Probabilistic (Logic) Programming and Statistical Relational Learning

Luc De Raedt and Angelika Kimmig
Also many slides from Guy Van den Broeck and others

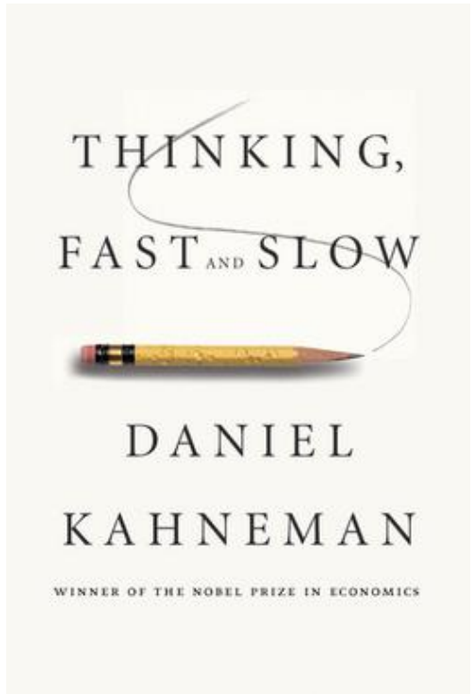






SYNTH project

Learning and Reasoning both needed



- System 1 - thinking fast - can do things like 2+2 = ? and recognise objects in image
- System 2 - thinking slow - can reason about solving complex problems - planning a complex task
- alternative terms — data-driven vs knowledge-driven, symbolic vs subsymbolic, solvers and learners, …
- AlphaGo incorporates both learning and reasoning
- Machine learned videogames — cannot change the rules of the game

see also arguments
by Darwiche, Levesque, Tenenbaum, Geffner

Reasoning

Two key paradigms

logic

probability

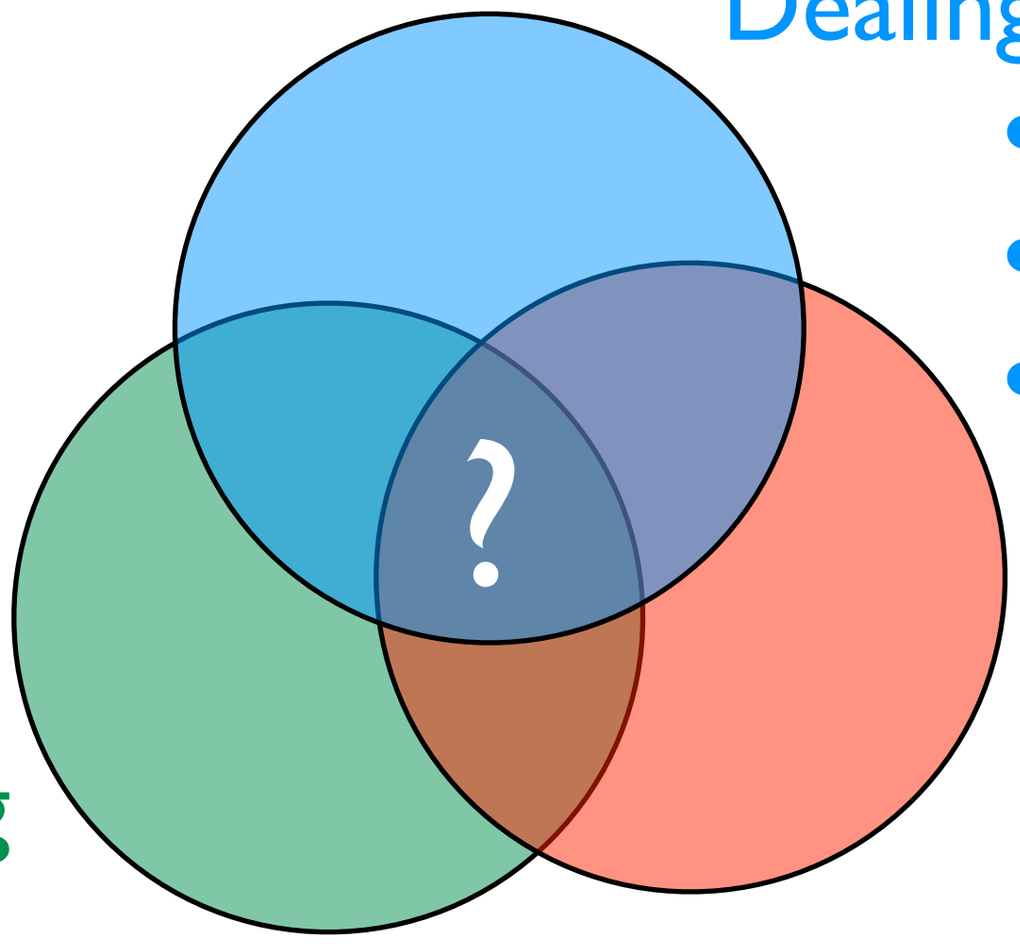
A key question in AI:

Dealing with uncertainty

- probability theory
- graphical models
- ...

Reasoning with relational data

- logic
- databases
- programming
- ...

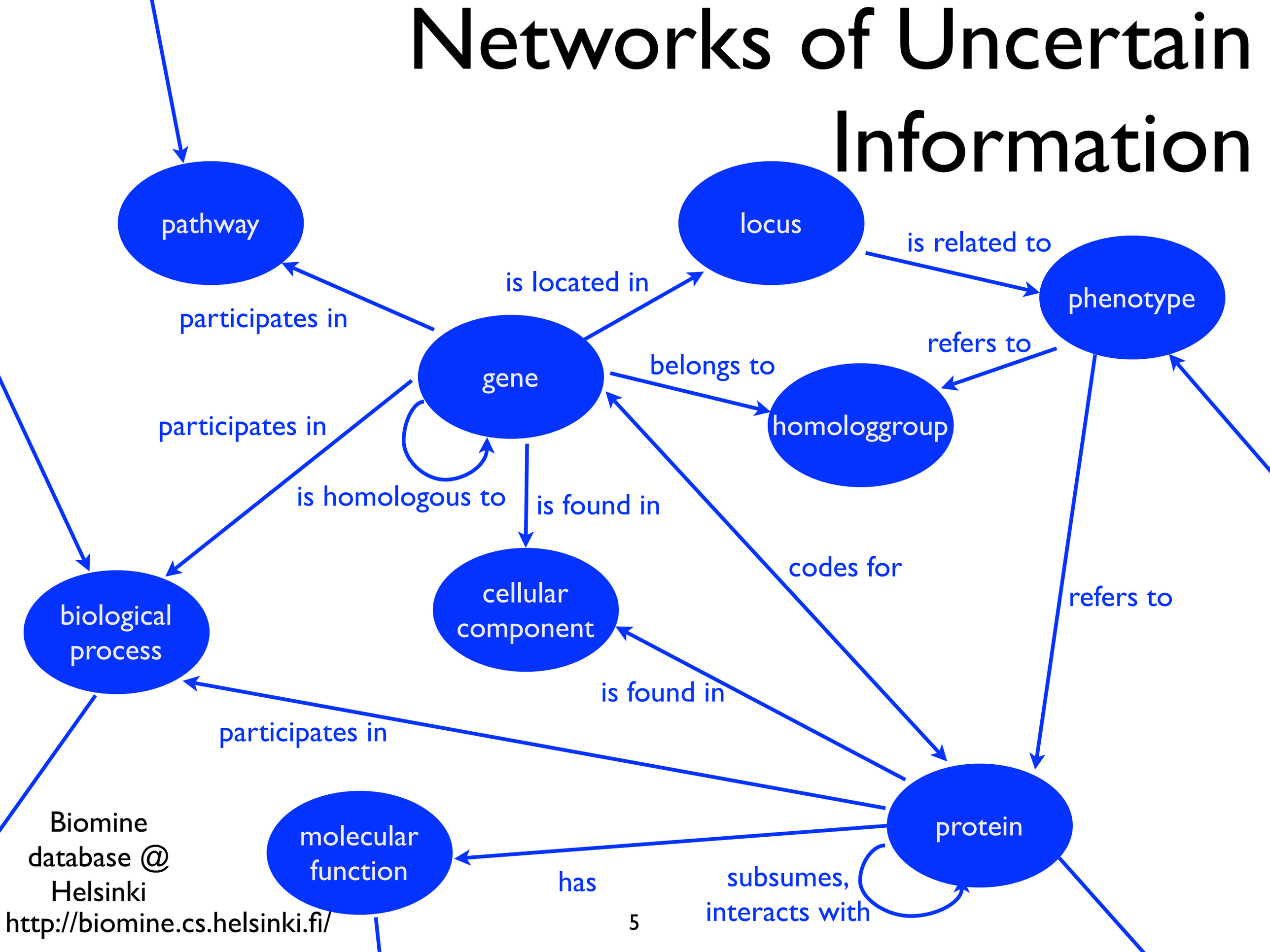


Learning

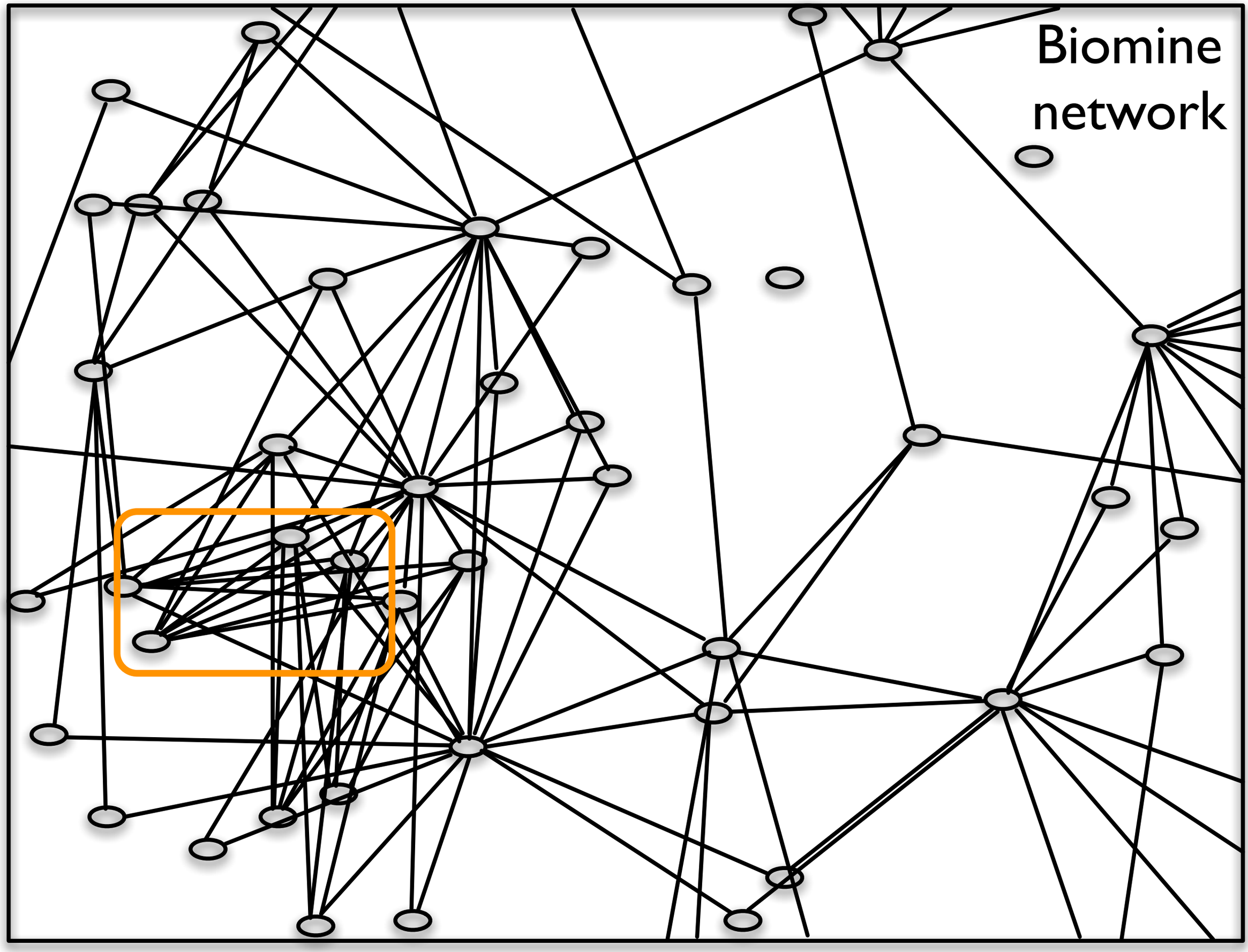
- parameters
- structure

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

Networks of Uncertain Information



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


Biomine
network

Notch receptor processing
Biologic
GO:GO

BiologicalProces

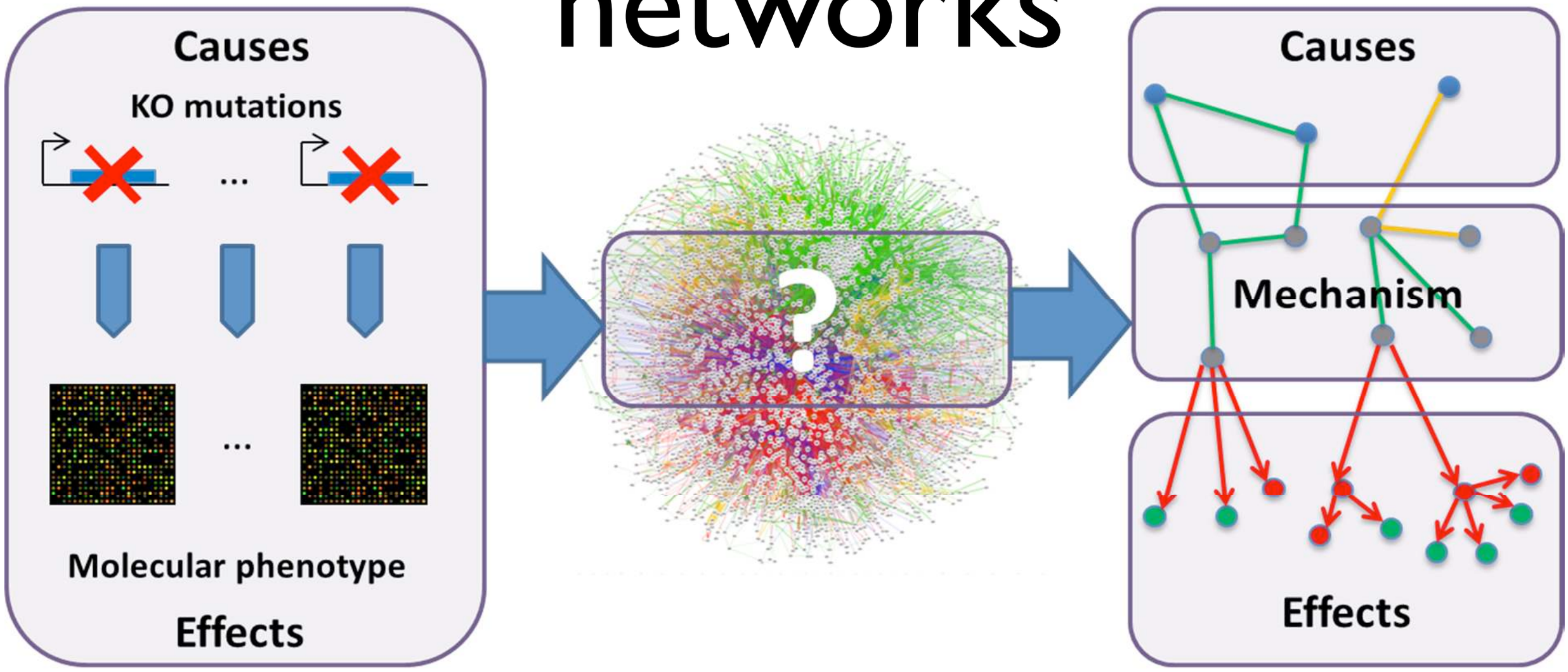
-participates_in
0.220

presenilin 2
Gene
EntrezGene:817

Gene

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

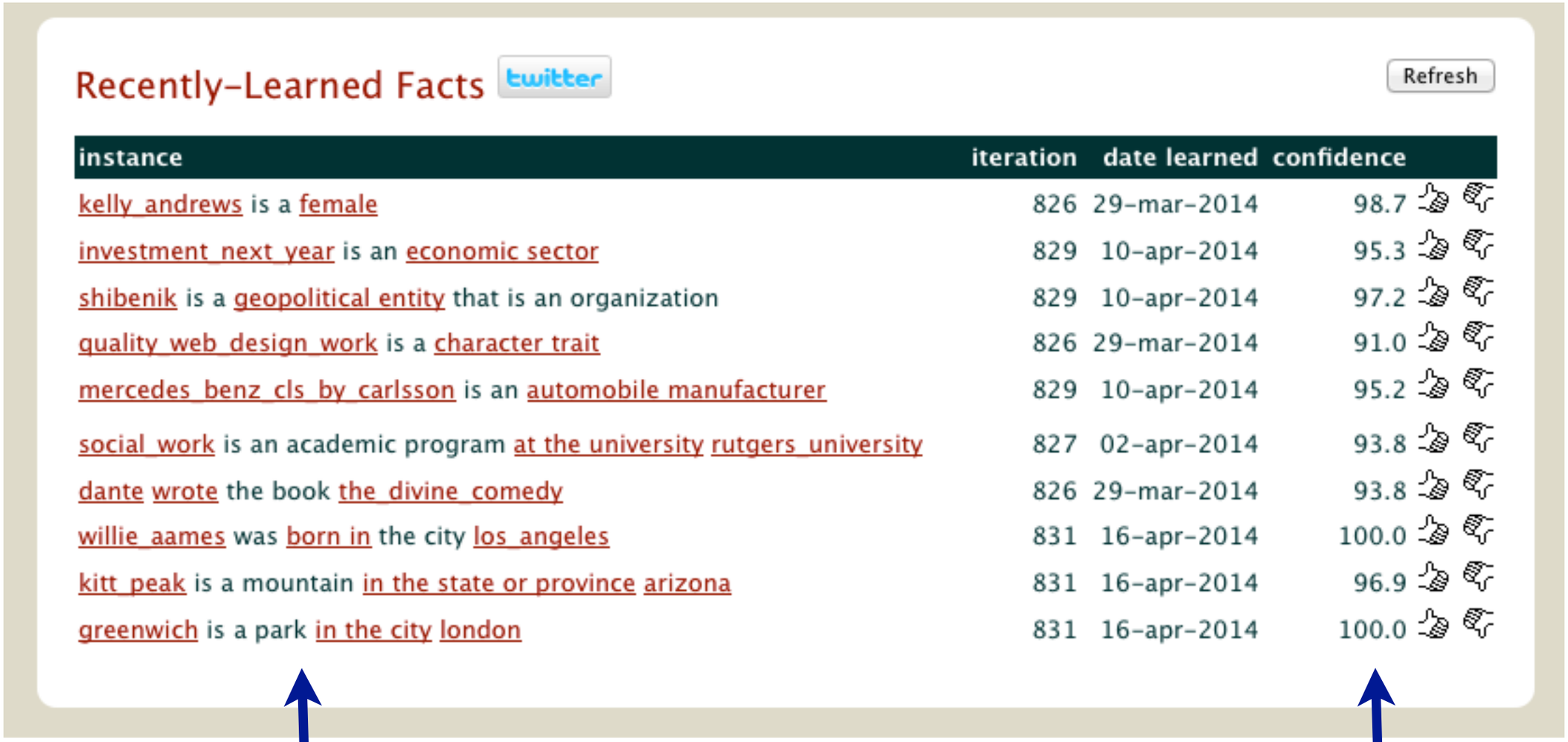
Molecular interaction networks



Can we find the mechanism connecting causes to effects?

[De Maeyer et al, Molecular Biosystems 13 NAR 15]

Example: Information Extraction



Recently-Learned Facts twitter

Refresh

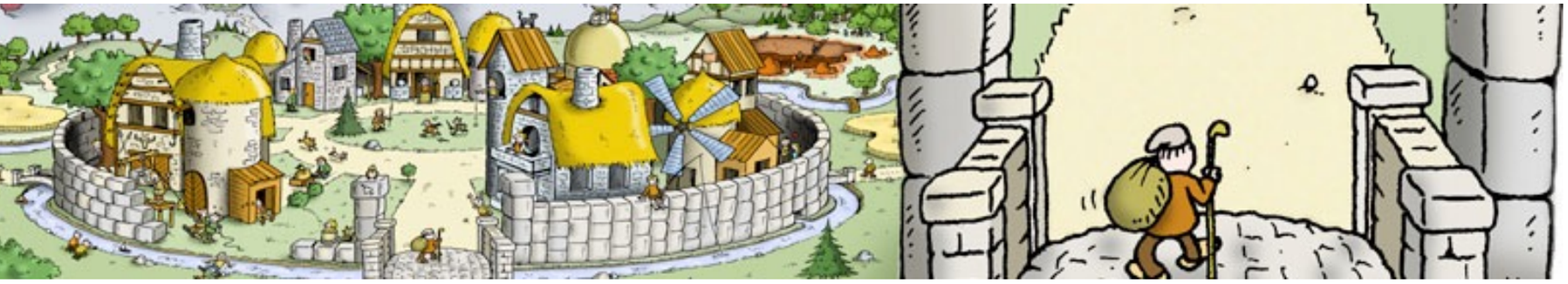
instance	iteration	date learned	confidence
kelly_andrews is a female	826	29-mar-2014	98.7
investment_next_year is an economic sector	829	10-apr-2014	95.3
shibenik is a geopolitical entity that is an organization	829	10-apr-2014	97.2
quality_web_design_work is a character trait	826	29-mar-2014	91.0
mercedes_benz_cls_by_carlsson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

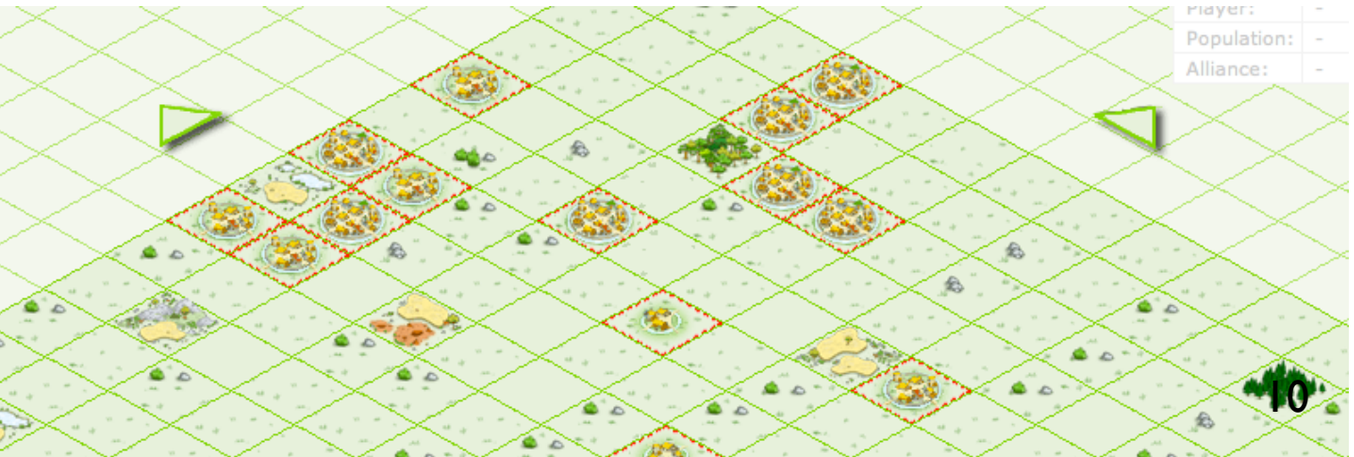
NELL: http://rtw.ml.cmu.edu/rtw/

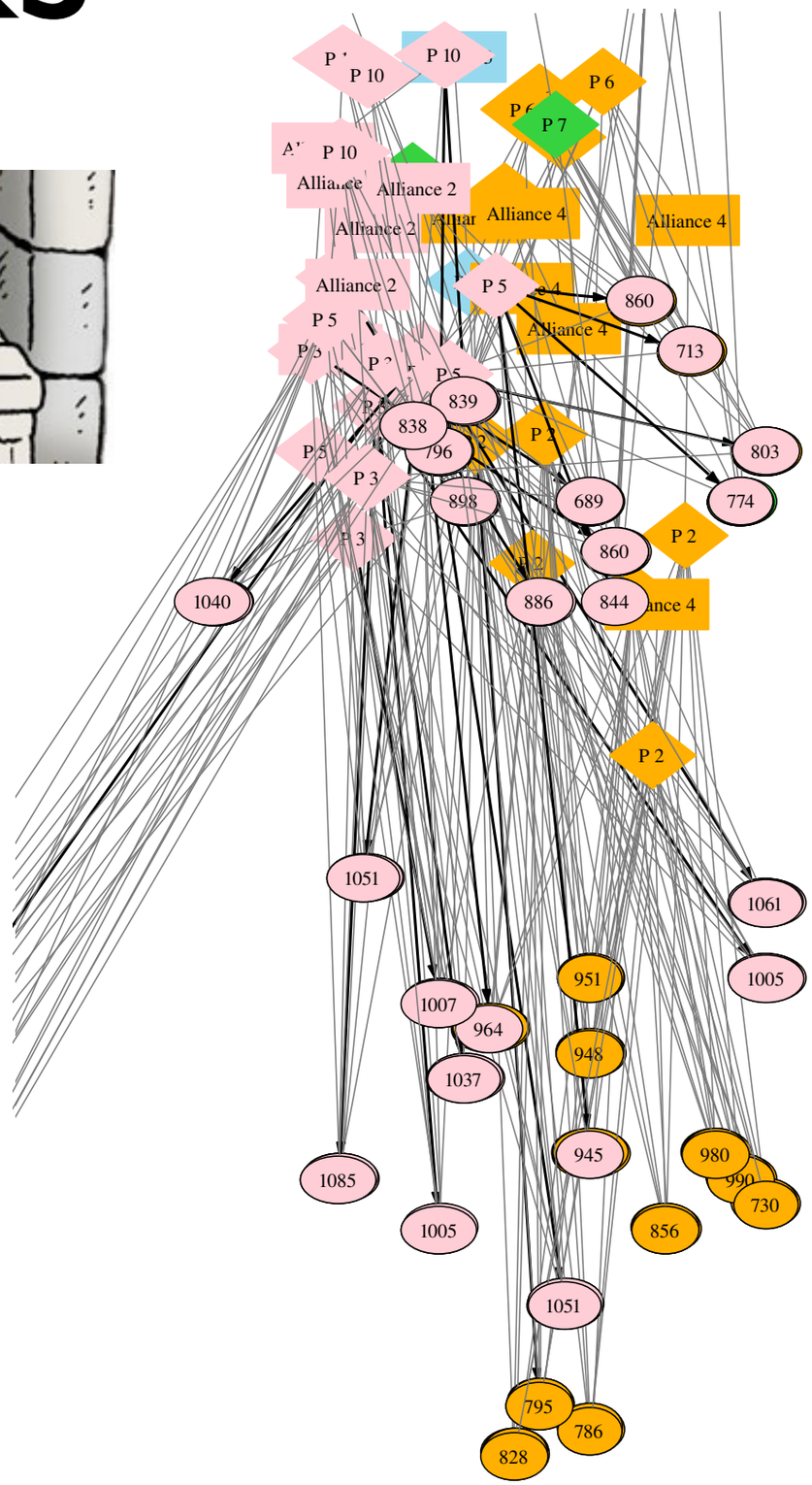
Dynamic networks



Travian: A massively multiplayer real-time strategy game

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



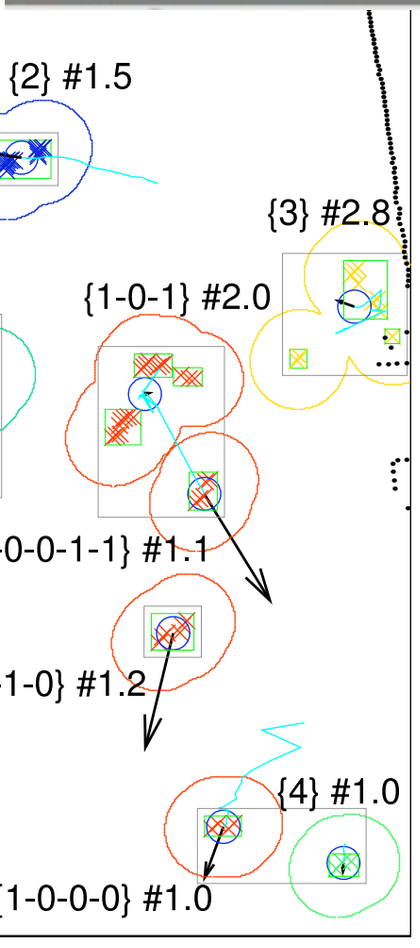


[Thon et al, MLJ 11]

Analyzing Video Data

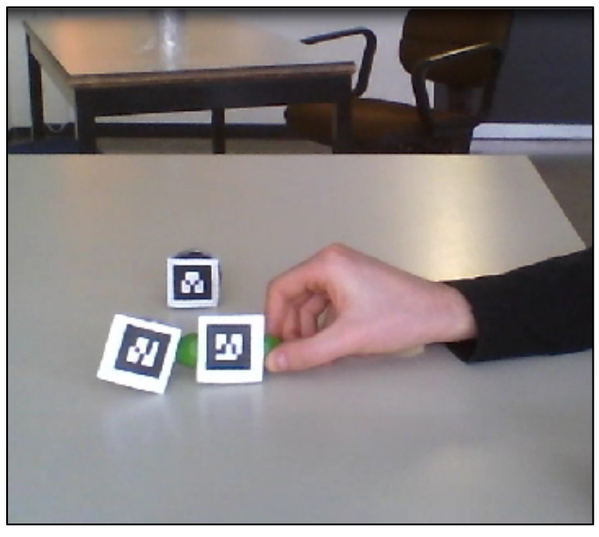






- Track people or objects over time? Even if temporarily hidden?
- Recognize activities?
- Infer object properties?

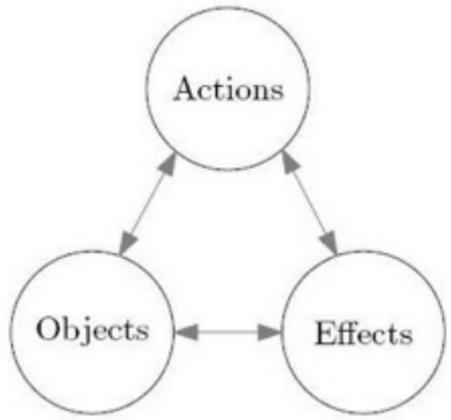




[Skarlatidis et al, TPLP 14;
Nitti et al, IROS 13, ICRA 14]

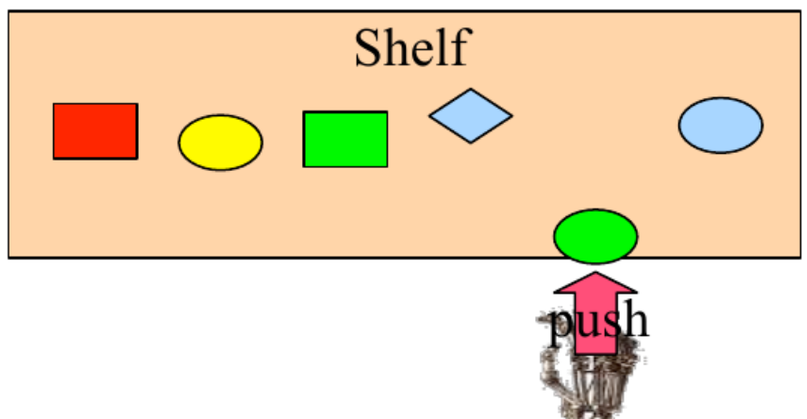
Relational Affordances

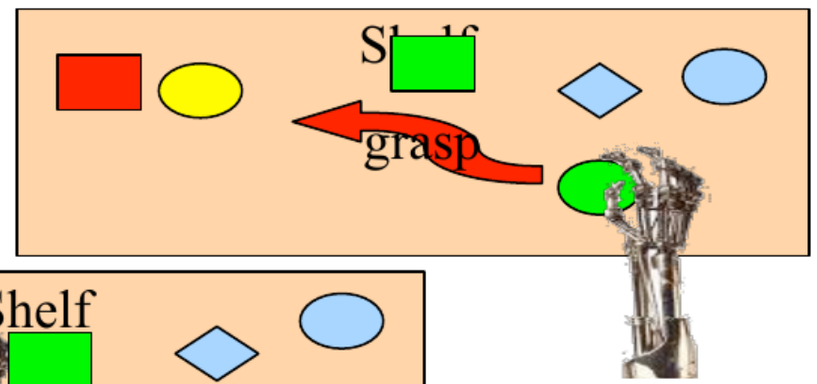
- **Object Affordance:**
  What can one do with particular object?
- **Relational Affordance:**
  in a particular context?
  with multiple objects and relations among them

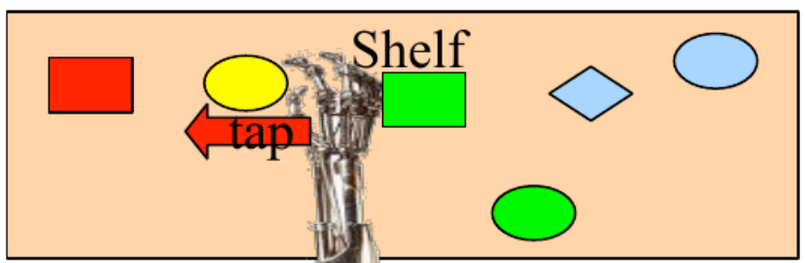


Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

- Use of statistical relational learning, **probabilistic programming for learning, reasoning and planning !**







Answering Probability Questions



Mike has a bag of marbles with 4 white, 8 blue, and 6 red marbles. He pulls out one marble from the bag and it is red. What is the probability that the second marble he pulls out of the bag is white?

The answer is 0.235941.



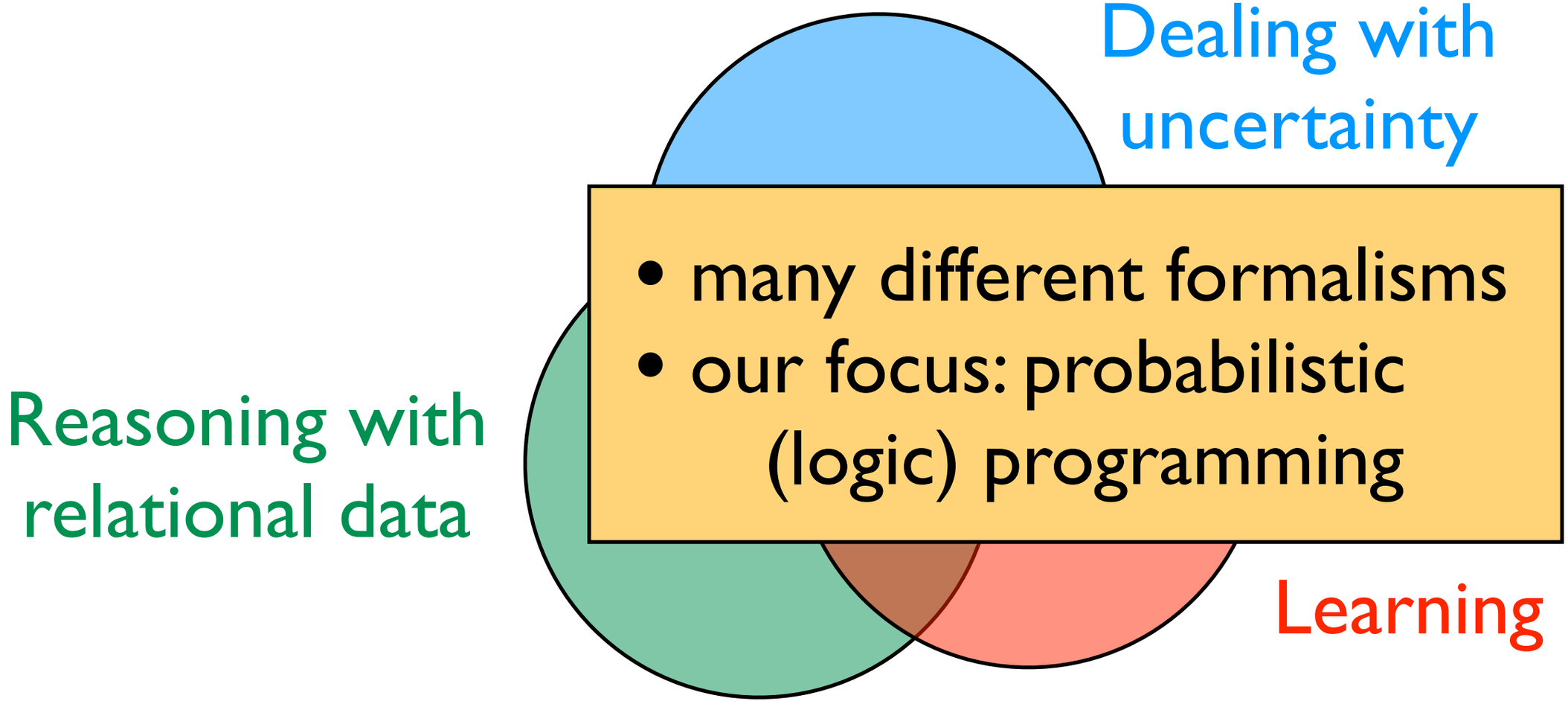
[Dries et al., IJCAI 17]

This requires dealing with

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

Statistical Relational Learning
Probabilistic Programming

Common theme



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

The (Incomplete) SRL Alphabet Soup

Relational Gaussian Processes

Infinite Hidden Relational Models

[names in alphabetical order]

´10 PSL: Broecheler, Getoor, Mihalkova

´07 RDNs: Jensen, Neville

´90 ´93 ´94 ´95 ´96 ´97 ´99 ´00 ´02 ´03

Relational Markov Networks

Object-Oriented Bayes Nets

Logical Bayesian Networks: Blockeel,Bruynooghe, Fierens,Ramon,

IBAL

Figaro

BUGS/Plates

LOHMMs: De Raedt, Kersting, Raiko

1BC(2): Flach, Lachiche

First KBMC approaches: Bresse, Bacchus, Charniak, Glesner, Goldman, Koller, Poole, Wellmann

Prob. Horn Abduction: Poole

RMMs: Anderson,Domingos, Weld

Multi-Entity Bayes Nets

SPOOK

PLP: Haddawy, Ngo

BLPs: Kersting, De Raedt

LPAD: Bruynoogh Vennekens,Verbaeter

PRMs: Friedman,Getoor,Koller, Pfeffer,Segal,Taskar

PRISM: Kameya, Sato

DAPER

Markov Logic: Domingos, Richardson

SLPs: Cussens,Muggleton

Church

CLP(BN): Cussens,Page, Qazi,Santos Costa

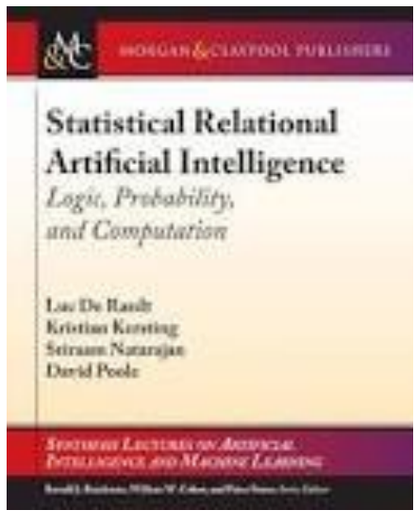
Prob. CLP: Eisele, Riezler

De Ra

Probabilistic Entity-Relationship Models

Many different angles

- Probabilistic programming
  - Logic programming and probabilistic databases (ProbLog and DS as representatives)
  - Functional and imperative (Church as representatives)
- Statistical relational AI and learning
  - Markov Logic
  - Relational Bayesian Networks (and variants)



Why are we doing this ?

- Reasoning (Probability + Logic) AND Learning
- SRL : Expressive Probabilistic Graphical Models
  - First order logic results supports entities + relationships + background knowledge — abstraction of multiple entities
  - Recursion (e.g. smokers cannot be represented by a plate model)
- PP : Power of a universal Turing machine = a prog. language
  - you can program in it and have builtin expressive prob. models
  - PP can learn -> so bring learning to programming languages
- ProbLog fits both paradigms

Logic: Syntax

Variable
constant

class(Book,Topic) :-

author(Book,Author), favorite-topic(Author,Topic).

the class of Book is Topic IF

the Book is written by Author AND

Topic is the favorite of the Author

favorite-topic(rowlings,fantasy)

the favorite topic of rowlings is fantasy

predicate/relation

Logic: three views

entailment/logical consequence

class(Book, Topic) :-

author(Book,Author), favorite-topic(Author,Topic)

favorite-topic(rowlings, fantasy).

author(rowlings, hp).

|=

class(harrypotter, fantasy) .

Logic: three views

models

{class(hp,fan), author(row,hp), fav(row,fan)}

is a (Herbrand) model for

it satisfies

class(Book,Topic) :-

author(Book,Author), favorite-topic(Author,Topic)

favorite-topic(rowlings,fantasy).

author(rowlings,hp).

Logic: three views

proof

class(hp, fan)



author(hp, Aut), fav(Aut, fan)



fav(row, fan)



[]

is a proof given H

```
class(Book,Topic) :-
    author(Book,Author),
    favorite-topic(Author,Topic)
favorite-topic(rowlings,fantasy).
author(rowlings,hp).
```

ProbLog

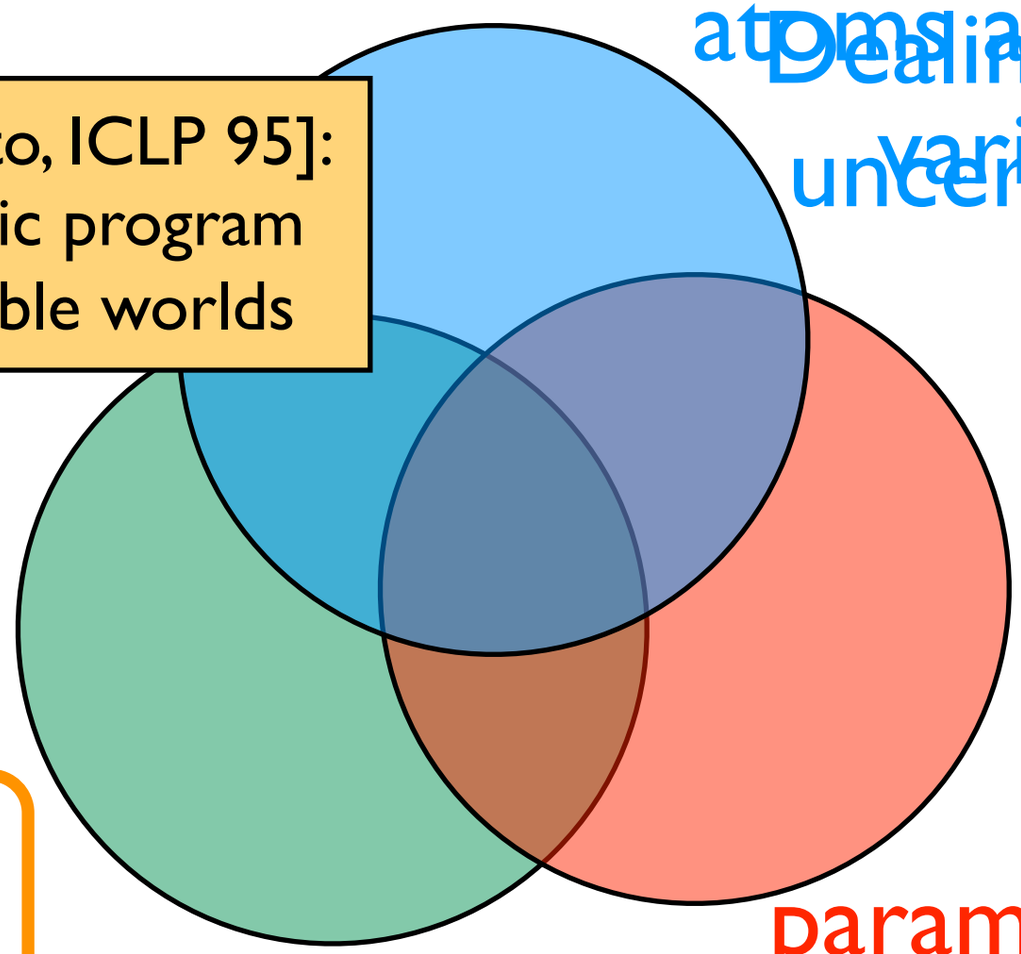
probabilistic Prolog

several possible worlds

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

Dealing with uncertainty

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



Reasoning with probabilities

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

one world

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

Learning
parameter learning,
adapted relational
learning techniques

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

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Graphs & Randomness

ProbLog, Phenetic, Prism, ICL, Probabilistic Databases, ...

- all based on a "random graph" model

Stochastic Logic Programs, ProPPR, PCFGs, ...

- based on a "random walk" model
- connected to PageRank

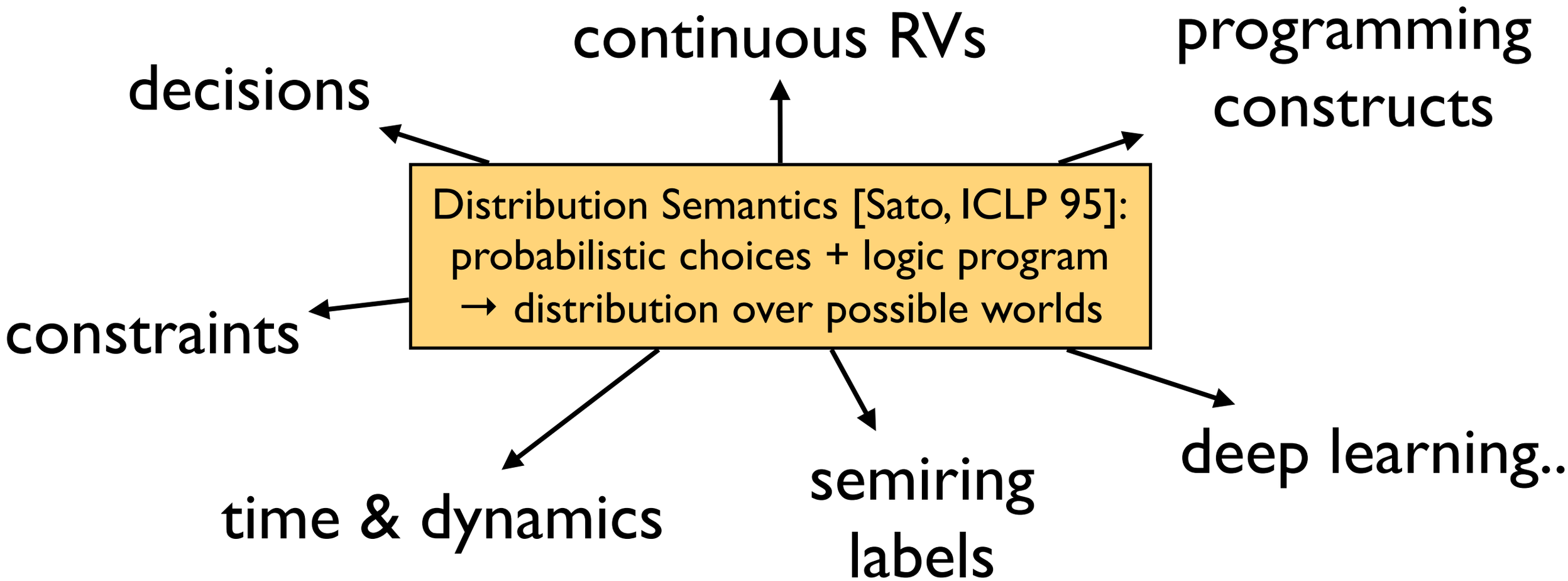
Probabilistic Logic Programming

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

Extensions of basic PLP



Roadmap

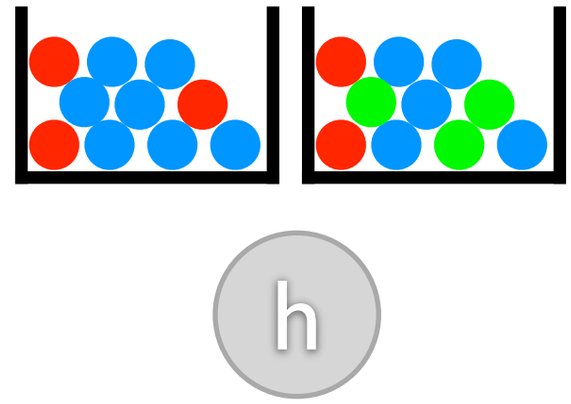
- Prob. Programming - Modeling
- Inference
- Learning
- Dynamics
- Logic, Probability and Neural Nets
- KBMC & Markov Logic
- Lifted Inference

... with some detours on the way

Part 1 : Modeling

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).
```

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

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

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

logical rule encoding background knowledge

consequences

Questions

```
0.4 :: heads.

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

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

marginal probability

- Probability of `win`?

query **conditional probability**

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

evidence

- Most probable world where `win` is true?

MPE inference

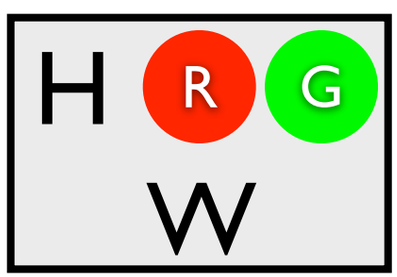
Possible Worlds

```
0.4 :: heads.

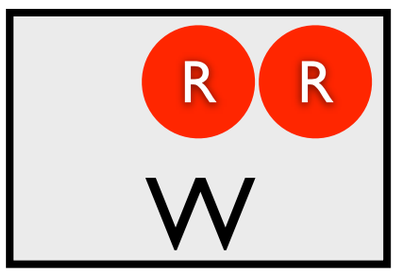
0.3 :: col(1,red); 0.7 :: col(1,blue).
0.2 :: col(2,red); 0.3 :: col(2,green); 0.5 :: col(2,blue).

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

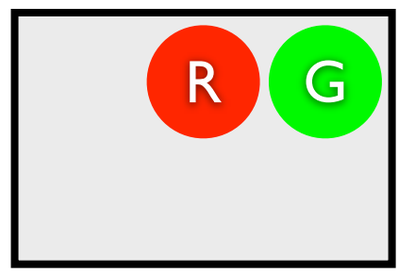
$0.4 \times 0.3 \times 0.3$



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

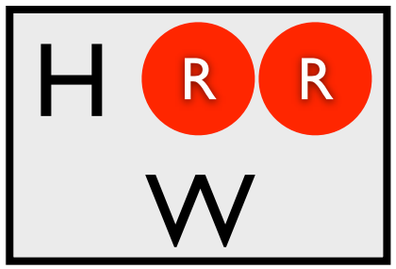


$(1-0.4) \times 0.3 \times 0.3$

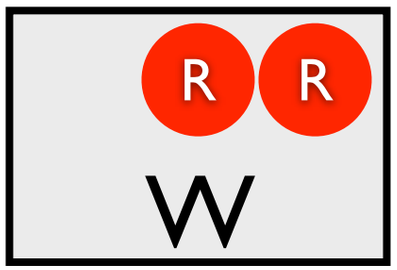


All Possible Worlds

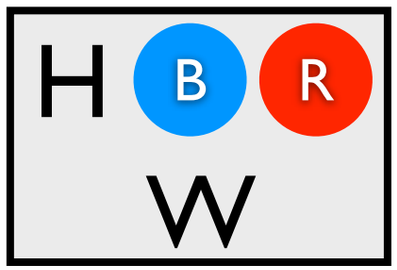
0.024



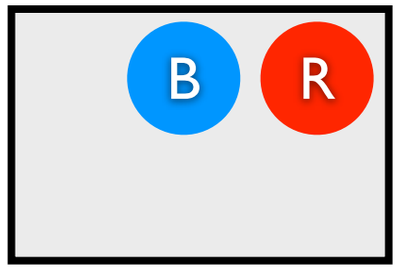
0.036



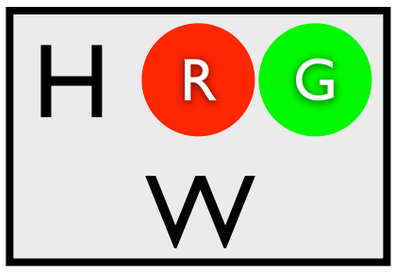
0.056



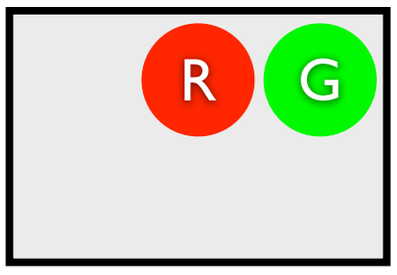
0.084



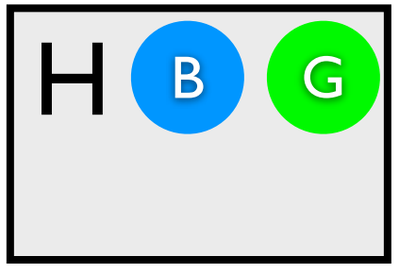
0.036



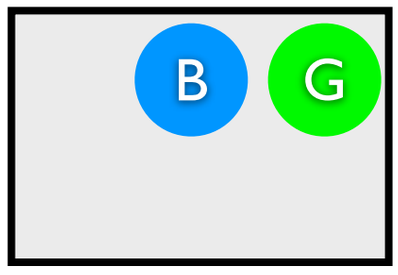
0.054



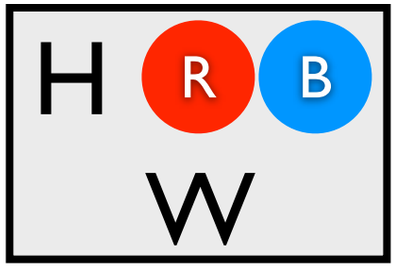
0.084



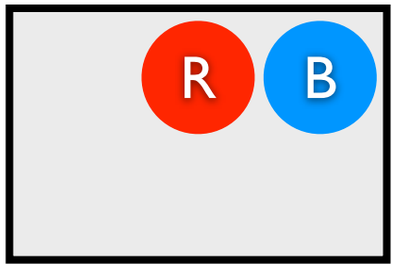
0.126



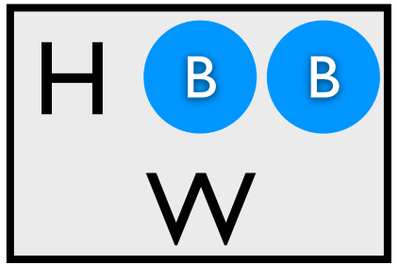
0.060



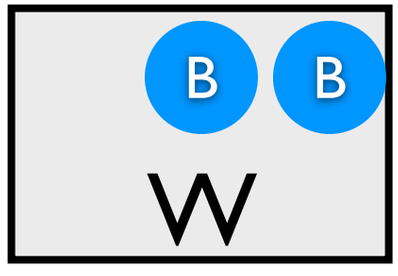
0.090



0.140

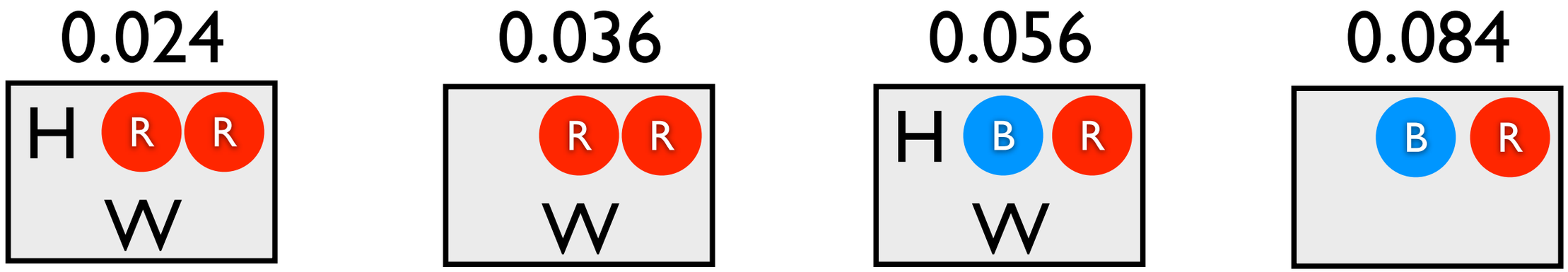


0.210



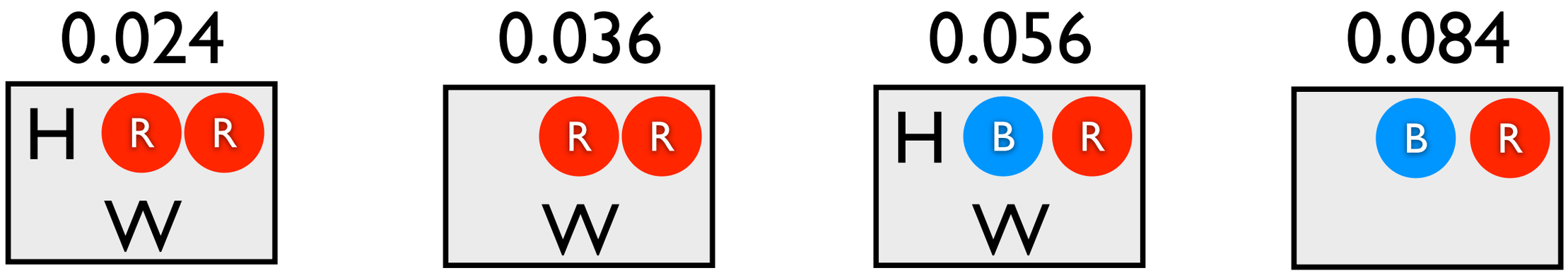
Most likely world where **`win`** is true?

MPE Inference



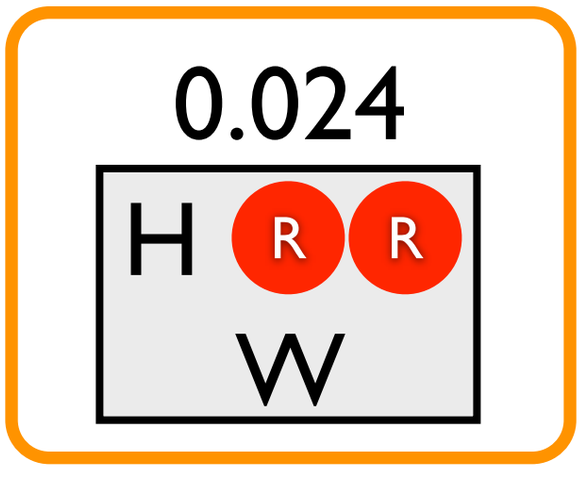
Most likely world where `col(2,blue)` is false?

MPE Inference



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

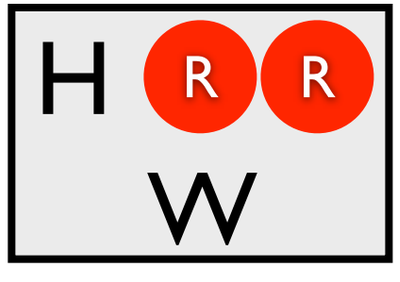
Marginal Probability

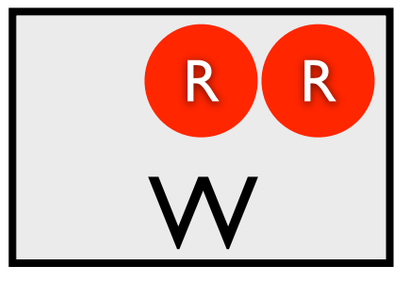


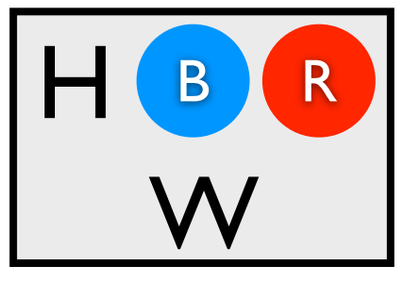
$$P(win|col(2,green)) = ?/\Sigma$$

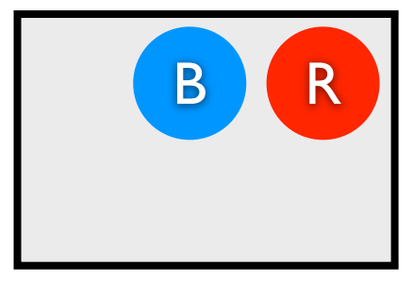
$$= P(win \wedge col(2,green))/P(col(2,green))$$

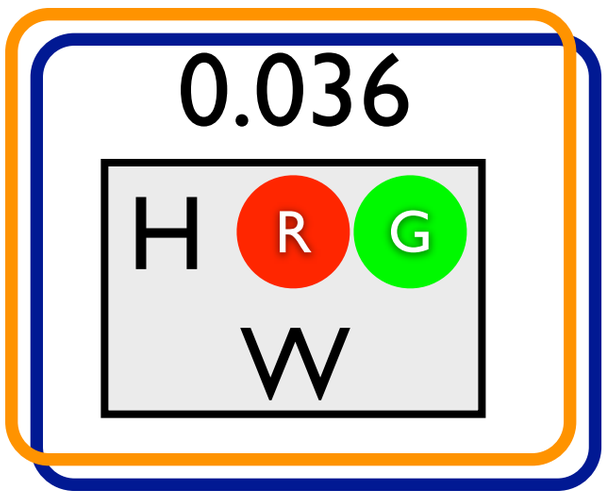
Conditional Probability

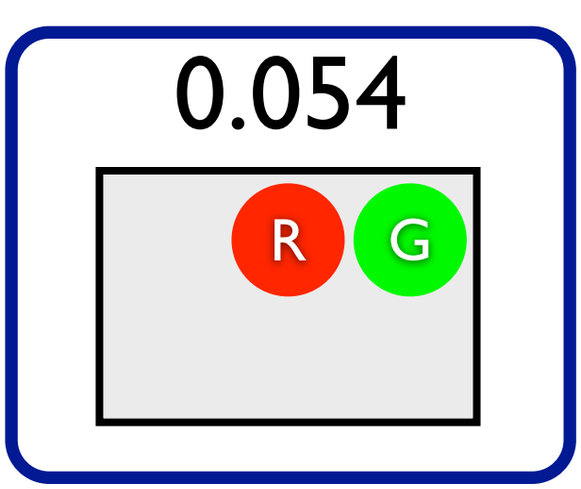


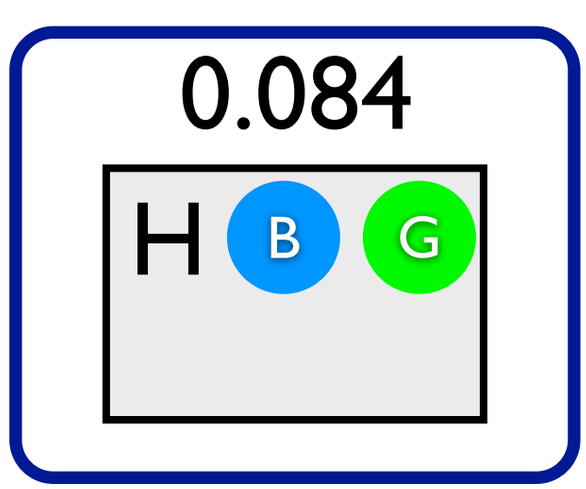


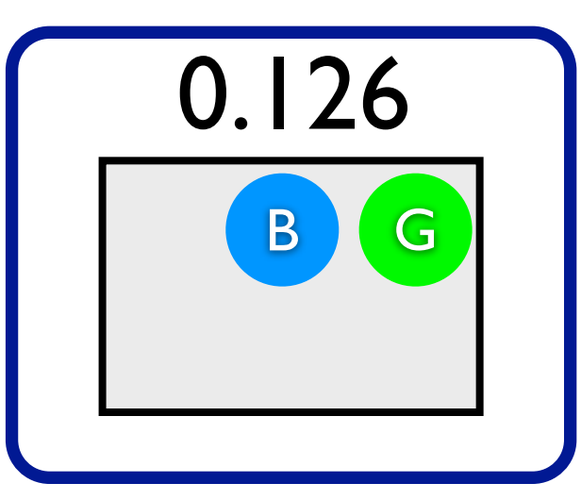


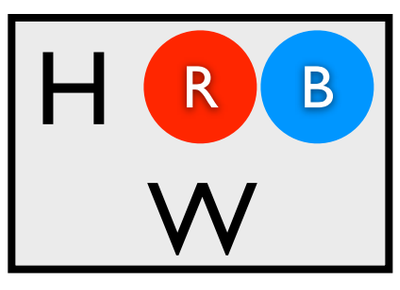


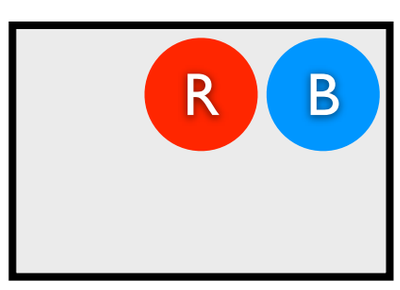


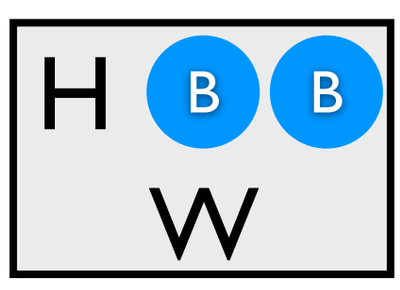


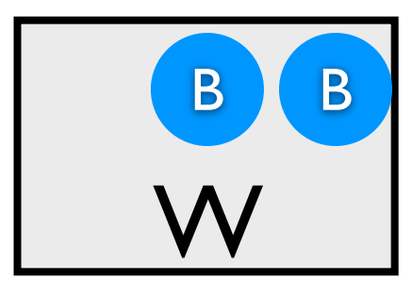












Distribution Semantics

(with probabilistic facts)

[Sato, ICLP 95]

query

sum over possible worlds where Q is true

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

subset of probabilistic facts

Prolog rules

probability of possible world

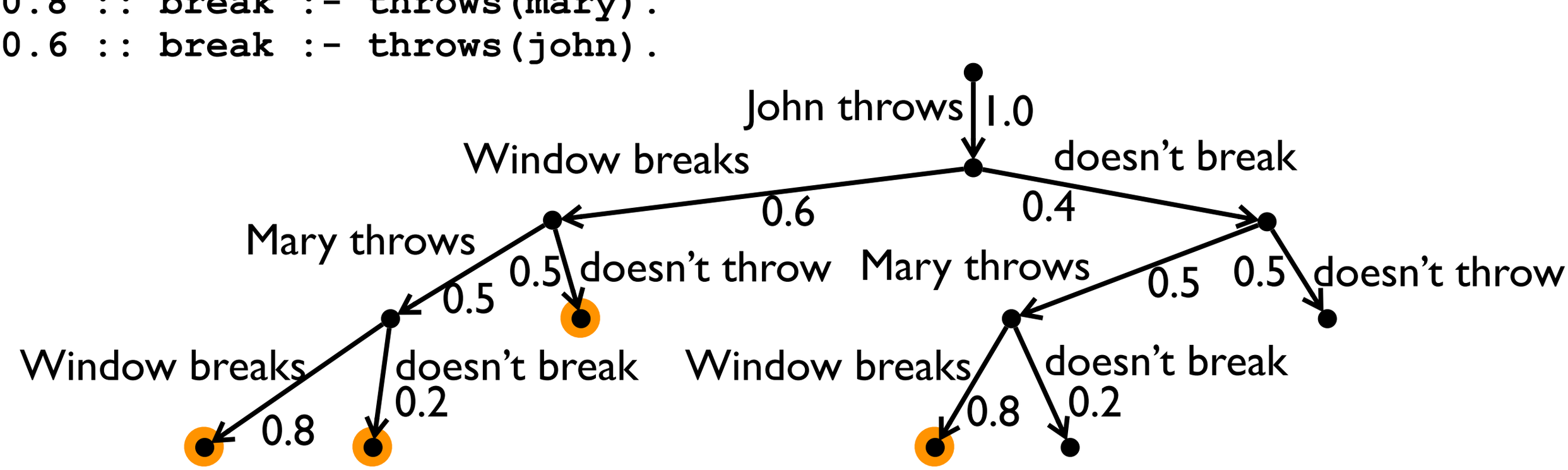
[Vennekens et al, ICLP 04]

Alternative view: CP-Logic

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

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

probabilistic causal laws



P(break)=0.6×0.5×0.8+0.6×0.5×0.2+0.6×0.5+0.4×0.5×0.8

CP-logic [Vennekens et al.]

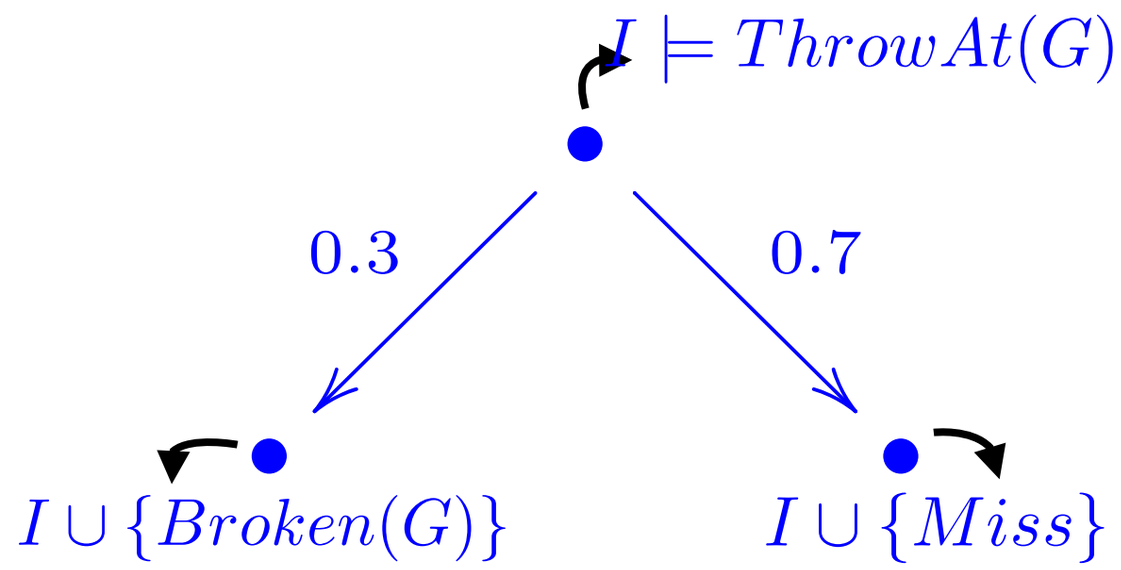
E.g., "throwing a rock at a glass breaks it with probability 0.3 and misses it with probability 0.7"

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

Note that the actual non-deterministic event ("rock flying at glass") is implicit

Slides CP-logic courtesy Joost Vennekens

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 same probability distribution over final states

Slides CP-logic courtesy Joost Vennekens

Distributional Clauses (DC)

- Discrete- and continuous-valued random variables

random variable with Gaussian distribution

```
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]

[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

tuple dealing with

probabilistic tables + database queries
→ distribution over possible worlds

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

one world

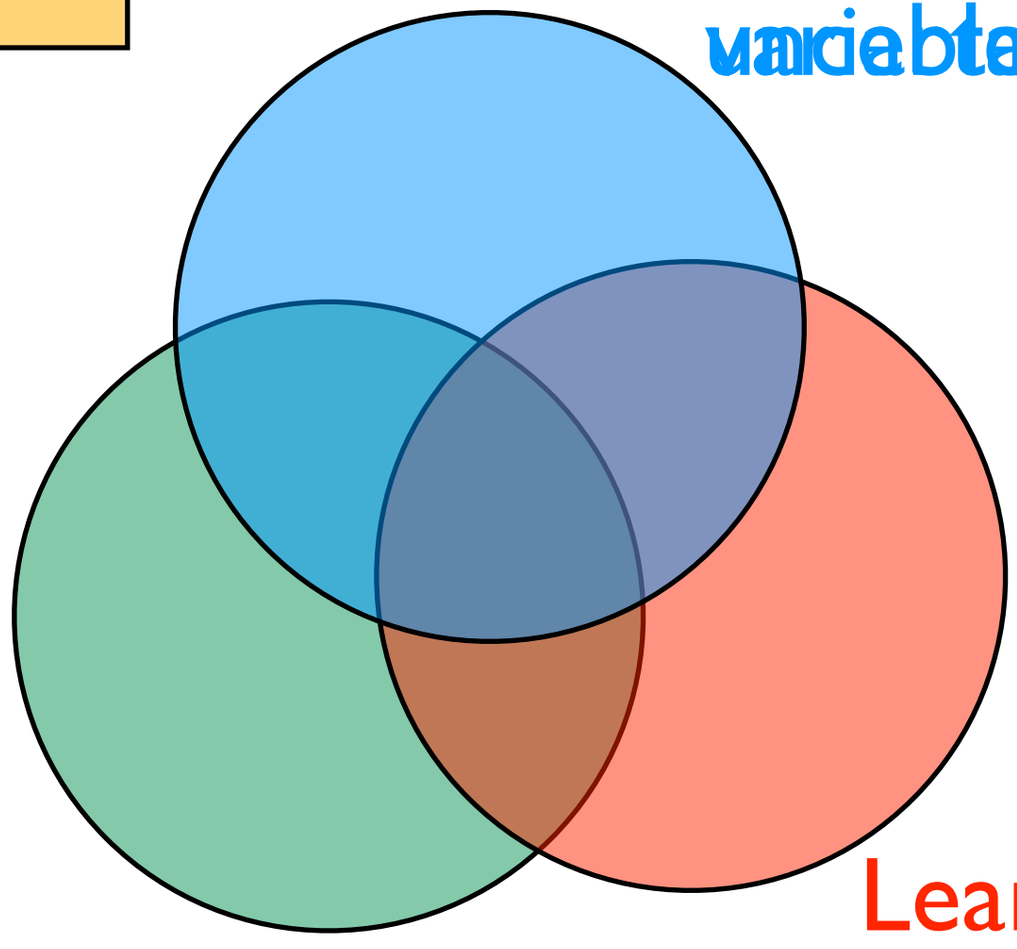
bornIn

person	city
ann	london
bob	york
eve	new york
tom	paris

cityIn

city	country
london	uk
york	uk
paris	usa

Reasoning with relational data



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
…	…	…

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

answer tuples ranked by probabilities handled implicitly

same query

0.96x0.99=0.95

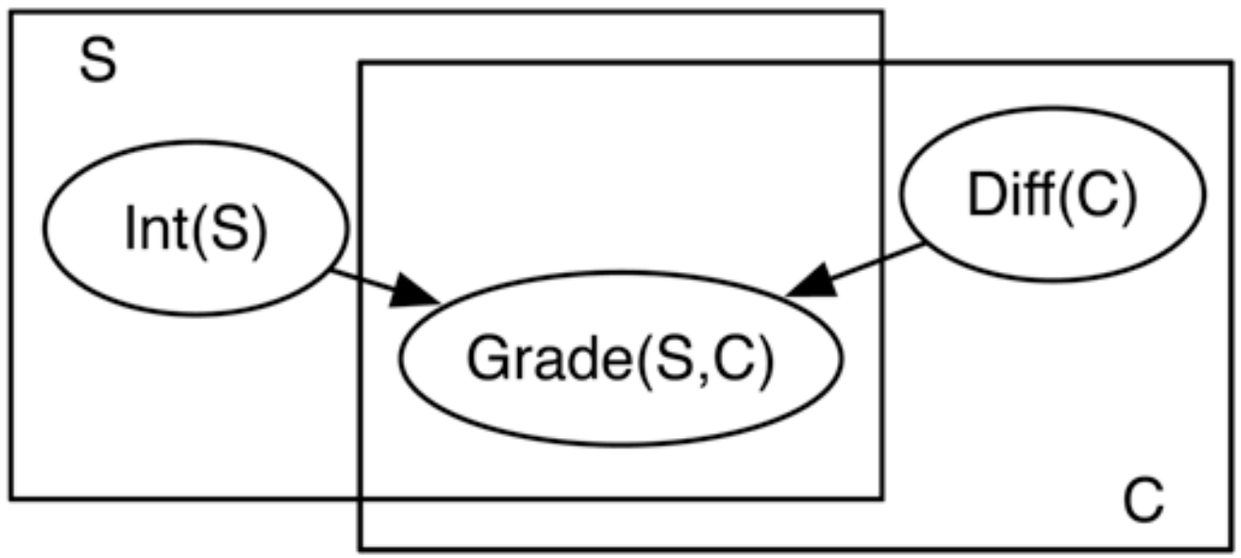
0.9x0.93=0.83

0.87x0.93=0.80

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

[Example from Suciu et al 2011]

Flexible and Compact Relational Model for Predicting Grades



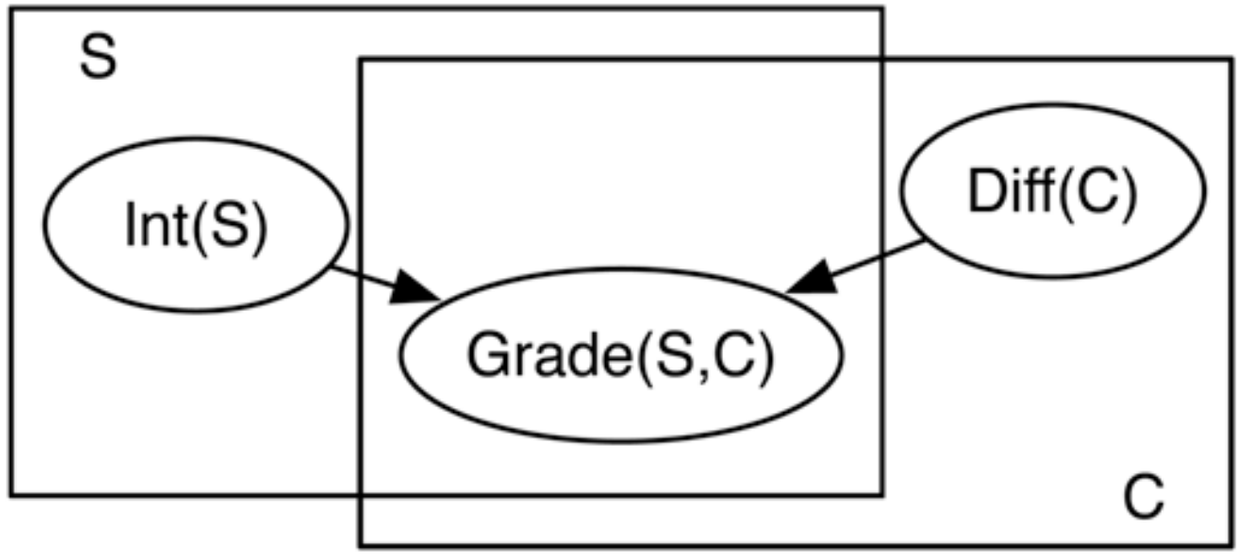
“Program” Abstraction:

- S, C logical variable representing students, courses
- the set of individuals of a type is called a population
- Int(S), Grade(S, C), D(C) are parametrized random variables

Grounding:

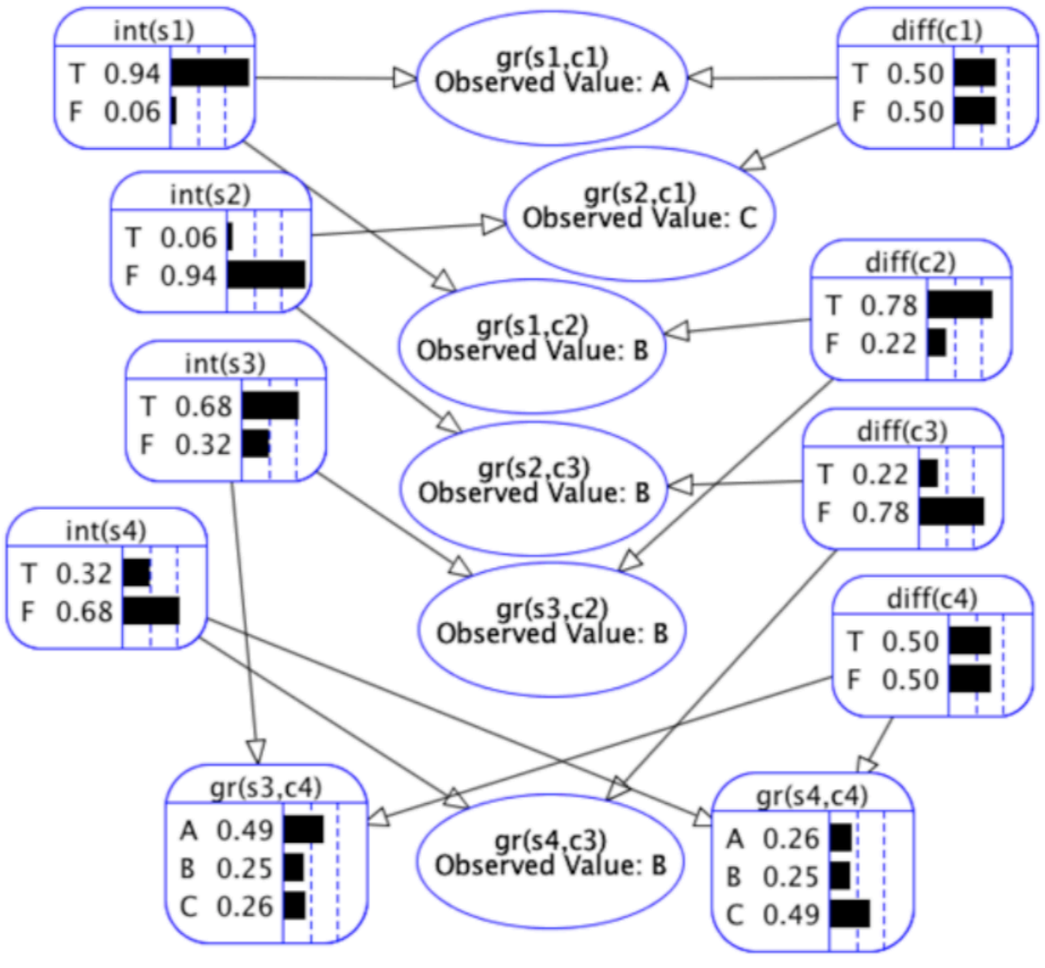
- for every student s, there is a random variable Int(s)
- for every course c, there is a random variable Di(c)
- for every s, c pair there is a random variable Grade(s,c)
- all instances share the same structure and parameters

ProbLog by example: Grading



Shows relational structure

grounded model: replace variables by constants



Works for any number of students / classes (for 1000 students and 100 classes, you get 101100 random variables); still only few parameters

With SRL / PP

build and learn compact models,

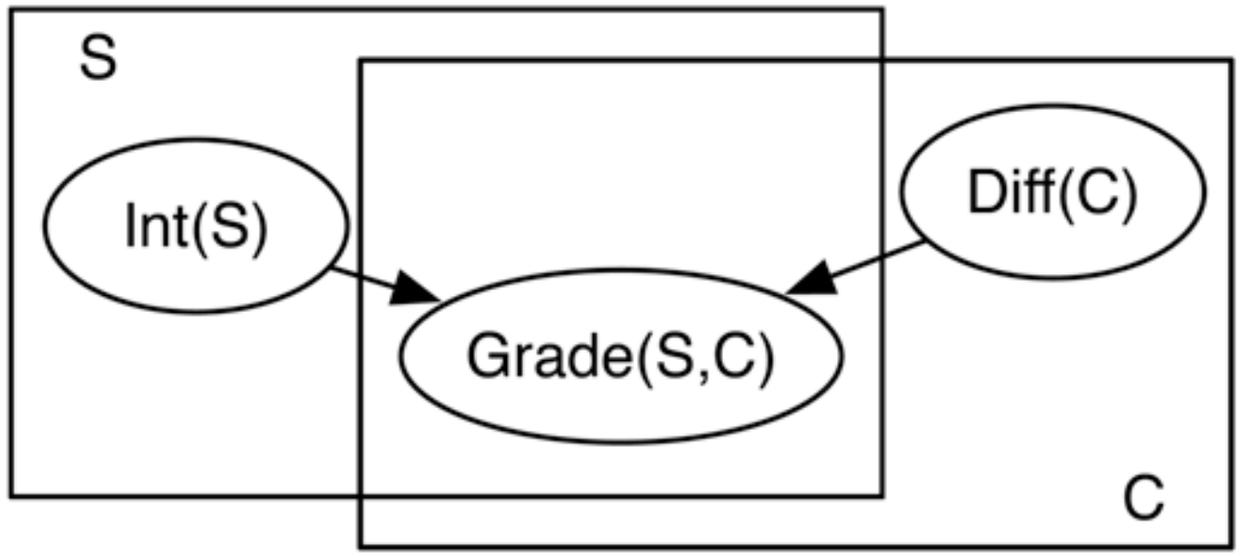
from one set of individuals - > other sets;

reason also about exchangeability,

build even more complex models,

incorporate background knowledge

ProbLog by example: Grading



Shows relational structure

grounded model: replace variables by constants

Works for any number of students / classes (for 1000 students and 100 classes, you get 101100 random variables); still only few parameters

With SRL / PP

build and learn compact models,

from one set of individuals - > other sets;

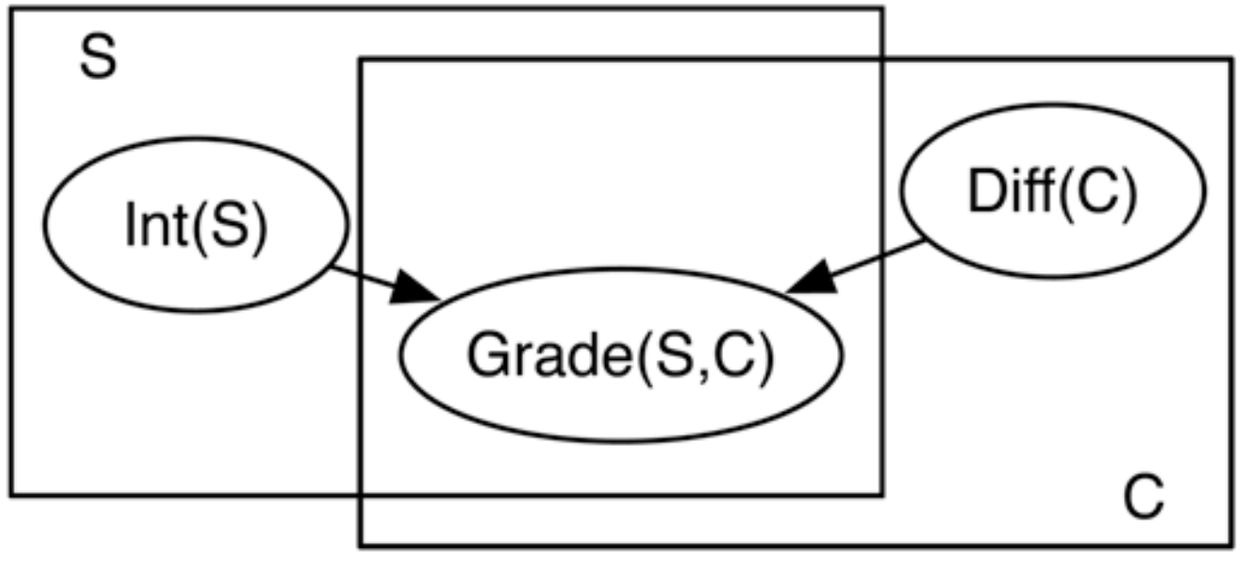
reason also about exchangeability,

build even more complex models,

incorporate background knowledge

Student	*Course*	*Grade*
s_1	c_1	A
s_2	c_1	C
s_1	c_2	B
s_2	c_3	B
s_3	c_2	B
s_4	c_3	B
s_3	c_4	?
s_4	c_4	?

ProbLog by example: Grading




```
0.4 :: int(S) :- student(S).
0.5 :: diff(C):- course(C).

student(john). student(anna). student(bob).
course(ai).    course(ml).    course(cs).

gr(S,C,a) :- int(S), not diff(C).
0.3::gr(S,C,a); 0.5::gr(S,C,b);0.2::gr(S,C,c) :-
            int(S), diff(C).
0.1::gr(S,C,b); 0.2::gr(S,C,c); 0.2::gr(S,C,f) :-
            student(S), course(C),
            not int(S), not diff(C).
0.3::gr(S,C,c); 0.2::gr(S,C,f) :-
            not int(S), diff(C).
```

ProbLog by example: Grading

```
unsatisfactory(S) :- student(S), grade(S,C,f).

excellent(S):- student(S), not(grade(S,C1,G),below(G,a)),
               grade(S,C2,a).

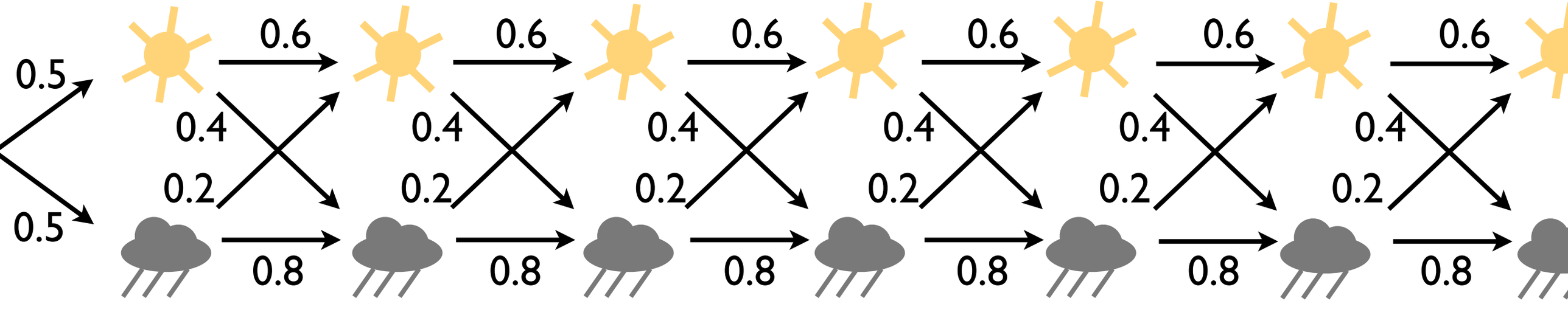
0.4 :: int(S) :- student(S).
0.5 :: diff(C):- course(C).

student(john). student(anna). student(bob).
course(ai).    course(ml).    course(cs).

gr(S,C,a) :- int(S), not diff(C).
0.3::gr(S,C,a); 0.5::gr(S,C,b);0.2::gr(S,C,c) :-
            int(S), diff(C).
0.1::gr(S,C,b); 0.2::gr(S,C,c); 0.2::gr(S,C,f) :-
            student(S), course(C),
            not int(S), not diff(C).
0.3::gr(S,C,c); 0.2::gr(S,C,f) :-
            not int(S), diff(C).
```

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

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

Concept: flexible probability

ProbLog by example:

Limited Luggage



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

```
weight(skis,6).
weight(boots,4).
weight(helmet,3).
weight(gloves,2).
P::pack(Item)    :- weight(Item,Weight), P is 1.0/Weight.
excess(Limit)    :- excess([skis,boots,helmet,gloves],Limit).

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

flexible probability:
computed from the weight of the item

list of all items

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

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

no items left: did we add too much?

ProbLog

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

Summary: ProbLog Syntax

- input database: ground facts

```
person(bob).
```

- probabilistic facts

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

- annotated disjunctions

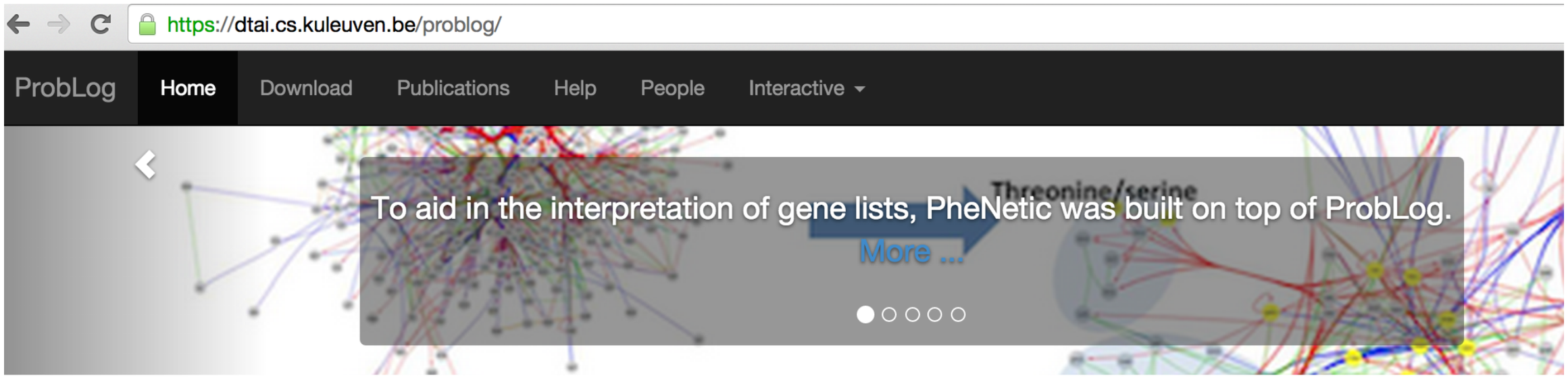
```
0.5::stress(X) :- person(X).
0.4::a(X); 0.3::b(X); 0.2::c(X); 0.1::d(X) :- q(X).
0.5::weather(sun,0) ; 0.5::weather(rain,0).
```

- 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).
```



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).
```

Why are we doing this ?

- Reasoning (Probability + Logic) AND Learning
- SRL : Expressive Probabilistic Graphical Models
  - First order logic results supports entities + relationships + background knowledge — abstraction of multiple entities
  - Recursion (e.g. smokers cannot be represented by a plate model)
- PP : Power of a universal Turing machine = a prog. language
  - you can program in it and have builtin expressive prob. models
  - PP can learn -> so bring learning to programming languages
- ProbLog fits both paradigms

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

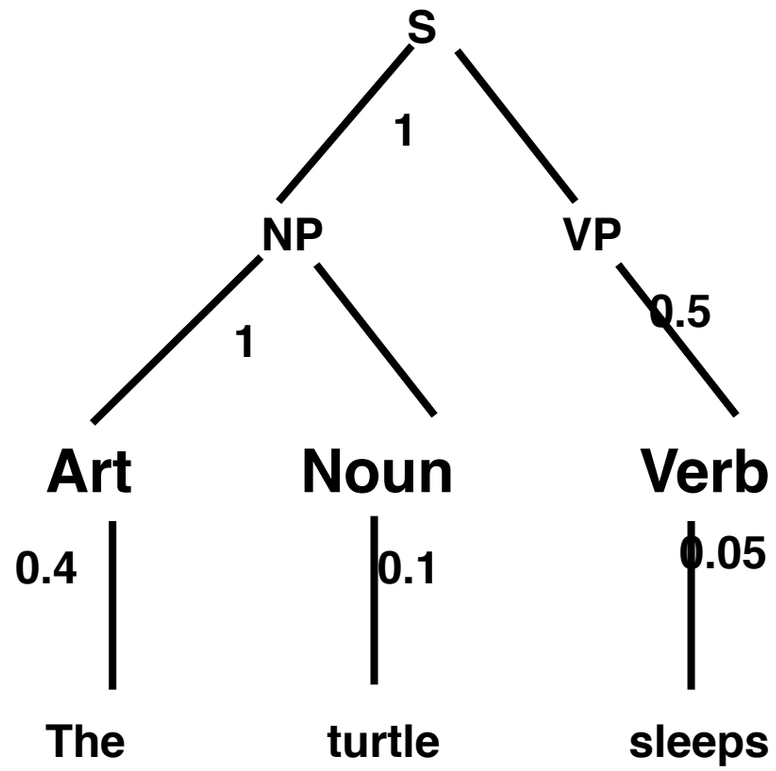
- Another probabilistic Prolog based on the distribution semantics
- Stochastic memoization : fresh result on each call
- Mutual exclusiveness assumption
  - allows for efficient inference by dynamic programming, cf. probabilistic grammars
  - but excludes certain models, e.g., smokers
- http://sato-www.cs.titech.ac.jp/prism/

Probabilistic Prologs: Two Views

- Distribution semantics:
  - probability distribution over interpretations
  - degree of belief
- Stochastic Logic Programs (SLPs):
  - probability distribution over query answers
  - like in probabilistic grammars

Probabilistic Context Free Grammars

1.0 : S -> NP, VP
1.0 : NP -> Art, Noun
0.6 : Art -> a
0.4 : Art -> the
0.1 : Noun -> turtle
0.1 : Noun -> turtles
…
0.5 : VP -> Verb
0.5 : VP -> Verb, NP
0.05 : Verb -> sleep
0.05 : Verb -> sleeps



P(parse tree) = 1x1x.5x.1x.4x.05

PCFGs

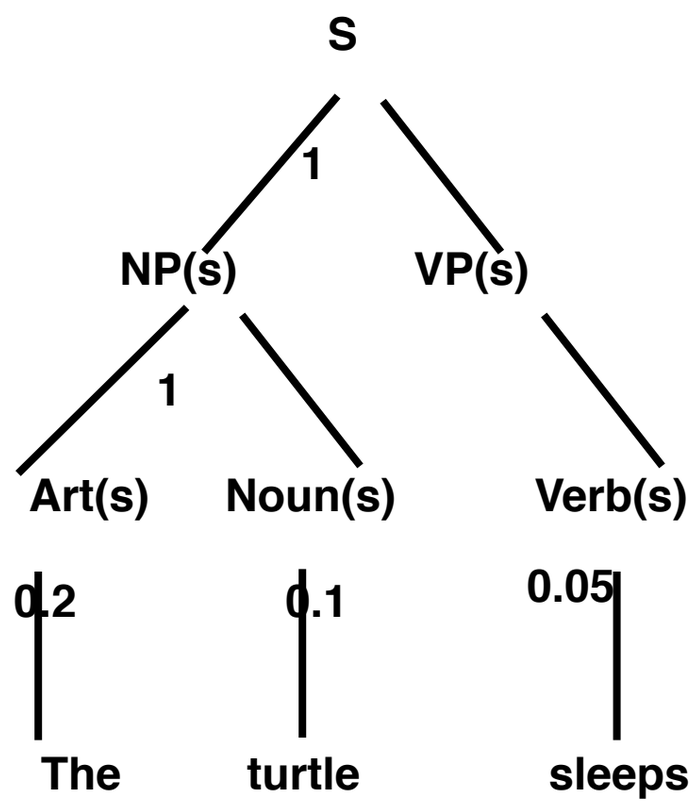
$P(parse\ tree) = \prod_i p_i^{c_i}$
where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

$P(sentence) = \sum_{p:parsetree} P(p)$

Observe that derivations always succeed, that is
$S \to T, Q$ and $T \to R, U$
always yields
$S \to R, U, Q$

Probabilistic Definite Clause Grammar

1.0 S -> NP(Num), VP(Num)
1.0 NP(Num) -> Art(Num), Noun(Num)
0.6 Art(sing) -> a
0.2 Art(sing) -> the
0.2 Art(plur) -> the
0.1 Noun(sing) -> turtle
0.1 Noun(plur) -> turtles
…
0.5 VP(Num) -> Verb(Num)
0.5 VP(Num) -> Verb(Num), NP(Num)
0.05 Verb(sing) -> sleeps
0.05 Verb(plur) -> sleep
….



P(derivation tree) = 1x1x.5x.1x .2 x.05

Stochastic Logic Programs

[Muggleton, Cussens]

In SLP notation

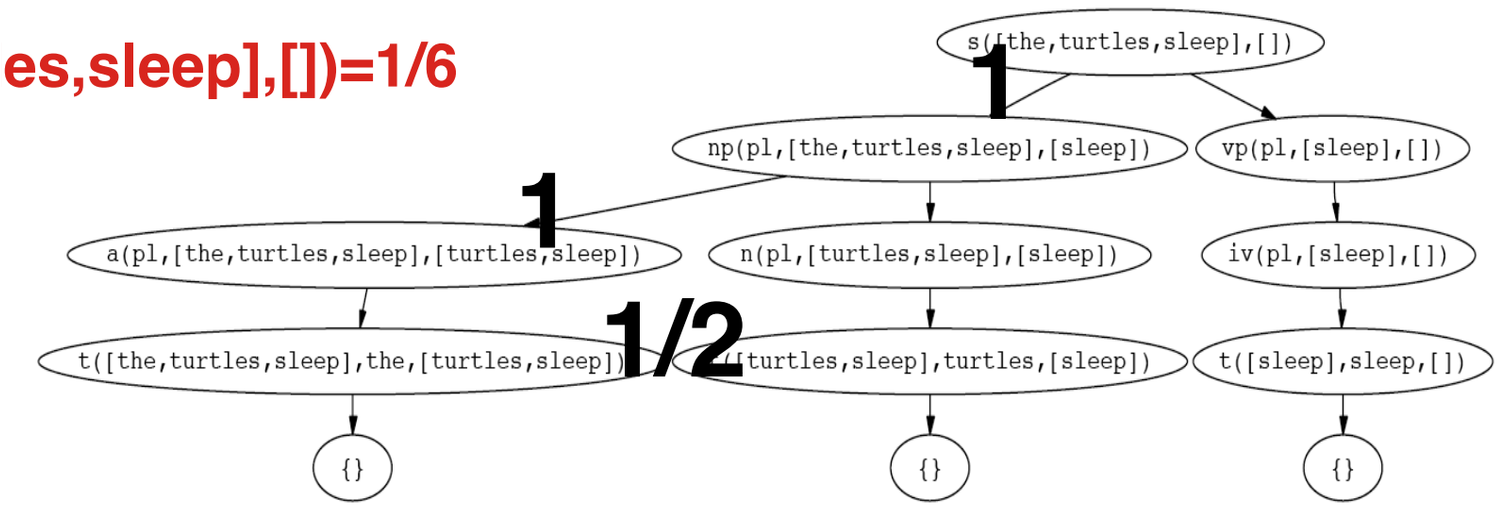
```
sentence(A, B) :- noun_phrase(C, A, D), verb_phrase(C, D, B).
noun_phrase(A, B, C) :- article(A, B, D), noun(A, D, C).
verb_phrase(A, B, C) :- intransitive_verb(A, B, C).
article(singular, A, B) :- terminal(A, a, B).
article(singular, A, B) :- terminal(A, the, B).
article(plural, A, B)   :- terminal(A, the, B).
noun(singular, A, B) :- terminal(A, turtle, B).
noun(plural, A, B)   :- terminal(A, turtles, B).
intransitive_verb(singular, A, B) :- terminal(A, sleeps, B).
intransitive_verb(plural, A, B) :- terminal(A, sleep, B).
terminal([A|B],A,B).
```

1

1/3

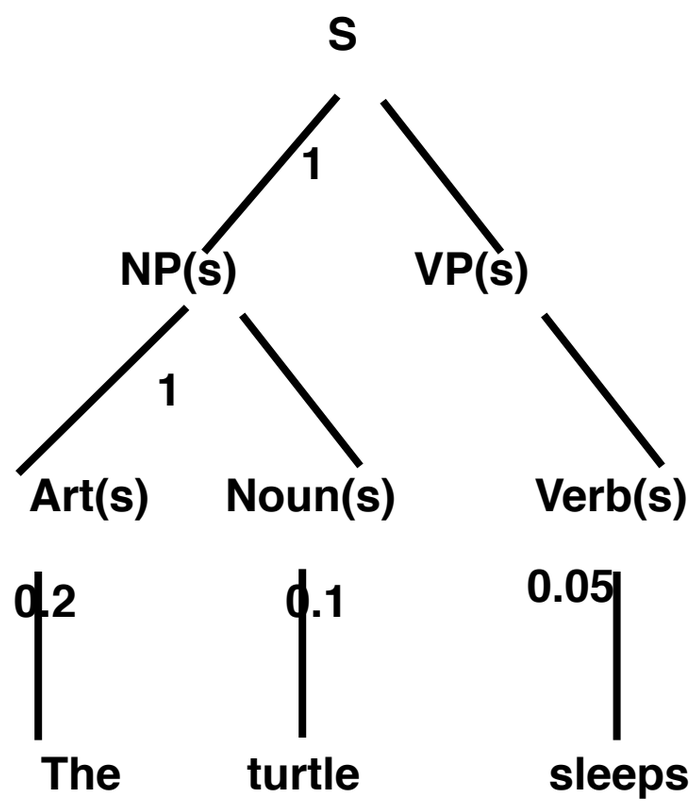
1/2

P(s([the,turtles,sleep],[])=1/6



Probabilistic DCG

1.0 S -> NP(Num), VP(Num)
1.0 NP(Num) -> Art(Num), Noun(Num)
0.6 Art(sing) -> a
0.2 Art(sing) -> the
0.2 Art(plur) -> the
0.1 Noun(sing) -> turtle
0.1 Noun(plur) -> turtles
…
0.5 VP(Num) -> Verb(Num)
0.5 VP(Num) -> Verb(Num), NP(Num)
0.05 Verb(sing) -> sleeps
0.05 Verb(plur) -> sleep
….



P(derivation tree) = 1x1x.5x.1x .2 x.05

What about "A turtles sleeps" ?

SLPs

$P_d(derivation) = \prod_i p_i^{c_i}$
where p_i is the probability of rule i
and c_i the number of times
it is used in the parse tree

Observe that some derivations now fail due to unification,
$np(Num) \rightarrow art(Num), noun(Num)$ and $art(sing) \rightarrow a$
$noun(plural) \rightarrow turtles$

Normalization necessary
$P_s(proof) = \frac{P_d(proof)}{\sum_i P_d(proof_i)}$

Probabilistic Programming Languages outside LP

- IBAL [Pfeffer 01]
- Figaro [Pfeffer 09]
- Church [Goodman et al 08]
- BLOG [Milch et al 05]
- Stan & Edward & Anglican
- and many more appearing recently such

Church

probabilistic functional programming

[Goodman et al, UAI 08]

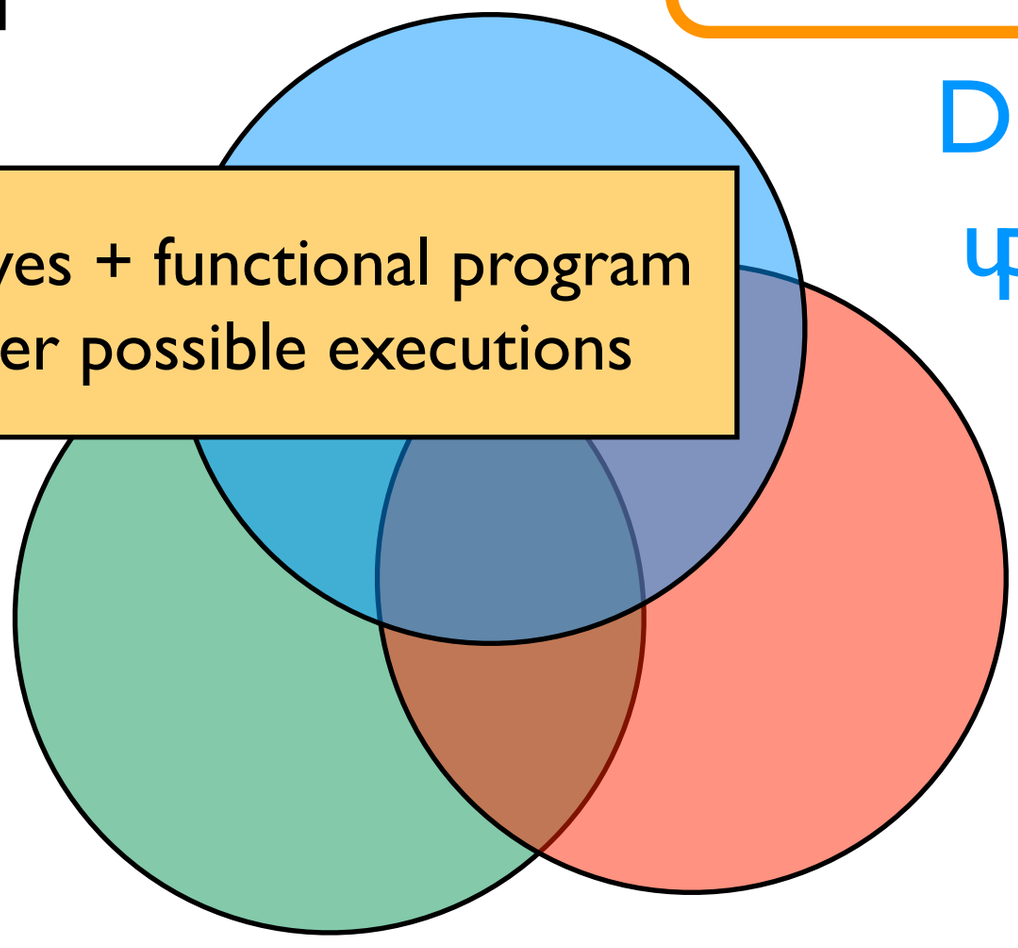
several possible executions

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

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

Dealing with uncertainty

probabilistic primitives + functional program
→ distribution over possible executions



Reasoning with logical data

Learning

one execution

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

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

Church vs ProbLog

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

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

Church result: (1 2) with 0.4×0.4
(1 7) with 0.4×0.6
(6 2) with 0.6×0.4
(6 7) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) :- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
     p5(N,M),
     lp5(L,K).

query(lp5([1,2],_)).
```

ProbLog result: (1 2) with 0.4×0.4
(1 7) with 0.4×0.6
(6 2) with 0.6×0.4
(6 7) with 0.6×0.6

results for [1,1]?

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

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

Church result: (1 1) with 0.4×0.4
(1 6) with 0.4×0.6
(6 1) with 0.6×0.4
(6 6) with 0.6×0.6

```
0.4::p5(N,N);0.6::p5(N,M) :- M is N+5.

lp5([],[]).
lp5([N|L],[M|K]) :-
     p5(N,M),
     lp5(L,K).

query(lp5([1,1],_)).
```

ProbLog result: (1 1) with 0.4
(1 6) with 0.0
(6 1) with 0.0
(6 6) with 0.6

stochastic memoization

Solution

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

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

```
0.4::p5(N,N,ID);0.6::p5(N,M,ID) :- M is N+5.
lp5([],[]).
lp5([N|L],[M|K]) :-
     p5(N,M,L),
     lp5(L,K).

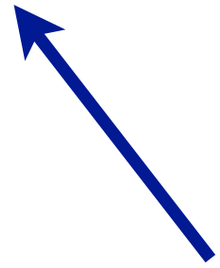
query(lp5([1,1],_)).
```

identifier distinguishes calls

Stochastic Memoization

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

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

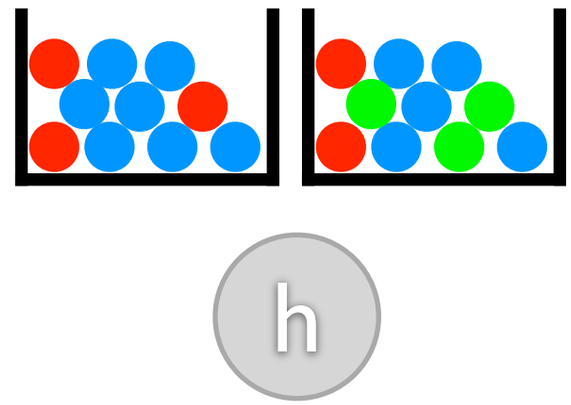


remember first value &
reuse for all later calls

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)

```
(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))
```

Probabilistic Programming Summary

- Church: functional programming + random primitives
- probabilistic generative model
- stochastic memoization
- sampling
- many 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

Probabilistic Programming Ingredients

inject random variables in programming language

- random var = simplest boolean concept in language
- extensions for continuous variables exist

define semantics :

- possible worlds for probabilistic logics / databases / StarAI
- trace-based semantics for functional / imperative languages
  - define probability of execution, whenever random variable encountered -> multiply by probabilty

inference to answer (conditional) queries

learning

- Bayesian inference : conditioning on observations and querying for distributions (typical for functional and imperative languages)
- Expected Maximisation (typical for Probabilistic Logics & StarAI)

see also [Russell, CACM 15]

Roadmap

- Prob. Programming - Modeling
- Inference
- Learning
- Dynamics
- Logic, Probability and Neural Nets
- KBMC & Markov Logic
- Lifted Inference

... with some detours on the way

PART II : Inference

Inference / Reasoning

- Most of the work in PP and StarAI is on inference
  - It is hard (complexity wise)
    - Many inference methods
      - exact, approximate, sampling and lifted …
- Inference is the key to learning

Two Steps

- **Logical inference** -
  - about a ground logical theory
    - proofs or model theoretic …
      - *Result: Weighted Model Counting problem*
- **Probabilistic propositional inference** —
  - Knowledge Compilation
  - Backtracking search — DPLL, VE, RC based
- **Advanced** — lifted inference

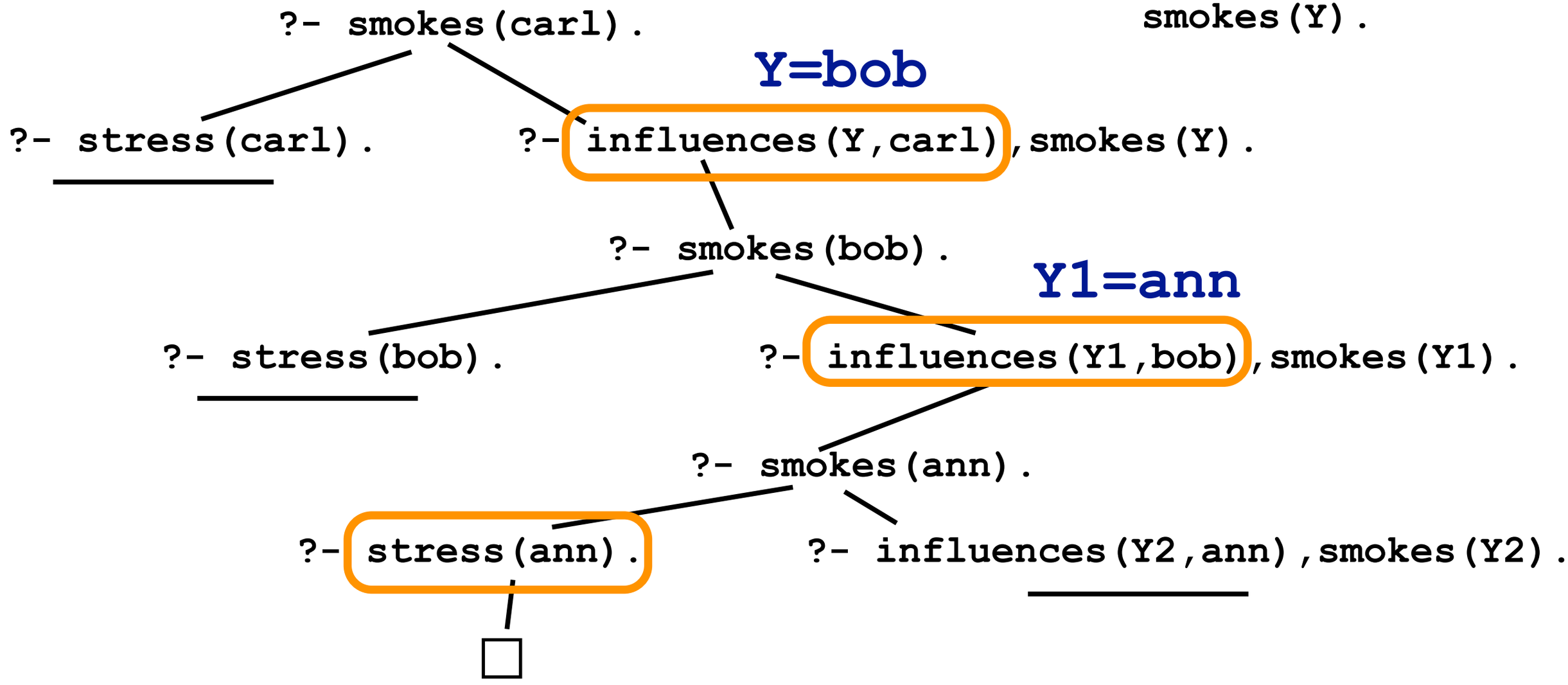
Two Steps

- **Logical inference** -
  - about a ground logical theory
    - proofs or model theoretic …
      - *Result: Weighted Model Counting problem*
- **Probabilistic propositional inference** —
  - Knowledge Compilation
  - Backtracking search — DPLL, VE, RC based
- **Advanced** — lifted inference

Logical Reasoning: Proofs in Prolog

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

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



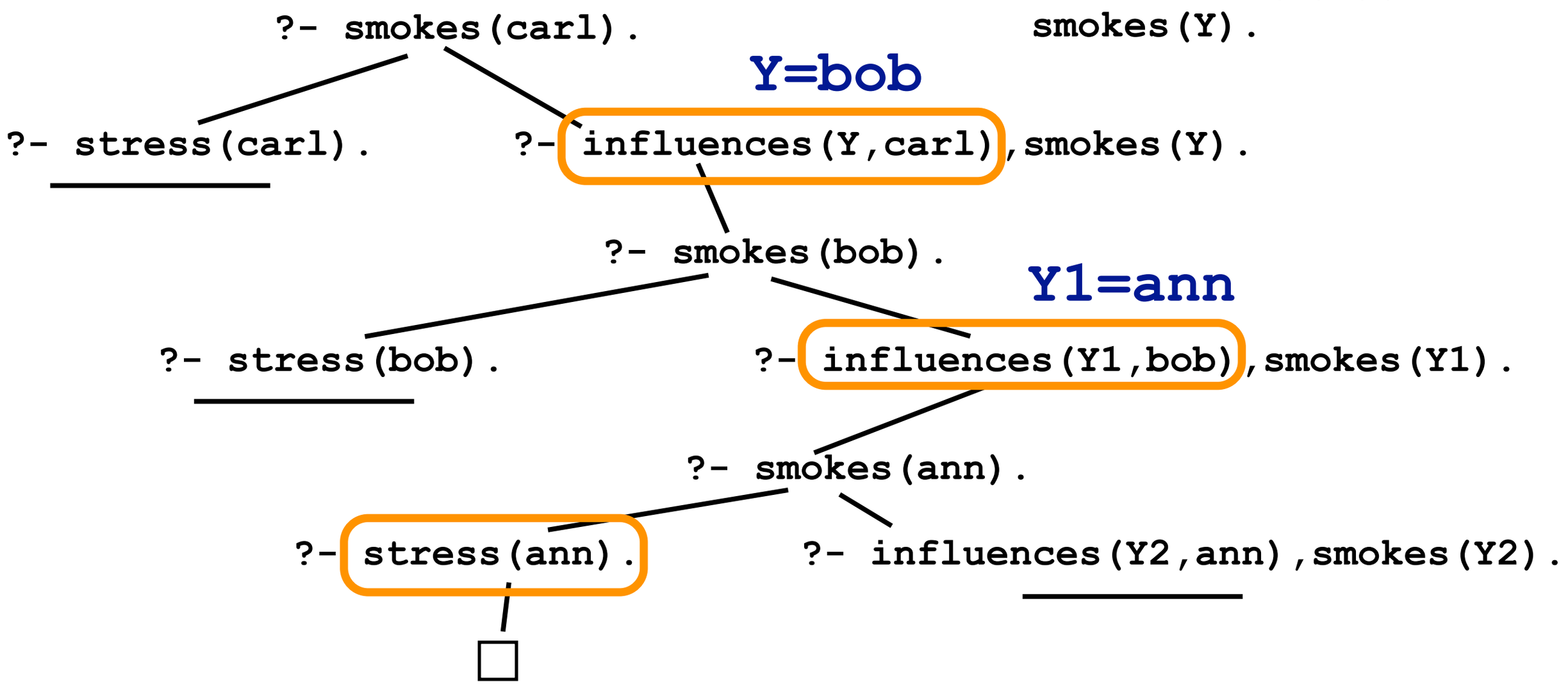
proof = facts used in successful derivation:

`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).
```



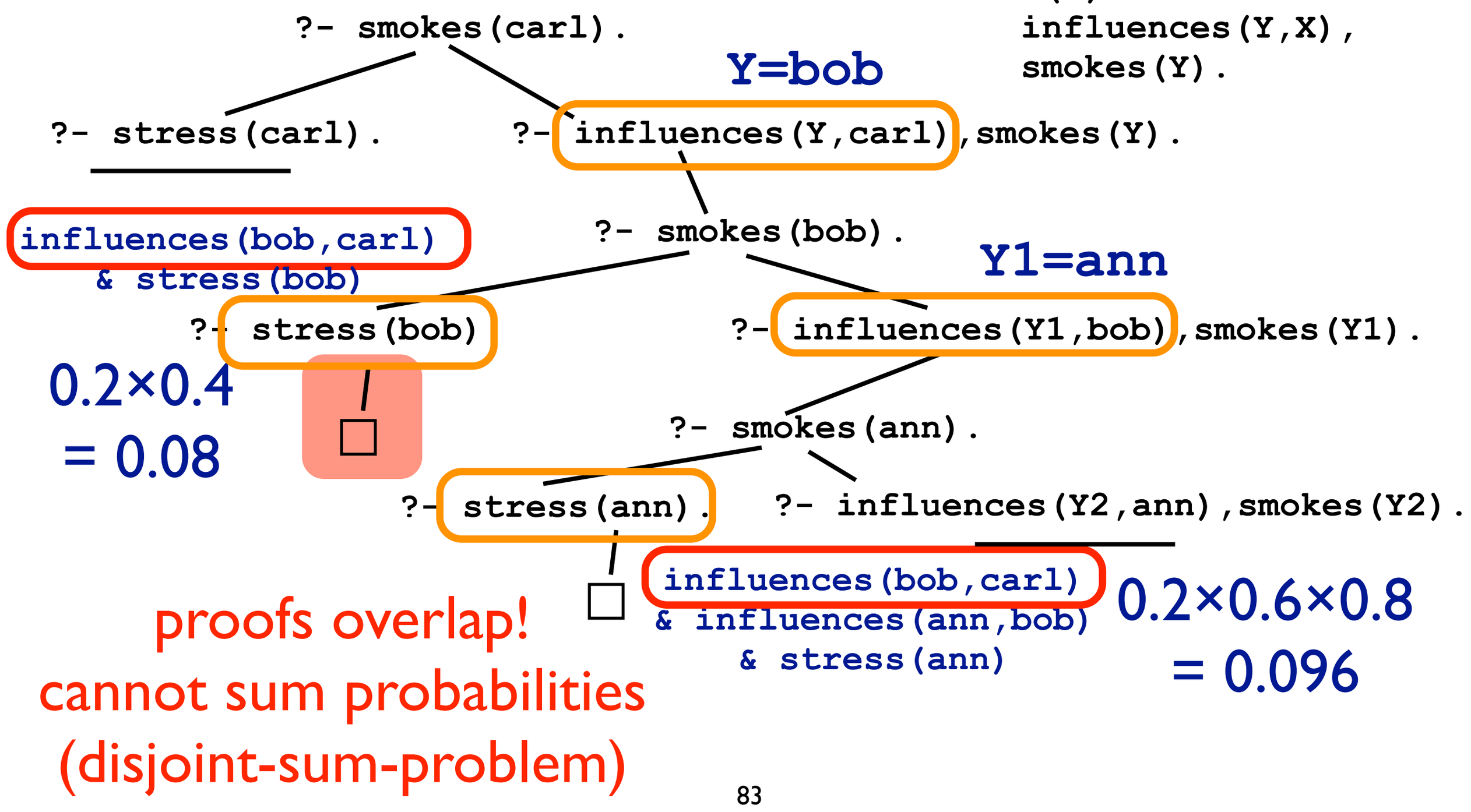
`influences(bob,carl)&influences(ann,bob)&stress(ann)`

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

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

`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

Complexity of querying in probabilistic databases

- queries have fixed size (no recursion)
- size of query << size of database
- complexity of evaluating given query measured in size of database (data complexity)
- standard relational database queries: polynomial

$$P(\mathtt{q}) = 1 - \prod_{j=1..n} (1 - \left(P(\mathtt{s}(\mathtt{X_j}))(1 - \prod_{k=1..n, k \neq j} (1 - P(\mathtt{i}(\mathtt{X_j}, \mathtt{Y_k})))) \right))$$

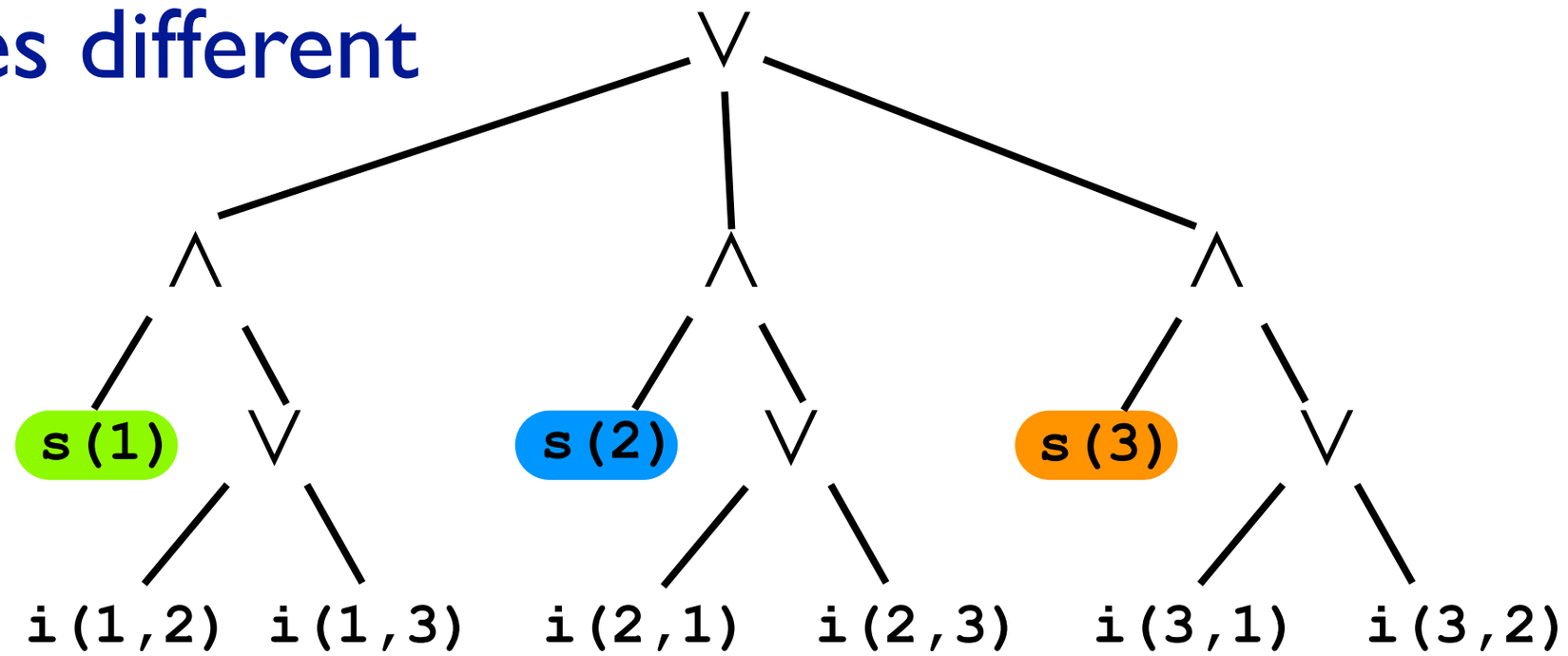
polynomial in database size / number of persons n

proofs

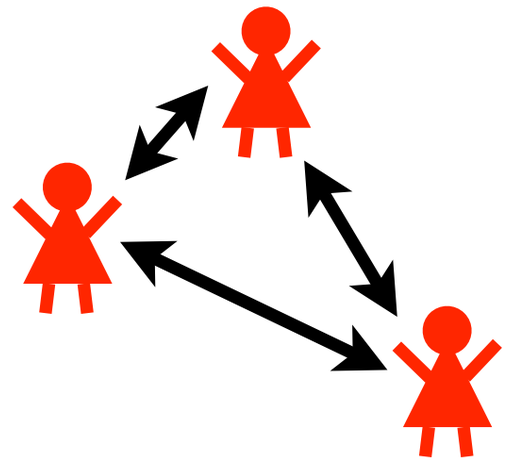
- s(1),i(1,2)
- s(1),i(1,3)
- s(2),i(2,1)
- s(2),i(2,3)
- s(3),i(3,1)
- s(3),i(3,2)

tree structure, all leaves different

$$P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \cdot P(\varphi_2)$$
$$P(\varphi_1 \vee \varphi_2) = 1 - (1 - P(\varphi_1)) \cdot (1 - P(\varphi_2))$$



How hard is evaluating q?



```
q2 :- stress(X), influences(X,Y), male(Y).
```

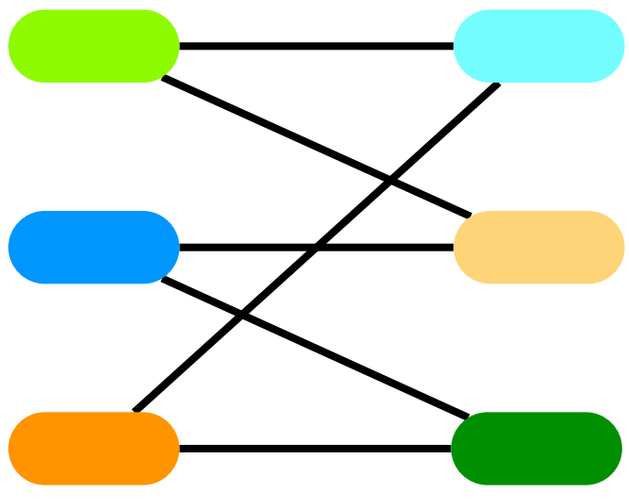
```
0.3::male(1).
0.8::male(2).
0.9::male(3).
```

```
0.7::stress(1).
0.4::stress(2).
0.9::stress(3).
```

```
0.83::influences(1,2).
0.41::influences(1,3).
...
0.17::influences(3,2).
```

proofs

```
s(1),i(1,2),m(2)
s(1),i(1,3),m(3)
s(2),i(2,1),m(1)
s(2),i(2,3),m(3)
s(3),i(3,1),m(1)
s(3),i(3,2),m(2)
```



cannot build tree structure with all leaves different

Dichotomy of UCQ Evaluation

- **U**nion of **C**onjunctive **Q**ueries
  ≈ Datalog without recursion and negation
- Theorem: UCQ evaluation is either polynomial in database size or #P-hard



counting version of NP decision problems, e.g., model counting

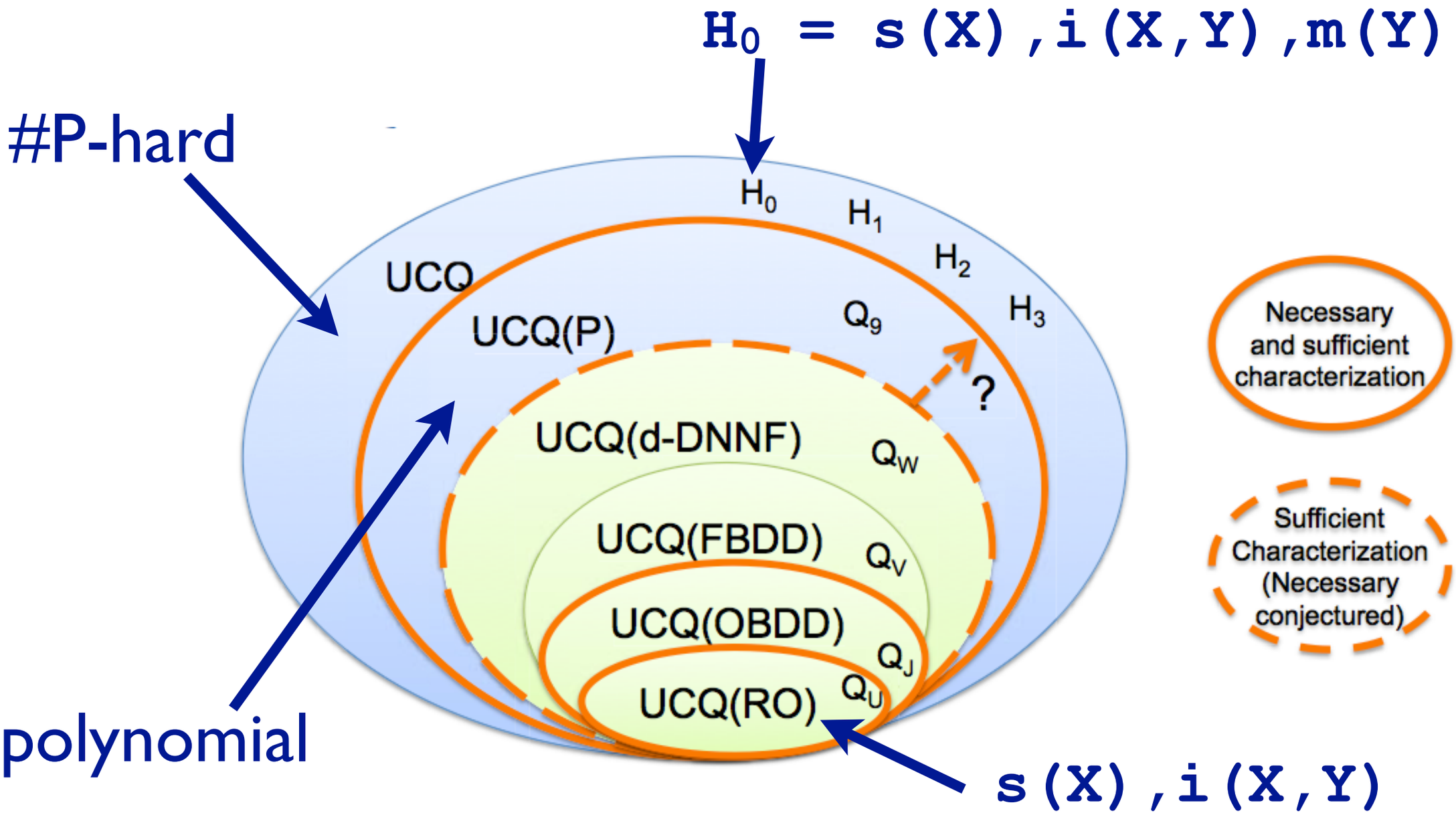


Figure 5.4: The query compilation hierarchy for Unions of Conjunctive Queries (UCQ).

Fig. from [Suciu et al 2011]

Two Steps

- **Logical inference** -
  - about a ground logical theory
    - proofs or model theoretic …
      - *Result: Weighted Model Counting problem*
- **Probabilistic propositional inference** —
  - Knowledge Compilation
  - Backtracking search — DPLL, VE, RC based
- **Advanced** — lifted inference

Disjoint-Sum-Problem

possible worlds

solution: knowledge compilation

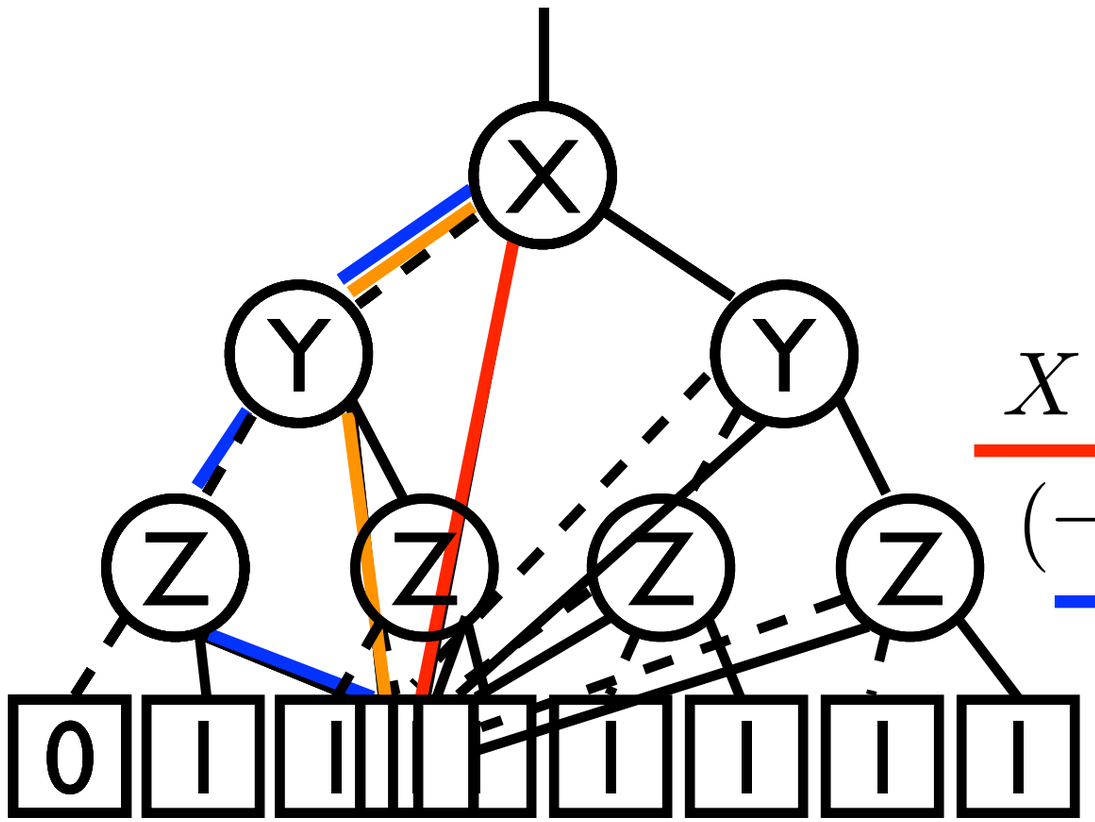
`infl(bob,carl) & infl(ann,bob) & st(ann) & \+st(bob)`	0.0576
`infl(bob,carl) & infl(ann,bob) & st(ann) & st(bob)`	0.0384
`infl(bob,carl) & \+infl(ann,bob) & st(ann) & st(bob)`	0.0256
`infl(bob,carl) & infl(ann,bob) & \+st(ann) & st(bob)`	0.0096
`infl(bob,carl) & \+infl(ann,bob) & \+st(ann) & st(bob)`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

sum of proof probabilities: 0.096+0.08 = 0.1760

Binary Decision Diagrams
[Bryant 86]

- compact graphical representation of Boolean formula
- popular in many branches of CS
- automatically disjoins proofs
  → efficient probability computation

Binary Decision Diagrams
[Bryant 86]



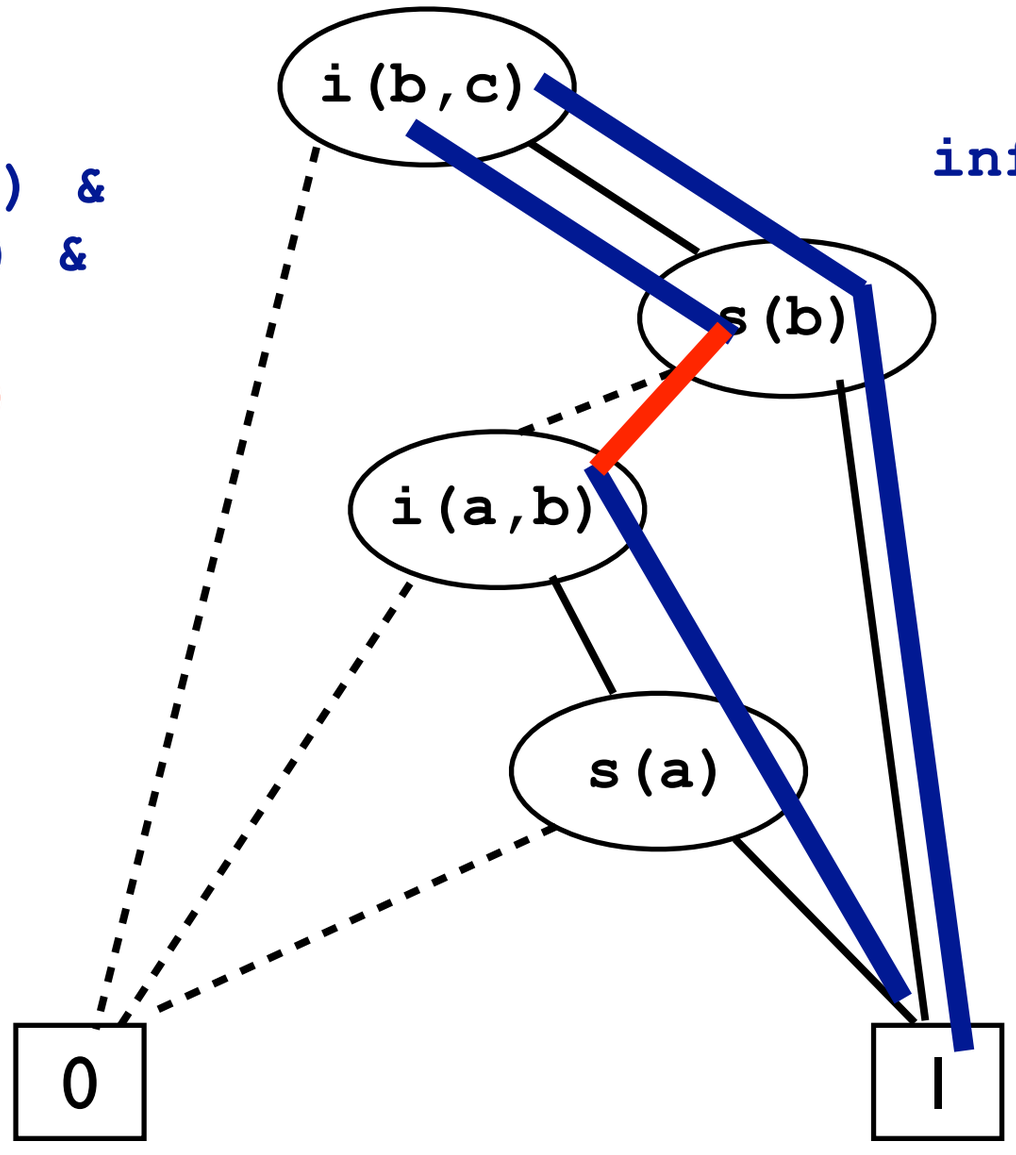
$X \vee (\neg X \wedge Y) \vee (\neg X \wedge \neg Y \wedge Z)$

$X \vee Y \vee Z$

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]

influences(bob,carl) &
influences(ann,bob) &
stress(ann)
& not stress(bob)



influences(bob,carl) &
stress(bob)

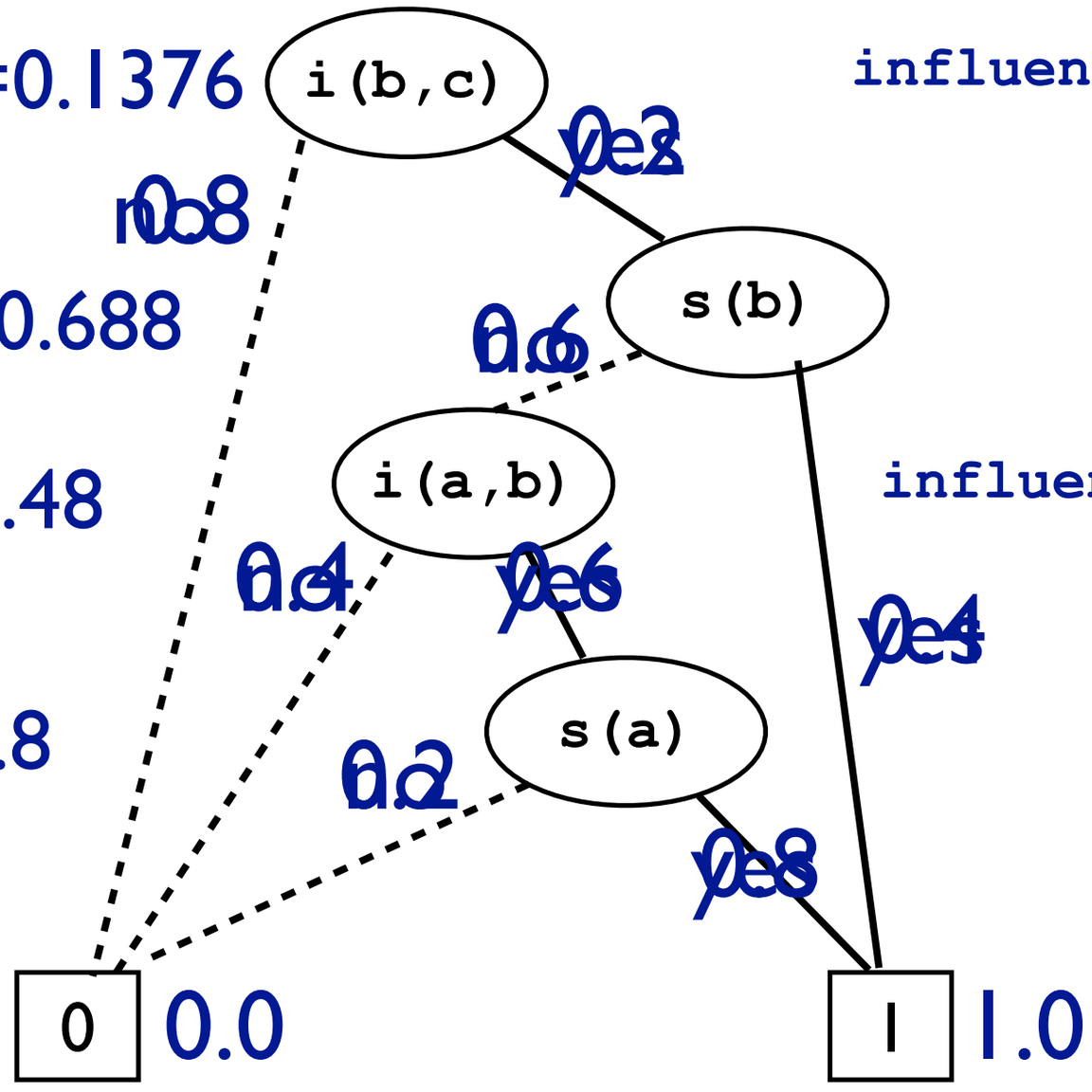
Binary Decision Diagrams

0.8×0.0+0.2×0.688=0.1376

0.6×0.48+0.4×1.0=0.688

0.4×0.0+0.6×0.8=0.48

0.2×0.0+0.8×1.0=0.8



`influences(bob,carl)?`

`stress(bob)?`

`influences(ann,bob)?`

`stress(ann)?`

probability of

`smokes(c)?`

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

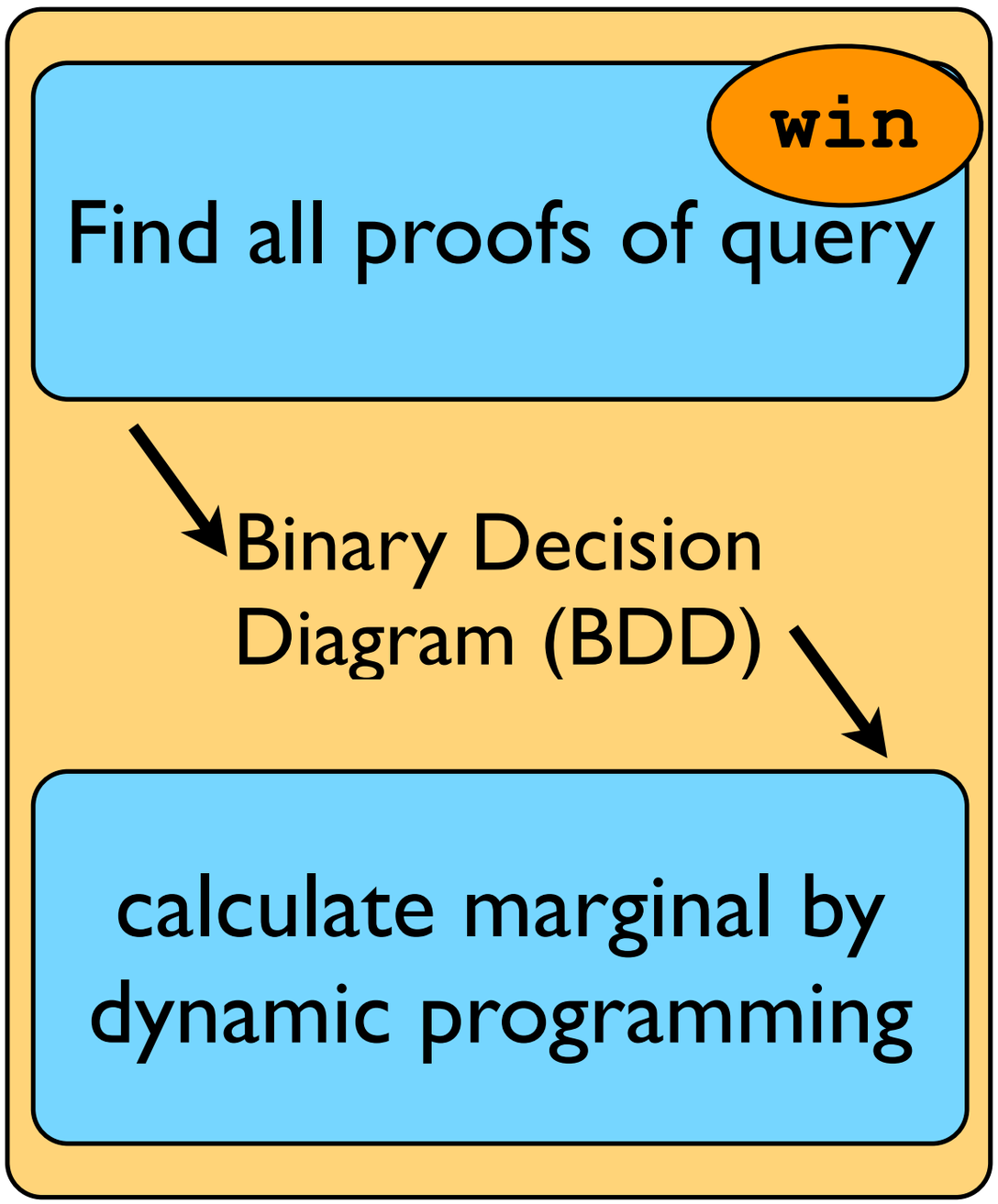
Knowledge Compilation

- Compile once - query many times …
- The knowledge compilation map (Darwiche and Marquis, JAIR 2002)
- Different representations, computational complexity of different operations, … (and, or, condition, ….)
- in StarAI — OBDDs, d-DNNFs, SDD (Darwiche), …
- Also state of the art for Bayesian net inference.

Initial Approach

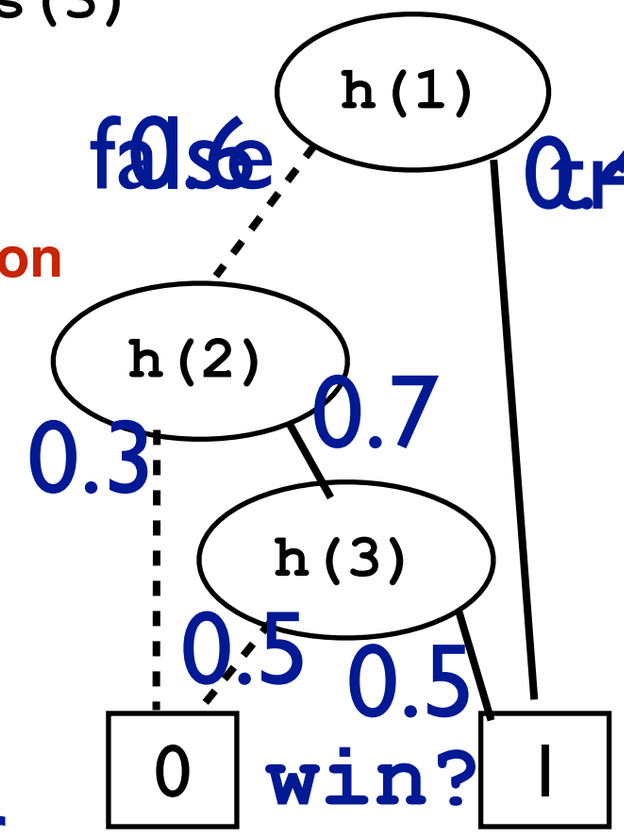
(ProbLog1 & others)

```
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)
```

Proofs + Knowledge Compilation



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

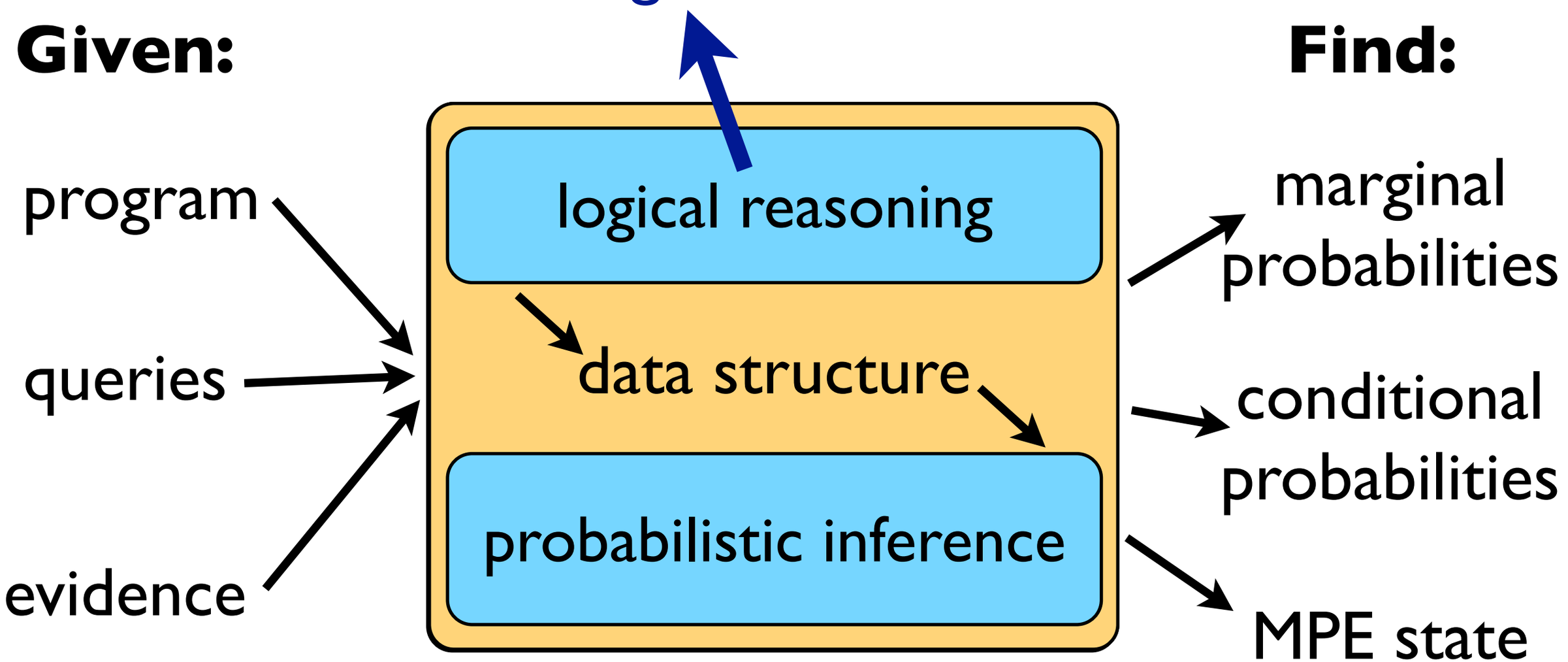
 [De Raedt et al, IJCAI 07; Kimmig et al, TPLP 11]

Answering Questions

1. using proofs
2. using models

Given:

Find:



Logical Reasoning:
Find a model (SAT)
Here : 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).
```

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

→ weighted model counting

ProbLog → CNF

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

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

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

- Find relevant ground rules by backward reasoning

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

- Convert to propositional logic formula

may require loop-breaking

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

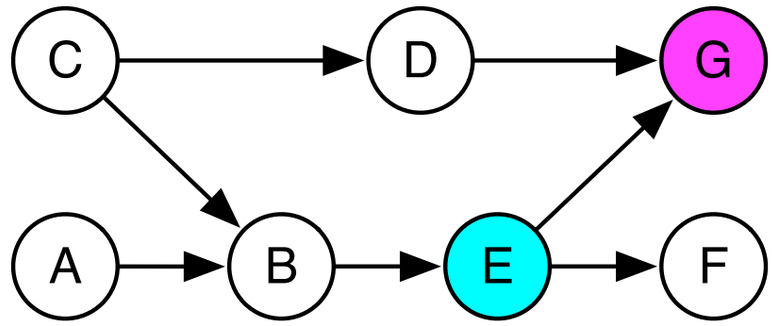
- Rewrite in CNF (as usual)

Two Steps

- **Logical inference** -
  - about a ground logical theory
    - proofs or model theoretic …
      - *Result: Weighted Model Counting problem*
- **Probabilistic propositional inference** —
  - Knowledge Compilation
  - Backtracking search — DPLL, VE, RC based
- **Advanced** — lifted inference

Inference in Bayesian Networks

$$P(E \mid g) = \frac{P(E \wedge g)}{\sum_E P(E \wedge g)}$$



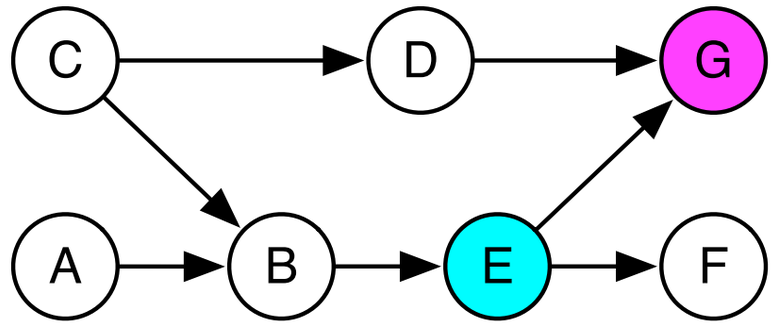
$$P(E \wedge g) = \sum_{ABCDF} P(ABCDEFg)$$

$$= \sum_F \sum_B \sum_C \sum_A \sum_D P(A)P(B \mid AC)P(C)P(D \mid C)$$
$$P(E \mid B)P(F \mid E)P(g \mid ED)$$

$$= \left(\sum_F P(F \mid E)\right)$$
$$\sum_B P(E \mid B) \sum_C \left(P(C) \left(\sum_A P(A)P(B \mid AC)\right)\right.$$
$$\left.\left(\sum_D P(D \mid C)P(g \mid ED)\right)\right)$$

Inference in Bayesian Networks

$$P(E \mid g) = \frac{P(E \wedge g)}{\sum_E P(E \wedge g)}$$



$$P(E \wedge g) = \sum_{ABCDEF} P(ABCDEFg)$$

SUM

$$= \sum_F \sum_B \sum_C \sum_A \sum_D P(A)P(B \mid AC)P(C)P(D \mid C)$$

PRODUCT

$$P(E \mid B)P(F \mid E)P(g \mid ED)$$

$$= \left(\sum_F P(F \mid E)\right)$$

$$\sum_B P(E \mid B) \sum_C \left(P(C)\left(\sum_A P(A)P(B \mid AC)\right)\right.$$

$$\left.\left(\sum_D P(D \mid C)P(g \mid ED)\right)\right)$$

SAT

Satisfiability of CNF formula

$$(A \vee B) \wedge (\neg B \vee C)$$

$$\bigvee_{A,B,C \in \{true,false\}} (A \vee B) \wedge (\neg B \vee C)$$

$$\bigvee_{A,B,C \in \{true,false\}} c_1(A,B) \wedge c_2(B,C)$$

$$\bigvee_{A,B,C \in \{true,false\}} \bigwedge_i c_i(A,B,C)$$

#SAT

Counting the number of satisfying assignments

$$\sum_{A,B,C \in \{true,false\}} true?(A \vee B) \times true?(B \vee C)$$

$$\sum_{A,B,C \in \{true,false\}} \prod_i c_i(A,B,C)$$

#SAT

Counting the number of satisfying assignments

$$\sum_{A,B,C \in \{true,false\}} true?(A \vee B) \times true?(B \vee C)$$

$$\sum_{A,B,C \in \{true,false\}} \prod_i c_i(A,B,C)$$

WMC

Counting the number of satisfying assignments
while every assignment is getting a weight

$$\sum_{A,B,C \in \{true,false\}} true?(A \vee B) \times true?(B \vee C)$$

$$\sum_{A,B,C \in \{true,false\}} \prod_i c_i(A,B,C)$$

$$\sum_{A,B,C \in \{true,false\}} w(A)w(B)w(C) \prod_i c_i(A,B,C)$$

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

possible worlds

weight of literal

for instance …

w(f) = p

w(not f) = 1−p

Weighted Model Counting

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

$$WMC((A \vee B) \wedge (\neg A \vee \neg B)) =$$
$$w(A) \times w(\neg B) + w(B) \times w(\neg B)$$

Extensions exist towards continuous distributions —
Weighted Model Integration (WMI) - see Belle et al. UAI 15 and follow up work

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

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

interpretations (truth value assignments) of propositional variables

possible worlds

weight of literal

for p::f,
w(f) = p
w(not f) = 1−p

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).
```

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

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

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]

Sum Product

$$\text{SUMPRODUCT}\ (V, F, \oplus, \otimes) \qquad \oplus_{X_1} \cdots \oplus_{X_m} \otimes_{i=1}^{m} f_i(V_i)$$

Variables

Factors

$\oplus, \otimes$ typically commutative, associative,
and $\otimes$ distributes over $\oplus$,
0 identity element for $\oplus$ and annihilating elements for
1 identity element for $\otimes$

semirings

Inference

- Many algorithms exist for many variants of SUMPRODUCT
- Variable elimination, recursive conditioning for Bayes;
- DPLL and variants for SAT and extensions
- AND/OR search tree …

DPLL

procedure $DPLL(Vars:$ variables, $Fs:$ set of factors):
    if $Fs = \{\}$ **return** 1
    else if $f \in Fs$ can be evaluated
        return $f \otimes DPLL(Vars, Fs - \{f\})$
    else select $v \in Vars$
        $Fs_T = Fs$ with v assigned $true$
        $Fs_F = Fs$ with v assigned $false$
        return $DPLL(V - \{v\}, Fs_T) \oplus DPLL(V - \{v\}, Fs_F)$

all variables assumed boolean (otherwise sum over all possible values *v* in last step)

DPLL

procedure $DPLL(Vars:$ variables, $Fs:$ set of factors):

if $Fs = \{\}$ **return** 1

else if $\{\} \in Fs$

return 0 (false)

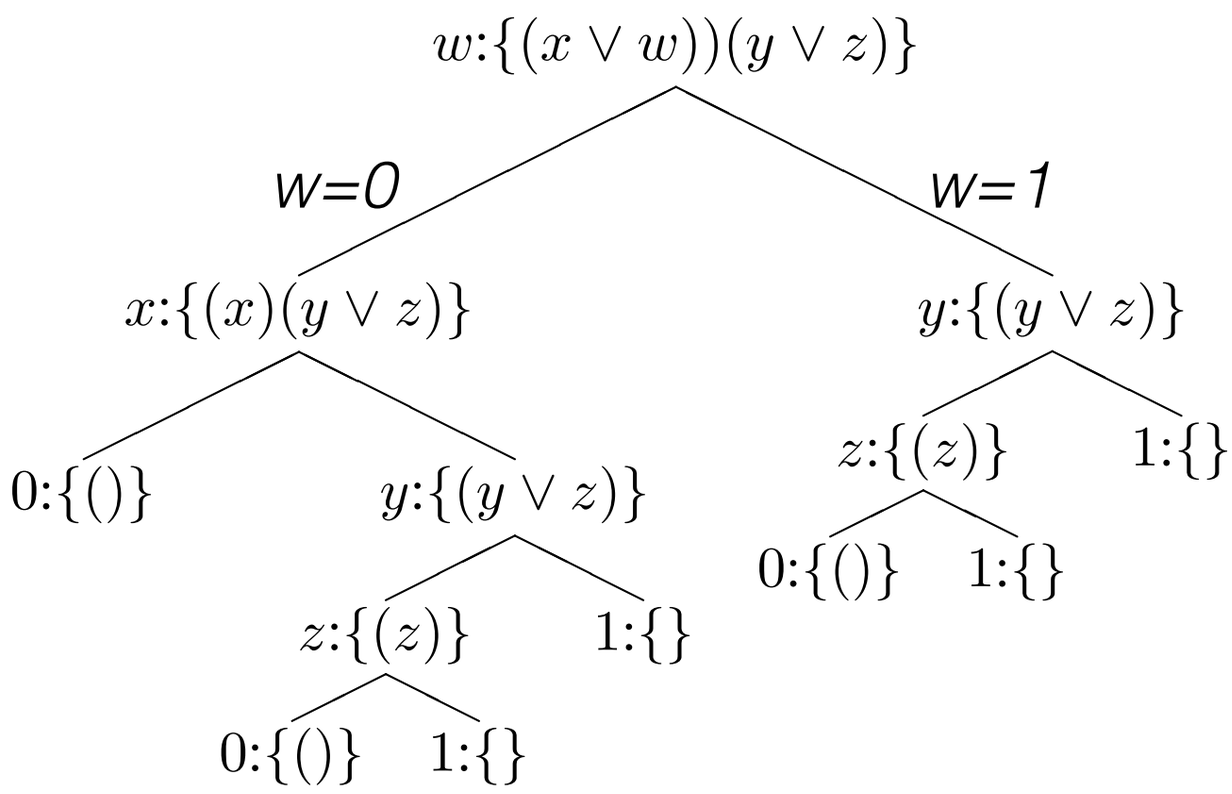
else select $v \in Vars$

$Fs_T = Fs$ with v assigned $true$

$Fs_F = Fs$ with v assigned $false$

return $DPLL(V - \{v\}, Fs_T) \oplus DPLL(V - \{v\}, Fs_F)$

Example DPLL

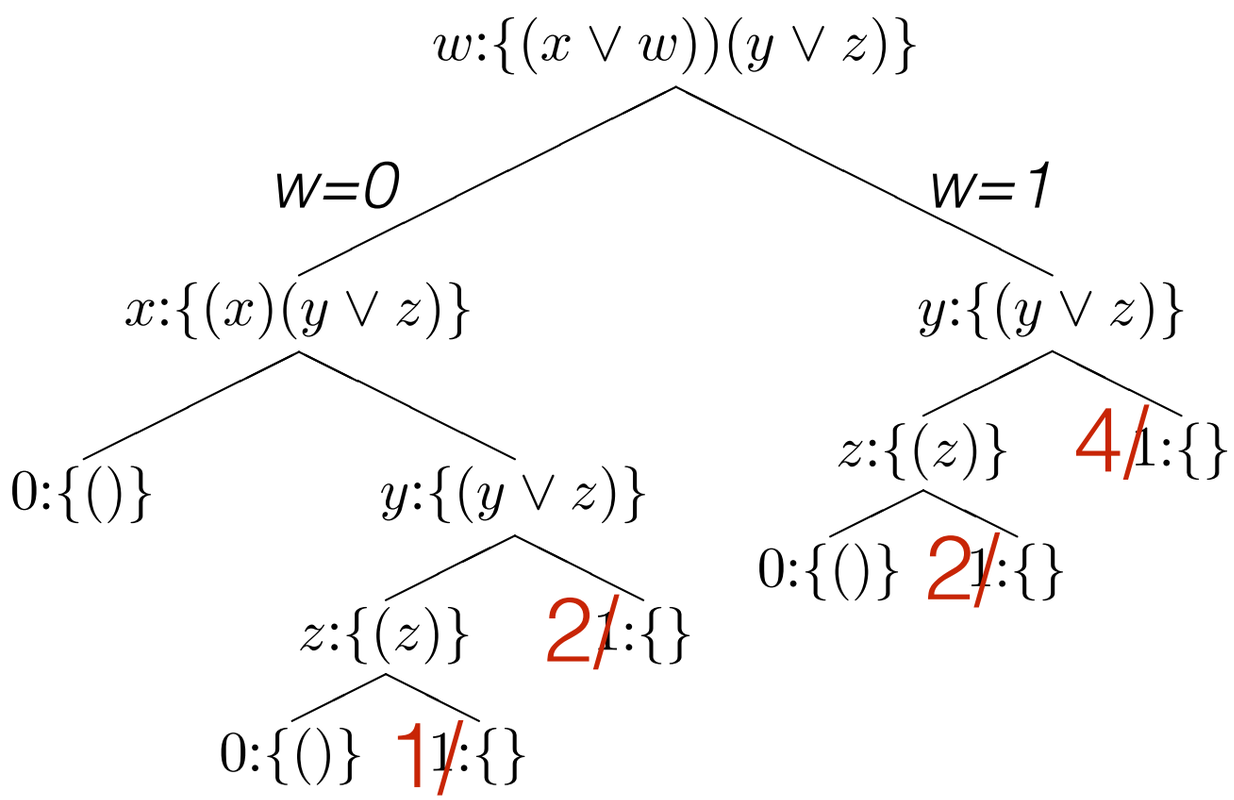


from Bacchus et al. JAIR 2009

#DPLL for #SAT

procedure $\#\ DPLL(Vars:$ variables, $Fs:$ set of factors):
    if $Fs = \{\}$ **return** $2^{|Vars|}$
    else if $\{\} \in Fs$
        return 0 (false)
else select $v \in Vars$
    $Fs_T = Fs$ with v assigned $true$
    $Fs_F = Fs$ with v assigned $false$
    return $DPLL(V - \{v\}, Fs_T) \oplus DPLL(V - \{v\}, Fs_F)$

Example #DPLL

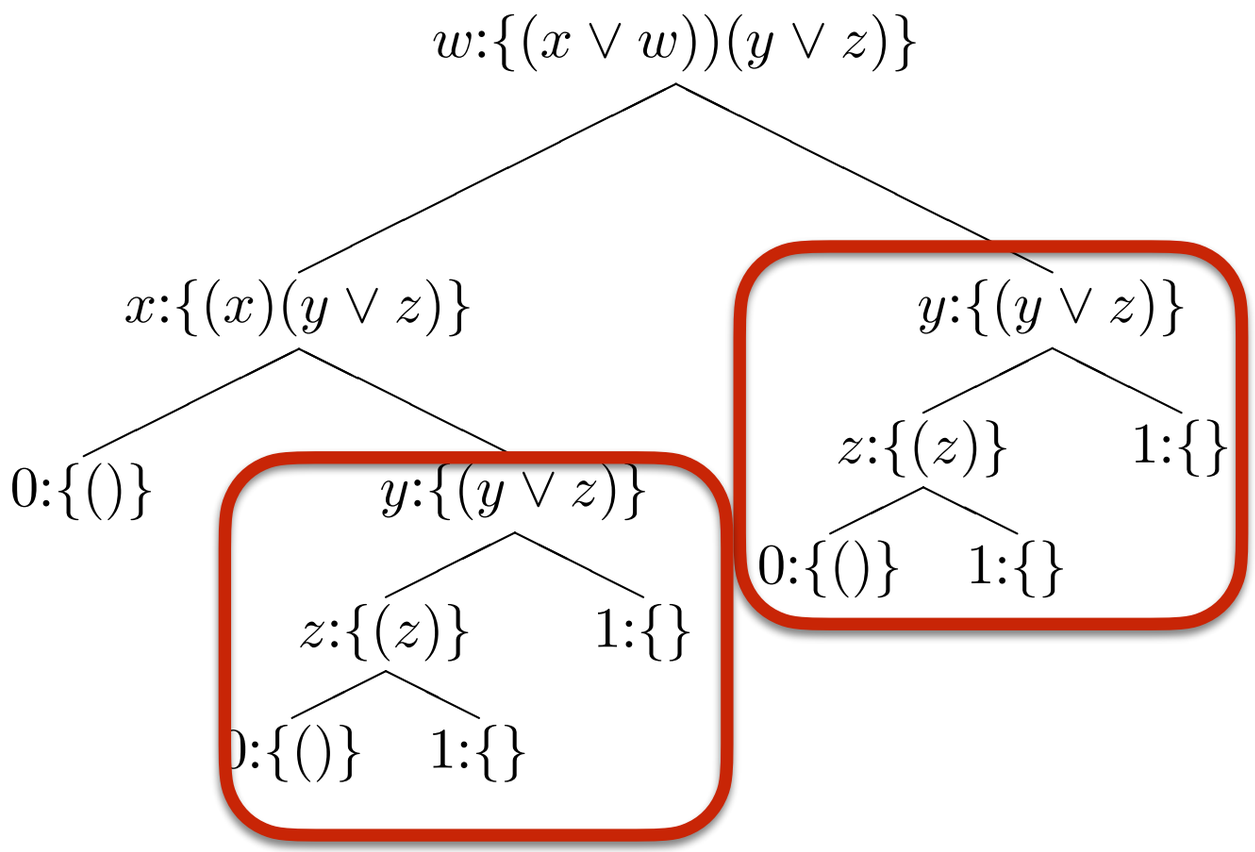


from Bacchus et al. JAIR 2009

#DPLL for #WMC

procedure $\#\ DPLL(Vars:$ variables, $Fs:$ set of factors, $w:$ weight funct
    if $Fs = \{\}$ **return** $\prod_{v \in Vars} w(v) + w(\neg v)$
    else if $\{\} \in Fs$
        return 0 (false)
else select $v \in Vars$
    $Fs_T = Fs$ with v assigned $true$
    $Fs_F = Fs$ with v assigned $false$
    return $w(v) \otimes DPLL(V - \{v\}, Fs_T) \oplus w(\neg v) \otimes DPLL(V - \{v\},$

Example DPLL



Caching

from Bacchus et al. JAIR 2009

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

Approximate Inference

Approximate Inference

- Lower and upper bounds

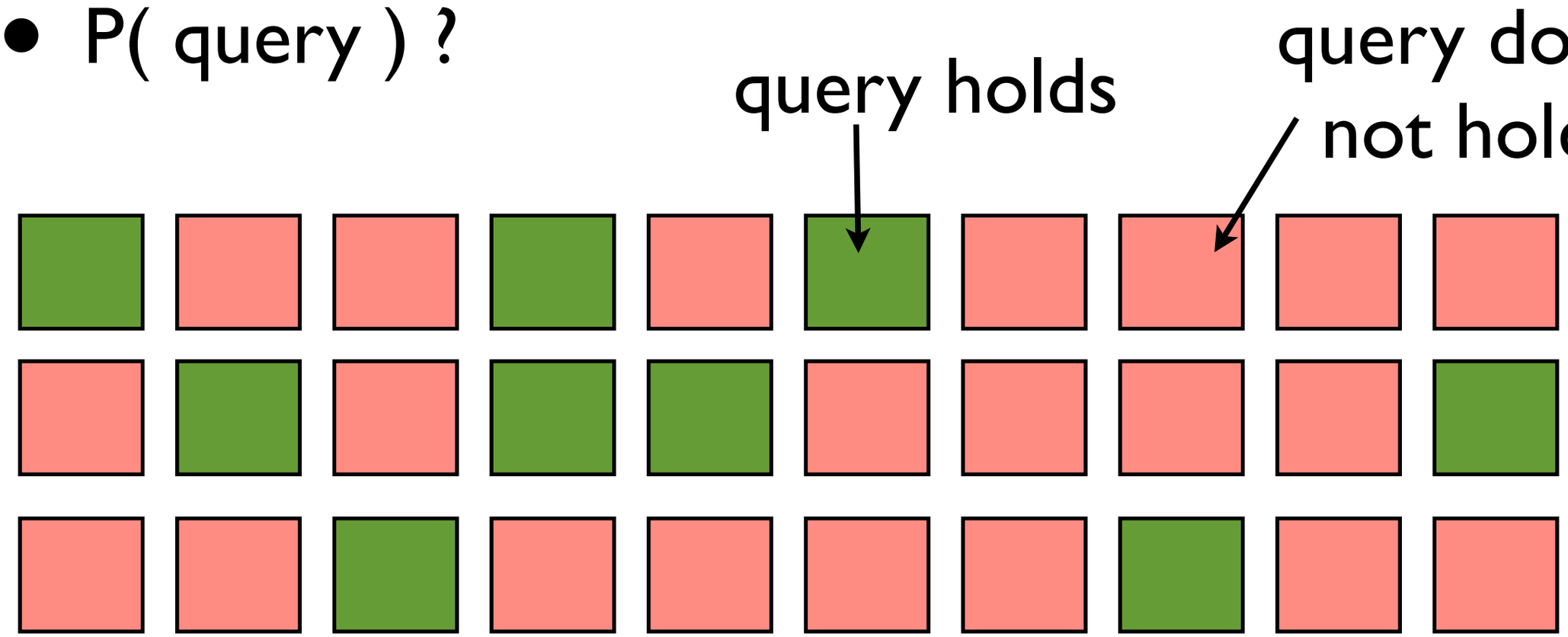
$$\phi_L \models \phi \models \phi_U$$

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

- Sampling

Sampling

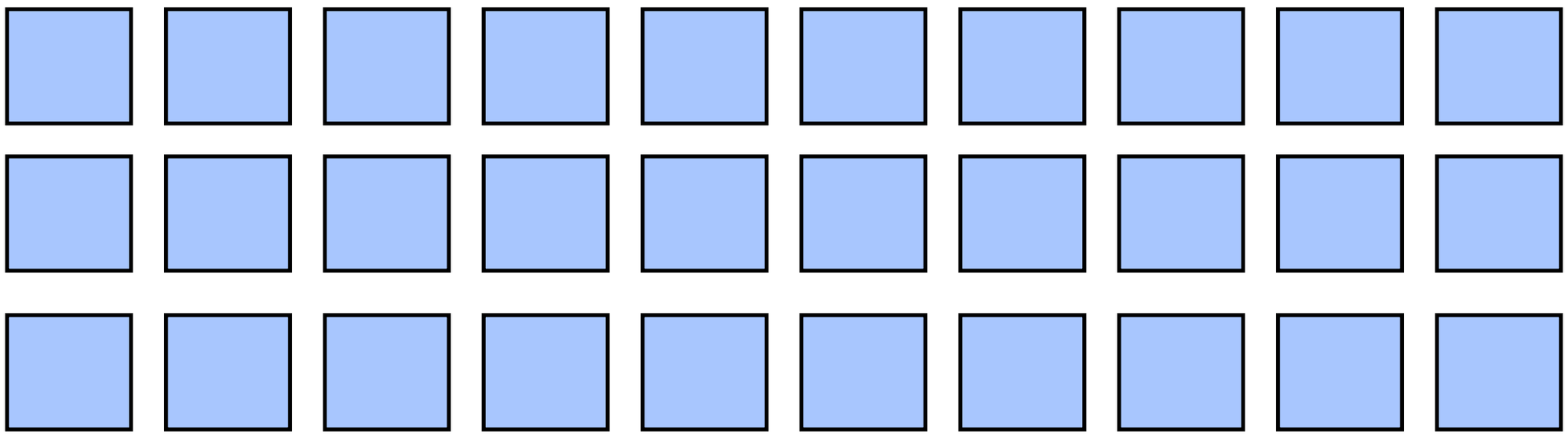
- P(query) ?



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

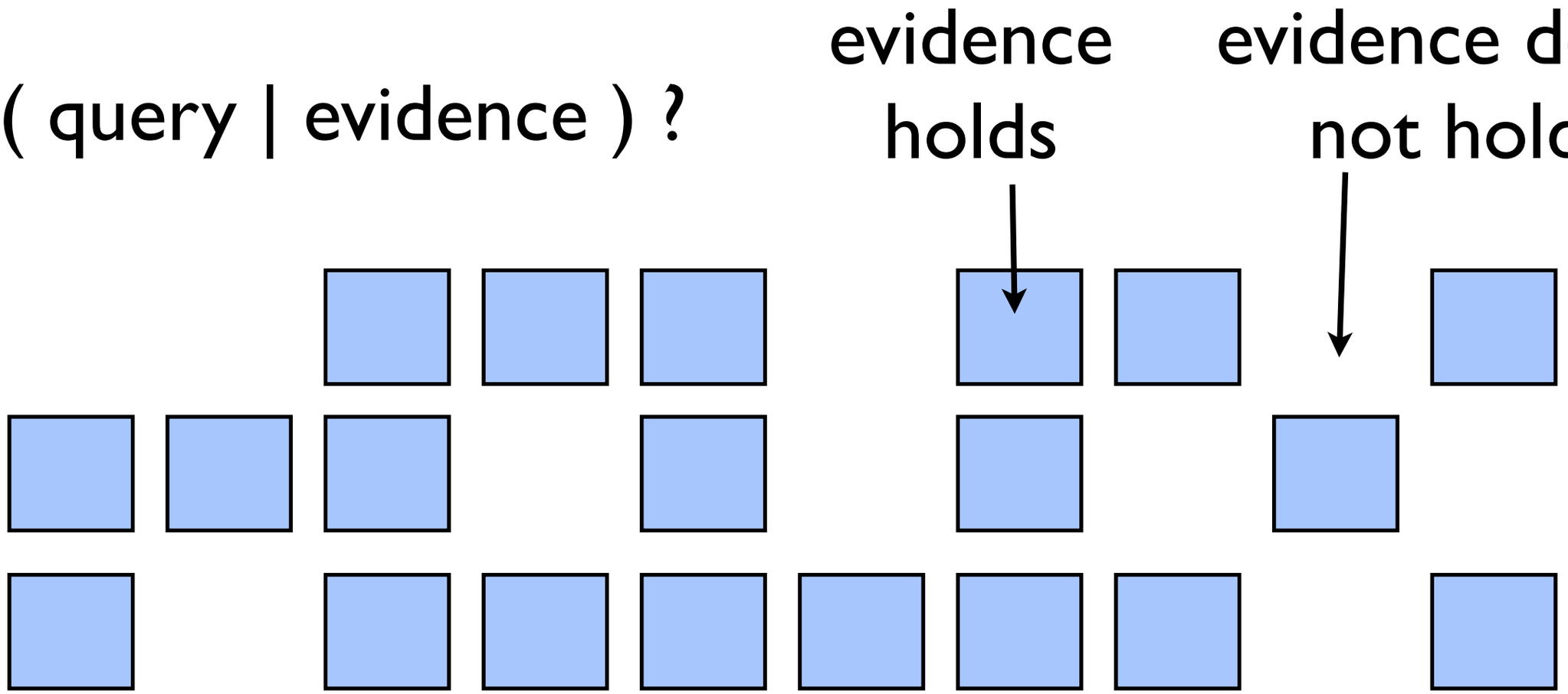
Rejection Sampling

- P(query | evidence) ?

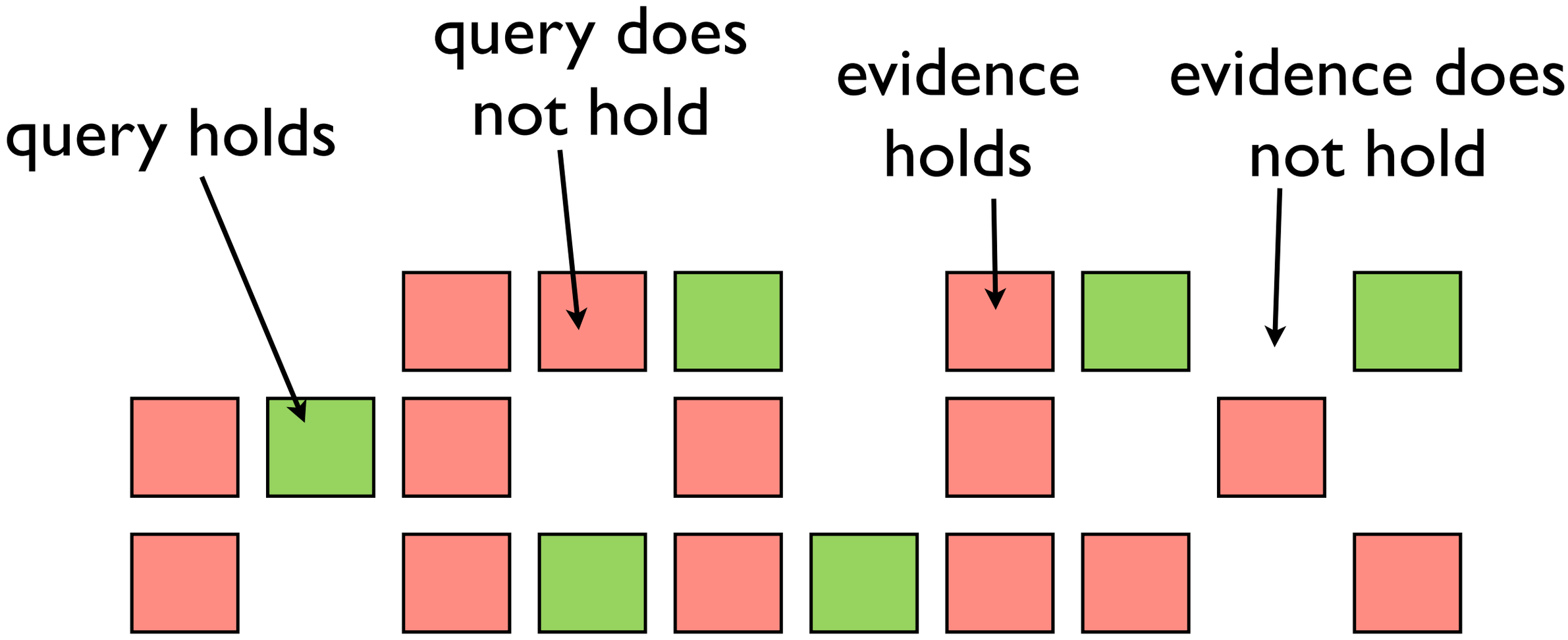


Rejection Sampling

- P(query | evidence) ?



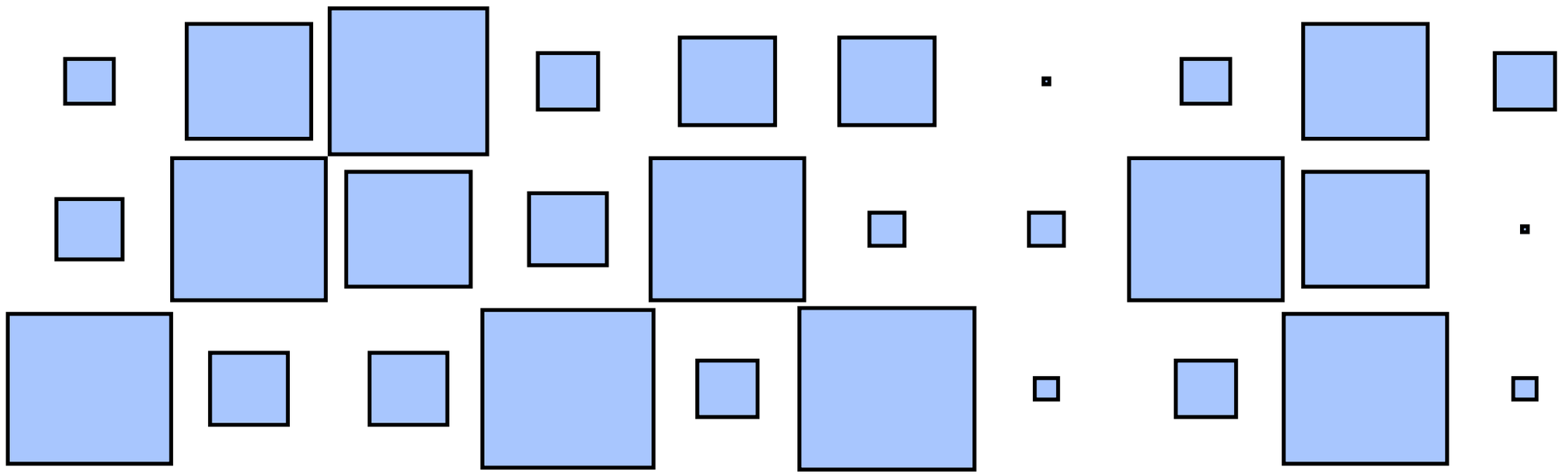
Rejection Sampling



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

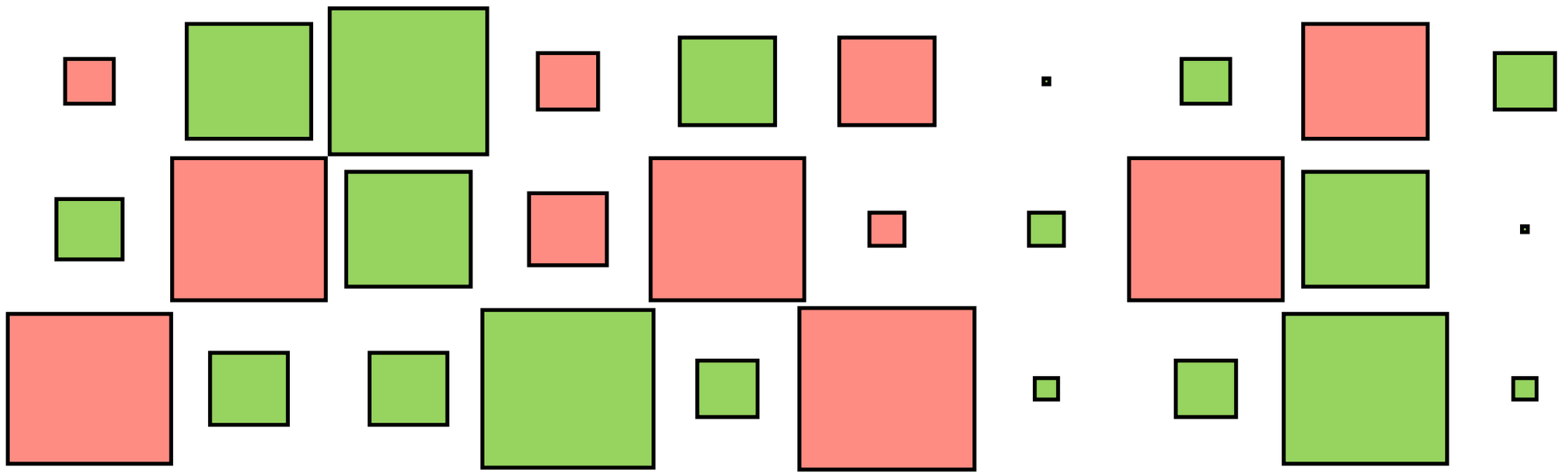
Likelihood Weighting

- P(query | evidence) ?



Likelihood Weighting

- P(query | evidence) ?



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

- Prob. Programming - Modeling
- Inference
- Learning
- Dynamics
- Logic, Probability and Neural Nets
- KBMC & Markov Logic
- Lifted Inference

... with some detours on the way