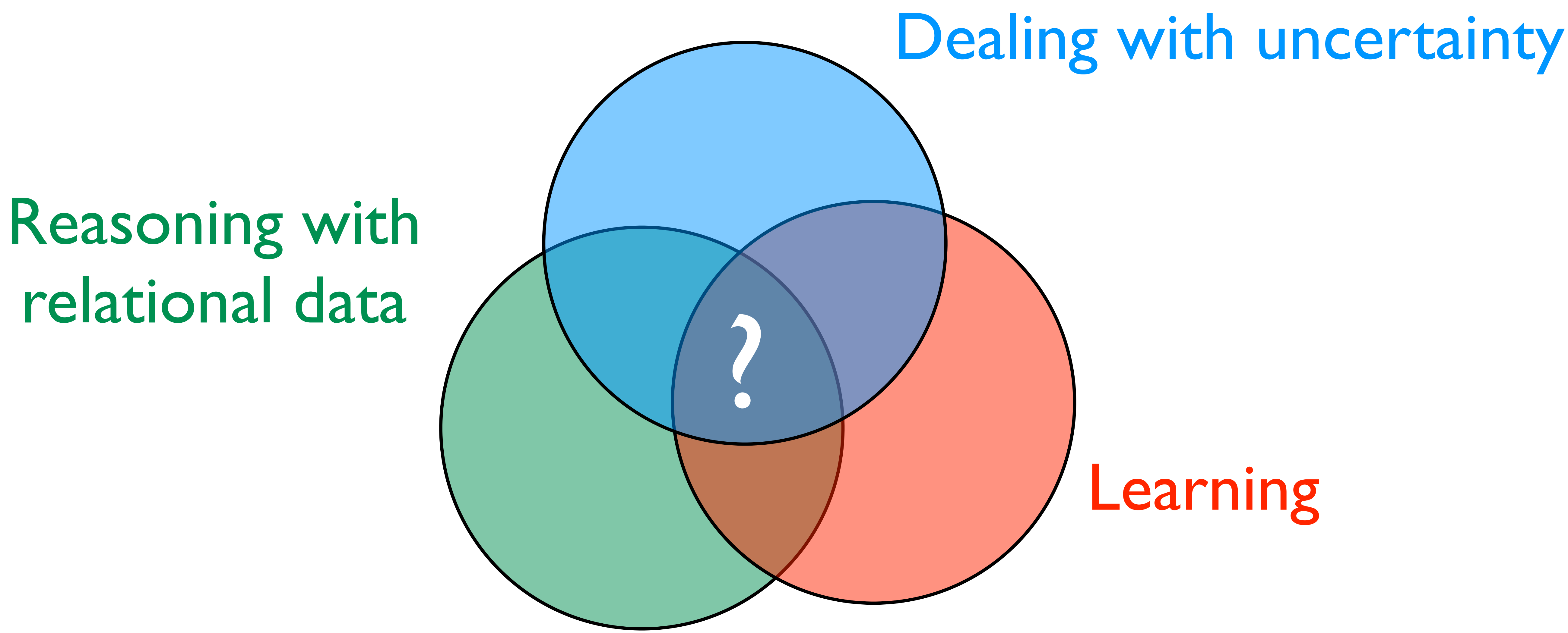
Luc De Raedt

with many slides from Angelika Kimmmig & Guy Van den Broeck

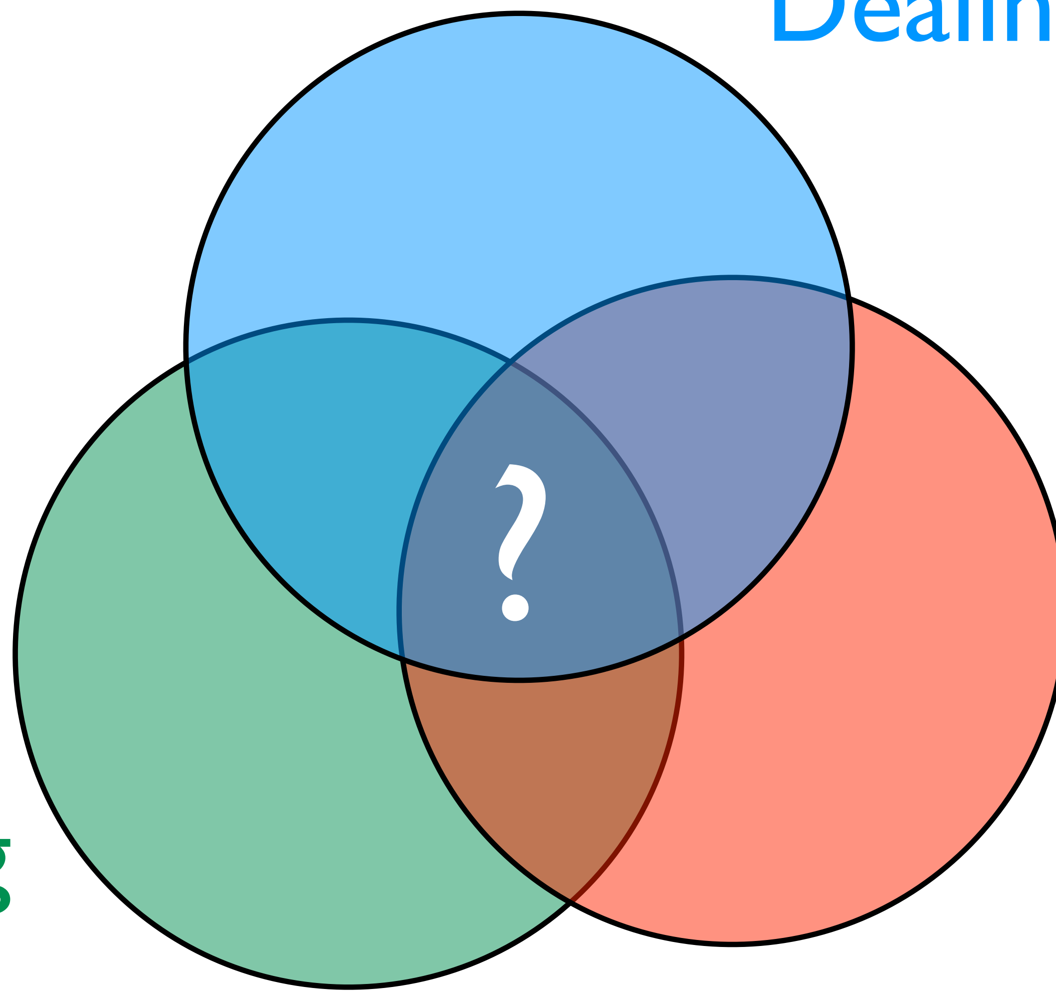
DTAI Machine Learning

KU LEUVEN

A key question in AI:



A key question in AI:

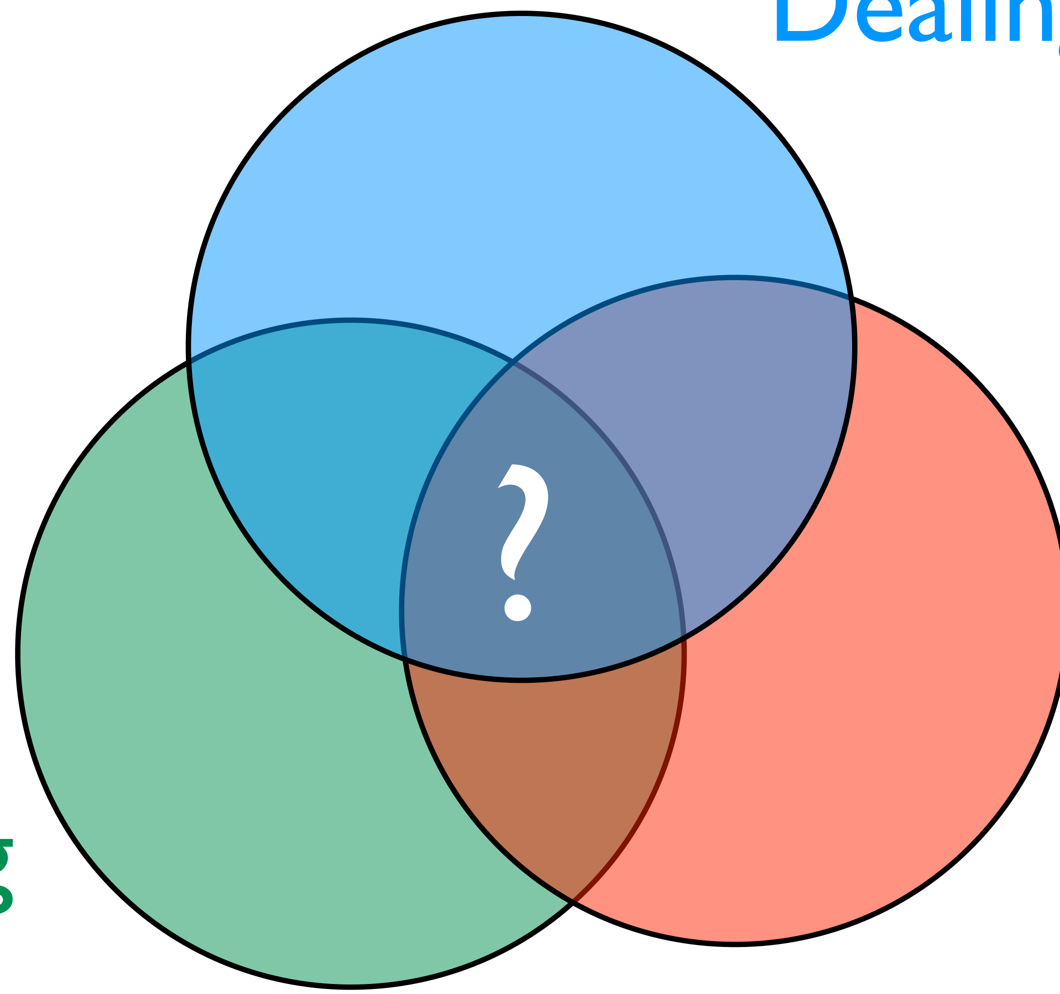


- logic
- databases
- programming
- ...

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...

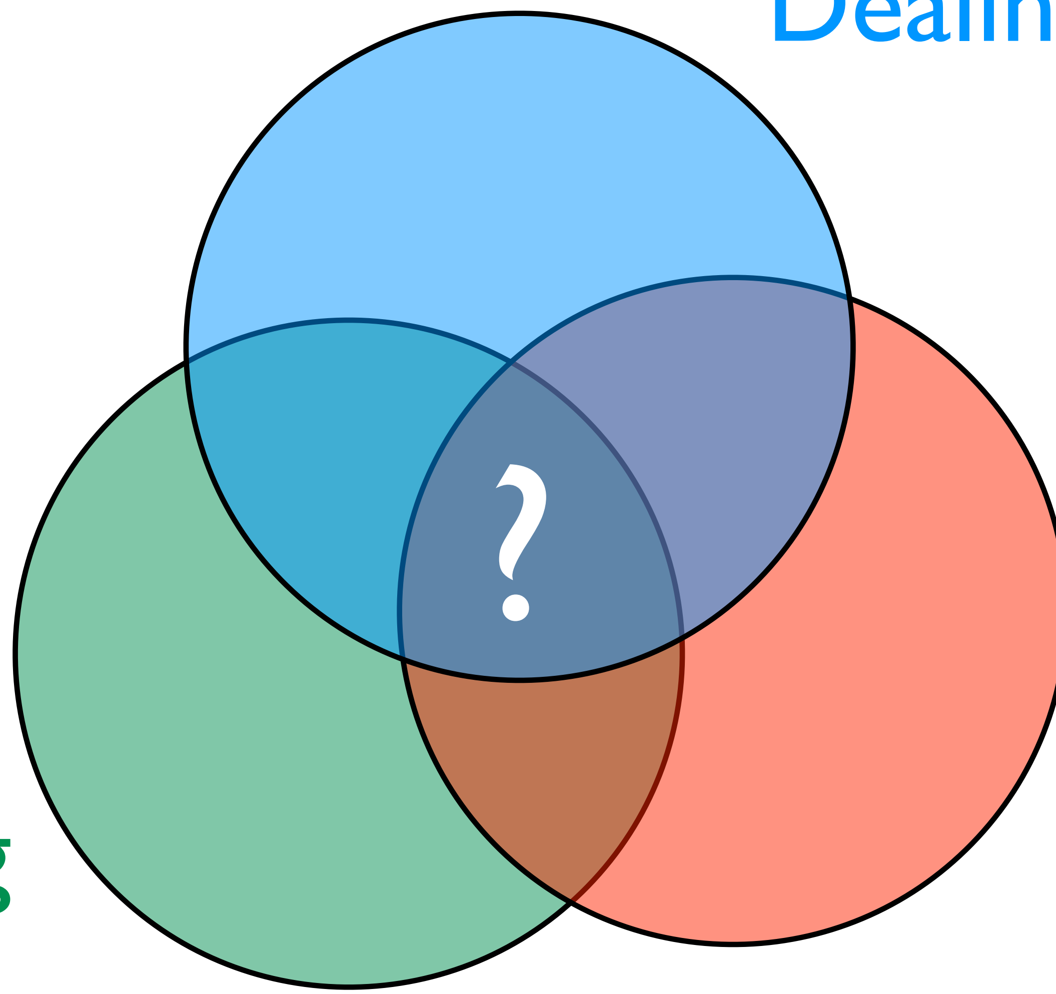


Learning

A key question in AI:

Reasoning with relational data

- logic
- databases
- programming
- ...



Dealing with uncertainty

- probability theory
- graphical models
- ...

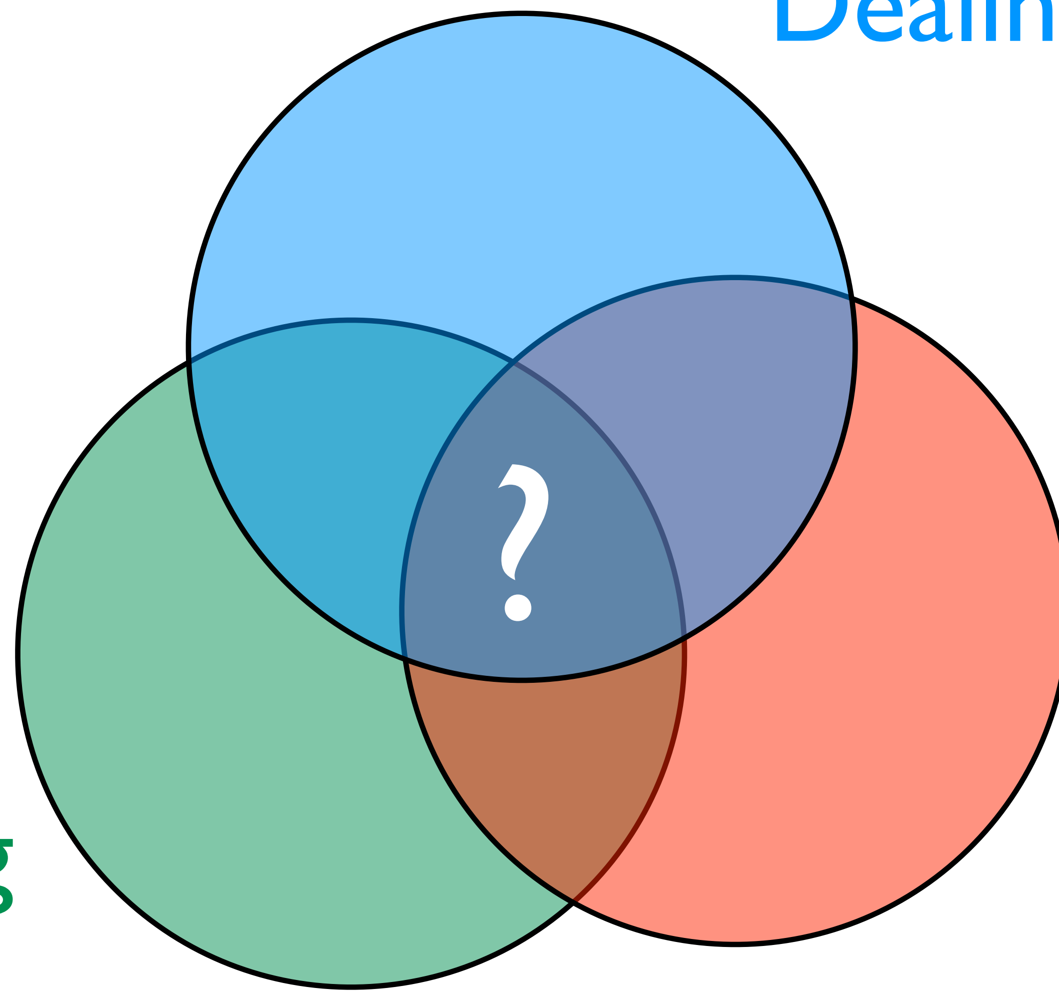
Learning

- parameters
- structure

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...

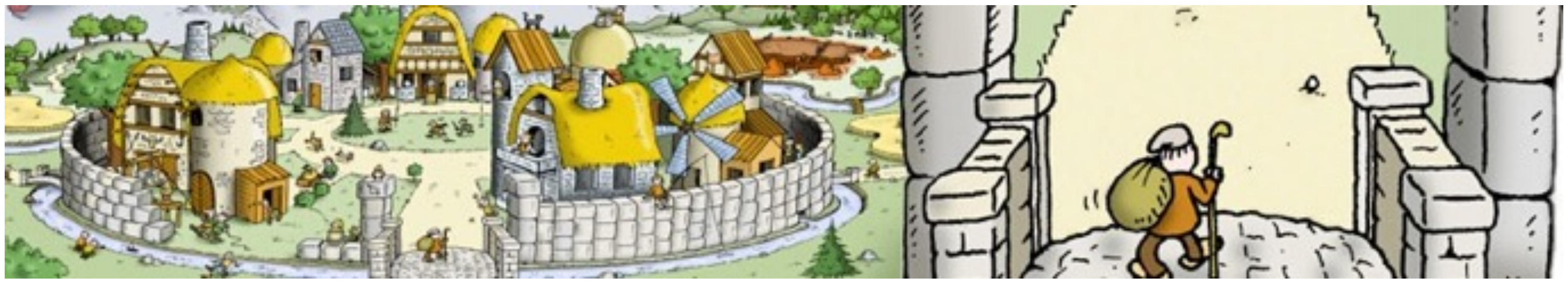


Learning
- parameters
- structure

Statistical relational learning
& Probabilistic Programming

The need for relations

Dynamics: Evolving Networks



- *Travian*: A massively multiplayer real-time strategy game
  - Commercial game run by TravianGames GmbH
  - ~3.000.000 players spread over different “worlds”
  - ~25.000 players in one world

[Thon et al. ECML 08]

World Dynamics

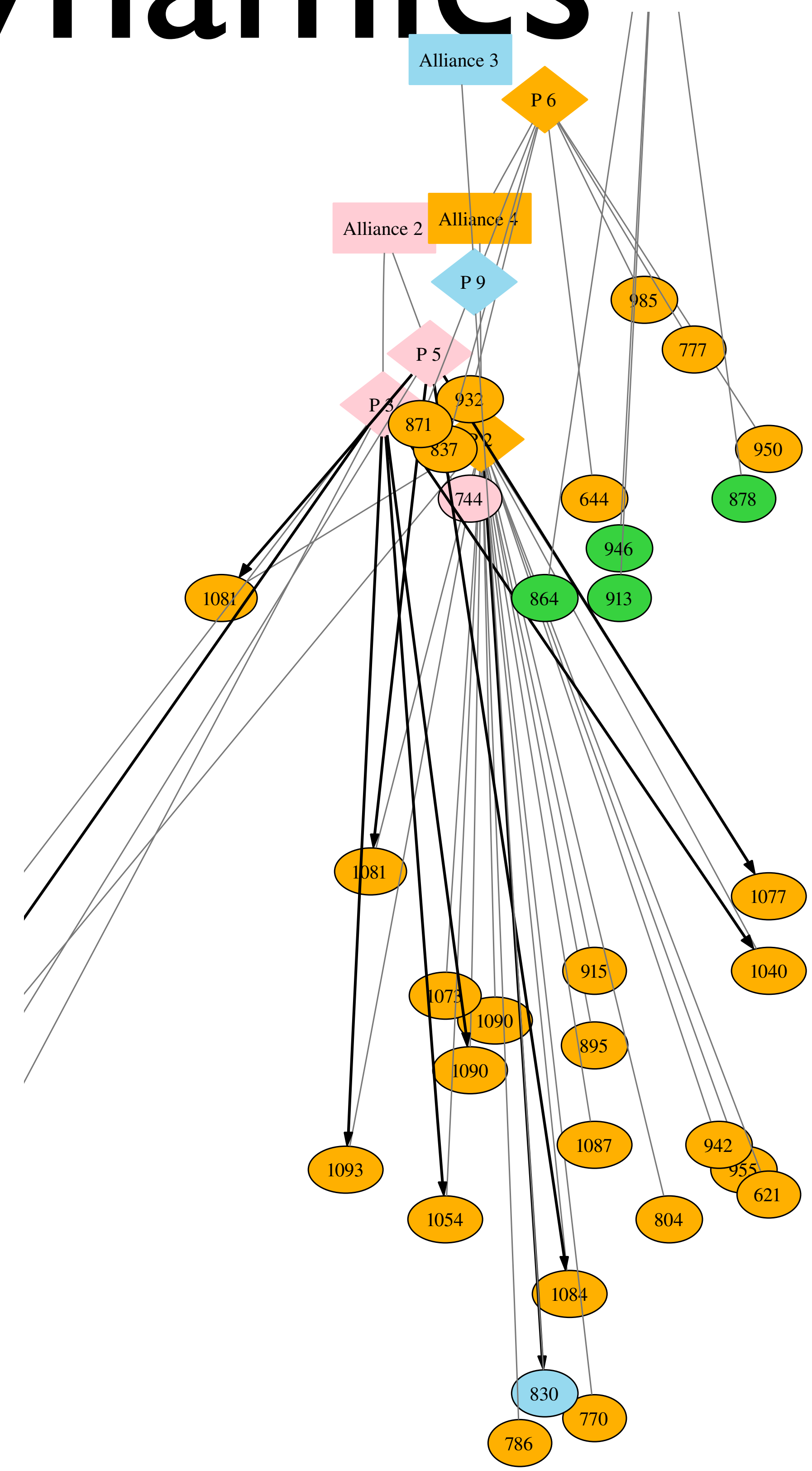
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

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

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

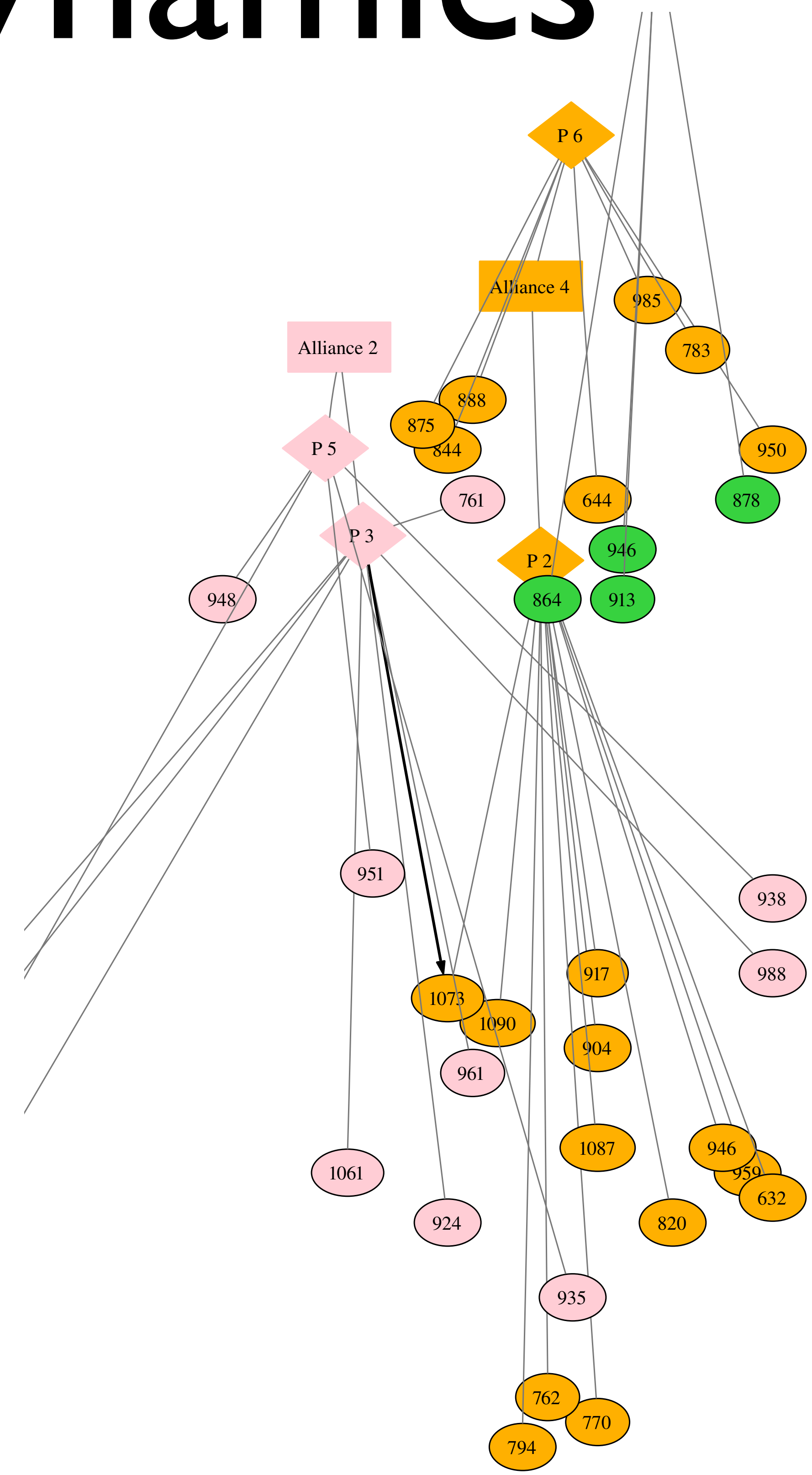
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

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

[Thon, Landwehr, De Raedt, ECML08]



World Dynamics

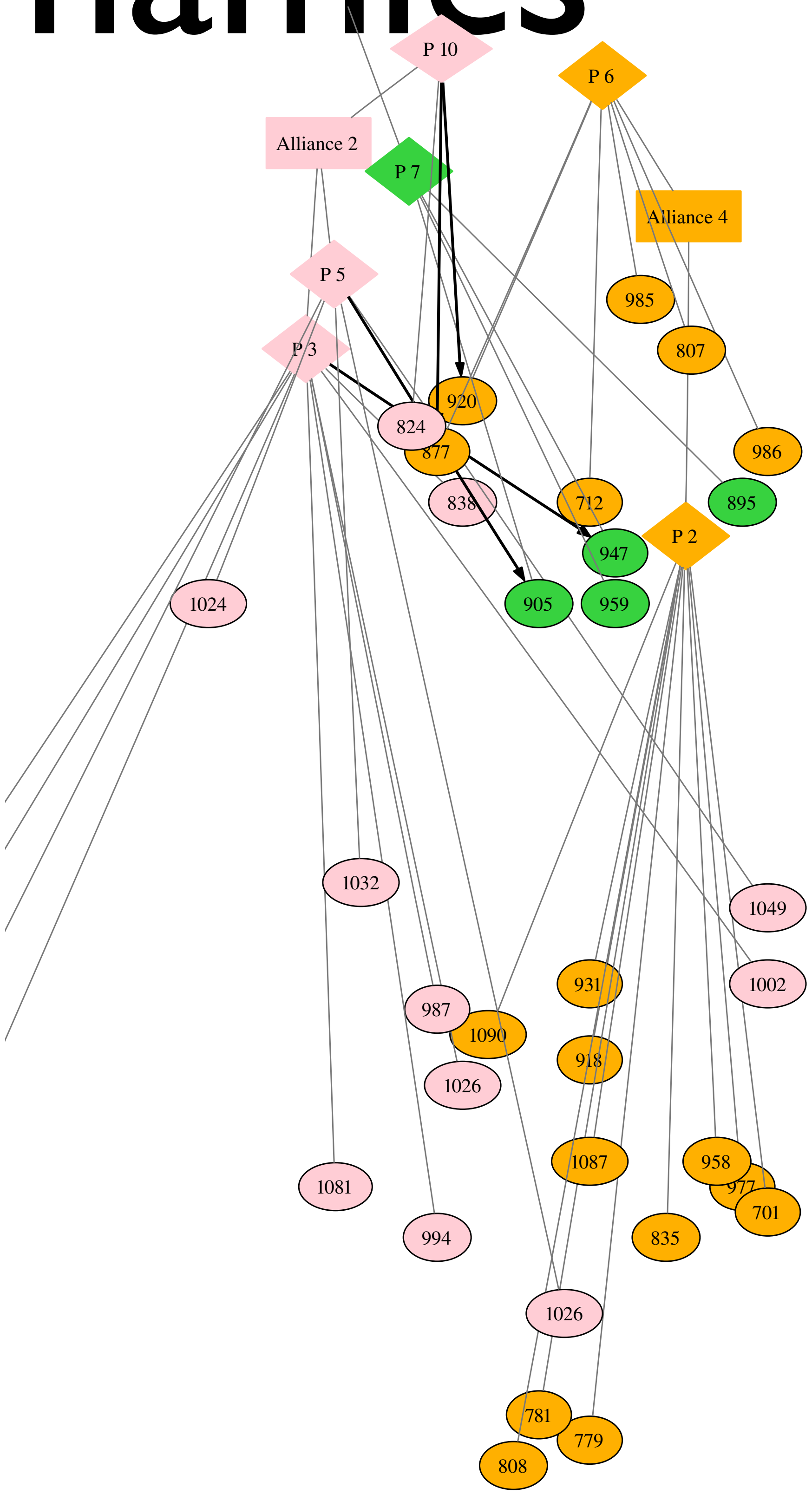
Fragment of world with

~10 alliances
~200 players
~600 cities

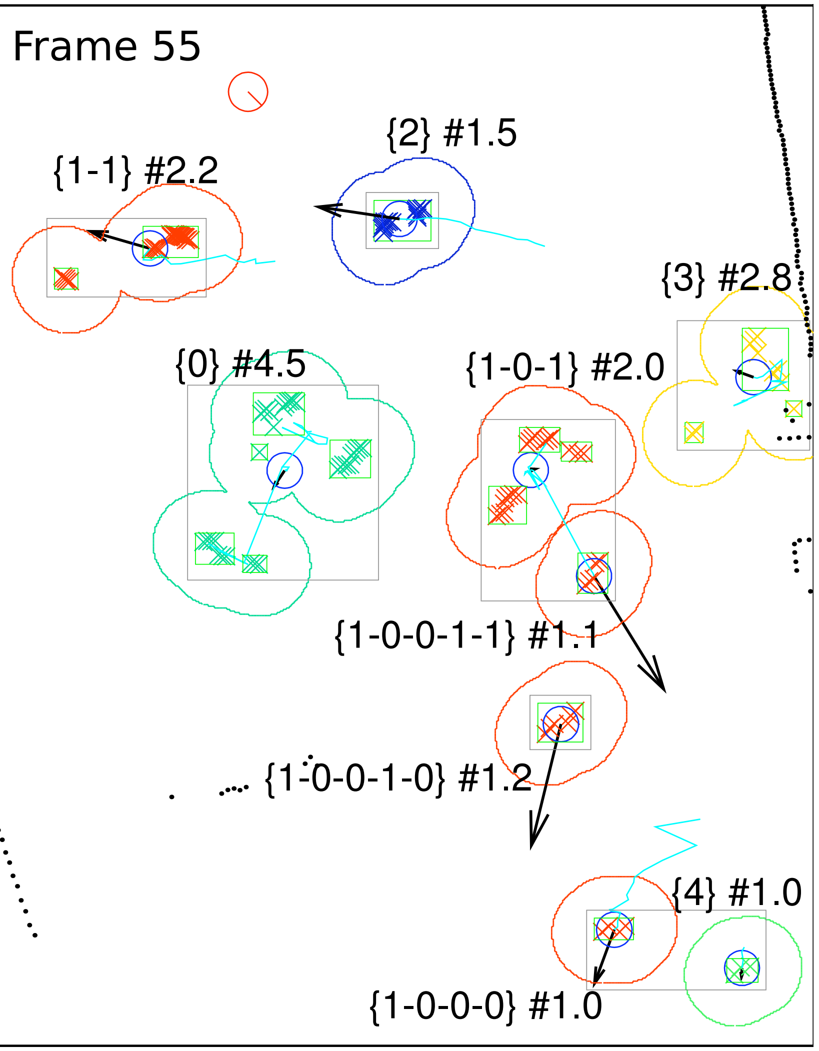
alliances color-coded

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

[Thon, Landwehr, De Raedt, ECML08]



Analyzing Video Data



- Track people or objects over time? Even if temporarily hidden?

[Skarlatidis et al, TPLP 14; Nitti et al, IROS 13, ICRA 14]

- Recognize activities?
- Infer object properties?

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

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

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

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

Example: Information Extraction

Recently-Learned Facts

Refresh

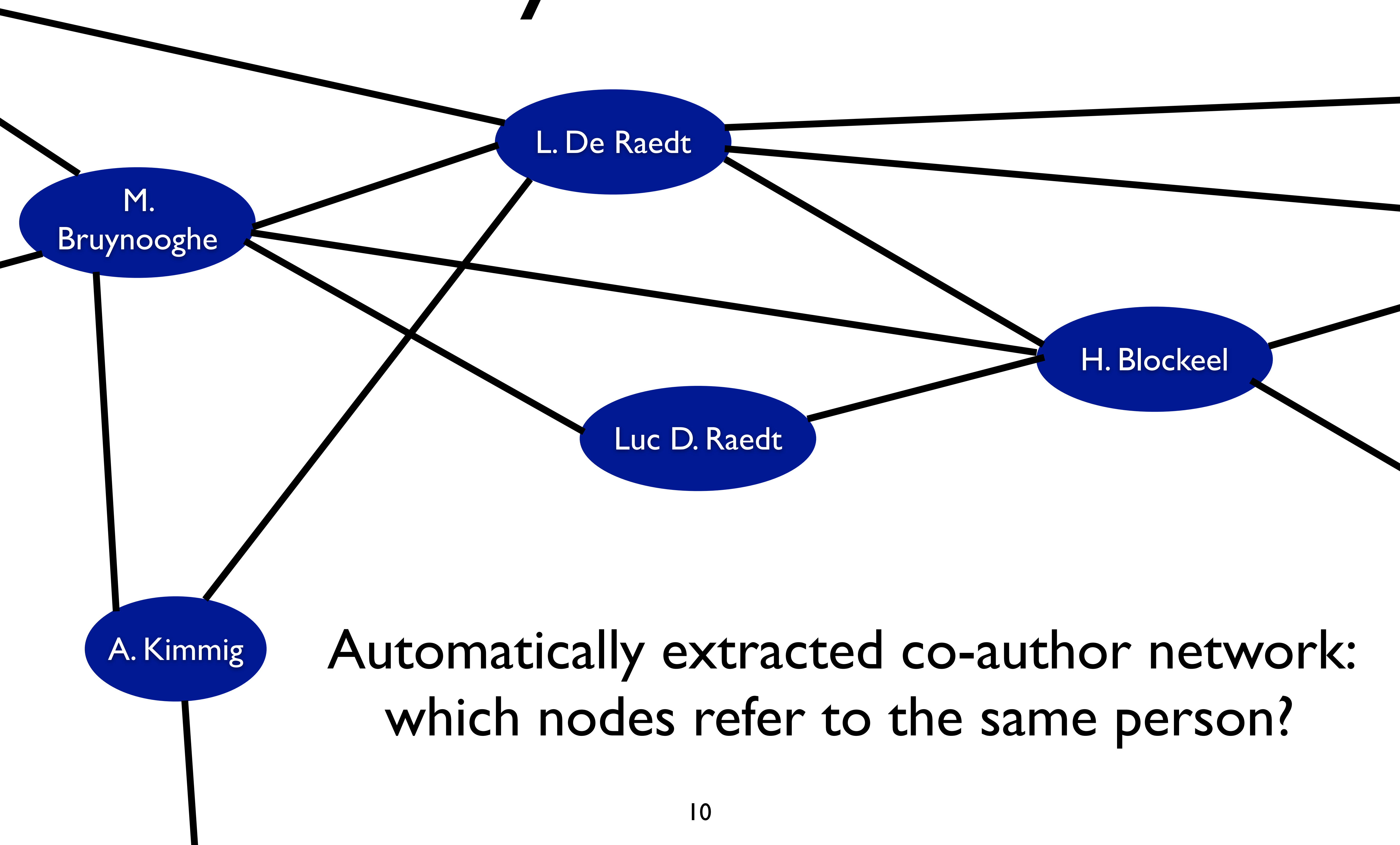
instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson 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

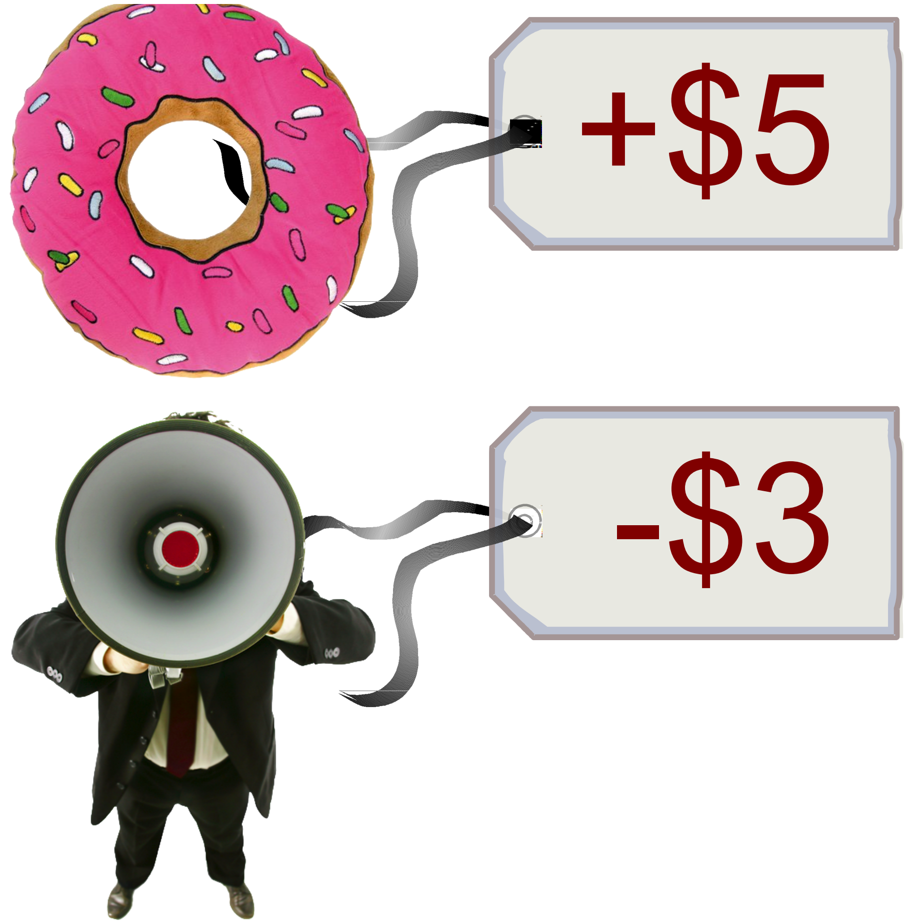
NELL: http://rtw.ml.cmu.edu/rtw/

Entity Resolution

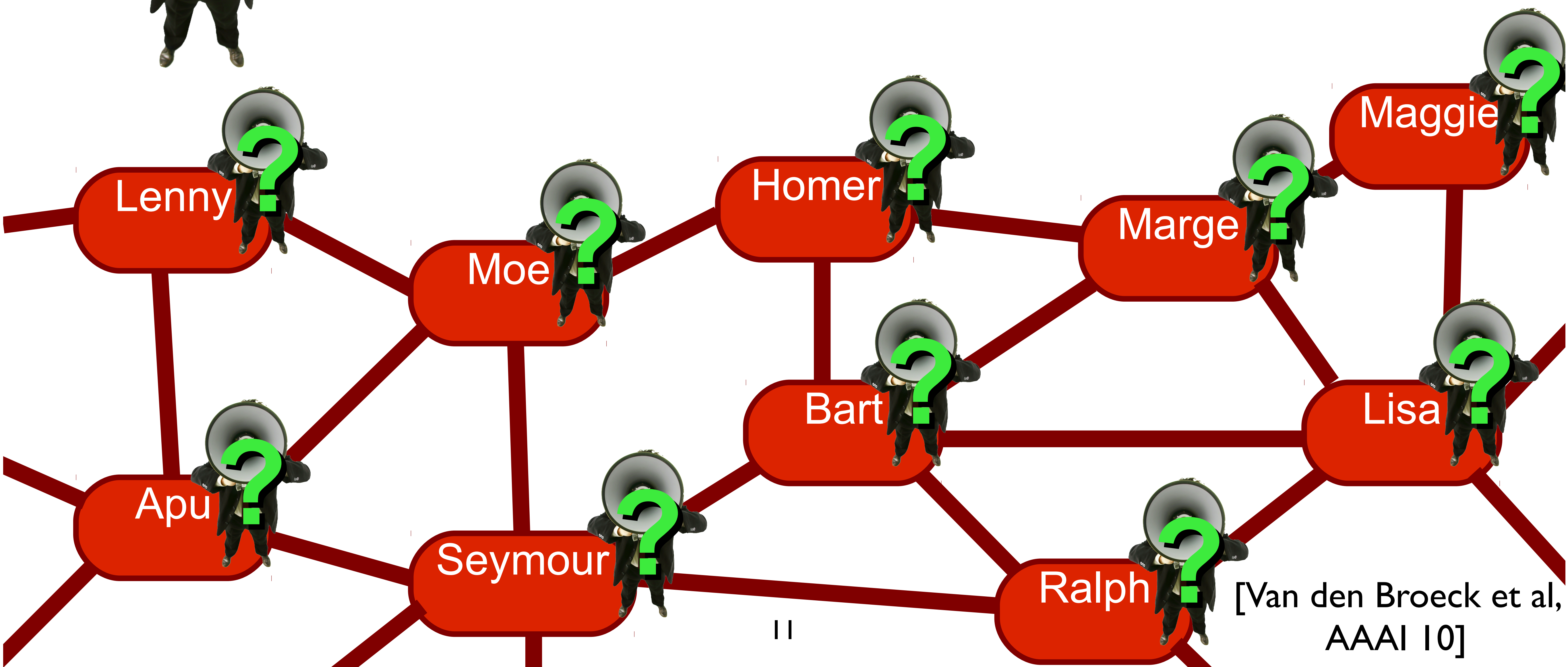


Automatically extracted co-author network: which nodes refer to the same person?

Viral Marketing

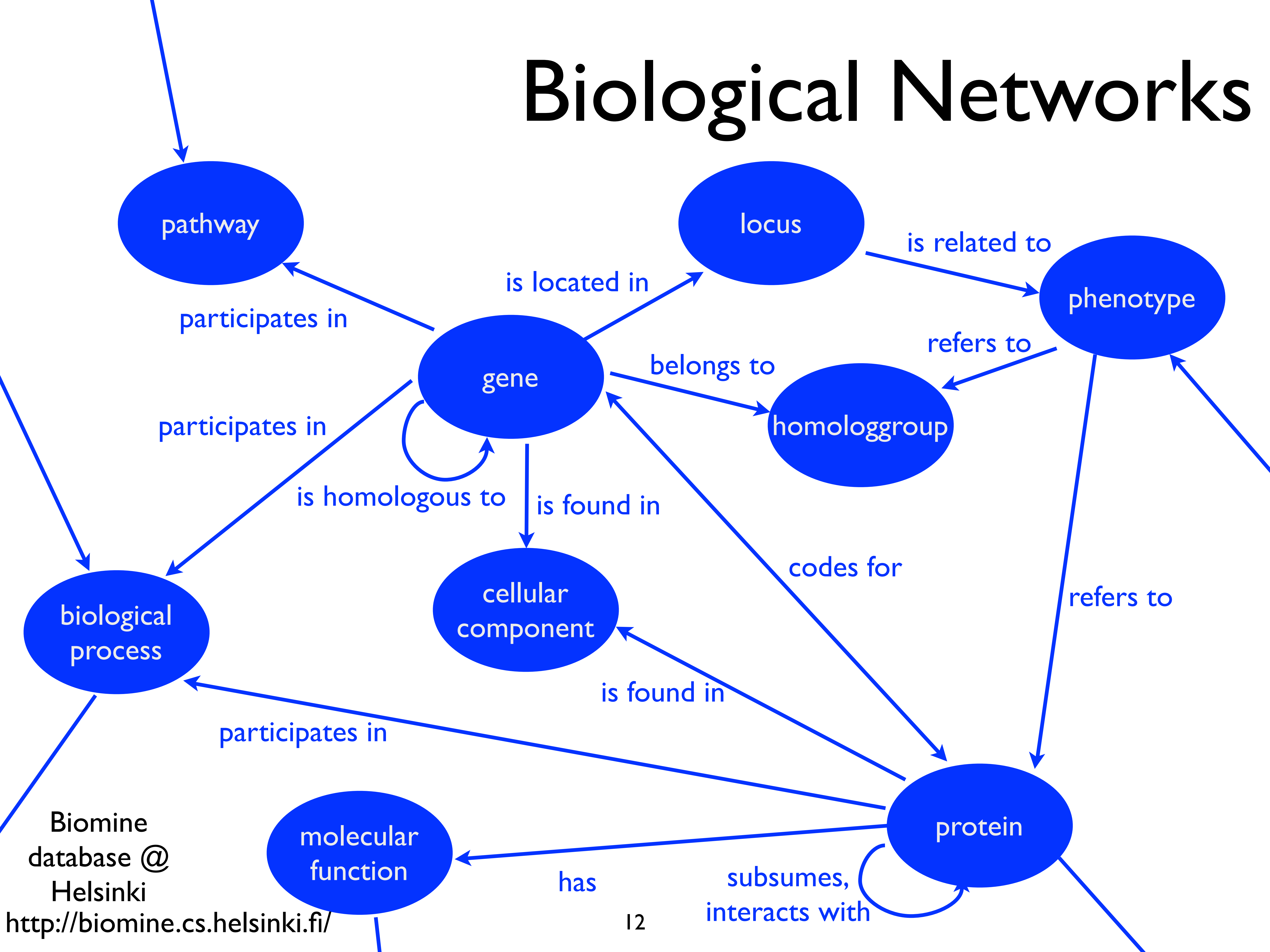


Which advertising strategy maximizes expected profit?



[Van den Broeck et al, AAAI 10]

Biological Networks



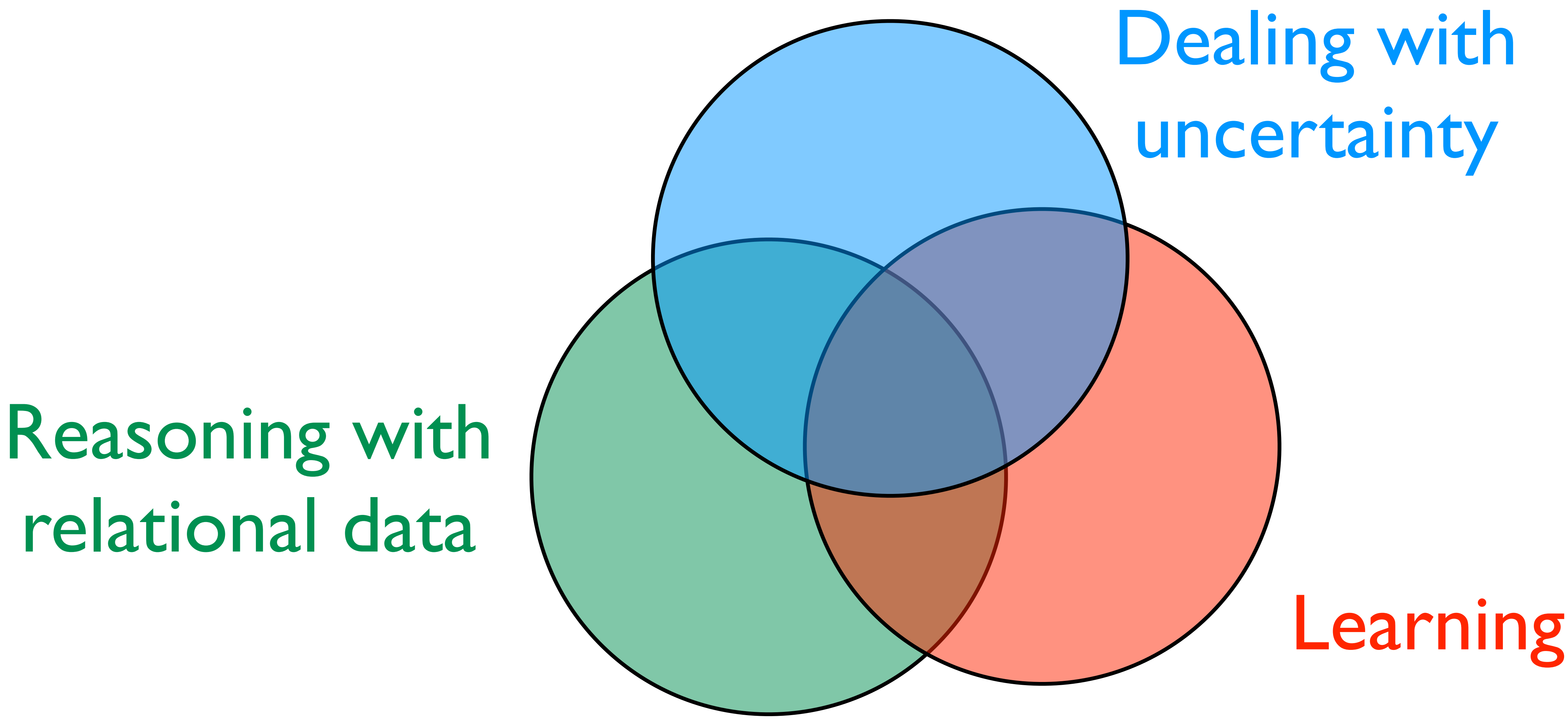
Biomine database @ Helsinki
http://biomine.cs.helsinki.fi/

This requires dealing with

- Structured environments
  - objects, and
  - relationships amongst them
- and possibly
  - using background knowledge
- cope with uncertainty
- learn from data

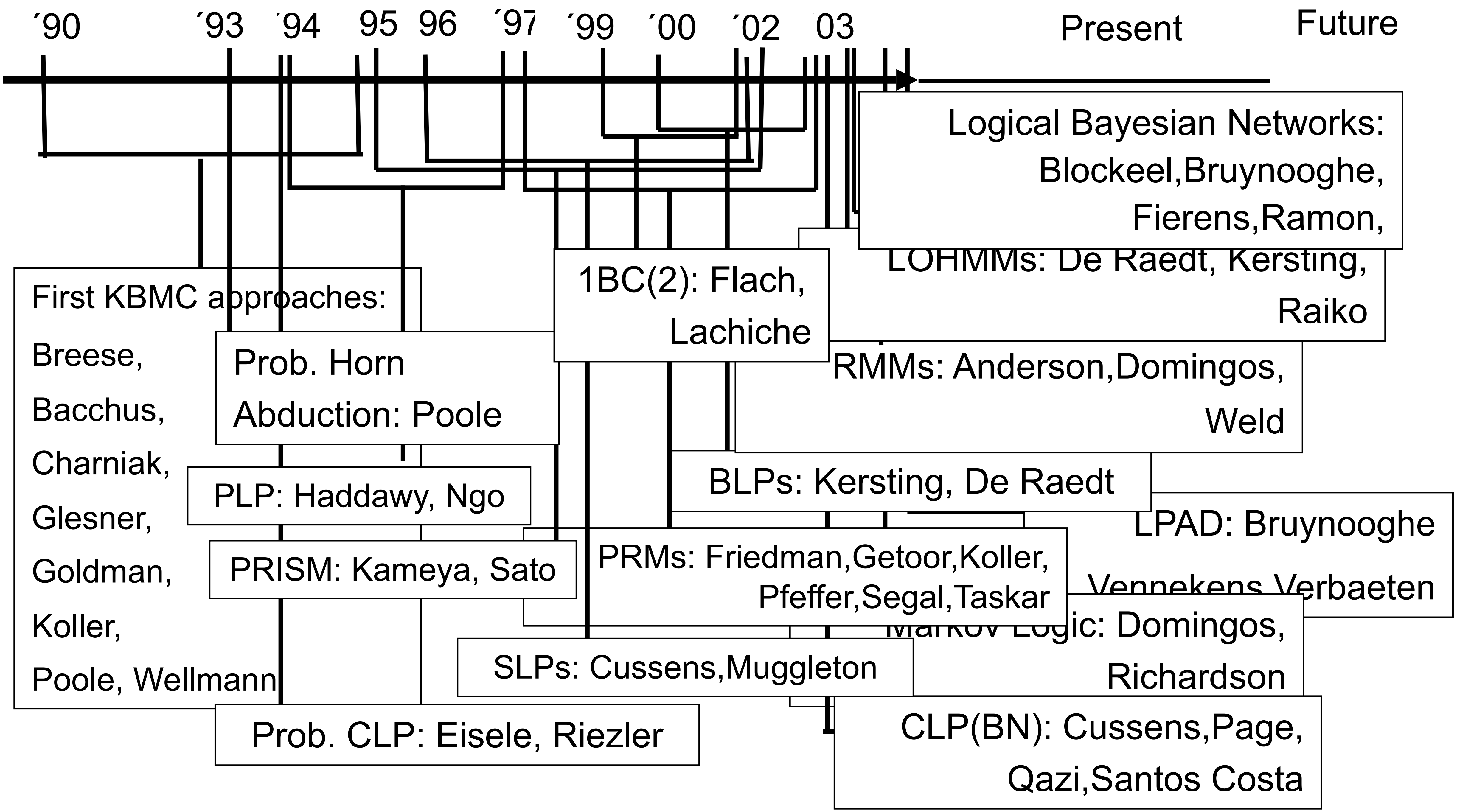
Statistical Relational Learning
Probabilistic Programming

Common theme

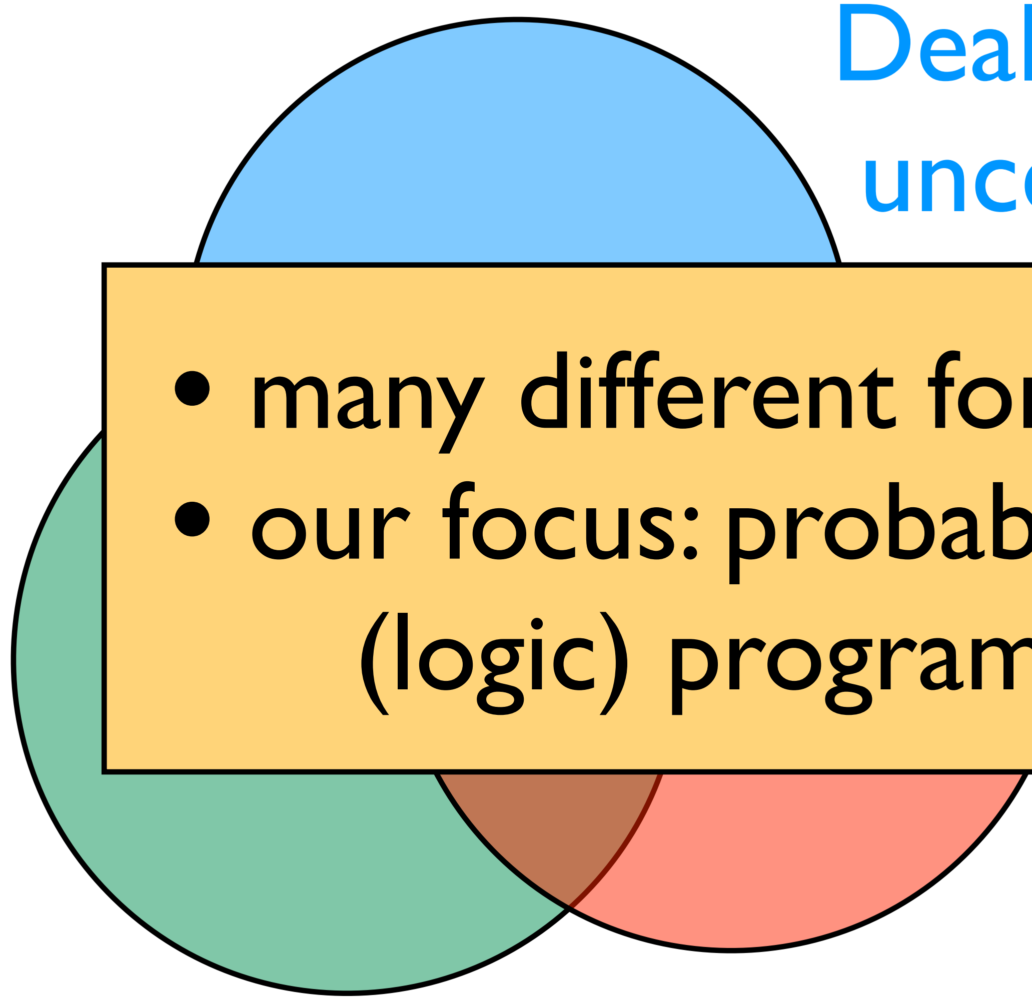


Statistical relational learning
& Probabilistic Programming, ...

Some formalisms



Common theme

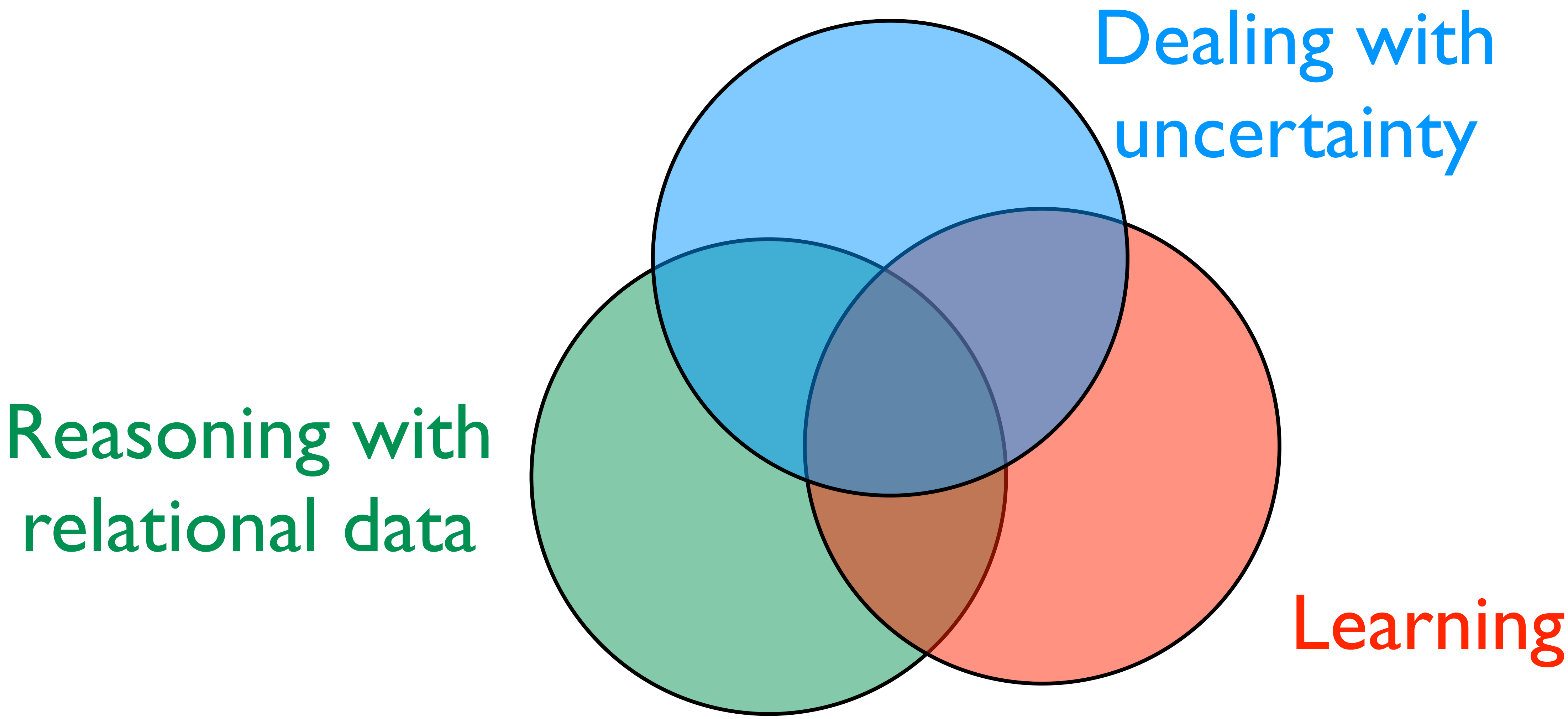


- many different formalisms
- our focus: probabilistic (logic) programming

Statistical relational learning
& Probabilistic Programming, ...

ProbLog

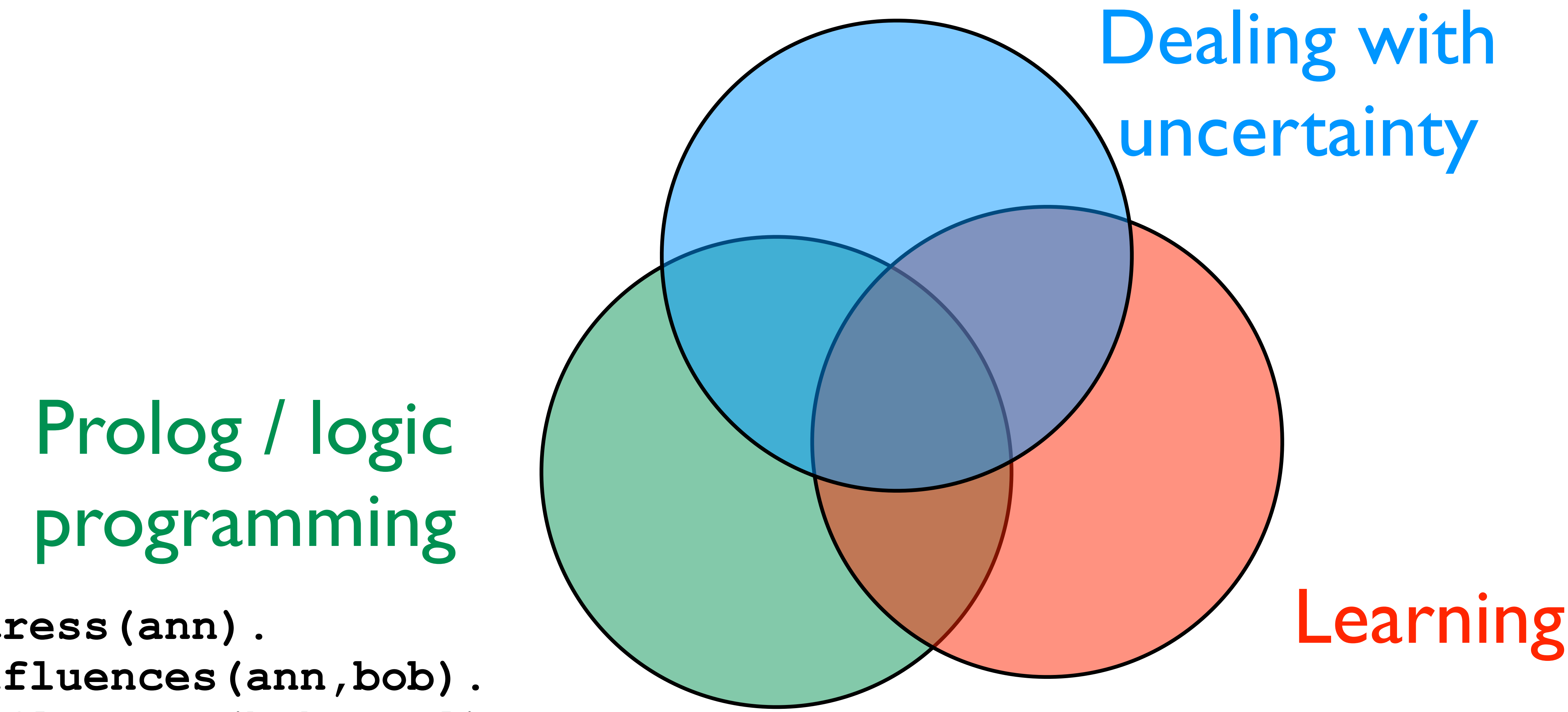
probabilistic Prolog



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

ProbLog

probabilistic Prolog

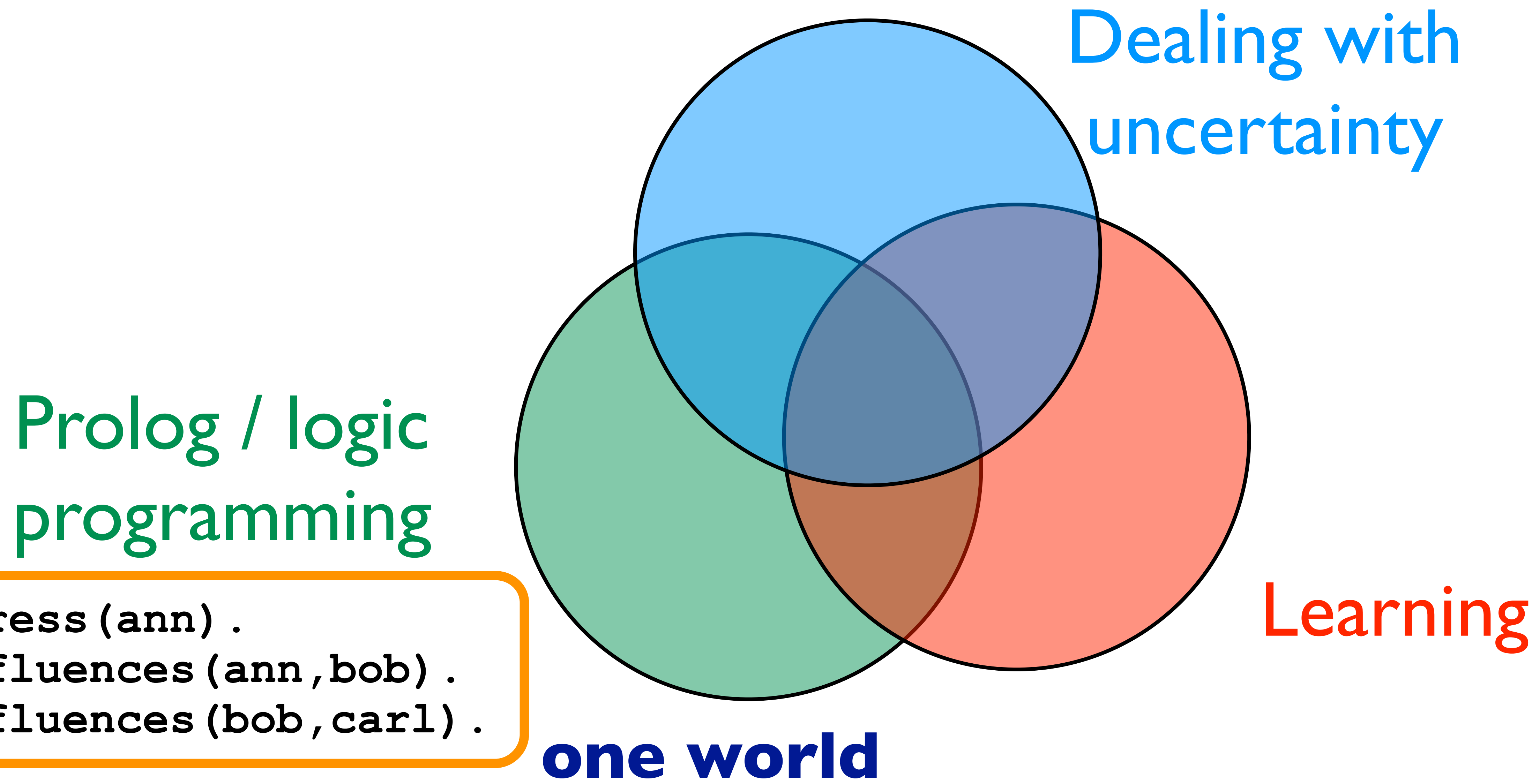



```
stress(ann).
influences(ann,bob).
influences(bob,carl).

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


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

ProbLog

probabilistic Prolog




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

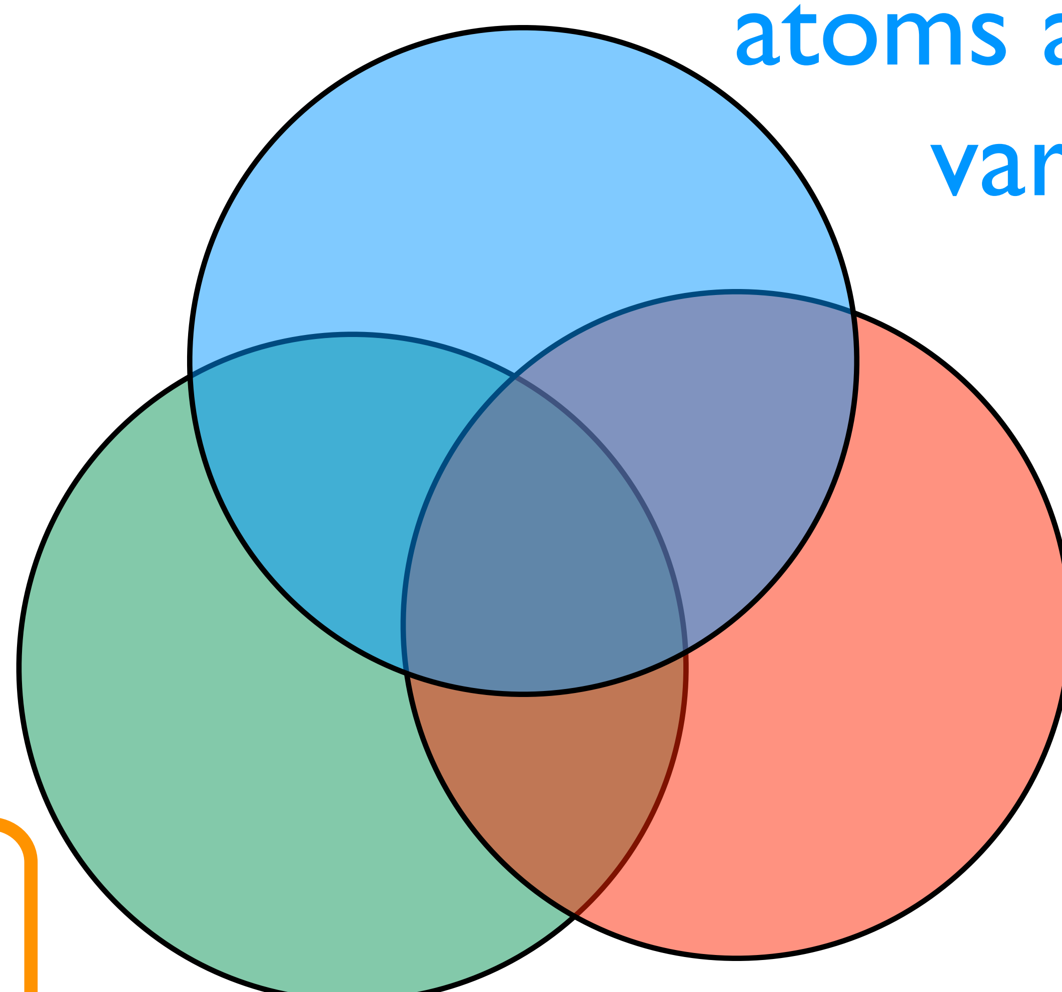
```
smokes(X) :- stress(X).
smokes(X) :-
     influences(Y,X), smokes(Y).
```


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

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).
```


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

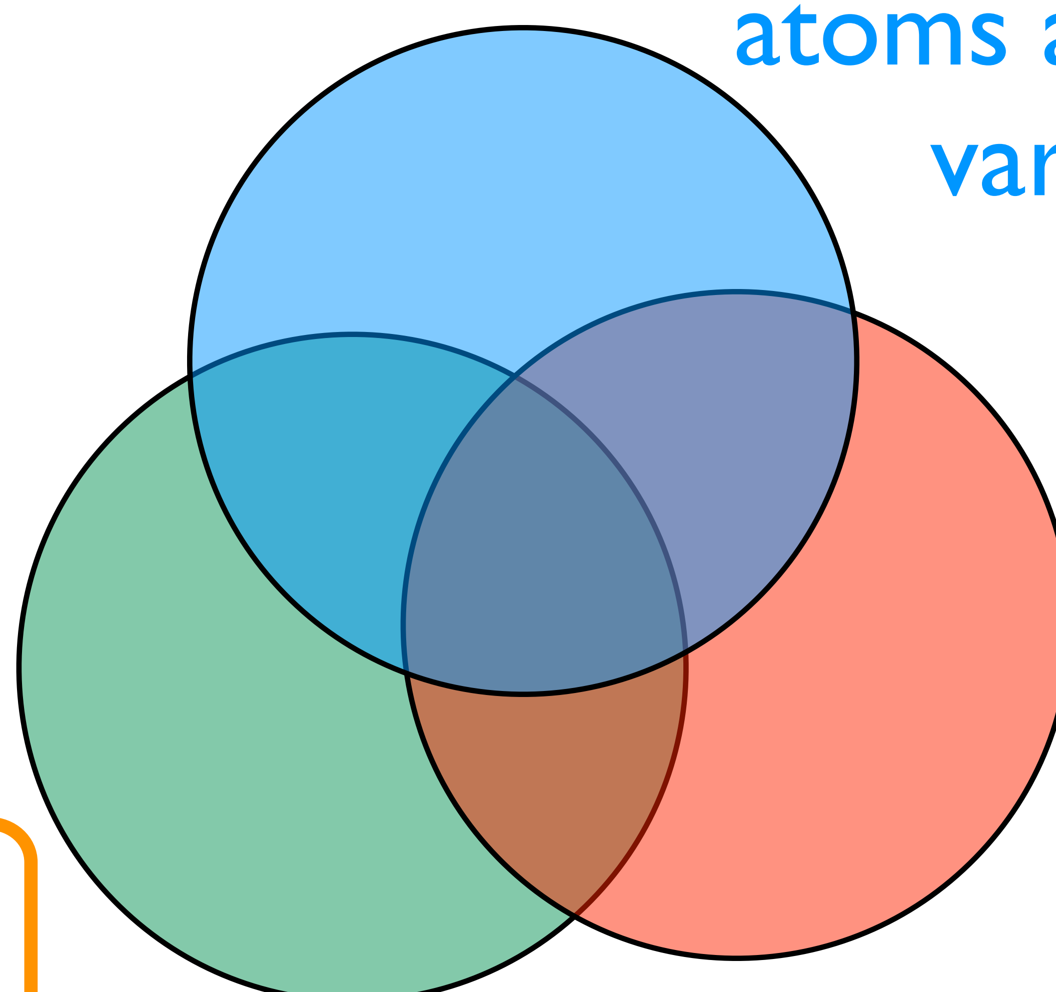
ProbLog

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).
```


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

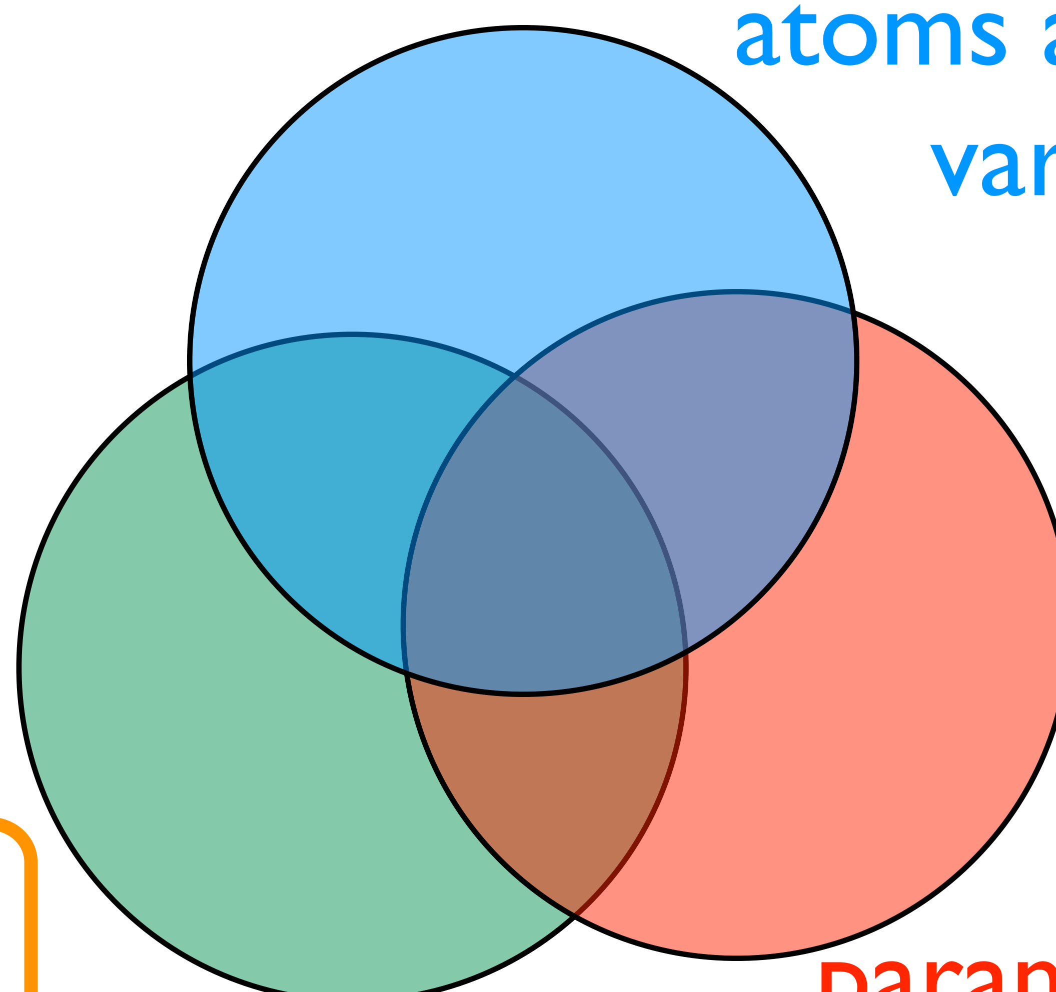
ProbLog

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

```
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

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

Probabilistic Logic Programming

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

e.g., PRISM, ICL, ProbLog, LPADs, CP-logic, ...

multi-valued switches

probabilistic alternatives

probabilistic facts

annotated disjunctions

causal-probabilistic laws

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions
- Markov Logic another representative of SRL

... with some detours on the way

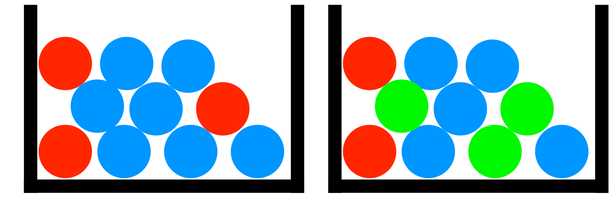
Roadmap

- **Modeling** (ProbLog and Church, another representative of PP)
- Inference
- Learning
- Dynamics and Decisions
- Markov Logic another representative of SRL

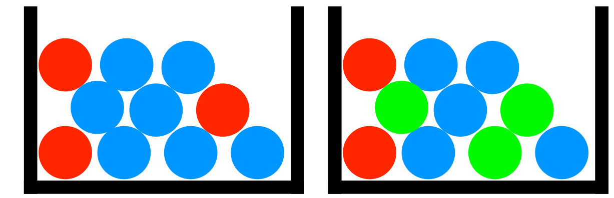
... with some detours on the way

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

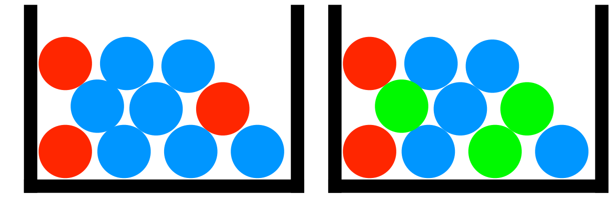
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
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)

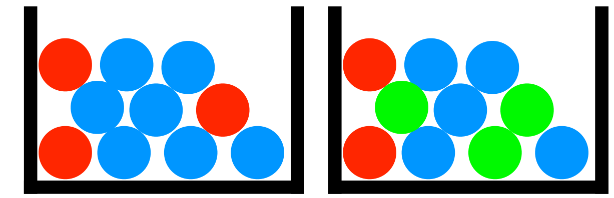
```
0.4 :: heads.
```

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).
```

ProbLog by example:

A bit of gambling



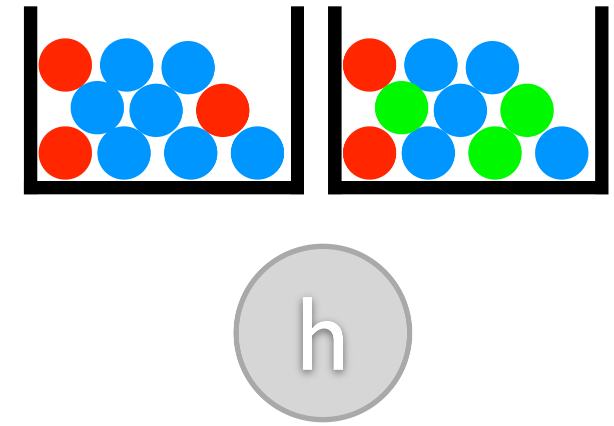
- toss (biased) coin & $\boxed{\text{draw ball from each urn}}$
- win if (heads and a red ball) or (two balls of same color)

```
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



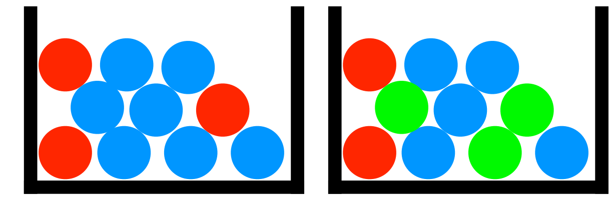
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
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



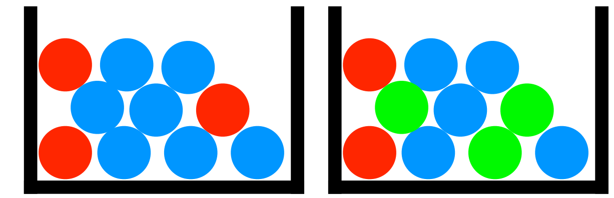
- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
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)

probabilistic choices

```
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).
```

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).
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`**?
- 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).
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).
```

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).
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).
```

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).
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).
```

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).
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).
```

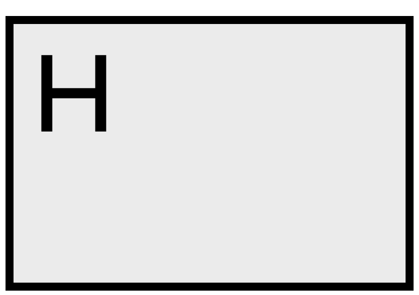
Possible Worlds

```
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).
```

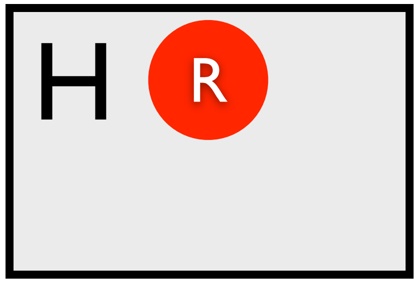
0.4



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).
```

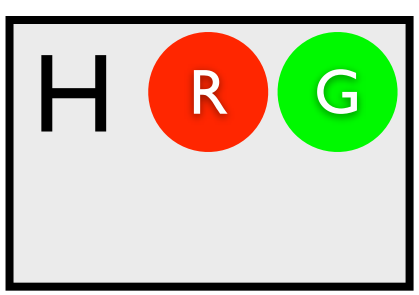
0.4 ×0.3



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).
```

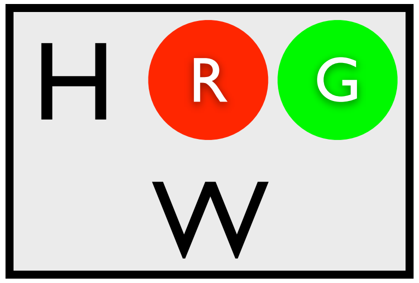
$0.4 \times 0.3 \times 0.3$



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).
```

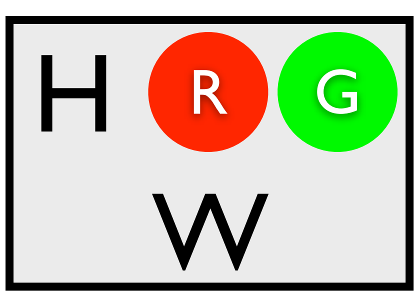
$0.4 \times 0.3 \times 0.3$



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).
```

$0.4 \times 0.3 \times 0.3$

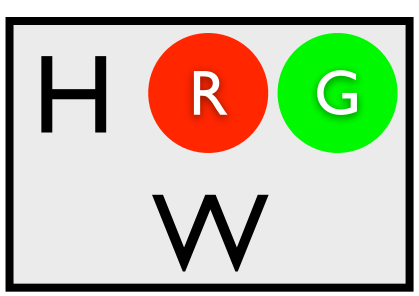


$(1-0.4)$

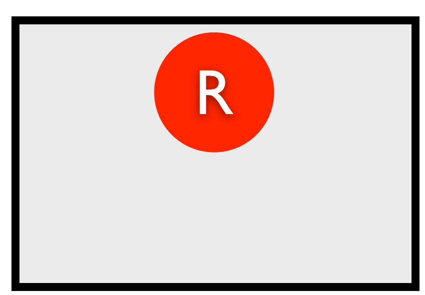
Possible Worlds

```
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).
```

$0.4 \times 0.3 \times 0.3$



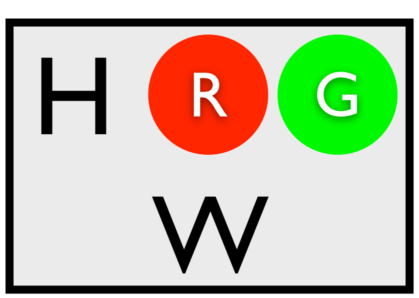
$(1-0.4)\times 0.3$



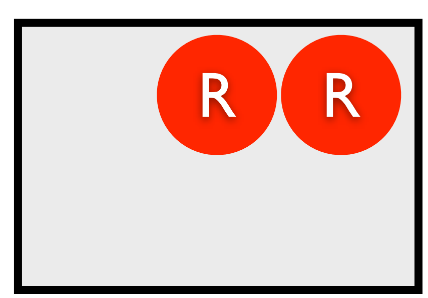
Possible Worlds

```
0.4 :: heads.

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

$0.4 \times 0.3 \times 0.3$



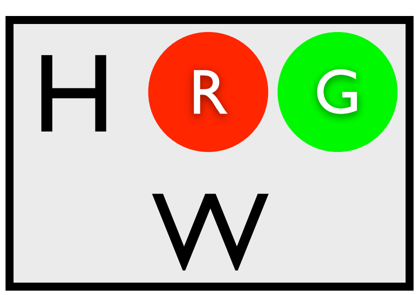
$(1-0.4) \times 0.3 \times 0.2$



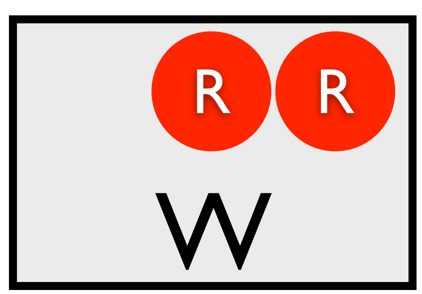
Possible Worlds

```
0.4 :: heads.

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

$0.4 \times 0.3 \times 0.3$



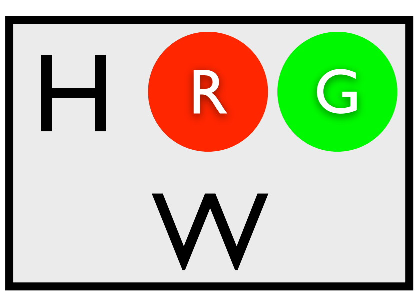
$(1-0.4) \times 0.3 \times 0.2$



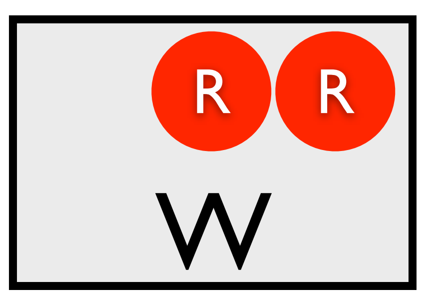
Possible Worlds

```
0.4 :: heads.

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

$0.4 \times 0.3 \times 0.3$



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

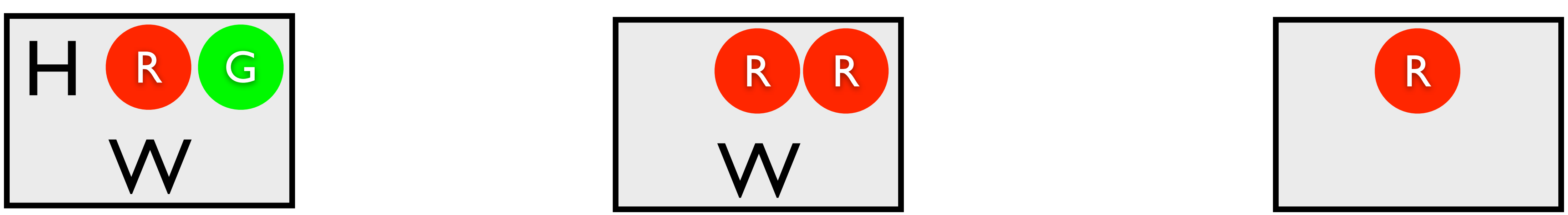


$(1-0.4)$

Possible Worlds

```
0.4 :: heads.

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

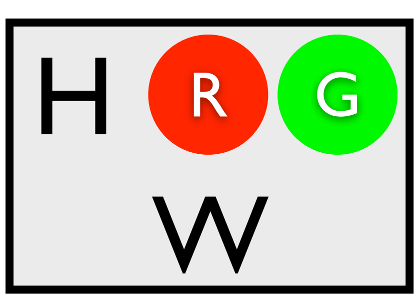
$0.4 \times 0.3 \times 0.3$ $\quad$ $(1-0.4) \times 0.3 \times 0.2$ $\quad$ $(1-0.4) \times 0.3$



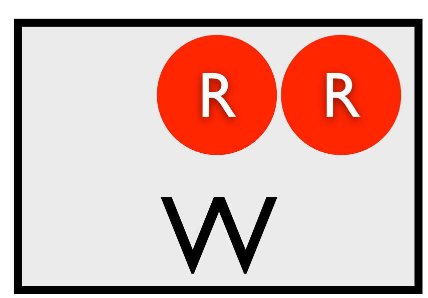
Possible Worlds

```
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).
```

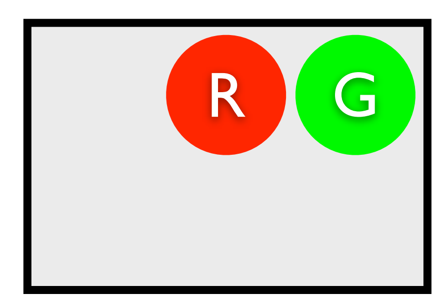
$0.4 \times 0.3 \times 0.3$



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



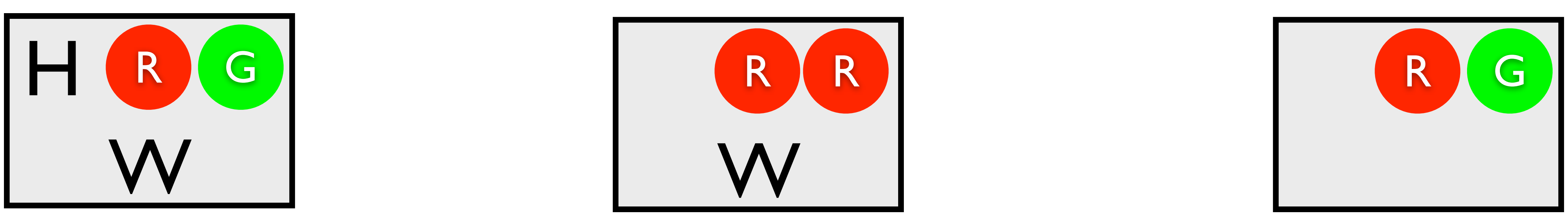
$(1-0.4) \times 0.3 \times 0.3$



Possible Worlds

```
0.4 :: heads.

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

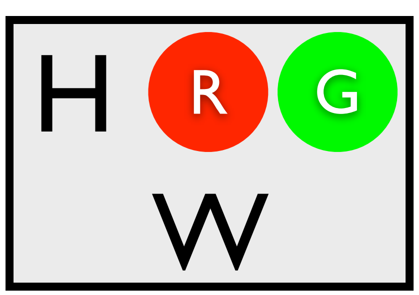
$0.4 \times 0.3 \times 0.3$ $\quad$ $(1-0.4) \times 0.3 \times 0.2$ $\quad$ $(1-0.4) \times 0.3 \times 0.3$



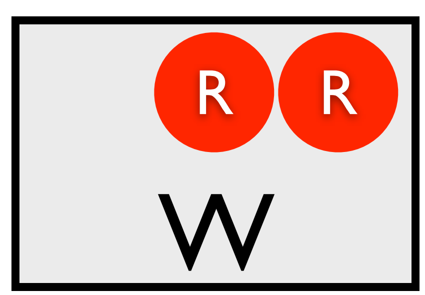
Possible Worlds

```
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).
```

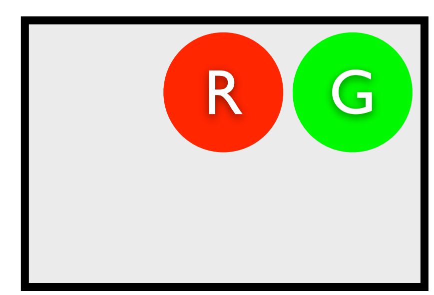
$0.4 \times 0.3 \times 0.3$



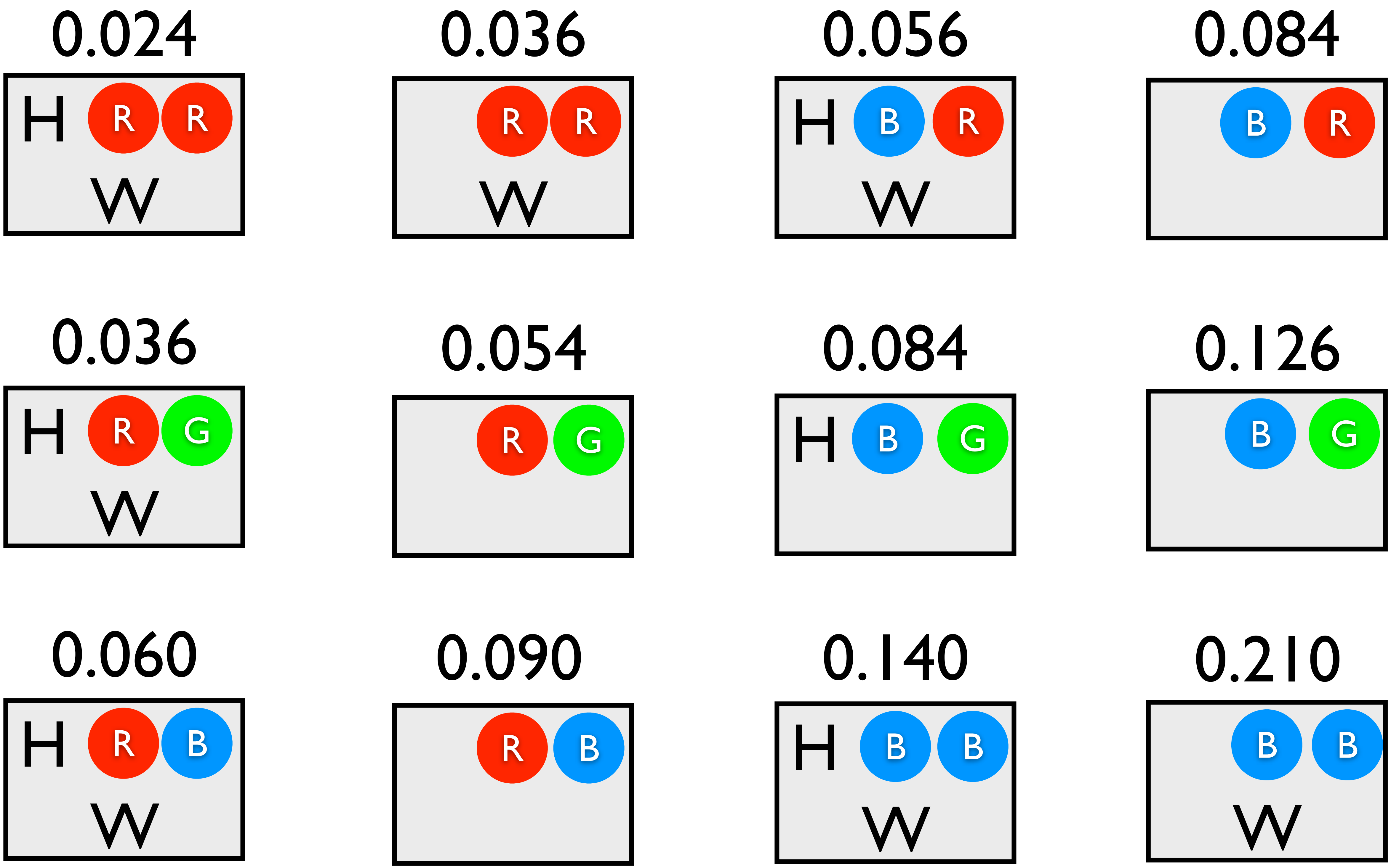
$(1-0.4) \times 0.3 \times 0.2$



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

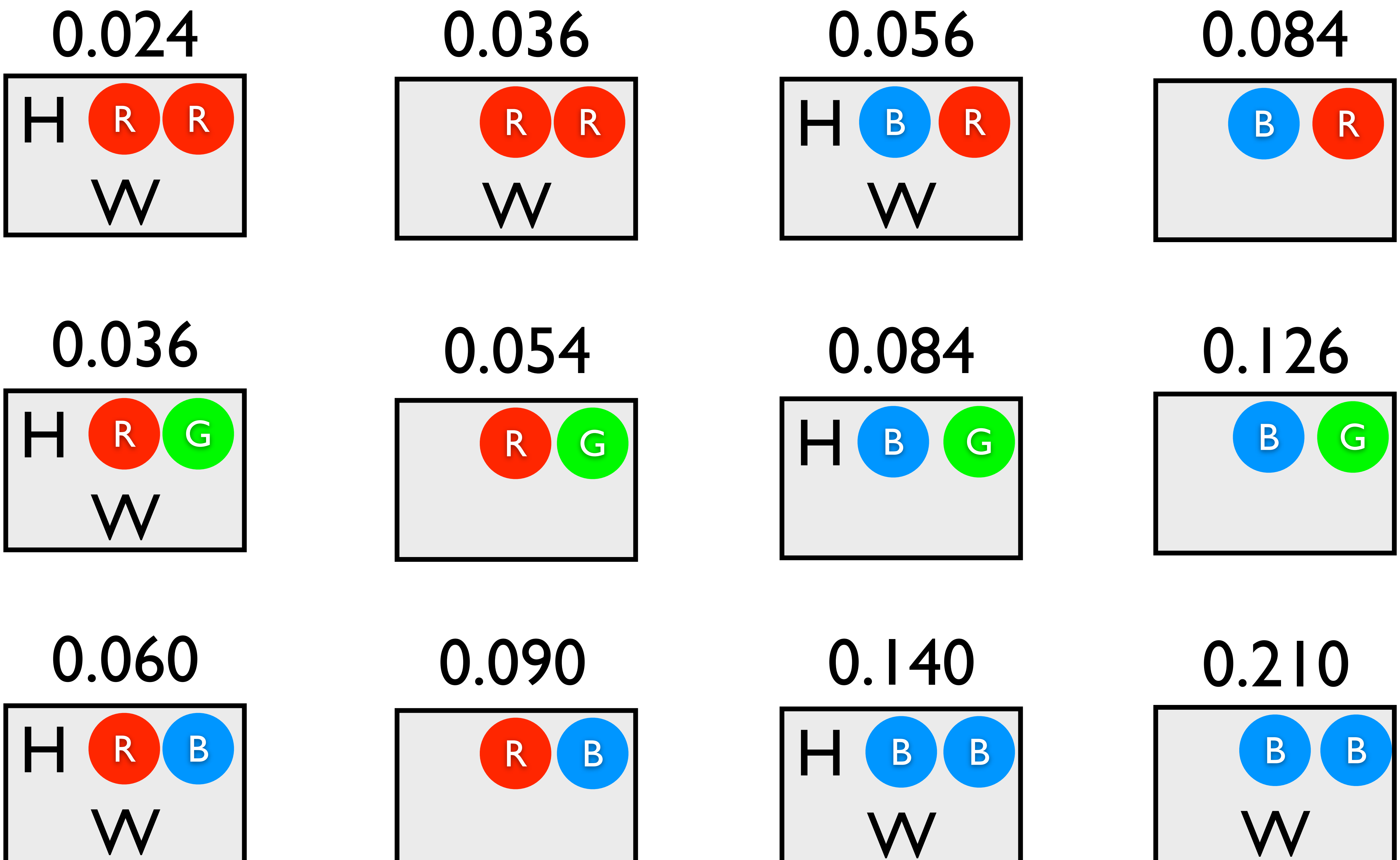


All Possible Worlds



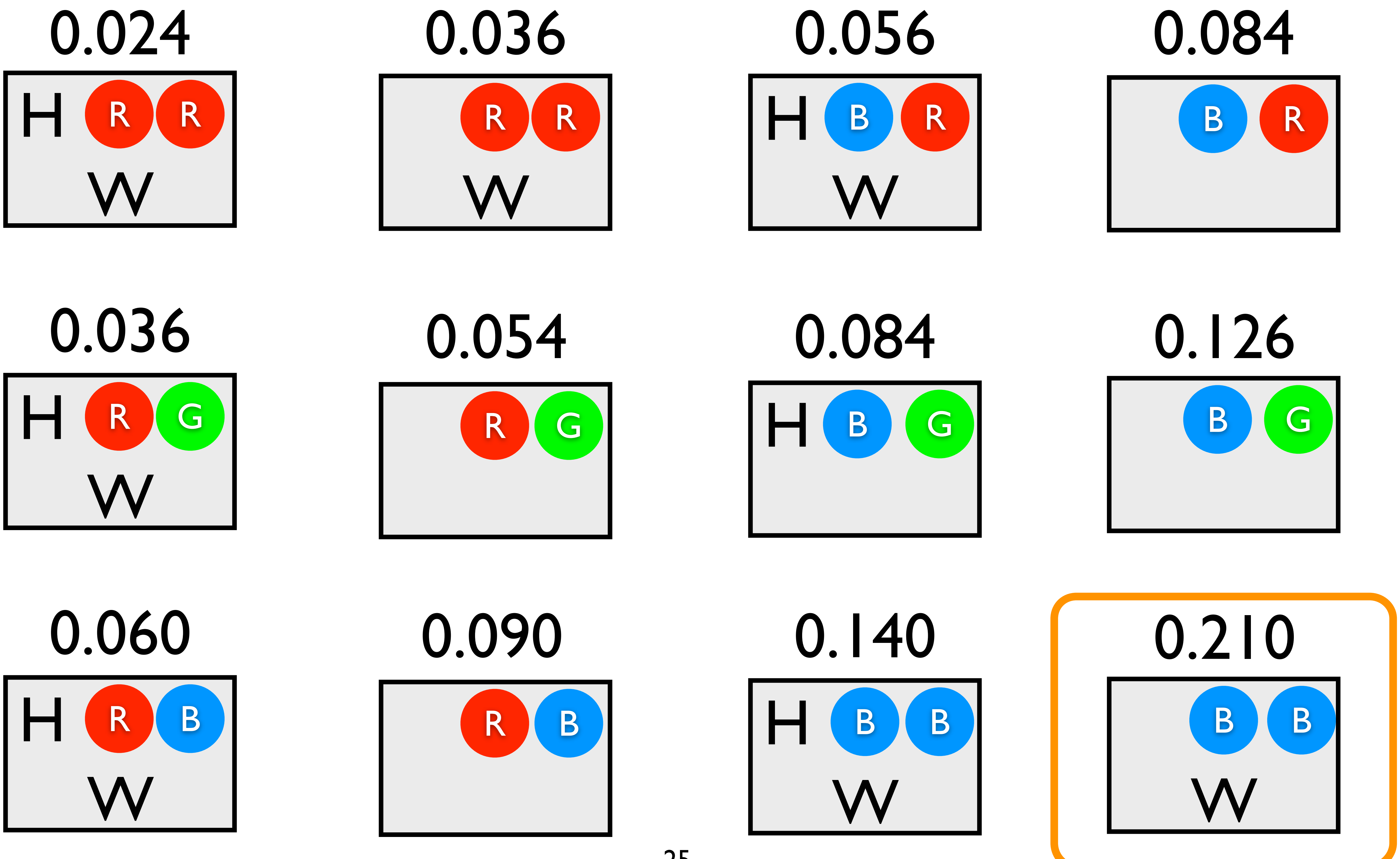
Most likely world where `win` is true?

MPE Inference



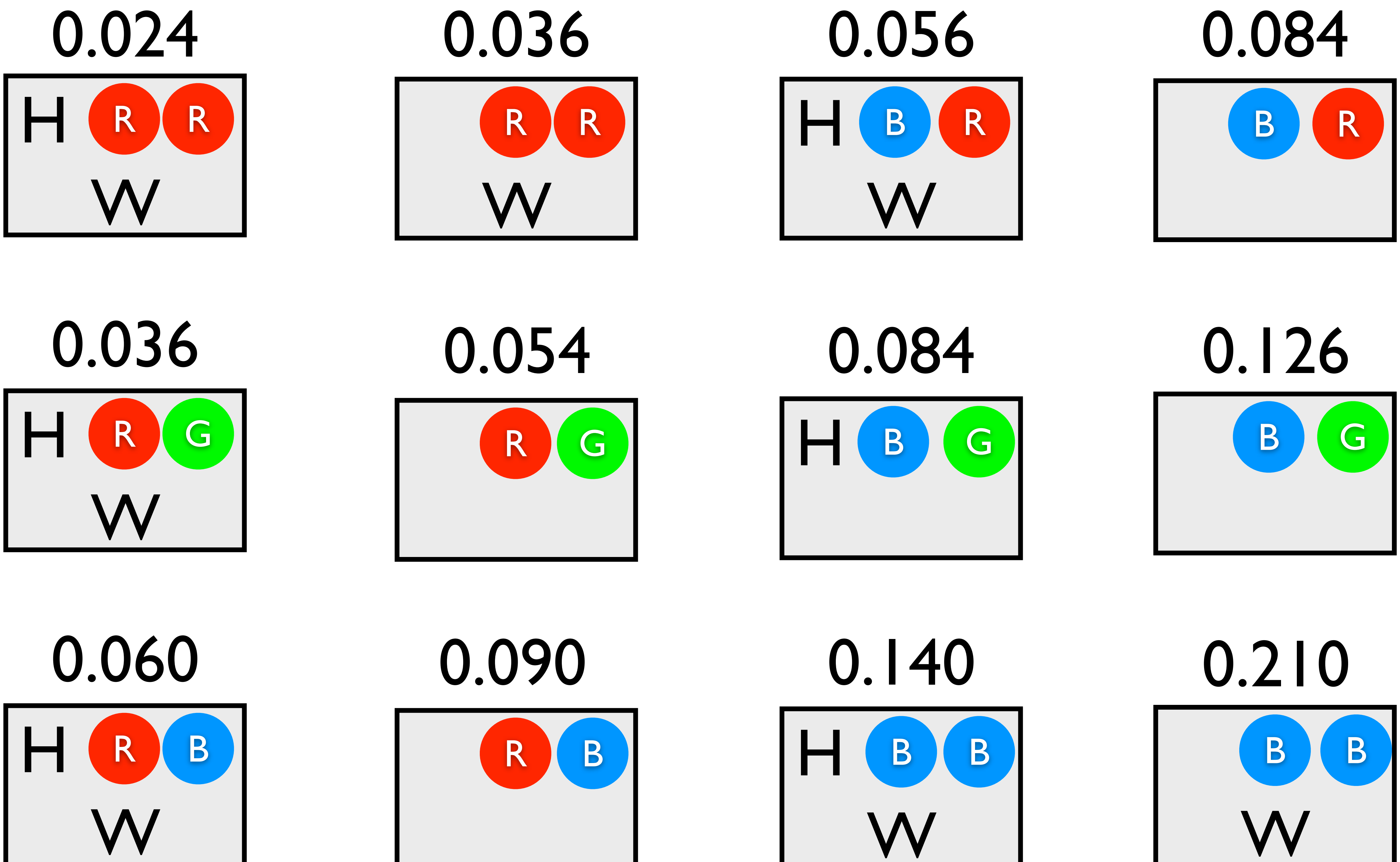
Most likely world where `win` is true?

MPE Inference



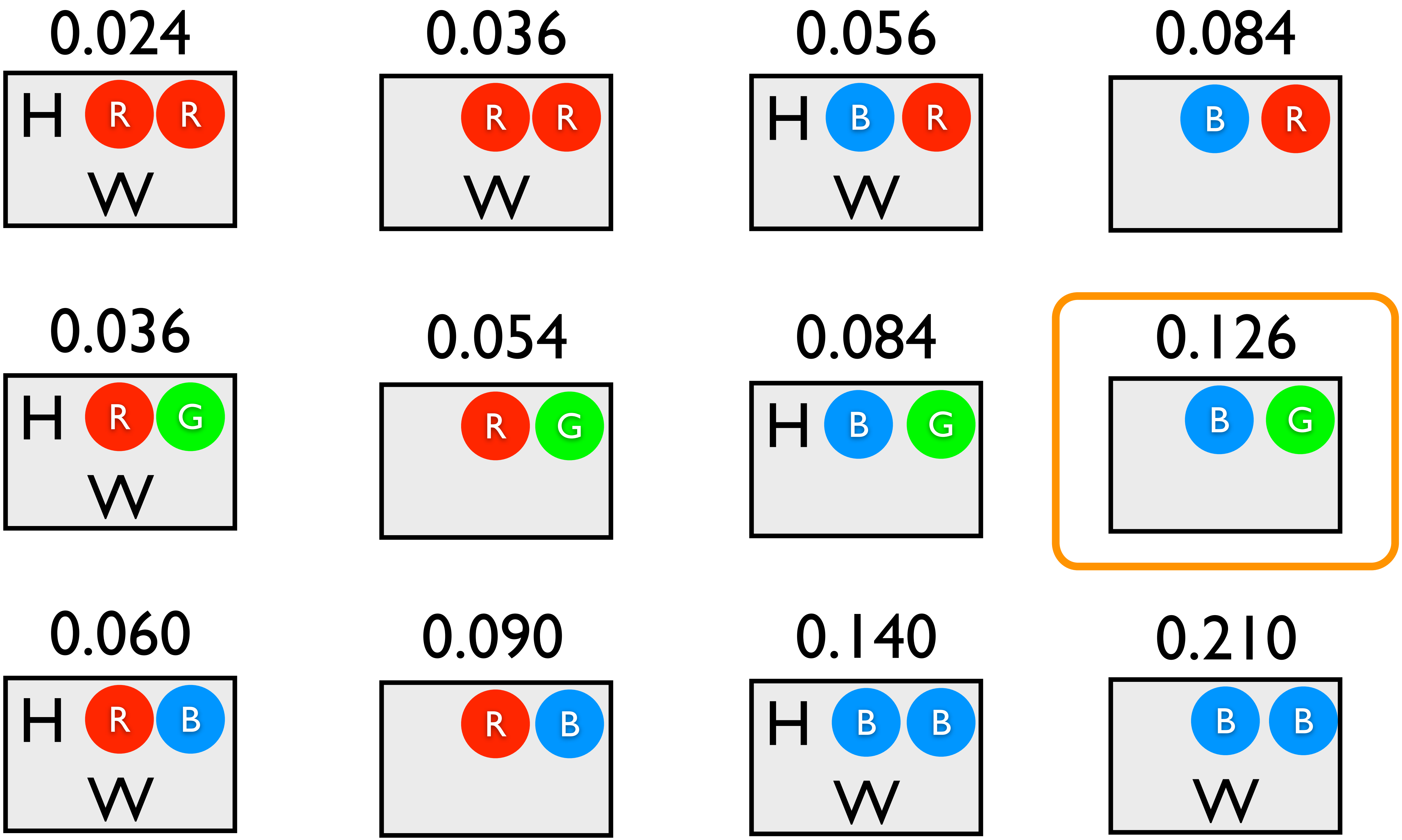
Most likely world where `col(2,blue)` is false?

MPE Inference



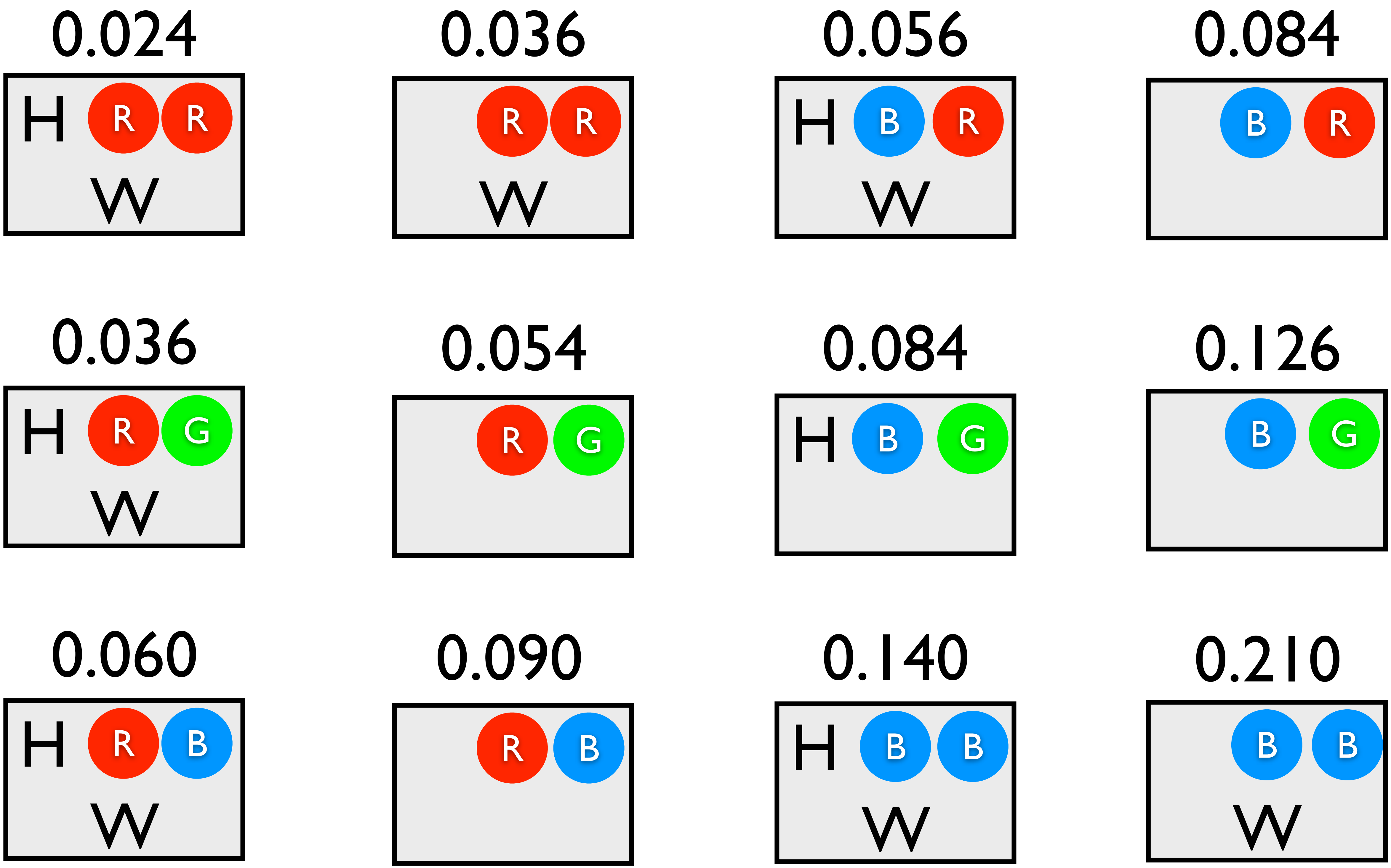
Most likely world where `col(2,blue)` is false?

MPE Inference



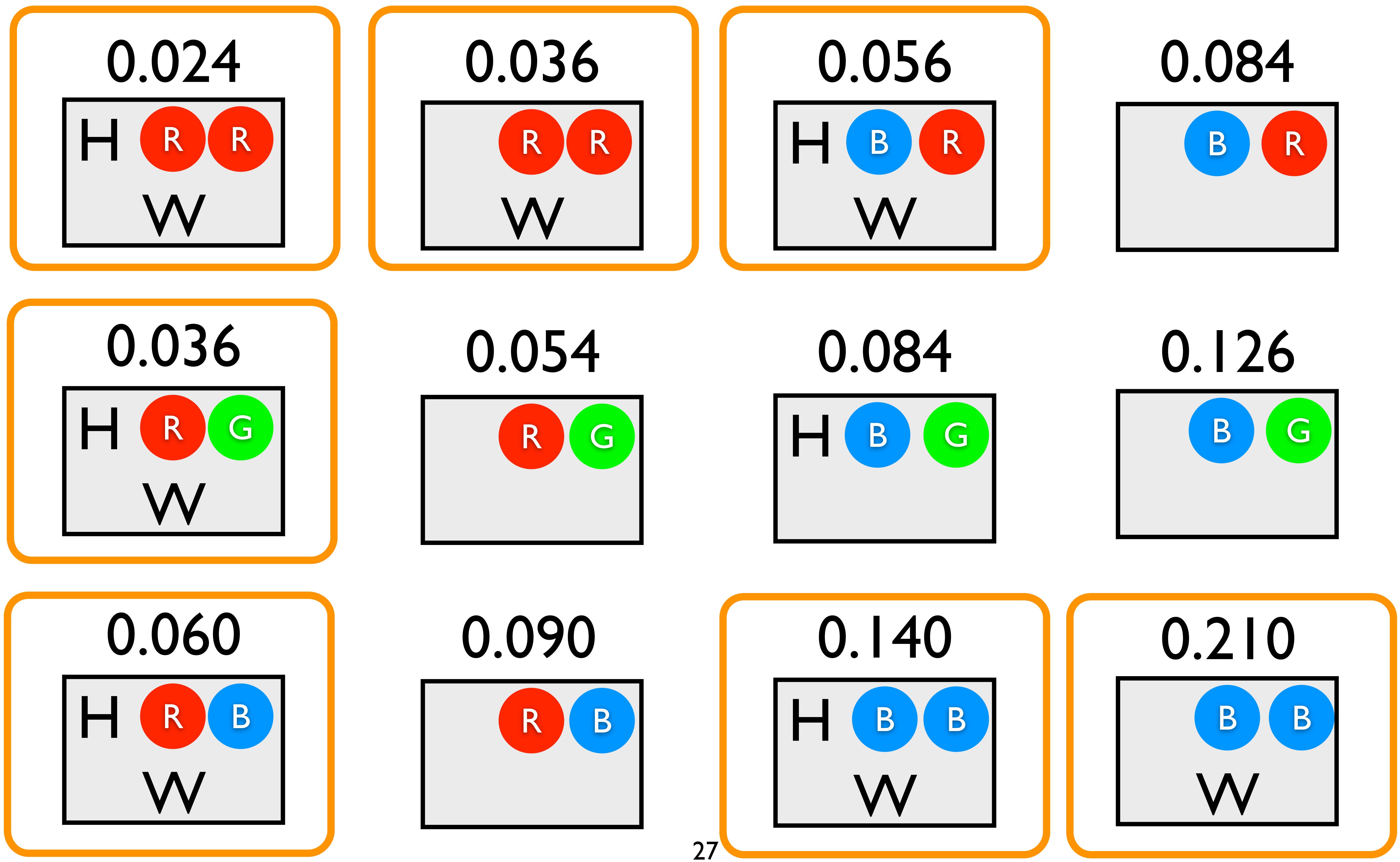
P(win)=?

Marginal Probability



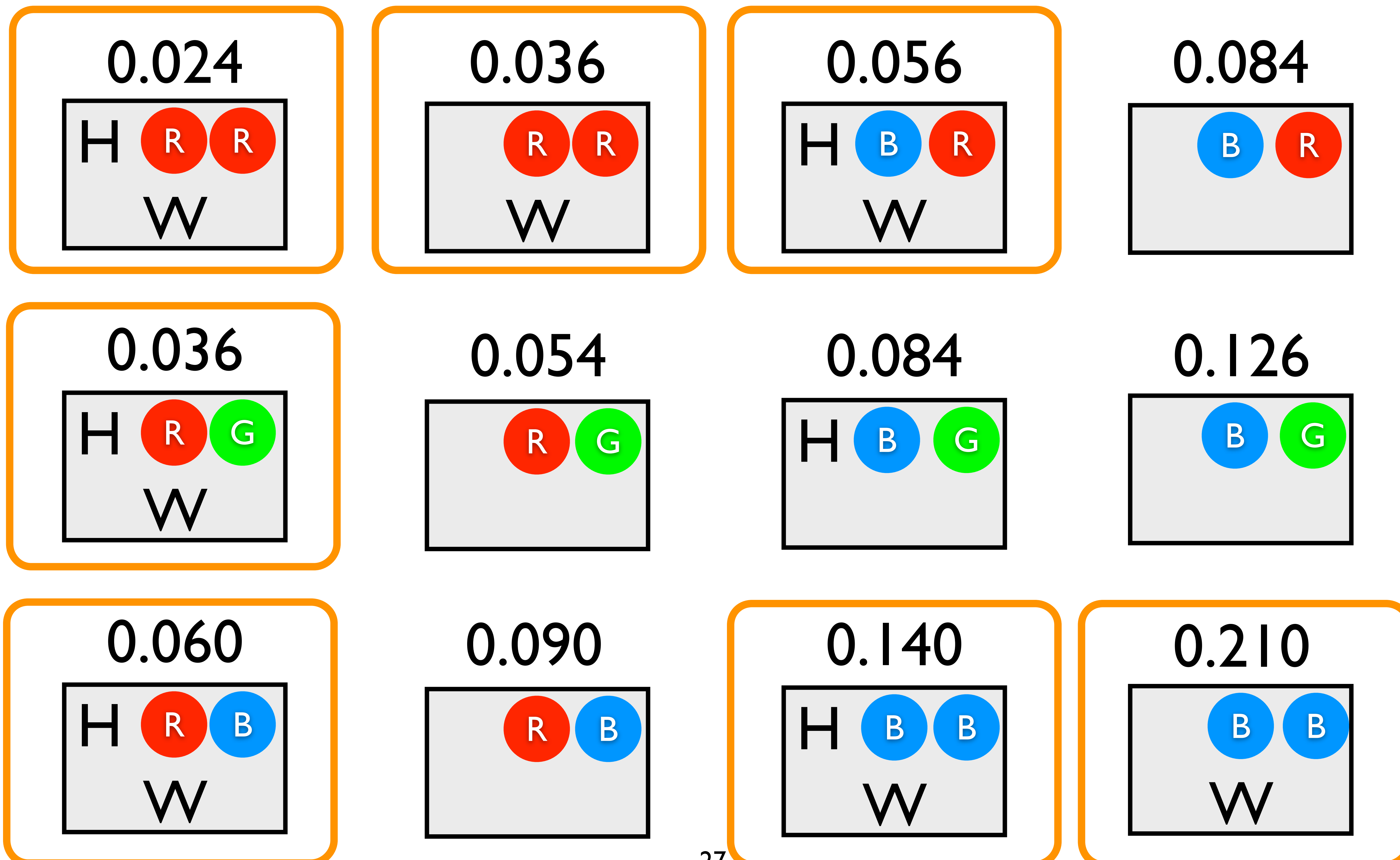
$P(\underline{win}) = \Sigma$

Marginal Probability



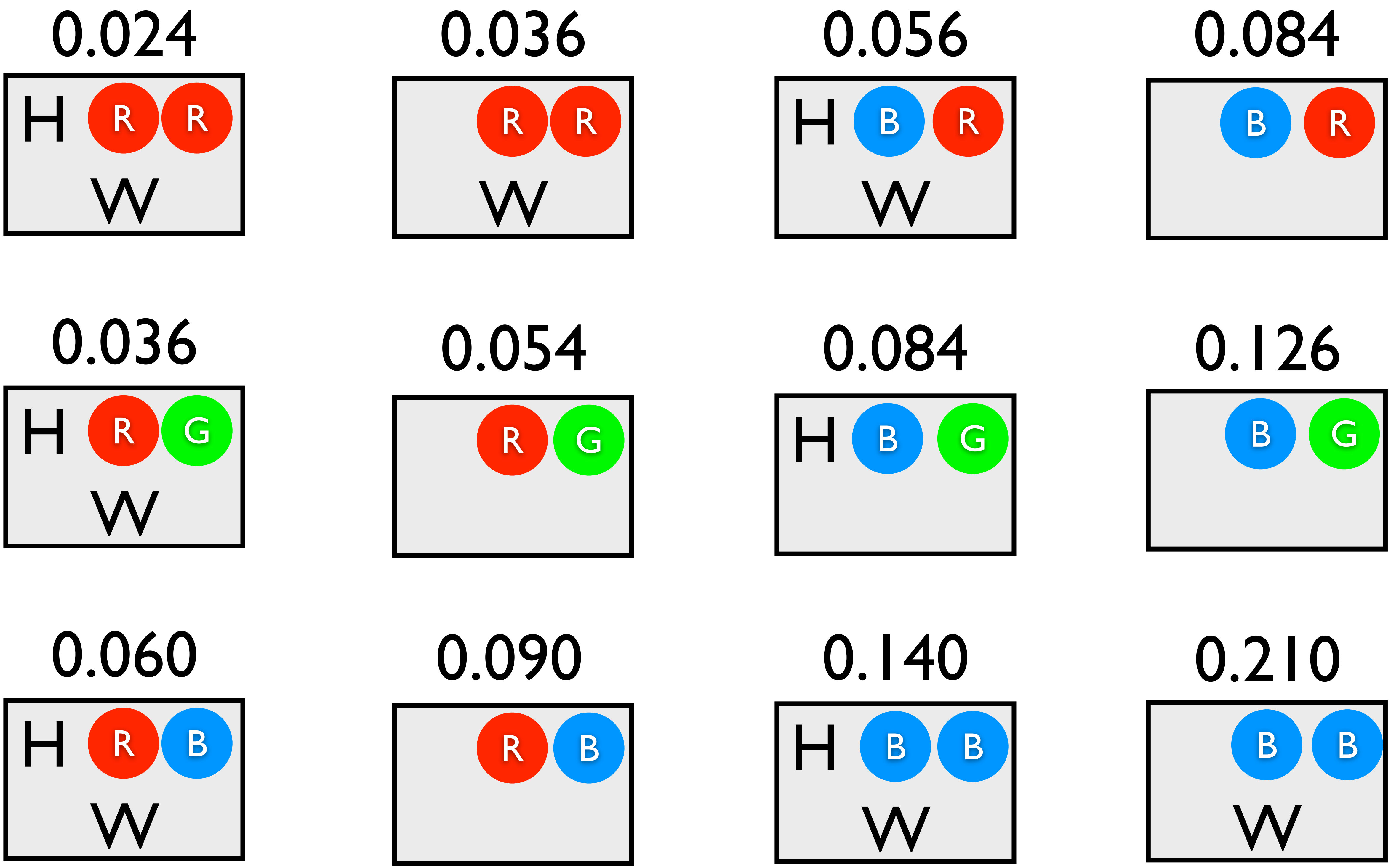
$$P(\underline{win}) = \sum = 0.562$$

Marginal Probability



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

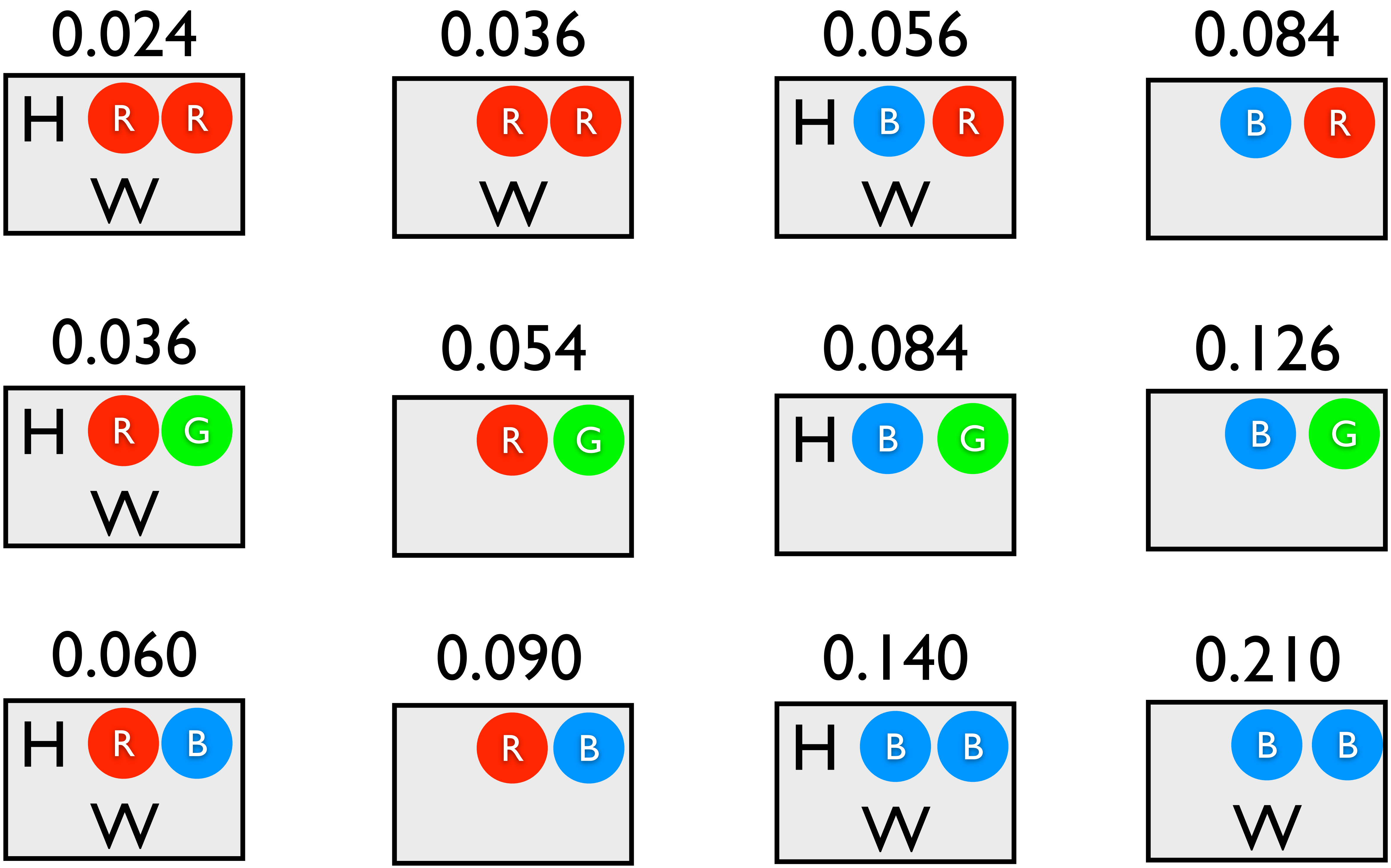
Conditional Probability



Conditional Probability

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

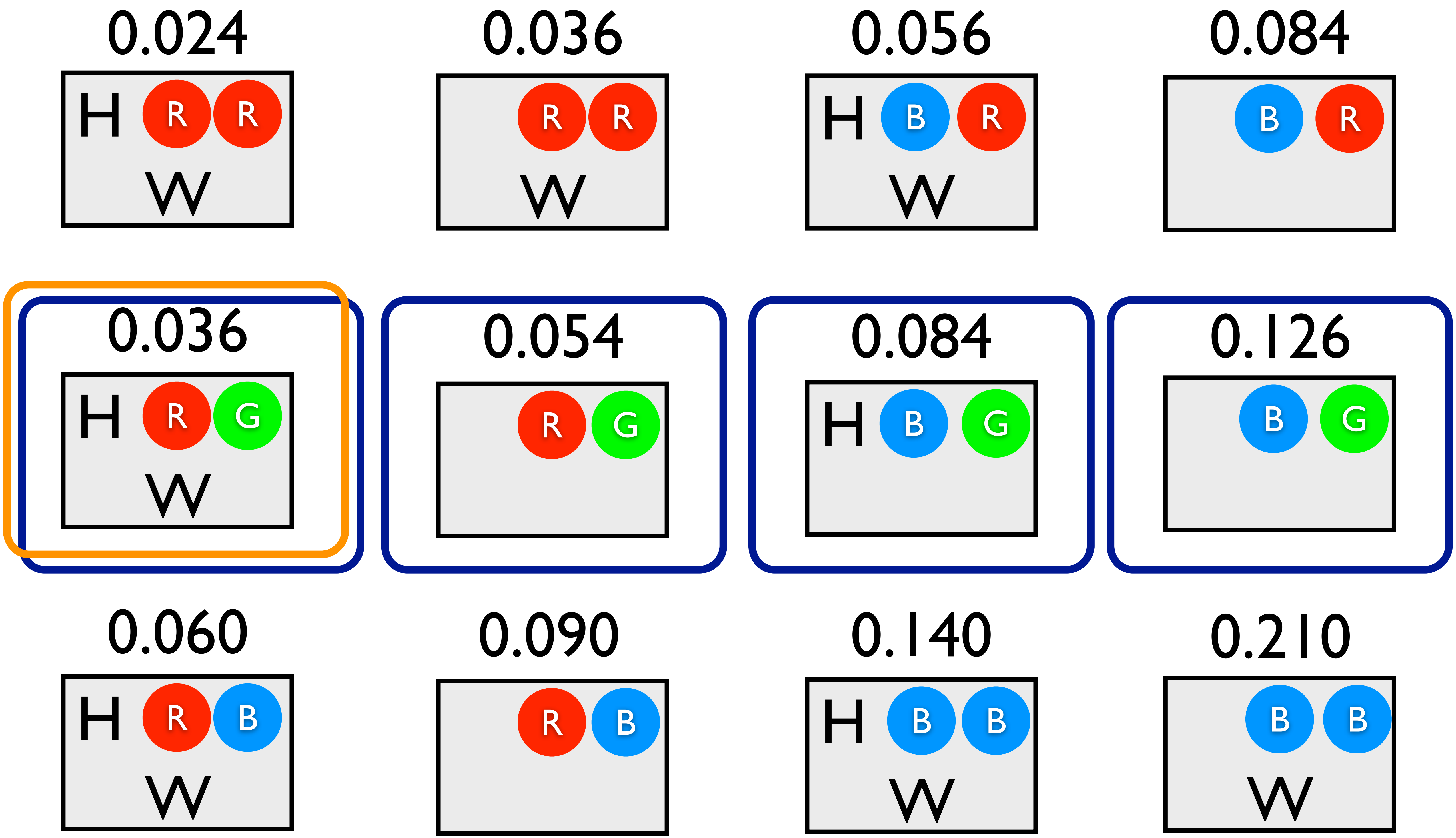
$= P(win \wedge col(2,green)) / P(col(2,green))$



Conditional Probability

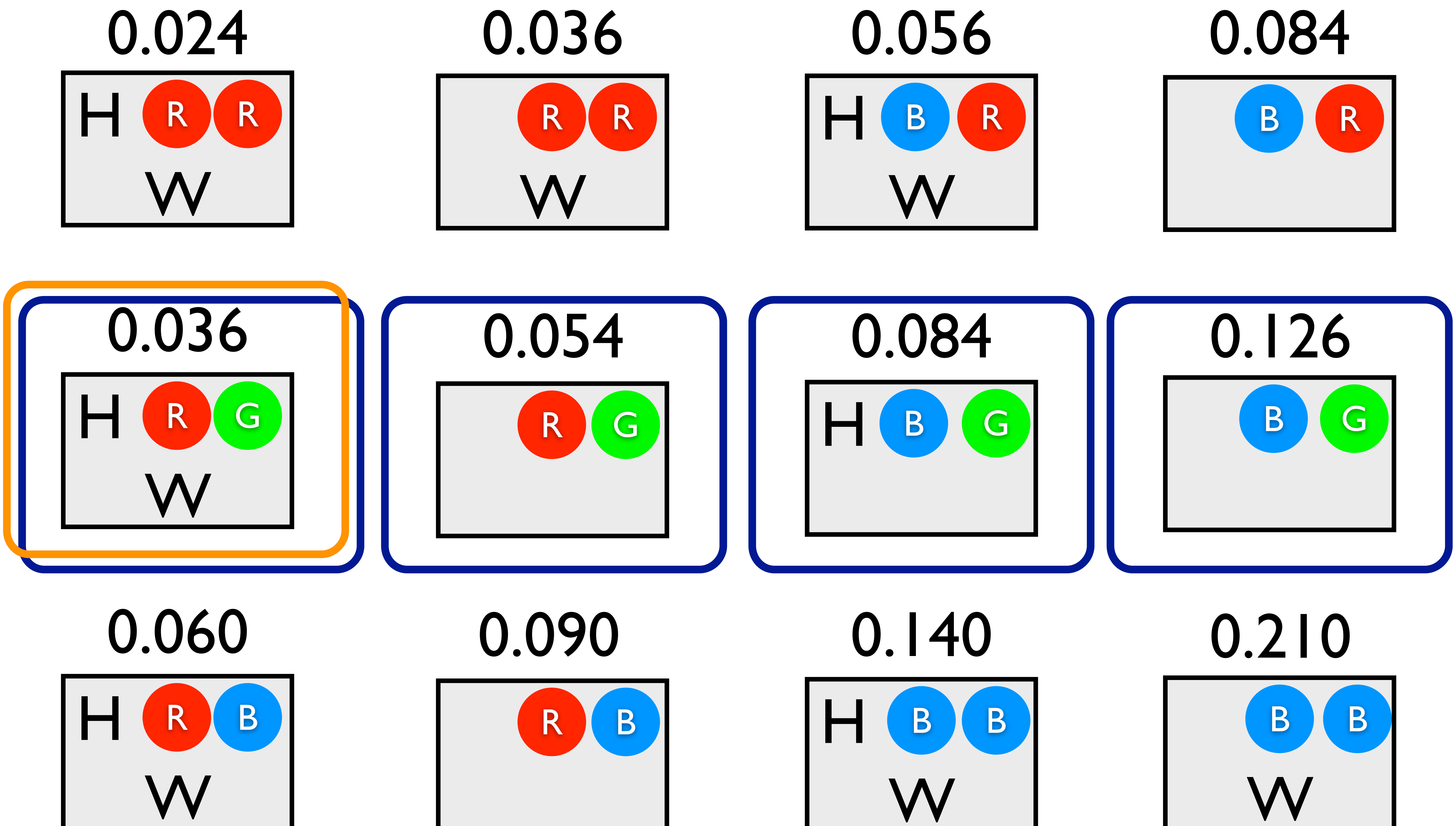
$$P(win|col(2,green)) = \Sigma/\Sigma$$

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



$$P(\text{win}|\text{col}(2,\text{green})) = \Sigma / \Sigma$$
$$= 0.036/0.3 = 0.12$$

Conditional Probability



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).
```

distribution over **all** possible worlds

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 FOL formulas
treat as evidence

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 FOL formulas
treat as evidence

	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]

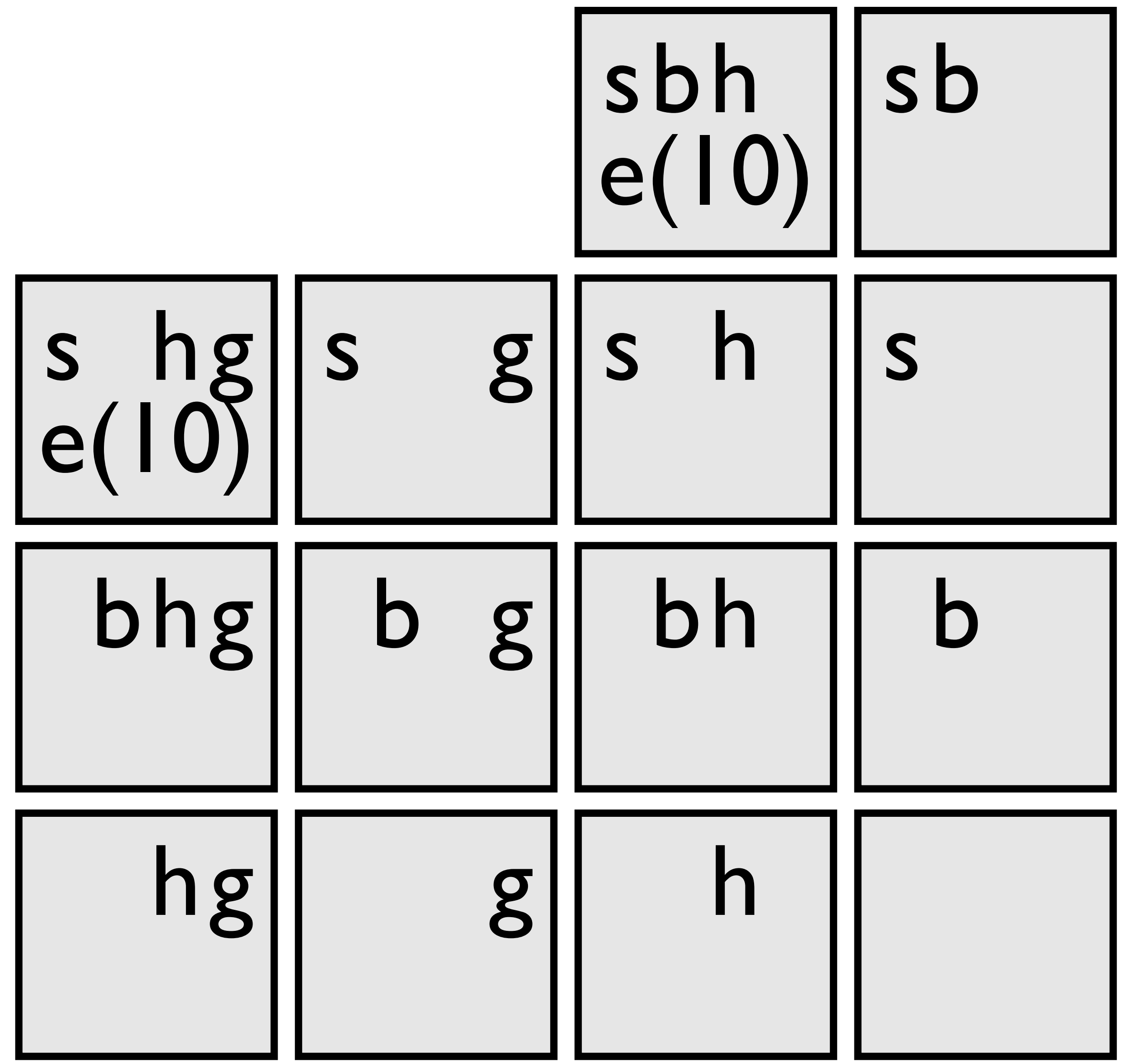
cProbLog: constraints on possible worlds

```
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 FOL formulas

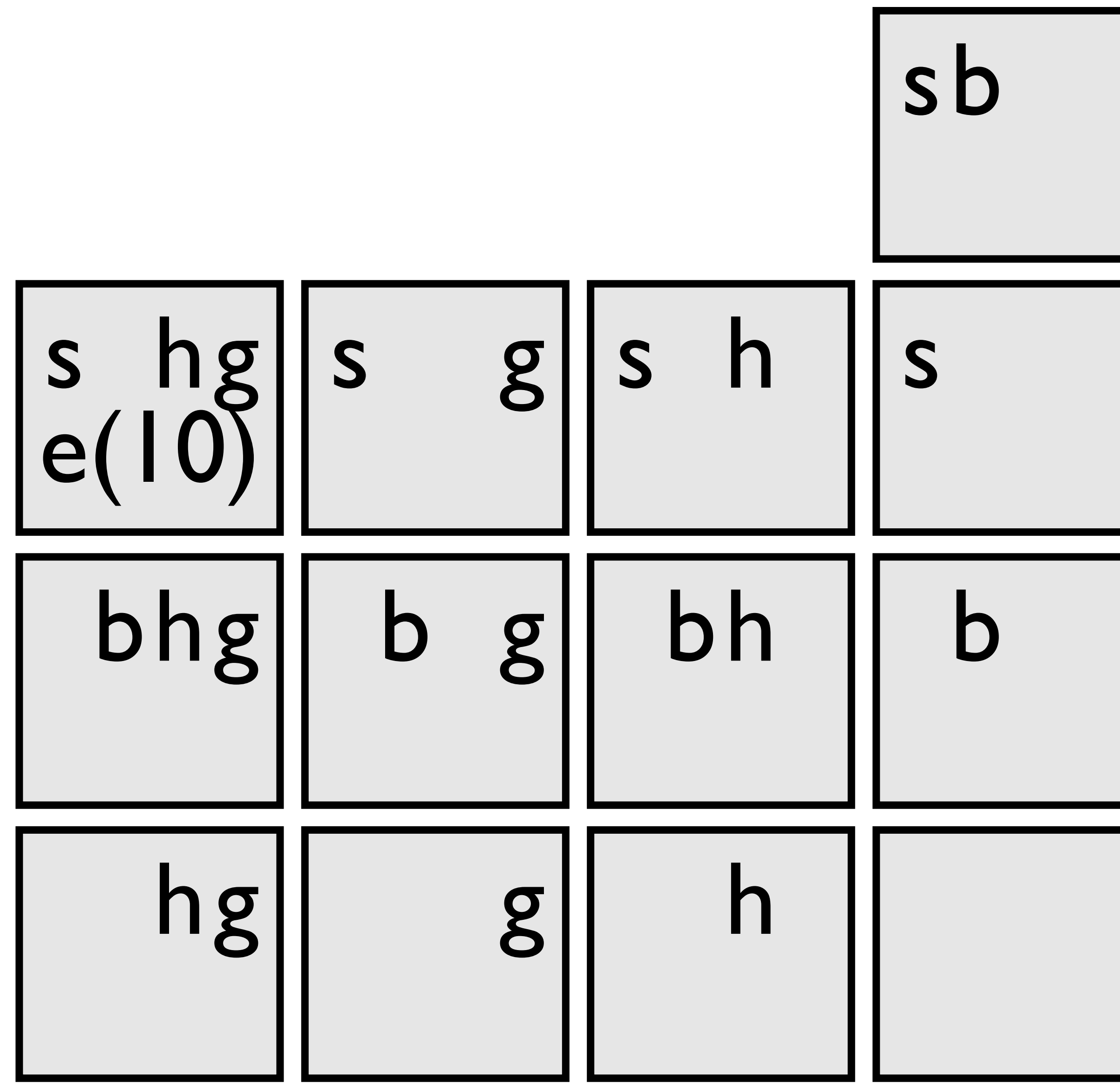
treat as evidence



[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

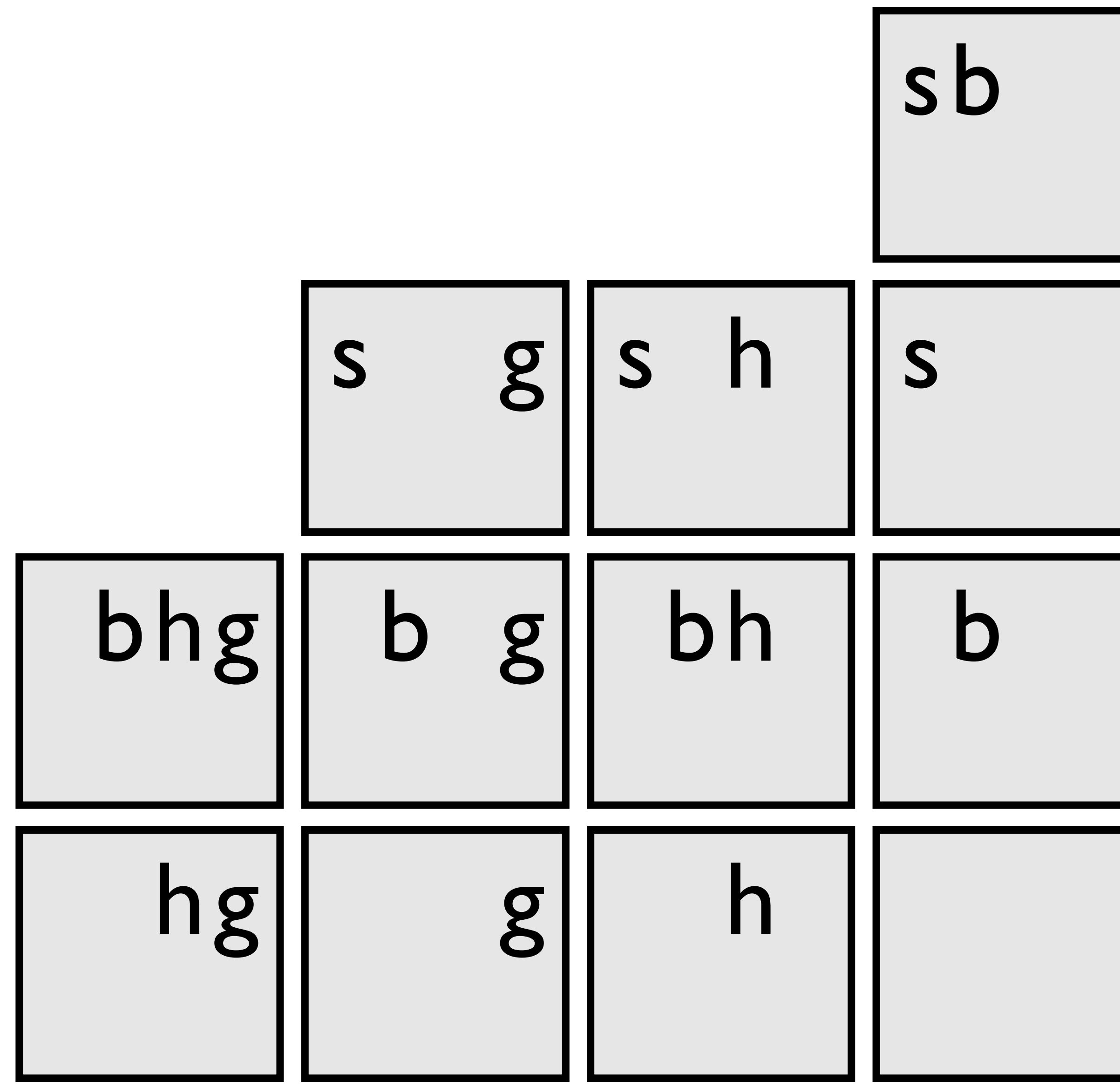
as FOL formulas

treat as evidence

[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 FOL formulas

treat as evidence

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

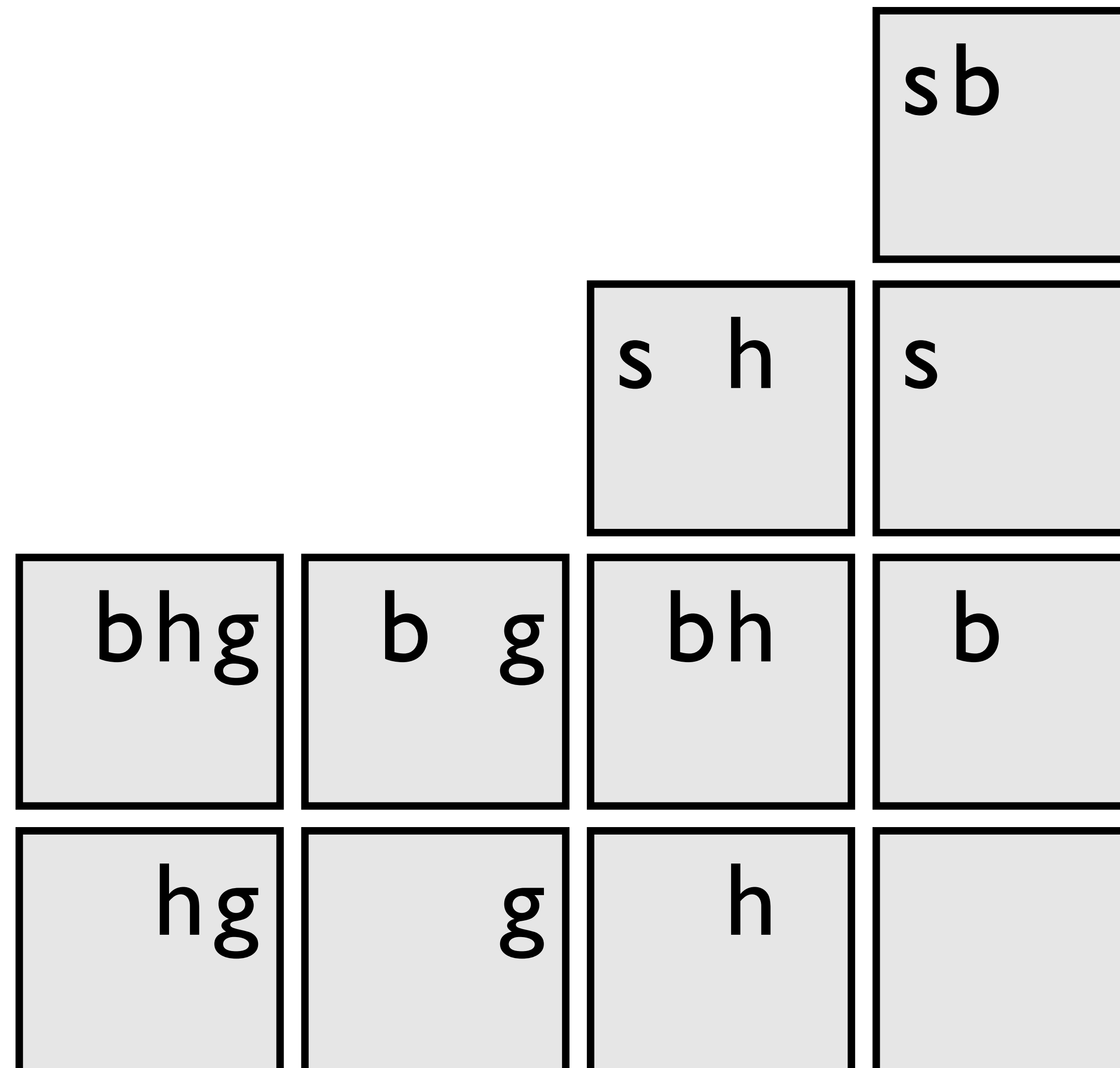
cProbLog: constraints on possible worlds

```
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 FOL formulas

treat as evidence

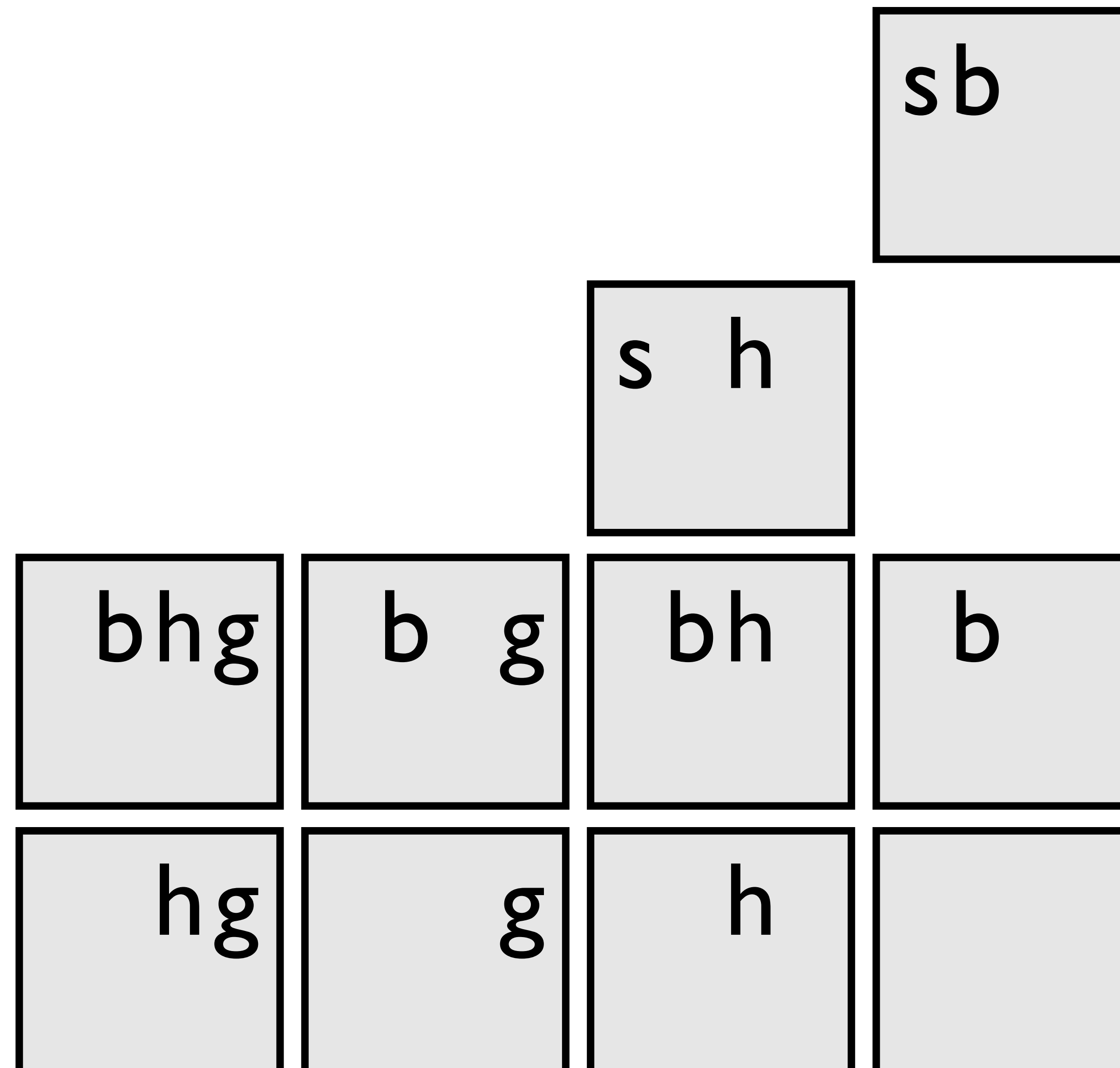


[Fierens et al, PP 12; Shterionov et al]

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).
```

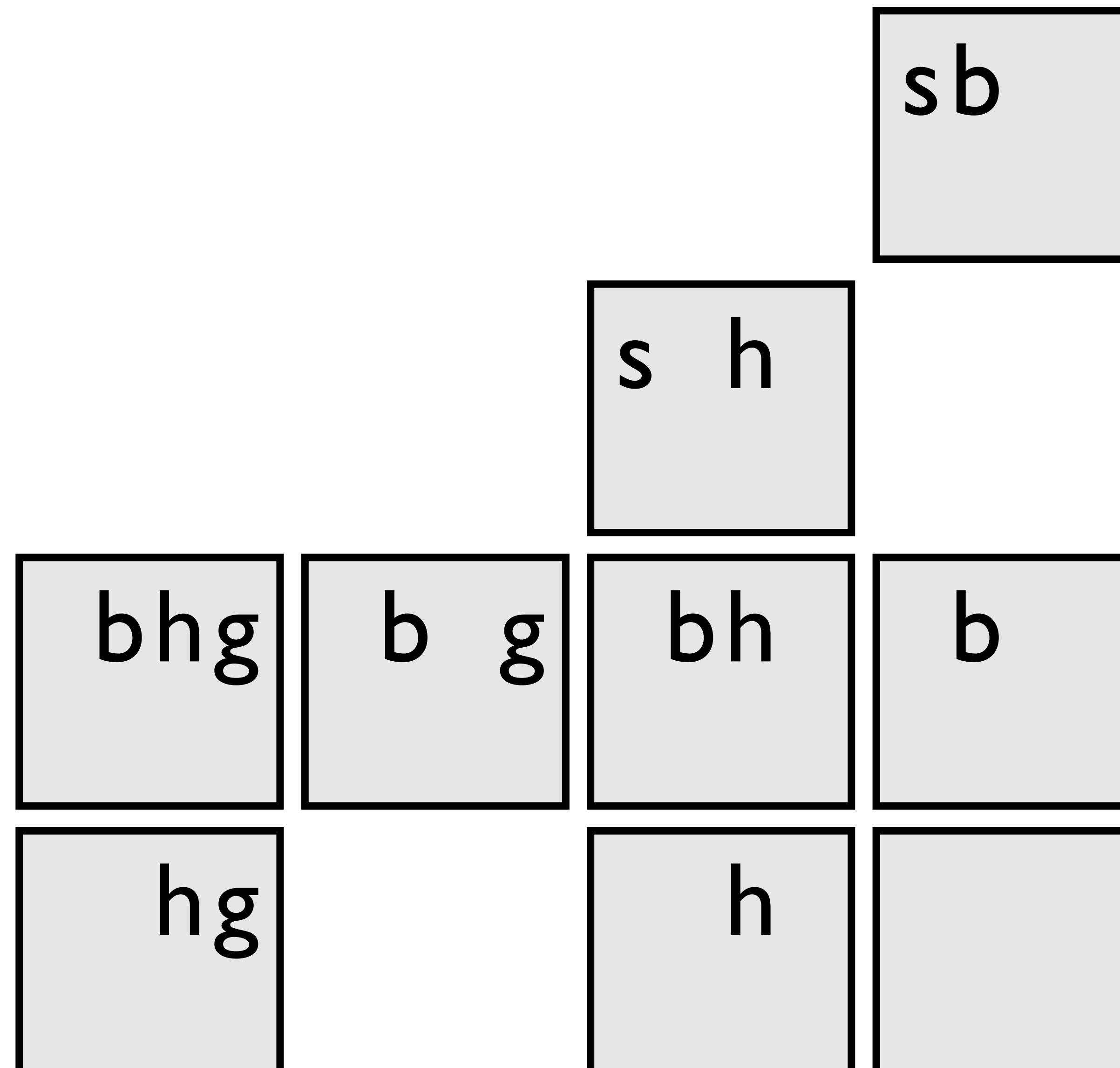
constraints
as FOL formulas
treat as evidence



[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

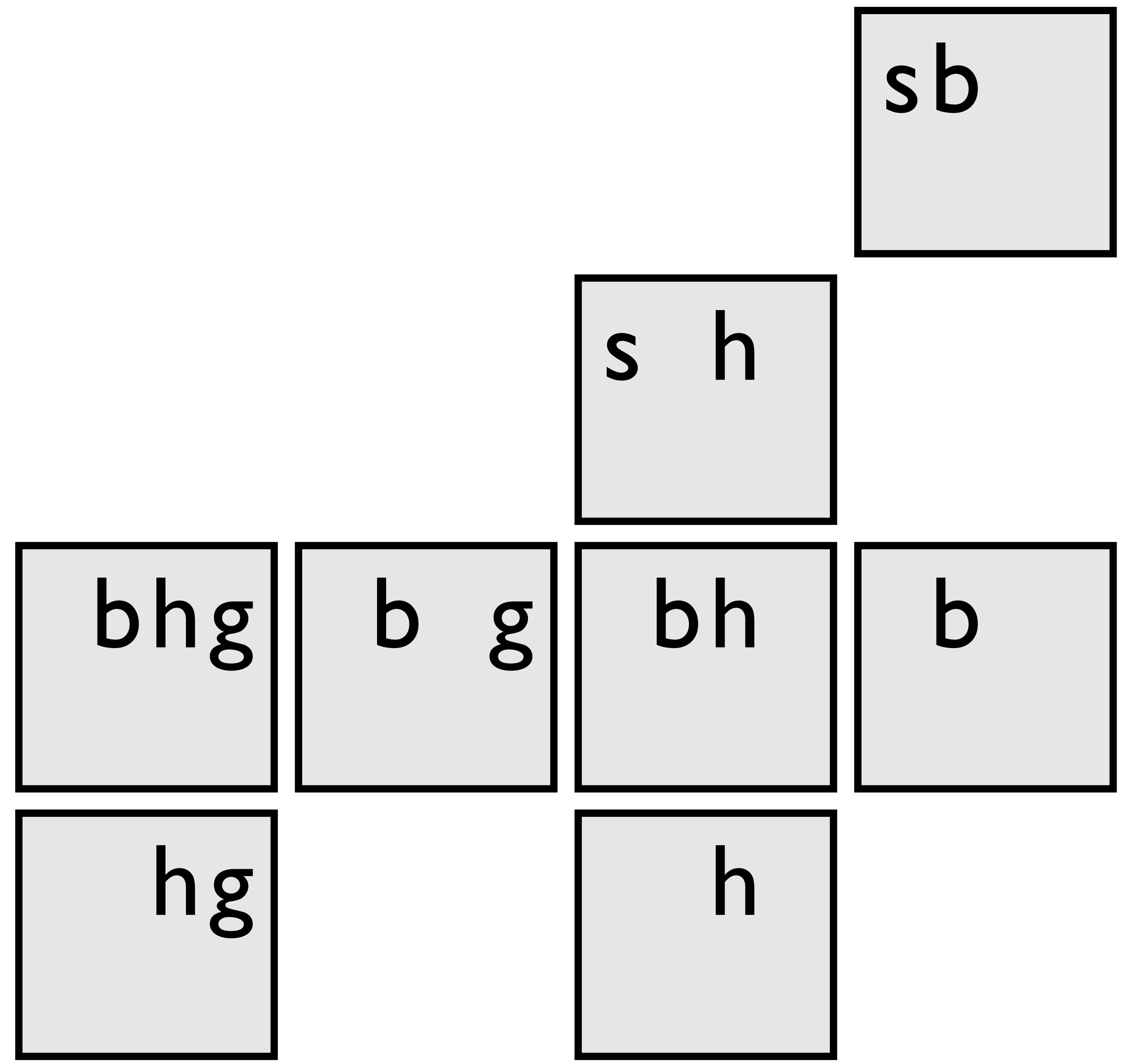
as FOL formulas

treat as evidence

[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).
```



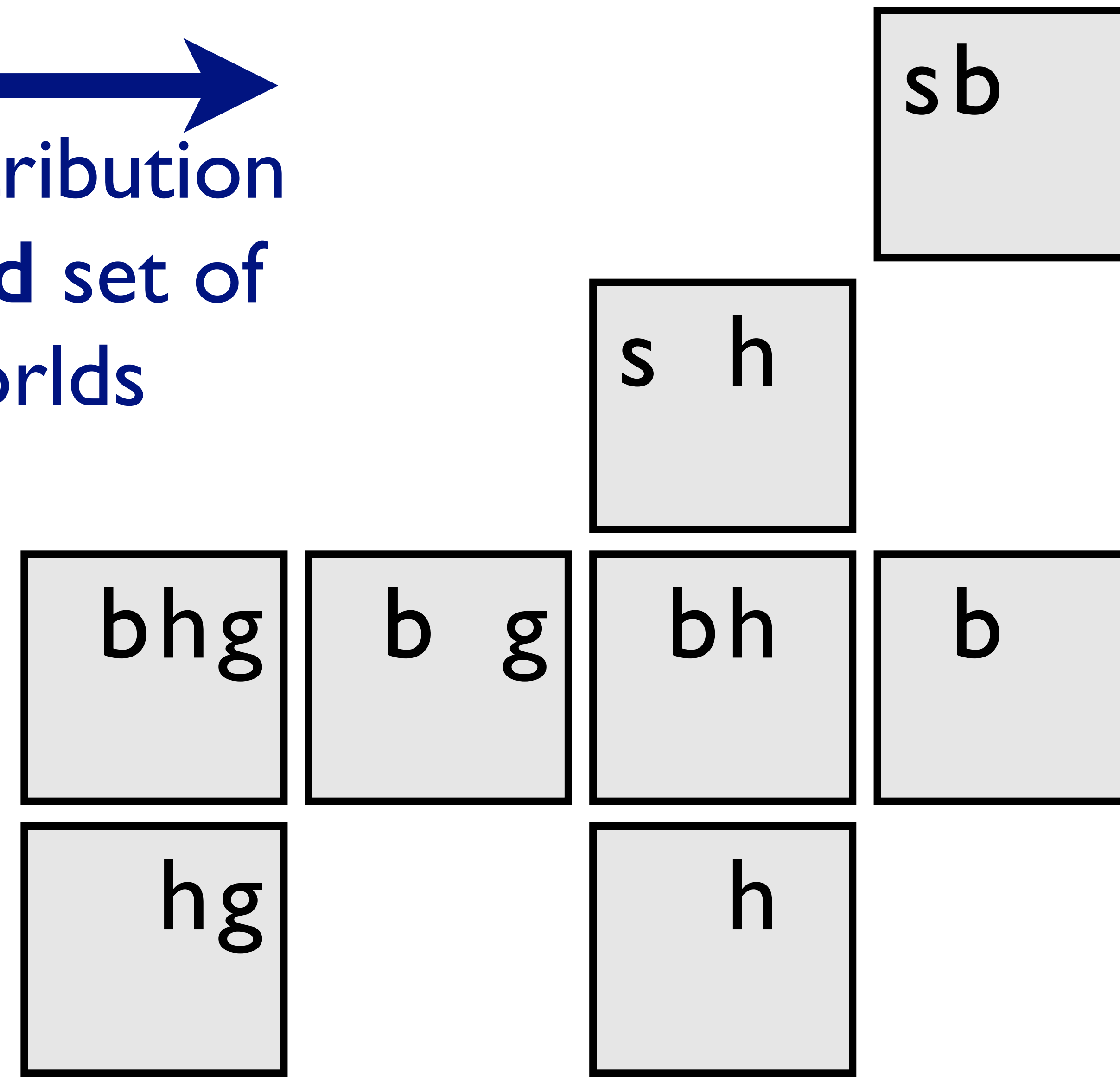
constraints
as FOL formulas
treat as evidence

[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 FOL formulas

treat as evidence

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

Distribution Semantics
(with probabilistic facts)

[Sato, ICLP 95]

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

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

$$P(Q) = \sum_{F \cup R \models Q} \prod_{f \in F} p(f) \prod_{f \not\in F} 1 - p(f)$$

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

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

subset of probabilistic facts

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

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

subset of probabilistic facts

Prolog rules

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

Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds where Q is true

$$P(Q) = \sum_{F \cup R \models Q} \underbrace{\prod_{f \in F} p(f) \prod_{f \notin F} 1 - p(f)}_{\text{probability of possible world}}$$

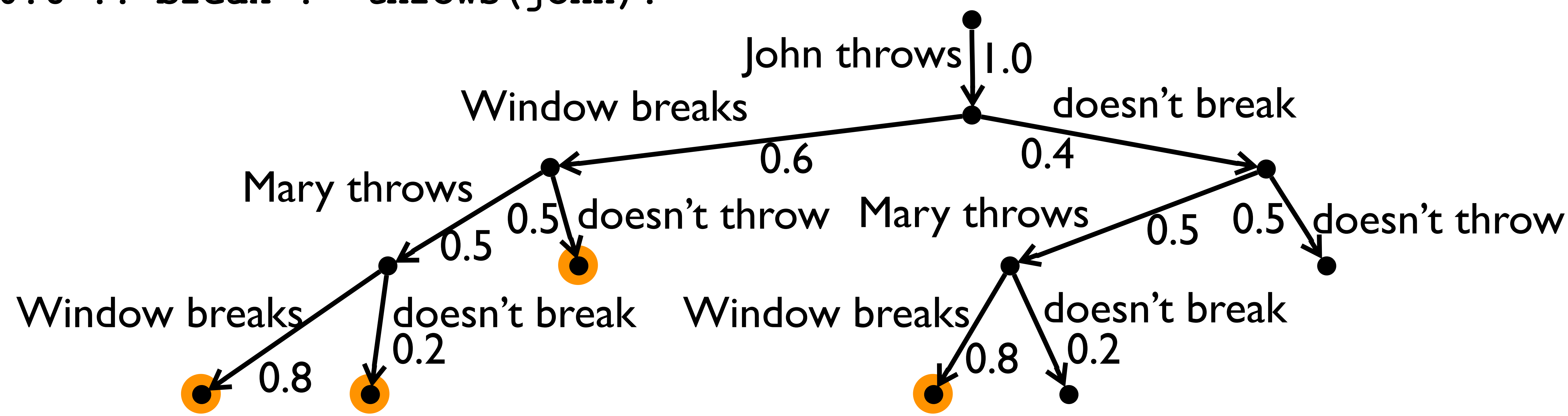
subset of probabilistic facts

Prolog rules

Alternative view: CP-Logic

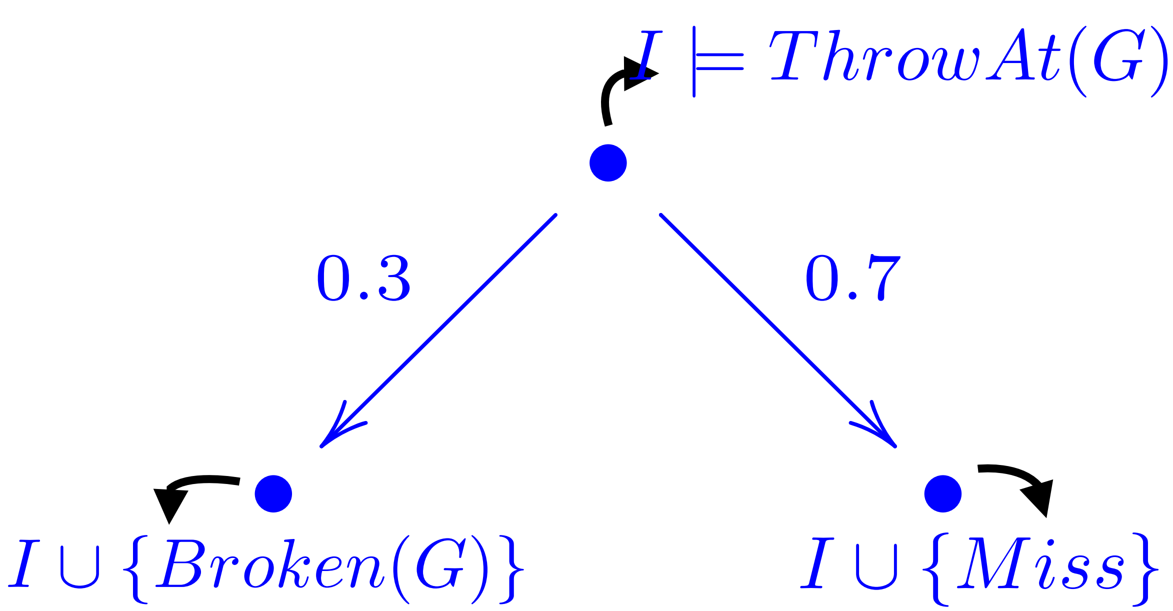
```
throws(john).
0.5::throws(mary).

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

probabilistic causal laws



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

Semantics



(Broken(G) 0.3) ∨ (Miss 0.7)
← ThrowAt(G)

Probability tree is an execution model of theory iff:

- Each tree-transition matches causal law
- The tree cannot be extended

Each execution model defines the <u>same</u> probability distribution over final states

Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

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

Distributional Clauses (DC)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

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)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

```
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)

Closely related to BLOG [Russell et al.]

- Discrete- and continuous-valued random variables

```
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)

Closely related to BLOG [Russell et al.]

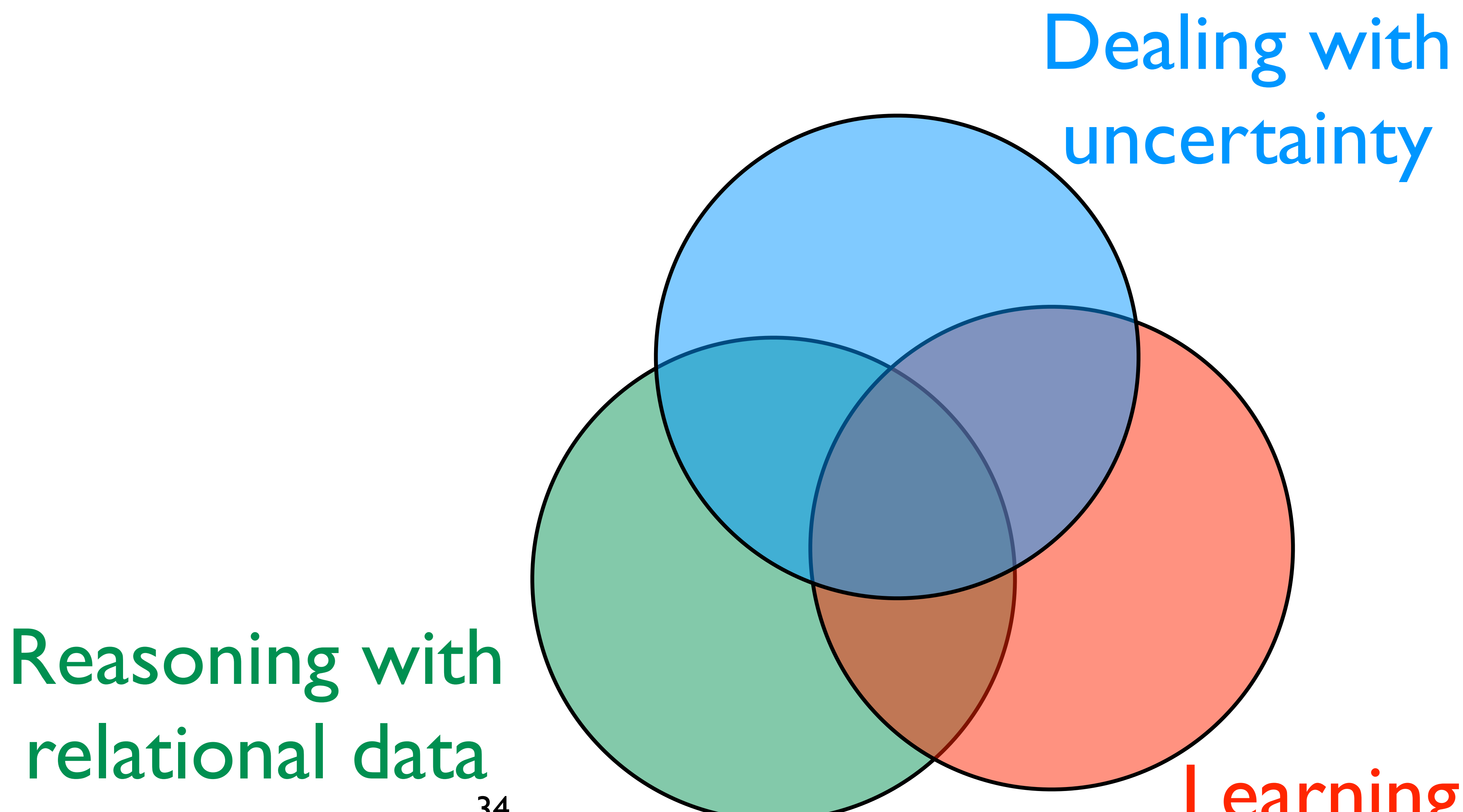
- Discrete- and continuous-valued random variables

```
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]

[Suciu et al 2011]

Probabilistic Databases



[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

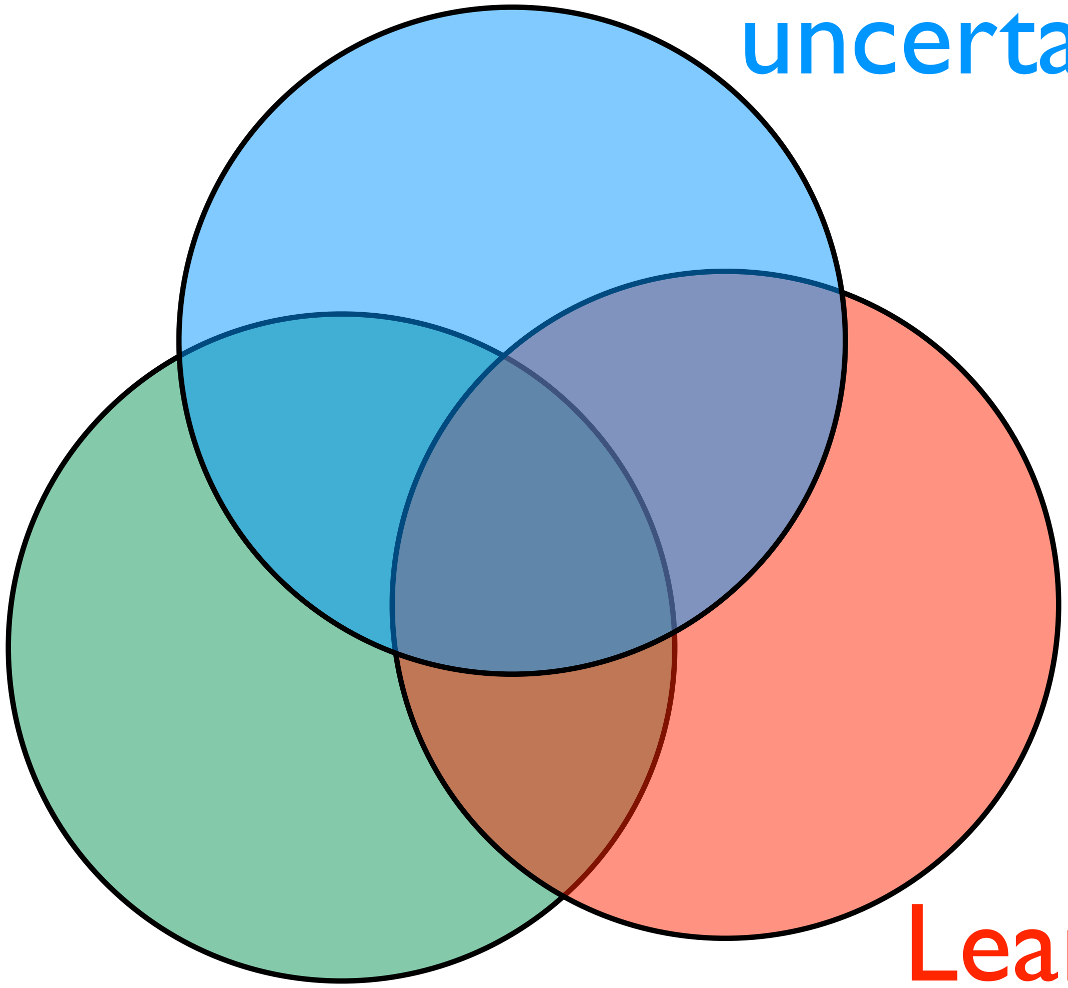
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

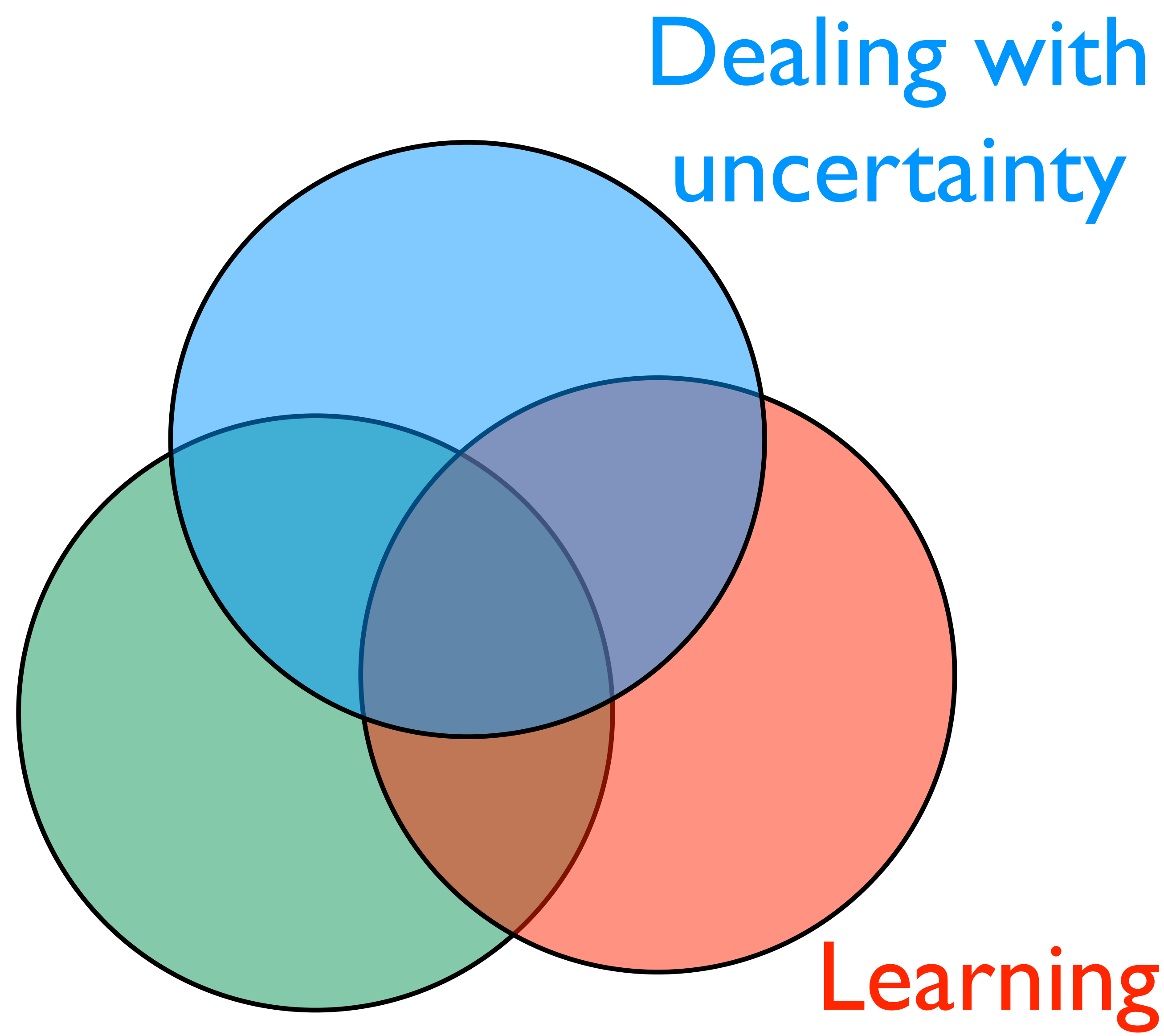
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



[Suciu et al 2011]

Probabilistic Databases

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,9
tom	paris	0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

tuples as random variables

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

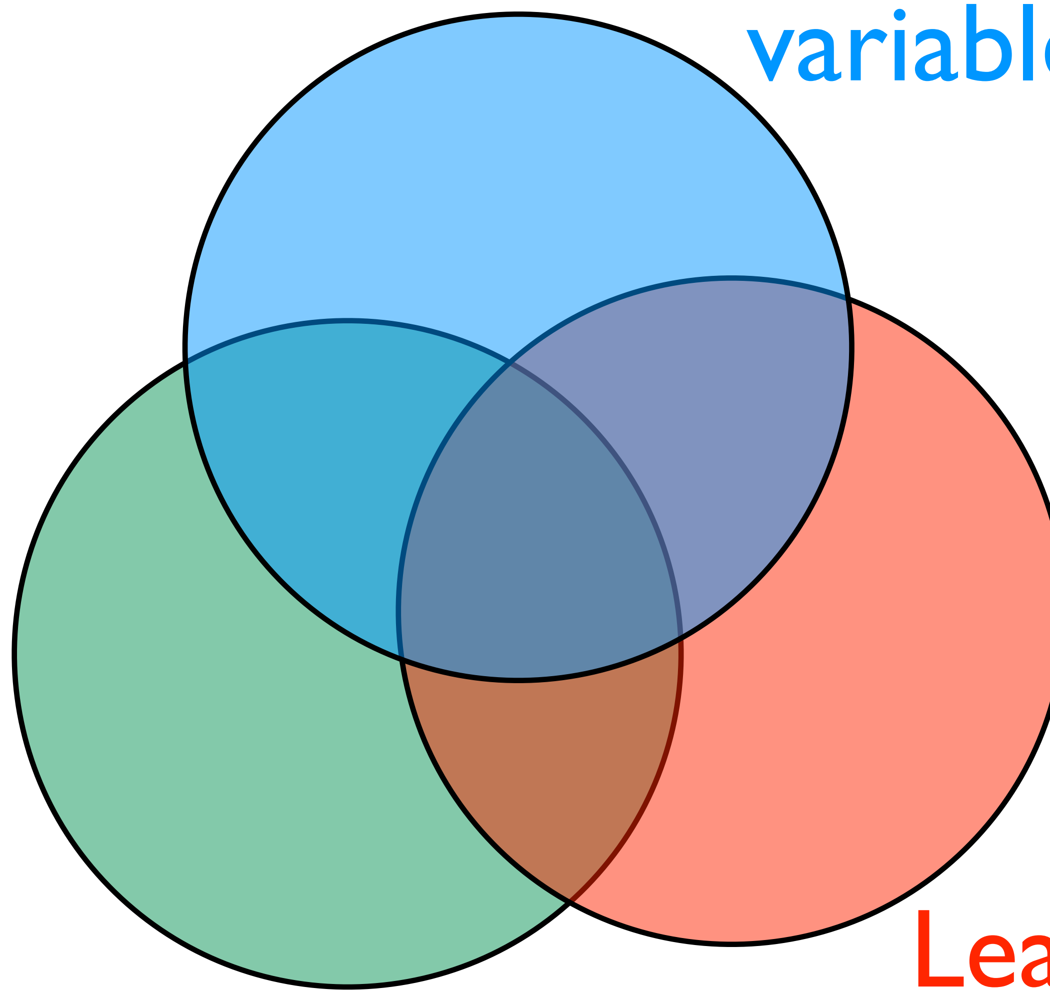
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Learning

[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
eve	new york	0,9
tom	paris	0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

tuples as random variables

```
select x.person, y.country
from bornIn x, cityIn y
where x.city=y.city
```

one world

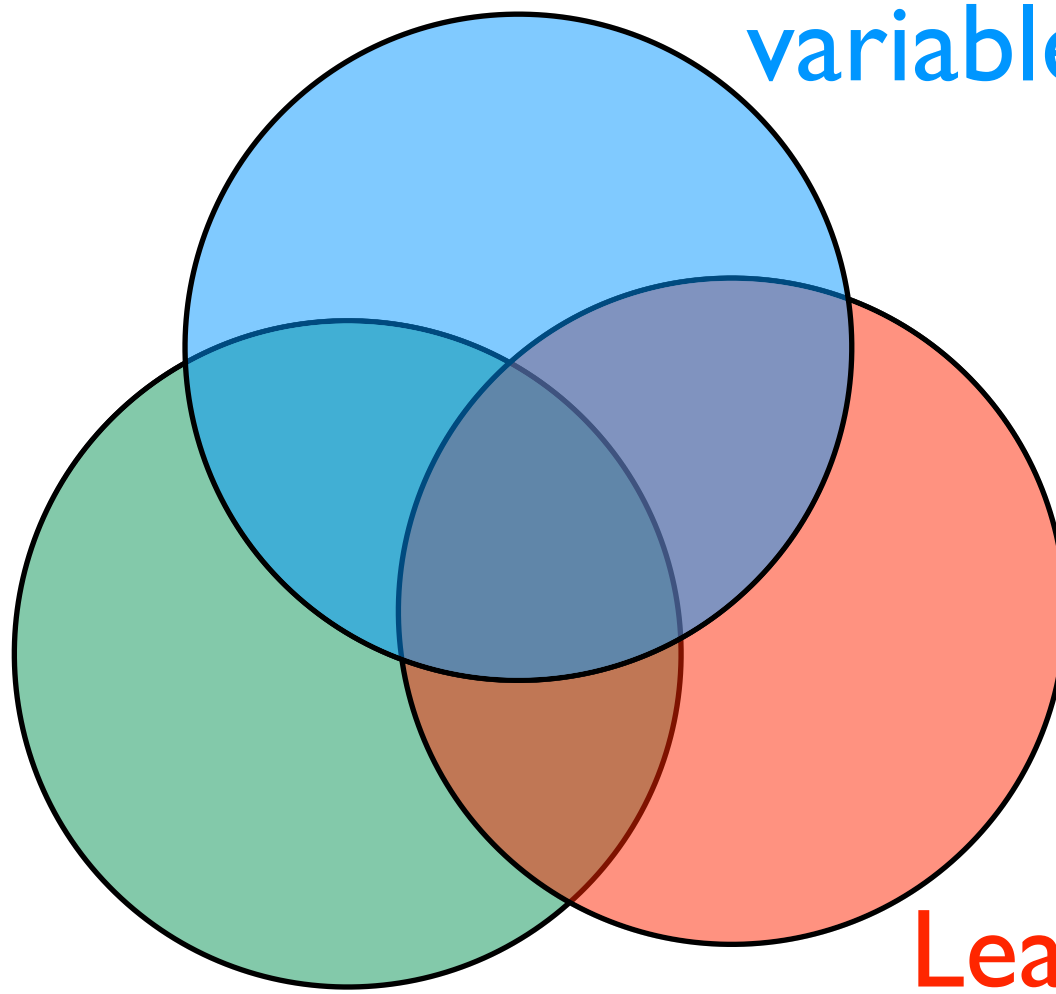
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



Learning

[Suciu et al 2011]

Probabilistic Databases

several possible worlds

bornIn

person	city	P
ann	london	0,87
bob	york	0,95
		0,9
		0,56

cityIn

city	country	P
london	uk	0,99
york	uk	0,75
paris	usa	0,4

probabilistic tables + database queries
→ distribution over possible worlds

tuples as random variables

```
select
from bornIn x, cityIn y
where x.city=y.city
```

one world

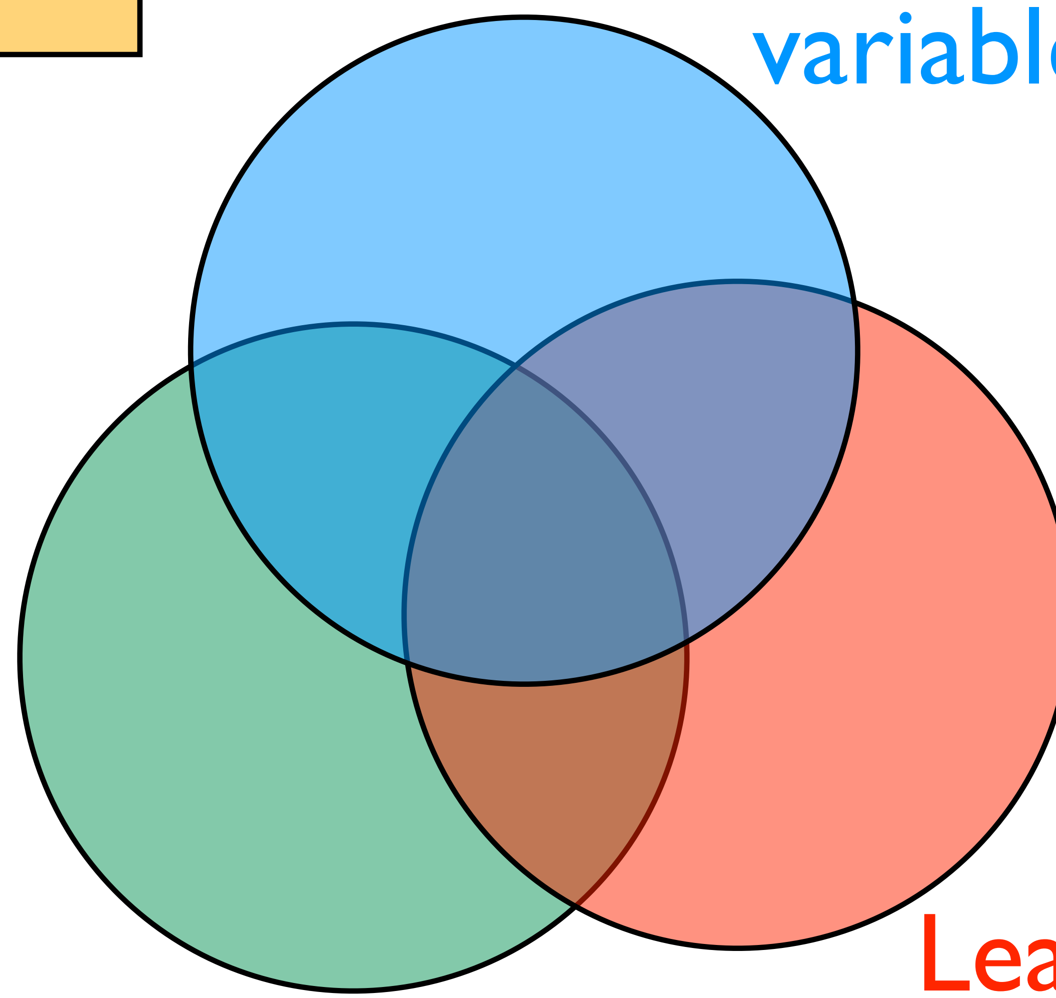
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

relational database



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/

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

same query -
probabilities handled implicitly

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

0.96x0.99=0.95

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

0.9×0.93=0.83

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

0.87x0.93=0.80

[Example from Suciu et al 2011]

Querying: probabilistic db

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

HeadquarteredIn

Company	City	P
microsoft	redmond	1.00
ibm	san_jose	0.99
emirates_airlines	dubai	0.93
honda	torrance	0.93
horizon	seattle	0.93
egyptair	cairo	0.93
adobe	san_jose	0.93
...	...	...

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and
y.City='san_jose'
```

answer tuples ranked by probability

Product	Company	P
personal_computer	ibm	0.95
adobe_indesign	adobe	0.83
adobe_dreamweaver	adobe	0.80

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

ProducesProduct

Company	Product	P
sony	walkman	0.96
microsoft	mac_os_x	0.96
ibm	personal_computer	0.96
microsoft	mac_os	0.9
adobe	adobe_indesign	0.9
adobe	adobe_dreamweaver	0.87
...	...	...

```
0.96::producesProduct(sony,walkman).
0.96::producesProduct(microsoft,mac_os_x).
0.96::producesProduct(ibm,personal_computer).
0.9::producesProduct(microsoft,mac_os).
0.9::producesProduct(adobe,adobe_indesign).
0.87::producesProduct(adobe,adobe_dreamweaver).
...
```


[Example from Suciu et al 2011]

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
    producesProduct(Company,Product),
    headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with tuple-level uncertainty in ProbLog?

```
select x.Product, x.Company
from ProducesProduct x, HeadquarteredIn y
where x.Company=y.Company and y.City='san_jose'
```

```
result(Product,Company) :-
   producesProduct(Company,Product),
   headquarteredIn(Company,san_jose).
query(result(_,_)).
```

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

PDB with attribute-level uncertainty in ProbLog?

color

item	color	P
mug	green blue	0.65 0.35
plate	pink red purple	0.23 0.14 0.63

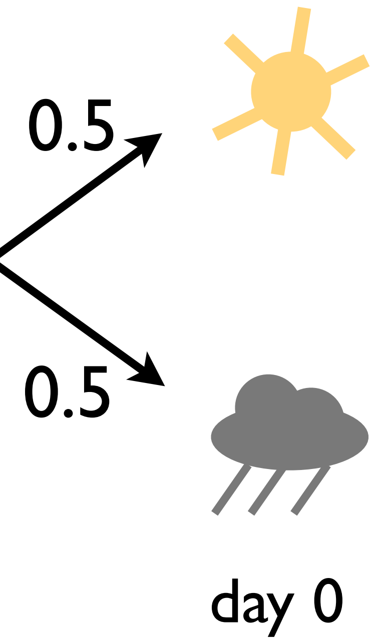
```
0.65::color(mug,green); 0.35::color(mug,blue).
0.23::color(plate,pink); 0.14::color(plate,red);
                                 0.63::color(plate,purple).
```

ProbLog by example:

Rain or sun?

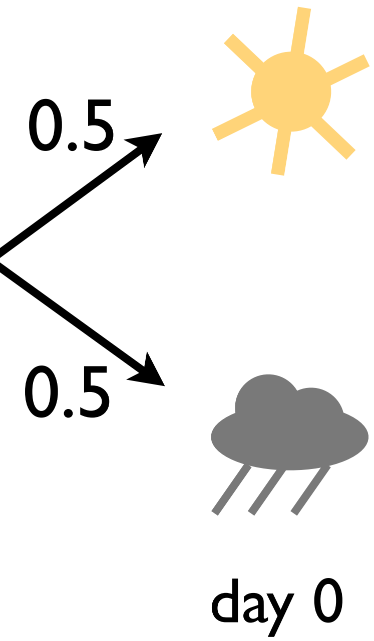
ProbLog by example:

Rain or sun?



ProbLog by example:

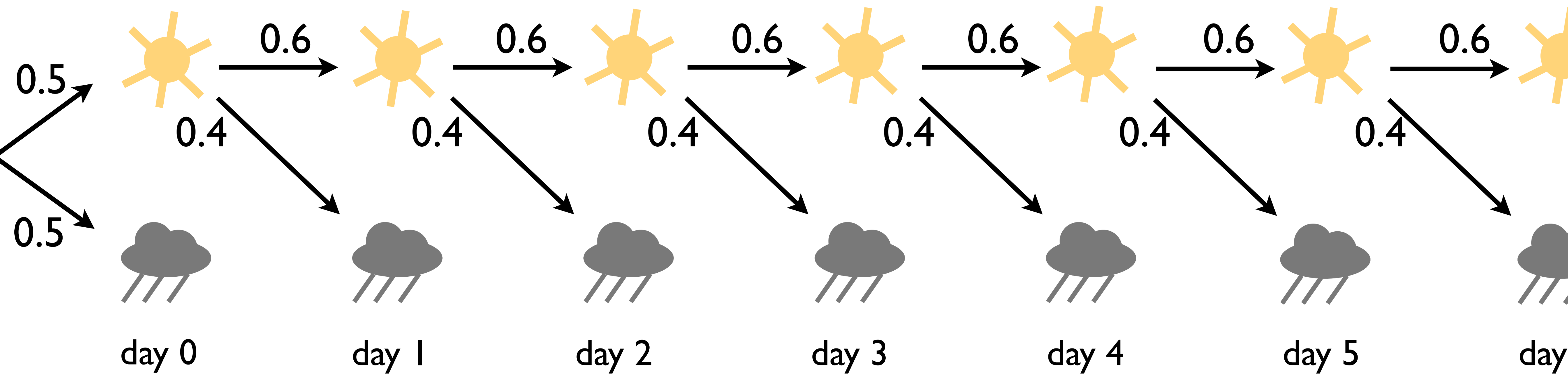
Rain or sun?




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

ProbLog by example:

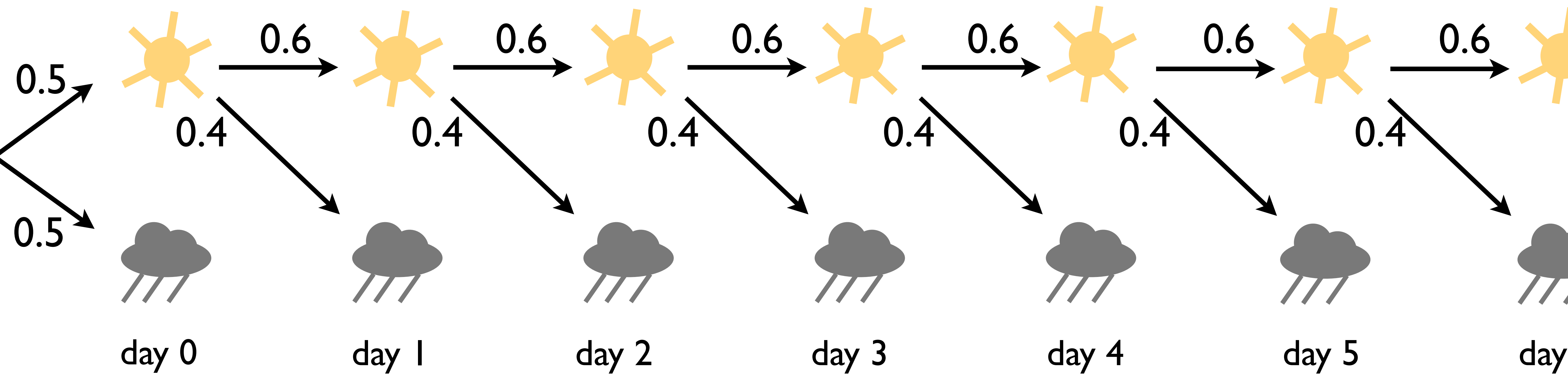
Rain or sun?




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

ProbLog by example:

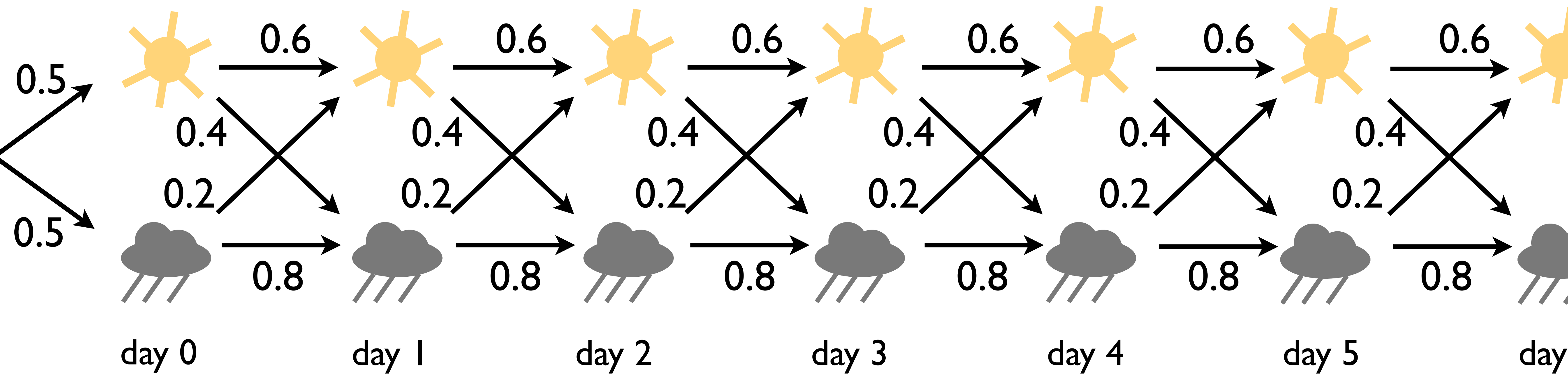
Rain or sun?




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

ProbLog by example:

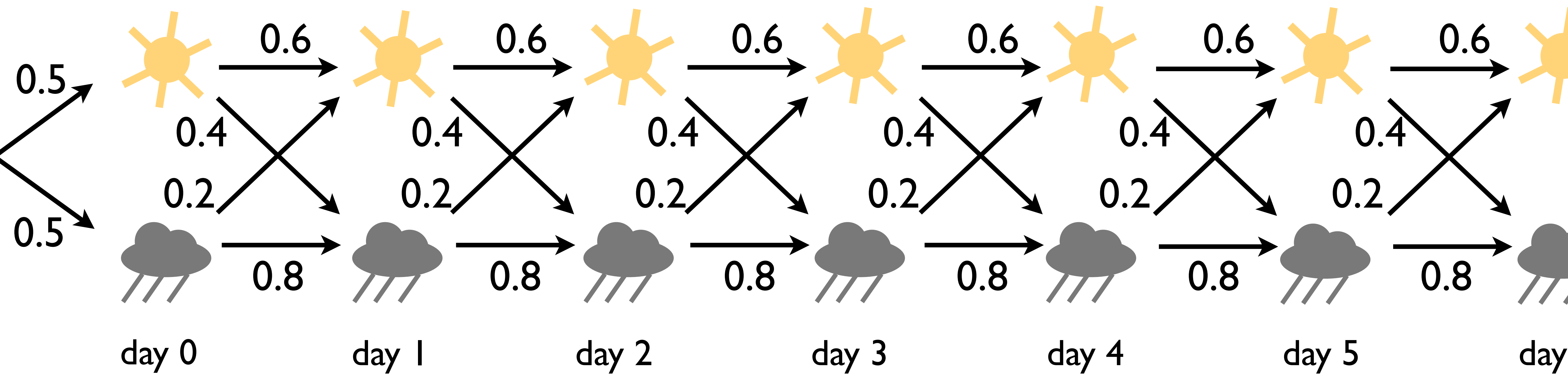
Rain or sun?




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

ProbLog by example:

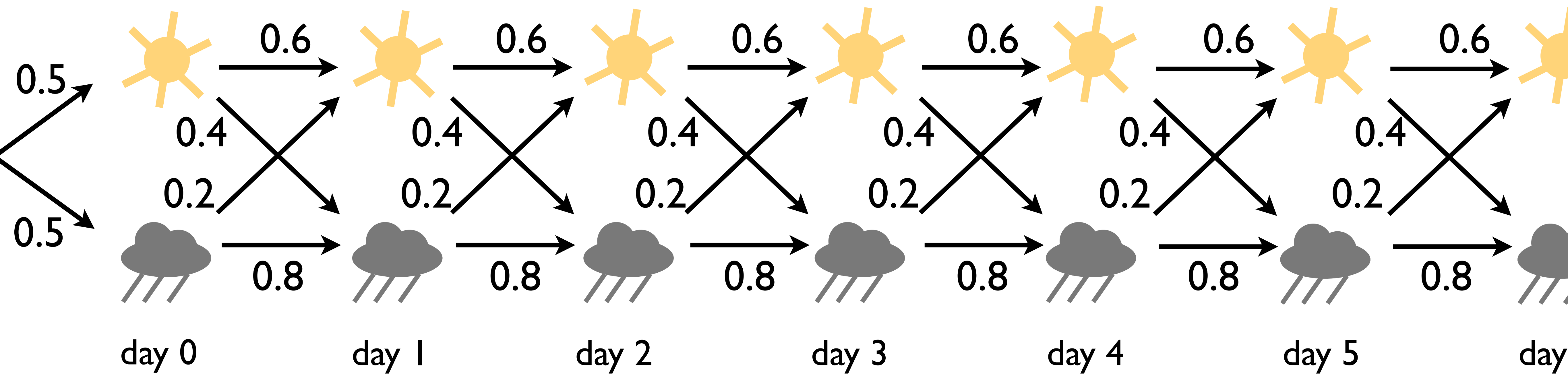
Rain or sun?




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

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?



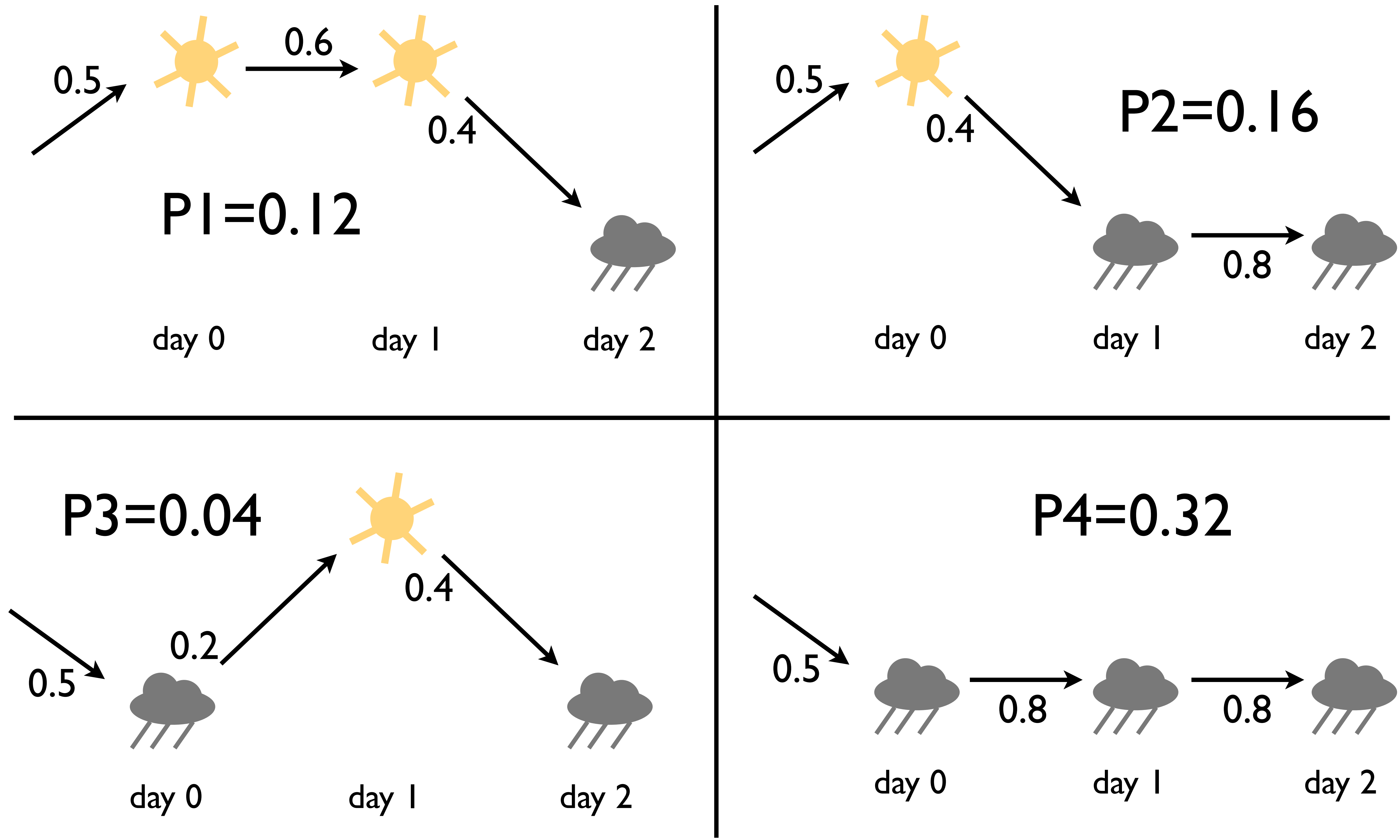

```
0.5::weather(sun,0) ; 0.5::weather(rain,0).

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

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

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).

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).

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).

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).

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

[Sato and Kameya, JAIR 01]

PRISM

- Another probabilistic Prolog based on the distribution semantics
- Mutual exclusiveness assumption
  - allows for efficient inference by dynamic programming, cf. probabilistic grammars
  - but excludes certain models, e.g., smokers
- a different syntax
- http://sato-www.cs.titech.ac.jp/prism/

Distribution Semantics

- **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

```
person(bob).
```

- probabilistic facts

```
0.5::stress(bob).
```

- annotated disjunctions

```
0.5::stress(X) :- person(X).
0.4::a(X); 0.3::b(X); 0.2::c(X); 0.1::d(X) :- q(X).
0.5::weather(sun,0) ; 0.5::weather(rain,0).
```

- flexible probabilities

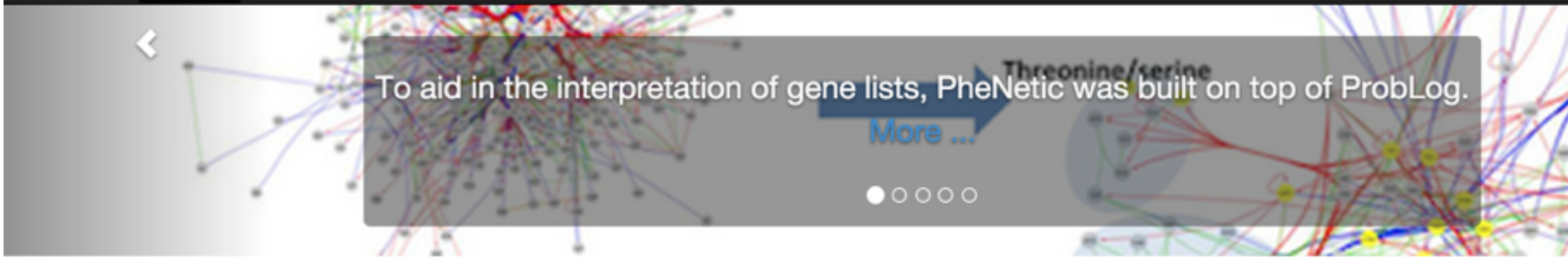
```
P::pack(Item) :- weight(Item,W), P is 1.0/W.
```

- Prolog clauses

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

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

ProbLog | Home | Download | Publications | Help | People | Interactive



Introduction.

Probabilistic logic programs are logic programs in which some of the facts are annotated with probabilities.

ProbLog is a tool that allows you to intuitively build programs that do not only encode **complex interactions** between a large sets of **heterogenous components** bu **uncertainties** that are present in real-life situations.

The engine tackles several tasks such as computing the marginals given evidence and learning from (partial) interpretations. ProbLog is a suite of efficient algorithms tasks. It is based on a conversion of the program and the queries and evidence to a weighted Boolean formula. This allows us to reduce the inference tasks to well-s weighted model counting, which can be solved using state-of-the-art methods known from the graphical model and knowledge compilation literature.

The Language. Probabilistic Logic Programming.

ProbLog makes it easy to express complex, probabilistic models.

```
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).
```

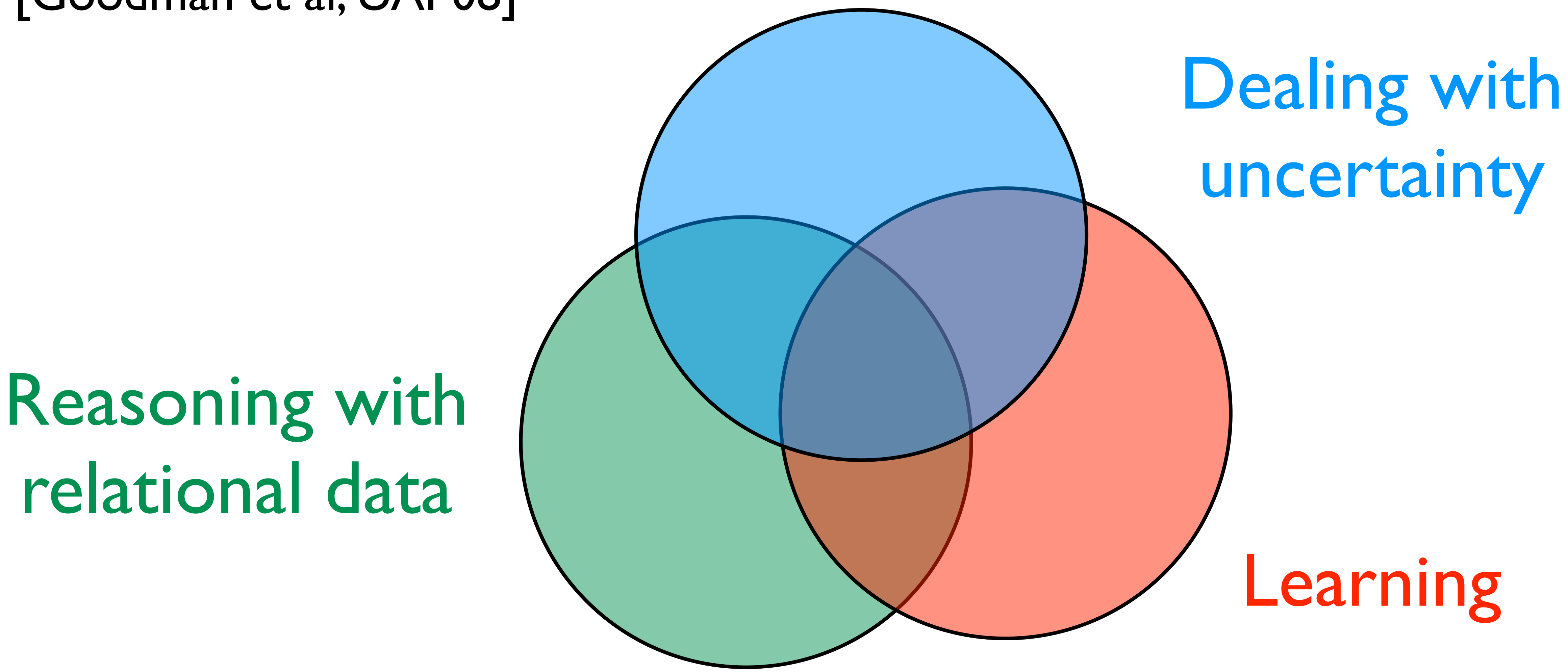
Some Probabilistic Programming Languages outside LP

- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- Venture [Mansingha et al.]
- Anglican and Probabilistic-C [Wood et al].
- and many more appearing recently

Church

probabilistic functional programming

[Goodman et al, UAI 08]

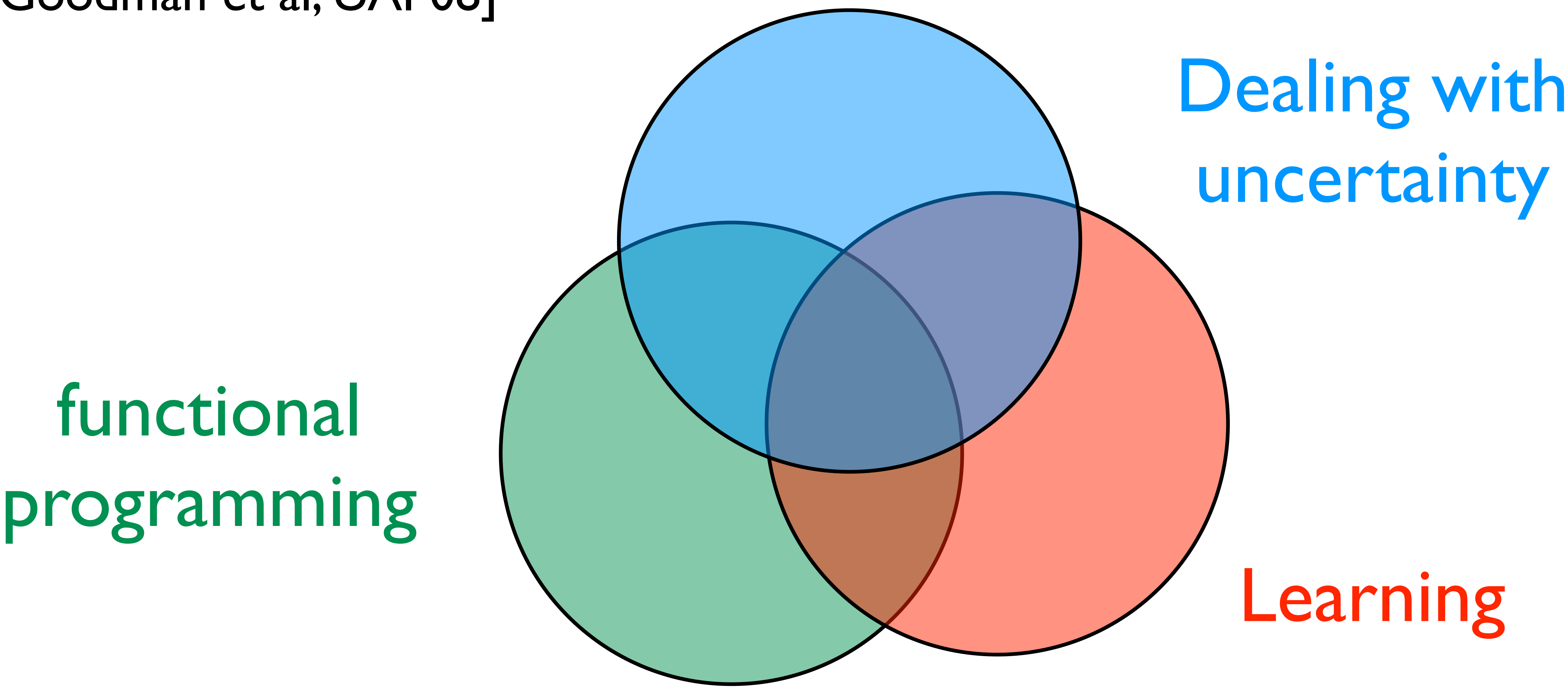


http://probmods.org

Church

probabilistic functional programming

[Goodman et al, UAI 08]



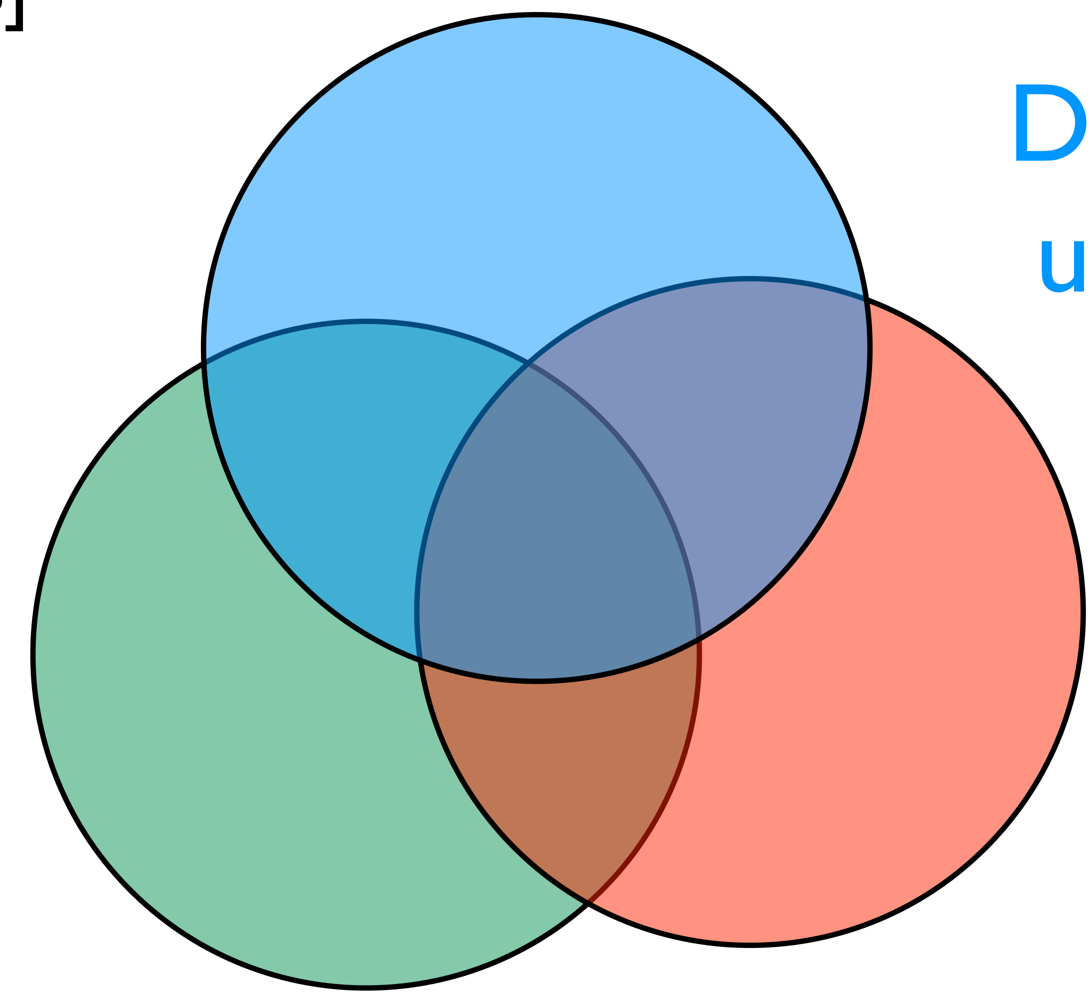

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```


http://probmods.org

Church

probabilistic functional programming

[Goodman et al, UAI 08]



one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```

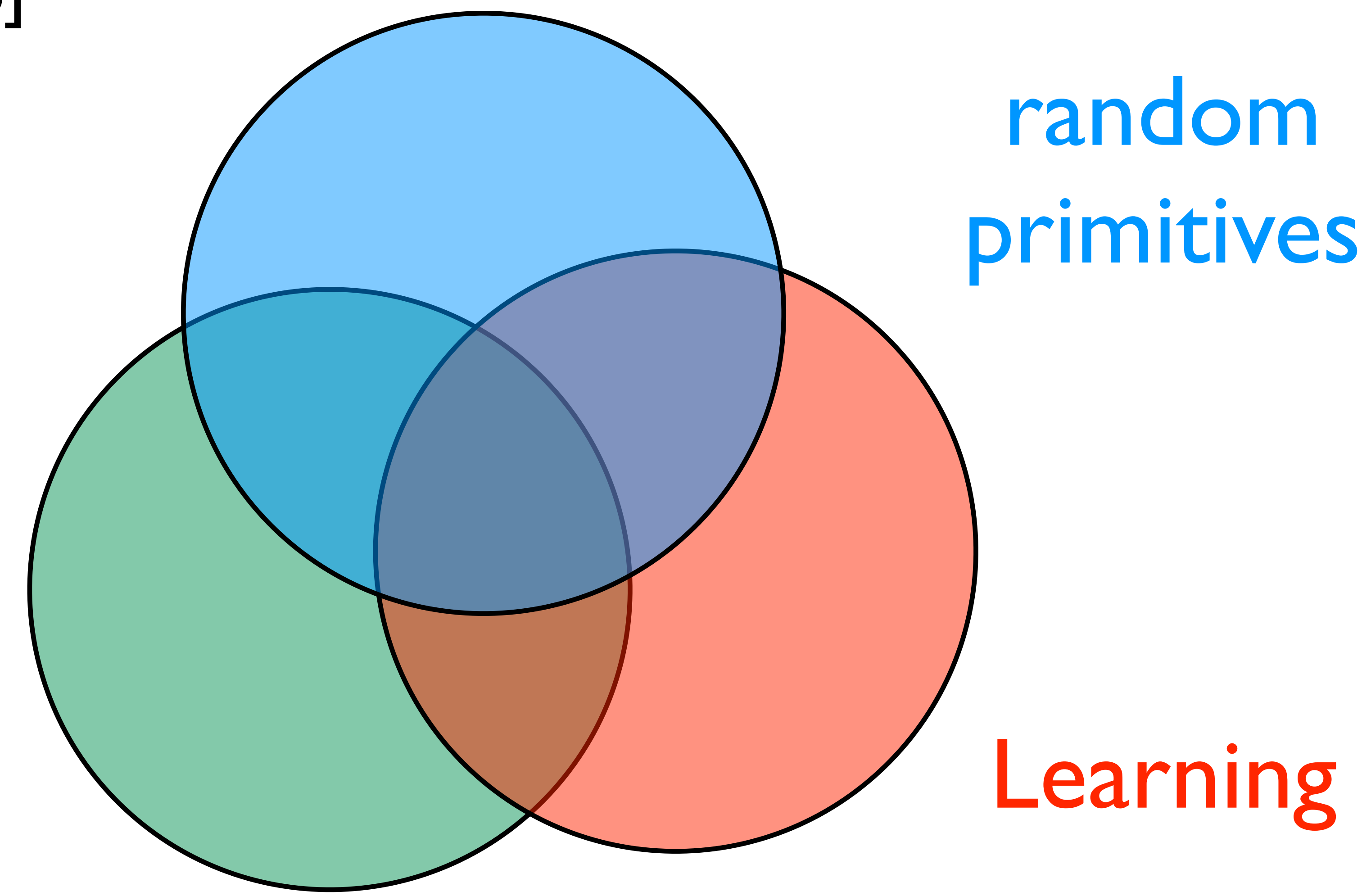

http://probmods.org

Church

probabilistic functional programming

[Goodman et al, UAI 08]

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                x)))

(map randplus5 '(1 2 3))
```



one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```


http://probmods.org

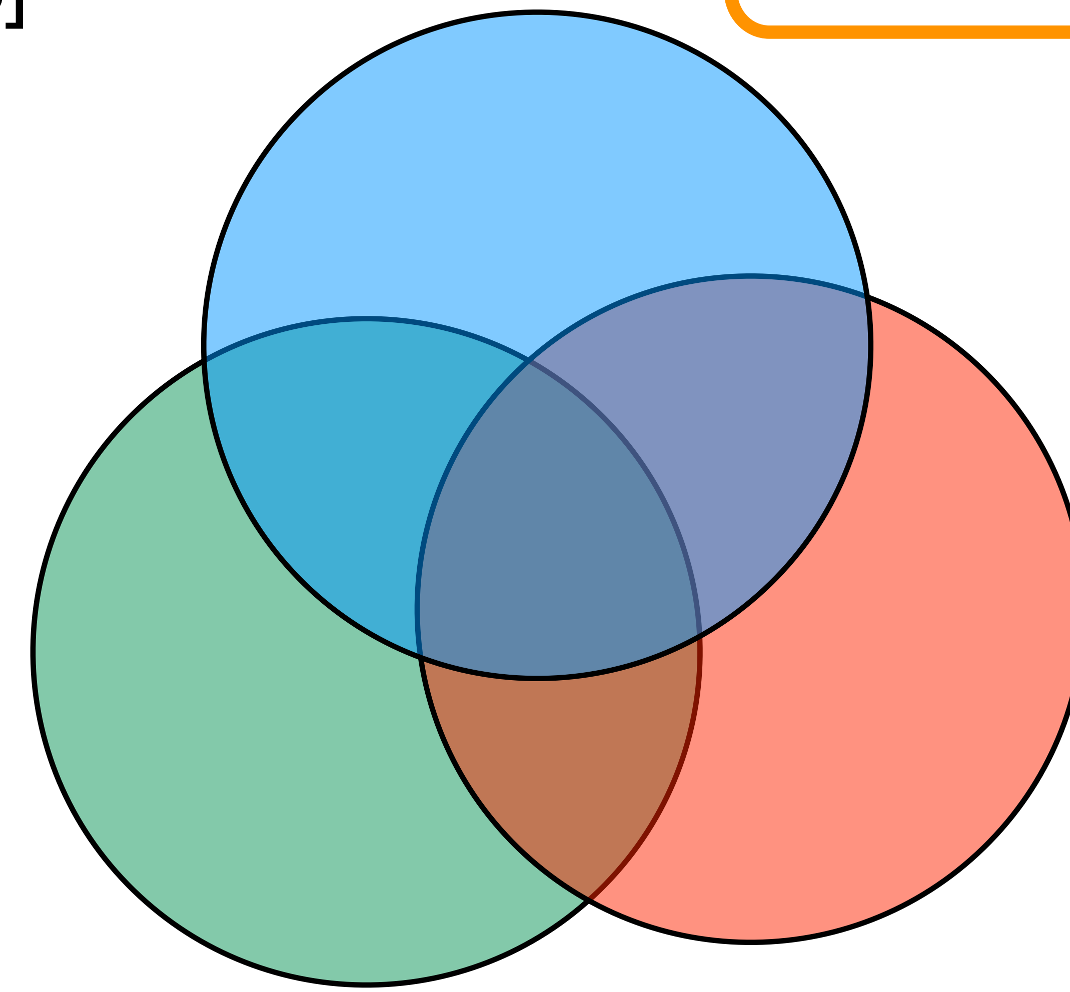
Church

probabilistic functional programming

[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                (+ x 5)
               x)))

(map randplus5 '(1 2 3))
```



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```


http://probmods.org

Church

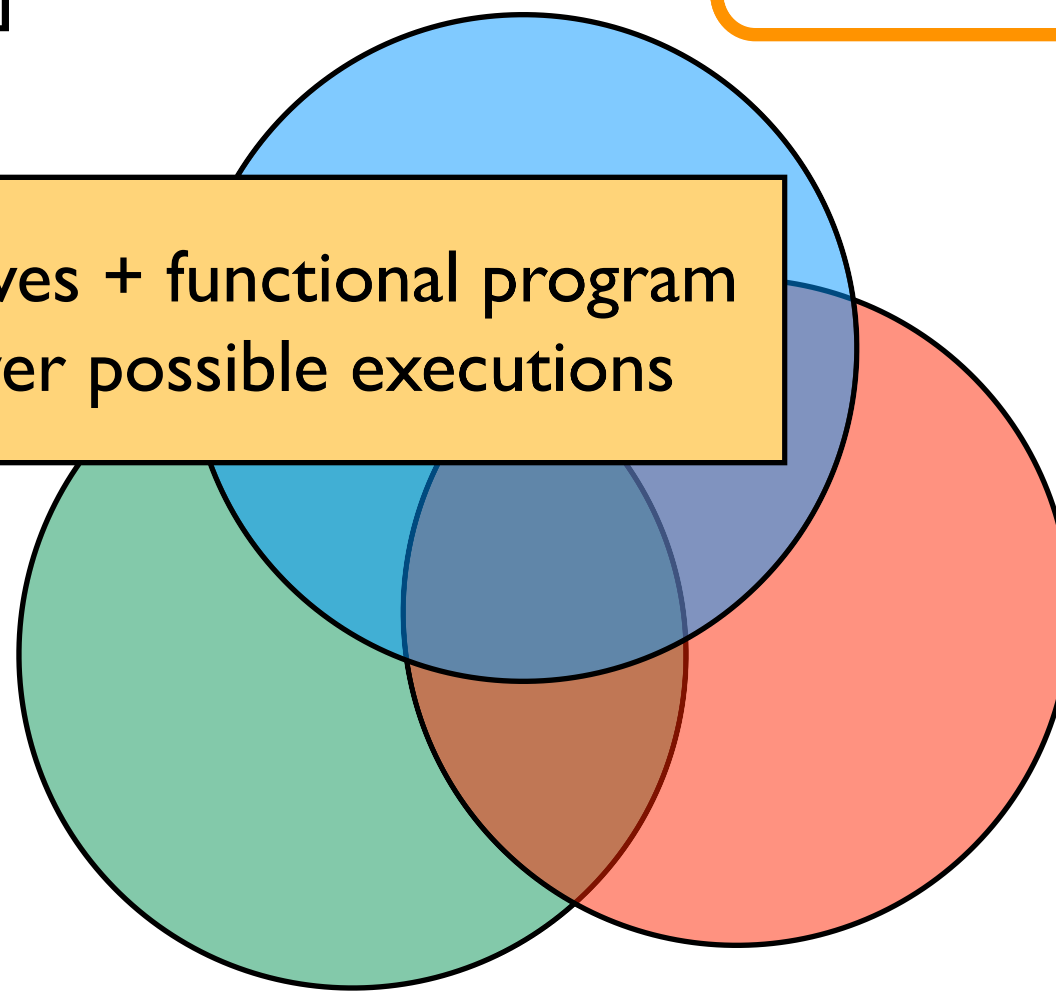
probabilistic functional programming

[Goodman et al, UAI 08]

several possible executions

```
(define randplus5
 (lambda (x) (if (flip 0.6)
                 (+ x 5)
                x)))

(map randplus5 '(1 2 3))
```

probabilistic primitives + functional program
→ distribution over possible executions



random primitives

functional programming

Learning

one execution

```
(define plus5 (lambda (x) (+ x 5)))

(map plus5 '(1 2 3))
```


http://probmods.org

Church Example

arguments

definition

function

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2 3))
```

random primitive
P(true) = 0.6

apply function
to each element

list

Church Example

arguments

function

definition

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2 3))
```

apply function
to each element

list

random primitive
P(true) = 0.6

direct execution: samples result

Church Example

function

arguments

definition

```
(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2 3))
```

apply function to each element

list

random primitive
P(true) = 0.6

direct execution: samples result

sampling also supports continuous RVs, e.g.,
`(* (gaussian 0 1) (gaussian 0 1))`

Computing probability distribution

```
(
enumeration-query

(define randplus5 (lambda (x) (if (flip 0.6) (+ x 5) x)))

(map randplus5 '(1 2))

true
)
```

enumerates all executions & sums probabilities per result

query

evidence

(((1 2) (1 7) (6 2) (6 7)) (0.16 0.24 0.24 0.36))

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```

remember first value & reuse for all later calls

(((1 1) (6 6)) (0.4 0.6))

Concept: stochastic memoization

Stochastic Memoization

```
(define randplus5 (mem (lambda (x) (if (flip 0.6) (+ x 5) x))))

(map randplus5 '(1 1))
```

remember first value &
reuse for all later calls

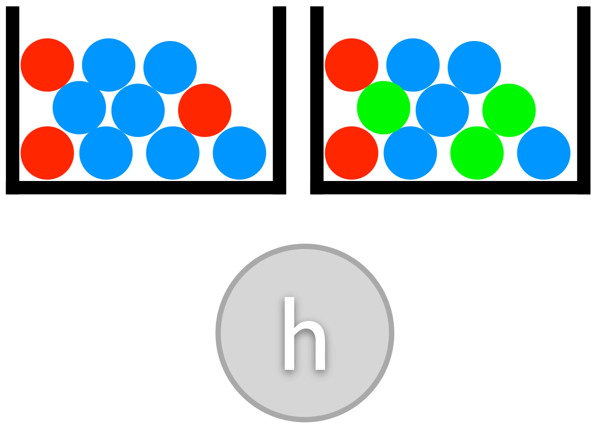
(((1 1) (6 6)) (0.4 0.6))

Concept:
stochastic
memoization

ProbLog always memoizes
PRISM never memoizes
Church allows fine-grained choice

Church by example:

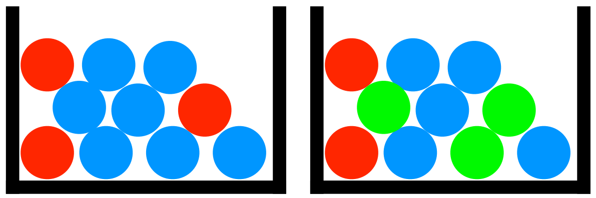
A bit of gambling



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

Church by example:

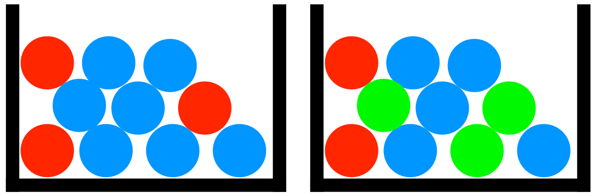
A bit of gambling



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

Church by example:

A bit of gambling

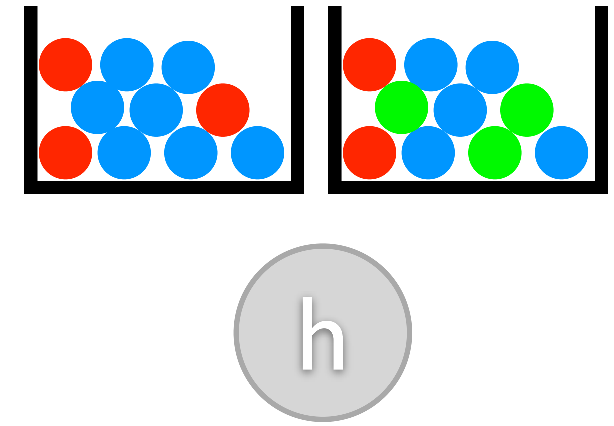


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling

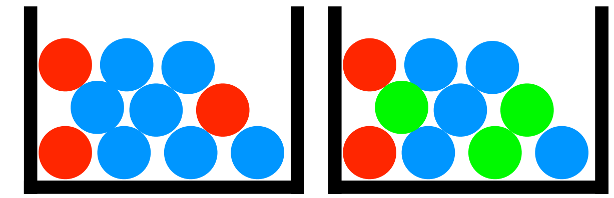


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))
```

Church by example:

A bit of gambling

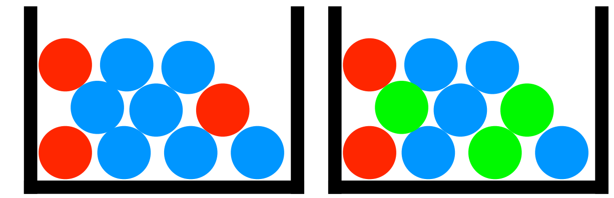


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
```

Church by example:

A bit of gambling

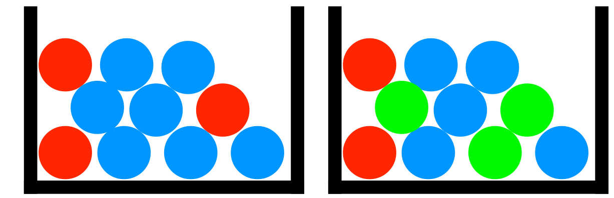


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

A bit of gambling

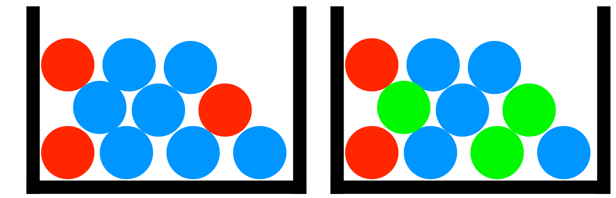


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))
```

Church by example:

A bit of gambling

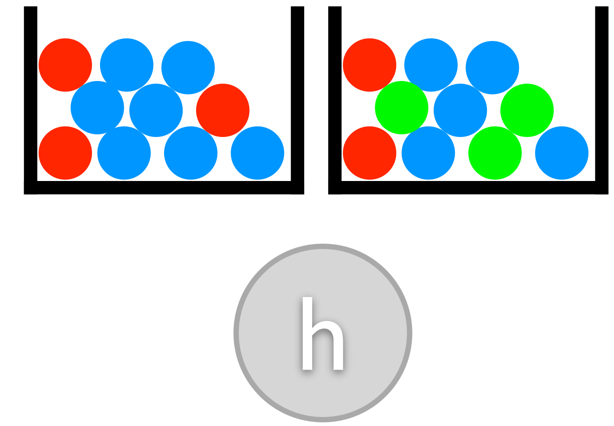


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
```

Church by example:

A bit of gambling

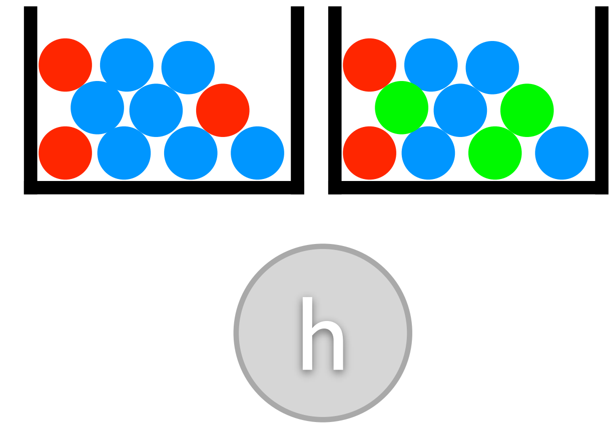


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

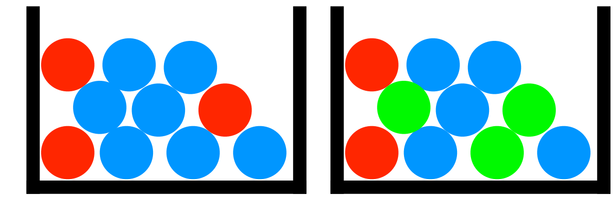


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))
```

Church by example:

A bit of gambling

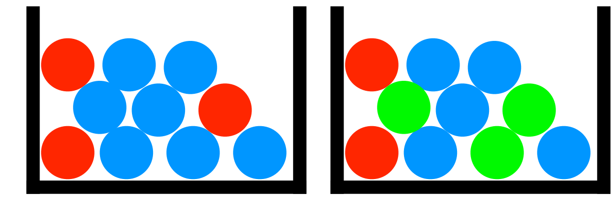


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

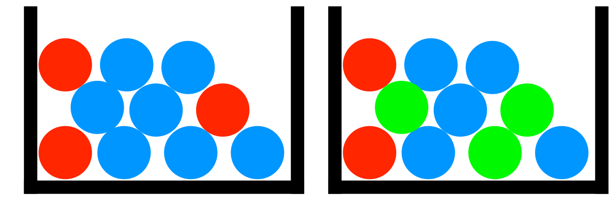


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))
```

Church by example:

A bit of gambling

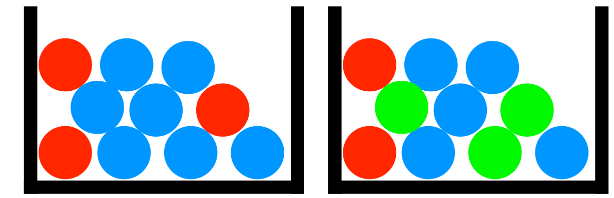


- toss (biased) coin & draw ball from each urn
- win if (heads and a red ball) or (two balls of same color)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                 (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

Church 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)

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))
```

Sampling execution

```
(define heads (mem (lambda () (flip 0.4))))

(define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

(define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))

(define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

(define win1 (and (heads) redball))

(define win2 (equal? (color1) (color2)))

(define win (or win1 win2))

win
```

query

Marginals via enumeration

```
(enumeration-query
 (define heads (mem (lambda () (flip 0.4))))
 (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))
 (define color2 (mem (lambda ()
                  (multinomial '(red green blue) '(0.2 0.3 0.5)))))
 (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))
 (define win1 (and (heads) redball))
 (define win2 (equal? (color1) (color2)))
 (define win (or win1 win2))

 win
 true )
```

query

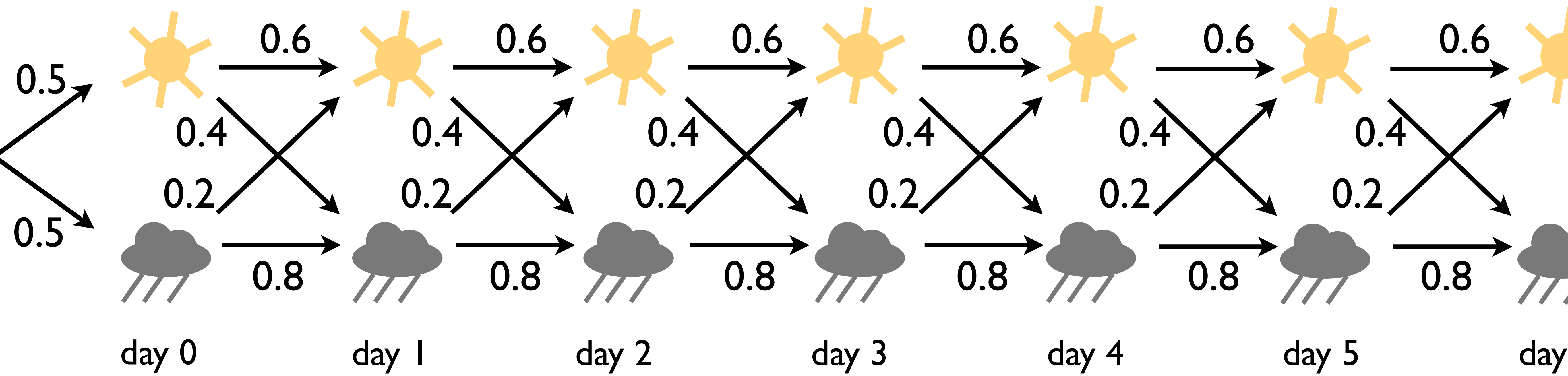
evidence

Histogram via sampling

```
(repeat 1000 (lambda ()
          (rejection-query

  (define heads (mem (lambda () (flip 0.4))))

  (define color1 (mem (lambda () (if (flip 0.3) 'red 'blue))))

  (define color2 (mem (lambda ()
                   (multinomial '(red green blue) '(0.2 0.3 0.5)))))

  (define redball (or (equal? (color1) 'red) (equal? (color2) 'red)))

  (define win1 (and (heads) redball))

  (define win2 (equal? (color1) (color2)))

  (define win (or win1 win2))

  win        query

  true ))))
       evidence
```

Church by example:

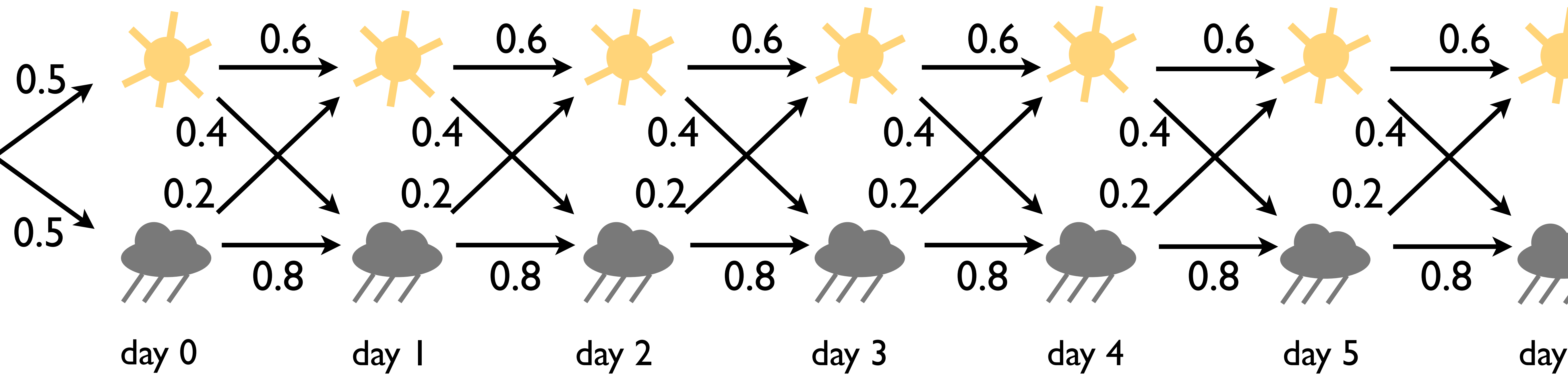
Rain or sun?




```
(define weather (mem (lambda (day) (if (equal? day 0)
                                       (weather0)
                                       (weatherN day (- day 1))))))
(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))
(define weatherN (lambda (today yesterday)
                  (if (equal? (weather yesterday) 'rain)
                      (if (flip 0.2) 'sun 'rain)
                      (if (flip 0.6) 'sun 'rain))))
```

Church by example:

Rain or sun?




```
(define weather (mem (lambda (day) (if (equal? day 0)
                                        (weather0)
                                        (weatherN day (- day 1))))))
(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))
(define weatherN (lambda (today yesterday)
                  (if (equal? (weather yesterday) 'rain)
                      (if (flip 0.2) 'sun 'rain)
                      (if (flip 0.6) 'sun 'rain))))

(list (weather 0) (weather 1) (weather 2))
```

exact inference with / without memoization

```
(enumeration-query

(define weather (mem (lambda (day) (if (equal? day 0)
                        (weather0)
                        (weatherN day (- day 1)))))))
(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))
(define weatherN (lambda (today yesterday)
                  (if (equal? (weather yesterday) 'rain)
                      (if (flip 0.2) 'sun 'rain)
                      (if (flip 0.6) 'sun 'rain))))

(list (weather 0) (weather 1))

true

 )
```

Run

```
(((rain rain) (rain sun) (sun rain) (sun sun)) (0.4 0.10000000000000002 0.2 0.30000000000000004))
```

```
(enumeration-query

(define weather (lambda (day) (if (equal? day 0)
                        (weather0)
                        (weatherN day (- day 1))))))
(define weather0 (lambda () (if (flip 0.5) 'sun 'rain)))
(define weatherN (lambda (today yesterday)
                  (if (equal? (weather yesterday) 'rain)
                      (if (flip 0.2) 'sun 'rain)
                      (if (flip 0.6) 'sun 'rain))))

(list (weather 0) (weather 1))

true

 )
```

Run

```
(((rain rain) (rain sun) (sun rain) (sun sun)) (0.30000000000000004 0.2 0.30000000000000004 0.2))
```

Probabilistic Programming Summary

- Church: functional programming + random primitives
- probabilistic generative model
- stochastic memoization
- sampling
- increasing number of probabilistic programming languages using various underlying paradigms

ProbLog	PRISM	Church
probabilistic facts & choices	probabilistic choices	random primitives
all RVs memoized	no RVs memoized	user-defined per RV
Prolog	Prolog with mutually exclusive derivations	λ-calculus functions
distribution over worlds	distribution over derivations / answers	distribution over computations / answers

Roadmap

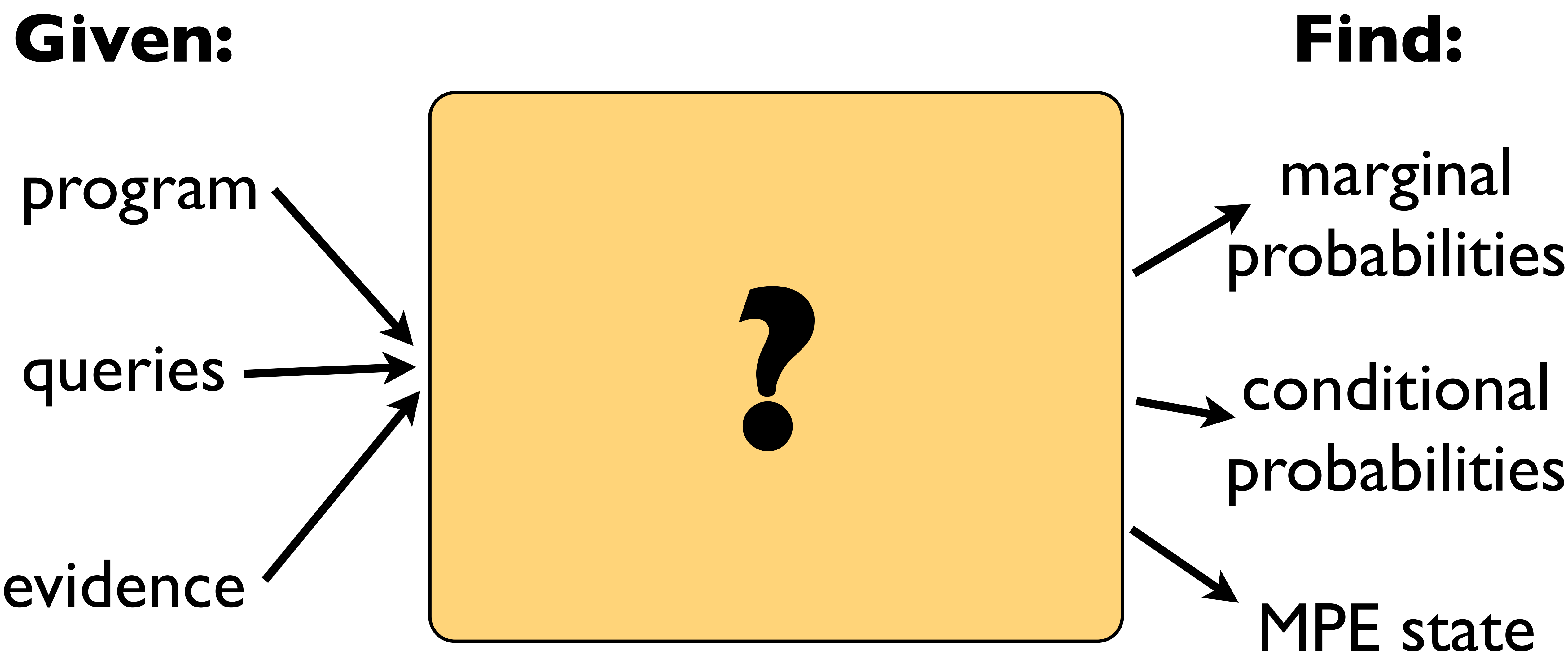
- Modeling (ProbLog and Church, another representative of PP)
- **Inference**
- Learning
- Dynamics and Decisions
- Markov Logic another representative of SRL

... with some detours on the way

Inference

- Exact inference with knowledge compilation
  - using proofs
  - using models
- Approximate inference by sampling

Answering Questions



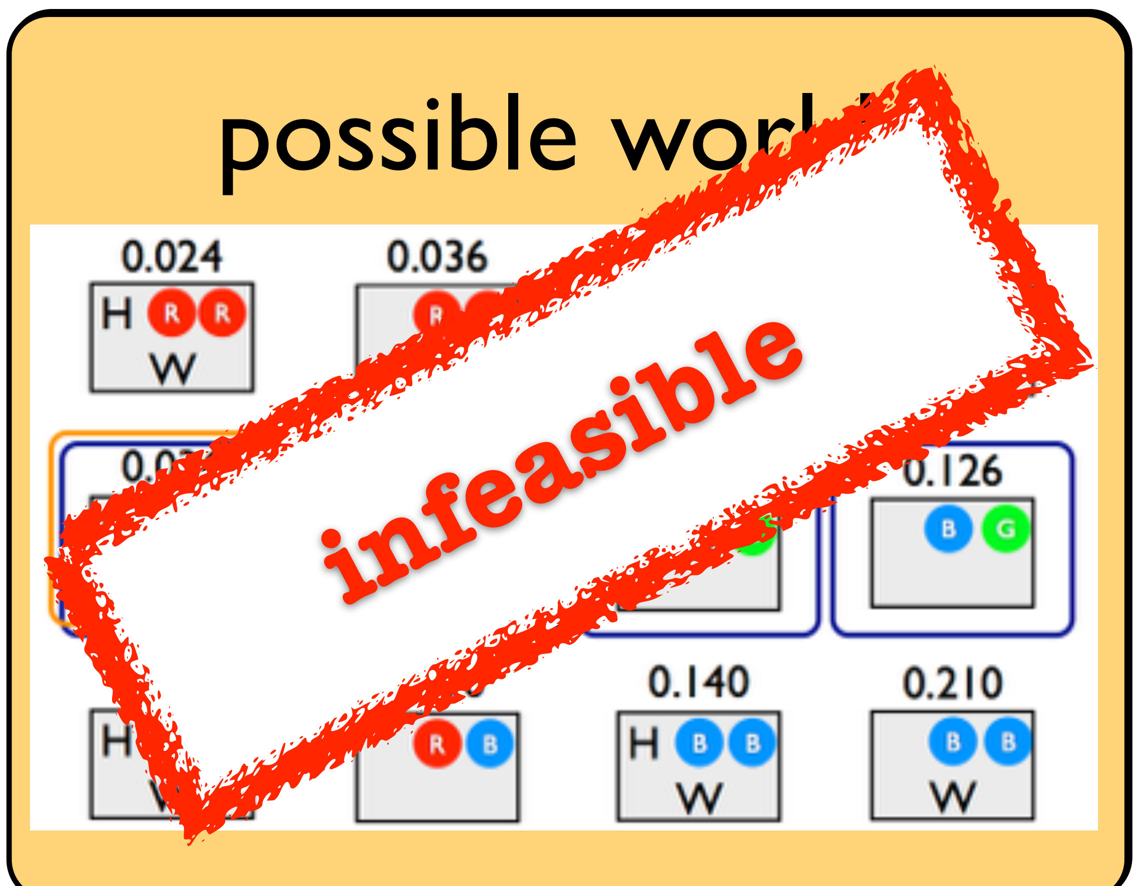
Answering Questions

Given:

program

queries

evidence



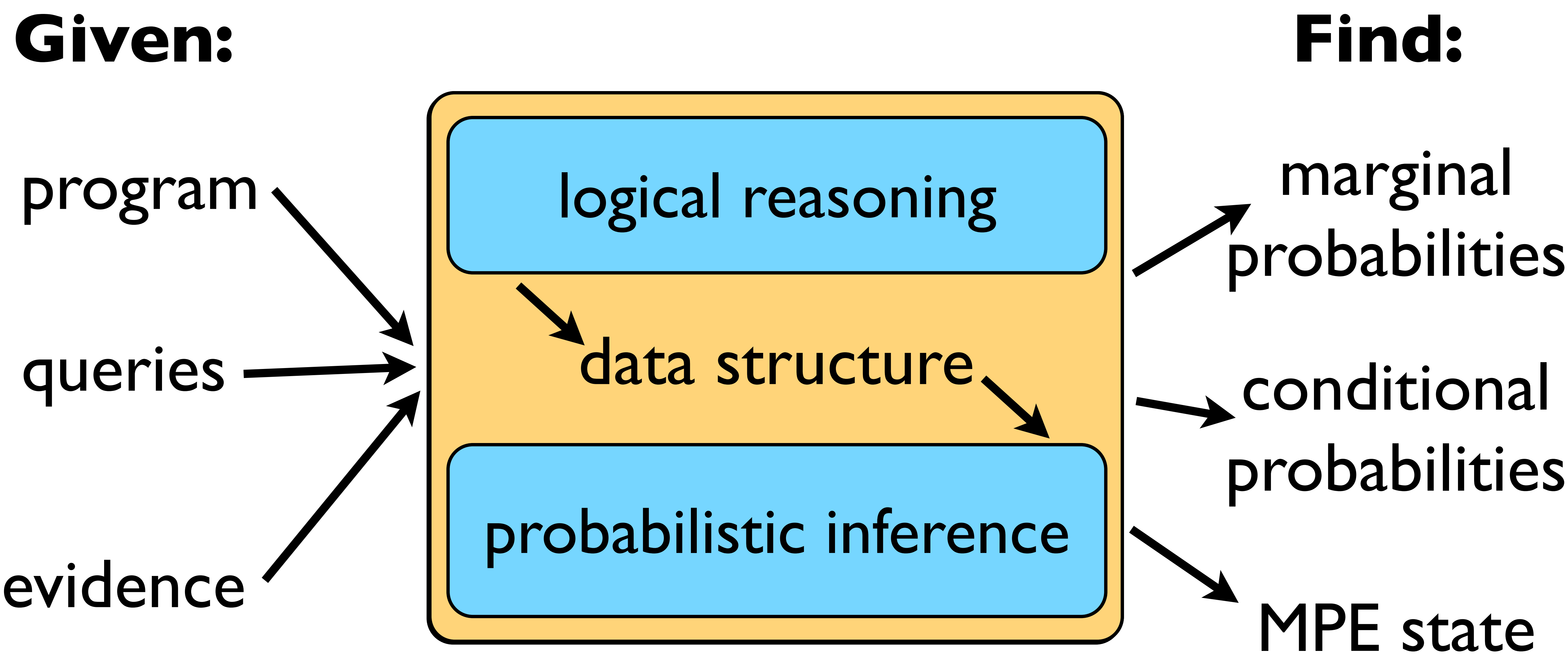
Find:

marginal probabilities

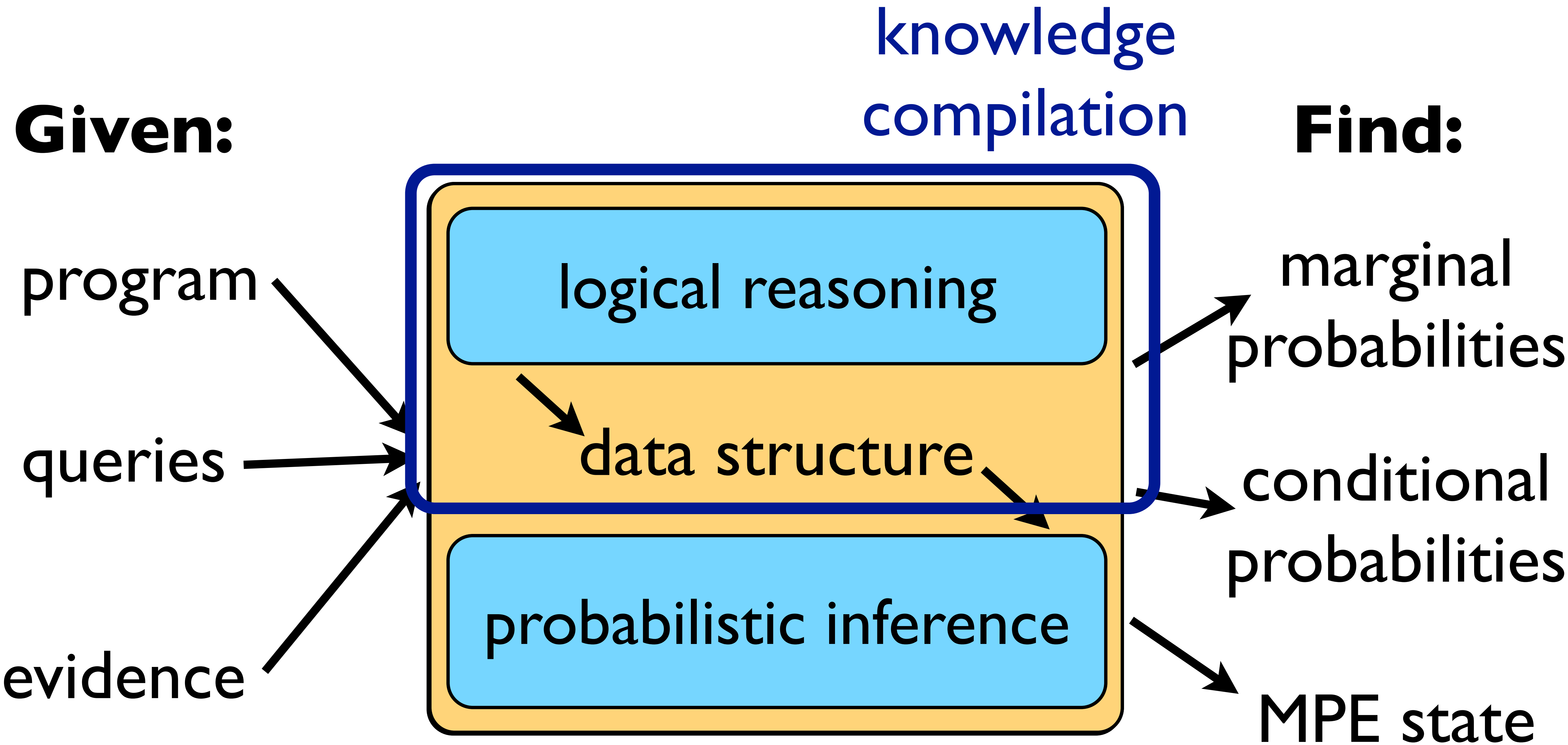
conditional probabilities

MPE state

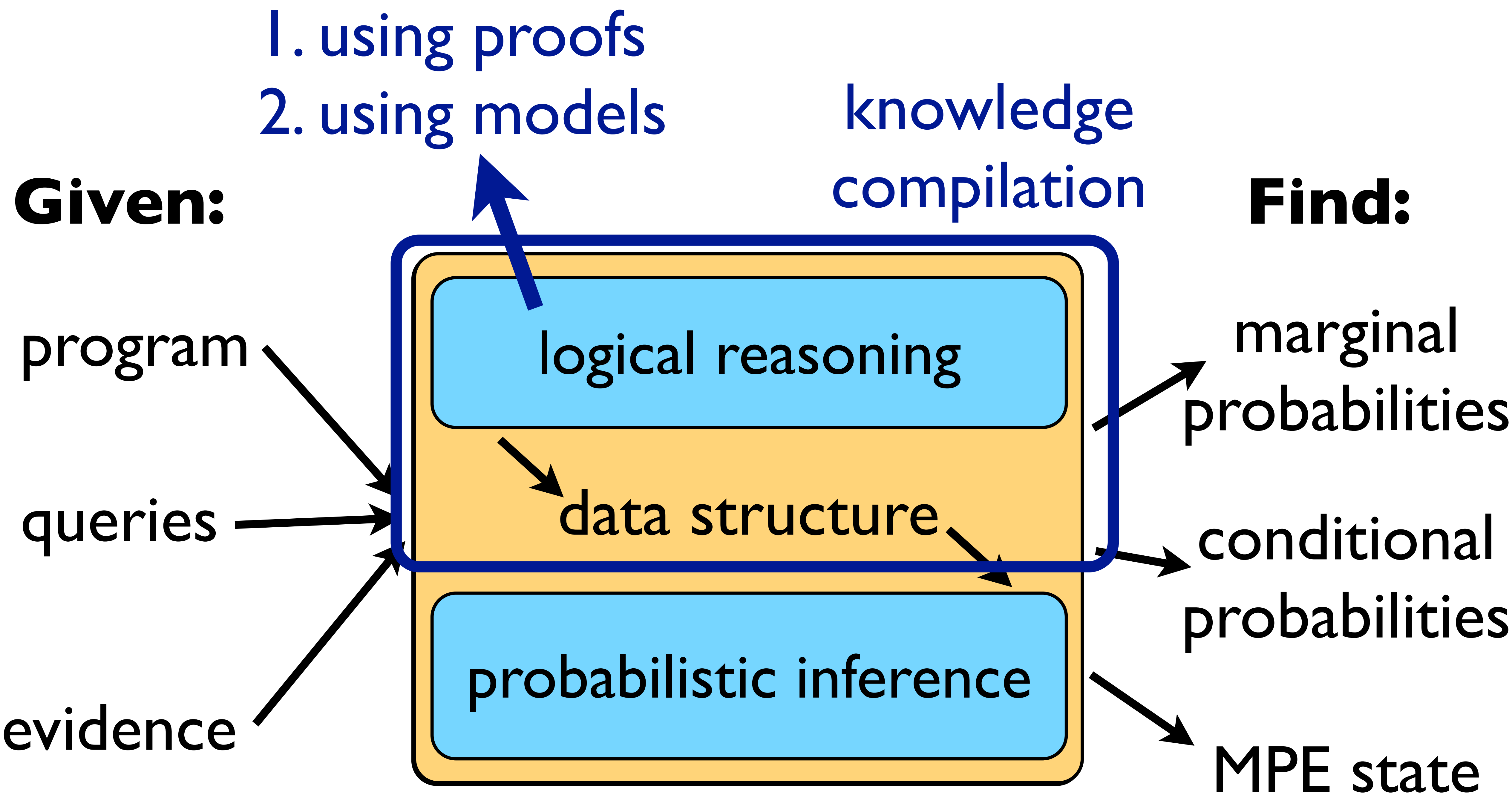
Answering Questions



Answering Questions



Answering Questions



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).
```

```
?- smokes(carl).

?- stress(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).
```

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

```
?- stress(carl).
```

```
?- influences(Y,carl),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).
```

```
?- stress(carl).          ?- influences(Y,carl),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).

?- stress(carl).        ?- influences(Y,carl),smokes(Y).

                          ?- smokes(bob).
```

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).

?- stress(carl).        ?- influences(Y,carl),smokes(Y).

?- smokes(bob).

?- stress(bob).
```

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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
?- smokes(bob).
?- stress(bob).
?- influences(Y1,bob),smokes(Y1).
```

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).

?- stress(carl).        ?- influences(Y,carl),smokes(Y).

?- smokes(bob).

?- stress(bob).        ?- influences(Y1,bob),smokes(Y1).
```

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).

?- stress(carl).
?- influences(Y,carl),smokes(Y).

?- smokes(bob).

?- stress(bob).
?- influences(Y1,bob),smokes(Y1).

?- smokes(ann).
```

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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
?- smokes(bob).
?- stress(bob).
?- influences(Y1,bob),smokes(Y1).
?- smokes(ann).
?- stress(ann).
```

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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
?- smokes(bob).
?- stress(bob).
?- influences(Y1,bob),smokes(Y1).
?- smokes(ann).
?- stress(ann).
?- influences(Y2,ann),smokes(Y2).
```

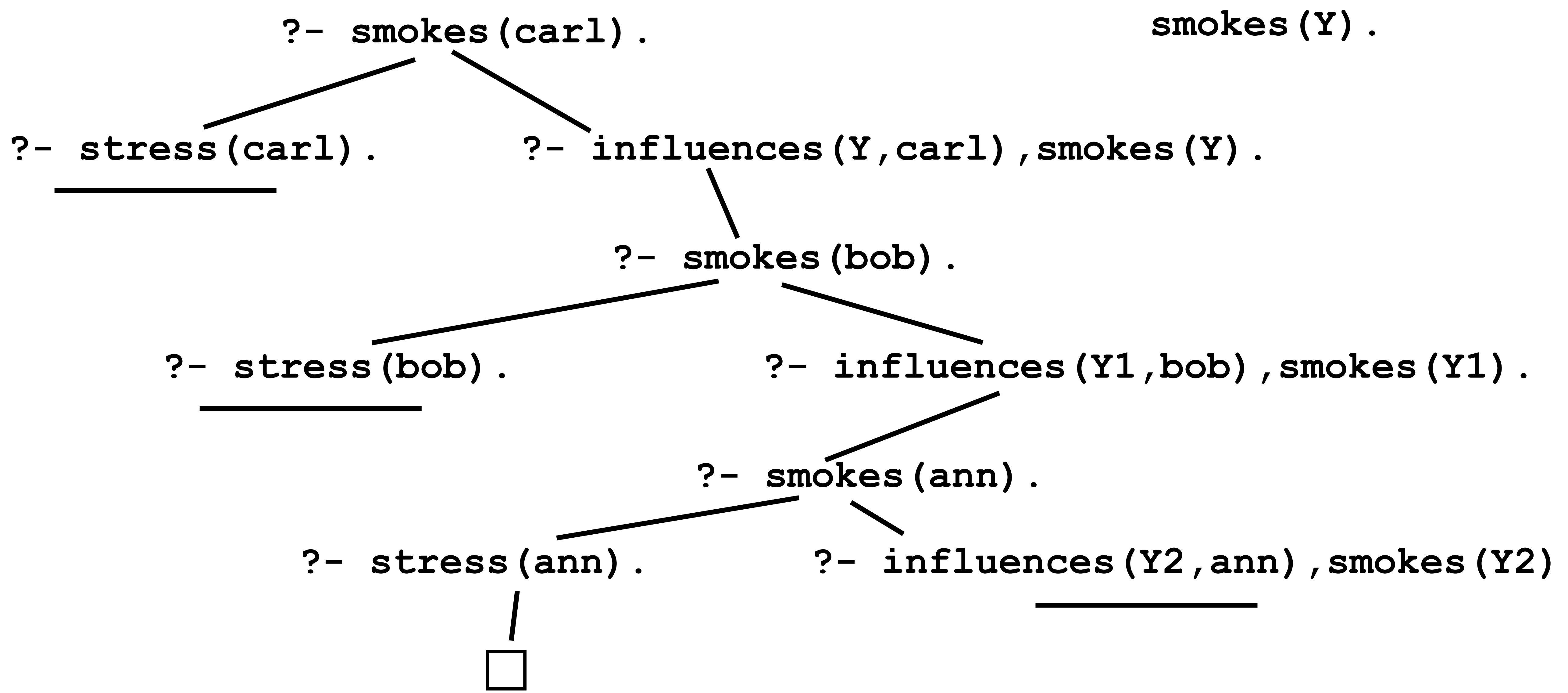
Logical Reasoning: Proofs in Prolog

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

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

```
?- smokes(carl).

?- stress(carl).        ?- influences(Y,carl),smokes(Y).

?- smokes(bob).

?- stress(bob).        ?- influences(Y1,bob),smokes(Y1).

?- smokes(ann).

?- stress(ann).        ?- influences(Y2,ann),smokes(Y2).

□
```

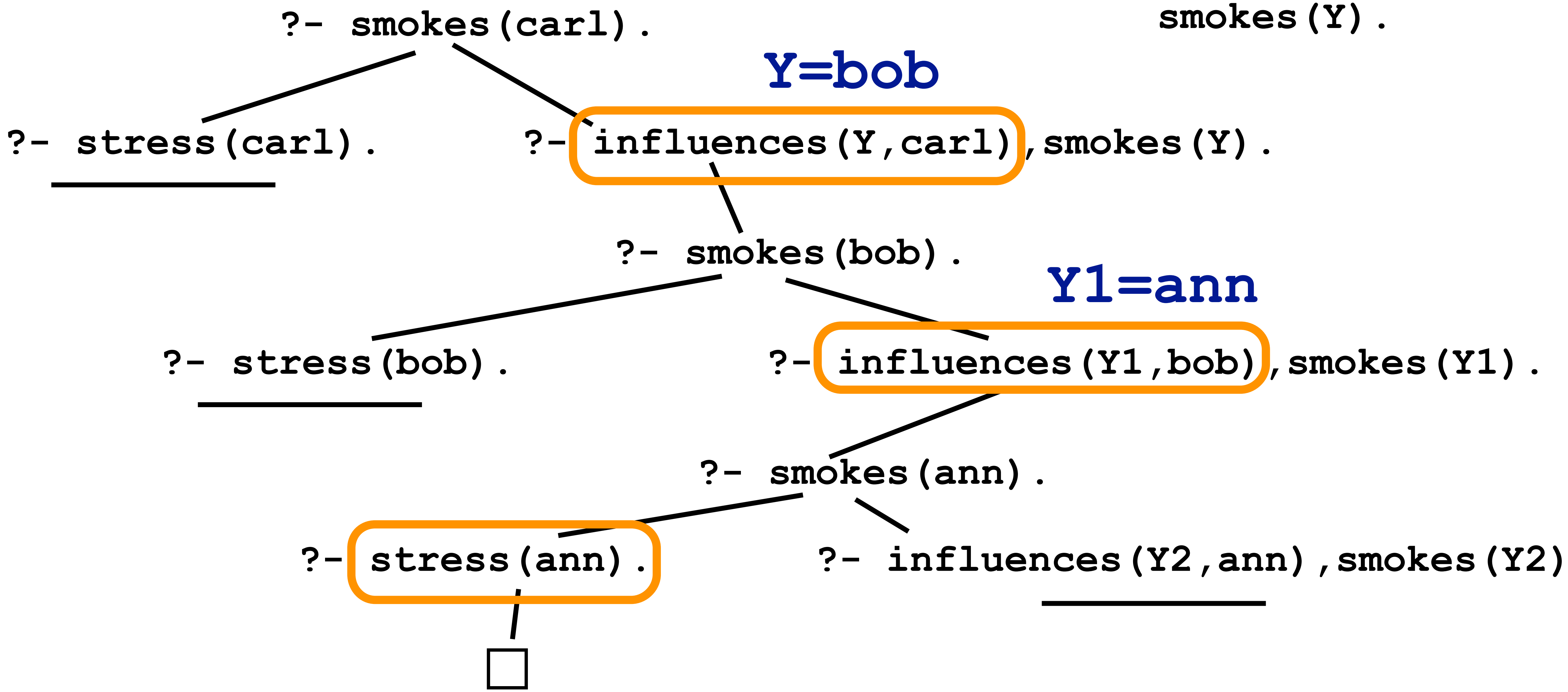
Logical Reasoning: Proofs in Prolog

```
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

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

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

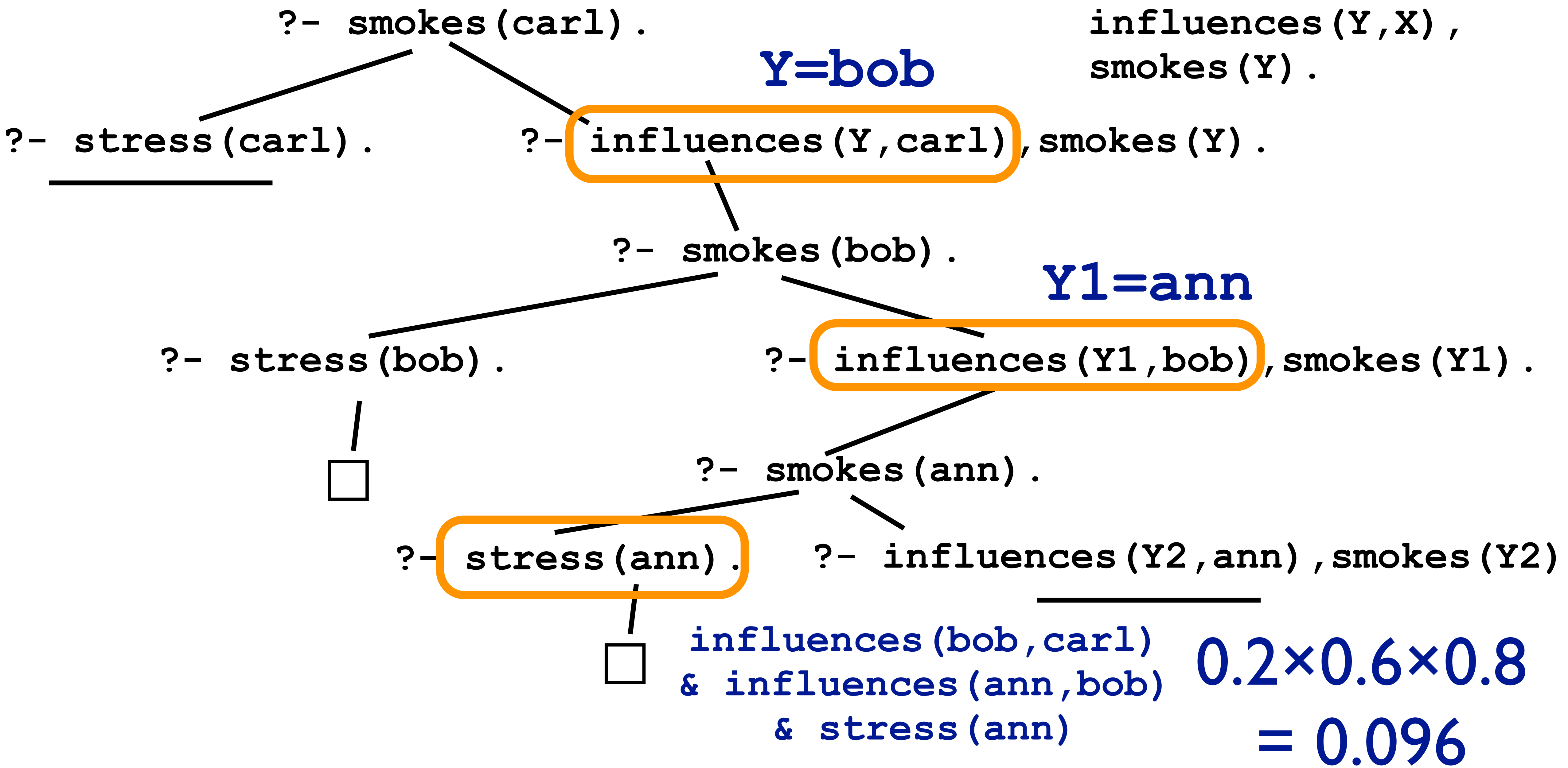
```
?- smokes(carl).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
```
Y=bob
```
?- smokes(bob).
?- stress(bob).
?- influences(Y1,bob),smokes(Y1).
```
Y1=ann
```
?- smokes(ann).
?- stress(ann).
?- influences(Y2,ann),smokes(Y2).
```
□

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

probability of proof = 0.2 × 0.6 × 0.8 = 0.096

Proofs in ProbLog

```
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).
```



Proofs in ProbLog

```
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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
```

Y=bob

```
?- smokes(bob).
?- stress(bob).
□
?- influences(Y1,bob),smokes(Y1).
```

Y1=ann

```
?- smokes(ann).
?- stress(ann).
□
?- influences(Y2,ann),smokes(Y2).
```

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

$0.2 \times 0.6 \times 0.8$

$= 0.096$

Proofs in ProbLog

```
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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
```

Y=bob

```
?- smokes(bob).
```

influences(bob,carl)
& stress(bob)

```
?- stress(bob).
□
```

0.2×0.4
$= 0.08$

```
?- influences(Y1,bob),smokes(Y1).
```

Y1=ann

```
?- smokes(ann).
?- stress(ann).
□
?- influences(Y2,ann),smokes(Y2).
```

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

$0.2 \times 0.6 \times 0.8$
$= 0.096$

Proofs in ProbLog

```
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).
?- stress(carl).
?- influences(Y,carl),smokes(Y).
```

Y=bob

```
?- smokes(bob).
?- stress(bob).
?- influences(Y1,bob),smokes(Y1).
```

Y1=ann

```
?- smokes(ann).
?- stress(ann).
?- influences(Y2,ann),smokes(Y2).
```

influences(bob,carl)
& stress(bob)

0.2×0.4
$= 0.08$

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

$0.2\times0.6\times0.8$
$= 0.096$

proofs overlap!
cannot sum probabilities
(disjoint-sum-problem)

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)
```

$\sum = 0.1376$

sum of proof probabilities: 0.096+0.08 = 0.1760

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

```
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)
...
```

0.0576
0.0384
0.0256
0.0096
0.0064

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

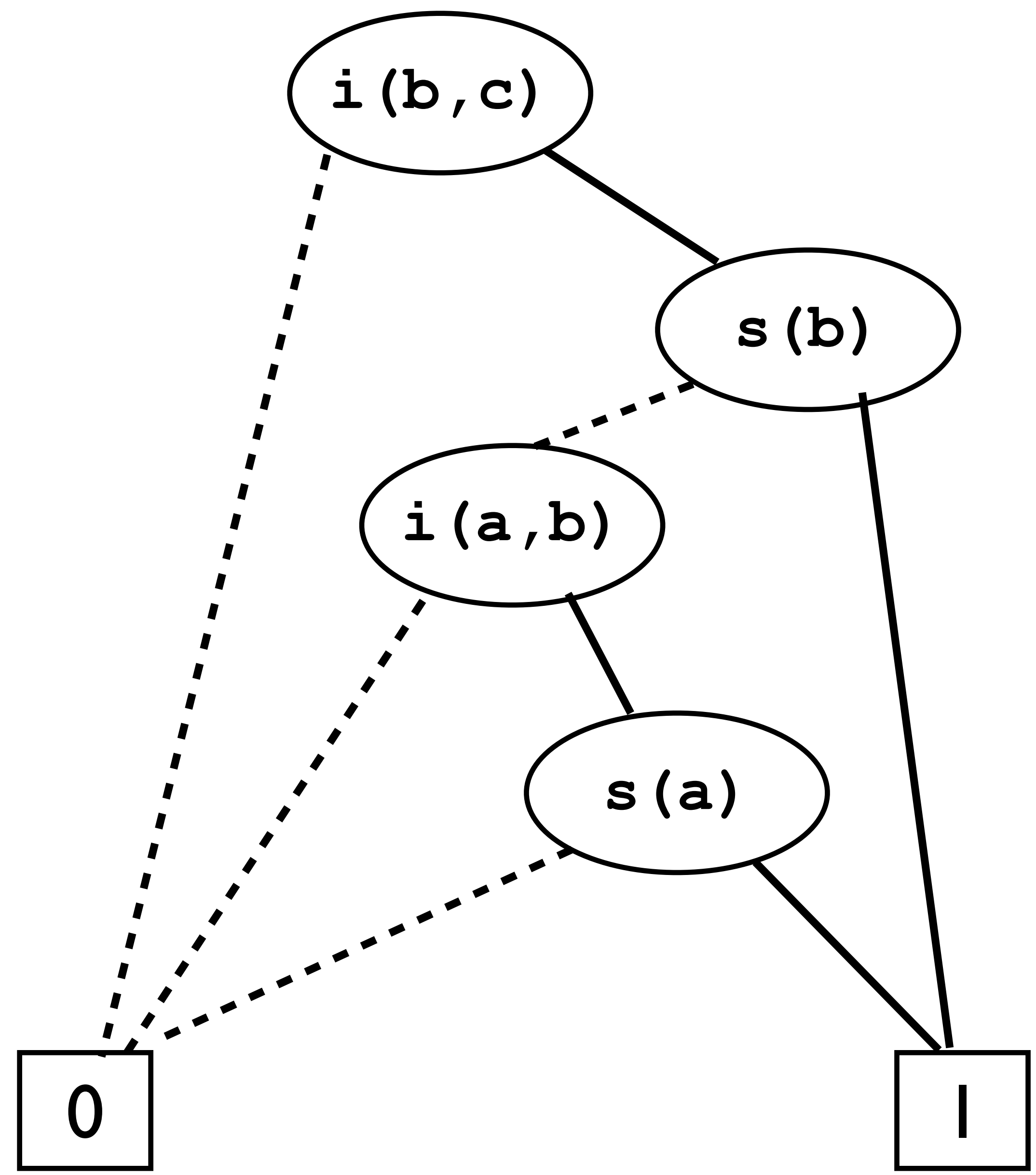
$\sum = 0.1376$

sum of proof probabilities: 0.096+0.08 = 0.1760

Binary Decision Diagrams

[Bryant 86]

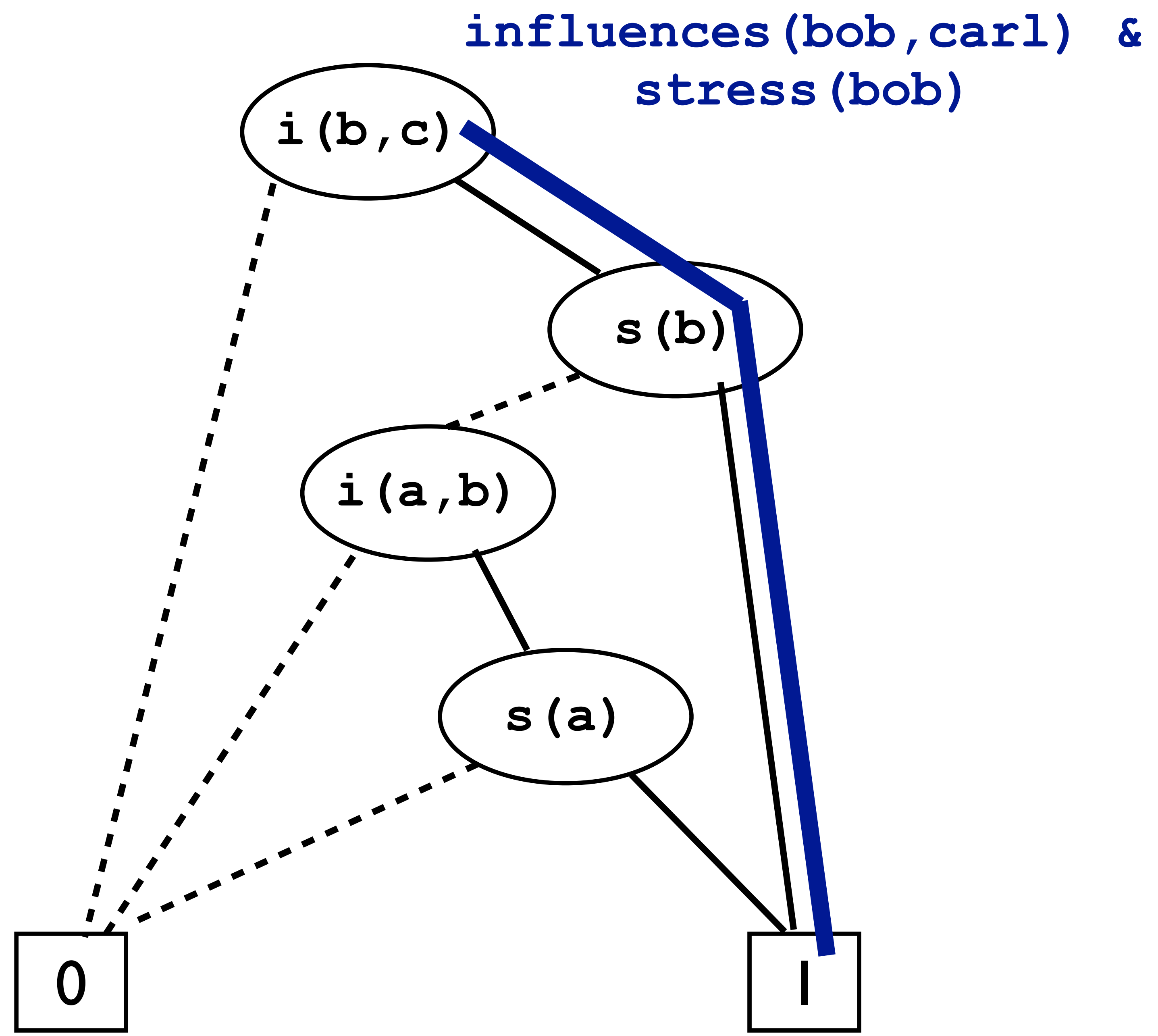
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

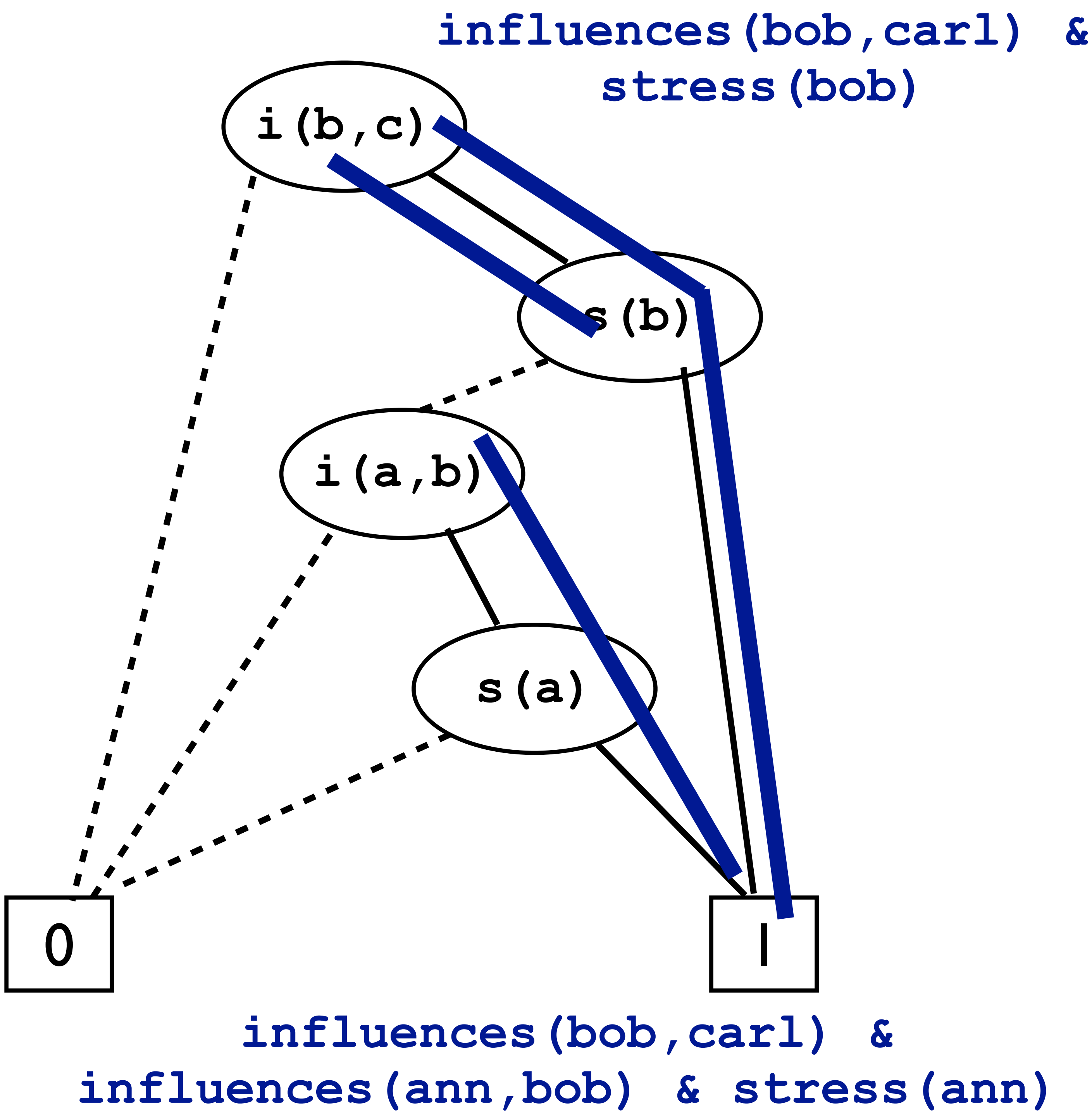
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

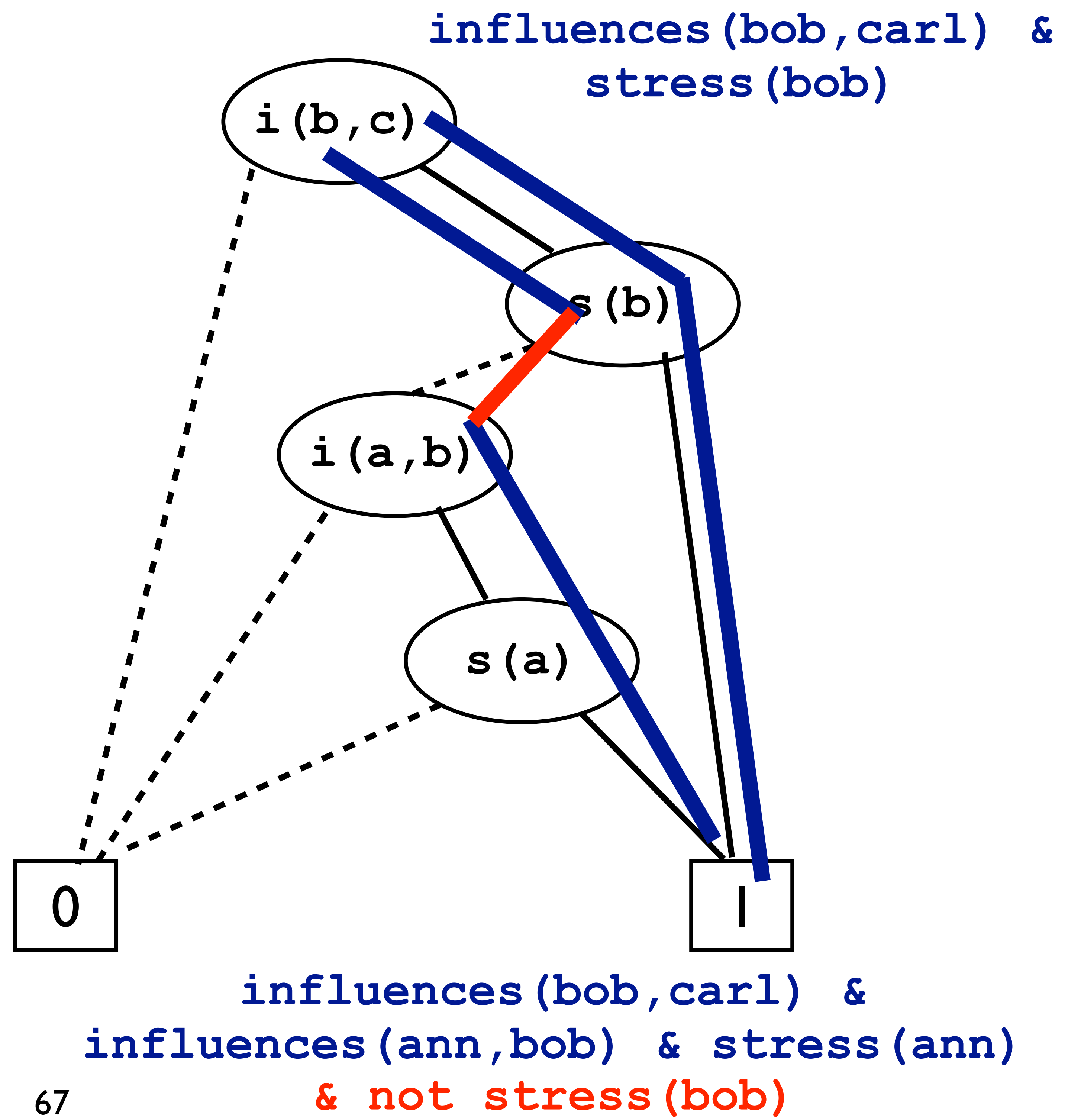
- compact graphical representation of Boolean formula
- automatically disjoins proofs
- popular in many branches of CS



Binary Decision Diagrams

[Bryant 86]

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



Binary Decision Diagrams

[Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
  → efficient probability computation
- other representations exist (SDDs, d-DNNFs)
- knowledge compilation is state of the art for probabilistic inference (Darwiche et al.)

Binary Decision Diagrams
[Bryant 86]

$X \vee Y \vee Z$

Binary Decision Diagrams
[Bryant 86]

$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

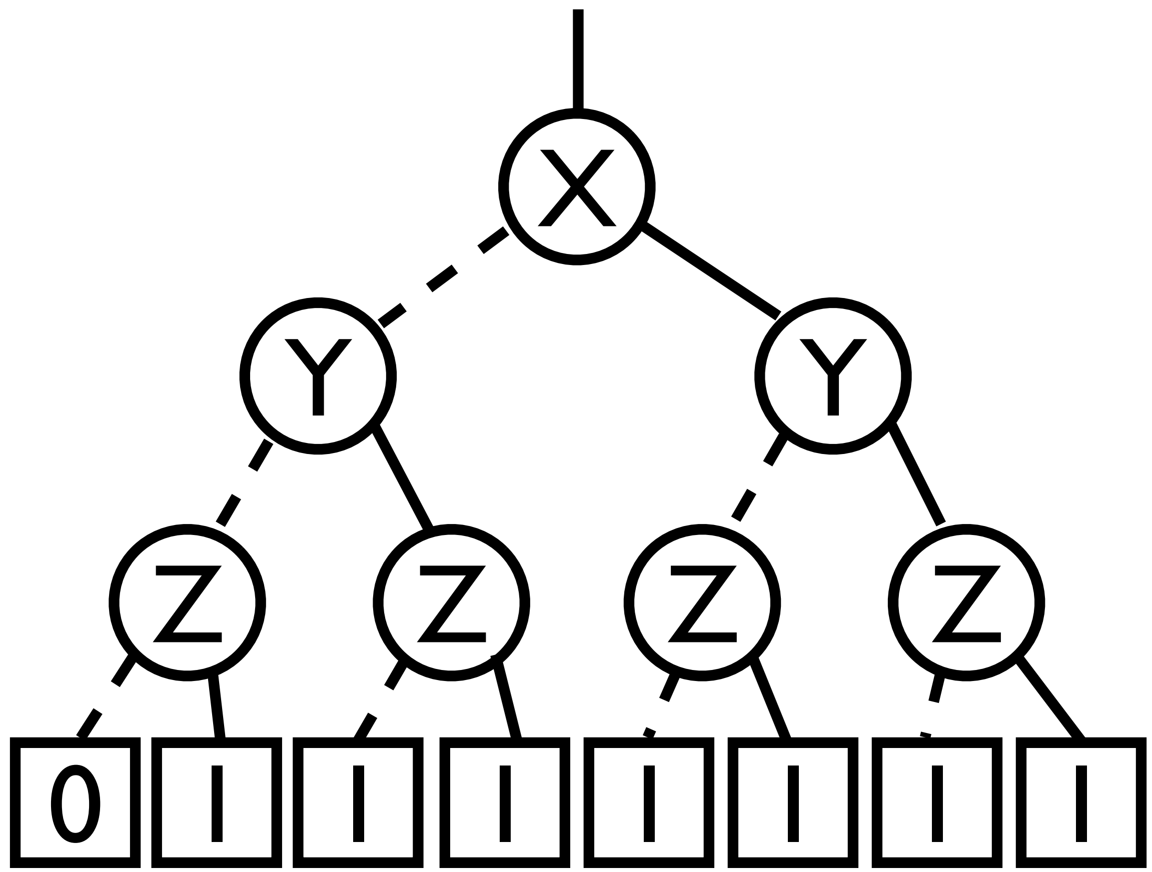
$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

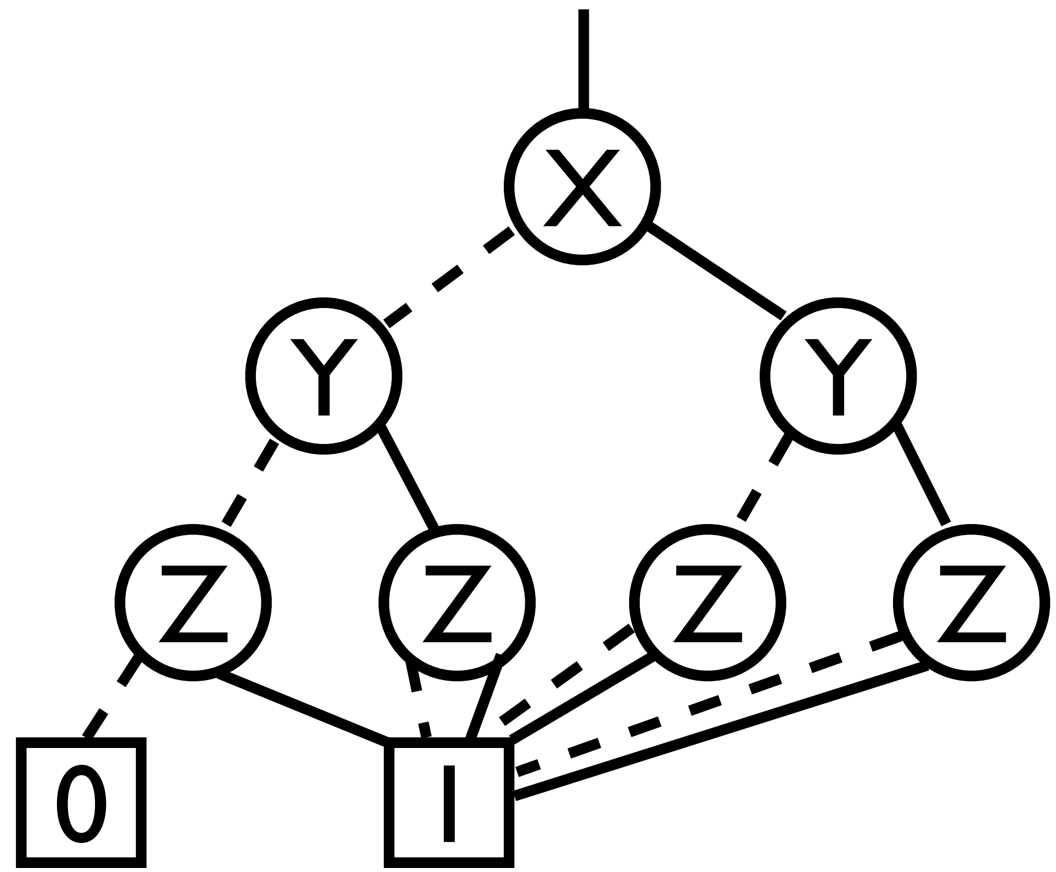
$X \vee Y \vee Z$



X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



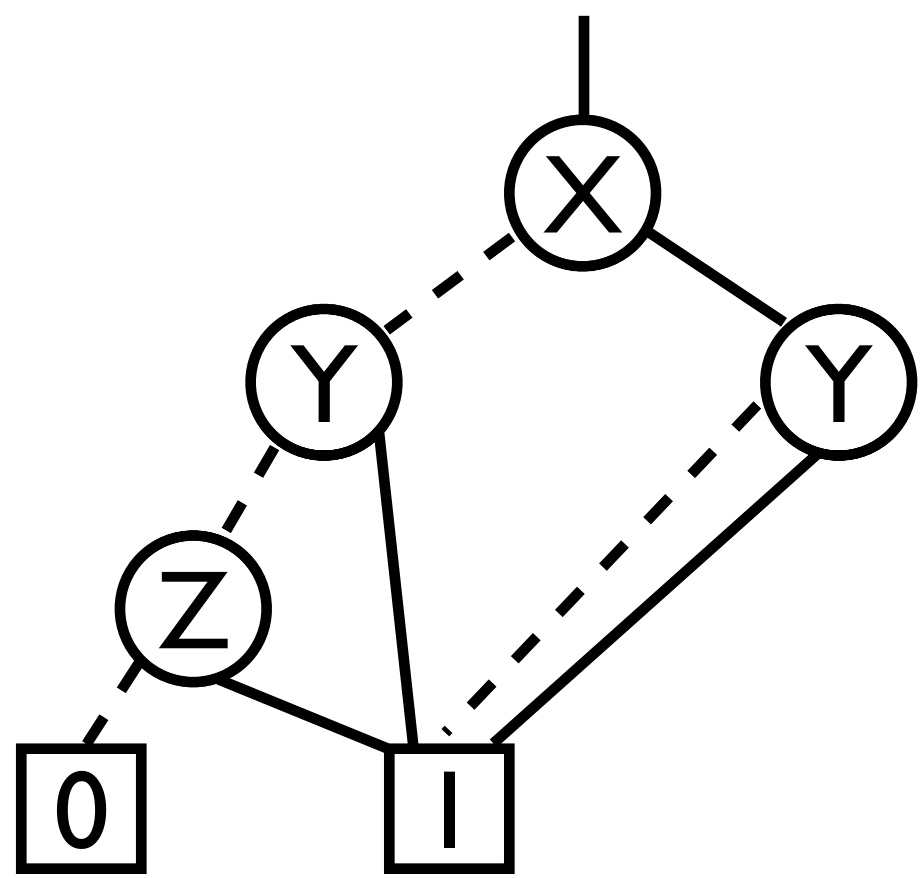
$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

<u>B</u>inary <u>D</u>ecision <u>D</u>iagrams

[Bryant 86]

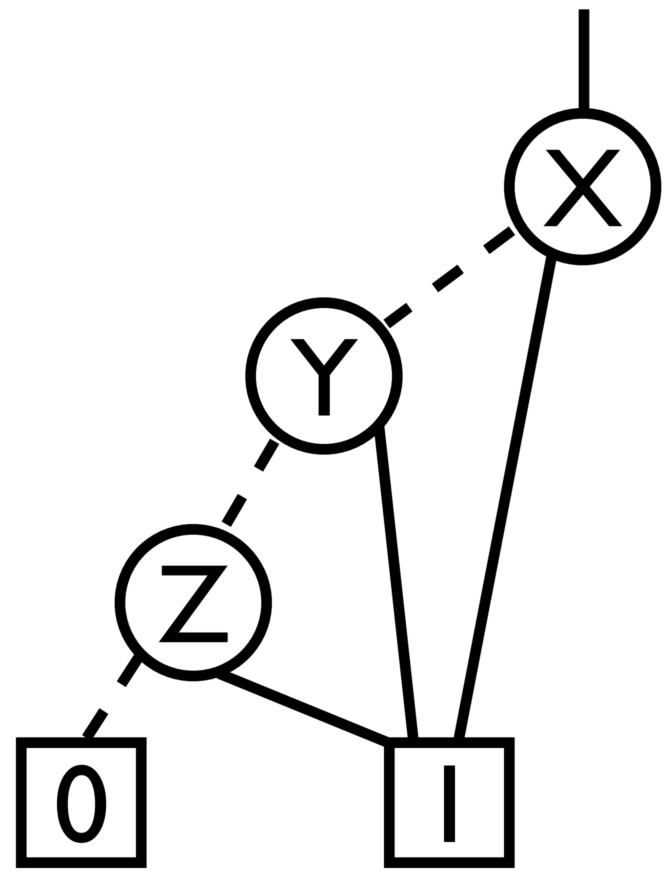
$X \vee Y \vee Z$



X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



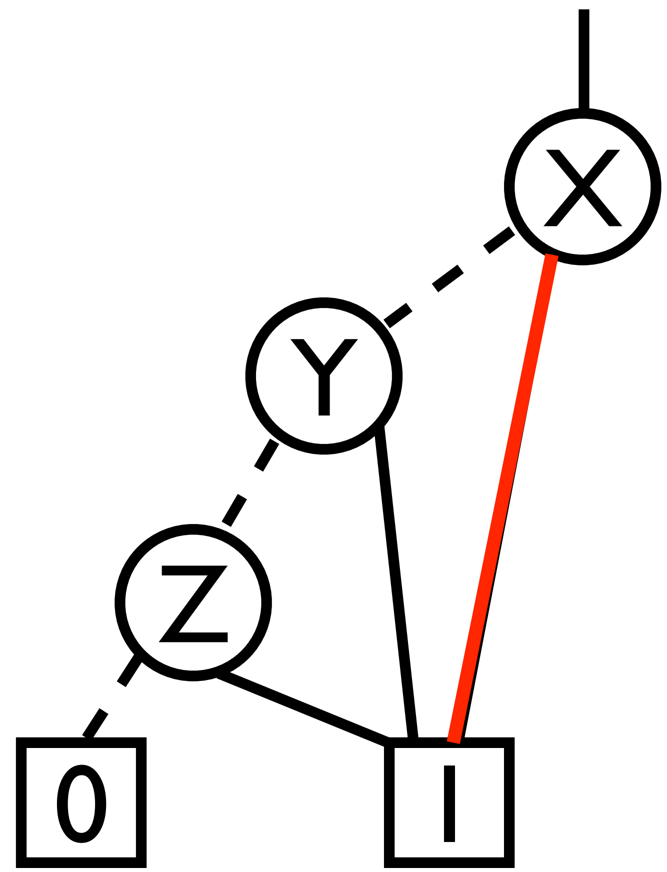
$X \vee Y \vee Z$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

$X \vee Y \vee Z$

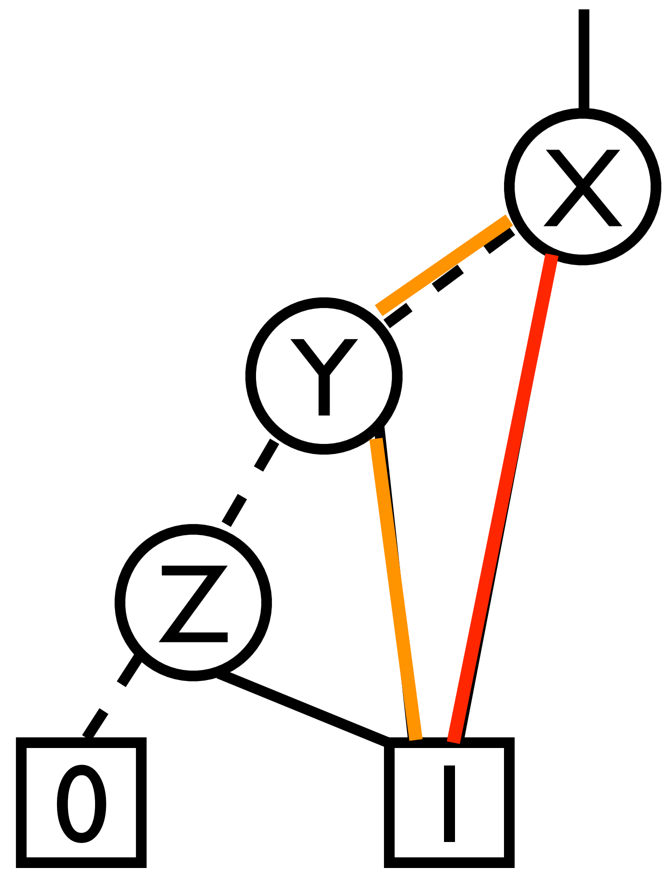


$$\underline{X} \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



$$X \lor (\neg X \land Y) \lor (\neg X \land \neg Y \land Z)$$

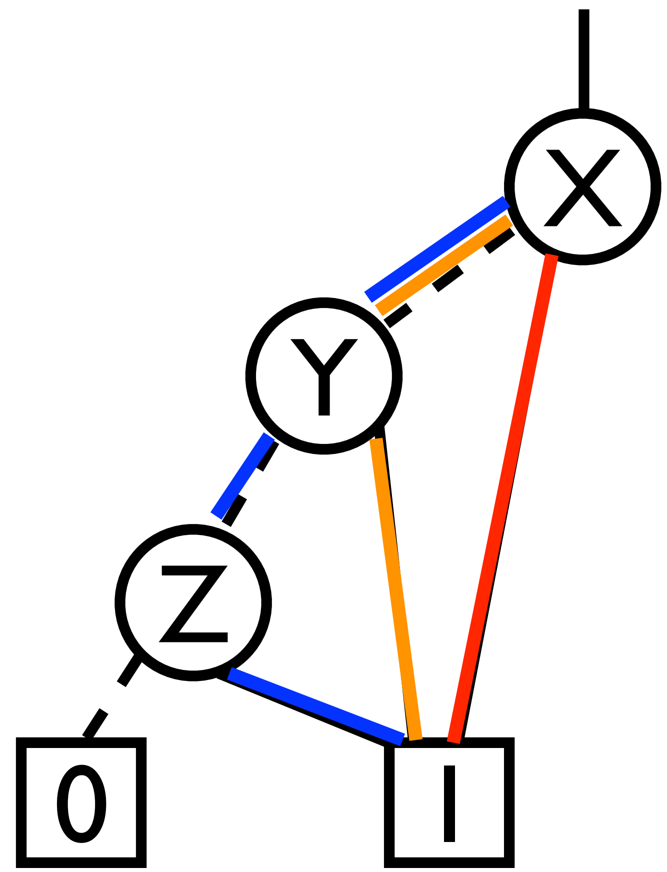
$$X \lor Y \lor Z$$

X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]

$X \vee Y \vee Z$

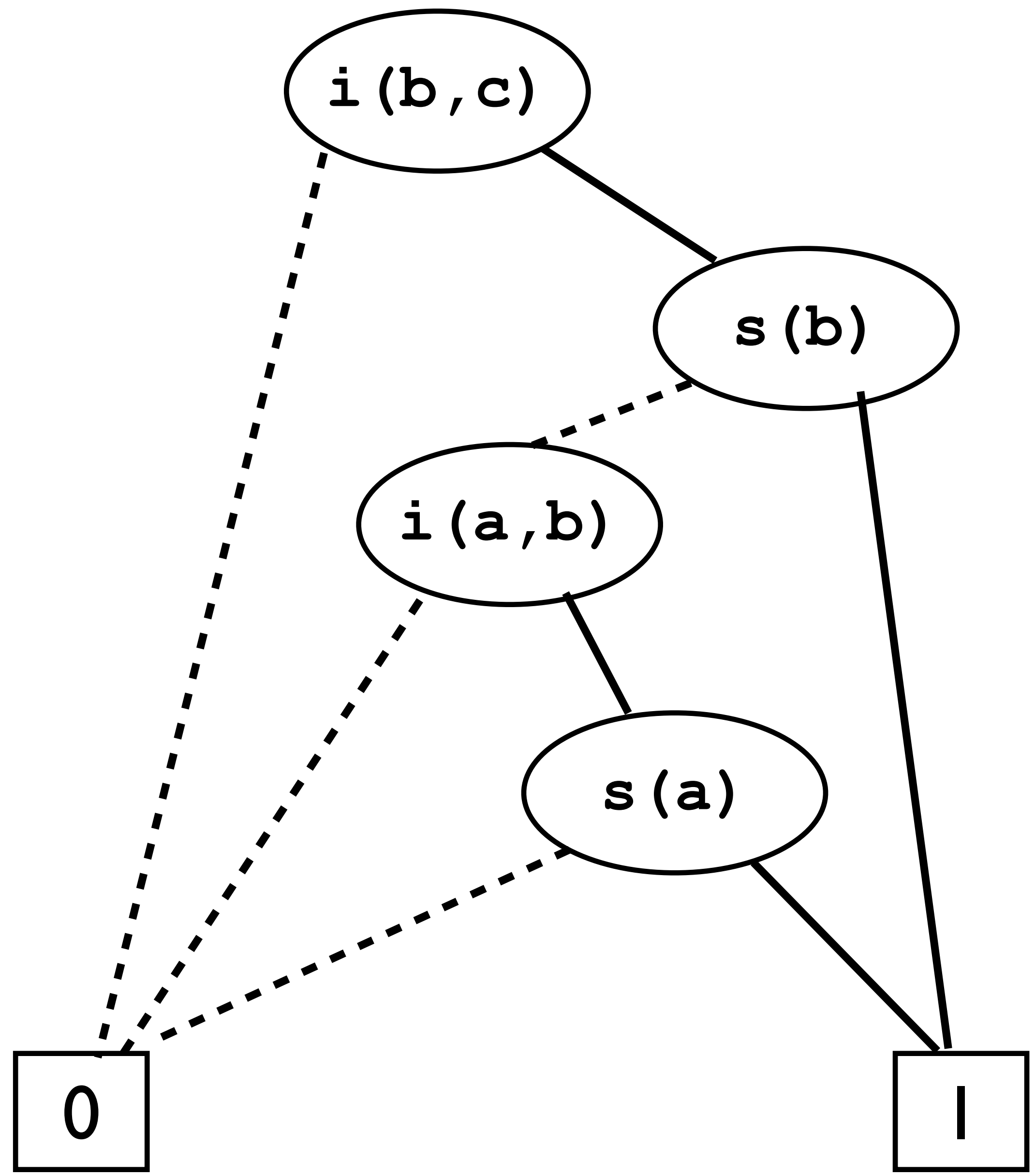


$X \vee (\neg X \wedge Y) \vee$
$(\neg X \wedge \neg Y \wedge Z)$

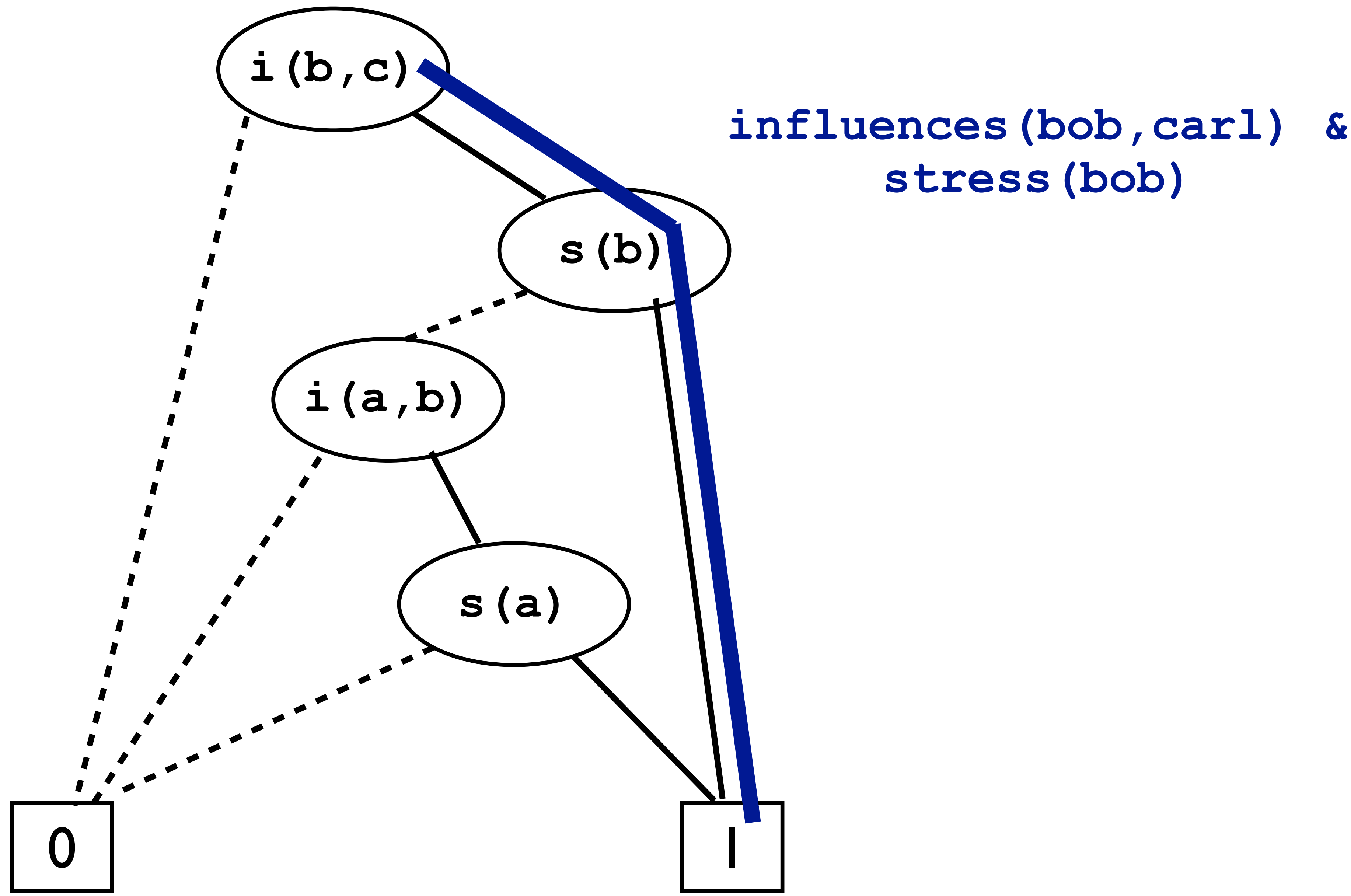
X	0	0	0	0	1	1	1	1
Y	0	0	1	1	0	0	1	1
Z	0	1	0	1	0	1	0	1

Binary Decision Diagrams

[Bryant 86]



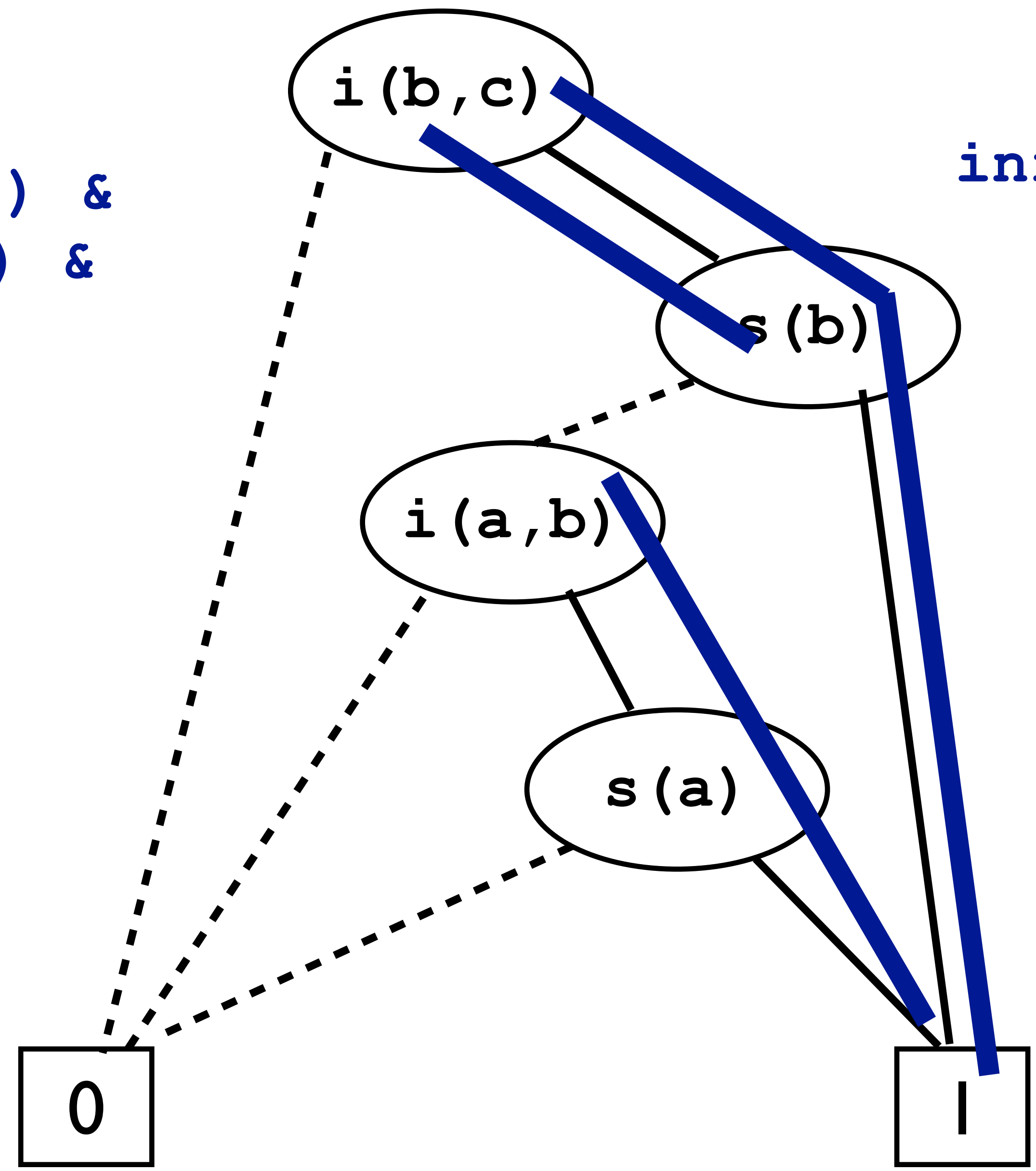
Binary Decision Diagrams
[Bryant 86]



Binary Decision Diagrams

[Bryant 86]

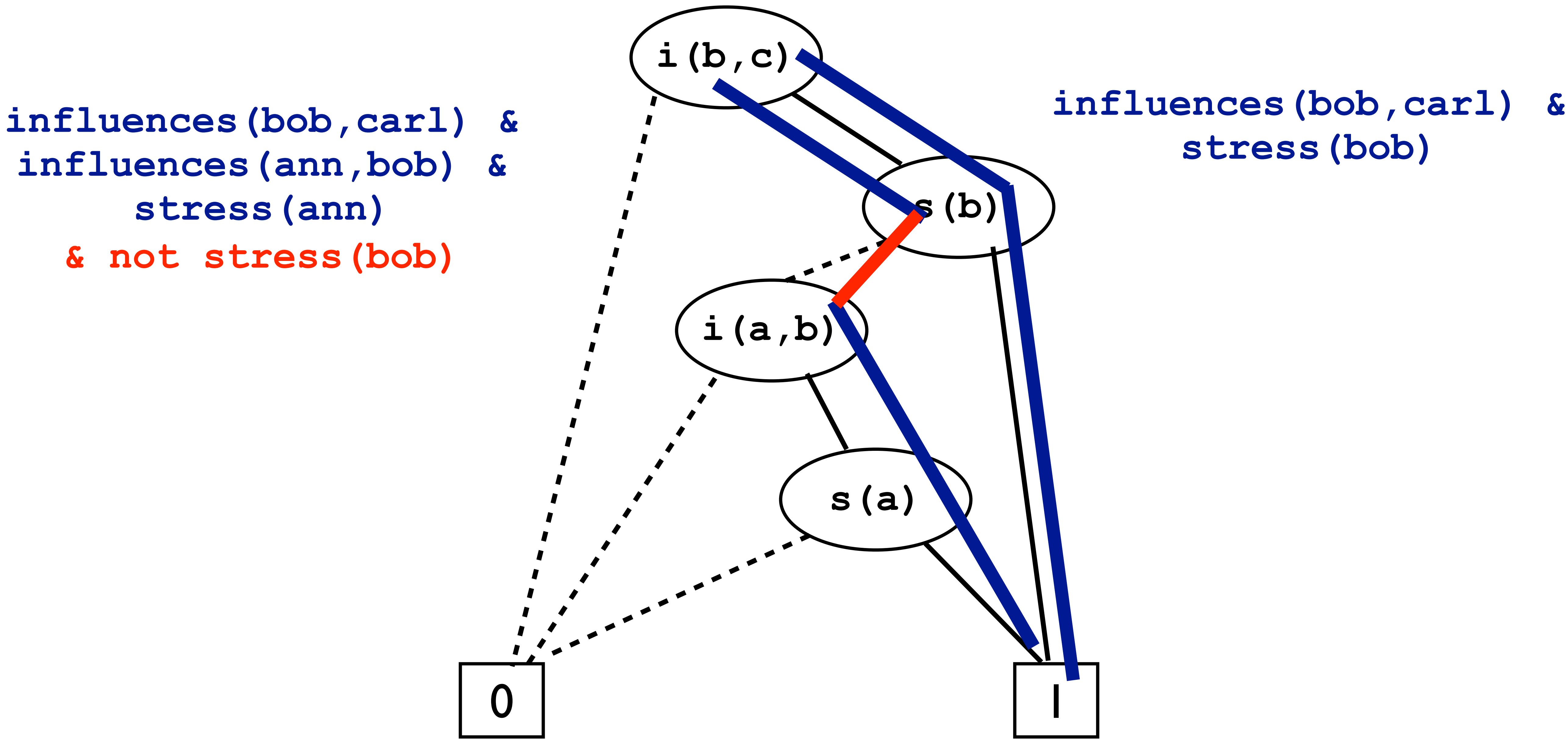
`influences(bob,carl) &`
`influences(ann,bob) &`
`stress(ann)`



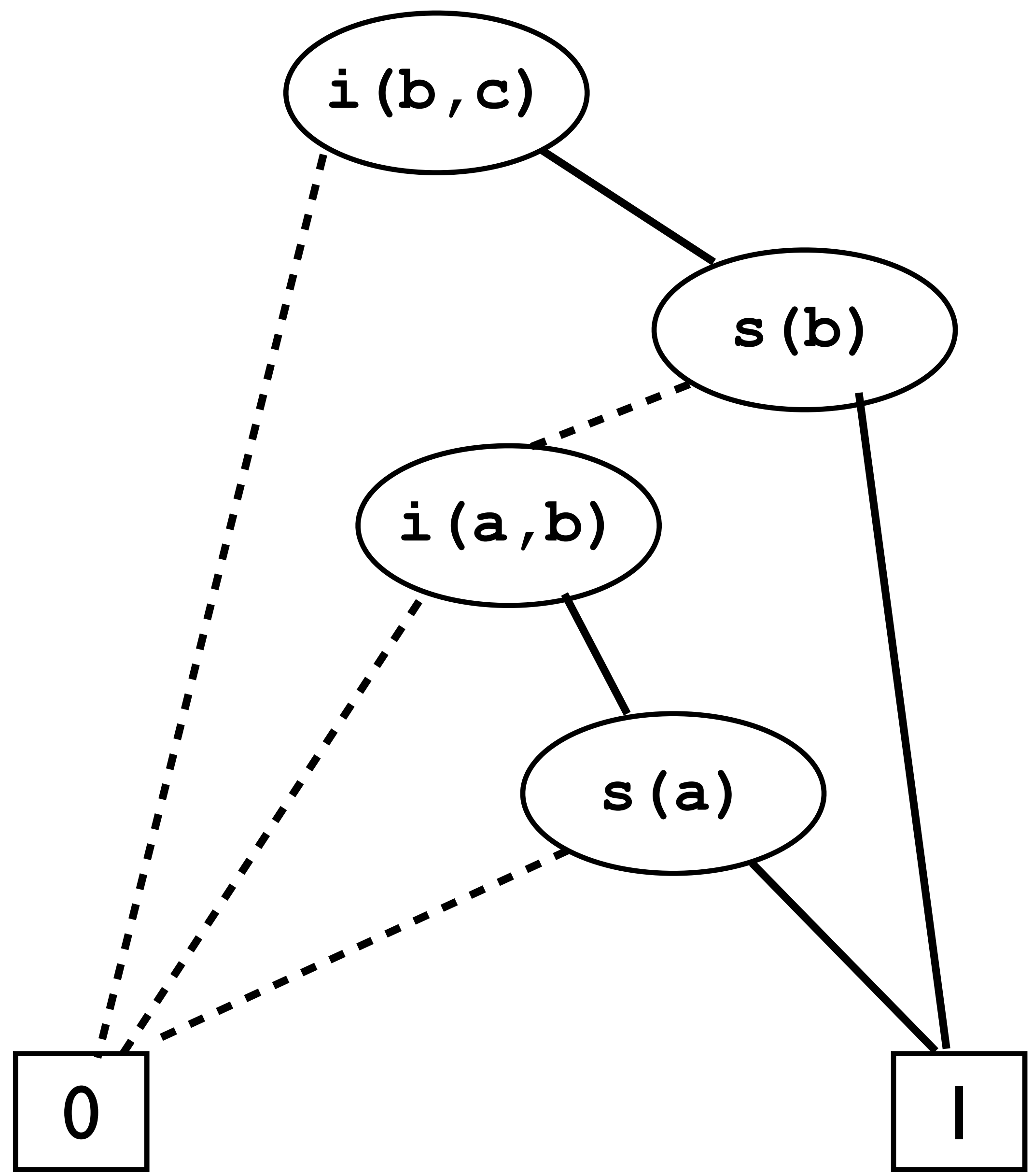
`influences(bob,carl) &`
`stress(bob)`

Binary Decision Diagrams

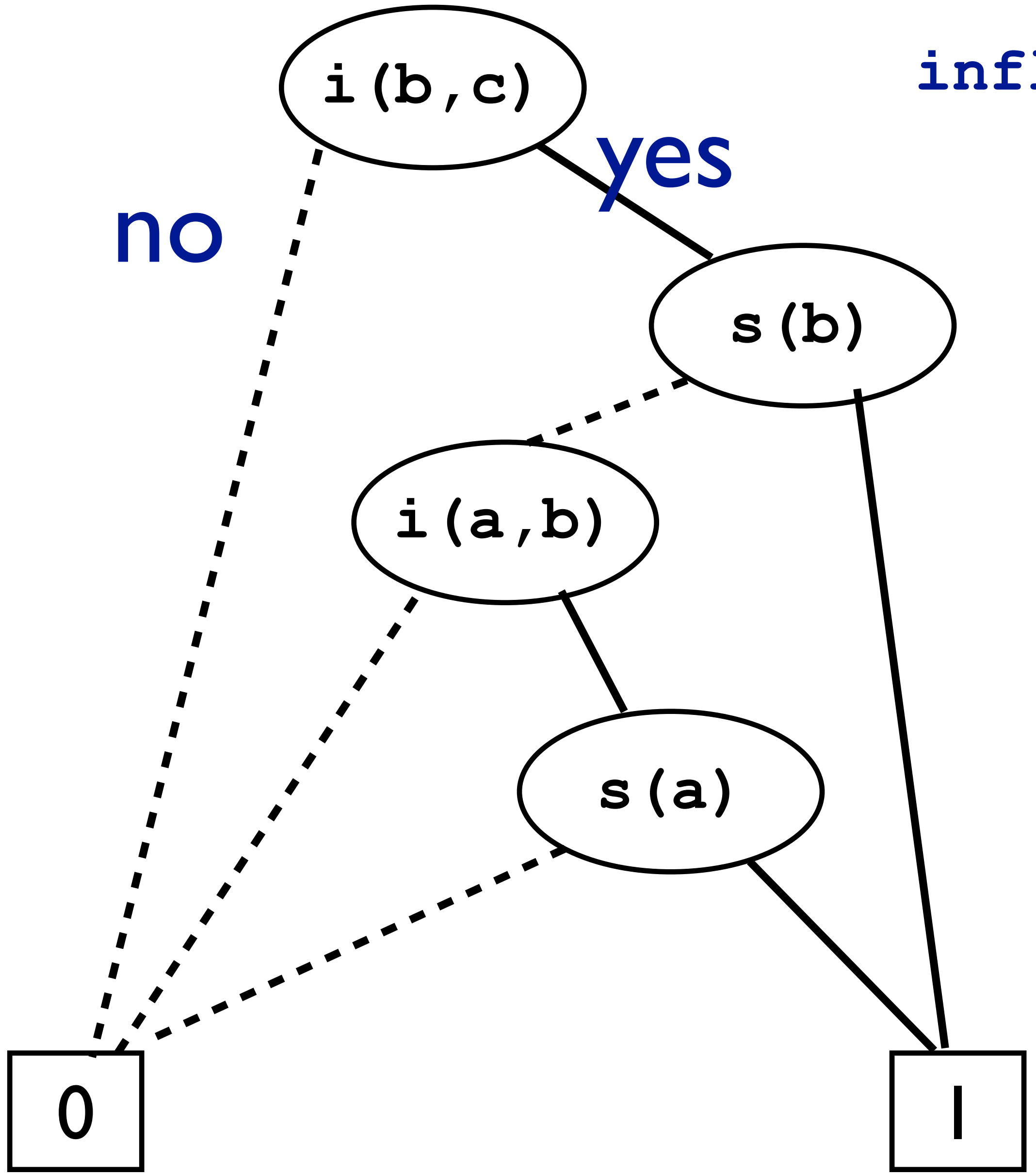
[Bryant 86]



Binary Decision Diagrams

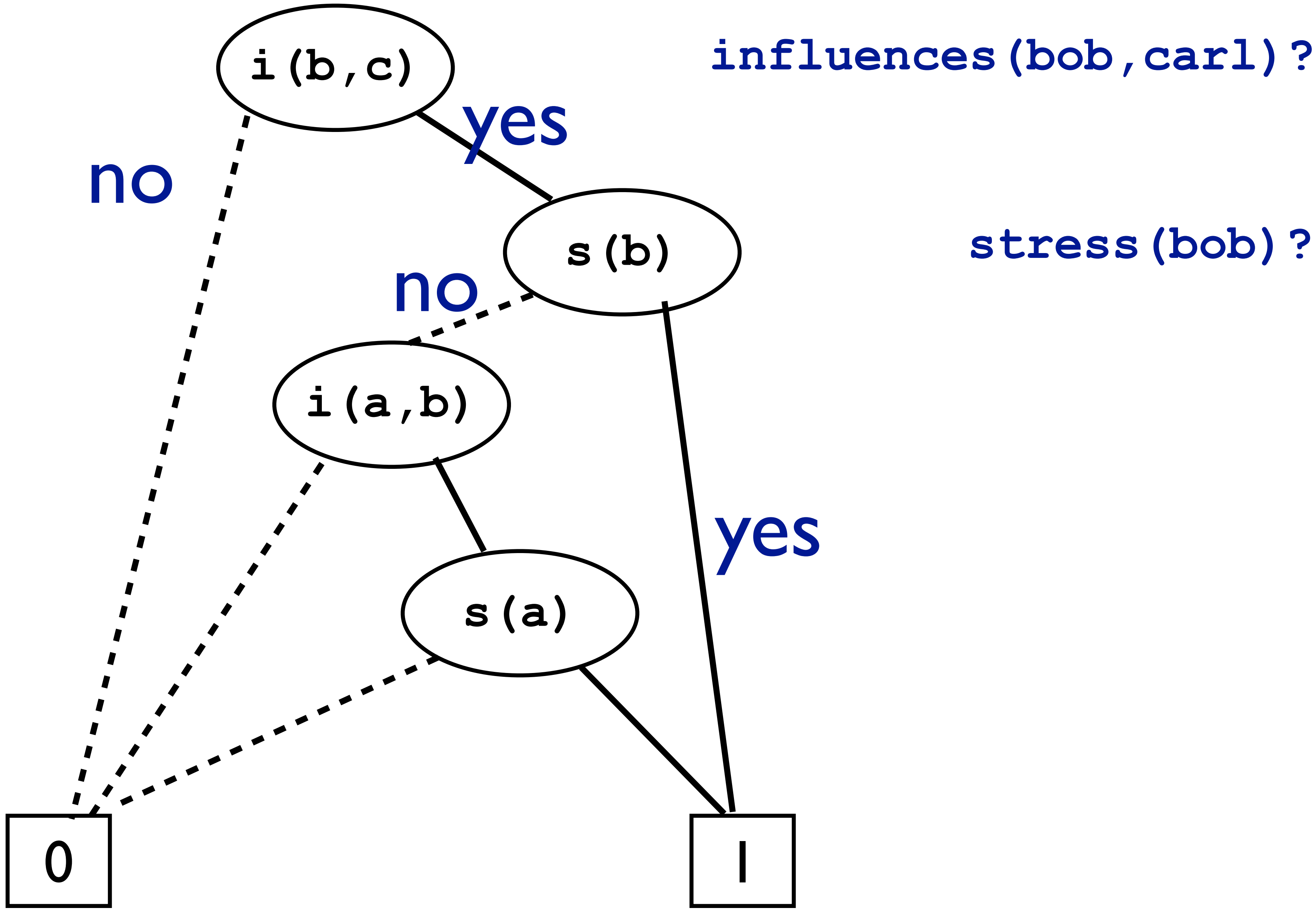


Binary Decision Diagrams

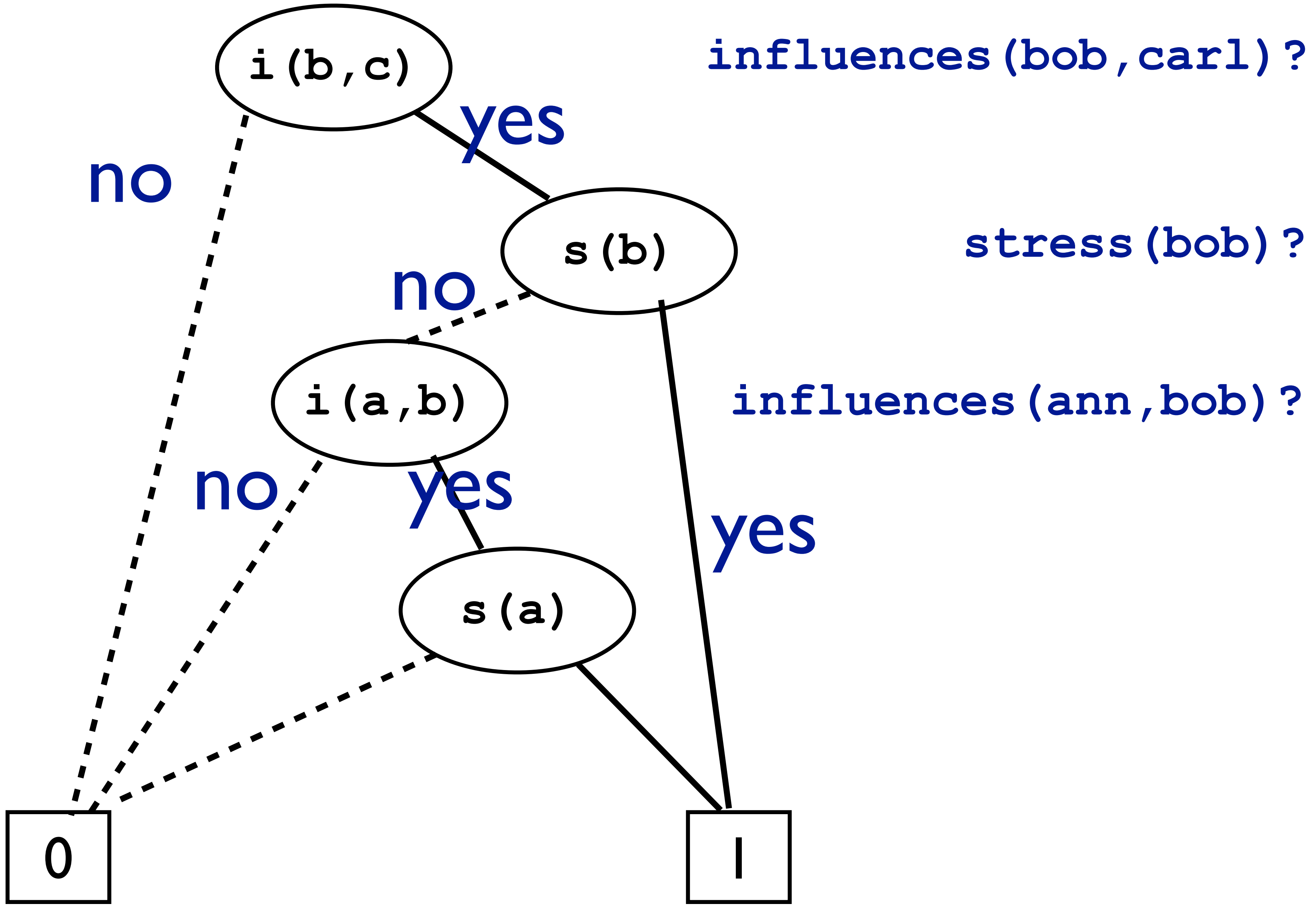


`influences(bob,carl)?`

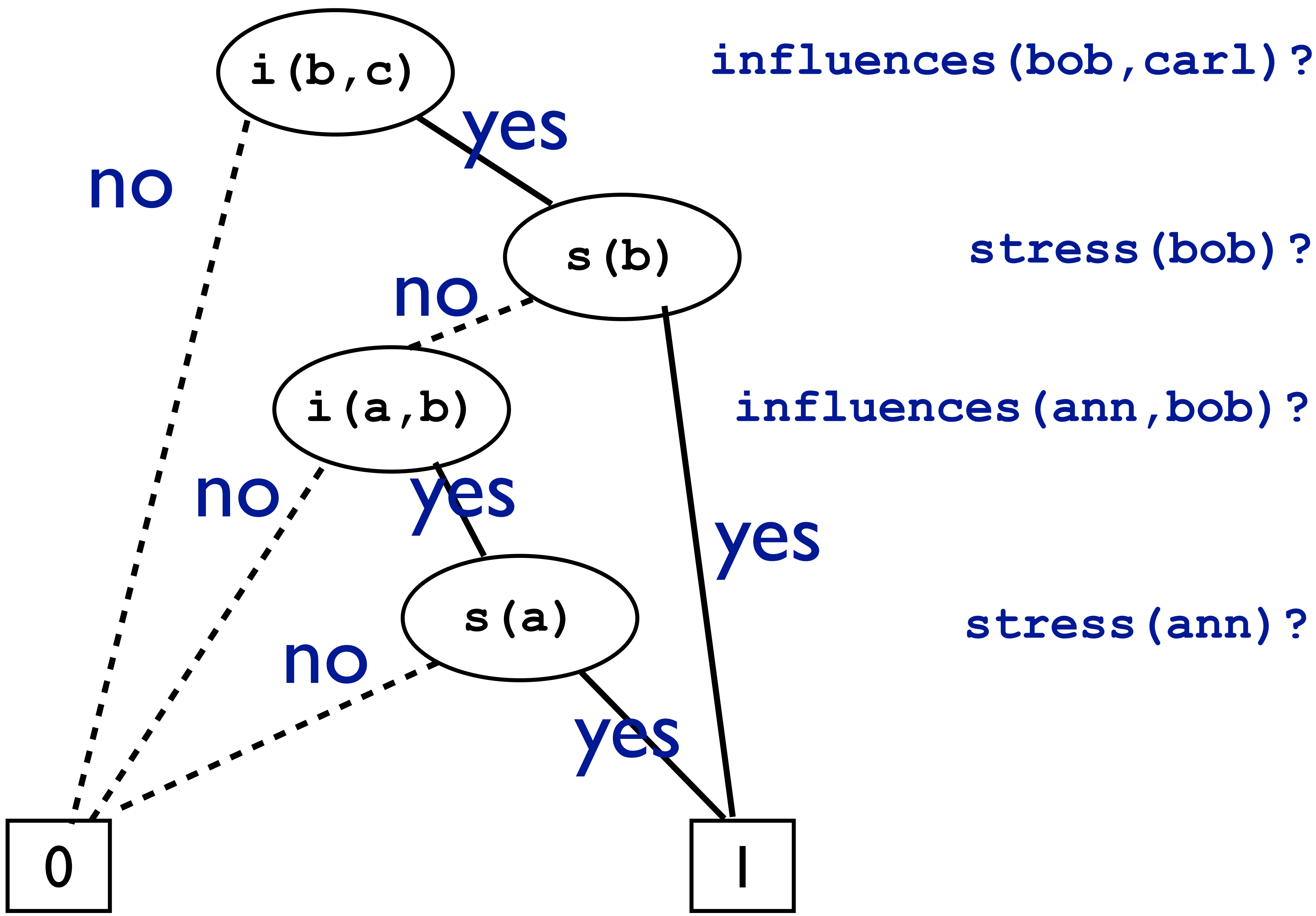
Binary Decision Diagrams



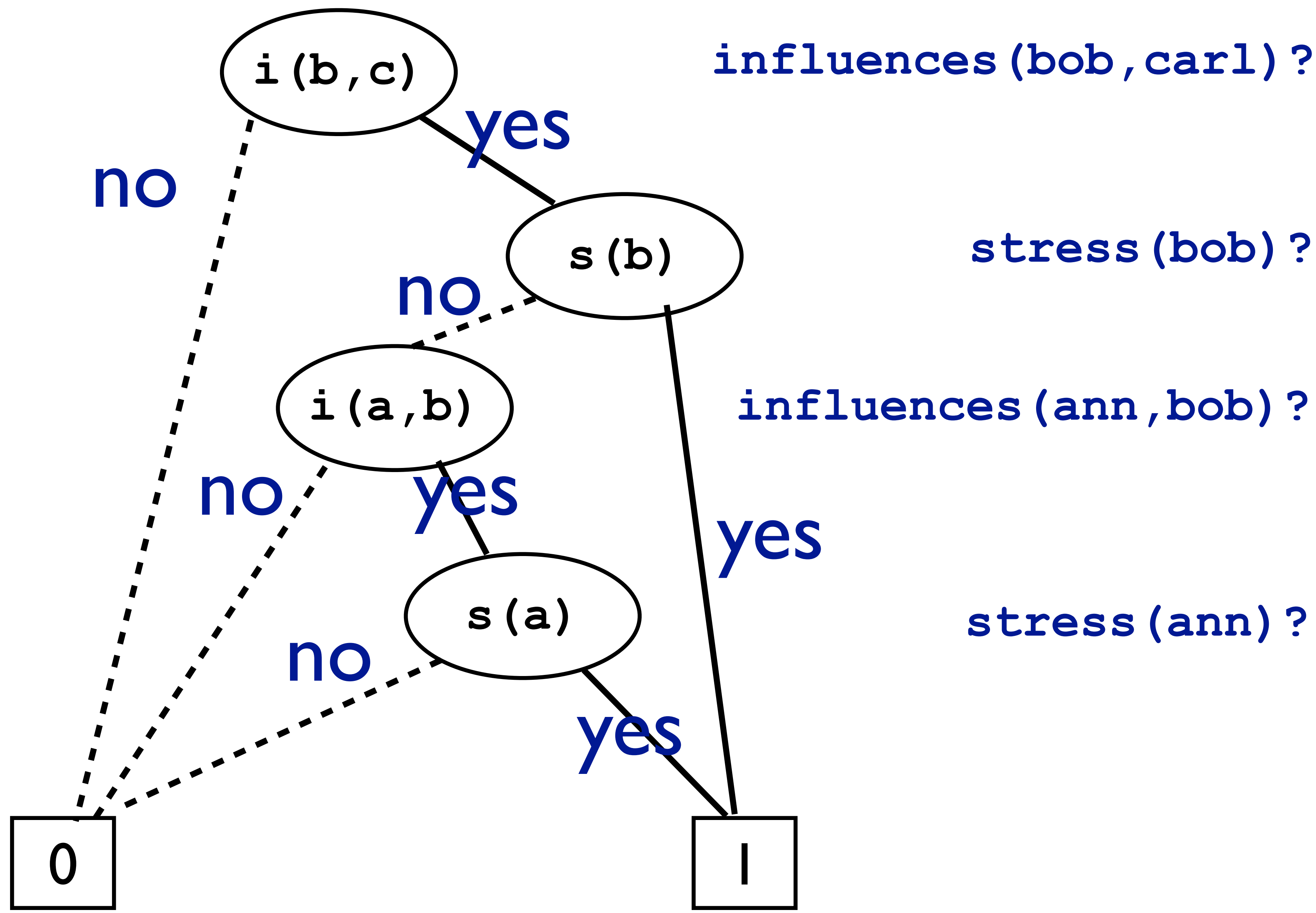
Binary Decision Diagrams



Binary Decision Diagrams

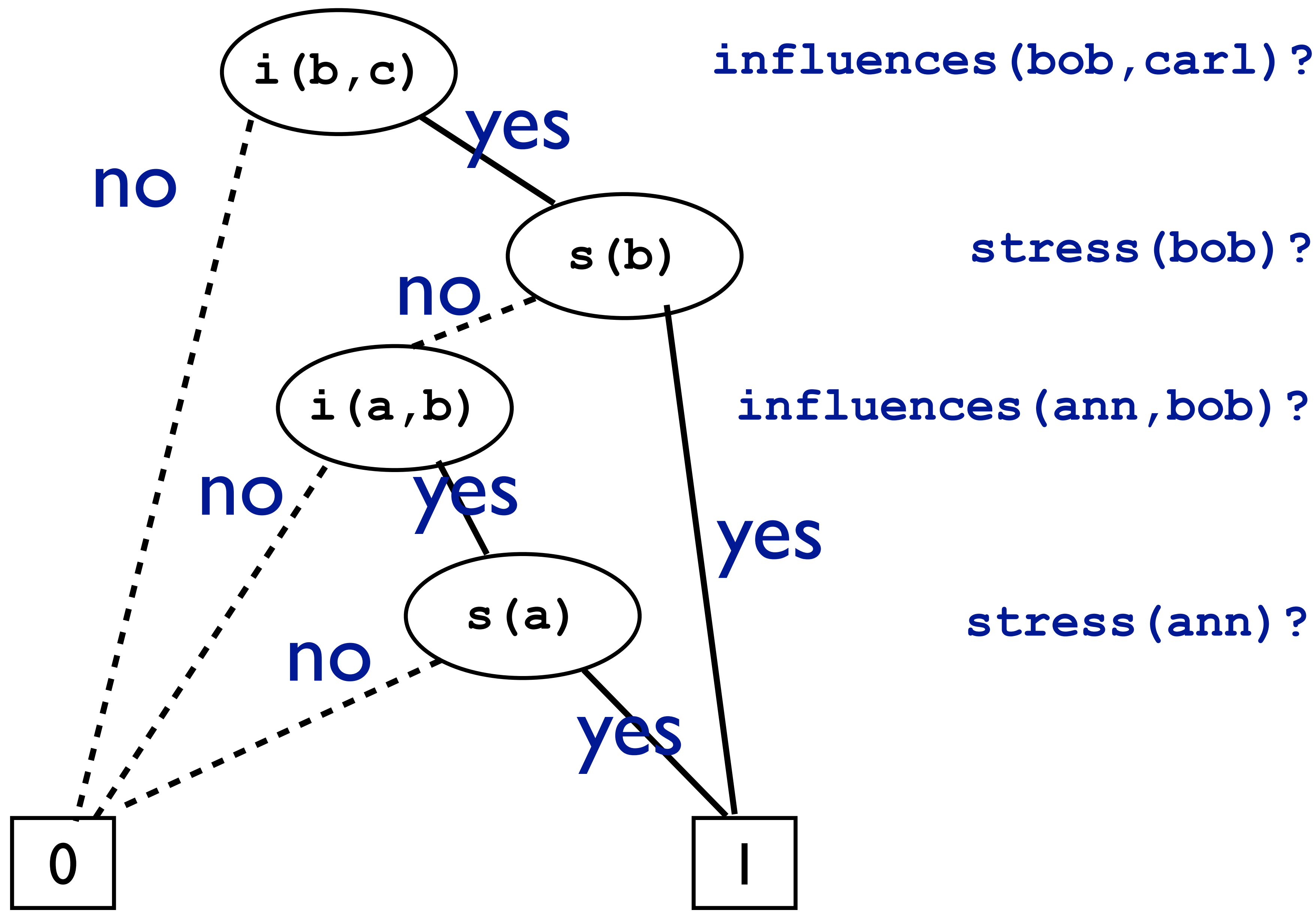


Binary Decision Diagrams



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

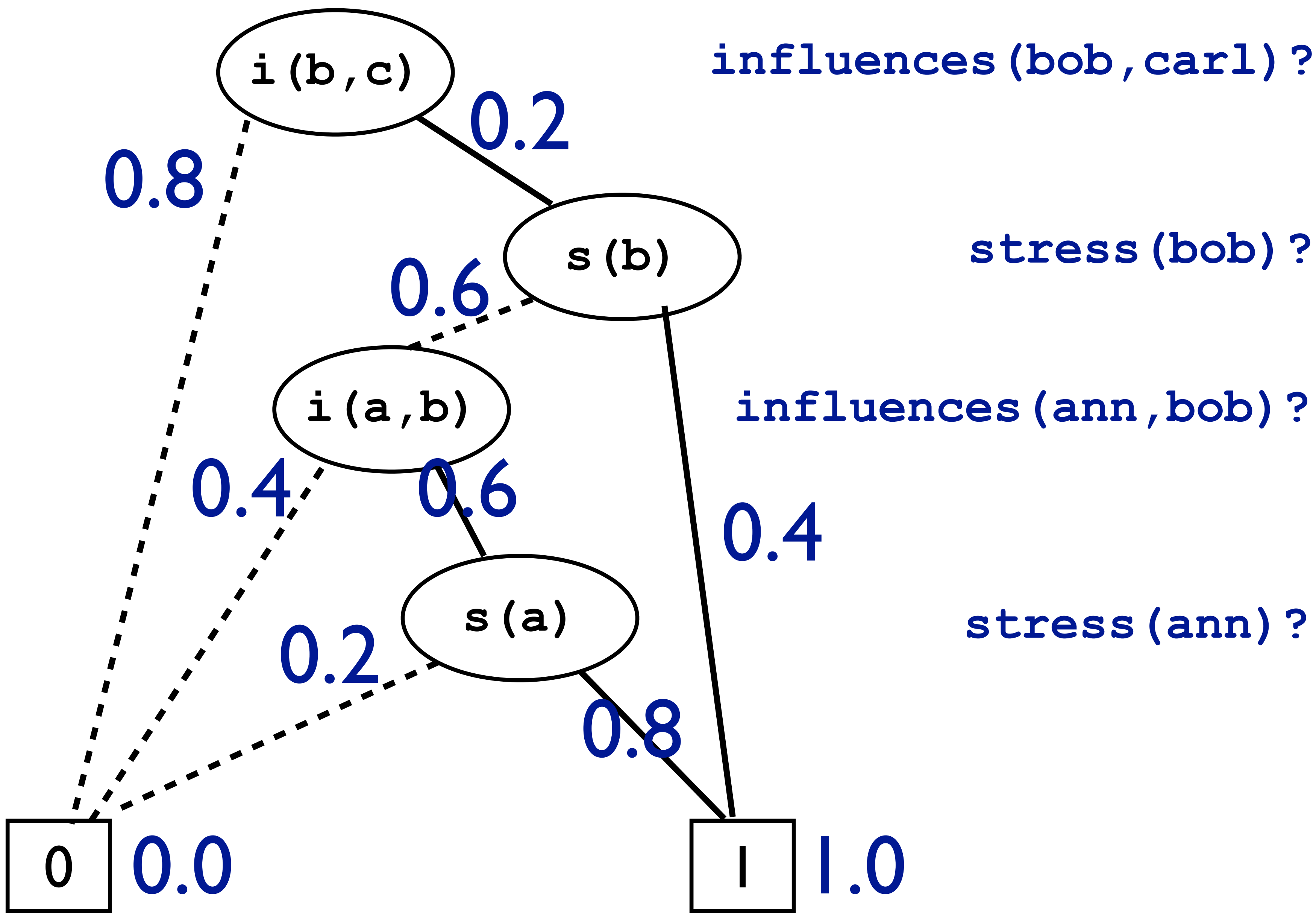
Binary Decision Diagrams



probability of `smokes(c)`?

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

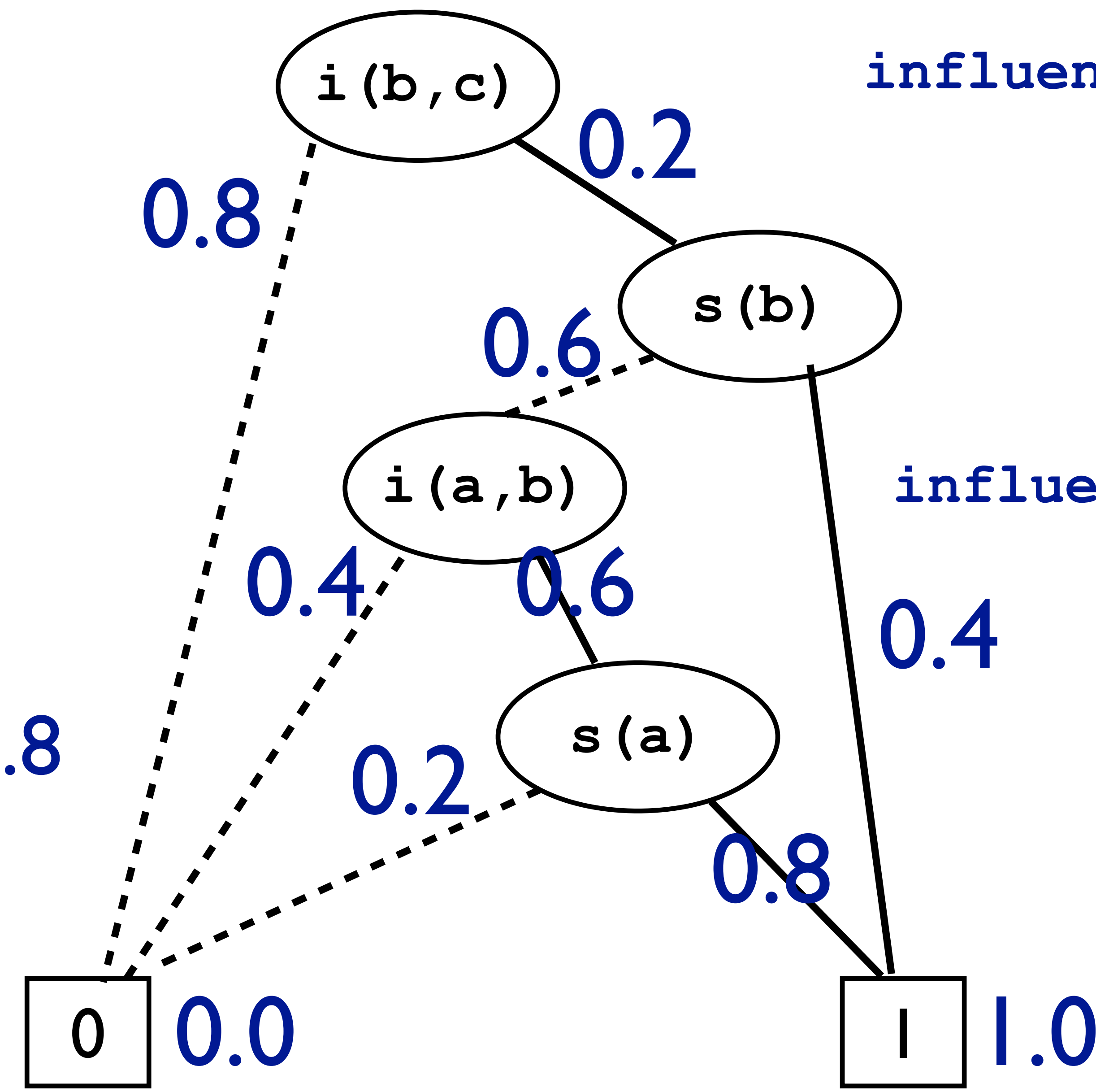
Binary Decision Diagrams



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)?

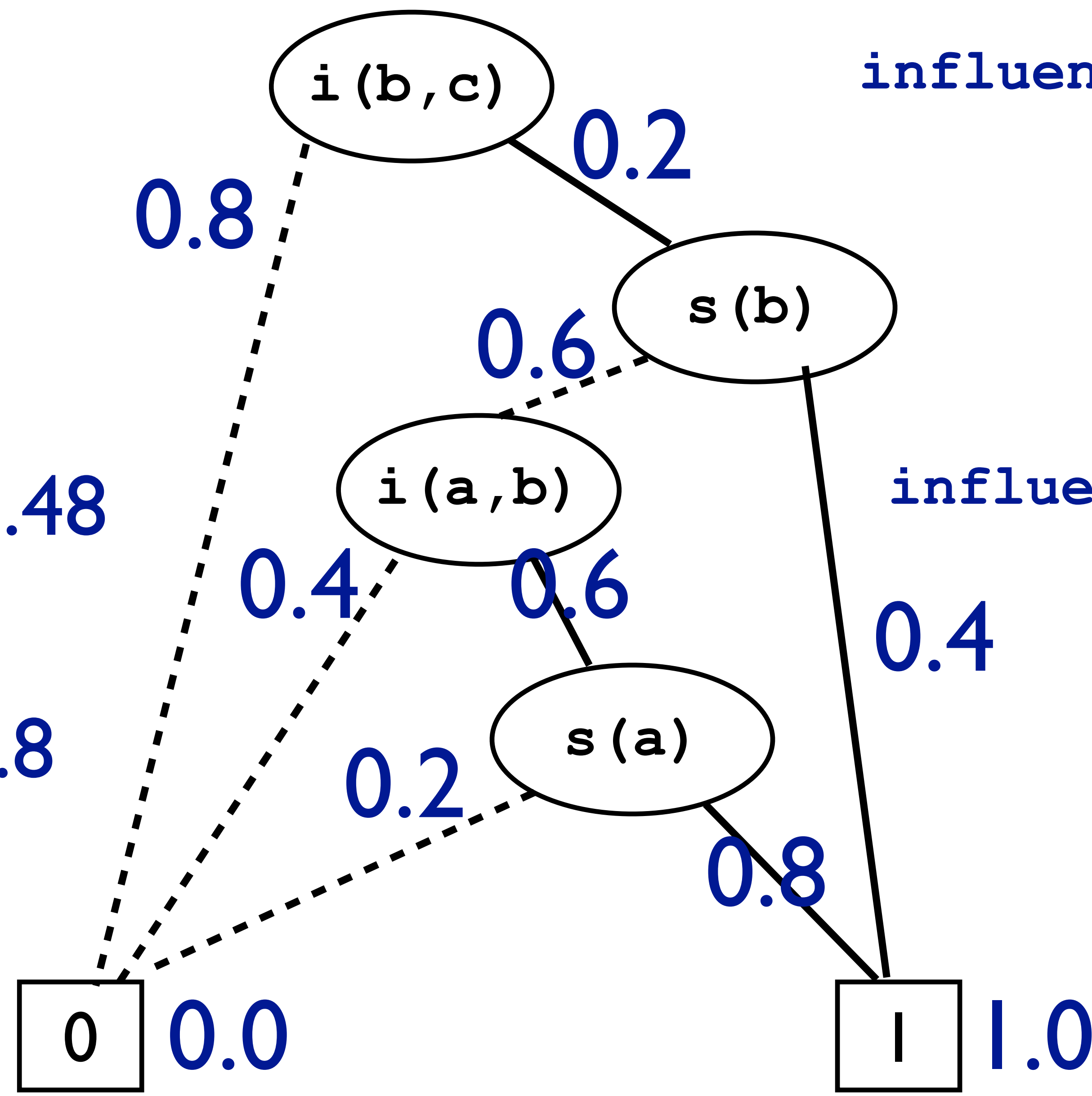
stress(ann)?

$0.2 \times 0.0 + 0.8 \times 1.0 = 0.8$

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)?`

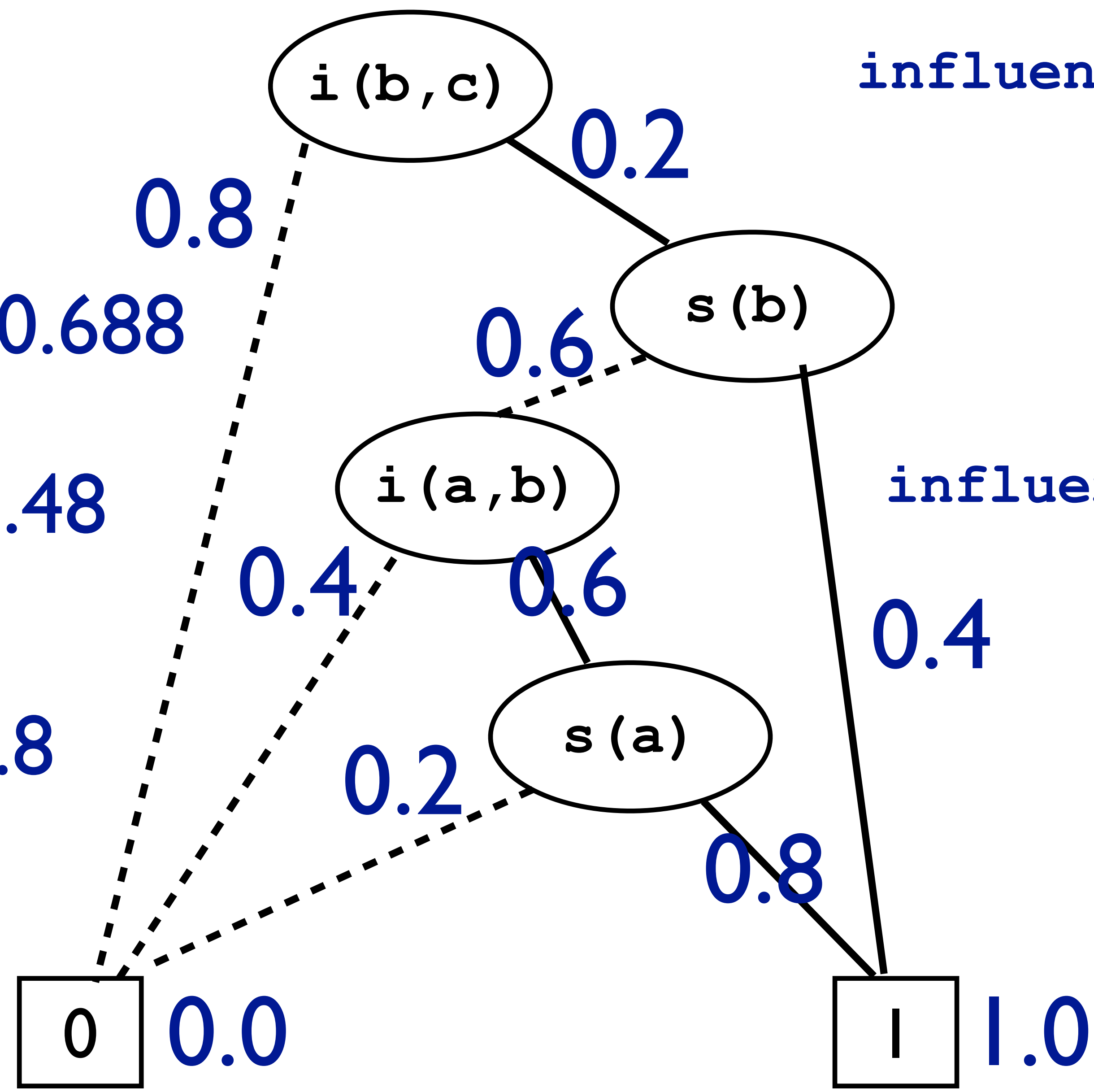
$0.4 \times 0.0 + 0.6 \times 0.8 = 0.48$

$0.2 \times 0.0 + 0.8 \times 1.0 = 0.8$

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)?

$0.6 \times 0.48 + 0.4 \times 1.0 = 0.688$

stress(bob)?

$0.4 \times 0.0 + 0.6 \times 0.8 = 0.48$

influences(ann,bob)?

$0.2 \times 0.0 + 0.8 \times 1.0 = 0.8$

stress(ann)?

probability of smokes(c)?

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

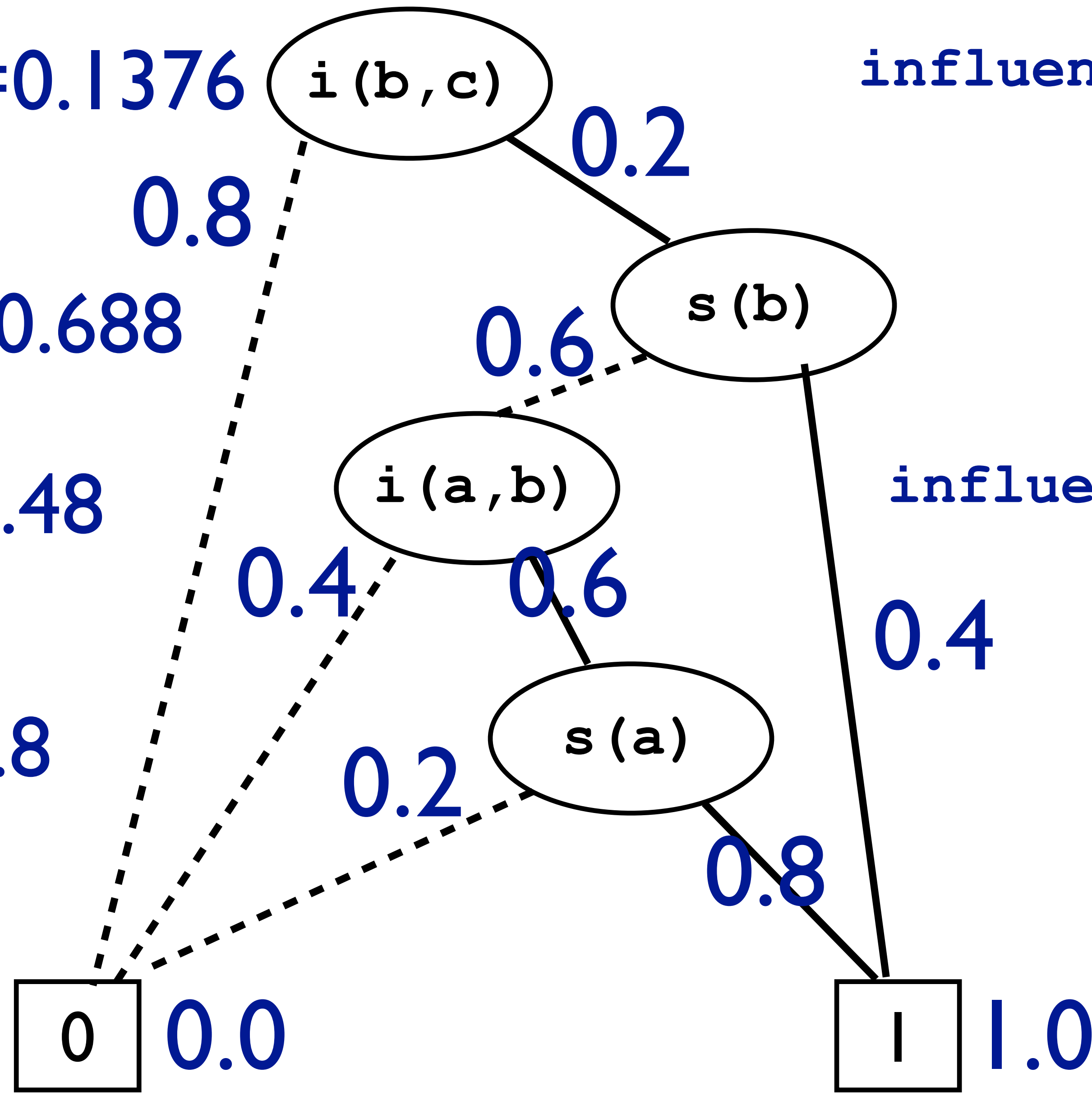
<u>B</u>inary <u>D</u>ecision <u>D</u>iagrams

$0.8 \times 0.0 + 0.2 \times 0.688 = 0.1376$

$0.6 \times 0.48 + 0.4 \times 1.0 = 0.688$

$0.4 \times 0.0 + 0.6 \times 0.8 = 0.48$

$0.2 \times 0.0 + 0.8 \times 1.0 = 0.8$



`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 & others)

Find all proofs of query

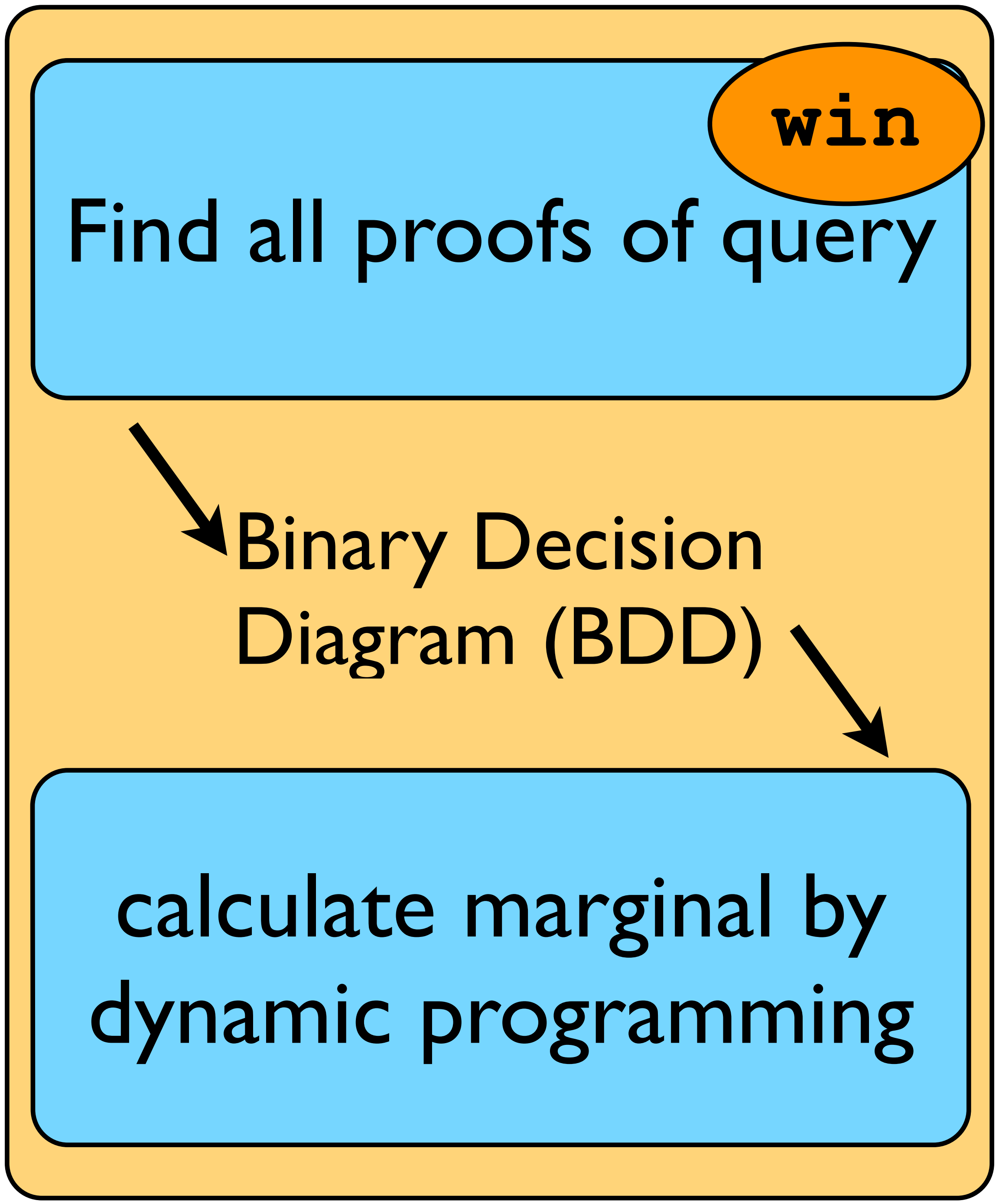
Binary Decision Diagram (BDD)

calculate marginal by dynamic programming

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

Initial Approach
(ProbLog1 & others)

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

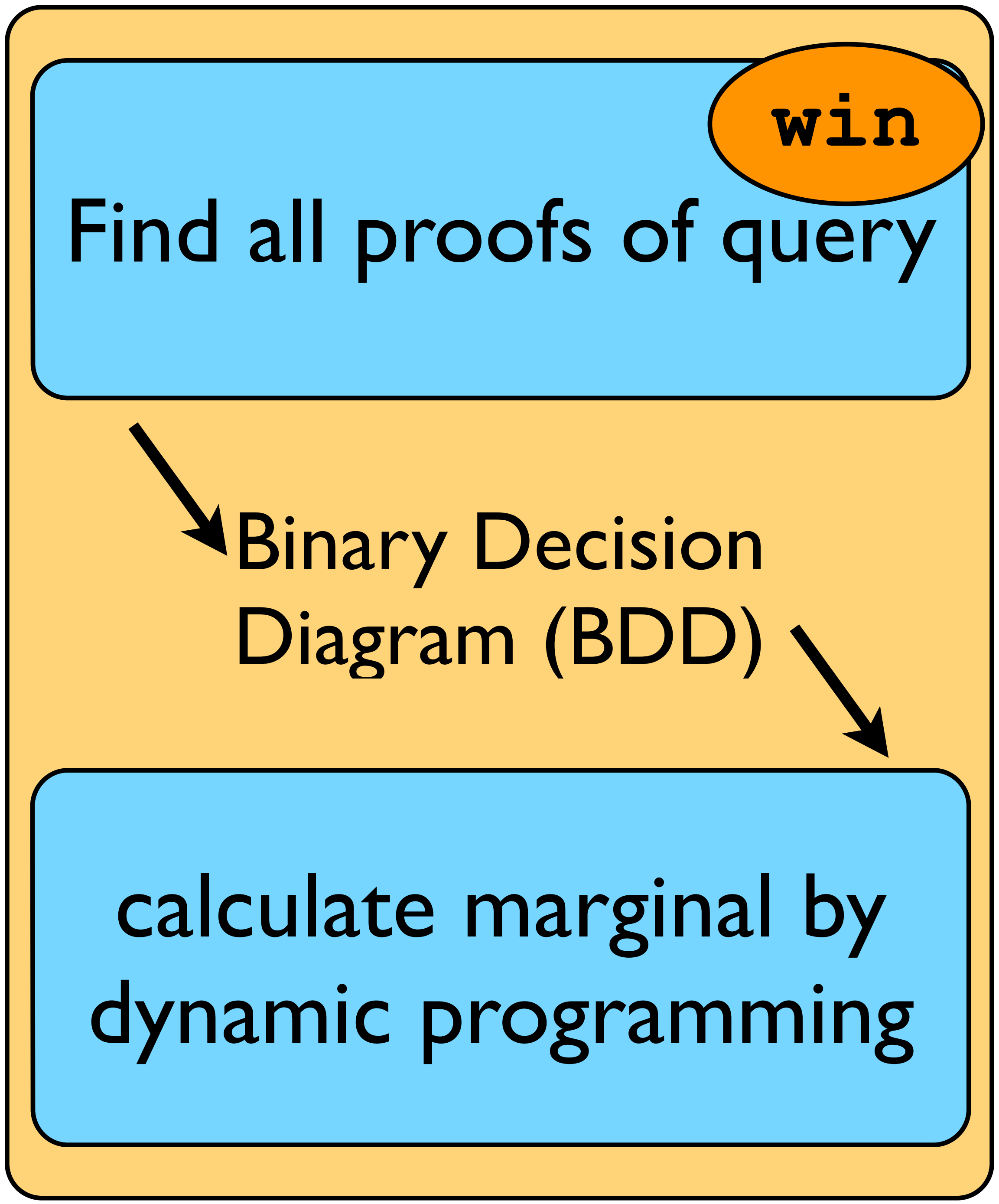


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Initial Approach

(ProbLog1 & others)

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



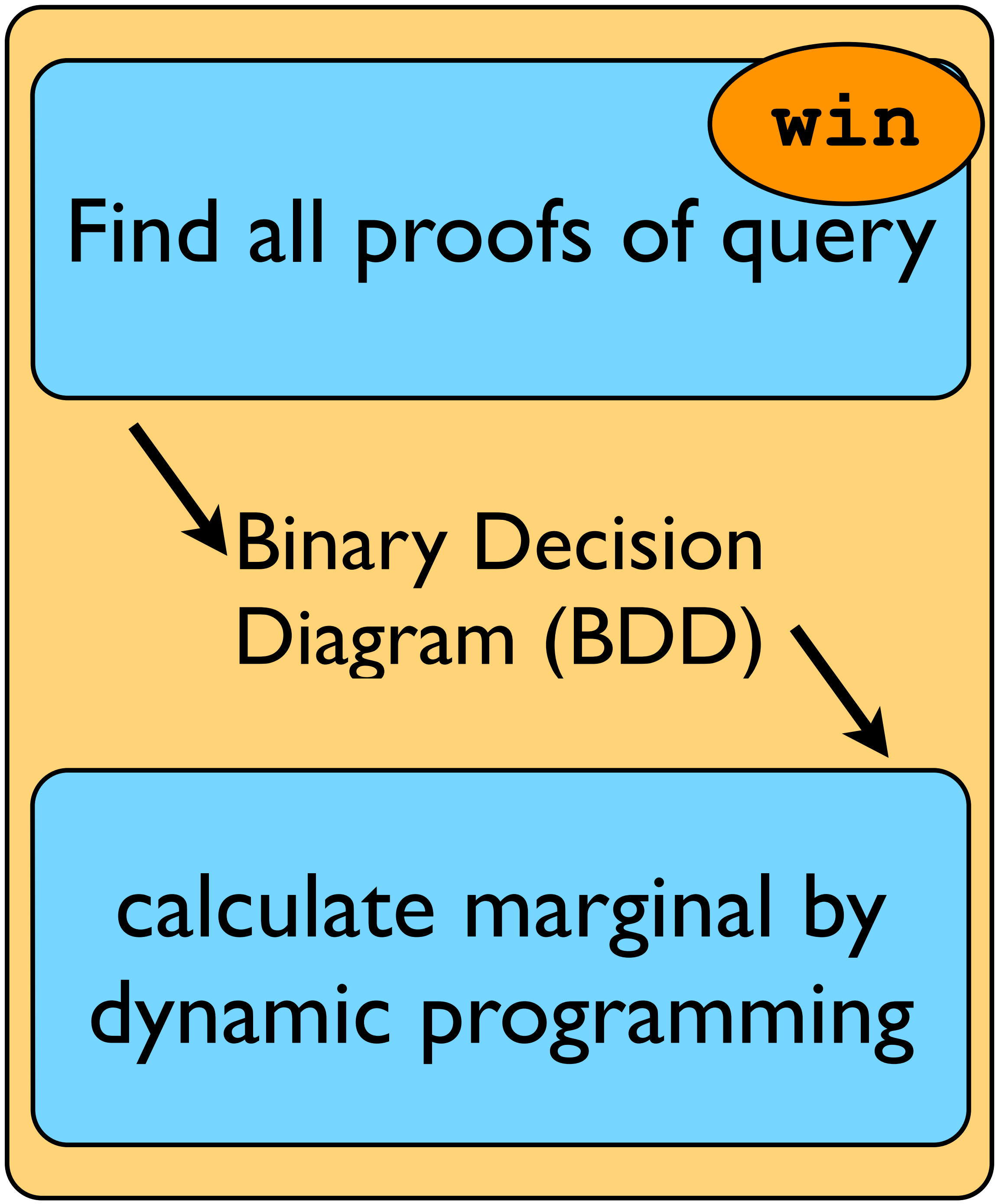

```
heads(1)
heads(2) & heads(3)
```

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

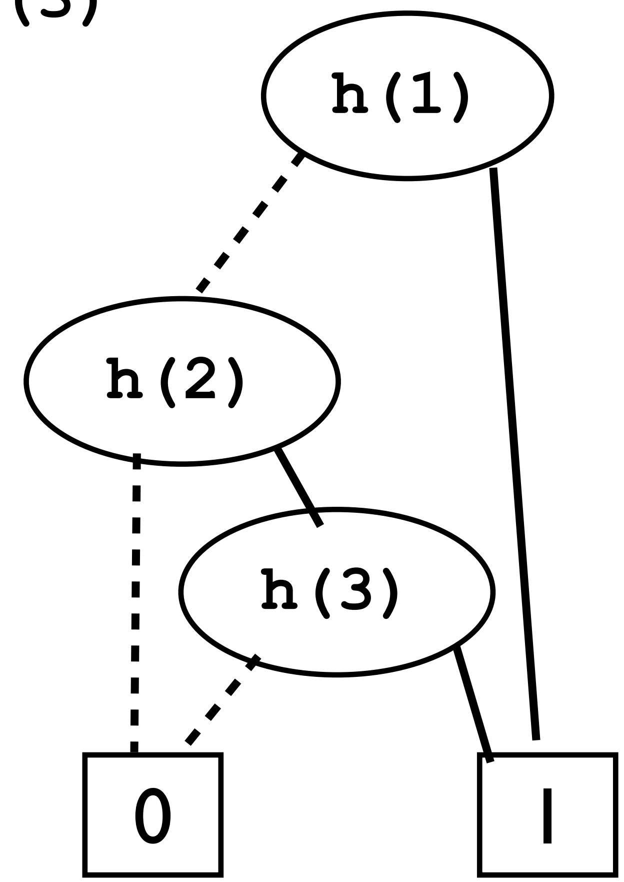
Initial Approach

(ProbLog1 & others)

```
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)
```

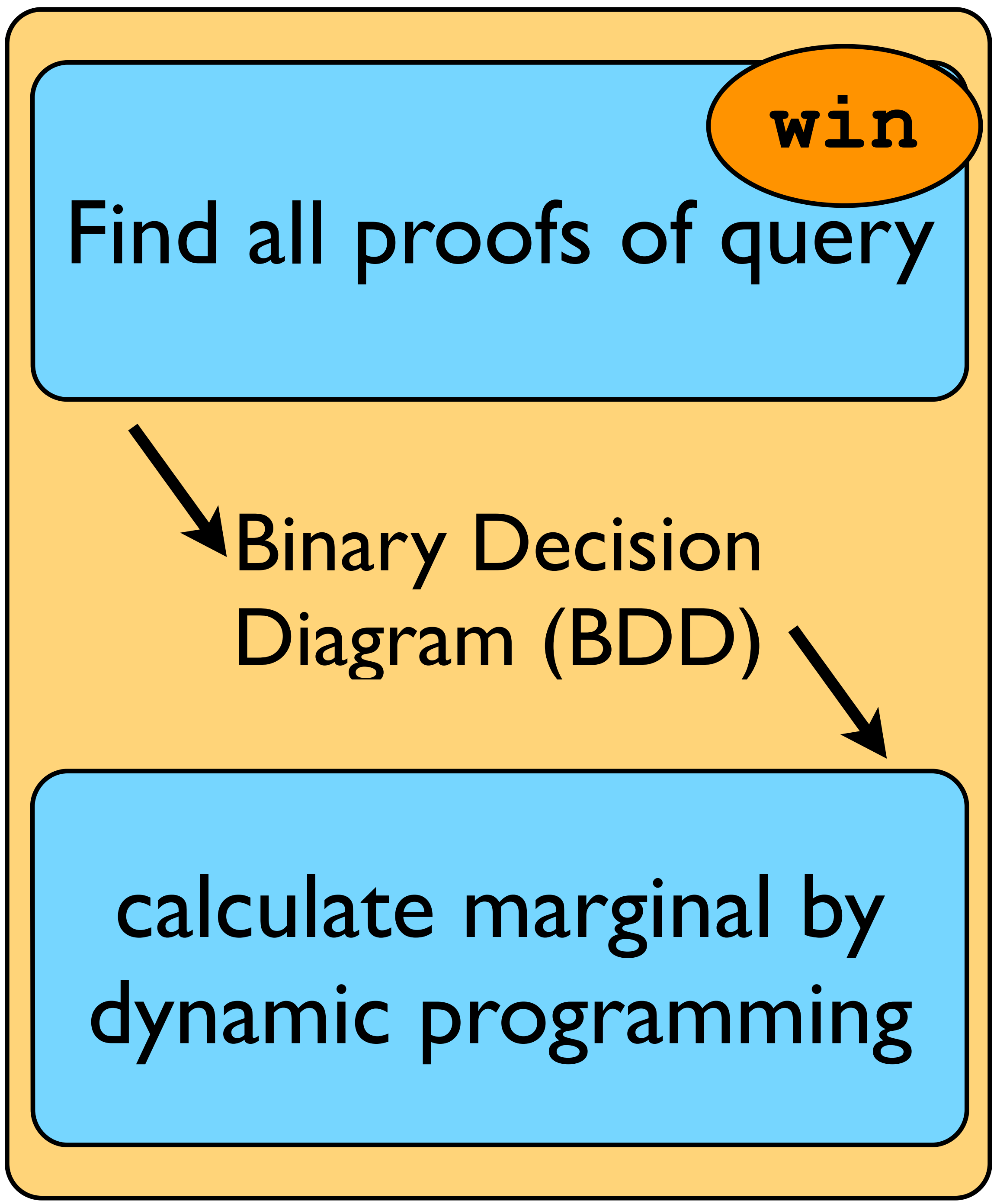


 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

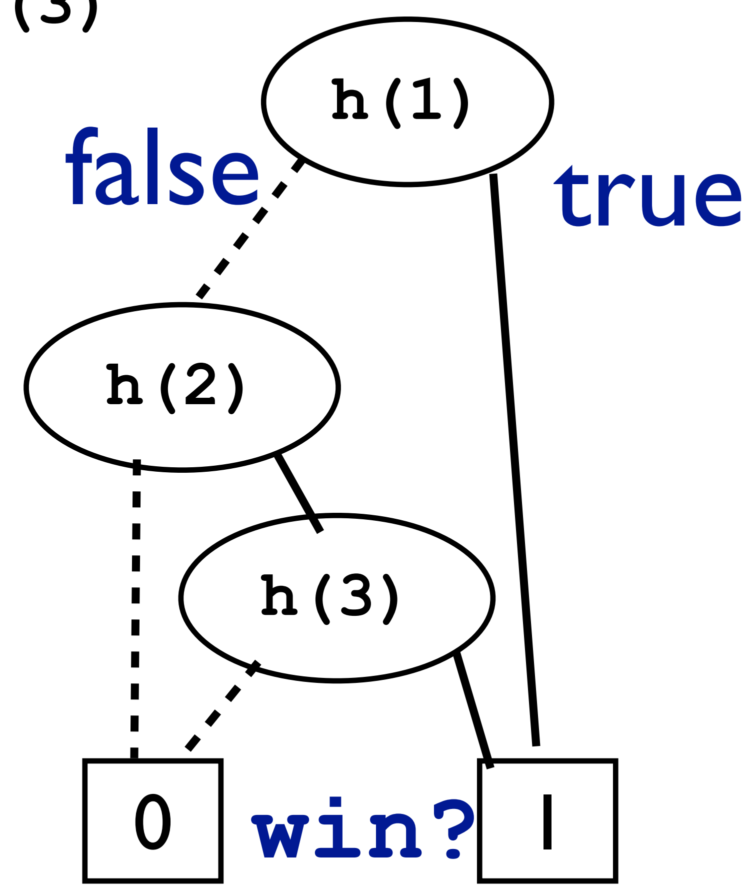
Initial Approach

(ProbLog1 & others)

```
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 & others)

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

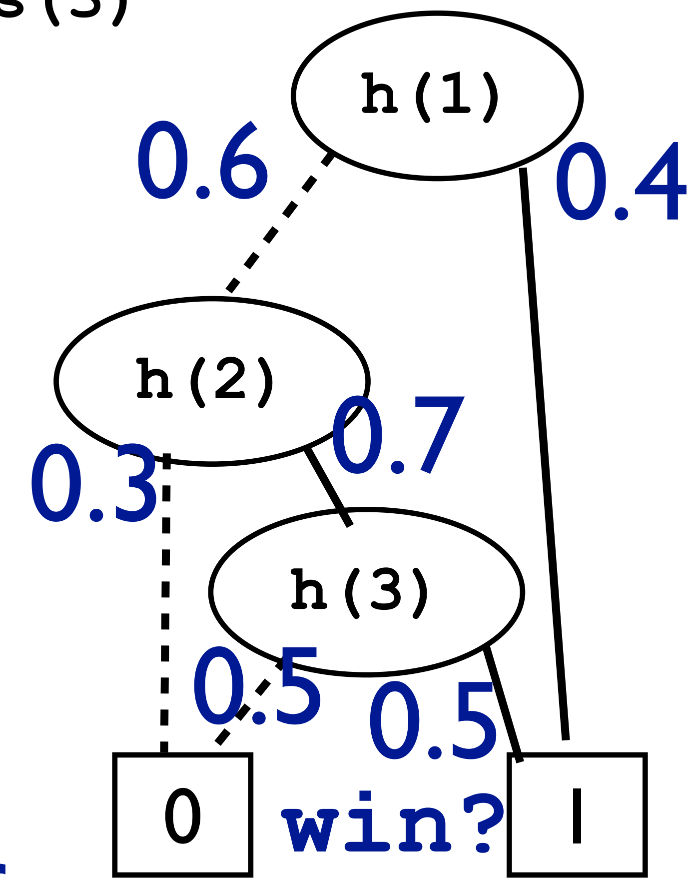
`win`

Find all proofs of query

Binary Decision Diagram (BDD)

calculate marginal by dynamic programming

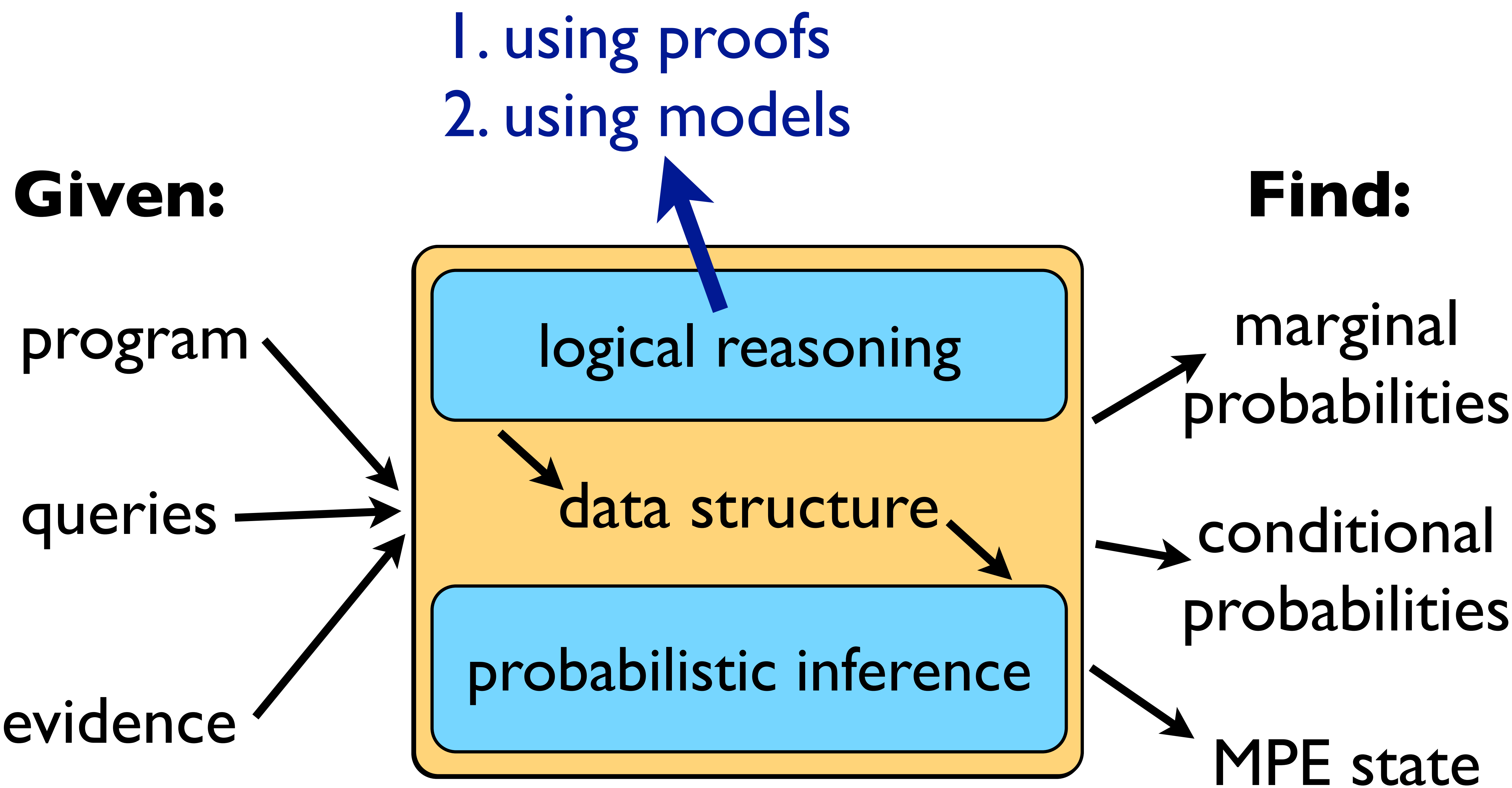
```
heads(1)
heads(2) & heads(3)
```



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

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

Answering Questions



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).
```

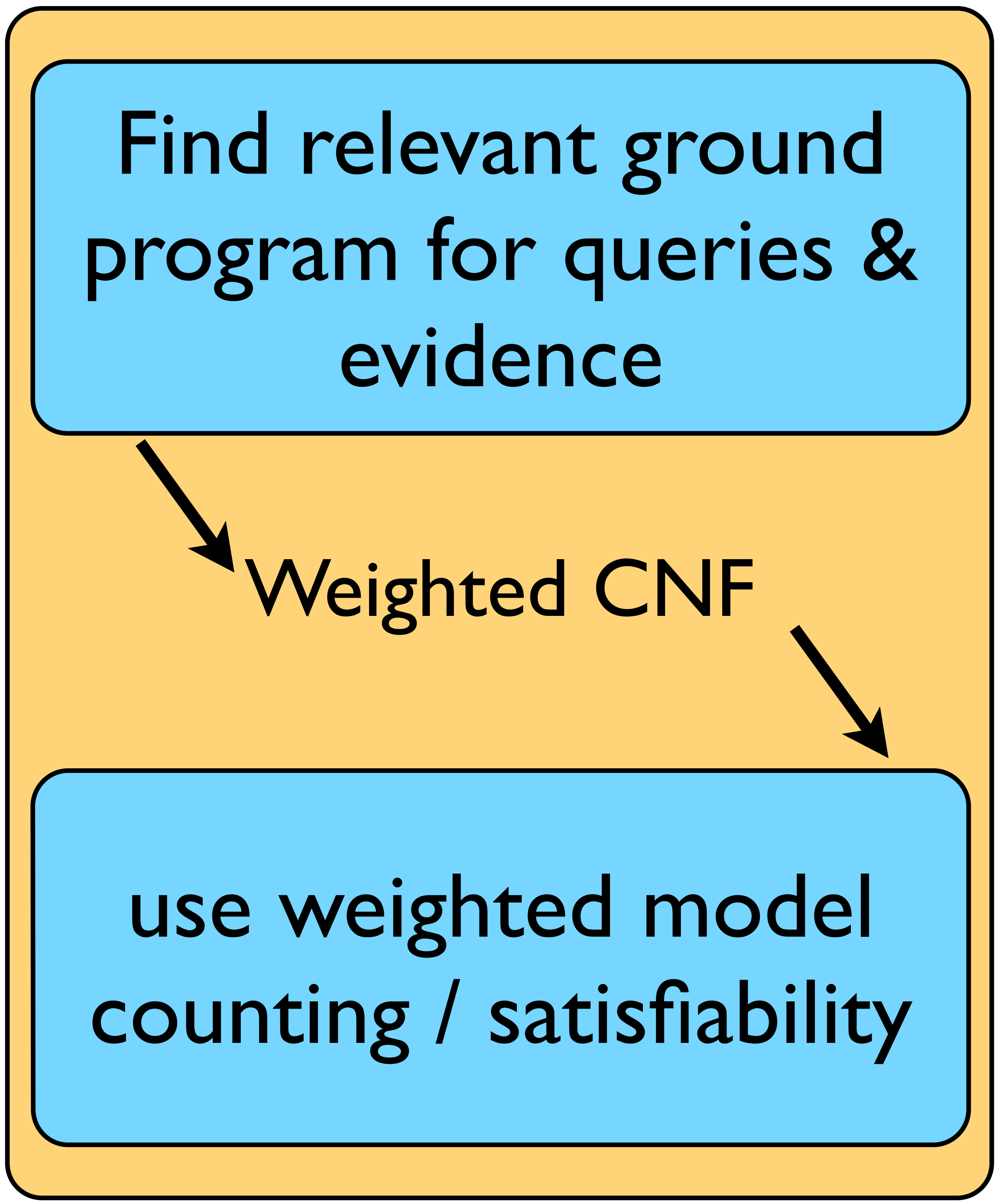
- Forward reasoning to construct unique model:
  - Start with database facts
  - Use rules to add more facts
- Query true iff in model

```
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

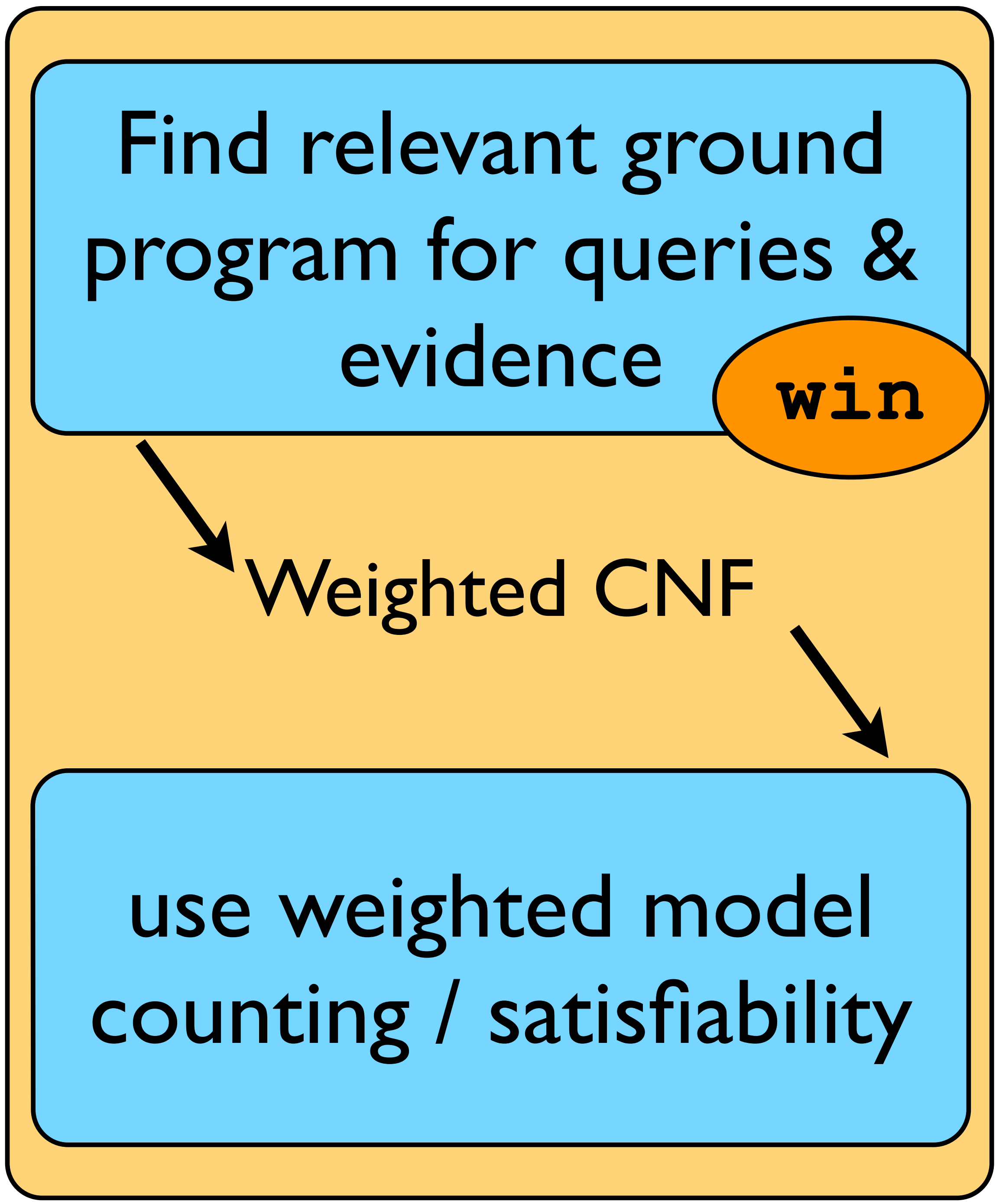
Current Approach
(ProbLog2)



[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).
```

Find relevant ground program for queries & evidence

`win`

Weighted CNF

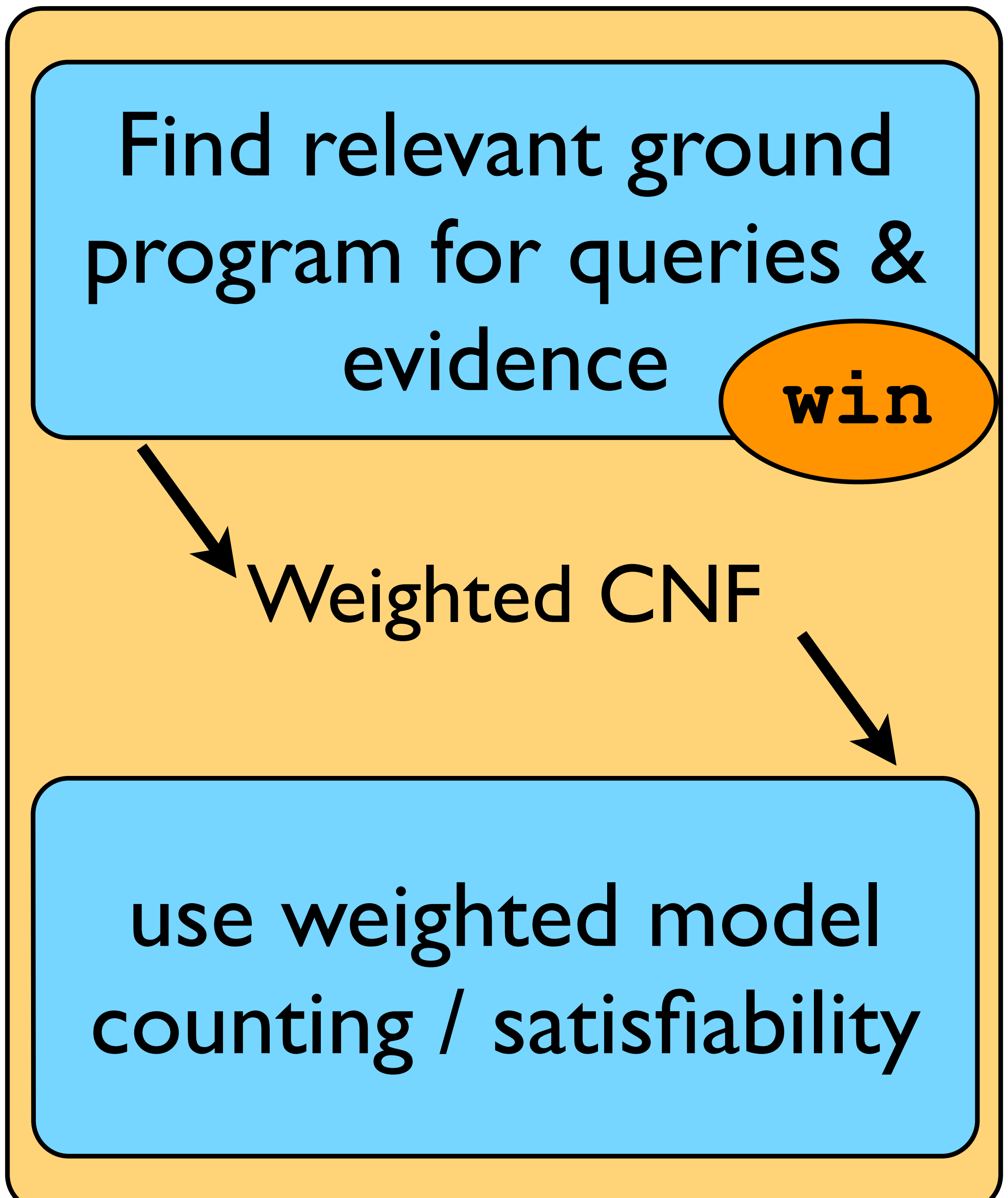
use weighted model counting / satisfiability

```
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).
```

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

[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).
```

Find relevant ground program for queries & evidence

win

Weighted CNF

use weighted model counting / satisfiability

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

$$\downarrow$$

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

$$\downarrow$$

$$\begin{aligned} &(\neg \mathrm{win} \vee h(1) \vee h(2)) \\ \wedge\ &(\neg \mathrm{win} \vee h(1) \vee h(3)) \\ \wedge\ &(\mathrm{win} \vee \neg h(1)) \\ \wedge\ &(\mathrm{win} \vee \neg h(2) \vee \neg h(3)) \end{aligned}$$

[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).
```

Find relevant ground program for queries & evidence

win

Weighted CNF

use weighted model counting / satisfiability

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

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

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

$h(1) \rightarrow 0.4$ $\quad h(2) \rightarrow 0.7$ $\quad h(3) \rightarrow 0.5$

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

[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).
```

Find relevant ground program for queries & evidence

win

Weighted CNF

use weighted model counting / satisfiability

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

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

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

use standard solver

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

[Fierens et al, TPLP 14]

Weighted Model Counting

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

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

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

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

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

interpretations (truth value assignments) of propositional variables

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

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

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

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

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

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

weight of literal

interpretations (truth value assignments) of propositional variables

possible worlds

Weighted Model Counting

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

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

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

interpretations (truth value assignments) of propositional variables

possible worlds

Weighte

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

propositional formula in conjunctive normal form (CNF)

given by ProbLog program & query

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

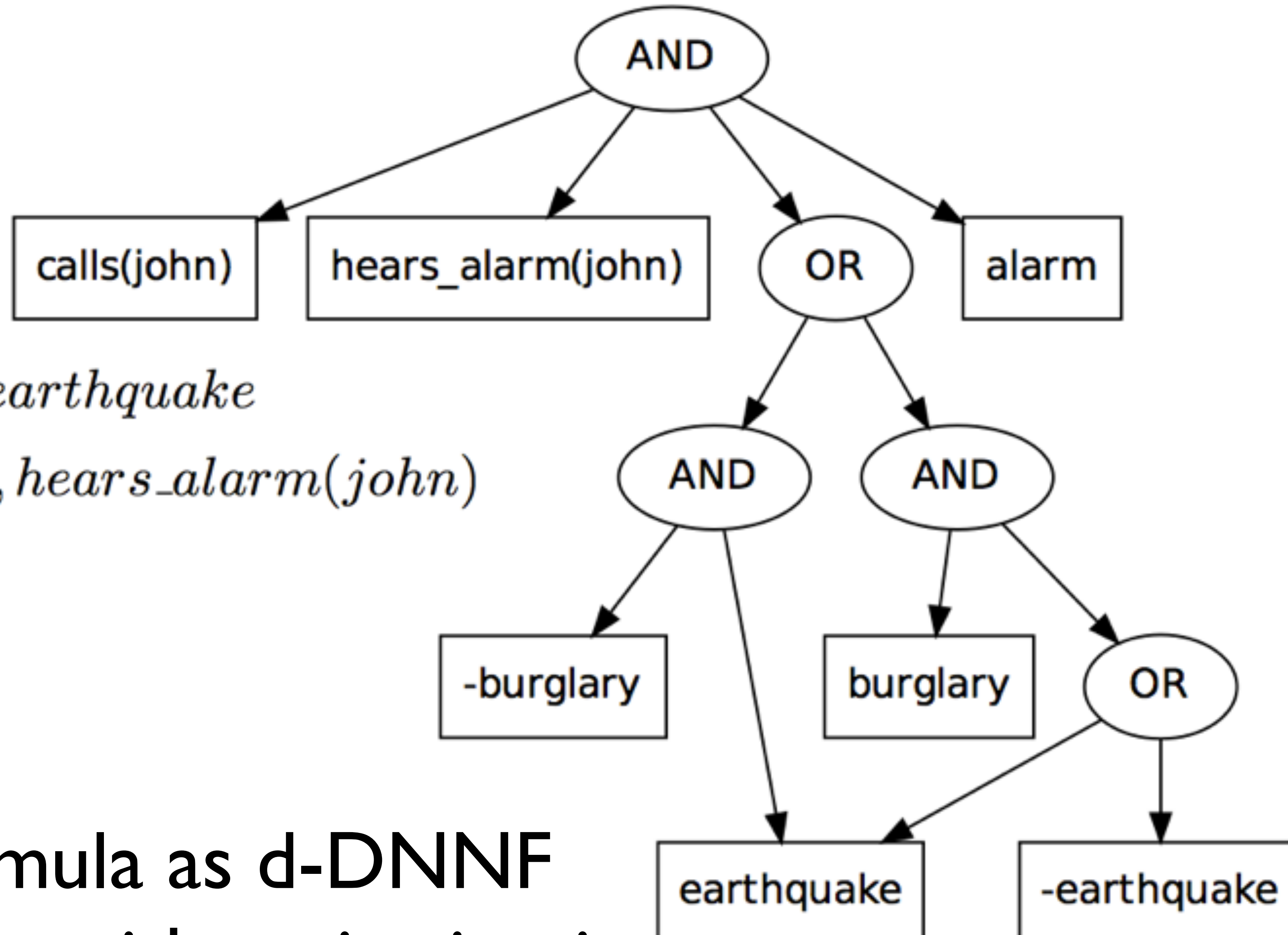
weight of literal

interpretations (truth value assignments) of propositional variables

possible worlds

for p::f,
w(f) = p
w(not f) = 1−p

WMC using d-DNNFs



$alarm \leftrightarrow burglary \vee earthquake$

$calls(john) \leftrightarrow alarm, hears\_alarm(john)$

$calls(john)$

1. represent formula as d-DNNF
2. transform into arithmetic circuit
3. evaluate bottom-up

[Figure: Fierens et al, TPLP 14]

ProbLog Inference

- reduction to propositional formula
- addresses disjoint-sum-problem
- **but**: not all probabilistic logic programs face this problem! e.g., weather
- more generally: mutually exclusive proofs as assumed in PRISM

Query Evaluation in PDB

Query Evaluation in PDB

- **Extensional** evaluation
  - guided by query expression only
  - exploit DB technology
  - for queries known to have polytime evaluation

Query Evaluation in PDB

- **Extensional** evaluation
  - guided by query expression only
  - exploit DB technology
  - for queries known to have polytime evaluation
- **Intensional** evaluation
  - construct **lineage** (= propositional formula)
  - compute probability of lineage
  - all queries

same idea as for ProbLog

Approximate Inference

- Lower and upper bounds

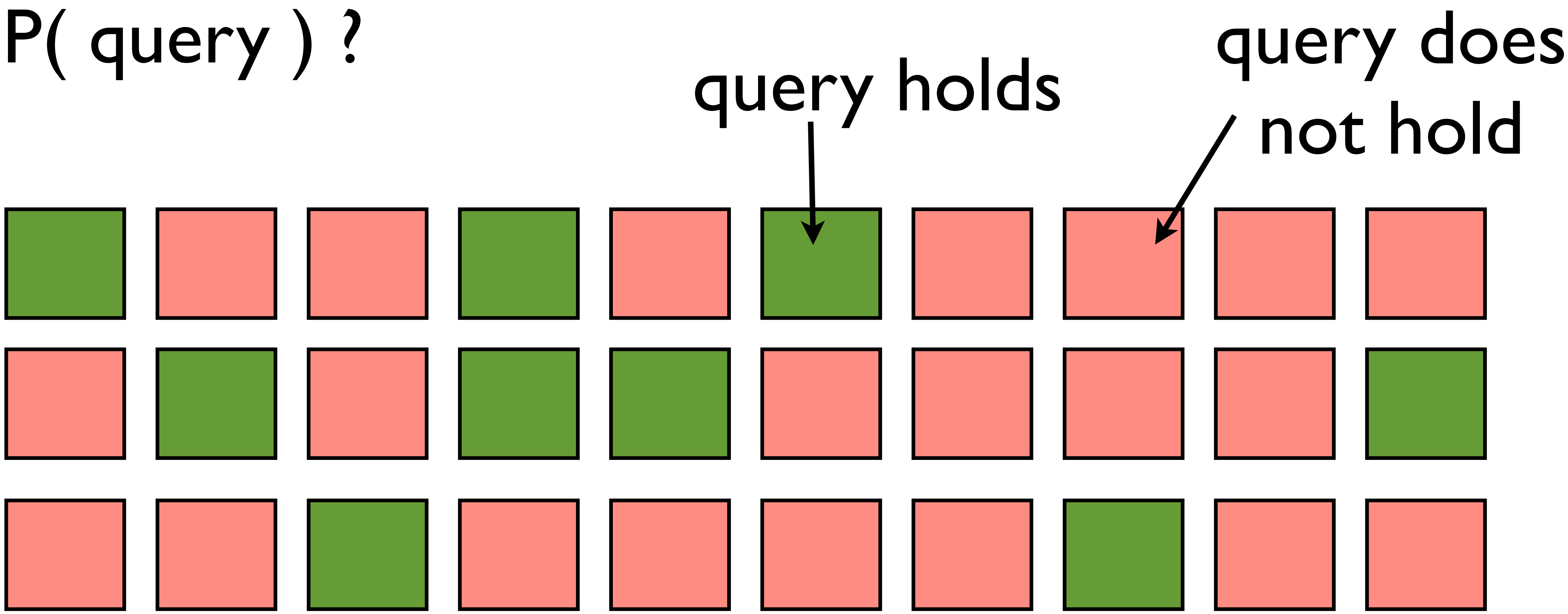
$$\phi_L \models \phi \models \phi_U$$

$$P(\phi_L) \leq P(\phi) \leq P(\phi_U)$$

- Sampling

Sampling

- P(query) ?



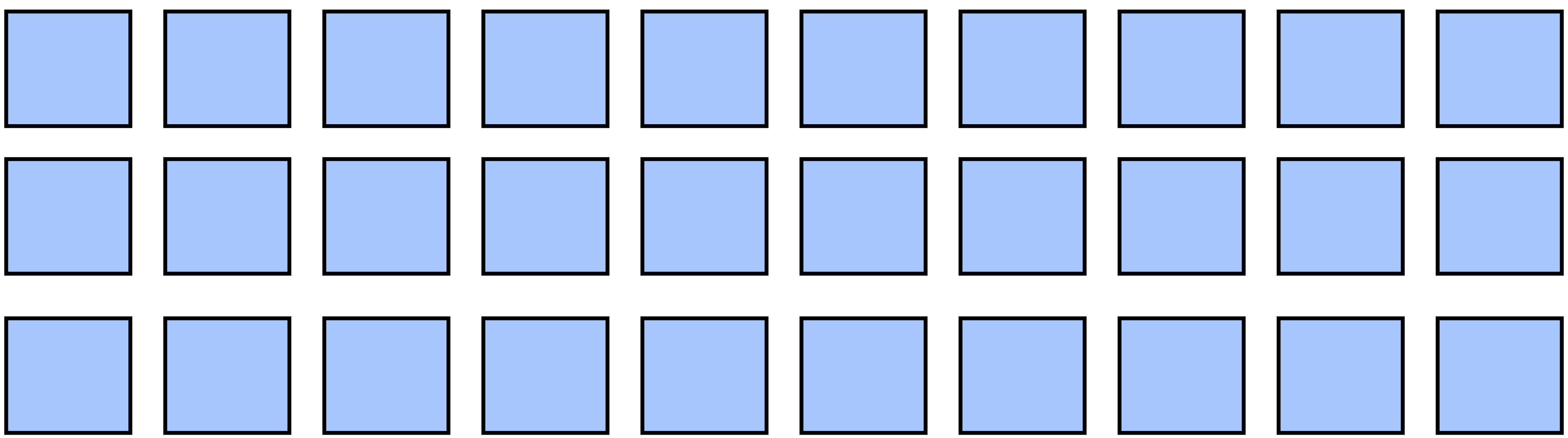
$$P(\text{query}) \approx \frac{\#\text{ query holds}}{\#\text{ worlds sampled}}$$

Rejection Sampling

- P(query | evidence) ?

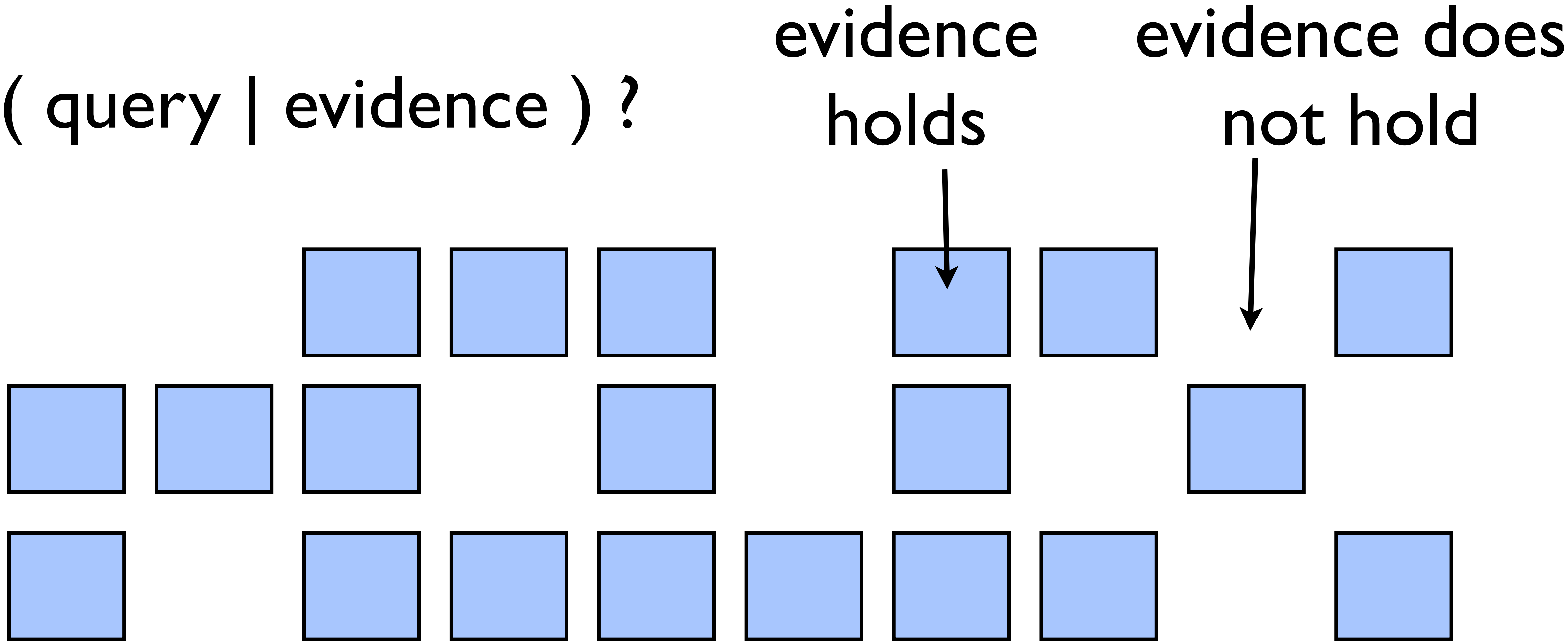
Rejection Sampling

- P(query | evidence) ?

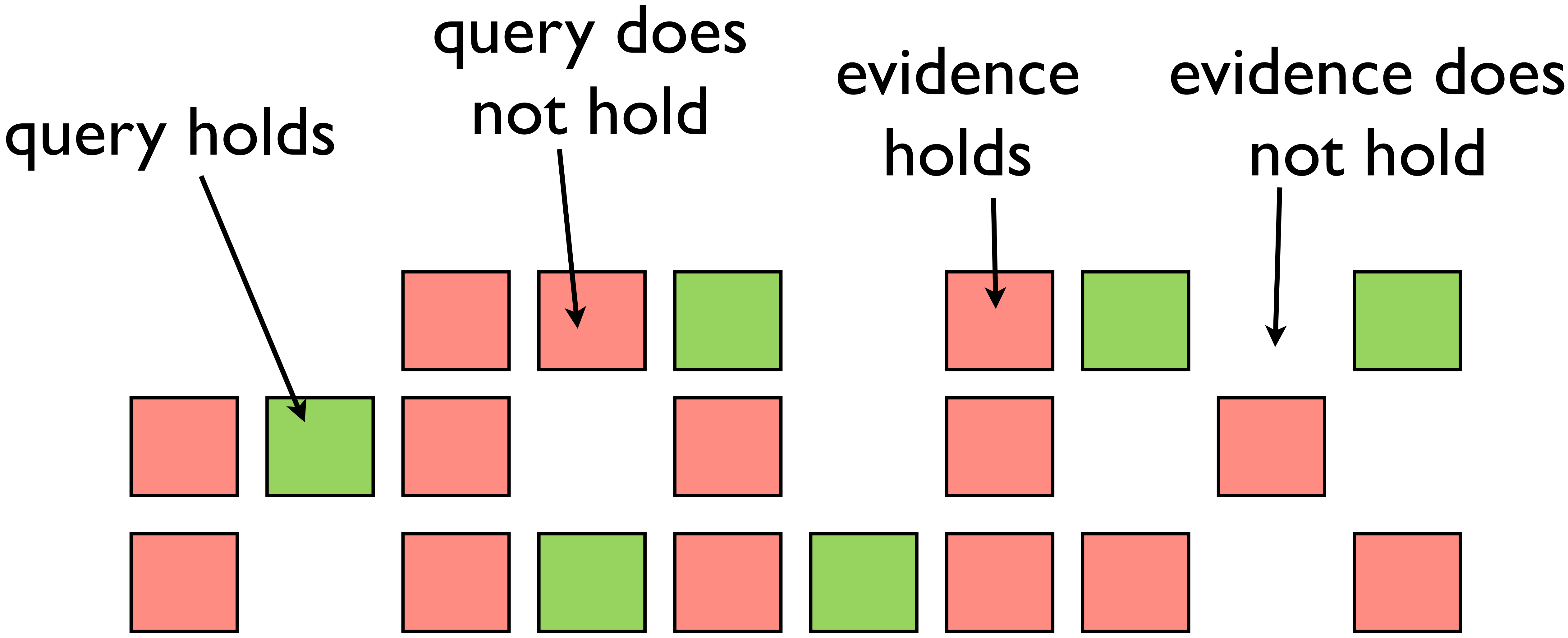


Rejection Sampling

- P(query | evidence) ?



Rejection Sampling



$$P(\text{query} \mid \text{evidence}) \approx \frac{\#\ \text{query \& evidence holds}}{\#\ \text{evidence holds}}$$

Markov Chain Monte Carlo (MCMC)

- Generate next sample by modifying current one
- Most common inference approach for PP languages such as Church, BLOG, ...
- Also considered for PRISM and ProbLog

Key challenges:
- how to propose next sample
- how to handle evidence

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- **Learning**
- Dynamics and Decisions
- Markov Logic another representative of SRL

... with some detours on the way

Parameter Learning

e.g., webpage classification model

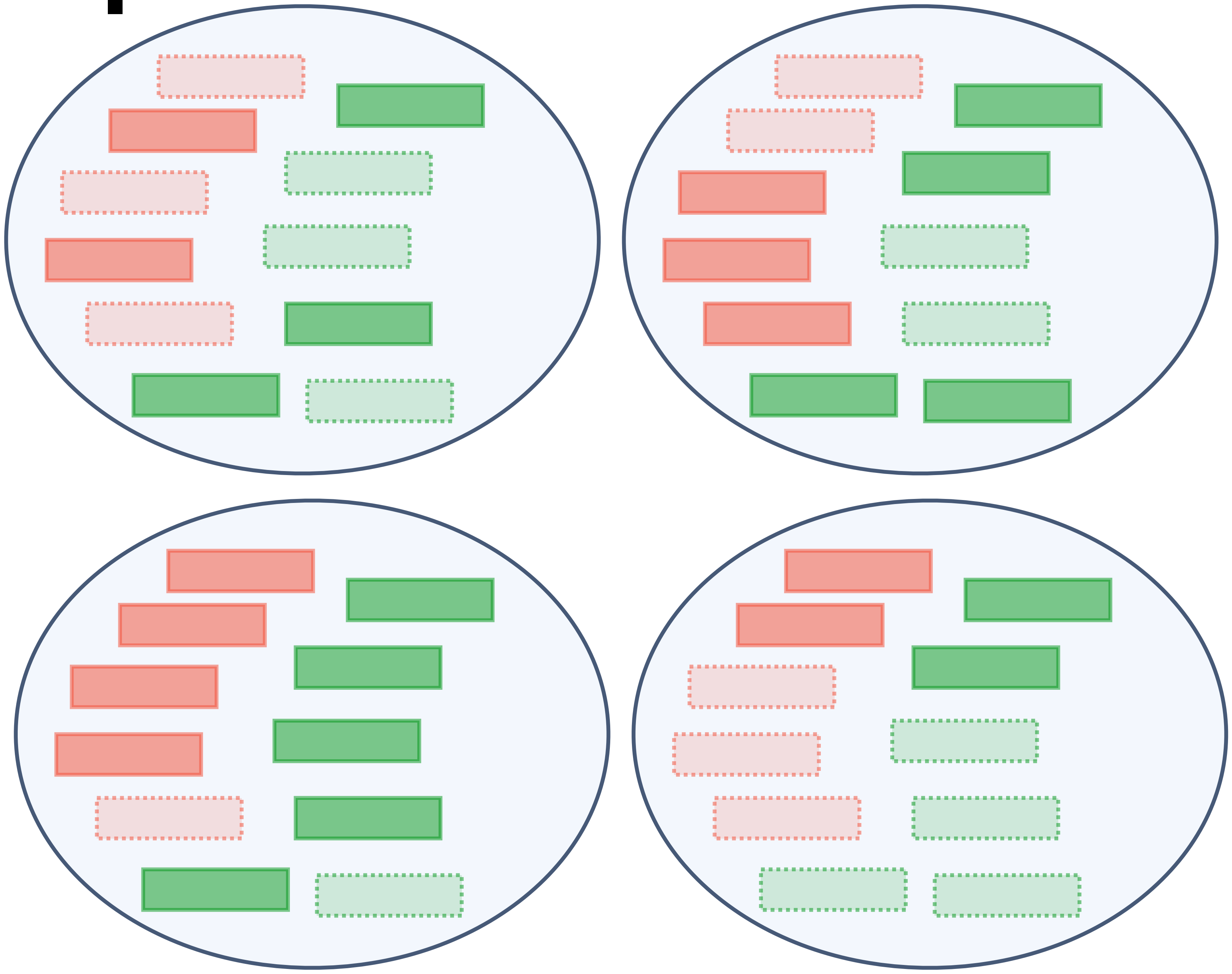
for each *CLASS1*, *CLASS2* and each *WORD*

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

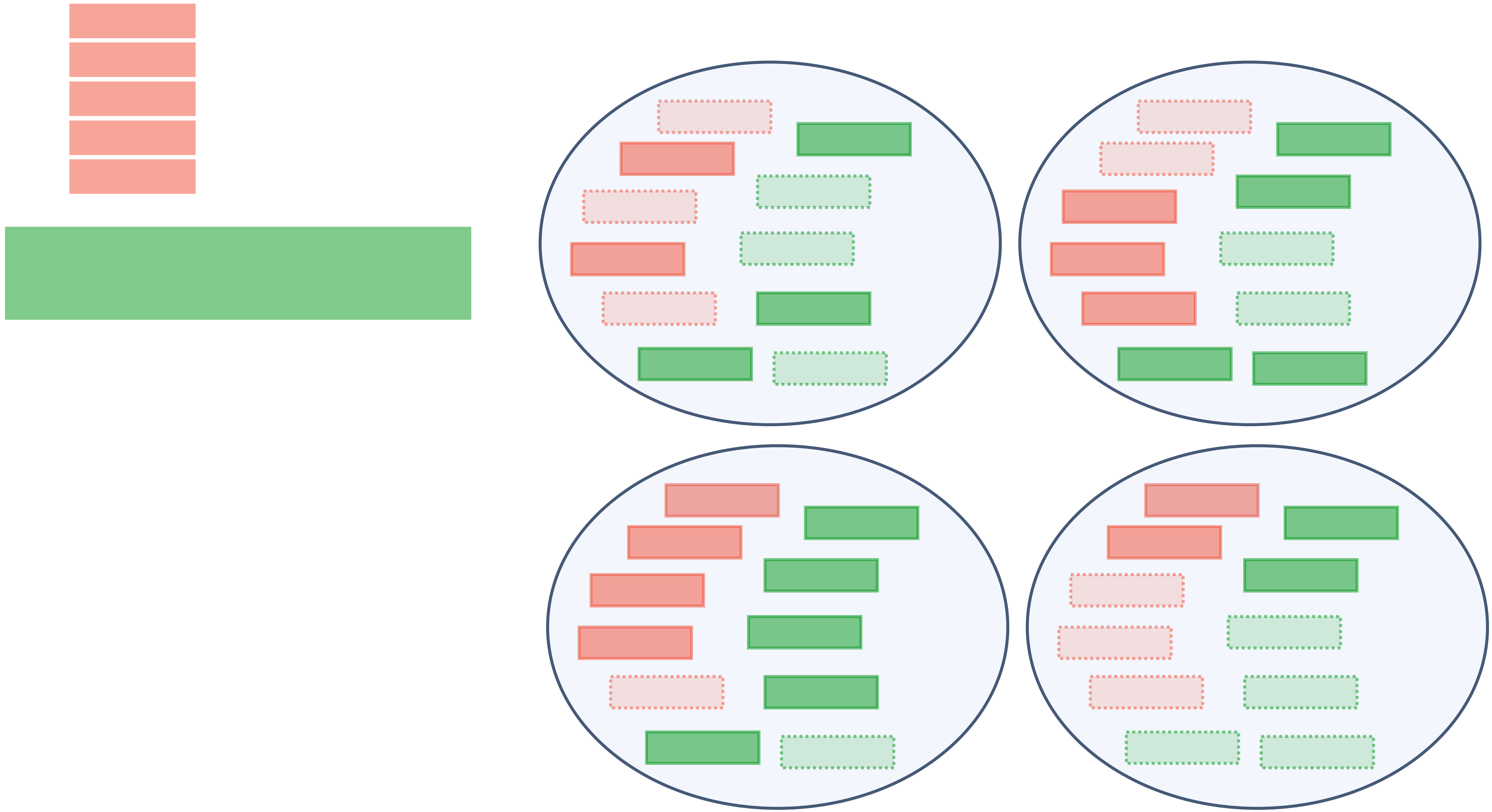
```
class(Page,C)    :-   has_word(Page,W), word_class(W,C).

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

Sampling Interpretations

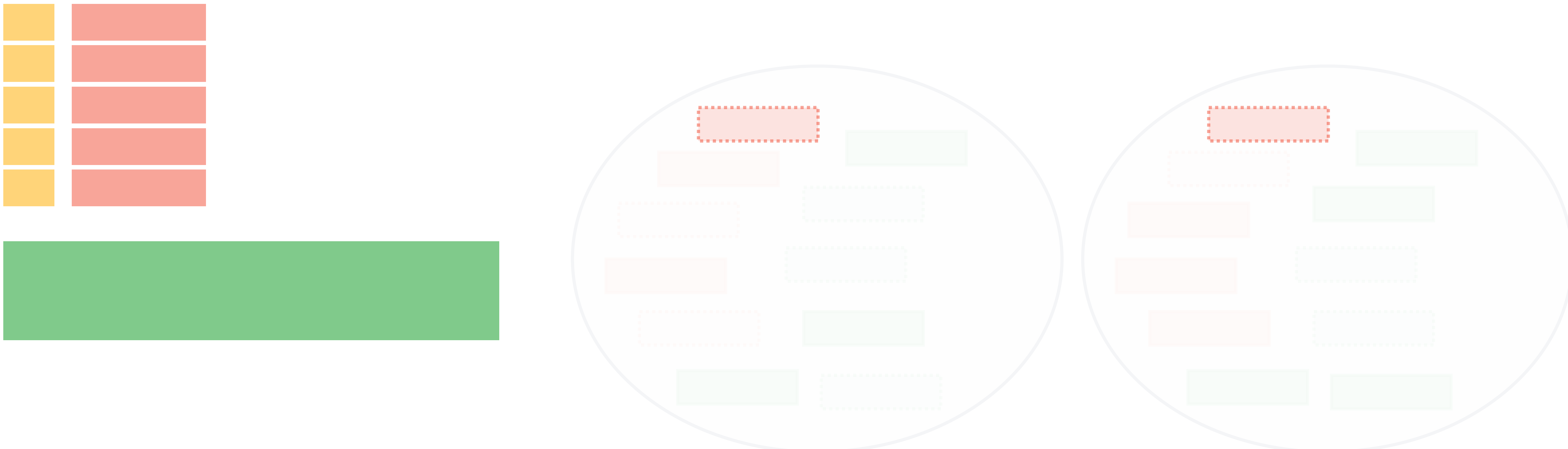
Sampling Interpretations



Parameter Estimation

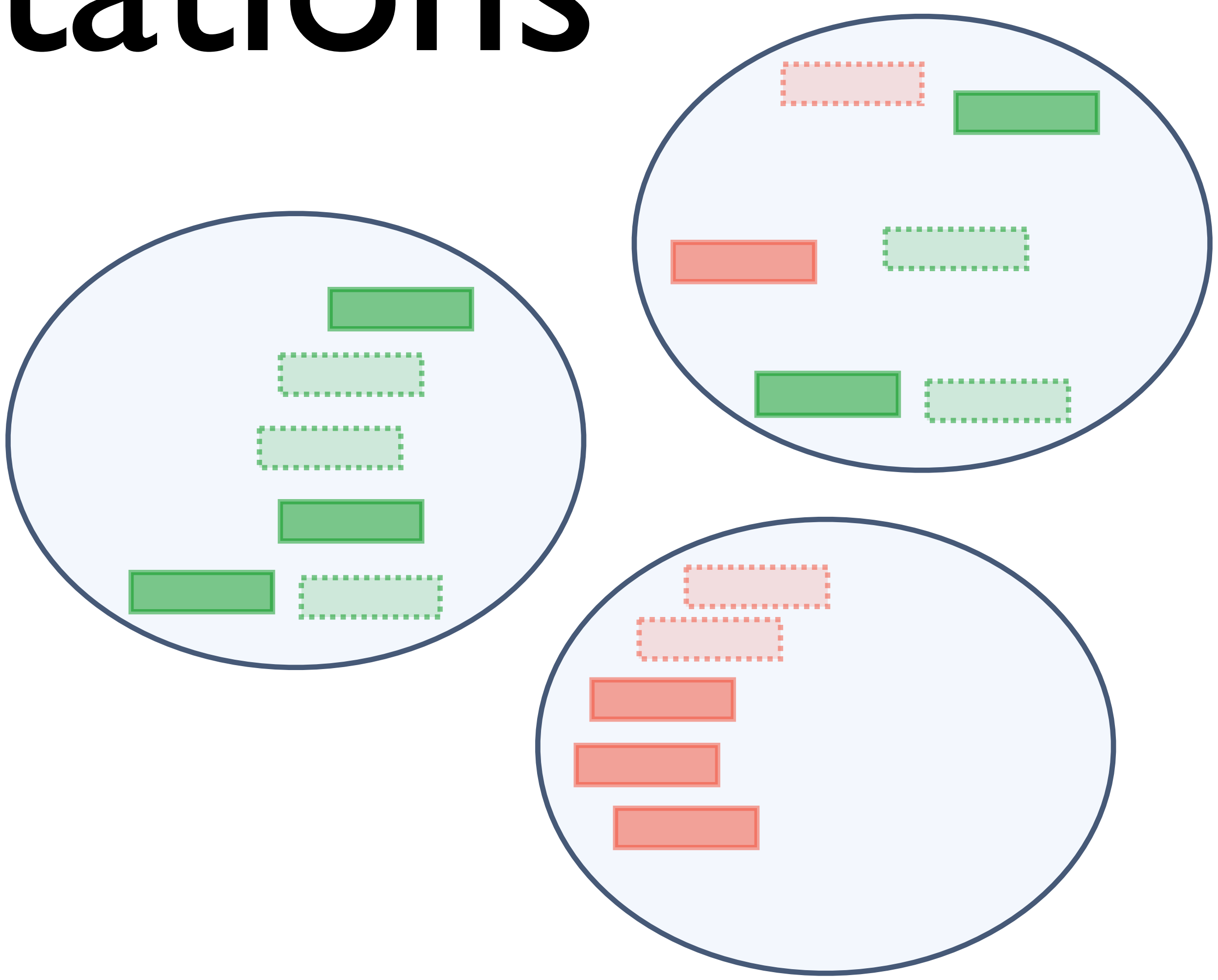


Parameter Estimation



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

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]

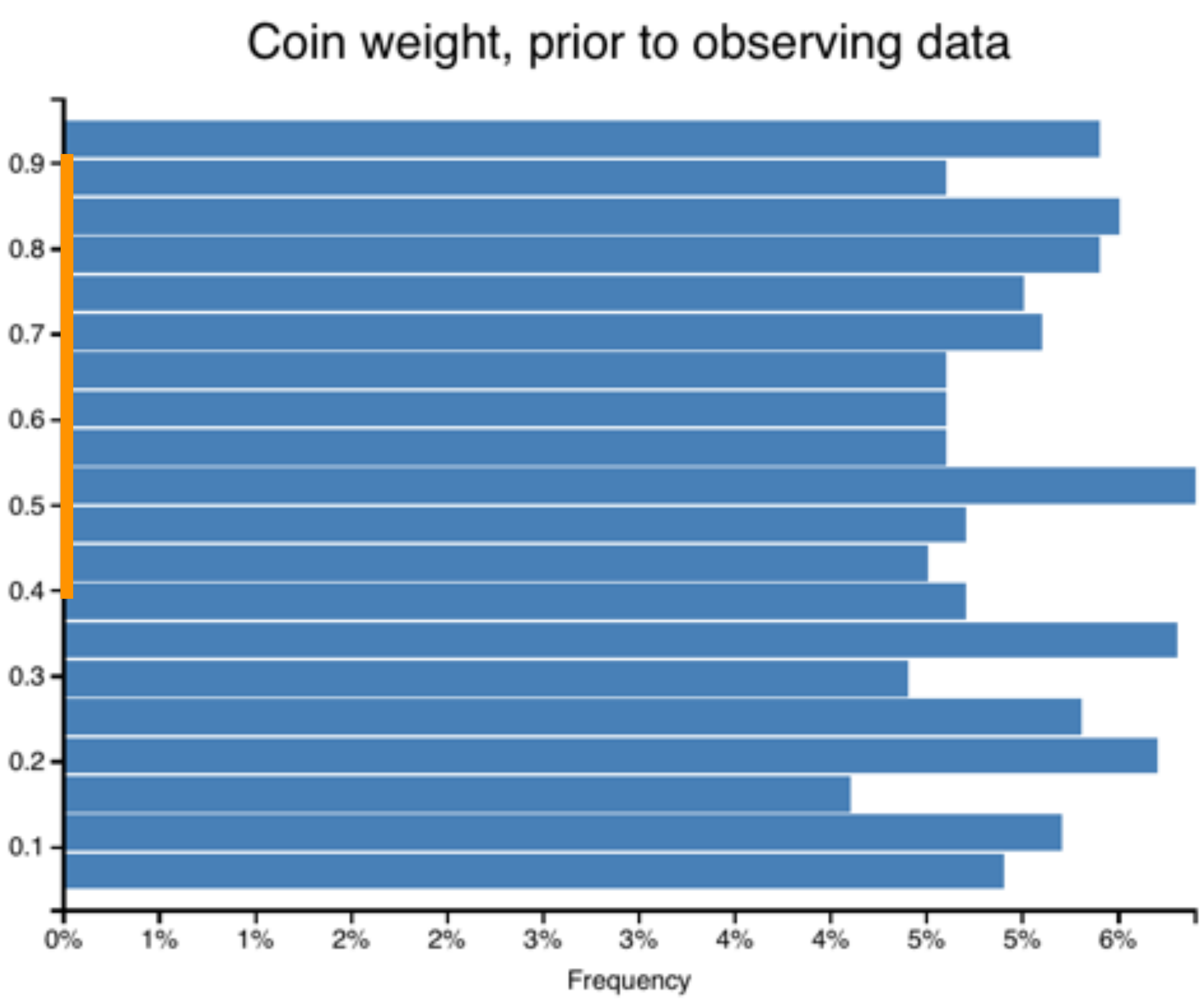
Bayesian Parameter Learning

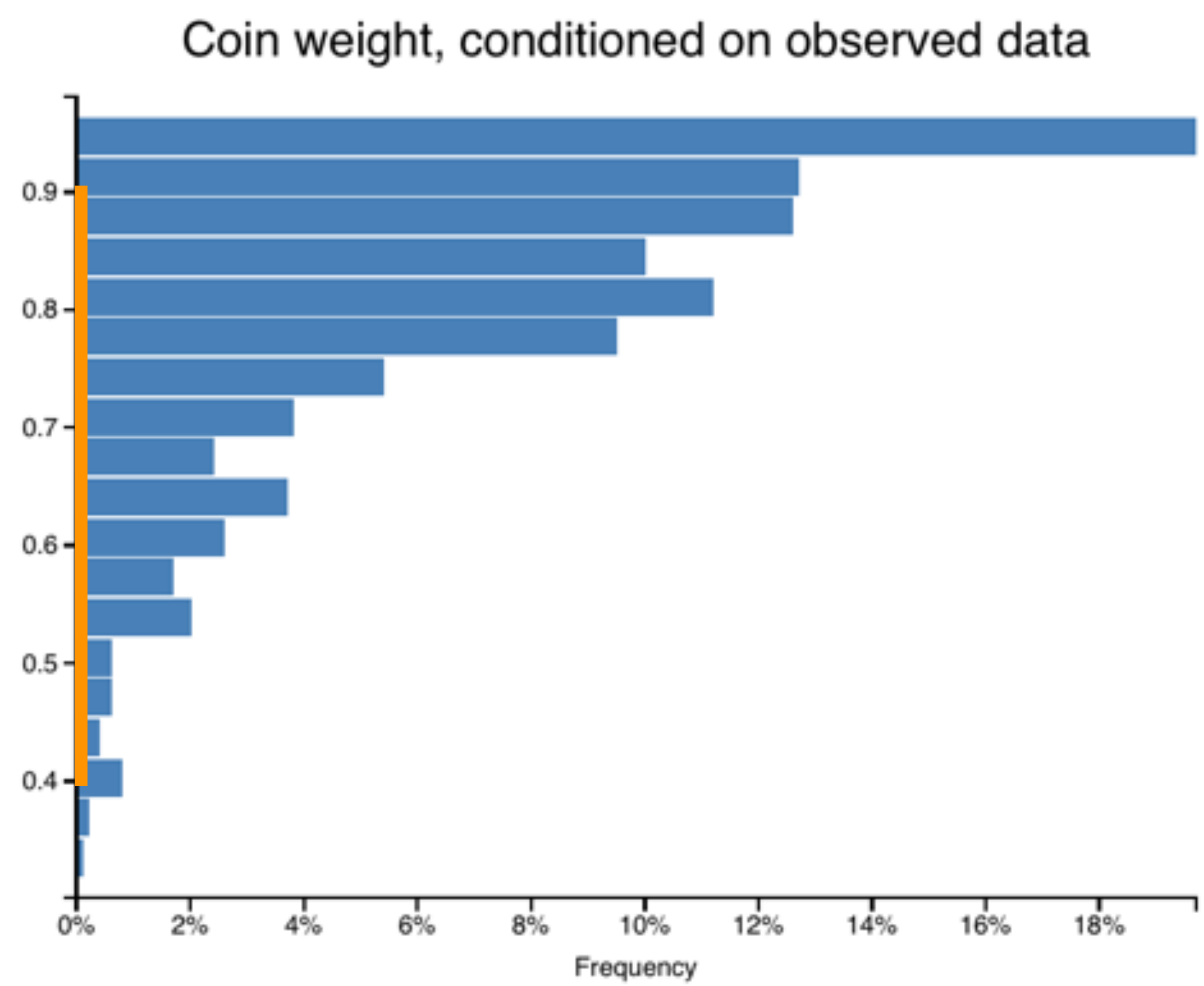
- Learning as inference (e.g., Church)
- Prior distributions for parameters
- Given data, find most likely parameter values

Example

[from probmods.org]

- Flipping a coin with unknown weight
- Prior: uniform distribution on [0,1]
- Observation: 5x heads in a row
- Sampling from Church model:





ProbLog Example

prior

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                 0.2::weight(C,0.7); 0.05::weight(C,0.9) :- coin(C).
```

```
Param::toss(_,Param,_).
heads(C,R) :- weight(C,Param),toss(C,Param,R).
tails(C,R) :- weight(C,Param),\+toss(C,Param,R).

data(C,[]).
data(C,[h|R]) :- heads(C,R), data(C,R).
data(C,[t|R]) :- tails(C,R), data(C,R).
```

```
coin(c1).
coin(c2).
param(0.1).
param(0.3).
param(0.5).
param(0.7).
param(0.9).
```

```
query(weight(C,X)) :- coin(C),param(X).
```

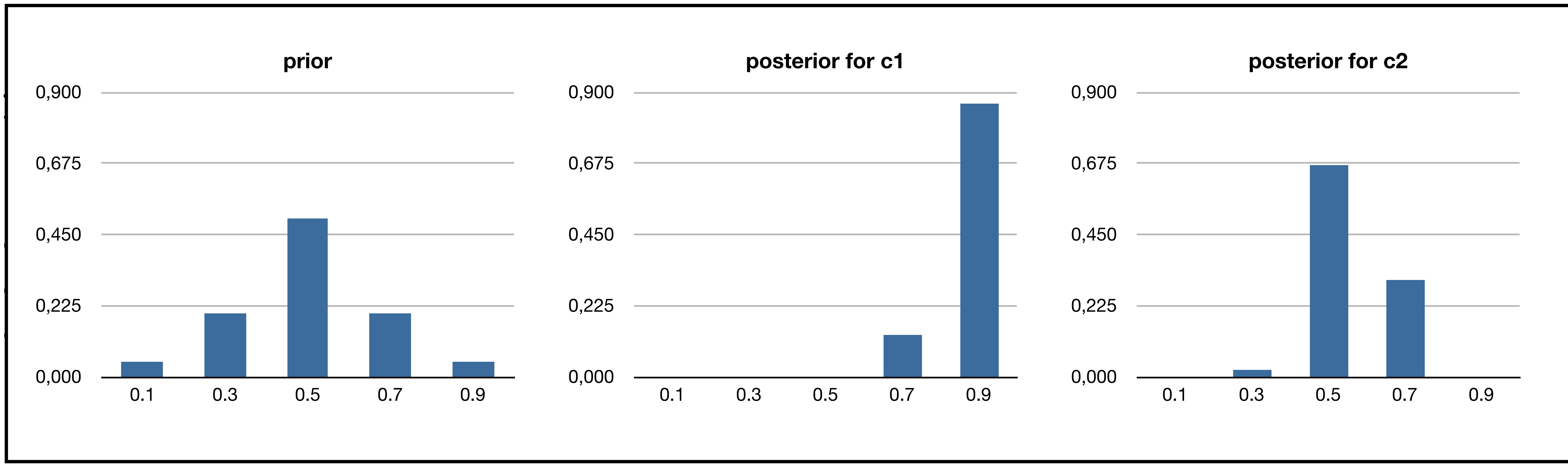
ask for posterior

```
evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

data

ProbLog Example

```
0.05::weight(C,0.1); 0.2::weight(C,0.3); 0.5::weight(C,0.5);
                 0.2::weight(C,0.7); 0.05::weight(C,0.9) :- coin(C).
```




```
query(weight(C,X)) :- coin(C),param(X).

evidence(data(c1,[h,h,h,h,h,h,h,h,h,h,h,h,h]),true).
evidence(data(c2,[h,t,h,h,h,h,h,t,t,h,t,t,h]),true).
```

Roadmap

- Modeling (ProbLog and Church, another representative of PP)
- Inference
- **Learning**
- **Dynamics and Decisions**
- Markov Logic another representative of SRL

... with some detours on the way

World Dynamics

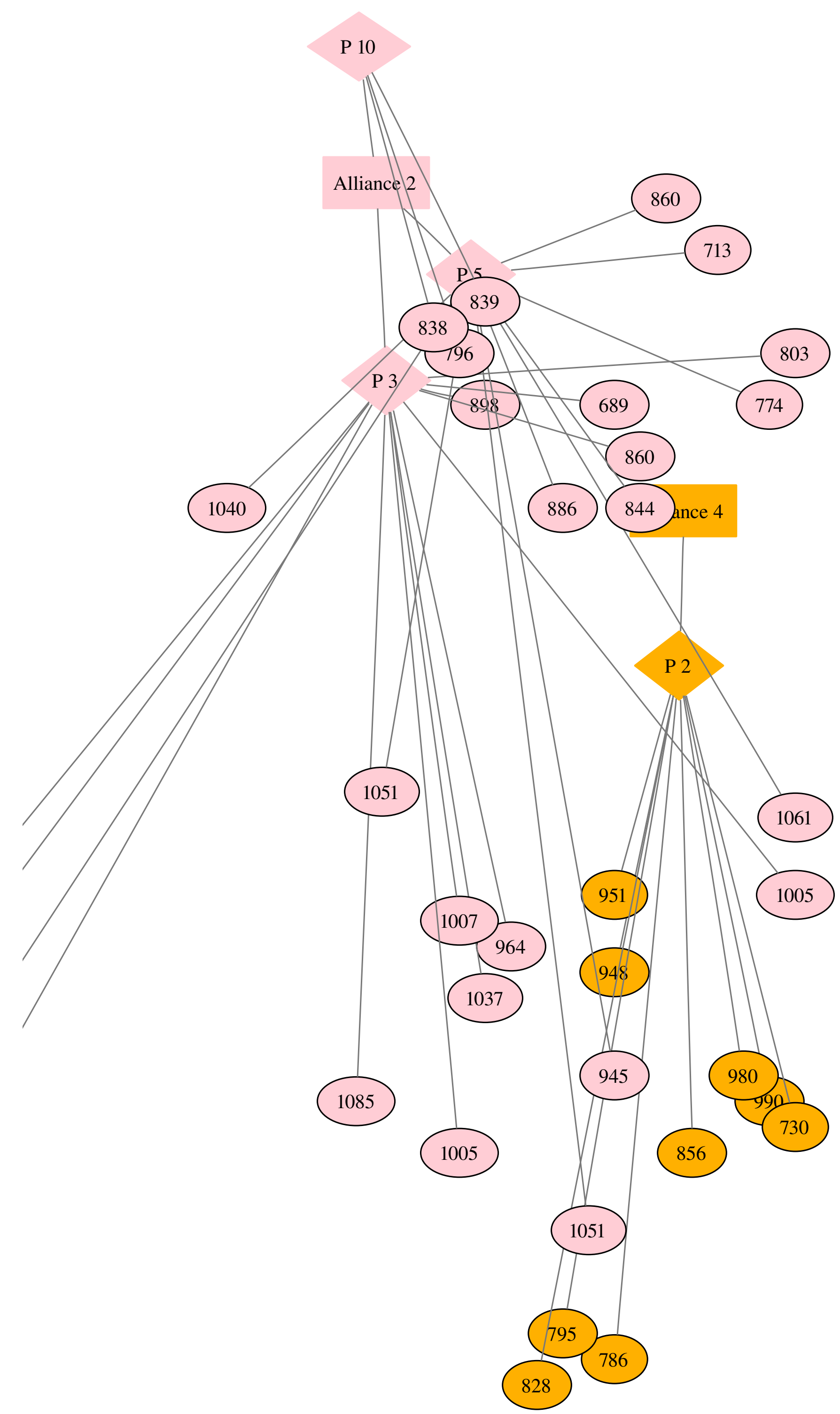
Fragment of world with

~10 alliances
~200 players
~600 cities

alliances color-coded

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

[Thon, Landwehr, De Raedt, ECML08]

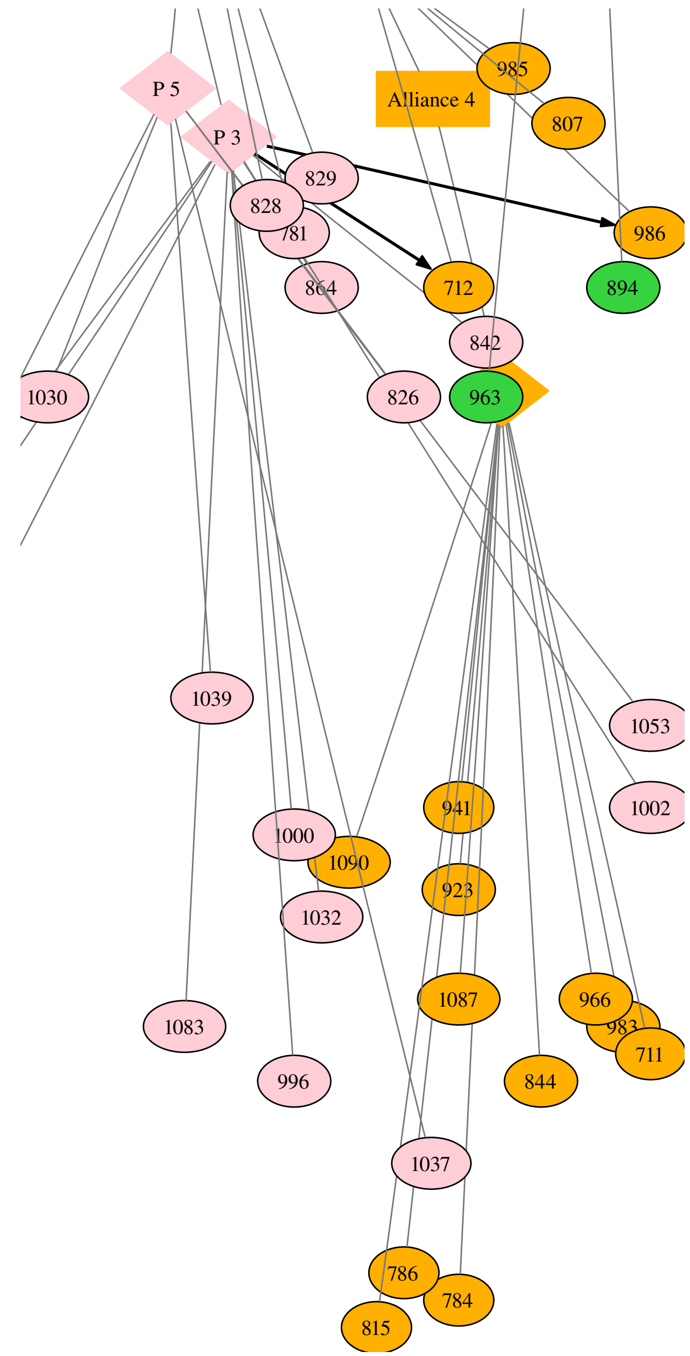


CPT-Rules

$$\underbrace{b_1, \ldots b_n}_{\text{cause}} \rightarrow \underbrace{h_1 : p_1 \vee \ldots \vee h_m : p_m}_{\text{effect}}$$

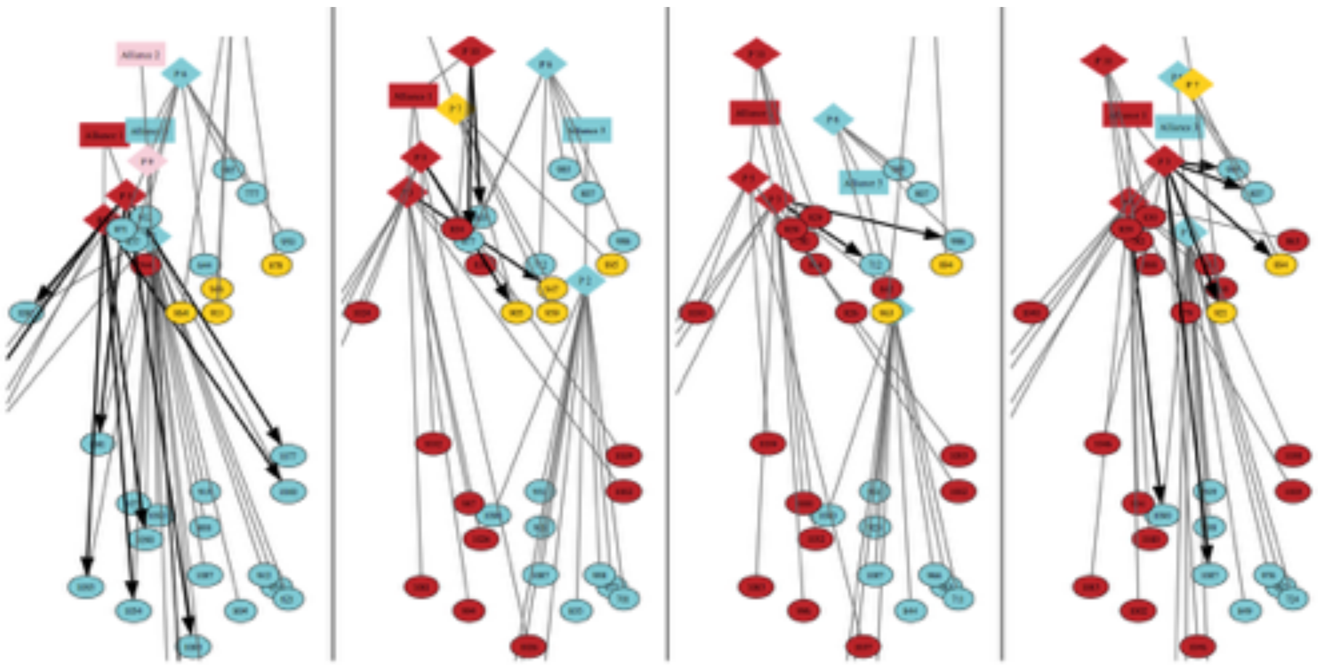
$$city(C, Owner), city(C2, Attacker), close(C, C2) \rightarrow$$
$$conquest(Attacker, C2) \ : \ p \ \vee \ nil \ : \ (1-p)$$

conquer a city which is close
P(conquest(),Time+5) ?
learn parameters



Thon et al. MLJ 11

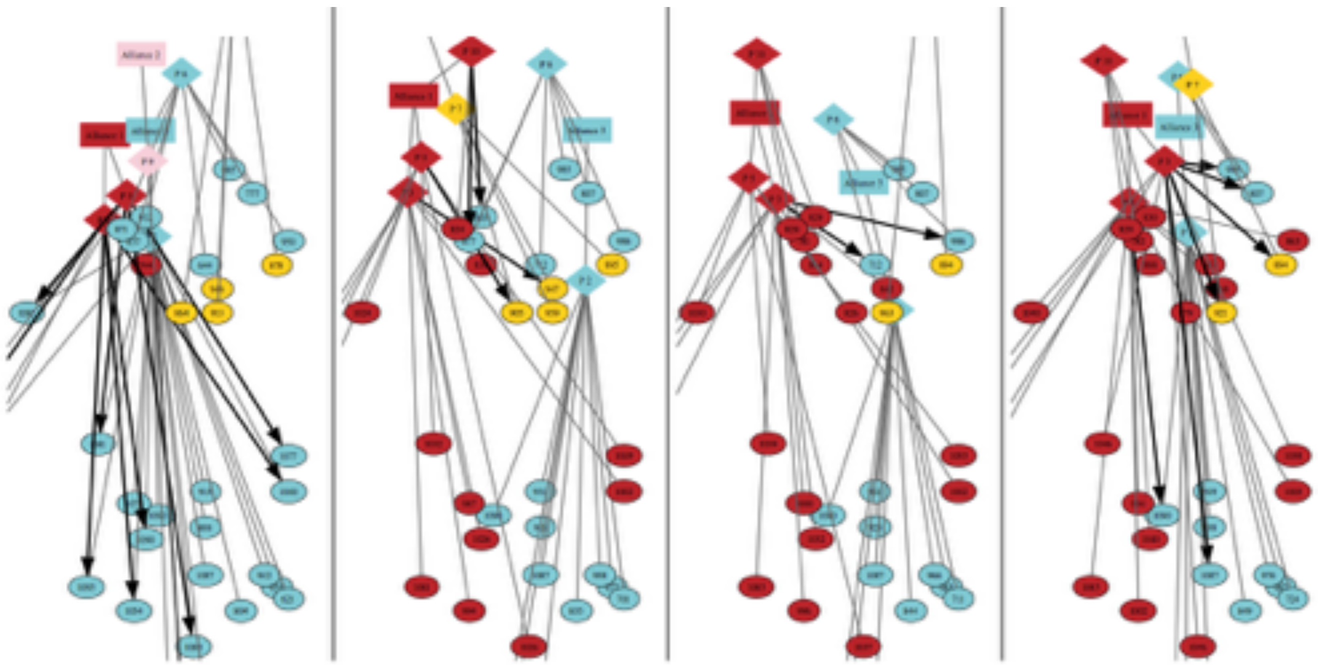
Causal Probabilistic Time-Logic (CPT-L)



how does the world change over time?

[Thon et al, MLJ 11]

Causal Probabilistic Time-Logic (CPT-L)



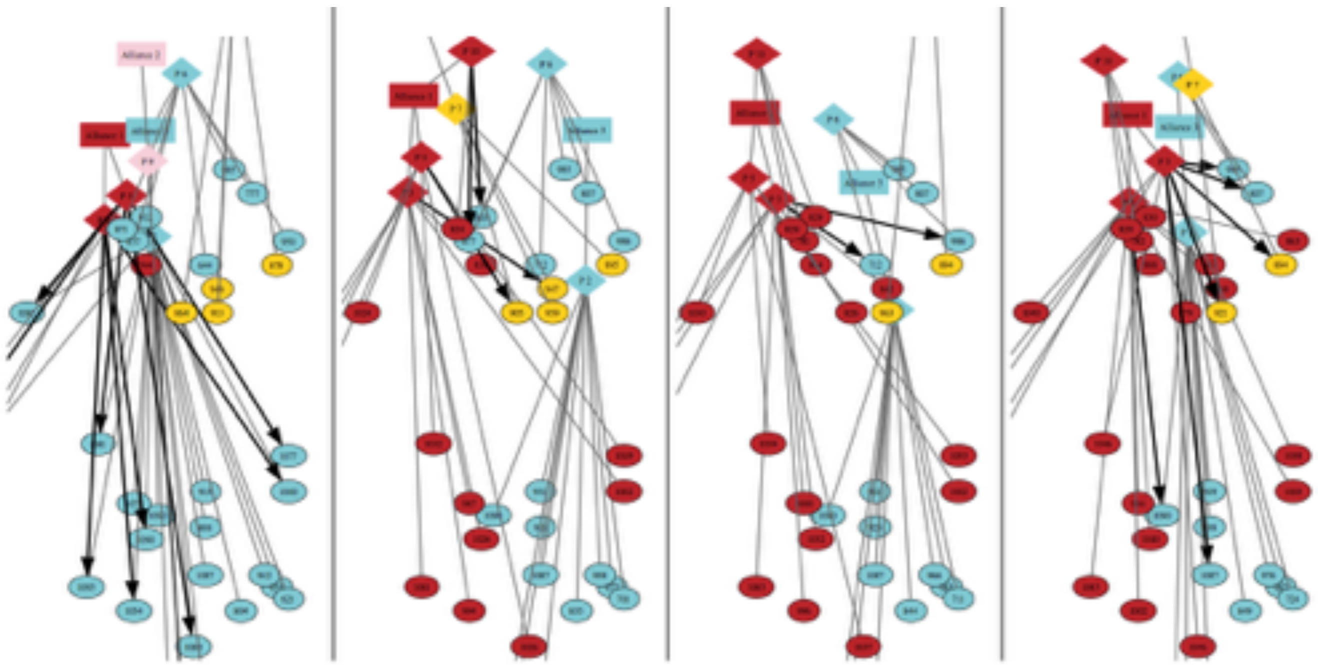
how does the world change over time?

```
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?

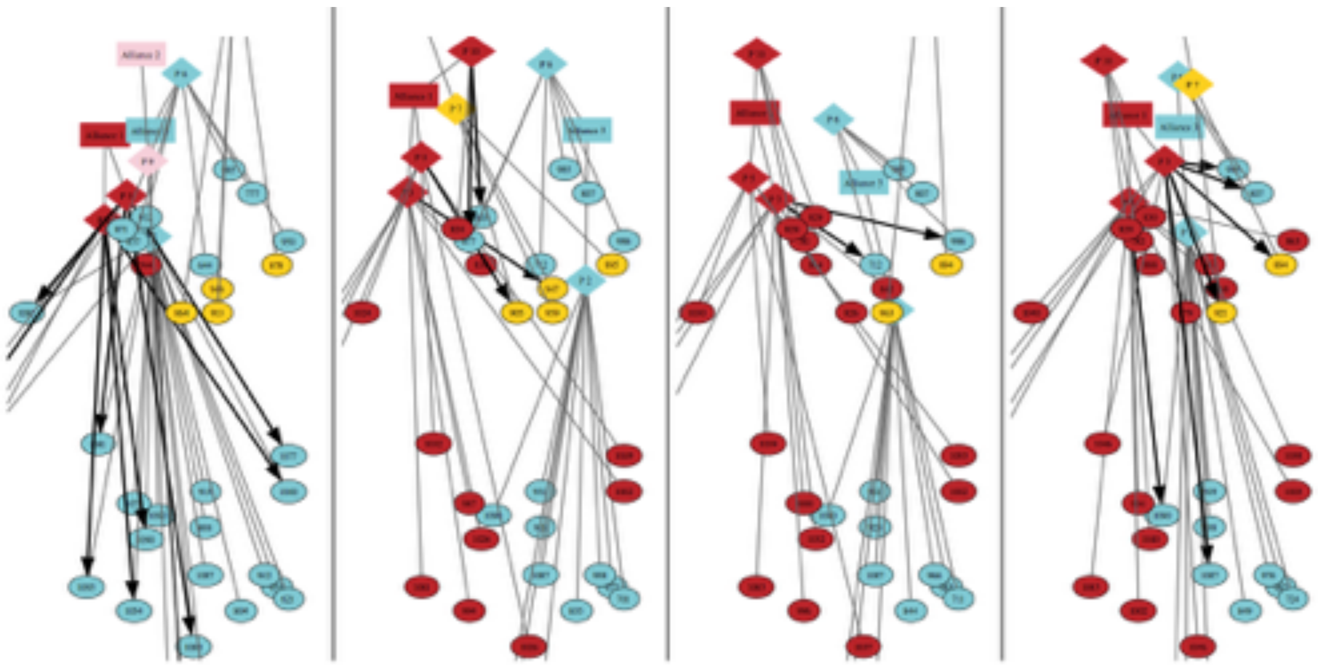
one of the **effects** holds at time T+1

```
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?

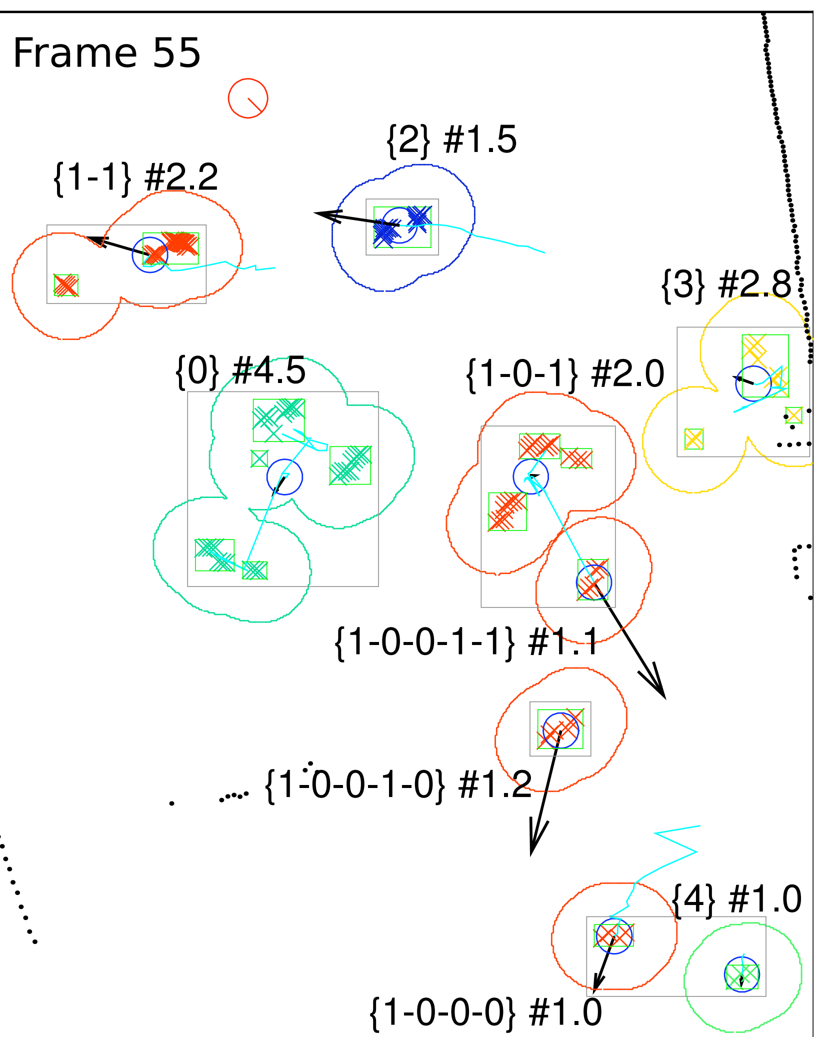
one of the **effects** holds at time T+1

```
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]

Analyzing Video Data



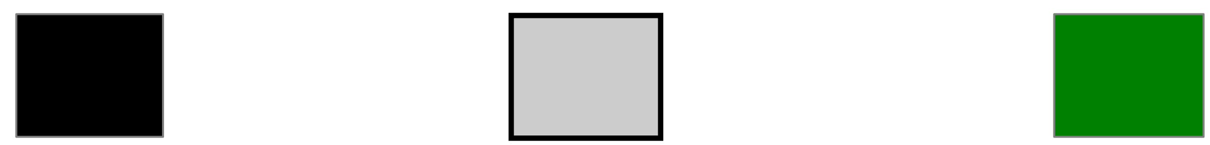
- Track people or objects over time? Even if temporarily hidden?

[Skarlatidis et al, TPLP 14; Nitti et al, IROS 13, ICRA 14]

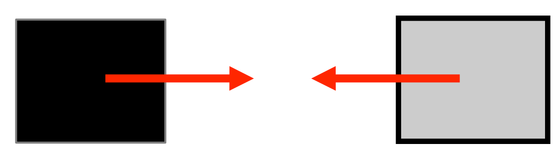
- Recognize activities?
- Infer object properties?

Magnetic scenario

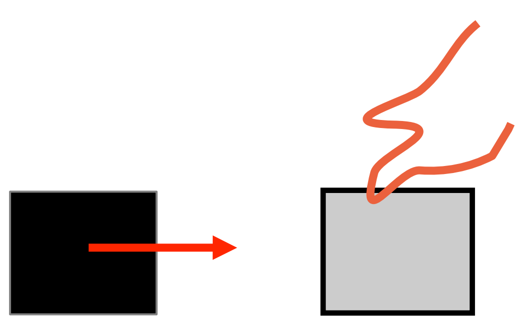
- 3 object types: magnetic, ferromagnetic, nonmagnetic

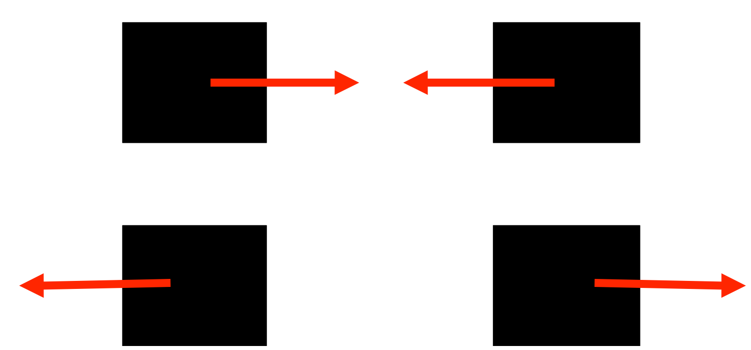


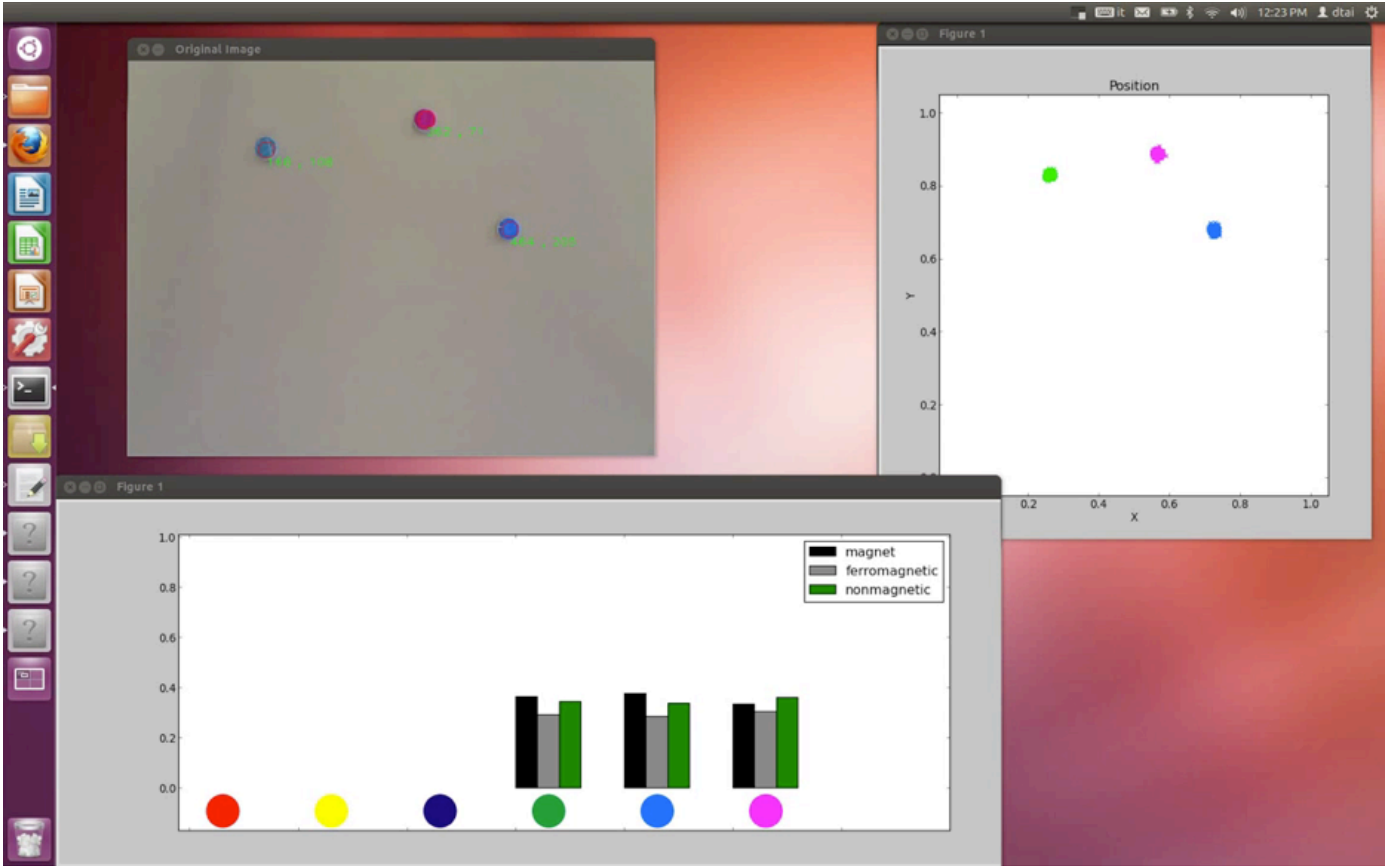
- Nonmagnetic objects do not interact
- A magnet and a ferromagnetic object attract each other



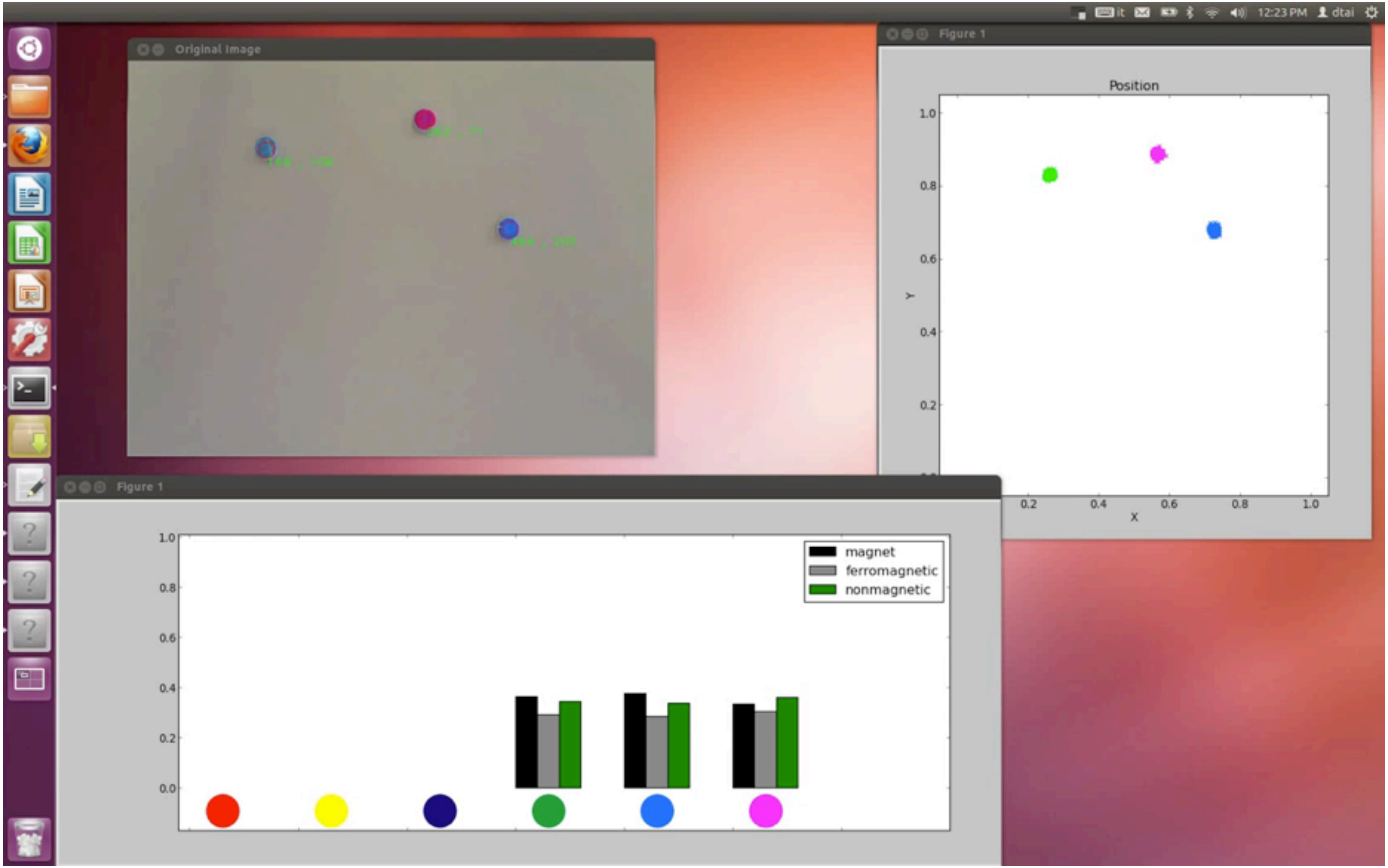
- Magnetic force that depends on the distance
- If an object is held magnetic force is compensated.







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
magnet
ferromagnetic
nonmagnetic
0.0


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
12:23 PM
dtai

Magnetic scenario

- 3 object types: magnetic, ferromagnetic, nonmagnetic

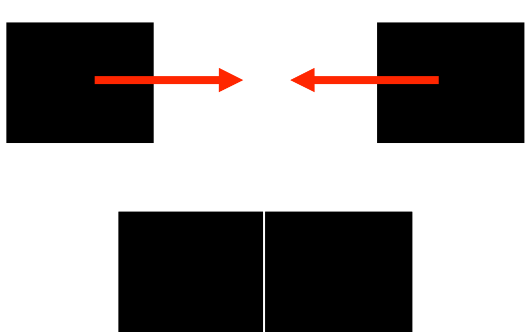
$\text{type}(X)_t \sim \text{finite}([1/3{:}\text{magnet}, 1/3{:}\text{ferromagnetic}, 1/3{:}\text{nonmagnetic}]) \leftarrow \text{object}(X).$

- 2 magnets attract or repulse

$\text{interaction}(A,B)_t \sim \text{finite}([0.5{:}\text{attraction}, 0.5{:}\text{repulsion}]) \leftarrow \text{object}(A), \text{object}(B), A<B, \text{type}(A)_t = \text{magnet}, \text{type}(B)_t = \text{magnet}.$

- Next position after attraction

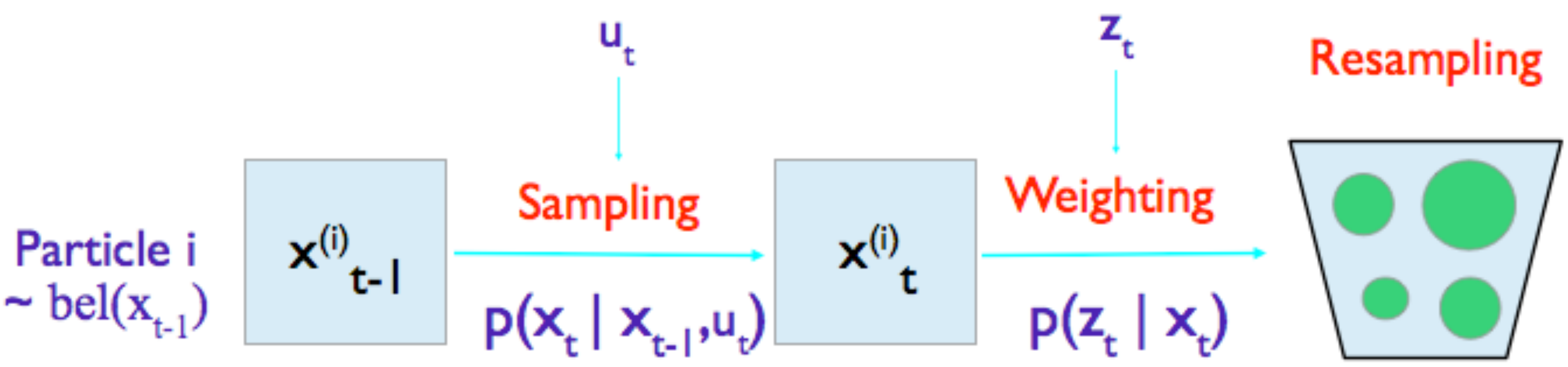
$\text{pos}(A)_{t+1} \sim \text{gaussian}(\text{middlepoint}(A,B)_t, \text{Cov}) \leftarrow \text{near}(A,B)_t, \text{not}(\text{held}(A)), \text{not}(\text{held}(B)), \text{interaction}(A,B)_t = \text{attr}, c/\text{dist}(A,B)_t^2 > \text{friction}(A)_t.$



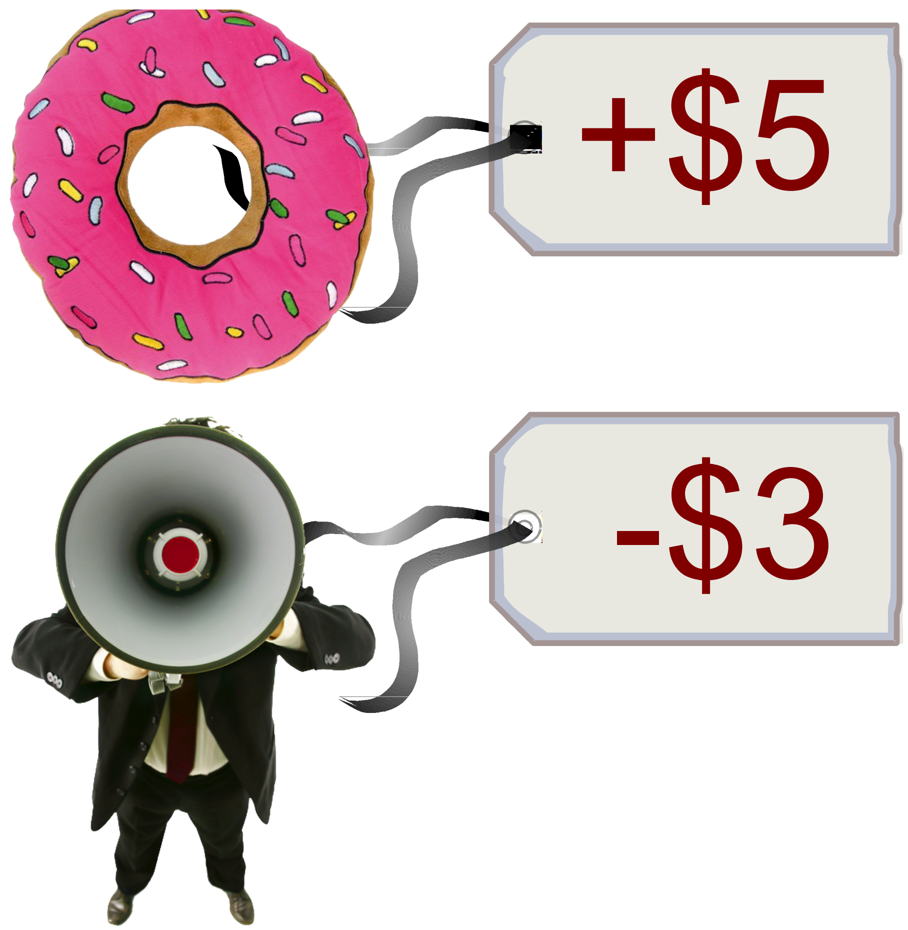
$\text{pos}(A)_{t+1} \sim \text{gaussian}(\text{pos}(A)_t, \text{Cov}) \leftarrow \text{not}(\ \text{attraction}(A,B)\ ).$

Particle Filter
(Sequential Monte Carlo)

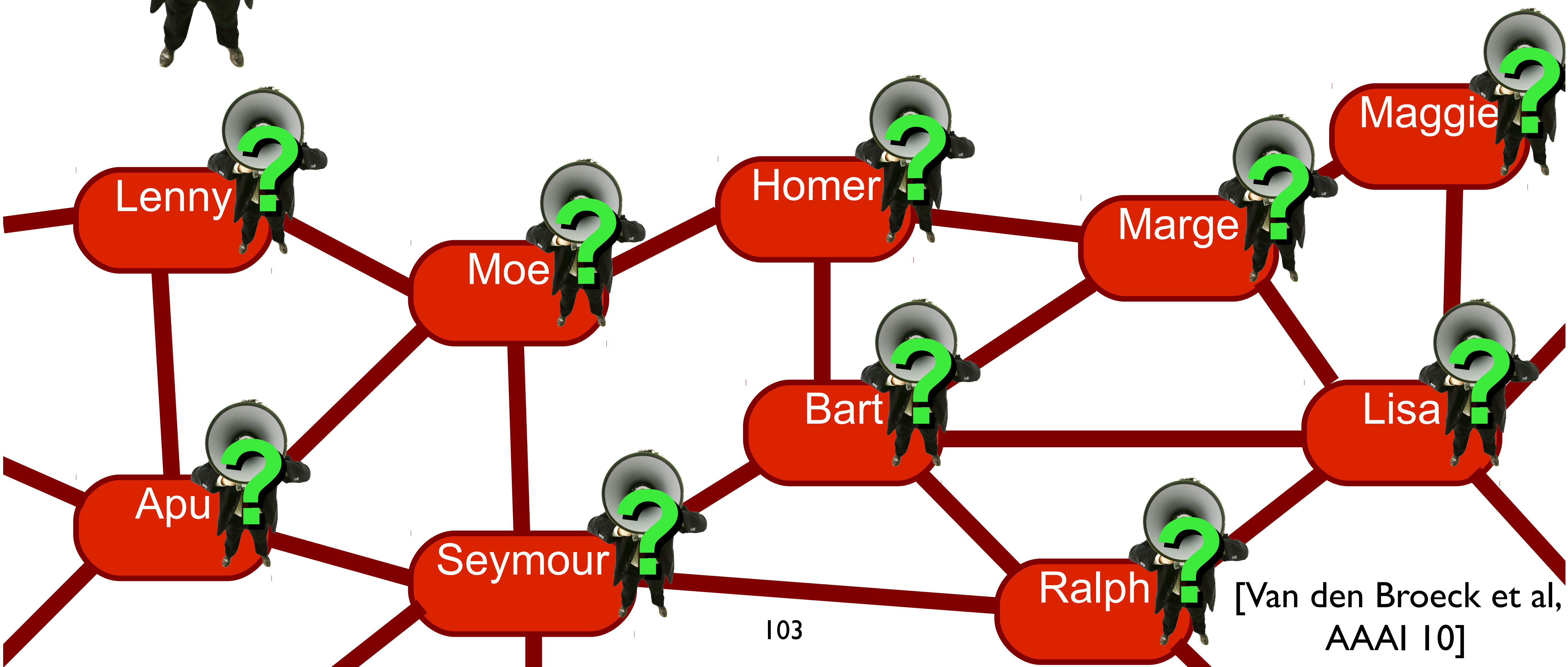
- Based on sampling → approximate inference
- Particles (samples) to represent bel(x_t)



Viral Marketing

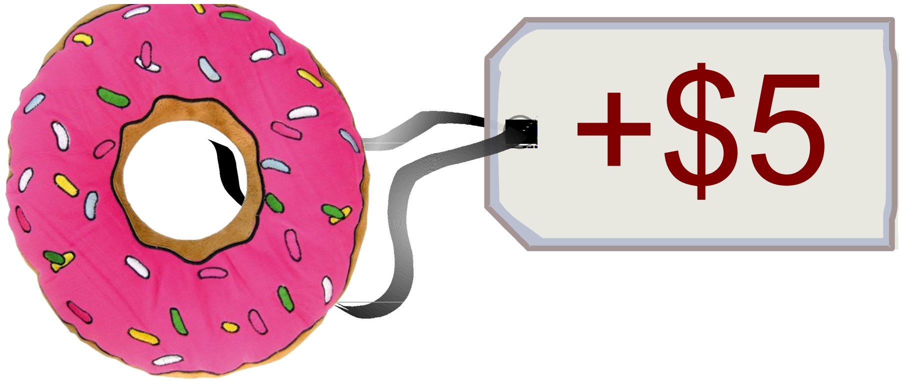


Which advertising strategy maximizes expected profit?

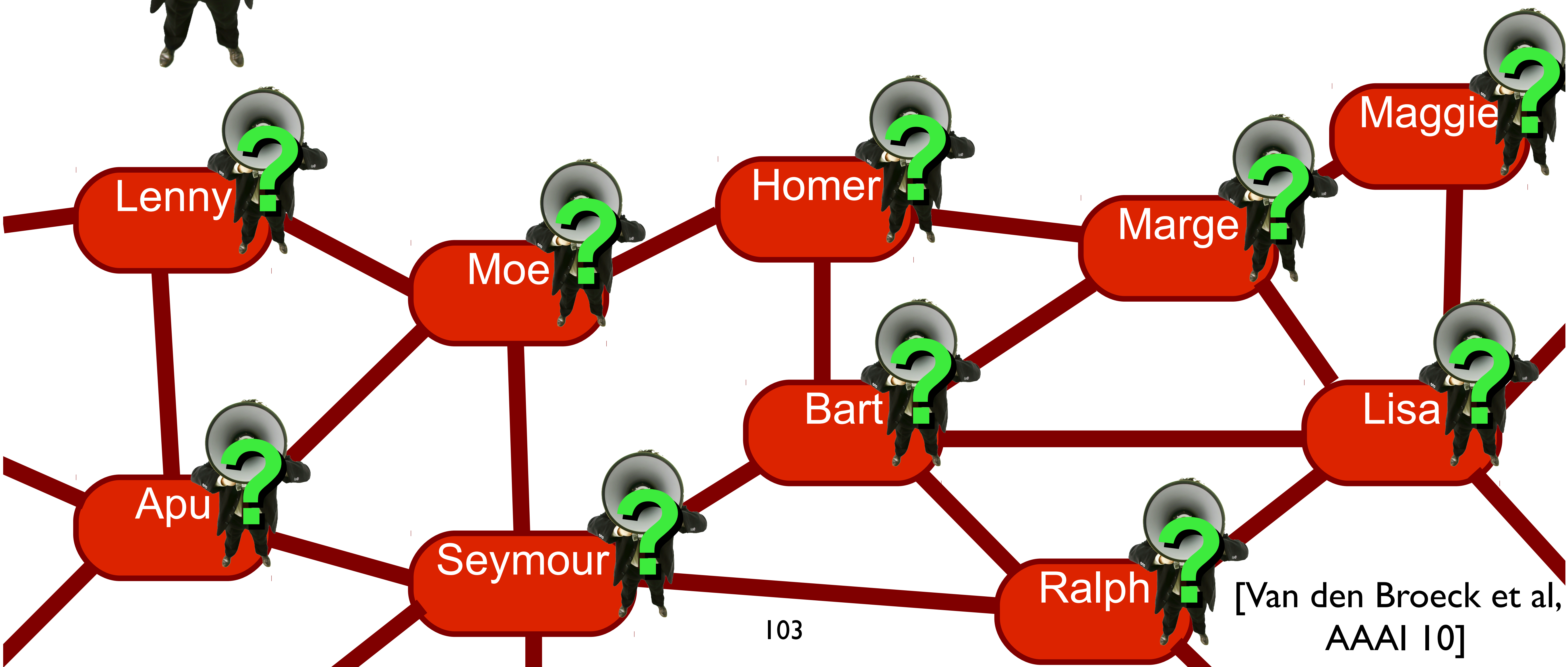


[Van den Broeck et al, AAAI 10]

Viral Marketing

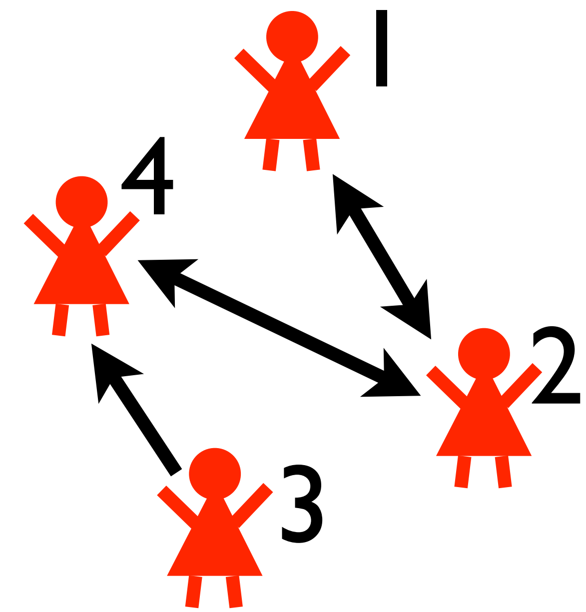


decide truth values of some atoms



[Van den Broeck et al, AAAI 10]

DTProbLog

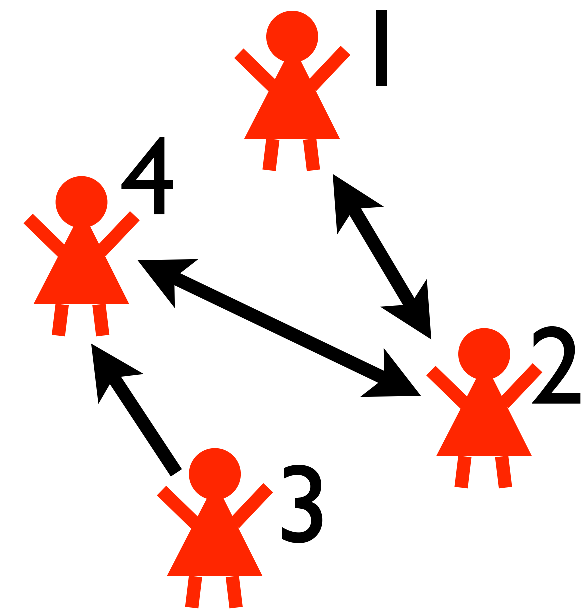



```
person(1).
person(2).
person(3).
person(4).

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

DTProbLog

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

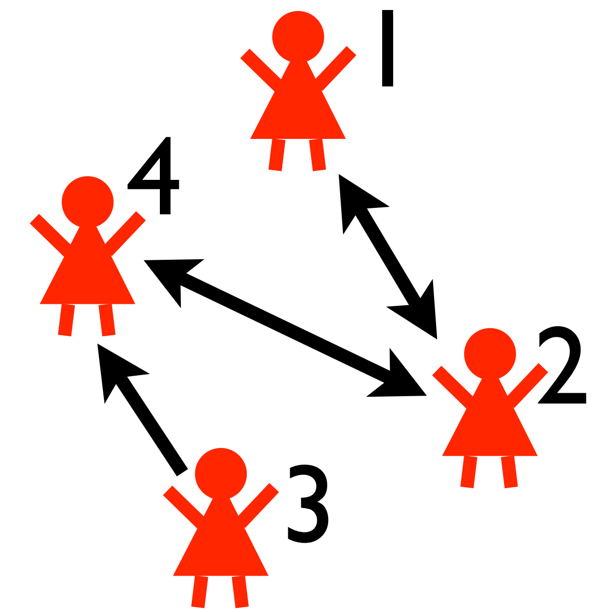
decision fact: true or false?




```
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).
```



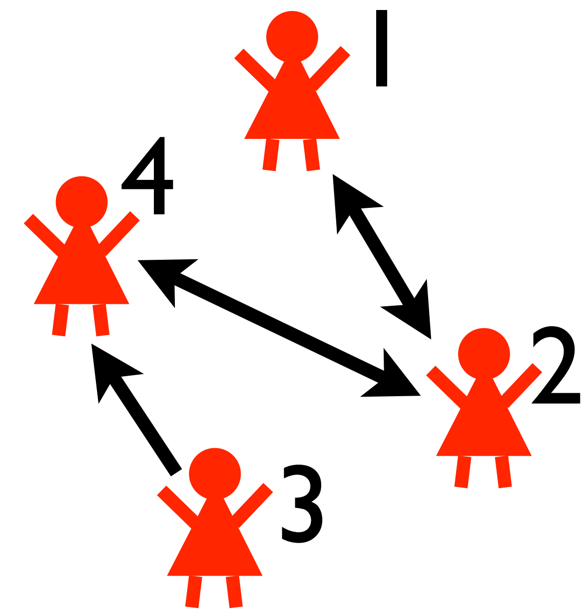

```
person(1).
person(2).
person(3).
person(4).

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

probabilistic facts
+ logical rules

DTProbLog

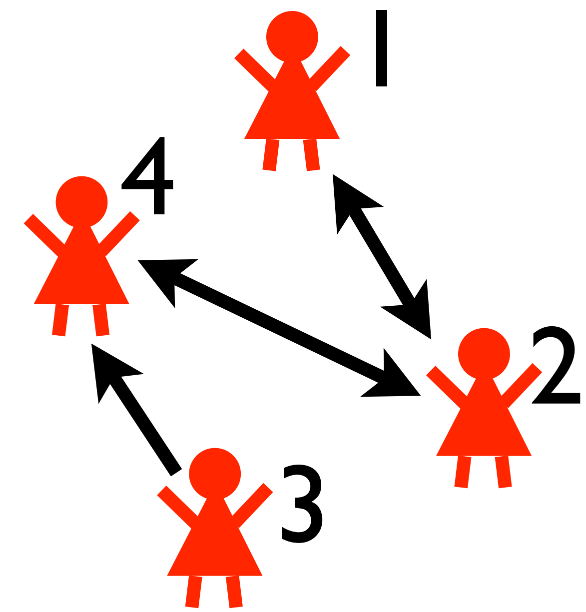
```
? :: marketed(P) :- person(P).

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).
```

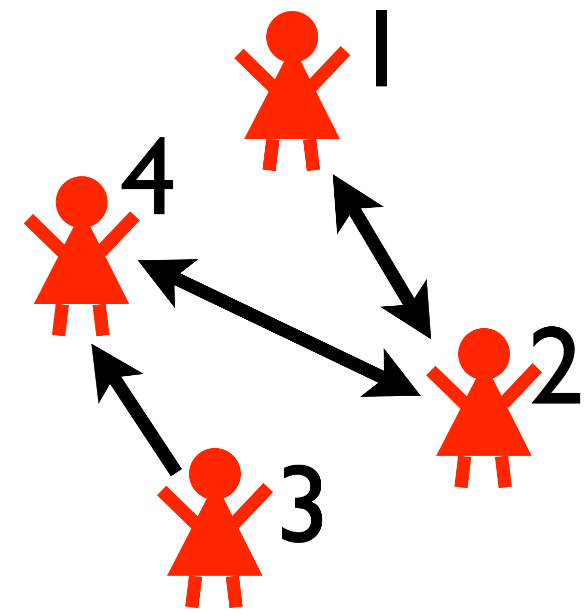
DTProbLog




```
? :: 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).
```

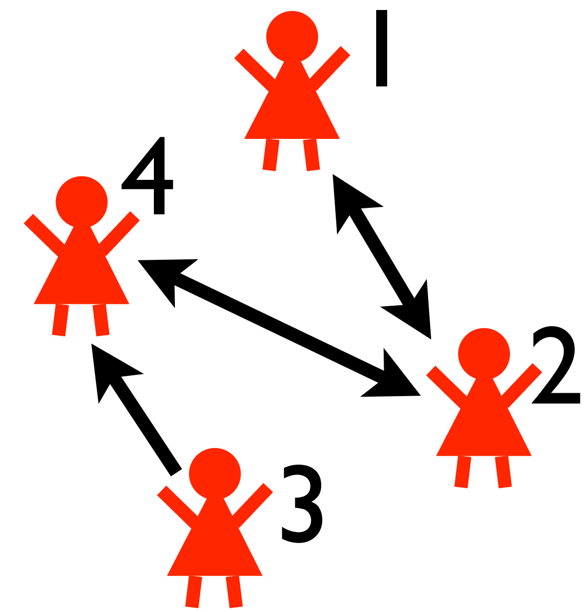
DTProbLog




```
? :: 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

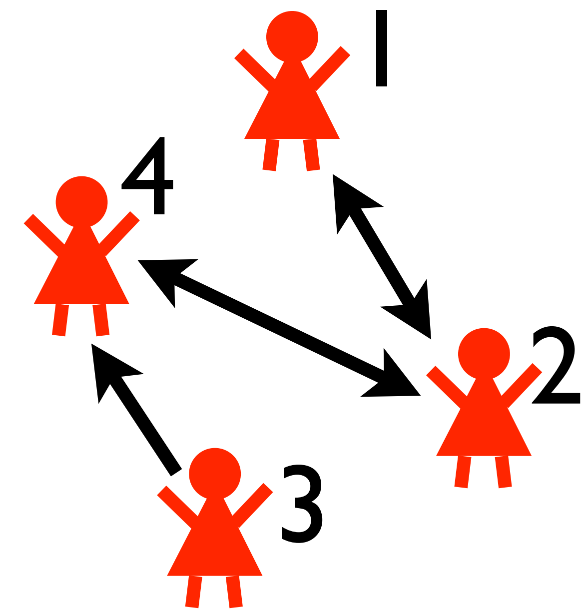



```
? :: marketed(P) :- person(P).

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

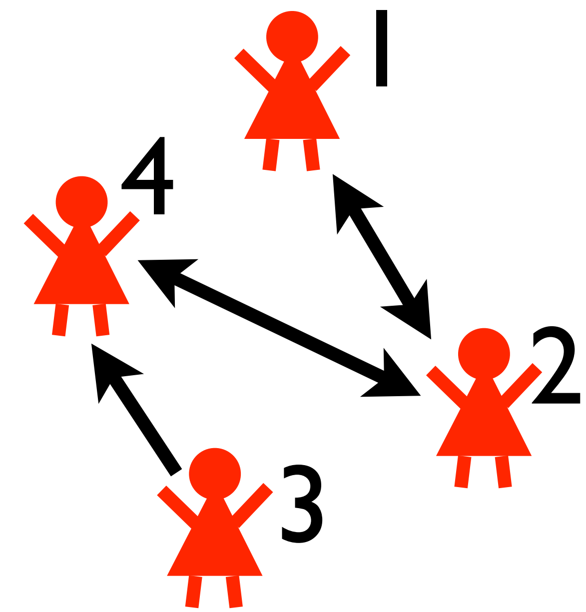



```
? :: marketed(P) :- person(P).

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

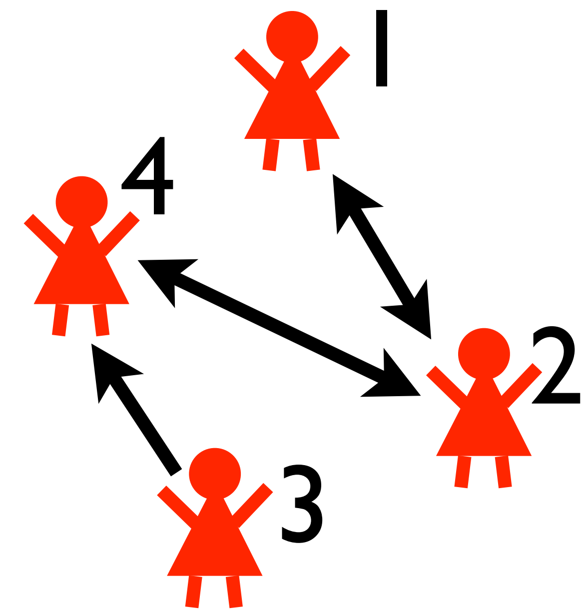



```
? :: marketed(P) :- person(P).

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).
```

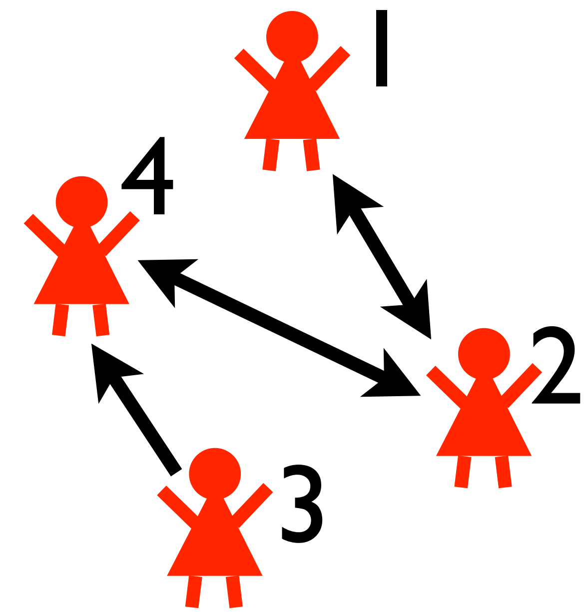
```
person(1).
person(2).
person(3).
person(4).

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).
```

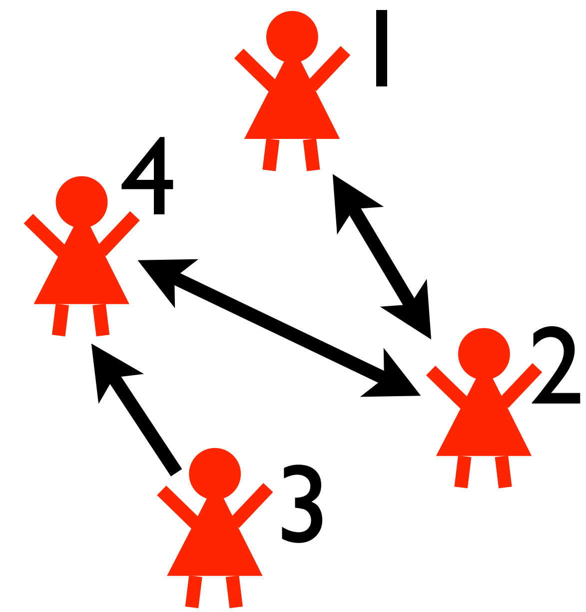
utility = −3 + −3 + 5 + 5 = 4

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

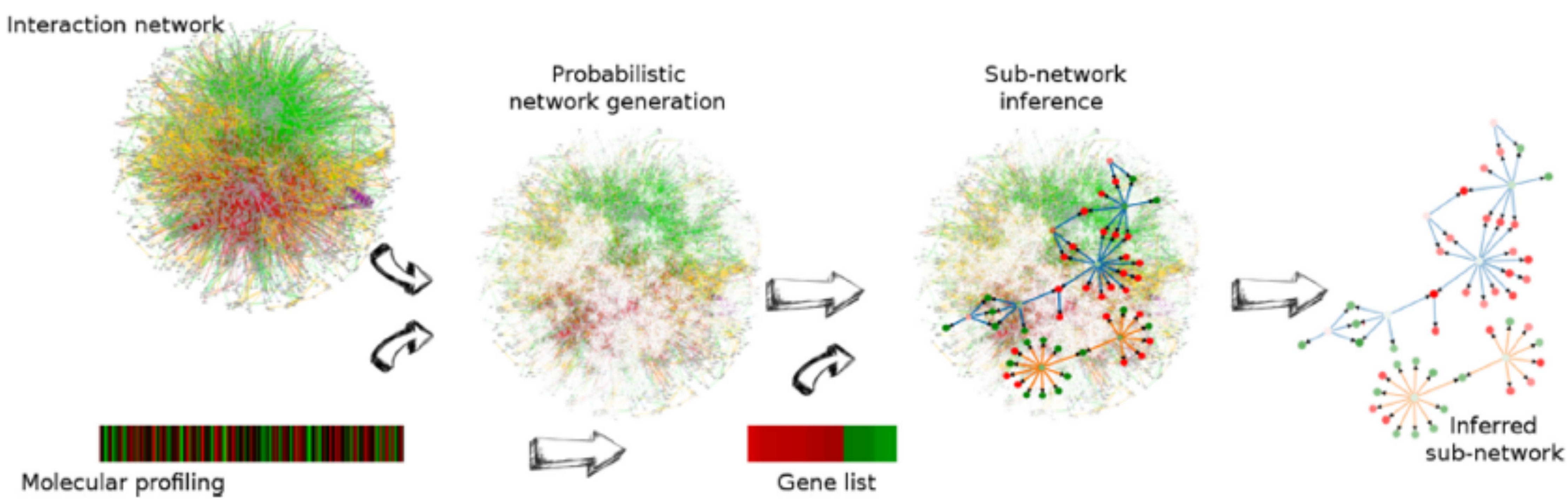
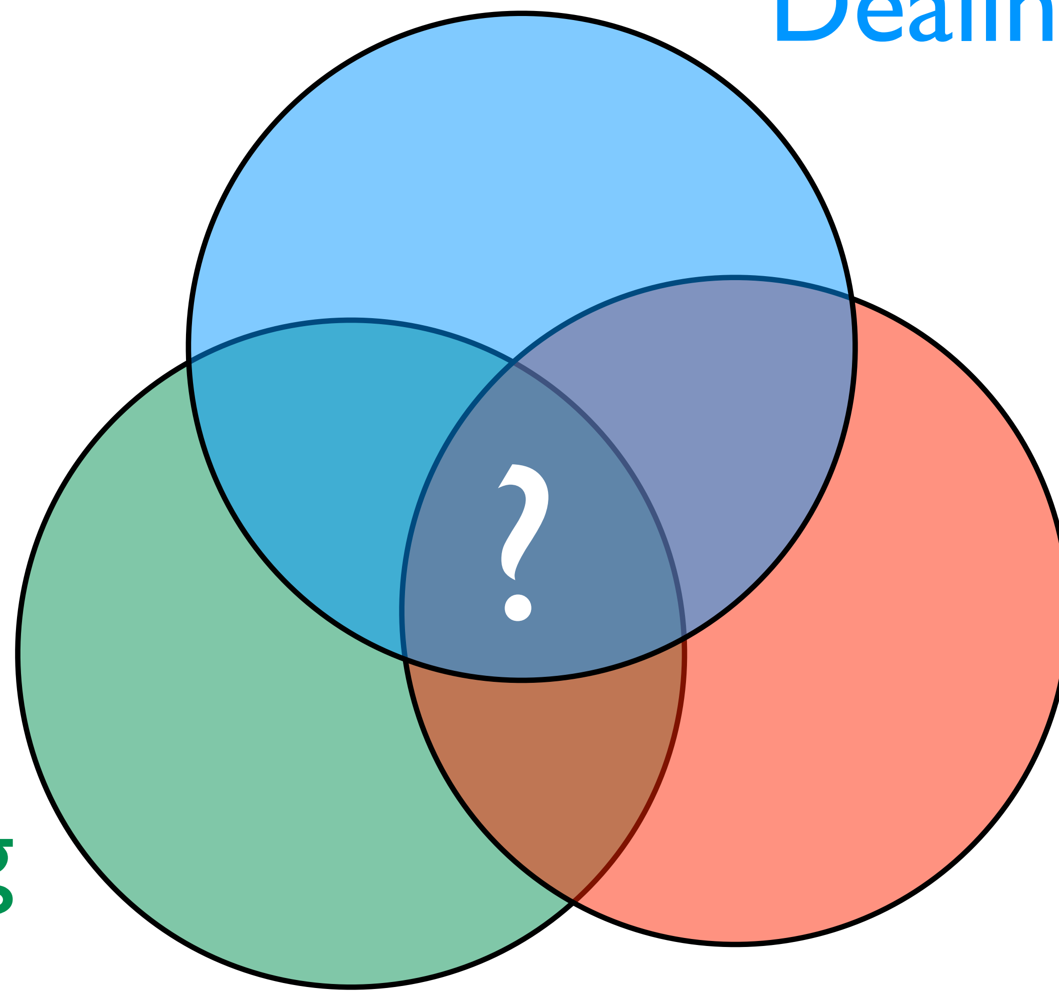


Figure 1. Overview of PheNetic, a web service for network-based interpretation of 'omics' data. The web service uses as input a genome wide interaction network for the organism of interest, a user generated molecular profiling data set and a gene list derived from these data. Interaction networks for a wide variety of organisms are readily available from the web server. Using the uploaded user-generated molecular data the interaction network is converted into a probabilistic network: edges receive a probability proportional to the levels measured for the terminal nodes in the molecular profiling data set. This probabilistic interaction network is used to infer the sub-network that best links the genes from the gene list. The inferred sub-network provides a trade-off between linking as many genes as possible from the gene list and selecting the least number of edges.

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

Reasoning with relational data
- logic
- databases
- programming
- ...



Learning
- parameters
- structure

Statistical relational learning
& Probabilistic programming, ...

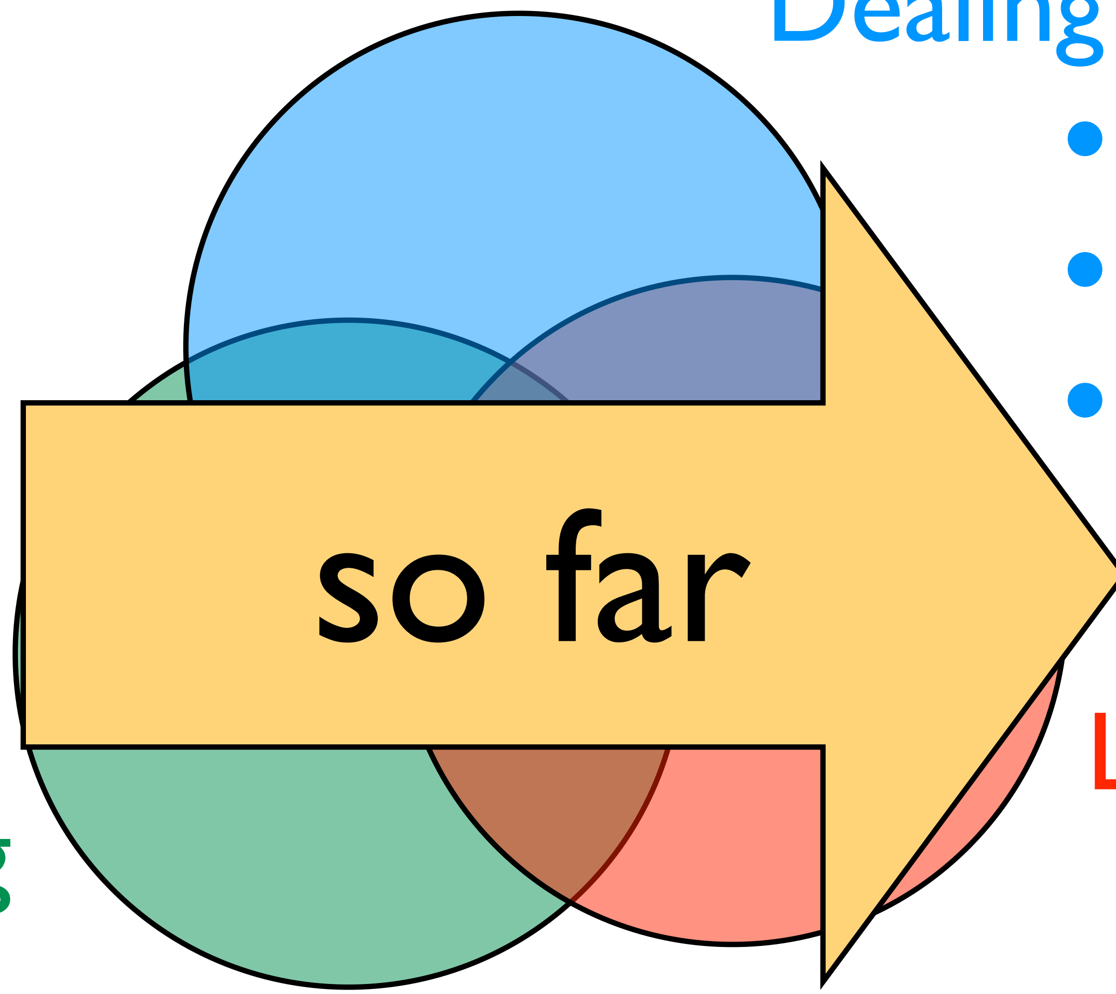
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

- parameters
- structure

Statistical relational learning
& Probabilistic programming, ...

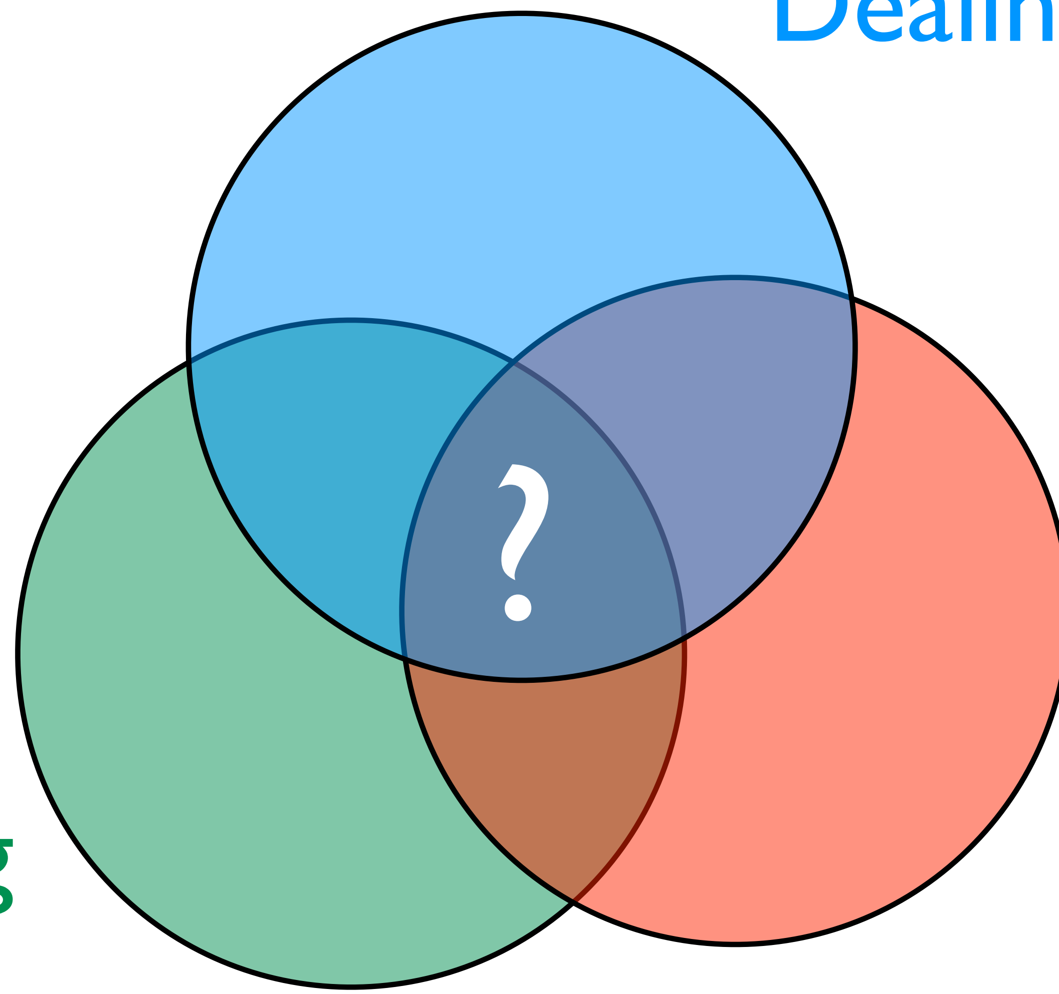
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

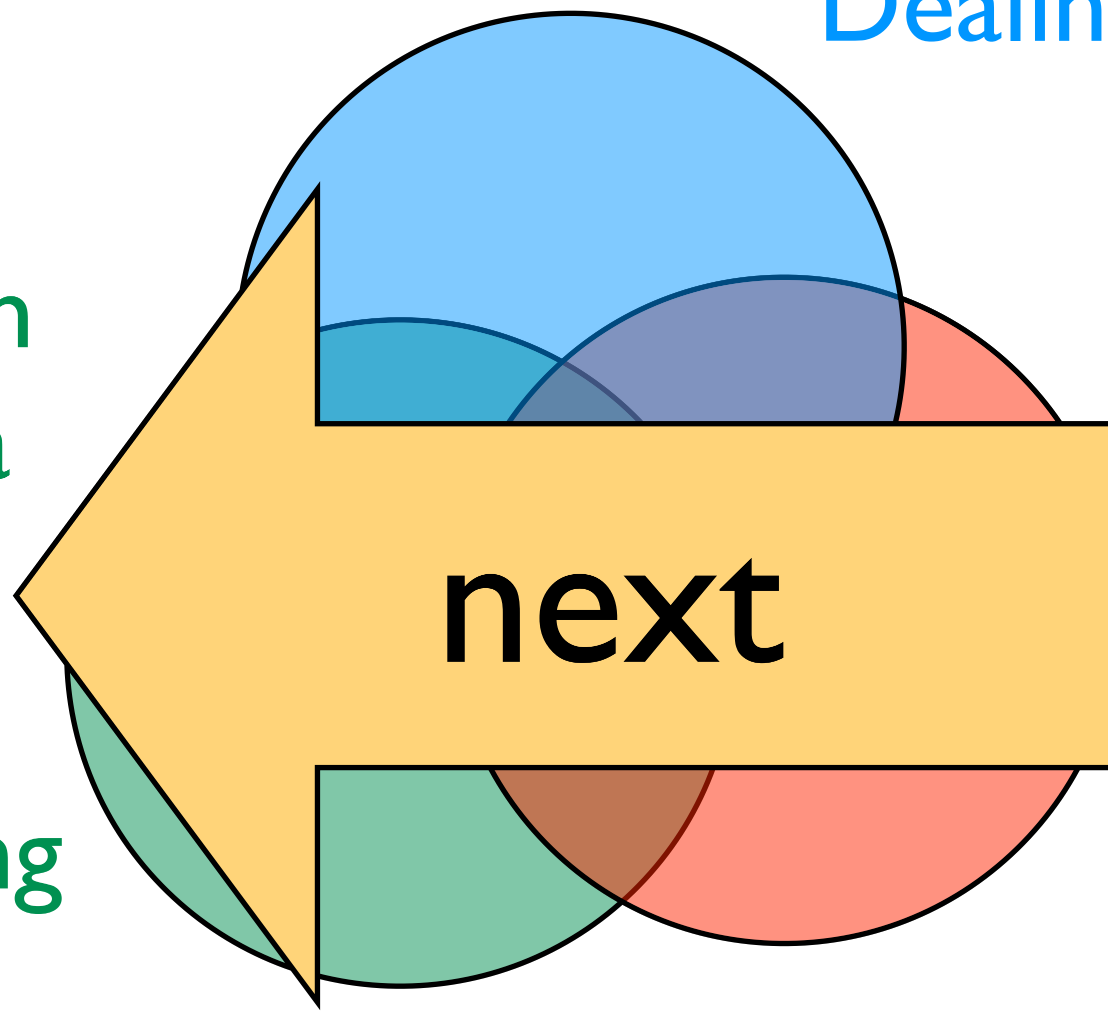
- parameters
- structure

Statistical relational learning
& Probabilistic programming, ...

A key question in AI:

Dealing with uncertainty
- probability theory
- graphical models
- ...

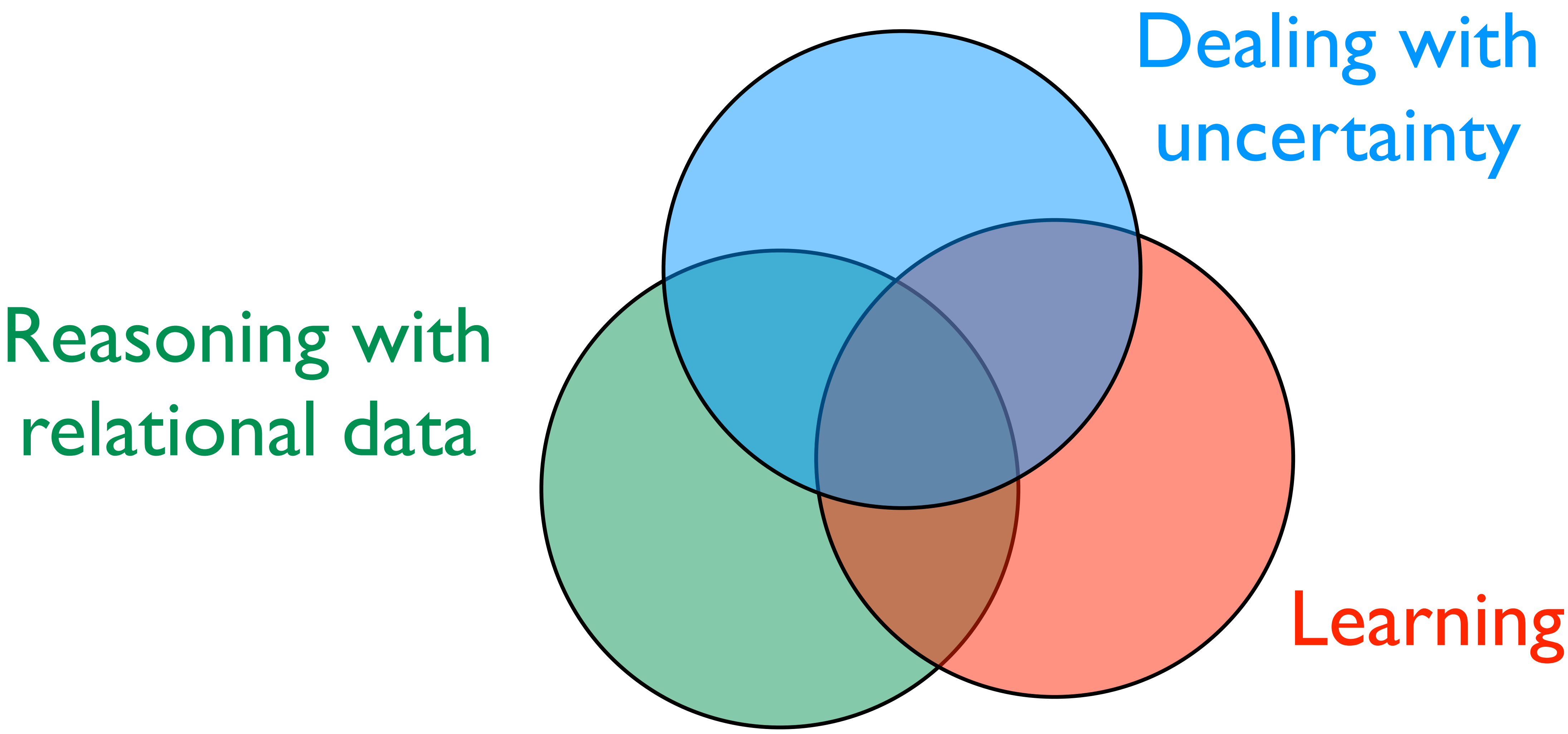
Reasoning with relational data
- logic
- databases
- programming
- ...



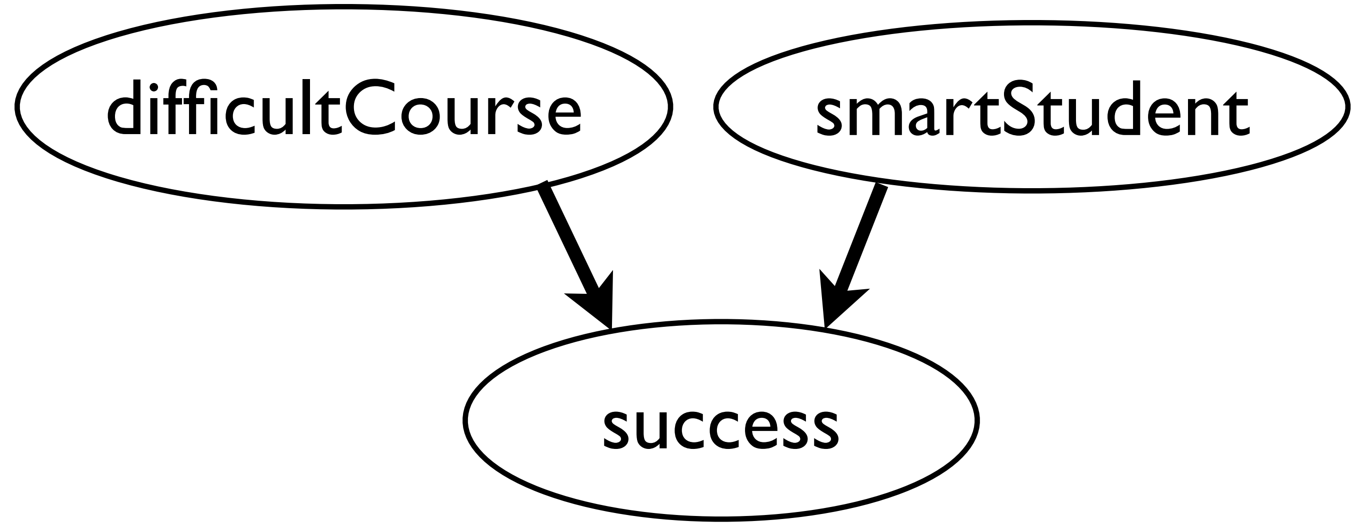
Learning
- parameters
- structure

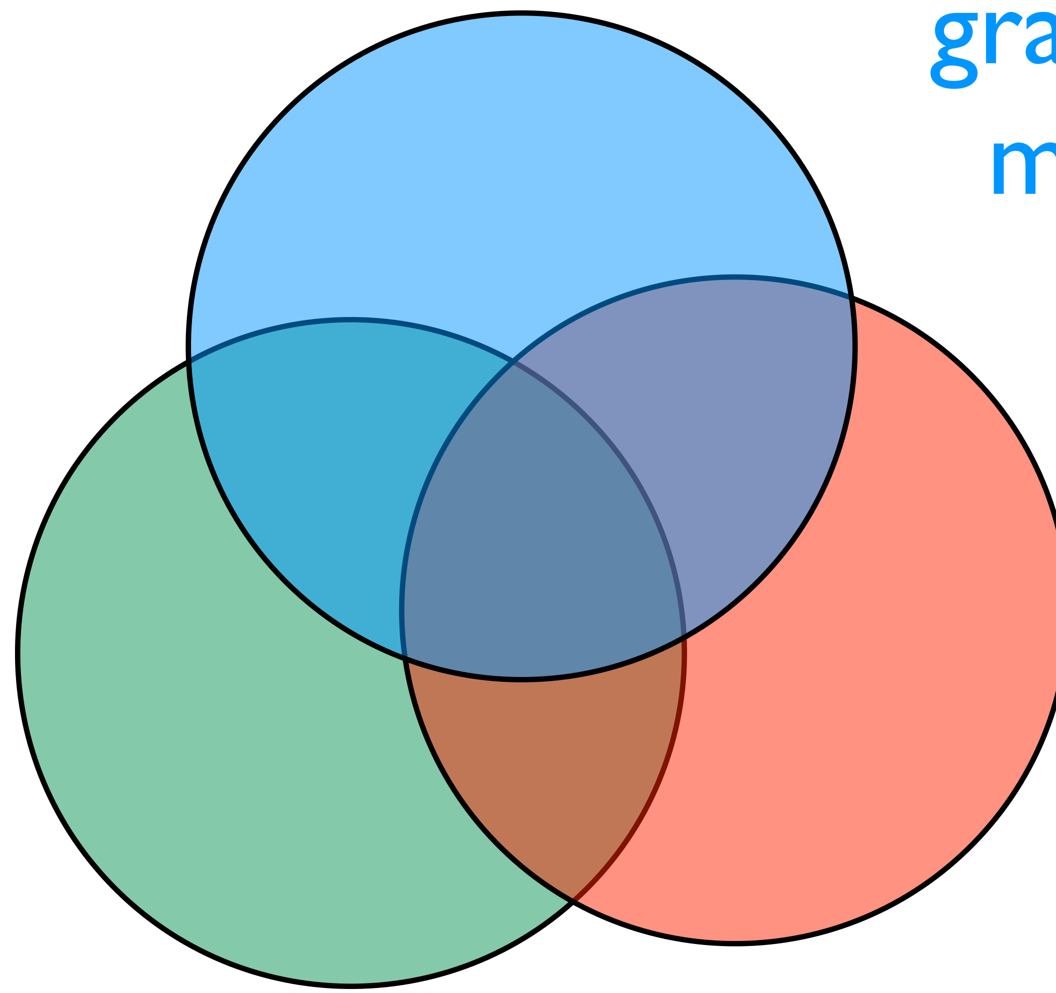
Statistical relational learning
& Probabilistic programming, ...

Lifted graphical models



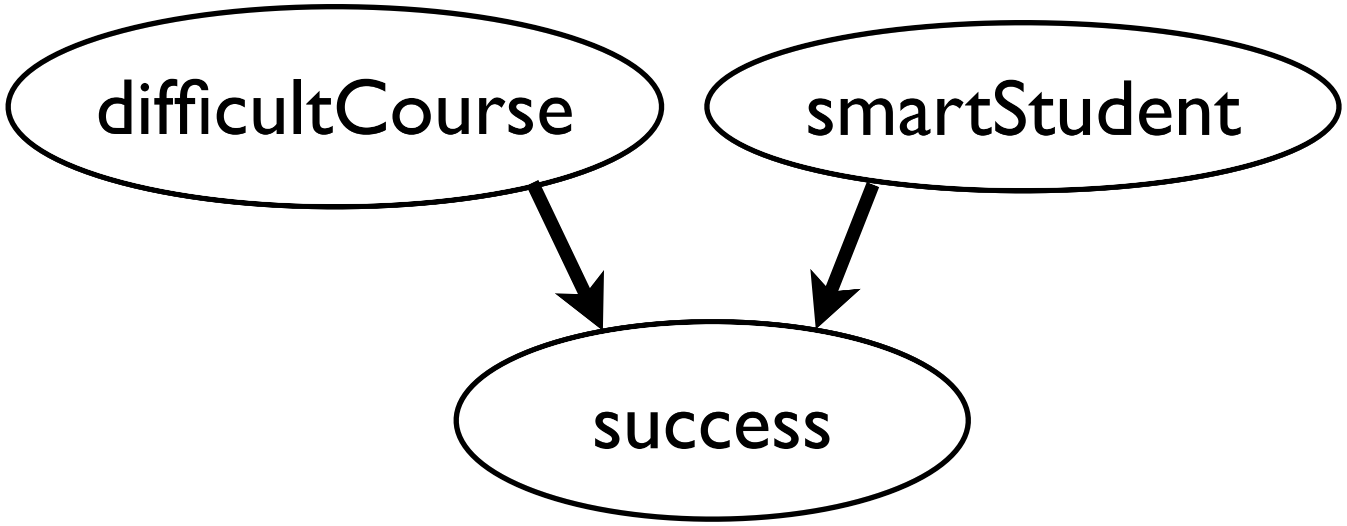
Lifted graphical models

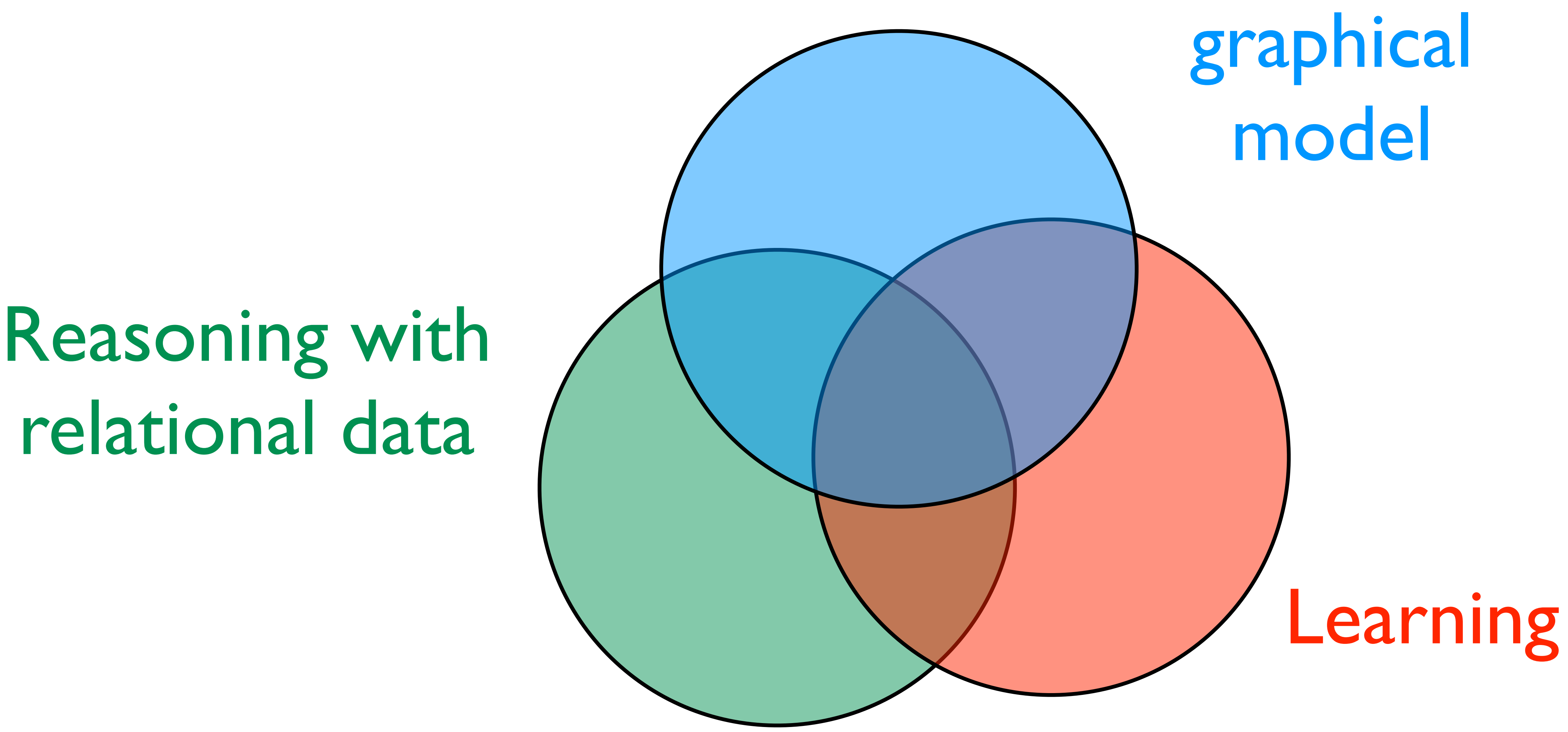




Lifted graphical models

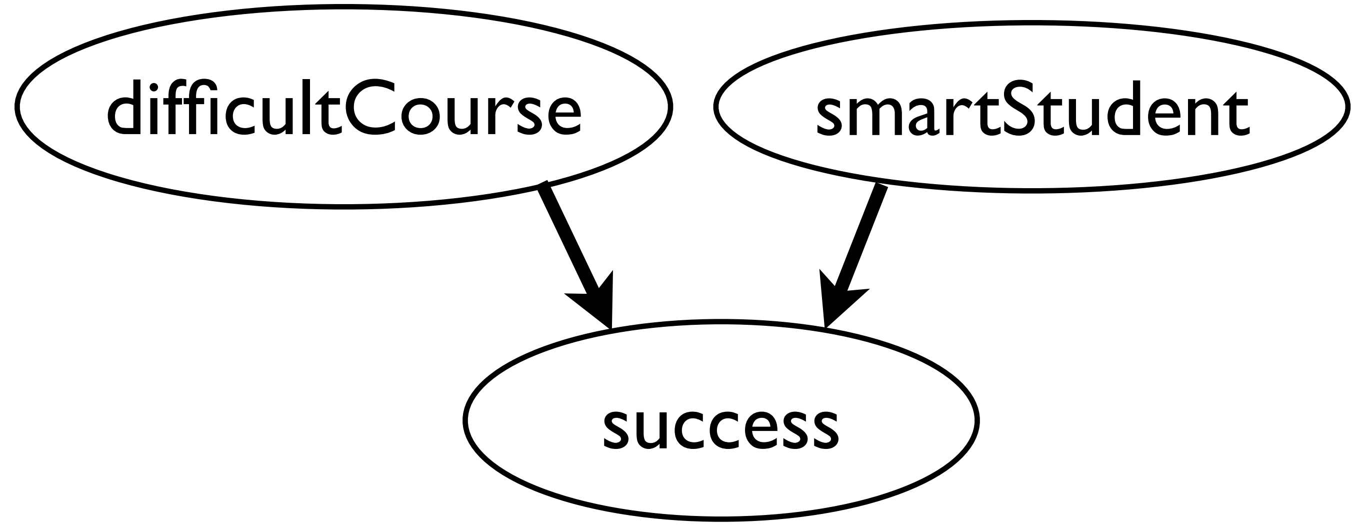
fixed set of random variables

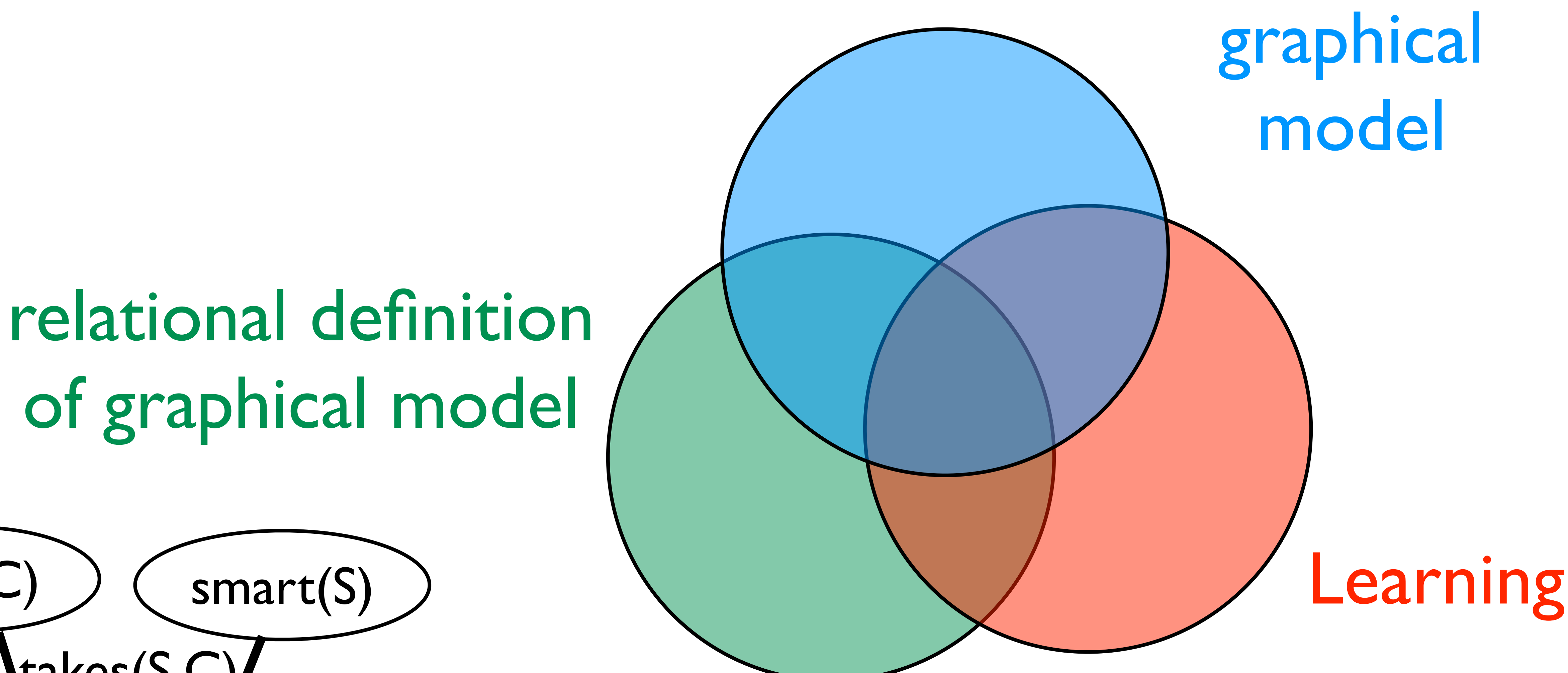


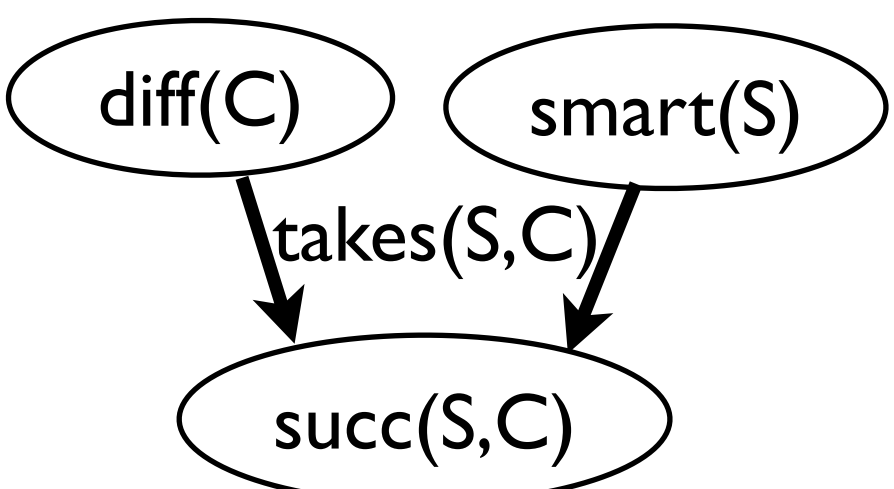


Lifted graphical models

fixed set of random variables

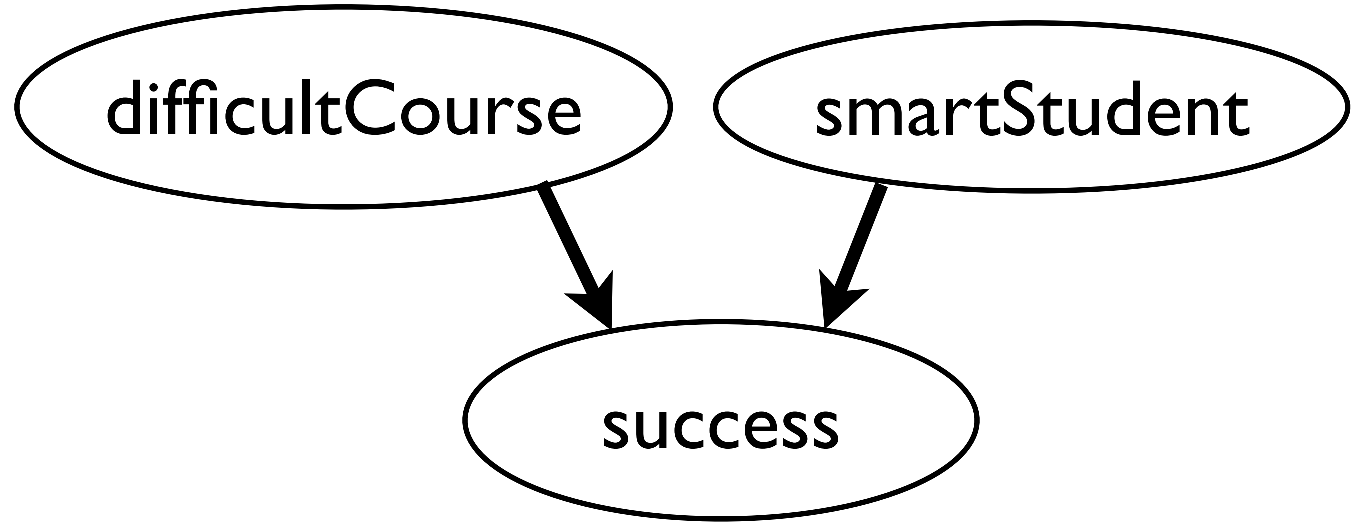




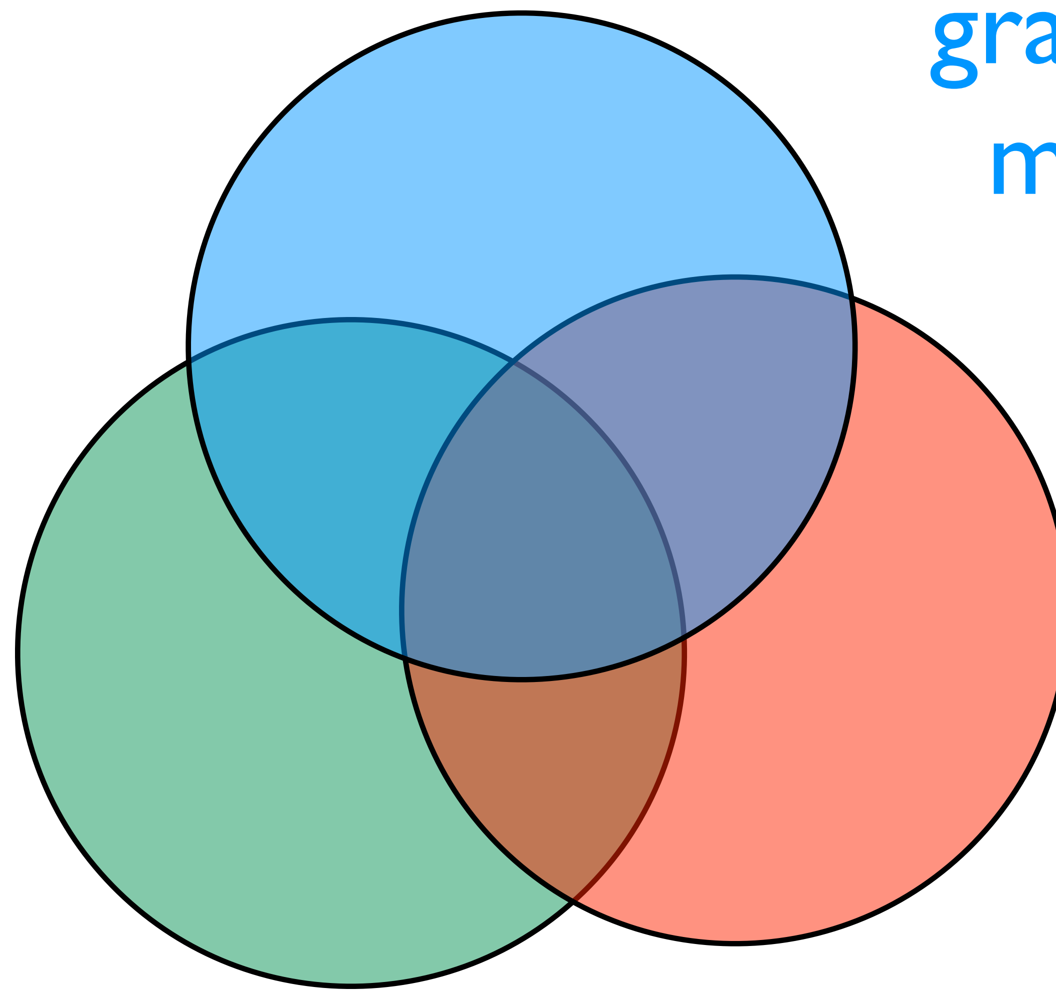


Lifted graphical models

fixed set of random variables

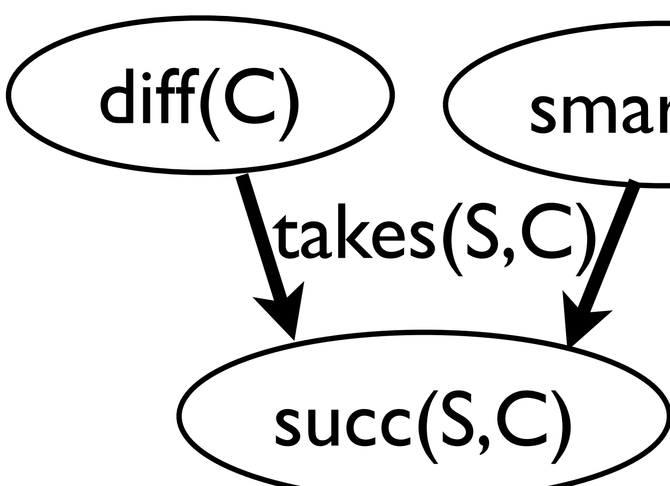


graphical model



relational definition of graphical model

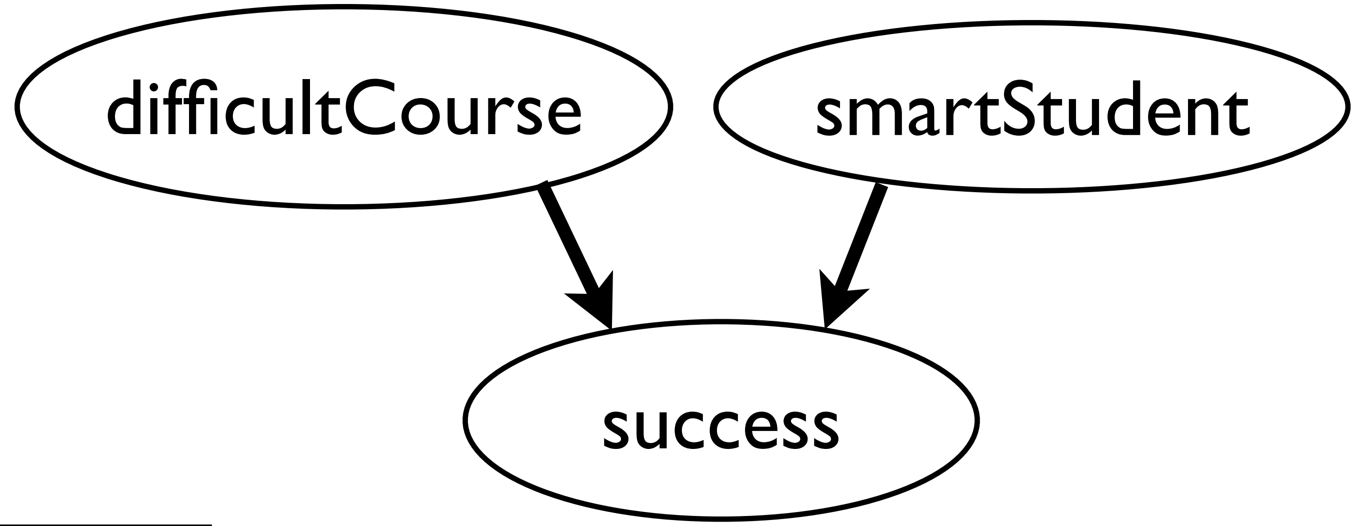
Learning



data-dependent set of random variables

Lifted graphical models

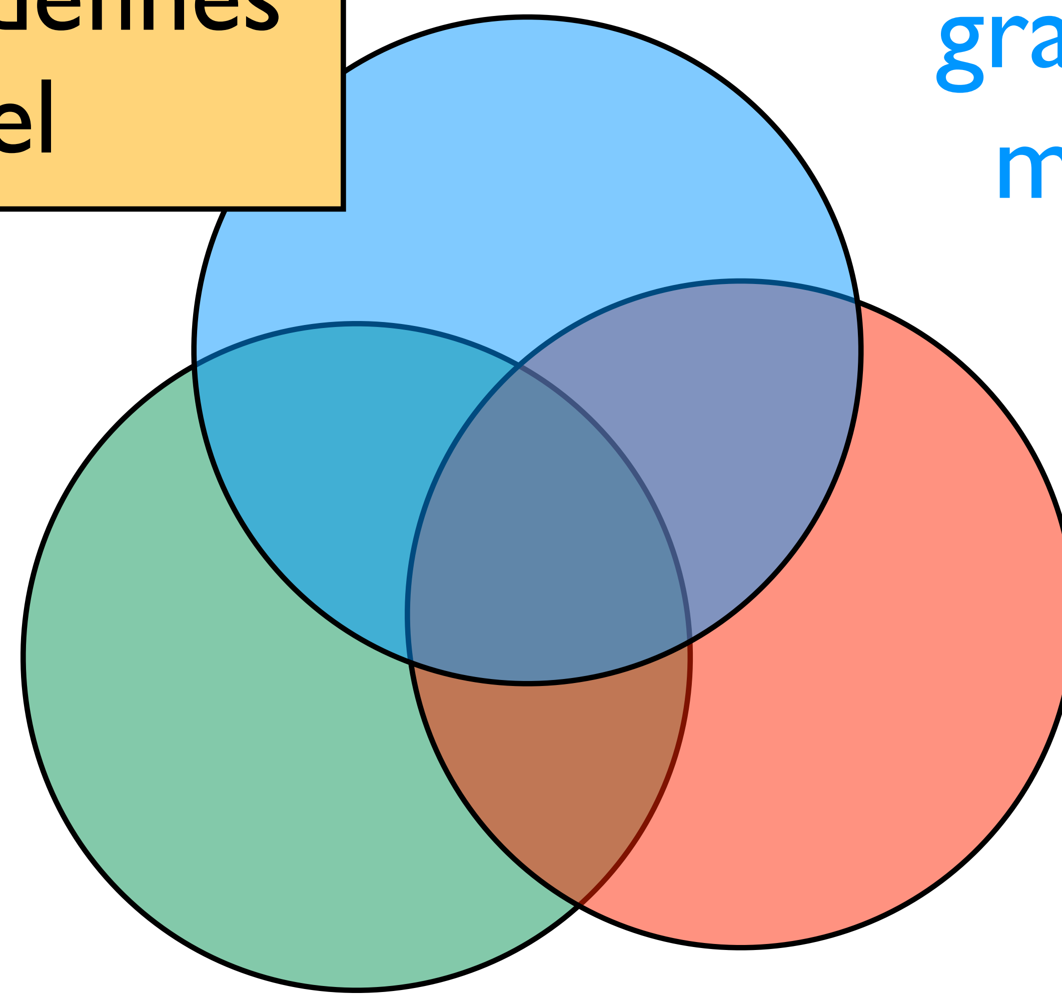
fixed set of random variables



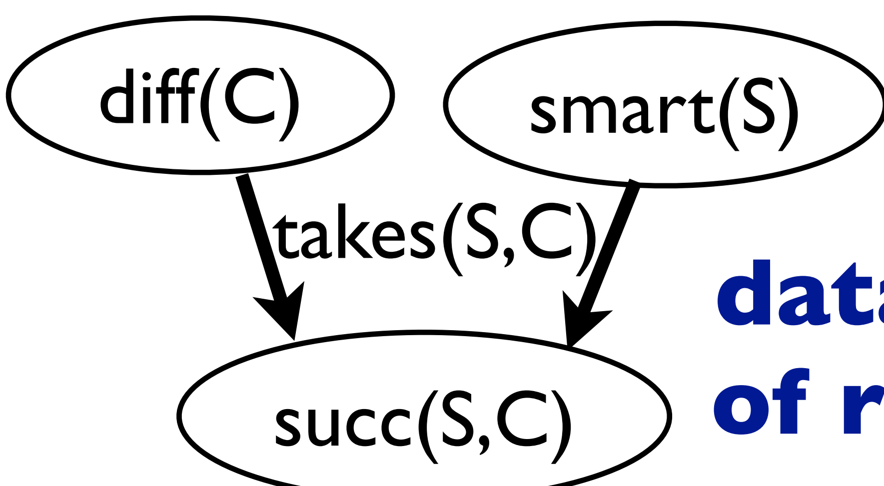
relational language defines graphical model

graphical model

relational definition of graphical model



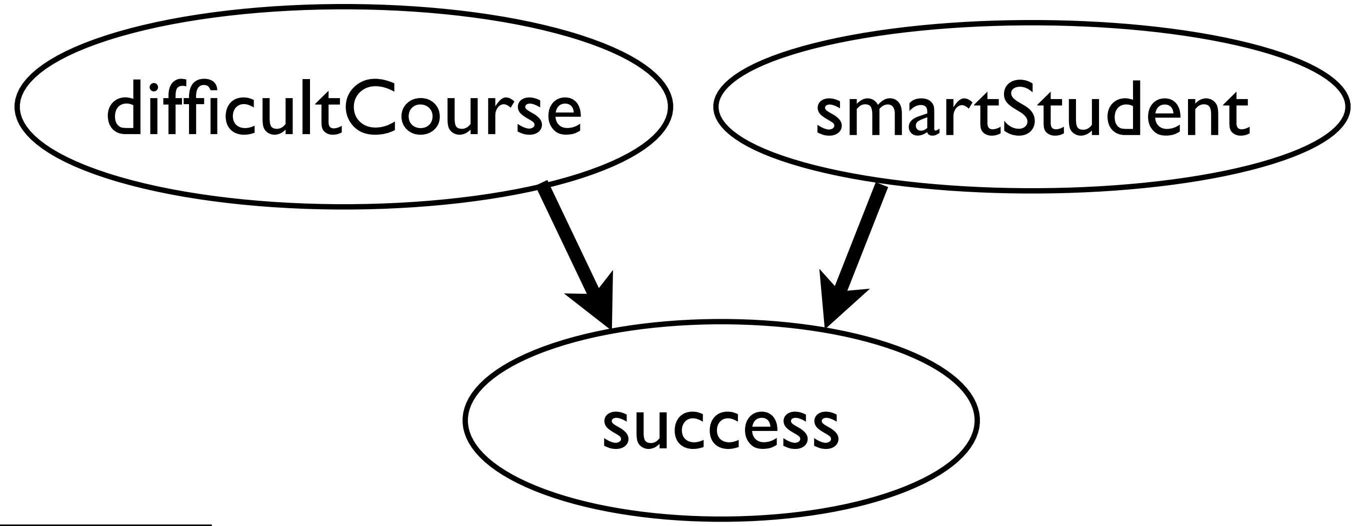
Learning



data-dependent set of random variables

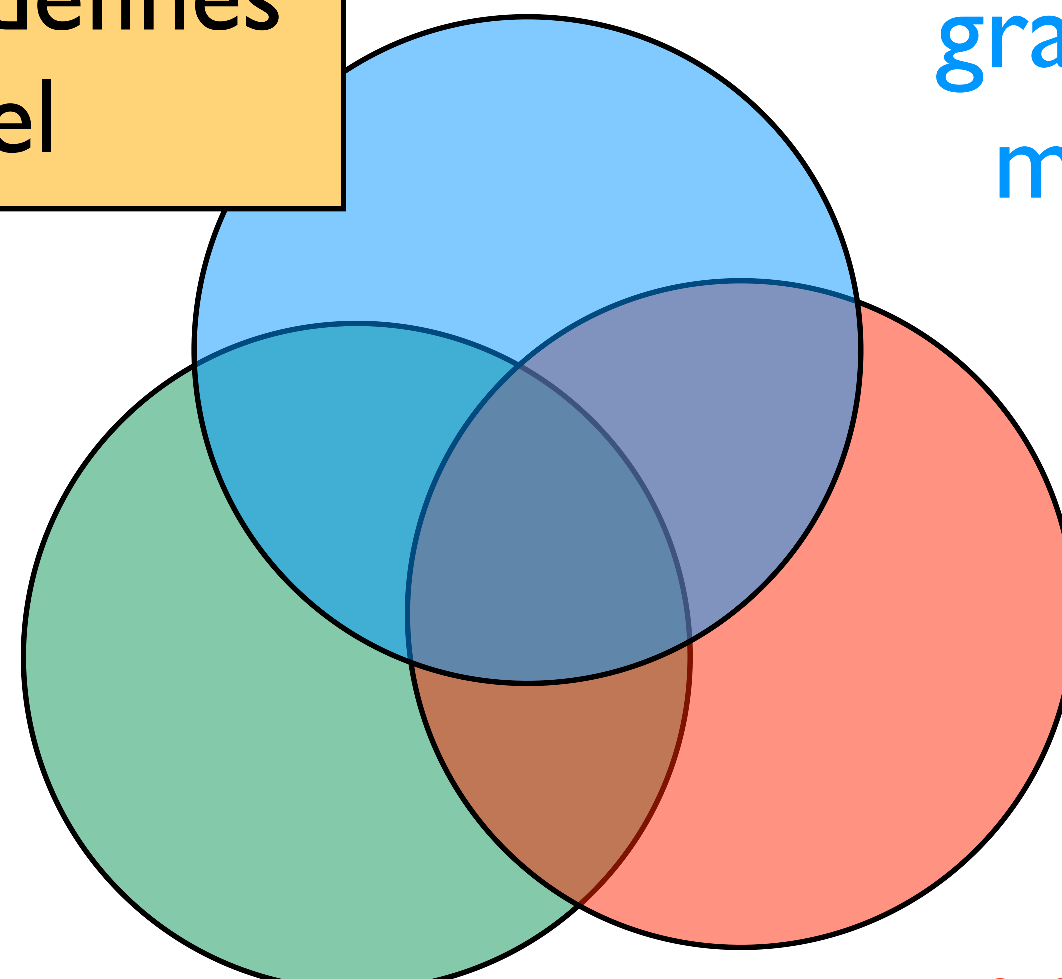
Lifted graphical models

fixed set of random variables



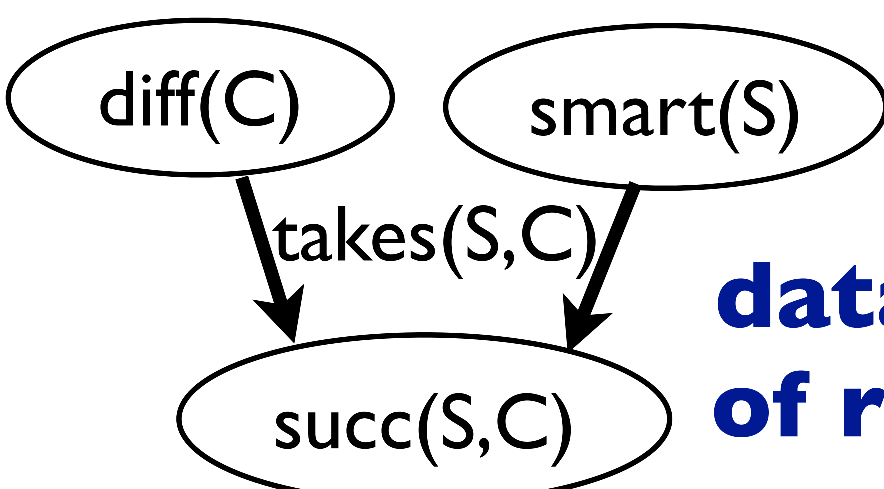
relational language defines graphical model

graphical model



relational definition of graphical model

Learning parameters & structure



data-dependent set of random variables

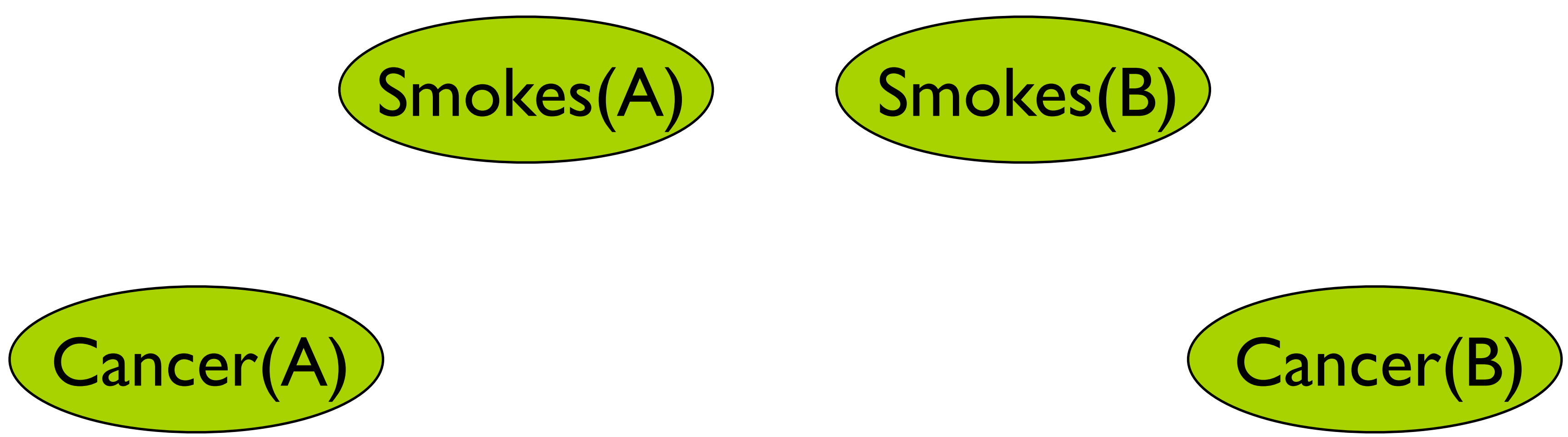
Lots of proposals in the literature, e.g.

- relational Markov networks (RMNs) [Taskar et al 2002]
- **Markov logic networks (MLNs)** [Richardson & Domingos 2006]
- *probabilistic soft logic (PSL) [Broecheler et al 2010]*
- FACTORIE [McCallum et al 2009]
- Bayesian logic programs (BLPs) [Kersting & De Raedt 2001]
- relational Bayesian networks (RBNs) [Jaeger 2002]
- logical Bayesian networks (LBNs) [Fierens et al 2005]
- probabilistic relational models (PRMs) [Koller & Pfeffer 1998]
- Bayesian logic (BLOG) [Milch et al 2005]
- CLP(BN) [Santos Costa et al 2008]
- and many more ...

Markov Logic

1.5	$\forall x\ Smokes(x) \Rightarrow Cancer(x)$
1.1	$\forall x, y\ Friends(x, y) \Rightarrow \big(Smokes(x) \Leftrightarrow Smokes(y)\big)$

Suppose we have two constants: **Anna** (A) and **Bob** (B)

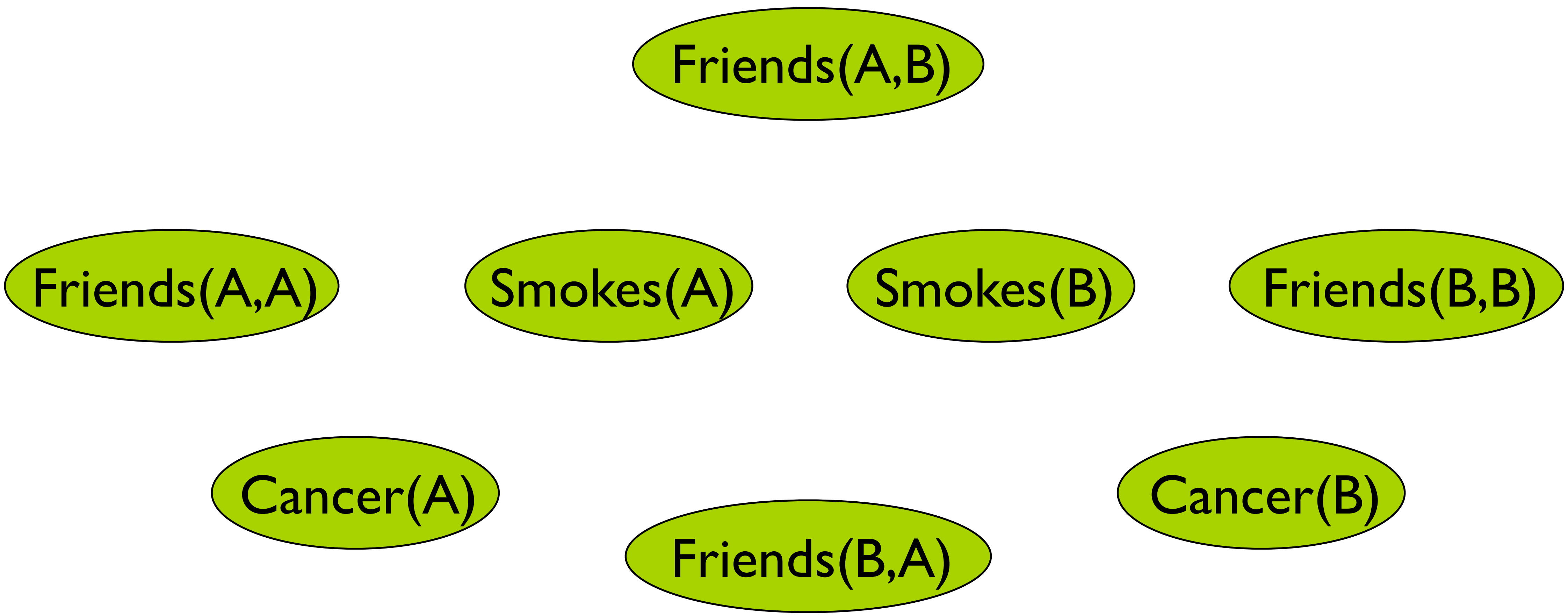


slides by Pedro Domingos

Markov Logic

1.5	$\forall x\ Smokes(x) \Rightarrow Cancer(x)$
1.1	$\forall x, y\ Friends(x, y) \Rightarrow \big(Smokes(x) \Leftrightarrow Smokes(y)\big)$

Suppose we have two constants: **Anna** (A) and **Bob** (B)

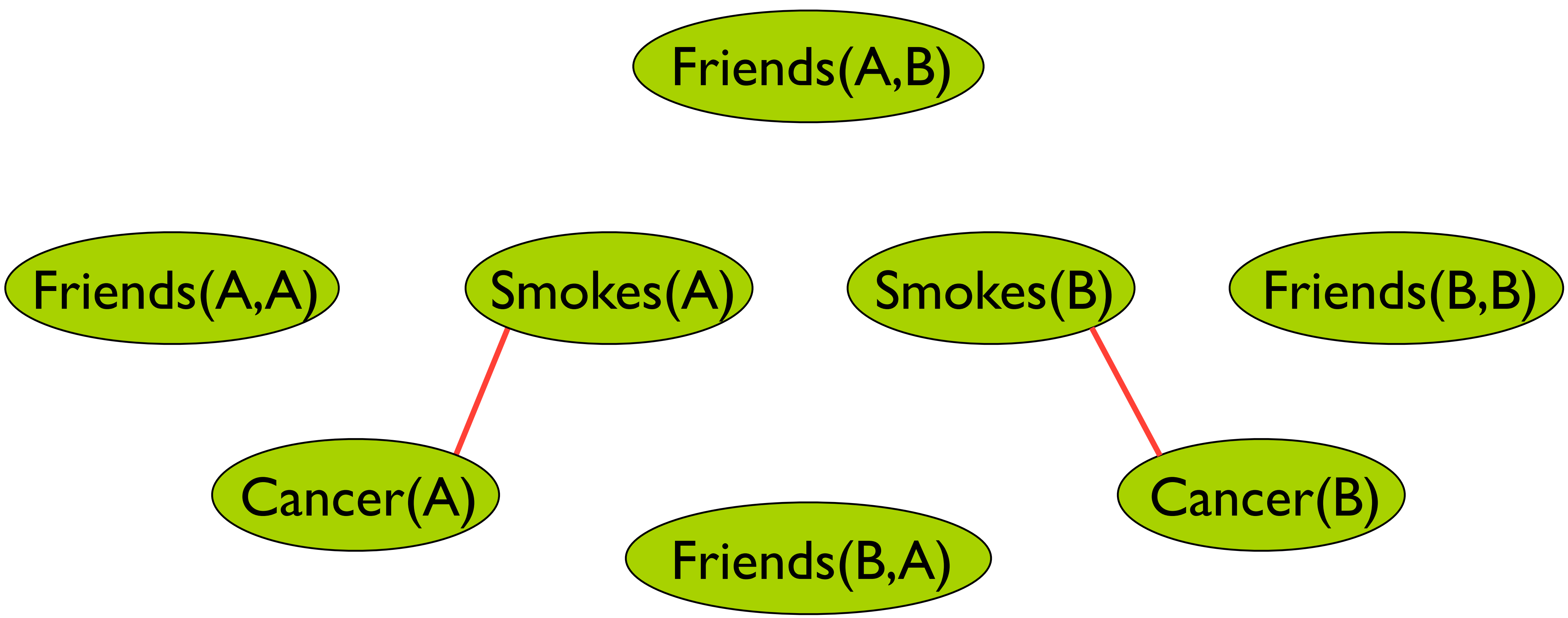


slides by Pedro Domingos

Markov Logic

1.5	$\forall x \; Smokes(x) \Rightarrow Cancer(x)$
1.1	$\forall x, y \; Friends(x, y) \Rightarrow \big(Smokes(x) \Leftrightarrow Smokes(y)\big)$

Suppose we have two constants: **Anna** (A) and **Bob** (B)

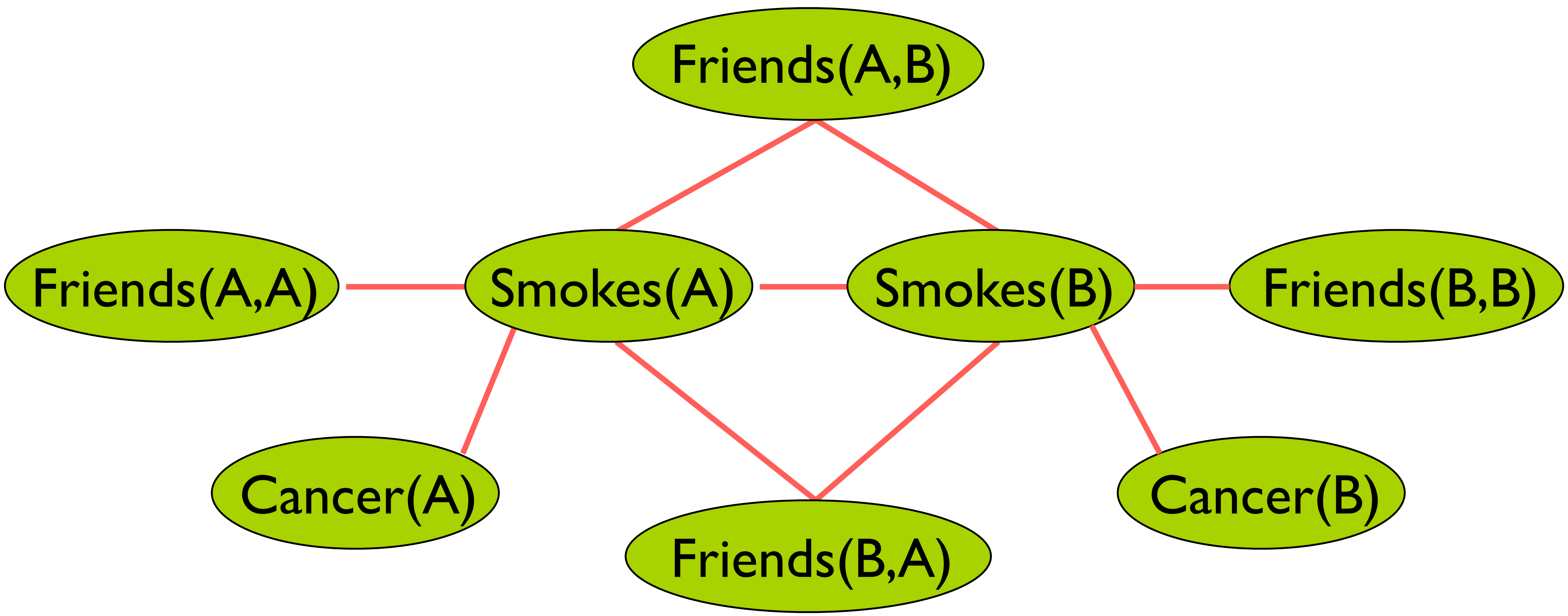


slides by Pedro Domingos

Markov Logic

1.5	$\forall x\ Smokes(x) \Rightarrow Cancer(x)$
1.1	$\forall x, y\ Friends(x, y) \Rightarrow \left(Smokes(x) \Leftrightarrow Smokes(y)\right)$

Suppose we have two constants: **Anna** (A) and **Bob** (B)



slides by Pedro Domingos

Markov Logic

- MLNs are a template for ground Markov Networks
- Probability of a world/interpretation
- If $n_i = 0$ then $P(x) = \frac{1}{Z}$

$$P(x) = \frac{1}{Z}\exp\left(\sum_i \underbrace{w_i}_{\text{Weight of formula } i} \underbrace{n_i(x)}_{\text{No. of true groundings of formula } i \text{ in } x}\right)$$

Possible Worlds

A vocabulary

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)
0	0	0	0
⋮	⋮	⋮	⋮
1	0	1	0
⋮	⋮	⋮	⋮
1	1	1	1

Possible worlds
Logical interpretations

Slides adapted from Guy Van den Broeck

Possible Worlds

A logical theory

$$\forall x,y,\ \mathrm{Smokes}(x) \wedge \mathrm{Friends}(x,y) \Rightarrow \mathrm{Smokes}(y)$$

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)	theory
0	0	0	0	1
⋮	⋮	⋮	⋮	⋮
1	0	1	0	0
⋮	⋮	⋮	⋮	⋮
1	1	1	1	1

Interpretations that satisfy the theory
Models

Slides adapted from Guy Van den Broeck

First-Order Model Counting

A logical theory

$$\forall x,y,\ Smokes(x) \wedge Friends(x,y) \Rightarrow Smokes(y)$$

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)	theory
0	0	0	0	1
⋮	⋮	⋮	⋮	⋮
1	0	1	0	0
⋮	⋮	⋮	⋮	⋮
1	1	1	1	1

$\sum$ First-order model count

~#SAT

Slides Guy Van den Broeck

Markov Logic

- MLNs are a template for ground Markov Networks
- Probability of a world/interpretation
- If $n_i = 0$ then $P(x) = \frac{1}{Z}$

$$P(x) = \frac{1}{Z} \exp\left(\sum_i \underbrace{w_i}_{\text{Weight of formula } i} \underbrace{n_i(x)}_{\text{No. of true groundings of formula } i \text{ in } x} \right)$$

Markov Logic

A Markov Logic theory

1.5 ∀x,y, Smokes(x) ∧ Friends(x,y) ⇒ Smokes(y)

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)	theory
0	0	0	0	$\frac{1}{Z} exp(1.5 * 2)$
⋮	⋮	⋮	⋮	
1	0	1	0	$\frac{1}{Z} exp(1.5 * 1)$
⋮	⋮	⋮	⋮	
1	1	1	1	$\frac{1}{Z} exp(1.5 * 2)$

counting only substitutions for which X =/= Y
X=Alice, Y=Bob
X=Bob, Y=Alice

Slides adapted from Guy Van den Broeck

Markov Logic

A Markov Logic theory

$1.5\ \forall x,y,\ \text{Smokes}(x) \wedge \text{Friends}(x,y) \Rightarrow \text{Smokes}(y)$

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)	theory
0	0	0	0	$\frac{1}{Z}exp(1.5 * 2)$
⋮	⋮	⋮	⋮	
1	0	1	0	$\frac{1}{Z}exp(1.5 * 1)$
⋮	⋮	⋮	⋮	
1	1	1	1	$\frac{1}{Z}exp(1.5 * 2)$

Σ

Z

partition function

Slides adapted from Guy Van den Broeck

Weighted First-Order Model Counting

A logical theory and a weight function for predicates

Smokes(Alice)	Smokes(Bob)	Friends(Alice,Bob)	Friends(Bob,Alice)	theory	weight
0	0	0	0	1	$2 \cdot 2 \cdot 1 \cdot 1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	0	1	0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	1	1	1	$1 \cdot 1 \cdot 4 \cdot 4$

Smokes $\rightarrow$ 1
$\neg$Smokes $\rightarrow$ 2
Friends $\rightarrow$ 4
$\neg$Friends $\rightarrow$ 1

$\sum$ Weighted first-order model count

Related to ProbLog Inference !

Markov Logic and FOWMC

MLN

$1.5 \quad \text{Smokes}(x) \wedge \text{Friends}(x,y) \Rightarrow \text{Smokes}(y)$

$\forall x,y,\ F(x,y) \Leftrightarrow [\ \text{Smokes}(x) \wedge \text{Friends}(x,y) \Rightarrow \text{Smokes}(y)\ ]$

Relational Logic

$$\begin{aligned} \text{Smokes} &\rightarrow 1 \\ \neg\text{Smokes} &\rightarrow 1 \\ \text{Friends} &\rightarrow 1 \\ \neg\text{Friends} &\rightarrow 1 \\ F &\rightarrow \exp(3.14) \\ \neg F &\rightarrow 1 \end{aligned}$$

Weight Function

Slides Guy Van den Broeck

Lifted Inference

- Usual approach is to ground out and apply propositional WMC solver
- Lifted inference : Can we avoid grounding out ?
- Focus of current research in SRL, prob. Databases
- Key to efficient inference — exploit symmetries
- A lot of progress in the last five years.
- See work by Van den Broeck, Suciu, Poole, Darwiche

Example:
First-Order Model Counting

Logical sentence

Domain

$\forall x,y, \text{Smokes}(x) \wedge \text{Friends}(x,y) \Rightarrow \text{Smokes}(y)$

n people

If you know there are *k* smokers

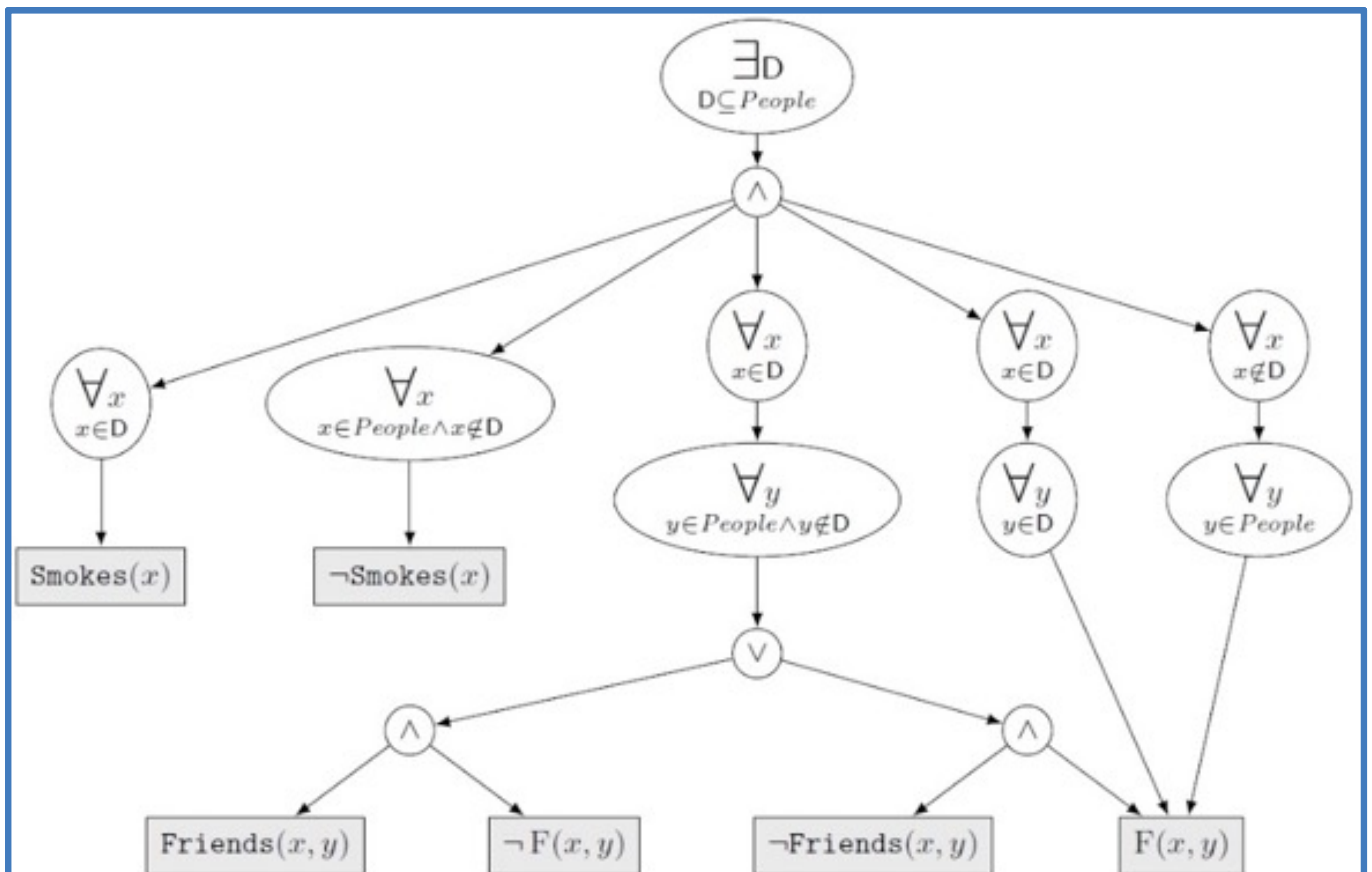
$$\sum_{k=0}^{n} \binom{n}{k} 2^{n^2 - k(n-k)}$$

Slides Guy Van den Broeck

The Full Pipeline

$$\forall x,y,\ F(x,y) \Leftrightarrow [\ Smokes(x) \wedge Friends(x,y) \Rightarrow Smokes(y)\ ]$$

Relational Logic



First-Order
d-DNNF Circuit

Slides Guy Van den Broeck

Parameter Learning

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

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

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

Has been used for generative and discriminative learning
Many applications in networks, NLP, bioinformatics, …

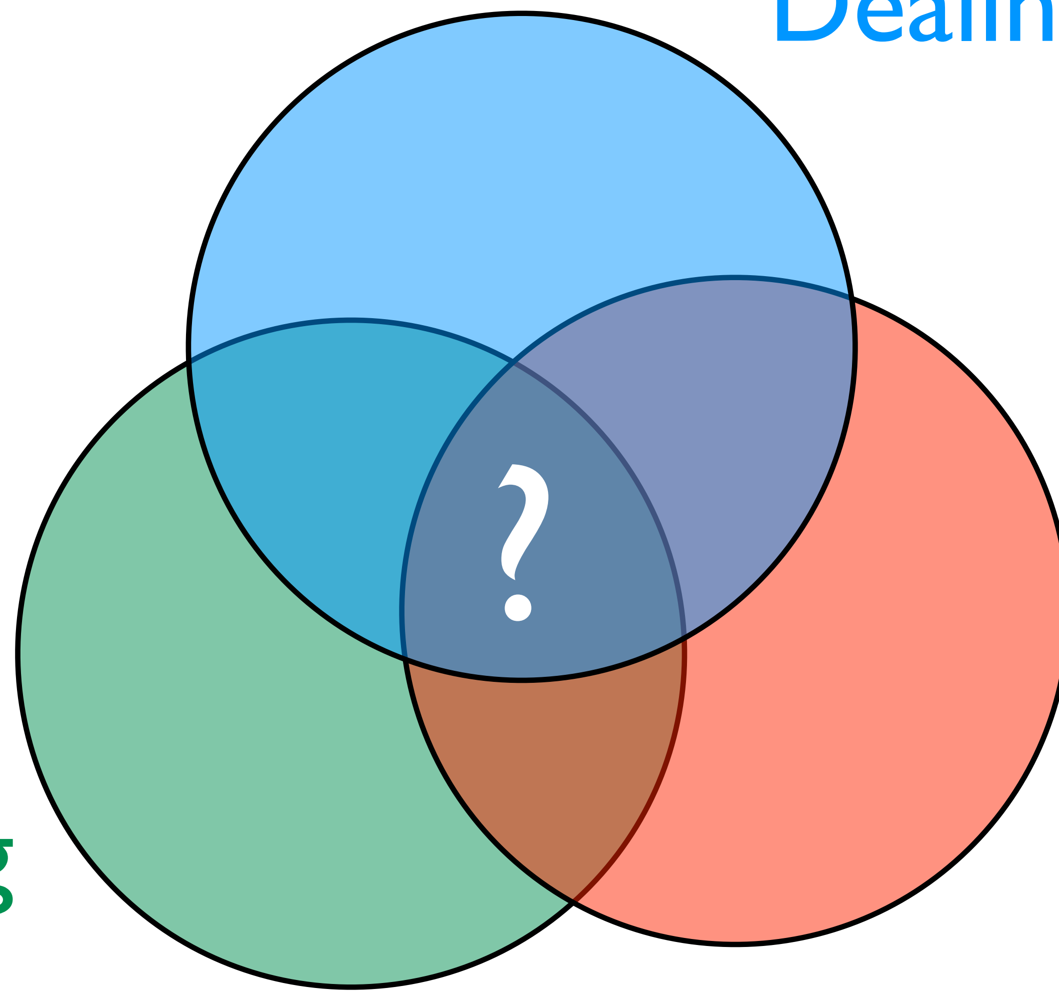
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...



Learning

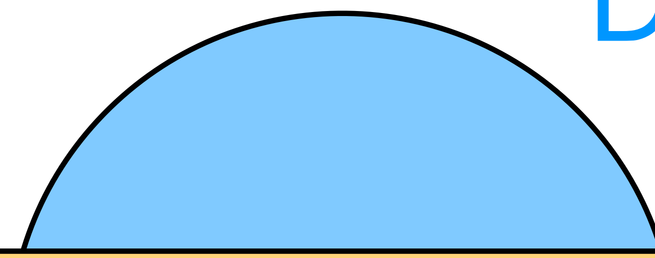
- parameters
- structure

Statistical relational learning
Probabilistic programming, ...

A key question in AI:

Dealing with uncertainty
• probability theory
dels

Reas
relat
• log
• da
• pr
• ...



• Many languages, systems, applications, ...
• not yet a technology ! but a lot of progress
• and a lot more to do !
• … excellent area for PhDs …

structure

Statistical relational learning
Probabilistic programming, ...

Further Reading

- Three websites to start
  - http://probmods.org/   Probabilistic Models of Cognition — Church
  - http://dtai.cs.kuleuven.be/problog/ — check also [DR & Kimmig, MLJ 15]
  - http://alchemy.cs.washington.edu/ —Markov Logic, check also [Domingos & Lowd] Markov Logic, Morgan Claypool.