Probabilistic space partitioning in Constraint Logic Programming

Nicos Angelopoulos
nicos@cs.york.ac.uk

http://www.cs.york.ac.uk/~nicos

Department of Computer Science

University of York

talk structure

- Motivating example
- Logic Programming (LP)
- Uncertainty and LP
- Constraint LP
- clp(pfd(Y))
- clp(pfd(c))
- Three prisoners revisited
- Conclusions

three prisoners, (Mosteller, 1965)

Grünwald and Halpern (2003):

> Of three prisoners a, b, and c, two are to be executed, but a does not know which. Thus, a thinks that the probability that i will be executed is 2/3 for $i \in \{a, b, c\}$. He says to the jailer, "Since either b or c is certainly going to be executed, you will give me no information about my own chances if you give the name of one man, either b or c, who is going to be executed." But then, no matter what the jailer says, naive conditioning leads a to believe that his chance of execution went down from 2/3 to 1/2.

probabilistic spaces

Unconditional space $W = \{w_a, w_b, w_c\}$

Observations $O = \{o_b, o_c\}$

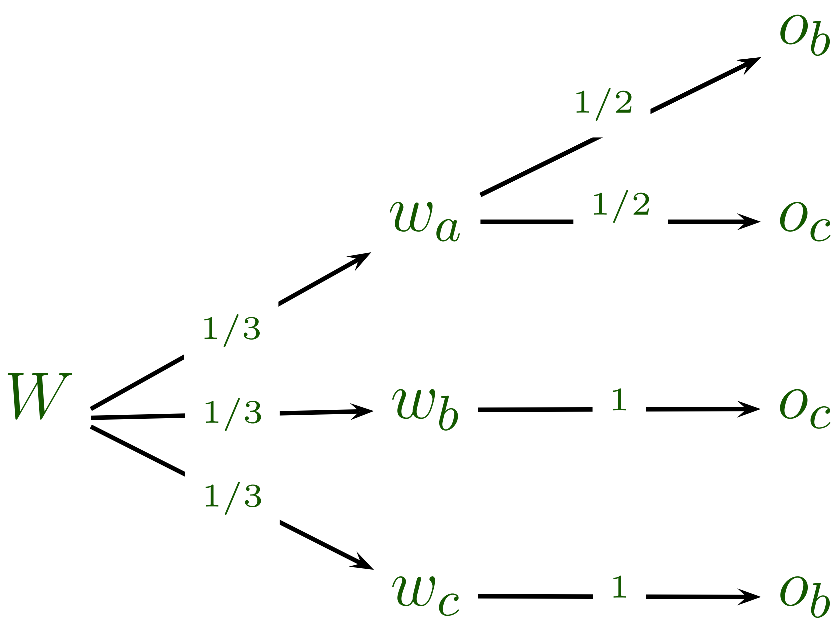
Naive space

$$N^{O_b} = \{w_a, w_c\}$$
$$N^{O_c} = \{w_a, w_b\}$$

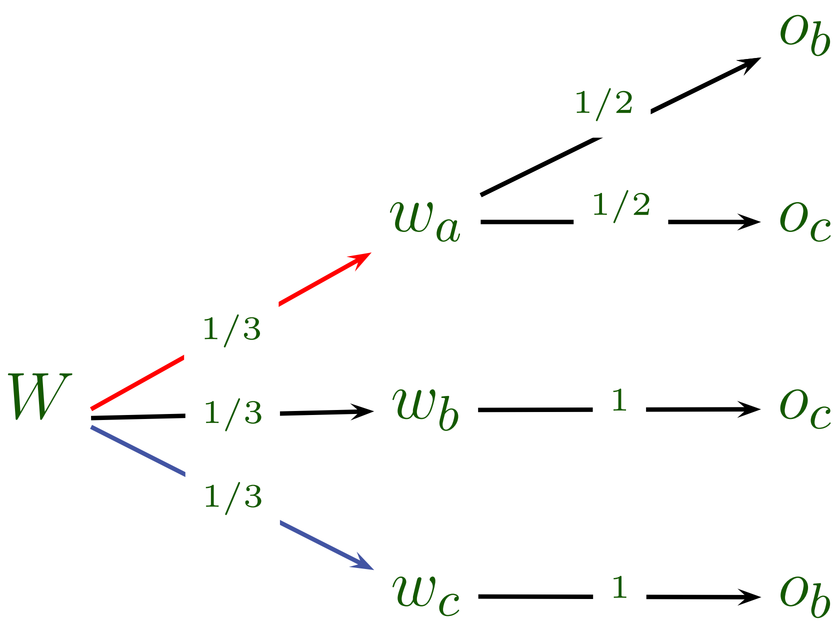
Sophisticated space

$$S^{O_b} = \{(w_a, o_b), (w_c, o_b)\}$$
$$S^{O_c} = \{(w_a, o_c), (w_b, o_c)\}$$

Graph representation



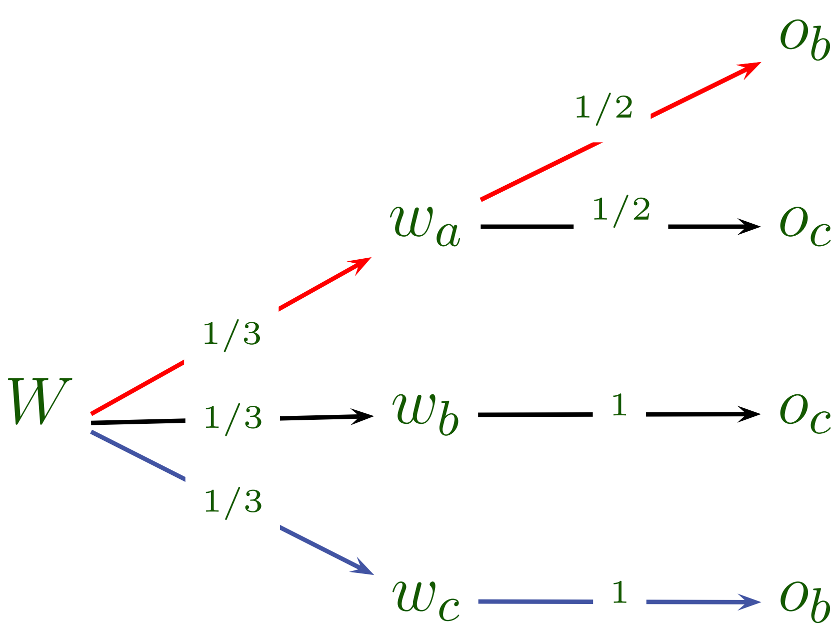
Graph representation



For $O = o_b$:

On naive space compute $P(W = w_a) = \frac{1/3}{1/3+1/3}$

Graph representation



For $O = o_b$:

On naive space compute $P(W = w_a) = \frac{1/3}{1/3+1/3}$

On sophisticated compute $P(W = w_a | O = o_b) = \frac{1/6}{1/6+1/3}$

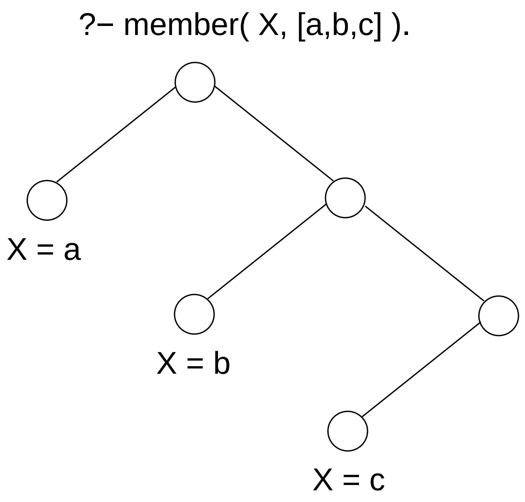
logic programming

Used in AI for crisp problem solving and for building executable models and intelligent systems.

Programs are formed from logic based rules.

```
member( H, [H|T] ).
member( El, [H|T] ) :- member( El, T ).
```

execution tree



member(H, [H|T]).

member(El, [H|T]) :- member(El, T).

uncertainty in logic programming

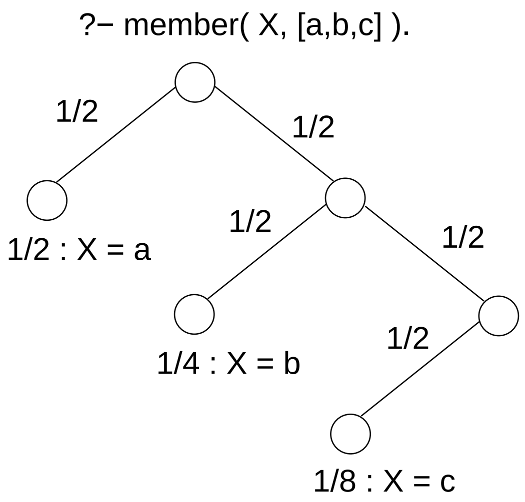
Most approaches use Probability Theory but there are fundamental questions unresolved.

In general,

0.5 : member(H, [H|T]).

0.5 : member(El, [H|T]) :- member(El, T).

stochastic tree



0.5 : member(H, [H|T]).

0.5 : member(El, [H|T]) :- member(El, T).

constraints in lp

Logic Programming :

- execution model is inflexible, and
- its relational nature discourages use of state information.

constraints in lp

Logic Programming :

- execution model is inflexible, and
- its relational nature discourages use of state information.

Constraints add

- specialised algorithms

constraints in lp

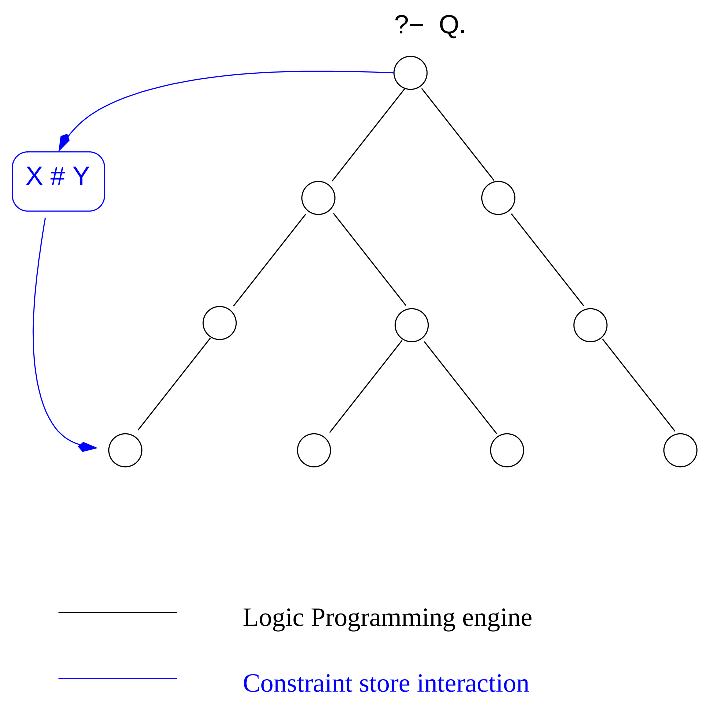
Logic Programming :

- execution model is inflexible, and
- its relational nature discourages use of state information.

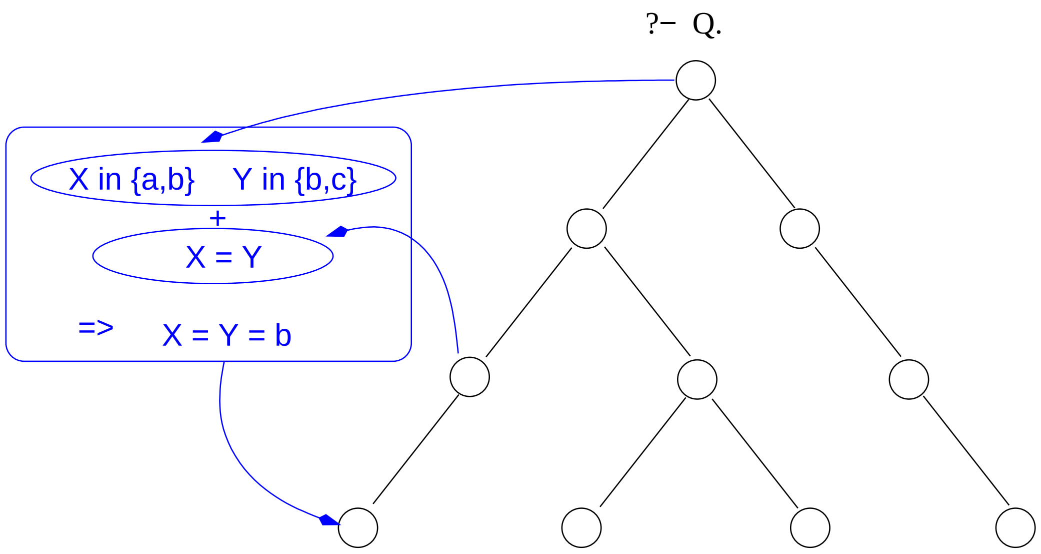
Constraints add

- specialised algorithms
- state information

constraint store



constraints inference



finite domain distributions

For discrete probabilistic models clp(pfd(Y)) extends the idea of finite domains to admit distributions.

from clp(fd)

$X \; in \; \{a, b\}$ (i.e. $X = a$ or $X = b$)

to clp(pfd(Y))

$$p(X = a) + p(X = b)$$

finite domain distributions

For discrete probabilistic models clp(pfd(Y)) extends the idea of finite domains to admit distributions.

from clp(fd)

$$X\ in\ \{a, b\} \qquad \text{(i.e. } X = a \quad \text{or} \quad X = b)$$

to clp(pfd(Y))

$$[\ p(X = a) + p(X = b)\ \ ] = 1$$

constraint based integration

Execution, assembles the probabilistic model in the store according to program and query.

Dedicated algorithms can be used for probabilistic inference on the model present in the store.

probability of predicates

- $pvars(E)$ - variables in predicate E,
- e - vector of finite domain elements
- $p(e_i)$ - probability of element e_i
- $\mathcal{S}$ - a constraint store.
- E/e - E with variables replaced by e.

The probability of predicate E with respect to store $\mathcal{S}$ is

$$P_{\mathcal{S}}(E) = \sum_{\substack{\forall e \\ \mathcal{S} \vdash E/e}} P_{\mathcal{S}}(e) = \sum_{\substack{\forall e \\ \mathcal{S} \vdash E/e}} \prod_i p(e_i)$$

clp(pfd(Y))

is a generic framework for probabilistic inference in CLP.
For example if the store can infer distributions

$$Dice - [i : 1/6, ii : 1/6, iii : 1/6, iv : 1/6, v : 1/6, vi : 1/6]$$

$$Coin - [head : 1/2, tail : 1/2]$$

and program defines

```
lucky( iv, head ).
lucky( v, head ).
lucky( vi, head ).
```

$$P(lucky(Dice, Coin)) = 1/4$$

clp(pfd(c))

Probabilistic variable definitions

$$Coin \quad \sim \quad finite\_geometric([h, m, l], 2)$$

clp(pfd(c))

Probabilistic variable definitions

$$Coin \quad \sim \quad finite\_geometric([h, m, l], 2)$$

If store allows $[h, m, l]$ for $Coin$ then

$$Coin - [h : 4/7, m : 2/7, l : 1/7]$$

clp(pfd(c))

Probabilistic variable definitions

$$Coin \quad \sim \quad finite\_geometric([h, m, l], 2)$$

If store allows $[h, m, l]$ for $Coin$ then

$$Coin - [h : 4/7, m : 2/7, l : 1/7]$$

If store allows $[h, l]$ for $Coin$ then

$$Coin - [h : 2/3, l : 1/3]$$

pfd(c) example

p_of_lucky(P) :-

$$Dice \sim uniform([i, ii, iii, iv, v, vi]),$$
$$Coin \sim uniform([head, tail]),$$
P is p($lucky(Dice, Coin)$).

$$?-p\_of\_lucky(LuckyP).$$

$$LuckyP = 1/4$$

conditional

Conditional constraint

$$D_1 : \pi_1 \oplus \ldots \oplus D_m : \pi_m \mid Q$$

conditional

Conditional constraint

$$D_1 : \pi_1 \oplus \ldots \oplus D_m : \pi_m \mid Q$$

Conditional difference is a special case

$$Dependent \mid\neq_{\pi} Qualifier$$

$$Dependent \neq V : \pi \oplus Dependent = V : (1-\pi) \mid Qualifier = V$$

Variable elimination

Algorithm: Compute probability of event
Input: Query Q and store S. Output: $P_{\mathcal{S}}(Q)$
Initialise:

- Construct dependency graph G for $pvars(Q)$.
- Find a topological ordering O of G.
- Place $pvars(Q)$ to B_0. Place each O_i and $dep(O_i)$ in B_i.

Iterate:

For i = n to 1

compute $P_{\mathcal{S}}(O_i)$ according to (Eq1)

add $P_{\mathcal{S}}(O_i)$ to each remaining bucket that mentions O_i

Compute:

- updated $P_{\mathcal{S}}(Q)$ based on probabilities of $pvars(Q)$ in B_0

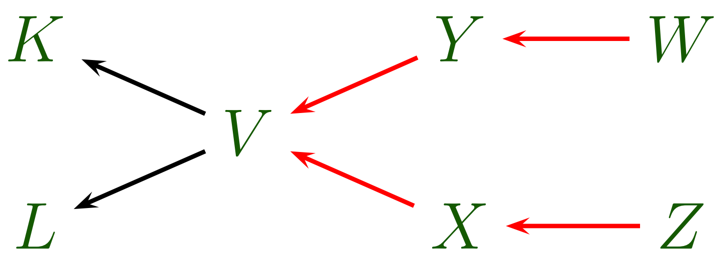
graph operation of algorithm

$$?-p(f(V)).$$

$$\begin{array}{ccccc} K & & & Y & \leftarrow W \\ & \nwarrow & & \swarrow & \\ & & V & & \\ & \swarrow & & \nwarrow & \\ L & & & X & \leftarrow Z \end{array}$$

graph operation of algorithm

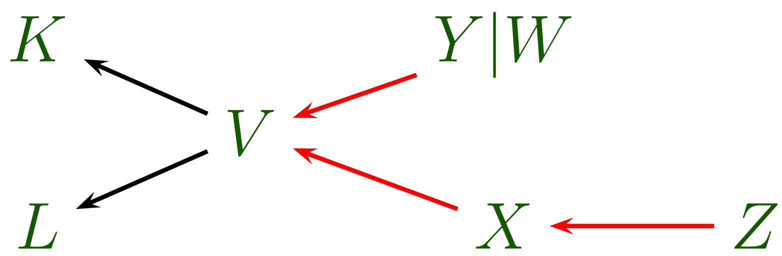
$$?-p(f(V)).$$



A valid ordering: $\{W, Z, Y, X, V\}$

graph operation of algorithm

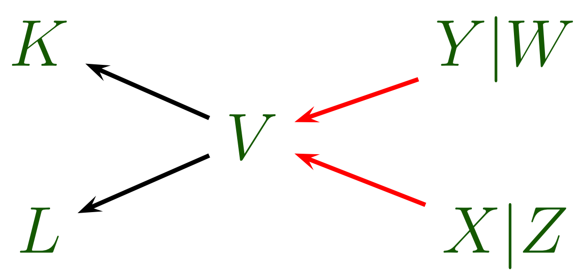
$$?-p(f(V)).$$



A valid ordering: $\{W, Z, Y, X, V\}$

graph operation of algorithm

$$?-p(f(V)).$$



A valid ordering: $\{W, Z, Y, X, V\}$

graph operation of algorithm

$$?-p(f(V)).$$

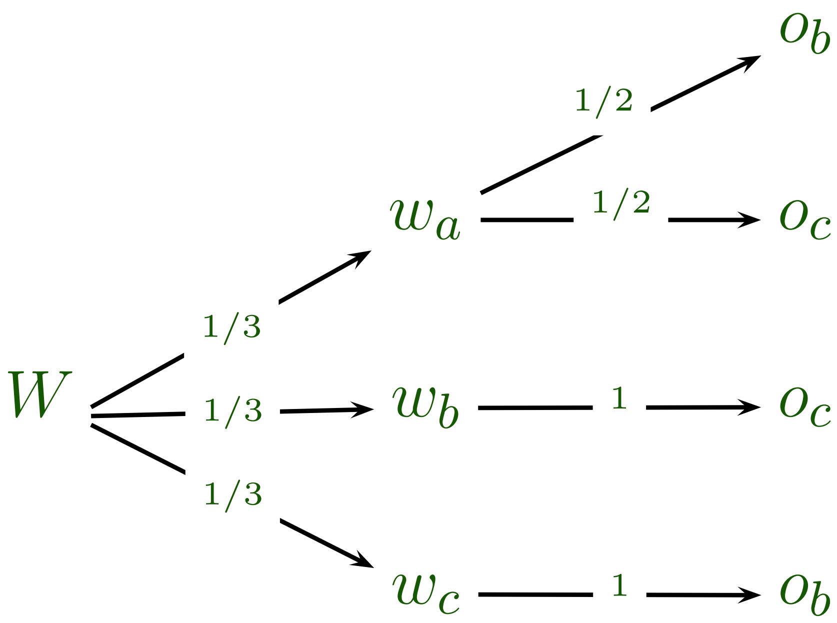
$$V|Y, X, W, Z$$

three prisoners model

$$
\begin{aligned}
tp(\ Obs, AWins\ ) :- \\
&W \;\sim\; uniform([a,b,c]), \\
&O \;\sim\; uniform([b,c]), \\
&O \neq W, \\
&AWins \;\; \text{is} \;\; \text{p(}\ a = W | O = Obs\ \text{)}\ .
\end{aligned}
$$

three prisoners computation

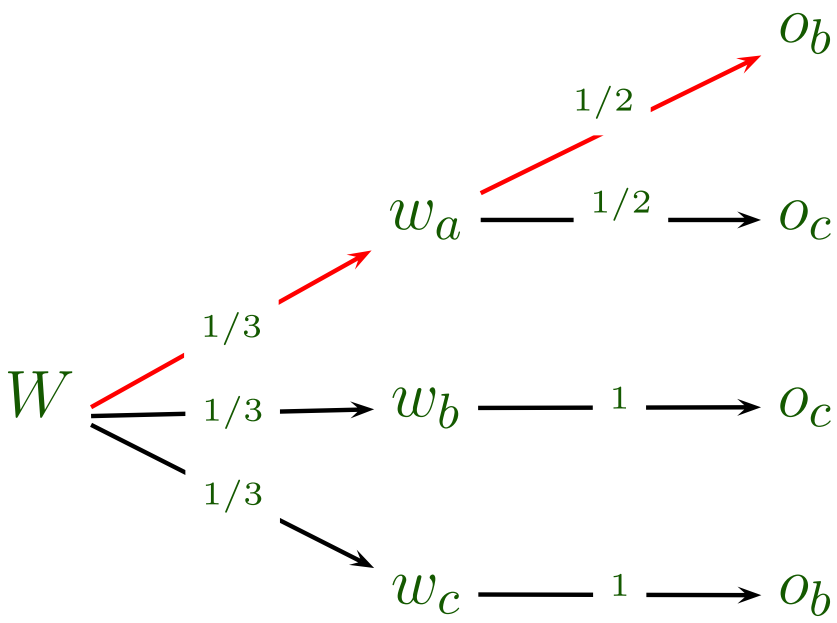
$$P(W = w_a | O = Obs) = P(W = w_a, O = Obs)/P(O = Obs)$$



$$P(W = w_a | O = o_b) = P(W = w_a, O = o_b)/P(O = o_b)$$

three prisoners computation

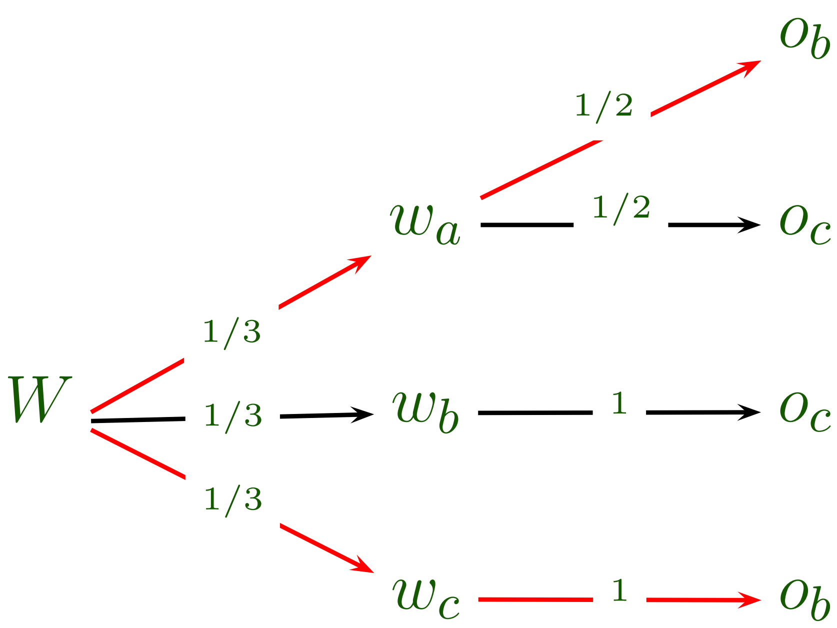
$$P(W = w_a | O = Obs) = P(W = w_a, O = Obs)/P(O = Obs)$$



$$P(W = w_a | O = o_b) = P(W = w_a, O = o_b)/P(O = o_b) = \frac{1}{6}$$

three prisoners computation

$$P(W = w_a | O = Obs) = P(W = w_a, O = Obs) / P(O = Obs)$$



$$P(W = w_a | O = o_b) = P(W = w_a, O = o_b) / P(O = o_b) = \frac{1}{6} / \frac{1}{2} = \frac{1}{3}$$

bottom line

Constraint LP based techniques can be used for frameworks that support probabilistic problem solving.

clp(pfd(Y)) can be used to take advantage of probabilistic information at an abstract level.

References

Grünwald, P., & Halpern, J. (2003). Updating Probabilities. *Journal of AI Research*, *19*, 243–278.

Mosteller, F. (1965). *Fifty challenging problems in probability, wi th solutions*. Addison-Wesley.