PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V9: Probabilistic Programming II

Özgür L. Özçep
Universität zu Lübeck
Institut für Informationssysteme

Probabilistic Logic Programs (PLP)

- devised by Poole and Sato in the 90s.
- built on top of the programming language Prolog
- upgrade directed graphical models
- Generalises probabilistic databases (Suciu et al.)
- combines the advantages / expressive power of programming languages (Turing equivalent) and graphical models
- Implementations: see next page

PLP Systems

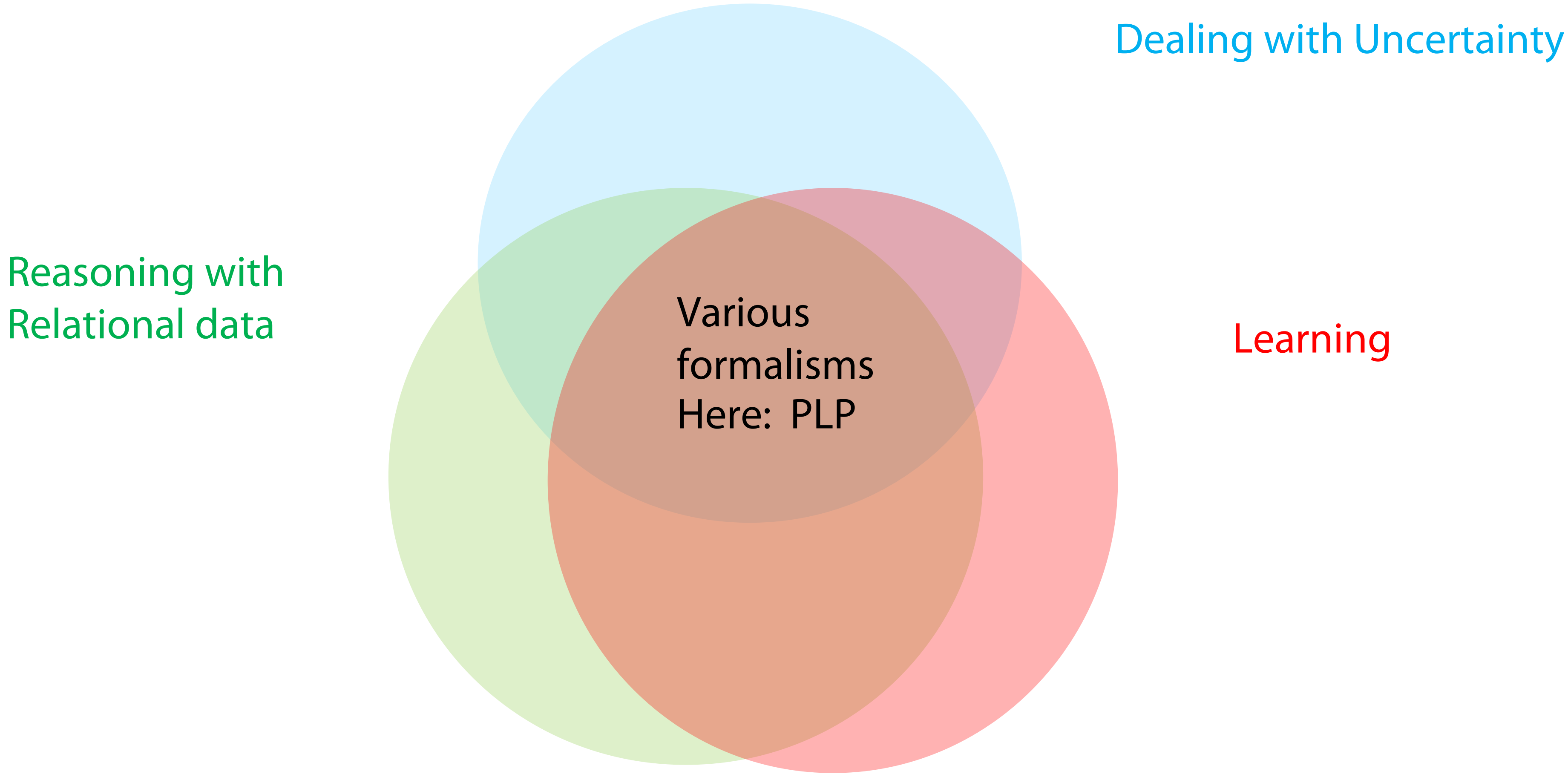
- **PRISM** https://www.prismmodelchecker.org/
- **ProbLog2** http://dtai.cs.kuleuven.be/problog/
- Yap Prolog https://github.com/vscosta/yap-6.3 includes
  - **ProbLog1**
  - **cplint**
  - **CLP(BN)**
  - **LP2**
- **PITA** in XSB Prolog http://xsb.sourceforge.net/
- **AILog2** http://artint.info/code/ailog/ailog2.html
- **SLPs** http://stoics.org.uk/~nicos/sware/pepl
- **contdist** http://www.cs.sunysb.edu/~cram/contdist/
- **DC** https://code.google.com/p/distributional-clauses
- **WFOMC** http://dtai.cs.kuleuven.be/ml/systems/wfomc

Today‘s Agenda (in classical linear form)

Probabilistic Logic Programming

1. Modeling
2. Reasoning
3. Learning

MODELING

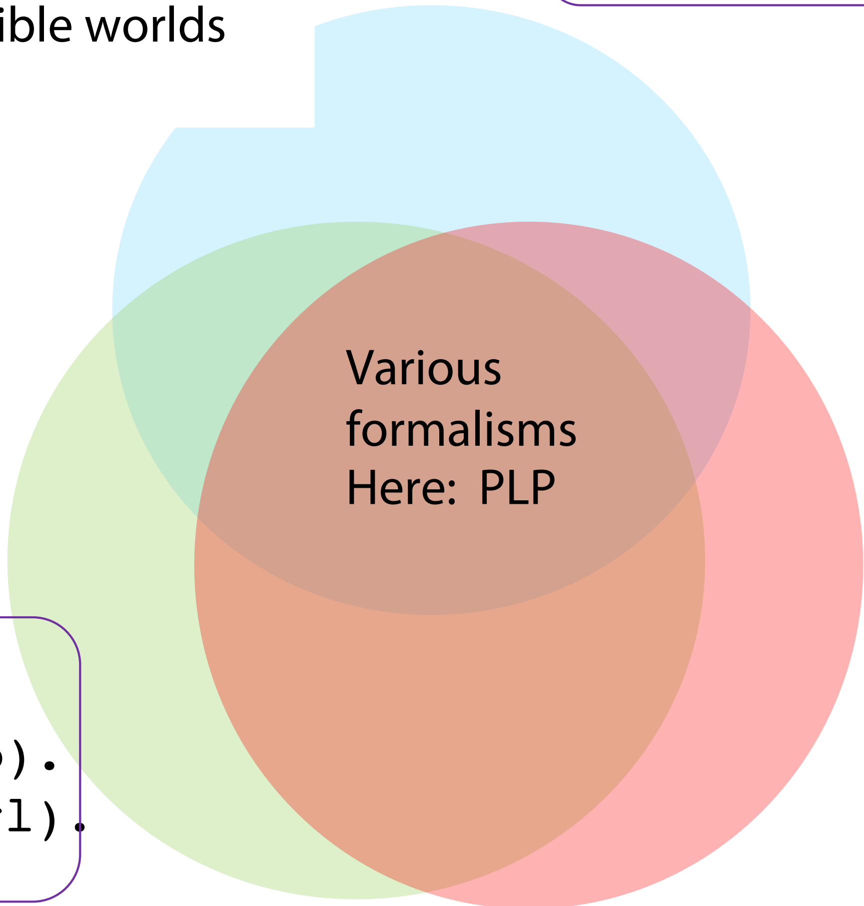

Dealing with Uncertainty
Reasoning with
Relational data
Various
formalisms
Here: PLP
Learning

Motivation (suffering from Vennitia)

Distribution Semantics (Sato, 95):
probabilistic choices + logic program
→ distribution over possible worlds

Several possible worlds

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

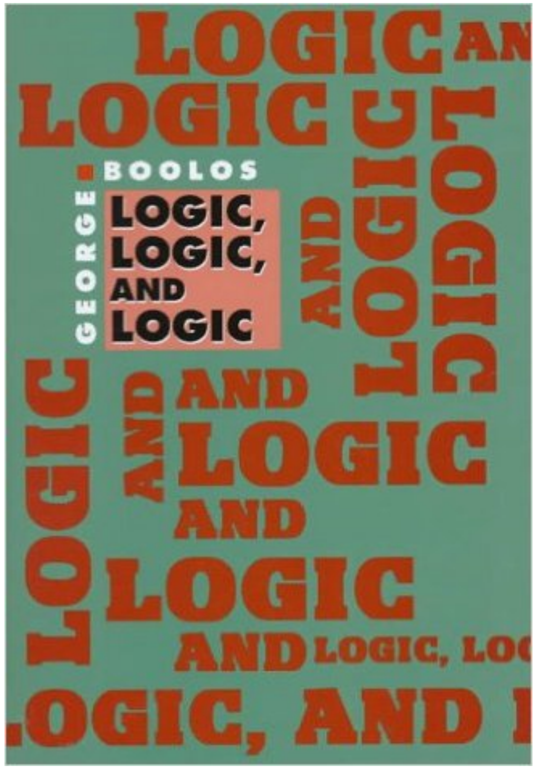


One world

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

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

The motto: Logic everywhere



See also IFIS course Information systems

The Logic programming (LP) paradigm

- The other big three paradigms of programming
  - Imperative (e.g. C)
  - Functional (e.g., Lisp)
  - Object-oriented (e.g. Java)
- Distinguishing feature of LP: Problem solving by specifying the „What“ not the “How to“
- Abstracting from
  - Control structures
  - Memory layout
  - Process direction
- Prominent examples: Prolog, Datalog, ASP (Answer set programing)

Logic

Science of logic investigates mathematical structures (static and dynamic) and formal languages to describe them by specifying a logic given by

- syntax (well-formed formula)
- semantics (truth conditions for sentences, entailment notion)
- calculus (provability, inference)

Introductory logic texbooks with CS in mind

- (Huth,Ryan 00)
- (Ben-Ari 01)

Where is the logic in logic programming?

- Specification of a domain with a set of formula (sometimes called a knowledge base)
  - Formula specifed by truth-condition semantics as in logic
  - In Prolog: formula are facts or rules

- Specifcation of the problem as a query (also a formula)
  - Query is Boolean or has variables to be bound

- Solving a problem according a logical calculus
  - try to infer (bindings for) query w.r.t. the knowledge base using rules
  - In Prolog use resolution

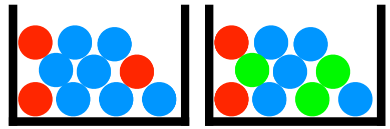
Prolog

- Prolog: Programmation en Logique

- Invented around 1970 when there was high interest in
  - Theorem proving
  - Language processing with formal grammars

- Protagonists
  - R. Kowalski: Theoretical contribution with SL-Resolution
  - A. Colmerauer and P. Roussel: developer

A bit of gambling with ProbLog



- 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).
```

- Probabilistic fact: Heads with probability 0.4
- 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

consequences

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

- Logical rule encoding background knowledge

Queries

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

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

- Probability of `win`? (marginal propability)
- Probability of `win` given `col(2,green)`? (conditional propability)
- Most probable world where `win` is true? (Most probable explanation (MPE))

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 x 0.3 x 0.3

H R G

W

Possible Worlds

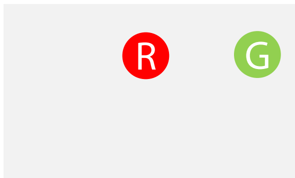
```
0.4 :: heads.
0.3 :: col(1,red); 0.7 :: col(1, blue).
0.2 :: col(2,red); 0.3 :: col(2,green);0.5 :: col(2,blue).

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

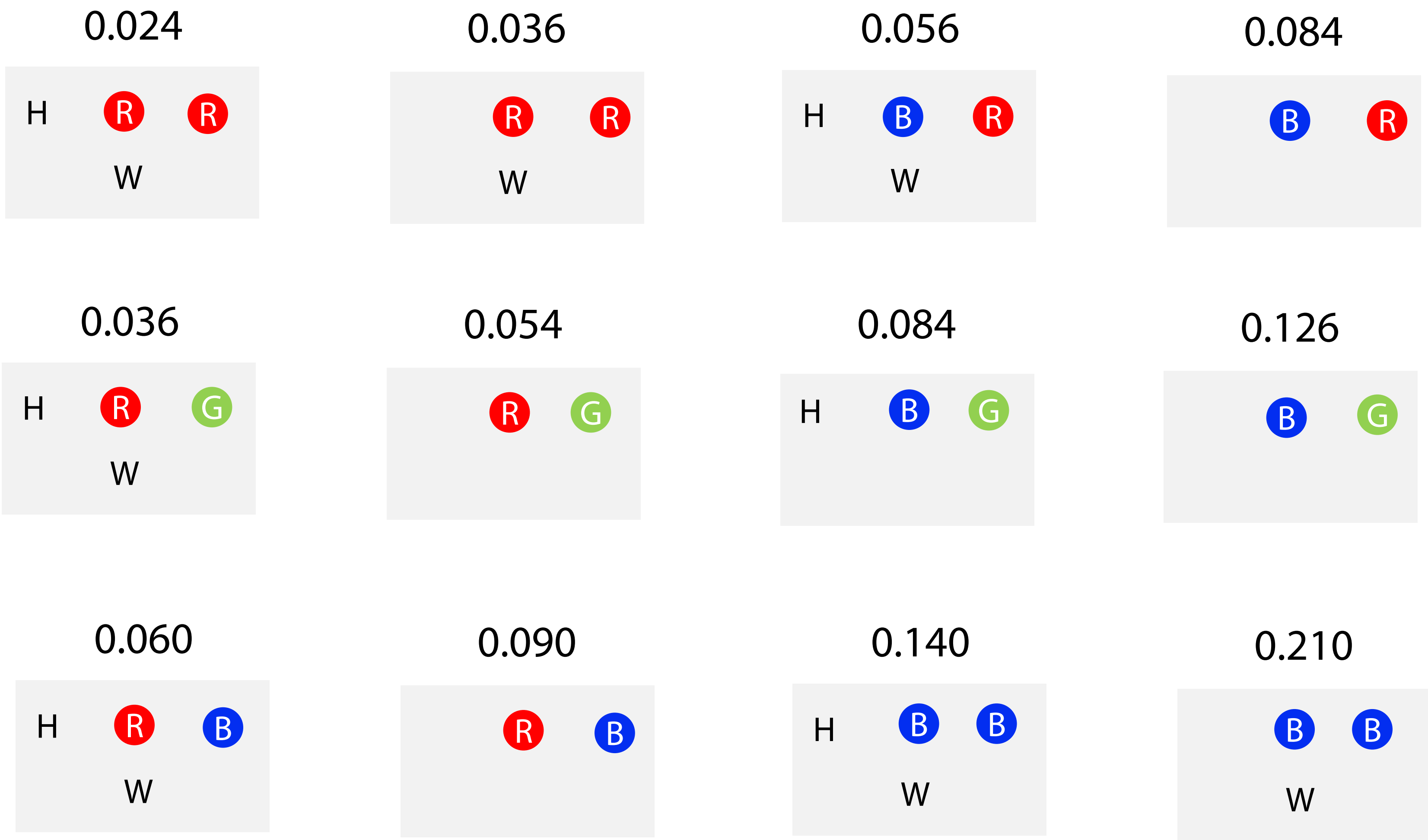
0.4 x 0.3 x 0.3

(1-0.4) x 0.3 x 0.2

(1-0.4) x 0.3 x 0.3

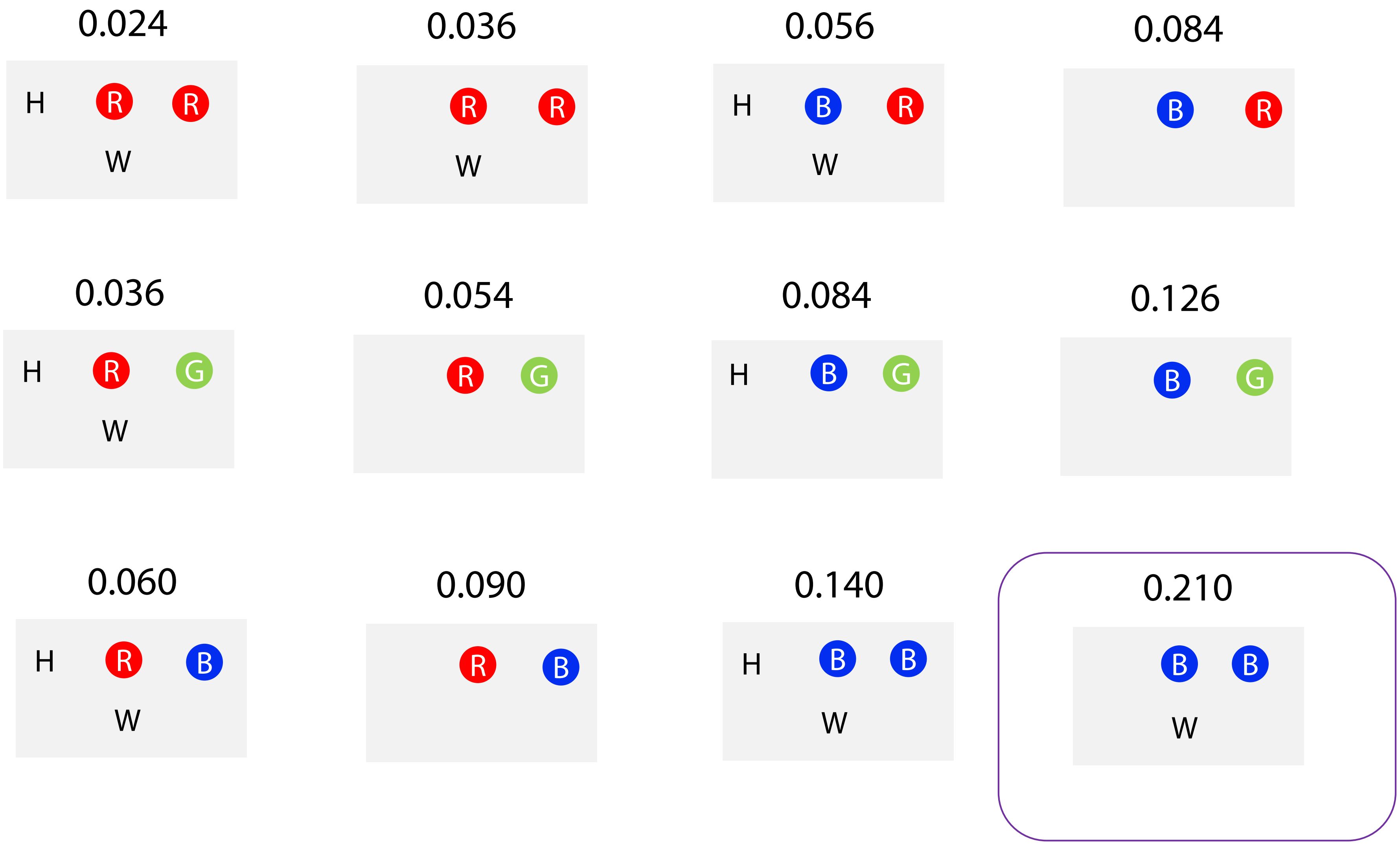


All Possible Worlds



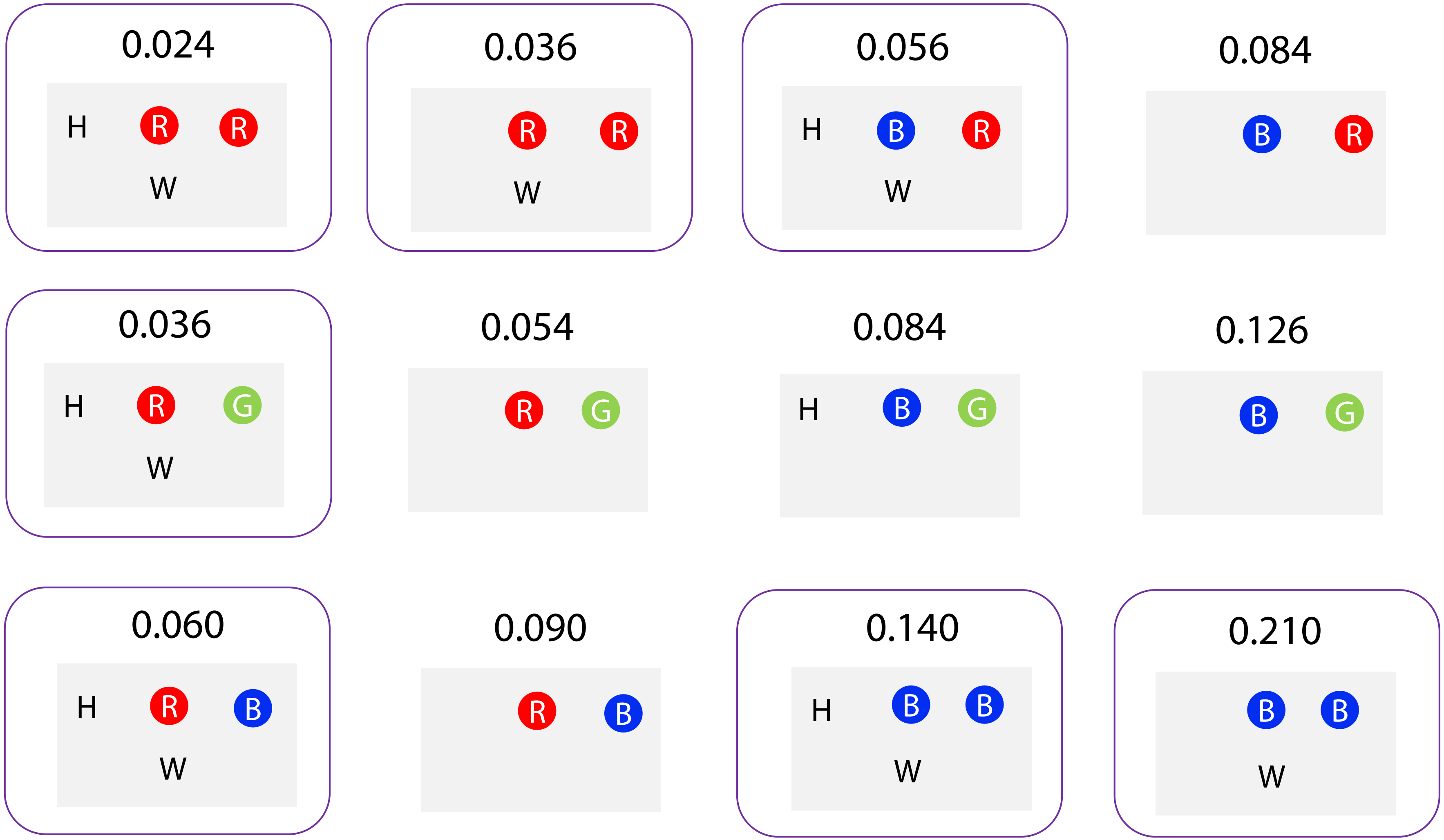
(remember the discussion in V8 on traces)

Most likely world with W (`win` = true)?



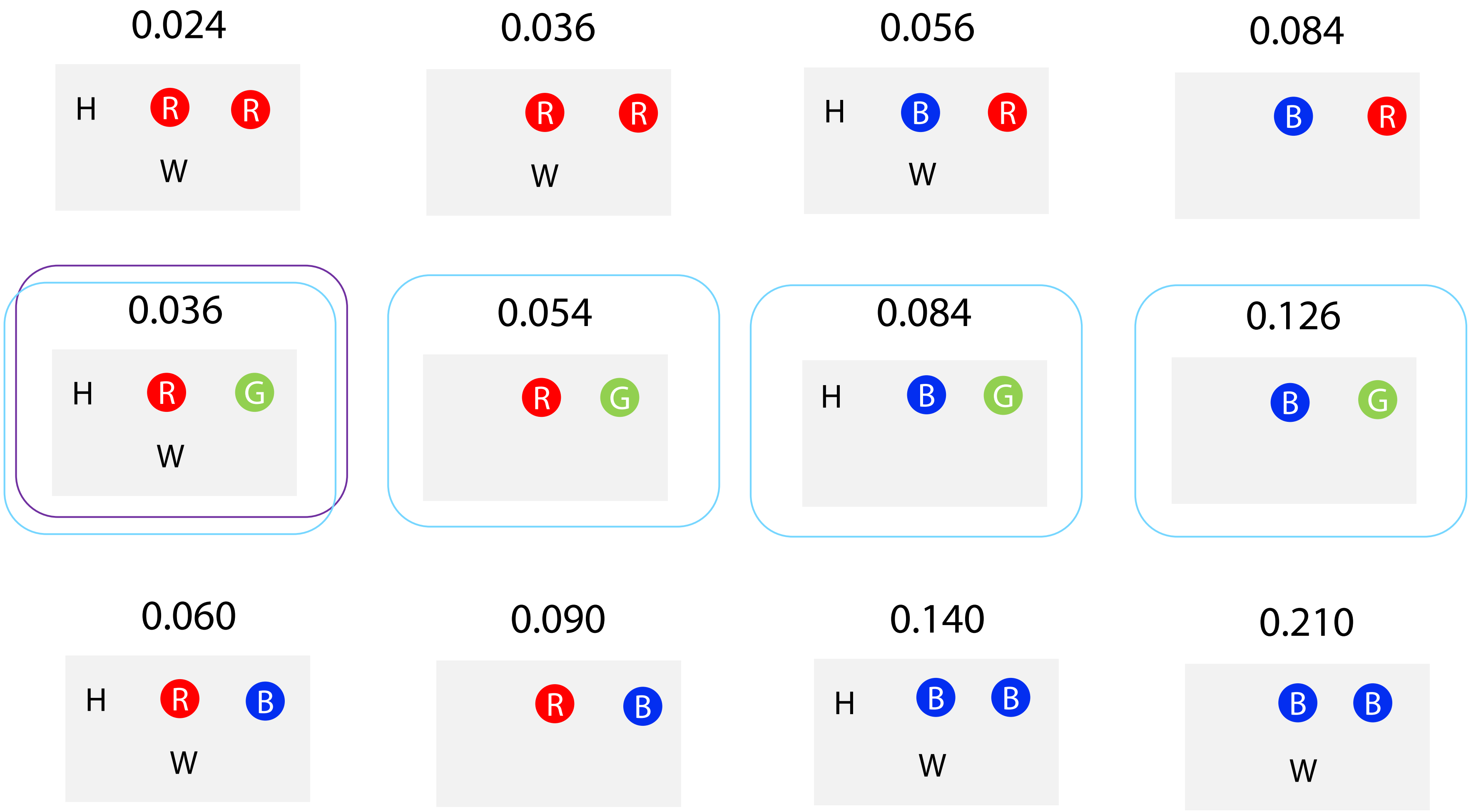
MPE inference

$$P(\texttt{win} = \text{true}) = P(\texttt{win}) = \Sigma_{world\ with\ w=true} = 0.562$$



$P(\texttt{win} \mid \texttt{col(2,green)}) = 0.036/0.3 = 0.12$

$P(\underline{\texttt{win,col(2,green)}})/P(\underline{\texttt{col(2,green)}}) = \Sigma/\Sigma$



Distribution semantics (Sato, 95)

Distribution semantics with probabilistic facts (Sato 95)

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

where

- Q = query
- F = subset of facts (assumed to hold in a possible world)
- R = Prolog rules
- $F \cup R \vDash Q$: Summing condition: possible worlds where Q is true

INFERENCE

The challenge: disjoint sum problem

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

win :- heads(1).
win :- heads(1), heads(3).   % win <-> h(1) v (h(2) & h(3))
```

- $P(win) = P\left(h(1) \vee \left(h(s) \wedge h(3)\right)\right) \neq$
  $$P(h(1)) + P(h(2) \wedge h(3))$$
- Rather should be
- $= P\big(h(1)\big) + P\big(h(2) \wedge h(3)\big) - P(h(1) \wedge h(2) \wedge h(3))$

Idea: Weighted Model Counting (WMC)

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

win :- heads(1).
win :- heads(1), heads(3).   % win <-> h(1) v (h(2) & h(3))
```

- Ground out
- Put formula in CNF (conjunctive normal form)
- Weights
- Call WMC

- $\left(\neg win \vee h(1) \vee h(2)\right)$
  $\wedge \left(\neg \mathrm{win} \vee h(1) \vee h(3)\right)$
  $\wedge \left(win \vee \neg h(1)\right)$
  $\wedge \left(win \vee \neg h(2) \vee \neg h(3)\right)$
- $h(1) \rightarrow 0.4 \quad \neg h(1) \rightarrow 0.6$
  $h(2) \rightarrow 0.7 \quad \neg h(2) \rightarrow 0.3$
  $h(3) \rightarrow 0.5 \quad \neg h(3) \rightarrow 0.5$

Recap on some terminlogy from logic

- A propositional formula is in conjunctive normal form (CNF) iff it is a conjunction of disjunctions of literals
- Literals = proposition symbol or its negation

- Every propositional formula can be transformed into CNF (using distribution, de Morgan rules and double negation elimination)

Recap on some terminlogy from logic

For the example note that

- $A \leftrightarrow B$ and $(A \rightarrow B) \wedge (B \rightarrow A)$ are equivalent
- $A \rightarrow B$ is equivalent to $\neg A \vee B$
- Interpretations I_i (truth value assignments) can also be recorded in set notation (as doen in the following)
- E.g. $I_2 = \{\neg A, B\}$ or even shorter: $\mathrm{I}_2 = \{B\}$ (considering only the propositional variables with value 1)

	A	B	$A \leftrightarrow B$	$(A \rightarrow B)$	$\wedge$	$(B \rightarrow A)$	$(\neg A \vee B)$	$\wedge$	$(\neg B \vee A)$
							in CNF		
I_1	0	0	1	1	1	1	1	1	1
I_2	0	1	0	1	0	0	1	0	0
I_3	1	0	0	0	0	1	0	0	1
I_4	1	1	1	1	1	1	1	1	1

Recap on some terminology from logic

- Grounding
  - Idea: „Propositionalize“ rules
  - Technically: Instantiate all variables with all possible constant combinations
  - E.g. successfulStudent(X):- lovesLogic(X) over constants {a,b}
    - successfulStudent(a):- lovesLogic(a),
    - successfulStudent(b):- lovesLogic(b)
  - (Grounding not used actually on the slides before, as rules contained no variables)

Weighted Model Counting

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

where

- ϕ : propositional formula in CNF

  (resulting from problog programm or any other statistical relational model (SRL))

- I_V: interpretation of propositional variables

  (in set notation; corresponds to possible world)

- $w(l)$: weight of literal

  (for $p::f$ one assigns $w(f) = p, w(\neg f) = 1 - p$)

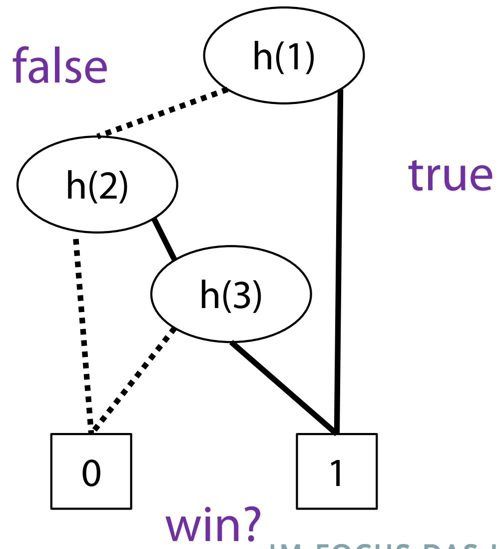
For $\phi = Q$:

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

Weighted Model Counting

- Simple WMC solvers based on a generalisation of DPLL algorithm for SAT (Davis Putnam Logeman Loveland algorithm)
- Current solvers often use knowledge compilation – here an OBDD (ordered binary decision diagram), many variations s-dDNNF, SDDs, (see also following lectures V10-V13)

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

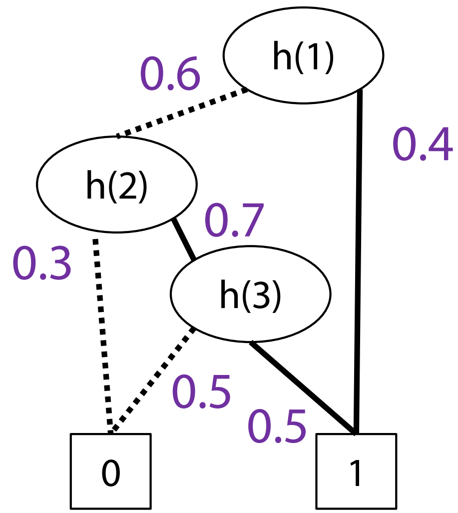


Weighted Model Counting

- Simple WMC solvers based on a generalisation of DPLL algorithm for SAT (Davis Putnam Logeman Loveland algorithm)
- Current solvers often use knowledge compilation – here an OBDD (ordered binary decision diagram), many variations s-dDNNF, SDDs, (see also following lectures V10-V13)

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

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



More inference

- Many variations / extensions
- Approximate inference
- Lifted inference (lifting from propositional to first order)
  - infected(X) :- contact(X,Y), sick(Y).

LEARNING

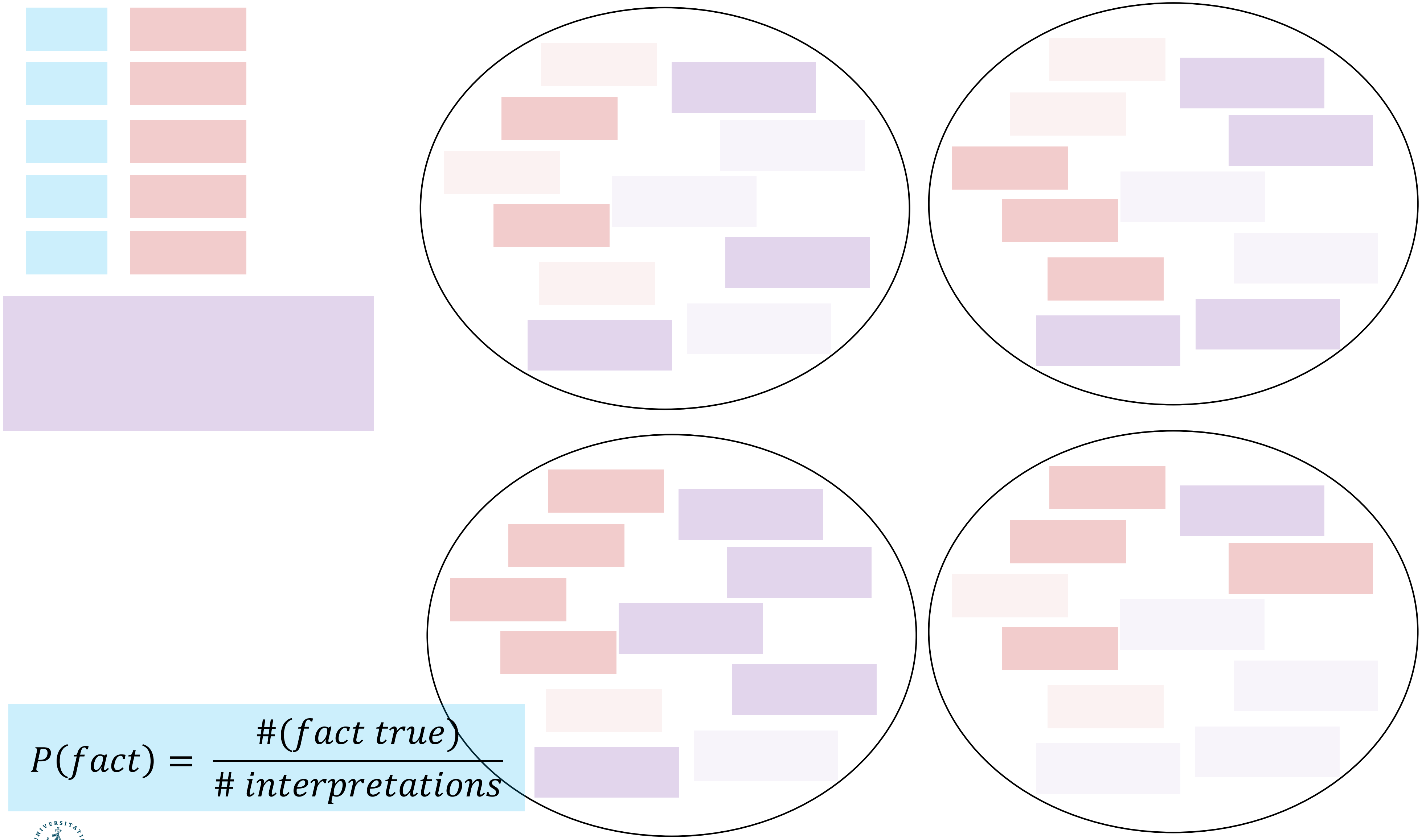
Parameter Learning: an example

- 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



$$P(fact) = \frac{\#(fact\ true)}{\#\ interpretations}$$

Partial interpretations

- Not all facts are observed
  - Note: this is different from some facts Being false
- Use for this some form of the EM-algorithm (Expectation maximization)
  - Expected count used instead of count
  - P(Q|E) –conditional queries

Reminder: EM: How it Works on Naive Bayes

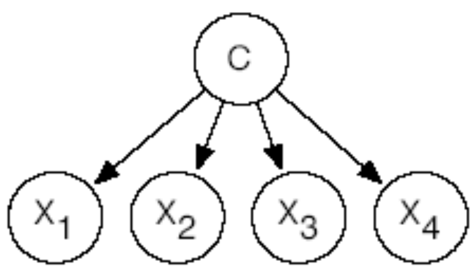
- Consider the following data,
  - N examples with Boolean attributes X1, X2, X3, X4

Data

X_1	X_2	X_3	X_4
t	f	t	t
f	t	t	f
f	f	t	t
	…		

- which we want to categorize in one of three possible values of class C = {1,2,3} (hidden, no observations given)
- We use a Naive Bayes classifier with hidden variable C

Model



→ **Probabiliti**

$P(C)$	?
$P(X_1 \mid C)$	?
$P(X_2 \mid C)$	?
$P(X_3 \mid C)$	?
$P(X_4 \mid C)$	?

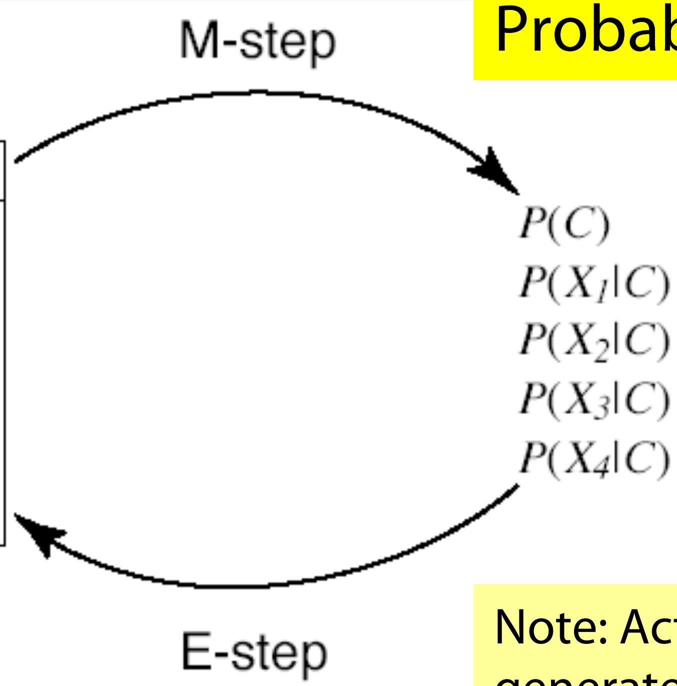
Reminder: EM: General Idea

- The algorithm starts from "invented" (e.g., randomly generated) information to solve the learning problem, i.e.
  - Determine the network parameters (CPT in Bayesian networks)
- It then refines this initial guess by cycling through two basic steps
  - Expectation (E): update the data with predictions generated via the current model
  - Maximization (M): given the updated data, update the model parameters using the Maximum Likelihood (ML) approach
    - ✓ This is the same step that is used for learning parameters for fully observable networks

EM Cycle

Expected Counts
("Augmented data")

Probabilities

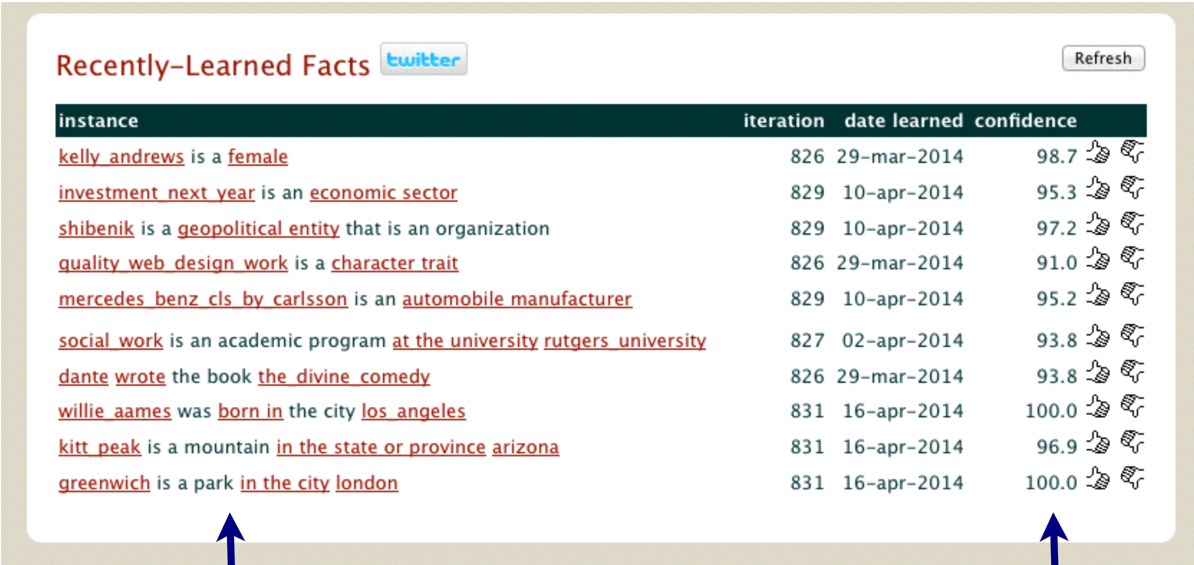


X_1	X_2	X_3	X_4	C	*count*
⋮	⋮	⋮	⋮	⋮	⋮
t	*f*	*t*	*t*	*1*	
t	*f*	*t*	*t*	*2*	
t	*f*	*t*	*t*	*3*	
⋮	⋮	⋮	⋮	⋮	⋮

Note: Actually you never generate any data in E-step but expected counts

Learning Rules/Structures

Information Extraction in NELL



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/

ProbFOIL

- Upgrade rule-learning to a probabilistic setting within a relational learning / inductive logic programming setting
  - Works with a probabilistic logic program instead of a deterministic one.

- Introduce ProbFOIL, an adaption of Quinlan's FOIL

- Apply to probabilistic databases like NELL

Example in Pro Log

```
surfing(X) :- not rain(X), windOK(X).        %H
surfing(X) :- not rain(X), sunshine(X).

rain(e1).                                     %B
windOK(e1).
sunshine(e1).

?- surfing(e1).  % Query
No  % Answer no, because surfing(e1) does not follow from H u B
```

Example in ProbLog

```
p1 :: surfing(X) :- not rain(X), windOK(X).    % H
p2 ::surfing(X) :- not rain(X), sunshine(X).

0.2 :: rain(e1).                                 %  B
0.7 :: windOK(e1).
0.6 :: Sunshine(e1).

?- P(surfing(e1)).  % Query
  % gives asnwer probability P(B U H |= e) =
  % (1-0.2) x    0.7 x p1 + (1-0.2) x 0.6     x (1-0.7)     x p2
  % no rain x windok x p1 + no rain x sunshine x not windOk x p2
```

Note: probabilities p_1, p_2 in front of rules are syntactic sugar.

Classical FOIL (Quinlan)

- Input
  - Prolog program (or any FOL theory)
  - Observed sequence of facts E (such as `surfing(e1)`)
  - Space of hypotheses L
- Output: Hypothesis set $H \subseteq L$ (rules) s.t. $B \cup H \models E$

- Hypothesis space contains all admissible rules over the language up to some complexity
- Various heuristics

Inductive Probabilistic Logic Programming

- Input
  - a set of example facts $e \in E$ together with the probability p that they hold
  - a background theory B in ProbLog

  (note: B may contain facts and rules, which we know to hold)
  - a hypothesis space L (a set of clauses)

- Output

$$argmin_H\ loss(H, B, E) = arming_H \sum_{e_i \in E} |P_s(B \cup H \vDash e_i) - p_i\ |$$

with optimal probabilities for rules.

Next weeks

- More details on the efficient representation of probabilities and formula.

Uhhh, a lecture with a hopefully useful

APPENDIX

Probability theory basics reminder

Random variable (RV)

- possible worlds defined by assignment of values to random variables.
- Boolean random variables
  e.g., Cavity (do I have a cavity?).
  Domain is < true , false >
- Discrete random variables
  e.g., possible value of Weather is one of
  < sunny, rainy, cloudy, snow >
- Domain values must be exhaustive and mutually exclusive
- Elementary propositions are constructed by assignment of a value to a random variable: e.g.,
  - Cavity = false (abbreviated as ¬cavity)
  - Cavity = true (abbreviated as cavity)
- (Complex) propositions formed from elementary propositions and standard logical connectives, e.g., Weather = sunny ∨ Cavity = false

Probabilities

- Axioms (for propositions $a, b, \top = (a \vee \neg a)$, and $\bot = \neg \top$):
  - $0 \leq P(a) \leq 1;\ P(\top) = 1;\ P(\bot) = 0$
  - $(P(a \vee b) = P(a) + P(b) - P(a \wedge b)$
- Joint probability distribution of $\mathbf{X} = \{X_1, \dots, X_n\}$
  - $\boldsymbol{P}(X_1, \dots, X_n)$
  - gives the probability of every atomic event on $\boldsymbol{X}$
- Conditional probability
  $P(a \mid b) = P(a \wedge b) / P(b)\ if\ P(b) > 0$
- Chain rule

$$\boldsymbol{P}(X_1, \dots, X_n) = \prod_{i=1}^{n} \boldsymbol{P}(X_i | X_1, \dots, X_{i-1})$$

- Marginalization: $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y, z)$
- Conditioning on Z:
  - $\boldsymbol{P}(Y) = \sum_{z \in Z} \boldsymbol{P}(Y|z)\boldsymbol{P}(z)$ (discrete)
  - $\boldsymbol{P}(Y) = \int \boldsymbol{P}(Y|z)\boldsymbol{P}(z)dz$ (continuous)
    $= \mathbb{E}_{\boldsymbol{z} \sim P(z)}$P(Y|z) (expected value notation)
- Bayes' Rule

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)} = \frac{P(D|H) \cdot P(H)}{\sum_h P(D|h)P(h)}$$

Color Convention in this Course

- Formulae, when occurring inline
- Newly introduced terminology and definitions
- Important results (observations, theorems) as well as emphasizing some aspects
- Examples are given with standard orange with possibly light orange frame
- Comments and notes in nearly opaque post-it
- Algorithms and program code
- Reminders (in the grey fog of your memory)

Today‘s lecture is based on the following

- Mainly
  - Luc de Raedt: Probabilistic Logic Programming and its Applications. Tutorial given at The Turing, London, September 11, 2017

    https://logic-data-science.github.io/Slides/DeRaedt.pdf
  - L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. Machine Learning, 100(1):5–47, 2015.
- A little bit
  - Tutorial on

    https://dtai.cs.kuleuven.be/problog/tutorial.html
  - Book: L. D. Raedt. Logical and relational learning. Cognitive Technologies. Springer, 2008.

References

- T. Sato. A statistical learning method for logic programs with distribution semantics. In IN PROCEED- INGS OF THE 12TH INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING (ICLP'95, pages 715–729. MIT Press, 1995.
- [1] M. Huth and M. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, 2000.
- M. Ben-Ari. Mathematical Logic for Computer Science. Springer, 2. edition, 2001.