
COMP3220: Document Processing and Semantic Technologies Combining Rules with Probabilities

Rolf Schwitter
Rolf.Schwitter@mq.edu.au

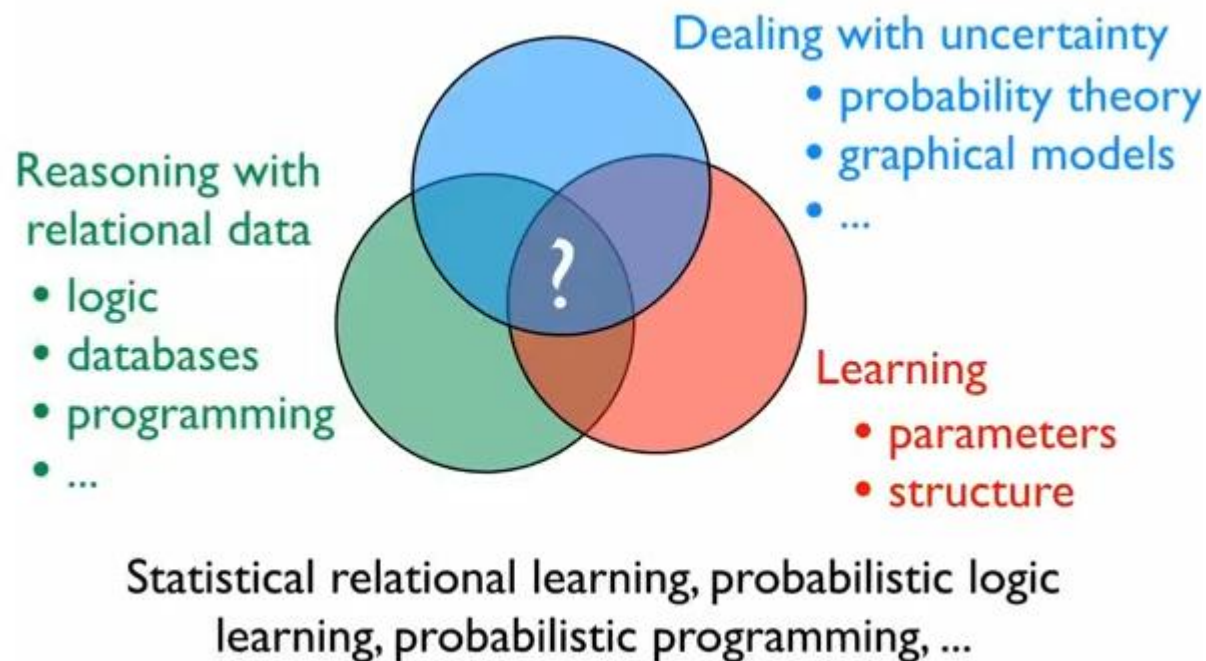
Today's Agenda

- Statistical Relational Learning
- A Motivating Example: Gambling
- Logic Programs with Annotated Disjunctions
 - MPE Inference
 - Marginal Probability
 - Conditional Probability
- Working with cplint on SWISH
- Exact and Approximate Inference
- Parameter and Structure Learning

A Key Question in Artificial Intelligence

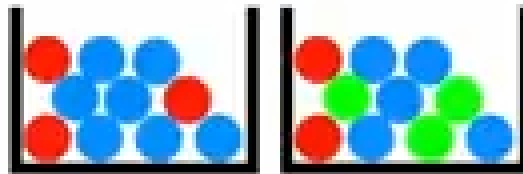


Statistical Relational Learning



A Motivating Example: Gambling

- Toss a biased coin and draw a ball from each of two urns.
- The first urn contains three red and seven blue balls; the second urn two red, three green and five blue balls.



- You win if you get
 - heads and a red ball or
 - two balls of the same colour.

How to Modell the Problem

- Random variables are represented as probabilistic facts.
- Probabilistic fact for the biased coin: **heads** is true with probability 0 . 4 and false with 0 . 6.

heads : 0 . 4 .

How to Modell the Problem

- Annotated disjunctions:
 - first ball is **red** with probability of 0.3 and blue with 0.7.
 - second ball is **red** with probability of 0.2, **green** with 0.3 and **blue** with 0.5.

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

`col(2,red):0.2 ; col(2,green):0.3 ;`

`col(2,blue):0.5.`

How to Modell the Problem

- Logic rules encode the background knowledge about what it means to win:

```
% You win if you get heads and a red ball.  
win :- heads, col(_,red) .
```

```
% You win if you get two balls of the same  
% colour.  
win :- col(1,C) , col(2,C) .
```


Logic Program with Annotated Disjunctions

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

Possible Worlds

- This program generates possible worlds.
- We start with the empty possible world and go through all possible choices that we can make.
- The empty possible world:



Logic Program with Annotated Disjunctions

heads : 0.4 .

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

col(2,red) : 0.2 ; col(2,green) : 0.3 ;

col(2,blue) : 0.5 .

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

win :- col(1,C), col(2,C) .

0.4



Logic Program with Annotated Disjunctions

`heads : 0.4 .`

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

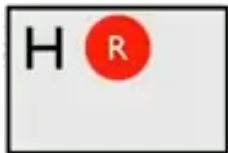
`col(2,red) : 0.2` ; `col(2,green) : 0.3` ;

`col(2,blue) : 0.5 .`

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

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

0.4×0.3



Logic Program with Annotated Disjunctions

`heads : 0.4 .`

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

`col(2,red) : 0.2 ; col(2,green) : 0.3 ;`

`col(2,blue) : 0.5 .`

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

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

$0.4 \times 0.3 \times 0.3$



Logic Program with Annotated Disjunctions

`heads : 0.4 .`

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

`col(2,red) : 0.2 ; col(2,green) : 0.3 ;`

`col(2,blue) : 0.5 .`

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

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

$0.4 \times 0.3 \times 0.3$



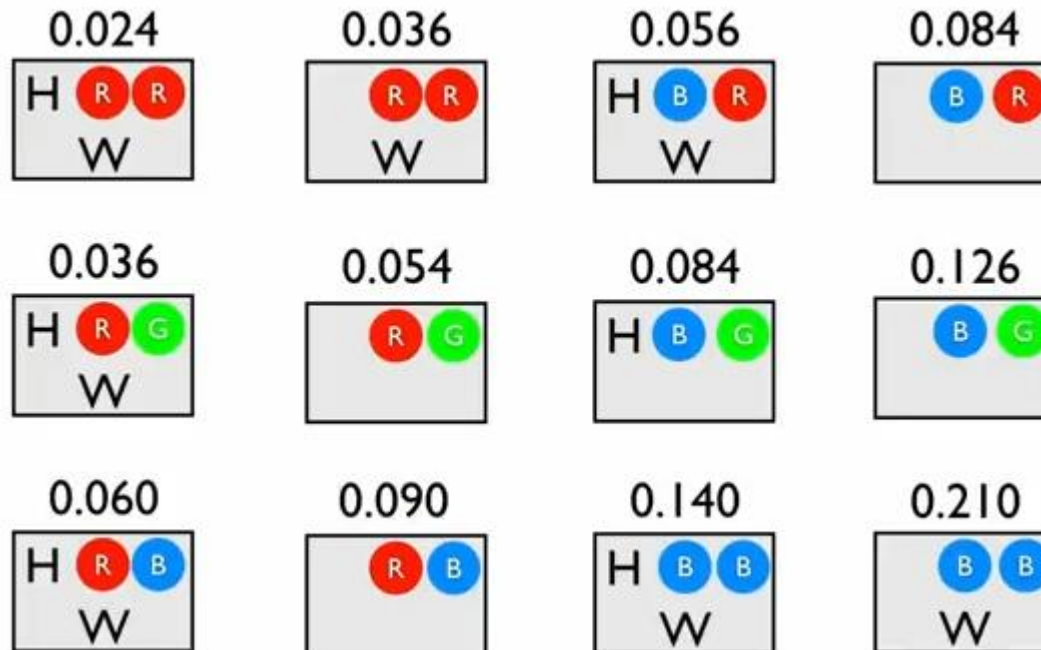
$(1-0.4) \times 0.3 \times 0.2$



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



All Possible Worlds

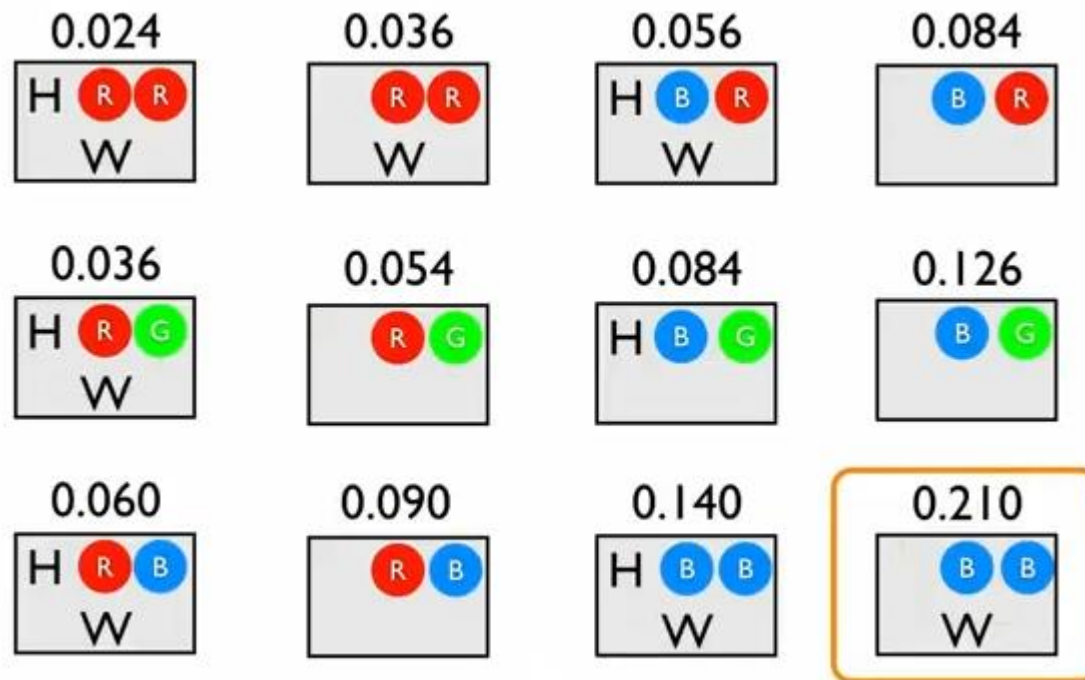


Questions

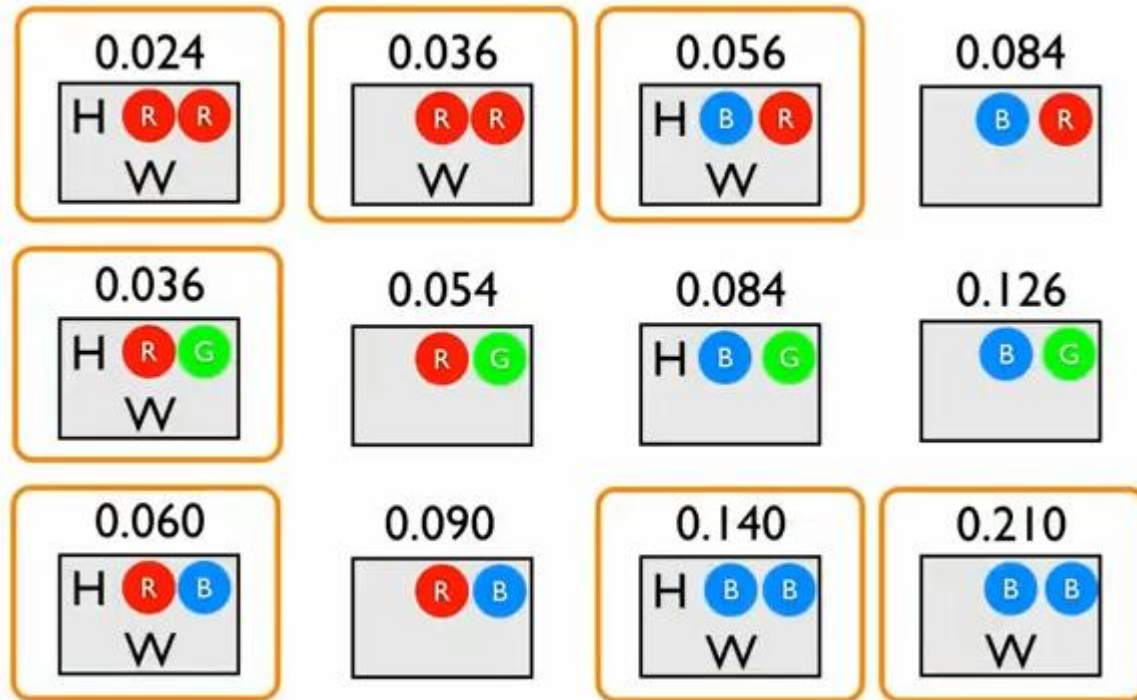
- What's the most probable world where **win** is true?
-> MPE¹ Inference
- What's the probability of **win**?
-> Marginal Probability
- What's the probability of **win** given **col(2, green)**?
-> Conditional Probability

¹ MPE = Most Probable Explanation

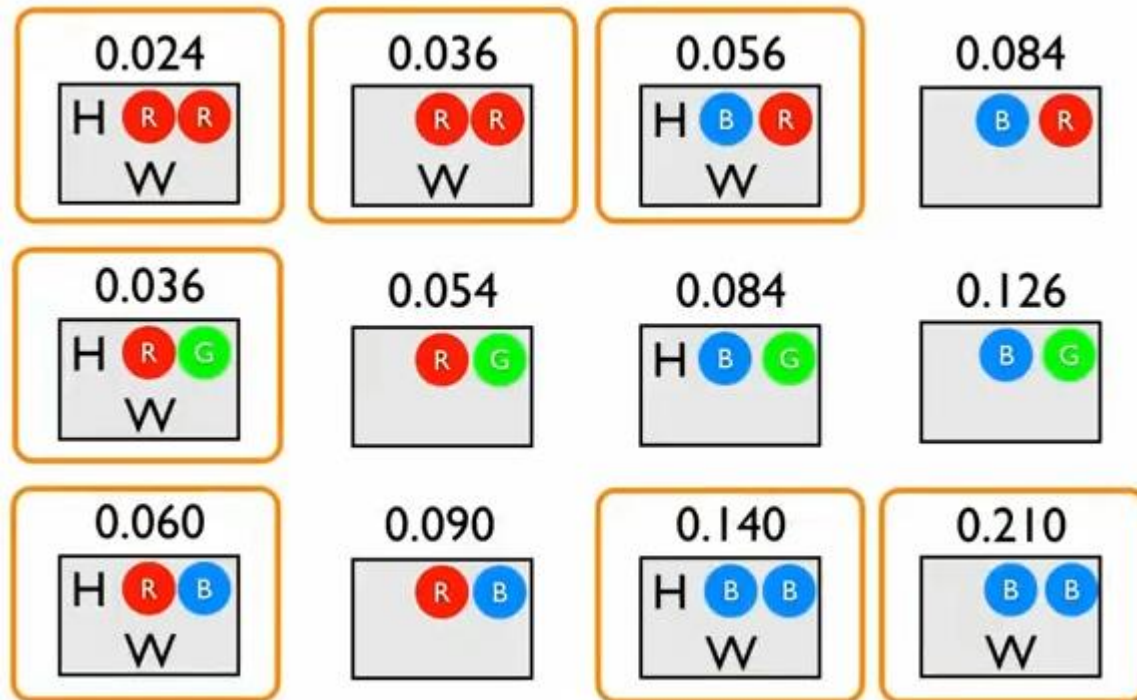
MPE Inference



Marginal Probability

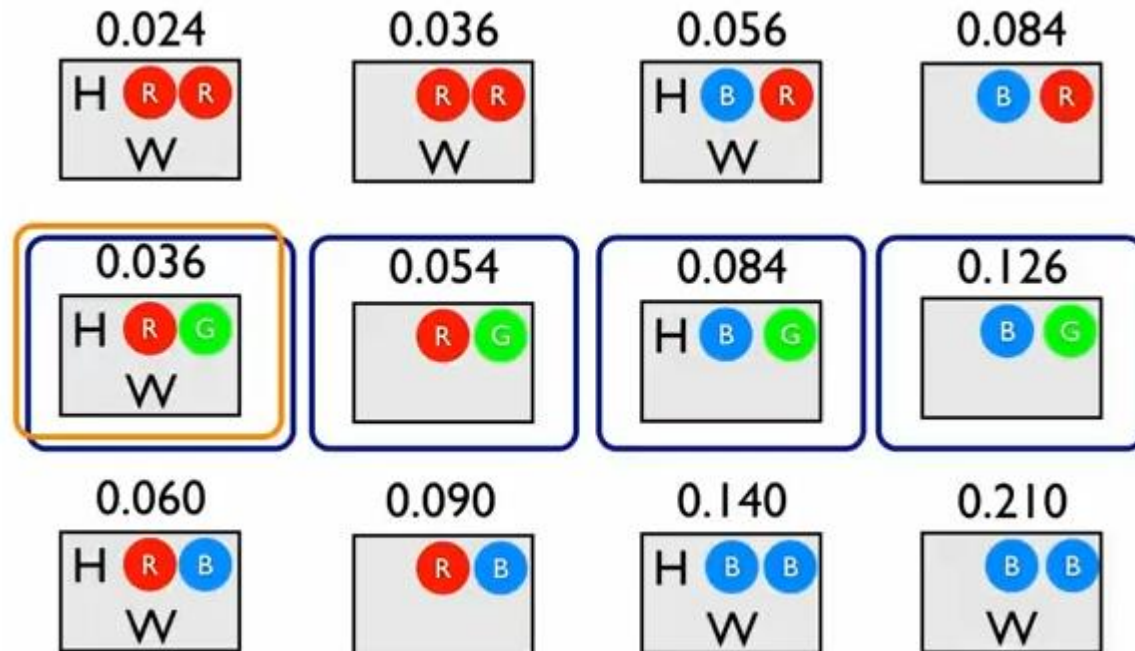


Marginal Probability

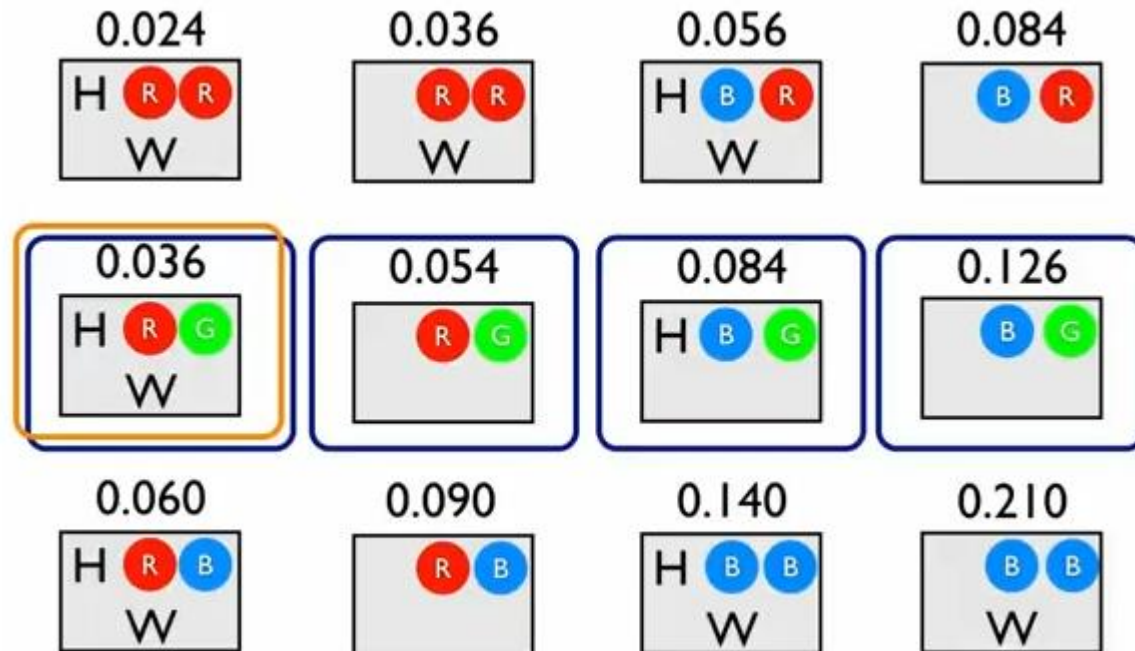


$$P(win) = \sum = 0.562$$

Conditional Probability



Conditional Probability



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

Working with cplint on SWISH



- cplint is a suite of programs for inference and learning of Logic Programs with Annotated Disjunctions.
- cplint is based on SWI Prolog, a logic programming language:
http://friguzzi.github.io/cplint/_build/html/index.html
- cplint can be used for asking queries using exact inference and approximate inference.
- SWISH is a web framework for logic programming.
- You can find cplint on SWISH here:
<http://cplint.lamping.unife.it/>

Example: Most Probable Explanation

The screenshot displays the 'cplint on SWISH' web interface. The left pane contains a Prolog program with the following code:

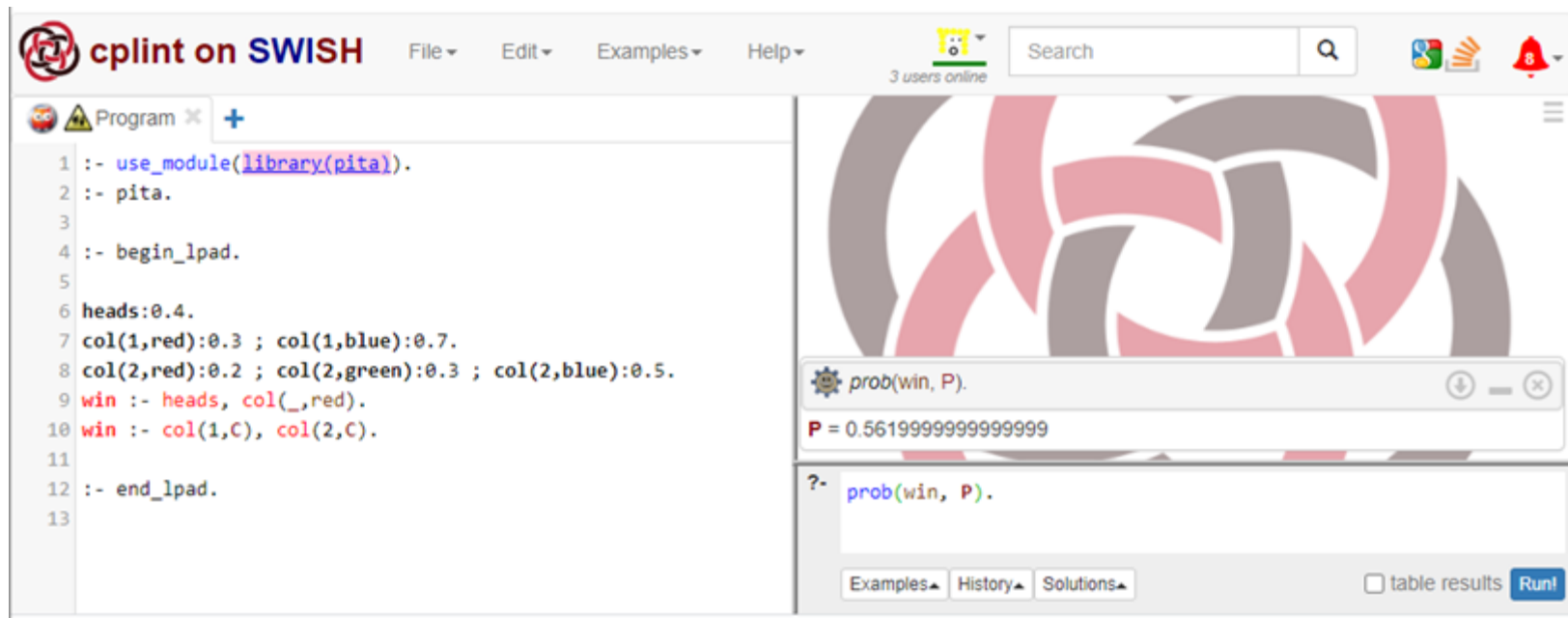
```
1 :- use_module(library(pita)).
2 :- pita.
3
4 :- begin_lpad.
5
6 map_query heads:0.4.
7 map_query col(1,red):0.3 ; col(1,blue):0.7.
8 map_query col(2,red):0.2 ; col(2,green):0.3 ; col(2,blue):0.5.
9 win :- heads, col(_,red).
10 win :- col(1,C), col(2,C).
11
12 :- end_lpad.
13
```

The right pane shows a large, abstract, pink and grey geometric pattern. Below this pattern, a window titled 'map(win,P,Exp)' displays the most probable explanation (MPE) for the query:

```
Exp = [rule(0, " , [heads:0.4, " : 0.6], true), rule(1, col(1, blue), [col(1, red):0.3,
col(1, blue):0.7], true), rule(2, col(2, blue), [col(2, red):0.2, col(2, green):0.3, col(2,
blue):0.5], true)].
P = 0.21
```

At the bottom of the right pane, there is a search bar with the query '?- map(win,P,Exp).' and buttons for 'Examples', 'History', 'Solutions', 'table results', and a 'Run!' button.

Example: Marginal Probability



The screenshot shows the 'cplint on SWISH' web interface. The left pane contains a Prolog program with the following code:

```
1 :- use_module(library(pita)).
2 :- pita.
3
4 :- begin_lpad.
5
6 heads:0.4.
7 col(1,red):0.3 ; col(1,blue):0.7.
8 col(2,red):0.2 ; col(2,green):0.3 ; col(2,blue):0.5.
9 win :- heads, col(_,red).
10 win :- col(1,C), col(2,C).
11
12 :- end_lpad.
13
```

The right pane displays a large, stylized, interlocking geometric pattern in shades of pink and grey. Below this pattern, a small window titled 'prob(win, P).' shows the result: $P = 0.5619999999999999$. At the bottom of the right pane, there is a search bar with the text '?- prob(win, P).', buttons for 'Examples', 'History', and 'Solutions', a checkbox for 'table results', and a 'Run!' button.

Example: Conditional Probability



The screenshot shows the 'cplint on SWISH' web interface. The left pane contains a Prolog program with the following code:

```
1 :- use_module(library(pita)).
2 :- pita.
3
4 :- begin_lpad.
5
6 heads:0.4.
7 col(1,red):0.3 ; col(1,blue):0.7.
8 col(2,red):0.2 ; col(2,green):0.3 ; col(2,blue):0.5.
9 win :- heads, col(_,red).
10 win :- col(1,C), col(2,C).
11
12 :- end_lpad.
13
```

The right pane displays a large geometric pattern in the background. Below it, a small window shows the execution of the query `prob(win, col(2, green), P).` with the result `P = 0.12000000000000009`. At the bottom of the right pane, there is a text input field containing the query `?- prob(win, col(2, green), P).` and buttons for 'Examples', 'History', 'Solutions', 'table results', and 'Run!'.

Example: Code

```
:- use_module(library(pita)).  
:- pita.  
  
:- begin_lpad.  
  
heads:0.4.  
col(1,red):0.3 ; col(1,blue):0.7.  
col(2,red):0.2 ; col(2,green):0.3 ; col(2,blue):0.5.  
win :- heads, col(_,red).  
win :- col(1,C), col(2,C).  
  
:- end_lpad.
```

Explanation

- The library `pita` is used to perform exact inference.
- This library is imported with the built-in predicate `use_module/1`.
- The actual code of the LPAD program is enclosed within two directives:

```
:- begin_lpad.  
% Code goes here ...  
:- end_lpad.
```
- We use a conditional query:

```
?- prob(win, col(2,green), P) .
```

Example: Approximate Inference

The screenshot shows the 'cplint on SWISH' web interface. The left pane contains a Prolog program with the following code:

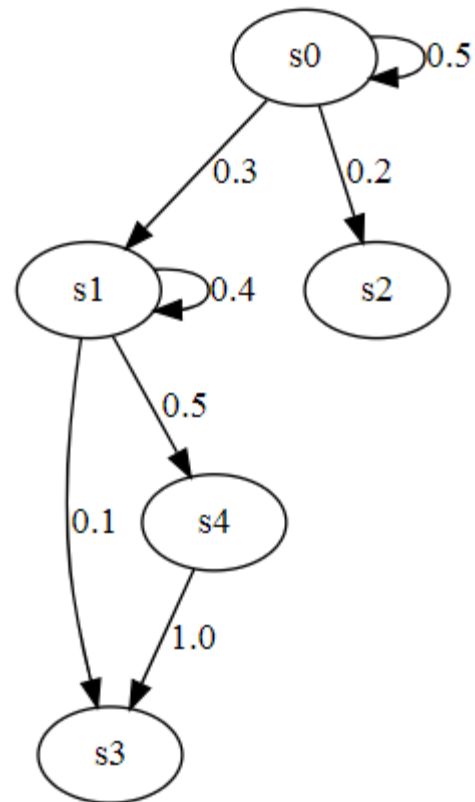
```
1 :- use_module(library(mcintyre)).
2 :- mc.
3
4 :- begin_lpad.
5
6 reach(S, I, T) :- trans(S, I, U), reach(U, next(I), T).
7 reach(S, _, S).
8 trans(s0,s0):0.5; trans(s0,s,s1):0.3; trans(s0,s,s2):0.2.
9 trans(s1,s,s1):0.4; trans(s1,s,s3):0.1; trans(s1,s,s4):0.5.
10 trans(s4,_,s3).
11
12 :- end_lpad.
13
14
```

The right pane displays the execution results for the query `mc_sample(reach(s0, 0, s3), 1000, P, [successes(T), failures(F)])`. The results are:

- F** = 395,
- P** = 0.605,
- T** = 605

Below the results, there is a section for the query `?- mc_sample(reach(s0, 0, s3), 1000, P, [successes(T), failures(F)])` with buttons for 'Examples', 'History', and 'Solutions'. At the bottom right, there is a checkbox for 'table results' and a 'Run!' button.

Markov Chain



Example: Code

```
:- use_module(library(mcintyre)).  
:- mc.  
  
:- begin_lpad.  
  
reach(S, I, T) :- trans(S, I, U), reach(U, next(I), T).  
reach(S, _, S).  
trans(s0,S,s0):0.5; trans(s0,S,s1):0.3; trans(s0,S,s2):0.2.  
trans(s1,S,s1):0.4; trans(s1,S,s3):0.1; trans(s1,S,s4):0.5.  
trans(s4,_,s3).  
  
:- end_lpad.
```

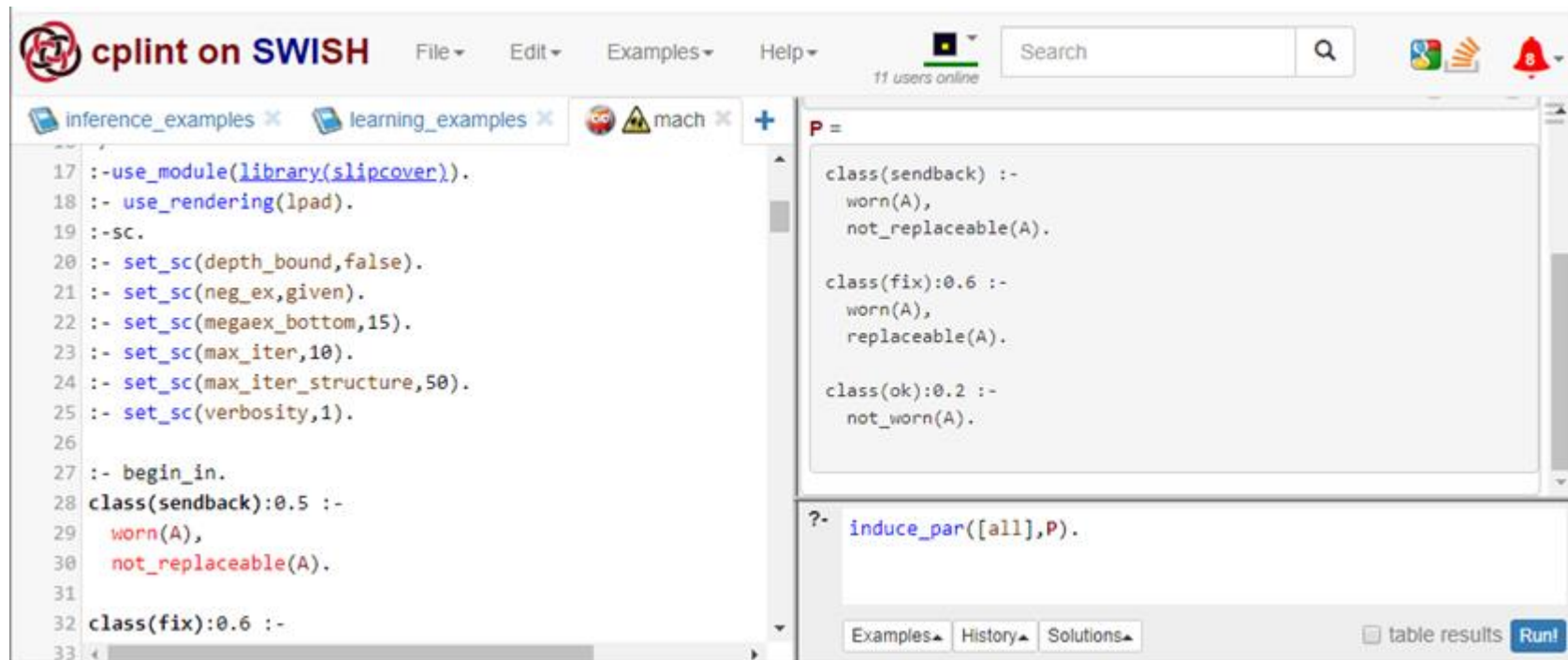
Explanation

- We want to know what is the likelihood that on an execution of a Markov chain from a start state 's', a final state 't' will be reached?
- The transitions between states can be modelled with the predicate **reach/3**.
- Starting at state **S** at instant **I**, state **T** is reachable if there is a transition from state **S** to state **U** at instant **I** and if **T** is reachable from **U** at the next instant.

Explanation

- The chains may be infinite so the query may have an infinite number of explanations.
- Therefore, we use MCINTYRE, a module that performs approximate inference with Monte Carlo sampling.
- Our query samples the predicate `reach(s0, 0, s3)` 1000 times and returns in \mathbf{T} the number of successes, in \mathbf{F} the number of failures and in \mathbf{P} the estimated probability.

Parameter Learning



The screenshot shows the 'cplint on SWISH' web interface. The left pane contains Prolog code for setting up a model and its parameters. The right pane shows the induced model 'P'.

```
17 :-use_module(library(slipcover)).
18 :- use_rendering(lpad).
19 :-sc.
20 :- set_sc(depth_bound,false).
21 :- set_sc(neg_ex,given).
22 :- set_sc(megaex_bottom,15).
23 :- set_sc(max_iter,10).
24 :- set_sc(max_iter_structure,50).
25 :- set_sc(verbosity,1).
26
27 :- begin_in.
28 class(sendback):0.5 :-
29     worn(A),
30     not_replaceable(A).
31
32 class(fix):0.6 :-
```

The right pane displays the induced model 'P':

```
P =
class(sendback) :-
    worn(A),
    not_replaceable(A).

class(fix):0.6 :-
    worn(A),
    replaceable(A).

class(ok):0.2 :-
    not_worn(A).
```

Below the model, the query `?- induce_par([all],P).` is shown. At the bottom right, there are buttons for 'Examples', 'History', 'Solutions', a checkbox for 'table results', and a 'Run!' button.

Explanation

- Parameters for clauses are learned from observations.
- Initial program with default parameters in red:

```
:- begin_in.  
class(sendback):0.5 :-  
    worn(A) ,  
    not_replaceable(A) .  
class(fix):0.5 :-  
    worn(A) ,  
    replaceable(A) .  
class(ok):0.5 :-  
    not_worn(_A) .  
:- end_in.
```

Explanation

```
% Background theory

:- begin_bg.

    component(C) :- replaceable(C).
    component(C) :- not_replaceable(C).
    replaceable(gear).
    replaceable(wheel).
    replaceable(chain).
    not_replaceable(engine).
    not_replaceable(control_unit).
    not_worn(C) :- component(C), \+ worn(C).
    one_worn :- worn(_).
    none_worn :- \+ one_worn.

:- end_bg.
```

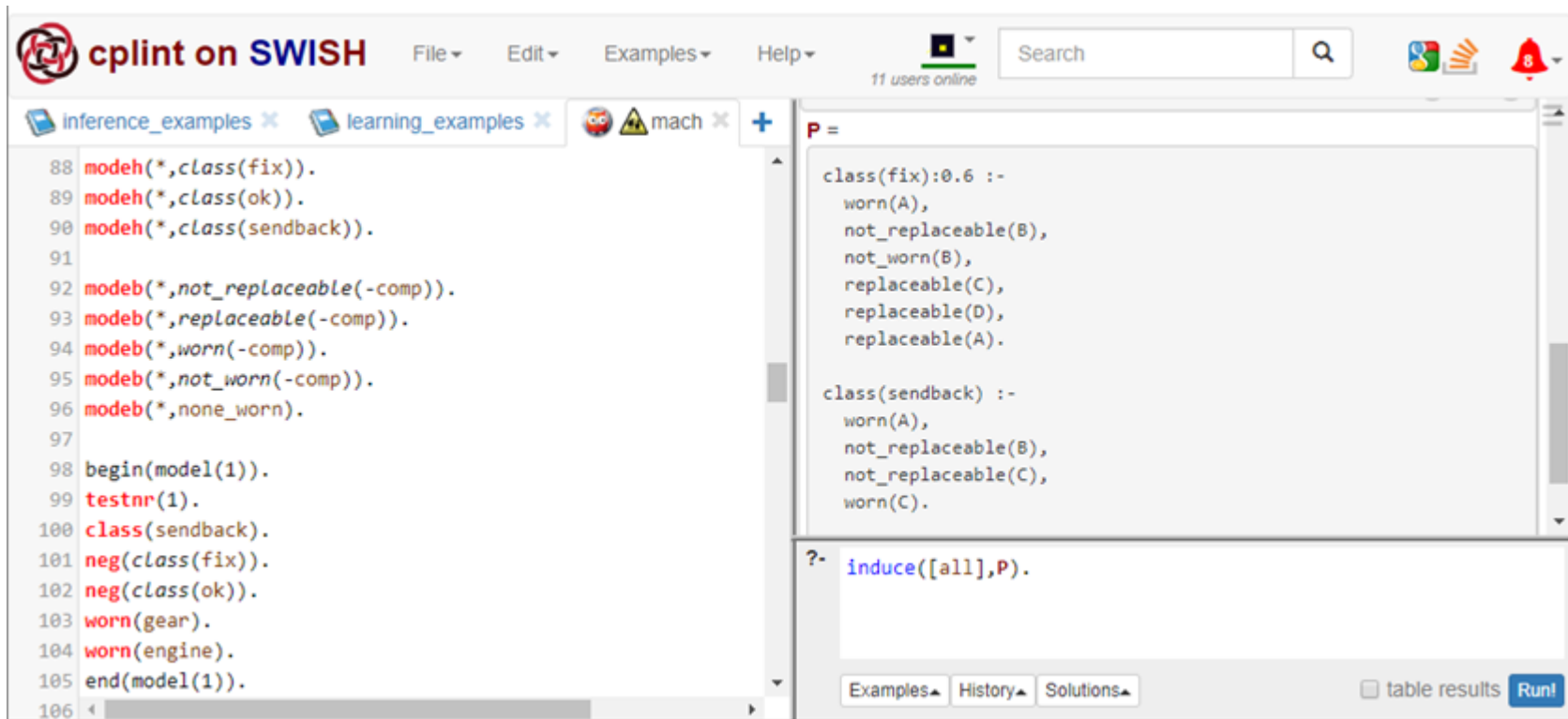
Explanation

- Here is an example of an observation with positive and negative information.

```
% We have to send back a car if the gear is worn and  
% the engine is worn.
```

```
begin(model(1)) .  
    class(sendback) .  
    neg(class(fix)) .  
    neg(class(ok)) .  
    worn(gear) .  
    worn(engine) .  
end(model(1)) .
```

Structure Learning



The screenshot shows the 'cplint on SWISH' web interface. The left pane contains Prolog code for a structure learning task, and the right pane shows the resulting Prolog program.

Left Pane Code:

```
88 modeb(*,class(fix)).
89 modeb(*,class(ok)).
90 modeb(*,class(sendback)).
91
92 modeb(*,not_replaceable(-comp)).
93 modeb(*,replaceable(-comp)).
94 modeb(*,worn(-comp)).
95 modeb(*,not_worn(-comp)).
96 modeb(*,none_worn).
97
98 begin(model(1)).
99 testnr(1).
100 class(sendback).
101 neg(class(fix)).
102 neg(class(ok)).
103 worn(gear).
104 worn(engine).
105 end(model(1)).
106
```

Right Pane Code:

```
P =
class(fix):0.6 :-
    worn(A),
    not_replaceable(B),
    not_worn(B),
    replaceable(C),
    replaceable(D),
    replaceable(A).

class(sendback) :-
    worn(A),
    not_replaceable(B),
    not_replaceable(C),
    worn(C).

?- induce([all],P).
```

At the bottom right, there are buttons for 'Examples', 'History', 'Solutions', a checkbox for 'table results', and a 'Run!' button.

Explanation

- Structure learning is more complex than parameter learning (it actually subsumes parameter learning).
- Structure learning requires mode declarations that specify what goes into the head and body of a rule:

```
modeh(*,class(fix)).  
modeh(*,class(ok)).  
modeh(*,class(sendback)).
```

```
modeb(*,not_replaceable(-comp)).  
modeb(*,replaceable(-comp)).  
modeb(*,worn(-comp)).  
modeb(*,not_worn(-comp)).  
modeb(*,none_worn).
```

Take-Home Messages

- Combining rules with probabilities is an emerging field of artificial intelligence, known as statistical relational learning.
- Probabilistic logic programming introduces probabilistic reasoning in logic programs in order to represent uncertain information.
- Reasoning can be exact or approximate.
- Parameters of clauses for probabilistic logic programs can be learned from observations.
- Entire structure of clauses for probabilistic logic programs can be learned from observations and mode declarations.