Questions & Answers

Gloria Beraldo (gloria.beraldo@unipd.it)
Department of Information Engineering, University of Padova

Topics:

- Lab 3: report function
- Lab 4: cut in Prolog
- Lab 5: chain rule to extract formula
- Lab 5: numpy to save probabilities
- Lab 7: getPosterior
- Lab 7: Sprinkler example via pyAgrum

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Question 1: Lab3

Riguardo la richiesta dell'ultimo esercizio, non riesco a capire cosa si intende per numero di azioni, in quanto applicando l'algoritmo e contando come numero di azioni le scelte possibili di espansione da un nodo che possono essere effettuate, il risultato è molto minore delle scelte possibili per la risposta

```
def report(searchers, problems, verbose=True):
    """Show summary statistics for each searcher (and on each problem unless verbose is false)."""
    for searcher in searchers:
        print(searcher.__name__ + ':')
        total_counts = Counter()
        for p in problems:
            prob   = CountCalls(p)
            soln   = searcher(prob)
            counts = prob._counts;
            print(type(soln))
            counts.update(actions=len(soln), cost=soln.path_cost)
            total_counts += counts
            #if verbose: report_counts(counts, str(p)[:40])
        #report_counts(total_counts, 'TOTAL\n')
```

I add **this line** to verify the type

From Colab Lab3

Question 1: Lab3

Riguardo la richiesta dell'ultimo esercizio, non riesco a capire cosa si intende per numero di azioni, in quanto applicando l'algoritmo e contando come numero di azioni le scelte possibili di espansione da un nodo che possono essere effettuate, il risultato è molto minore delle scelte possibili per la risposta

Now, I call the report function with an example of problem and searcher

```
report([uniform_cost_search], [c1])
```

```
uniform_cost_search:
<class '__main__.Node'>
```

```
class Node:
    "A Node in a search tree."
    def __init__(self, state, parent=None, action=None, path_cost=0):
        self.__dict__.update(state=state, parent=parent, action=action, path_cost=path_cost)

    def __repr__(self): return '<{}>'.format(self.state)
    def __len__(self): return 0 if self.parent is None else (1 + len(self.parent))
    def __lt__(self, other): return self.path_cost < other.path_cost
```

From Colab Lab3

Question 1: Lab3

Riguardo la richiesta dell'ultimo esercizio, non riesco a capire cosa si intende per numero di azioni, in quanto applicando l'algoritmo e contando come numero di azioni le scelte possibili di espansione da un nodo che possono essere effettuate, il risultato è molto minore delle scelte possibili per la risposta

```
def report(searchers, problems, verbose=True):
    """Show summary statistics for each searcher (and on each problem unless verbose is false)."""
    for searcher in searchers:
        print(searcher.__name__ + ':')
        total_counts = Counter()
        for p in problems:
            prob   = CountCalls(p)
            soln   = searcher(prob)
            counts = prob._counts;
            print(type(soln))
            counts.update(actions=len(soln), cost=soln.path_cost)
            total_counts += counts
            #if verbose: report_counts(counts, str(p)[:40])
        #report_counts(total_counts, 'TOTAL\n')
```

From Colab Lab3

Question 1: Lab3

Riguardo la richiesta dell'ultimo esercizio, non riesco a capire cosa si intende per numero di azioni, in quanto applicando l'algoritmo e contando come numero di azioni le scelte possibili di espansione da un nodo che possono essere effettuate, il risultato è molto minore delle scelte possibili per la risposta

Now, I call the report function with an example of problem and searcher

```
report([uniform_cost_search], [c1])
```

```
uniform_cost_search:
<class '__main__.Node'>
```

```
class Node:
    "A Node in a search tree."
    def __init__(self, state, parent=None, action=None, path_cost=0):
        self.__dict__.update(state=state, parent=parent, action=action, path_cost=path_cost)

    def __repr__(self): return '<{}>'.format(self.state)
    def __len__(self): return 0 if self.parent is None else (1 + len(self.parent))
    def __lt__(self, other): return self.path_cost < other.path_cost
```

From Colab Lab3

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = history

There are not Student
The second goal fails
so **backtracking**

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = english

There are two Student alice and angus

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = english

There are two Student alice and angus then **backtracking**

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = drama

There are one Student amelia

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = drama

There are one Student
amelia
then **backtracking**

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let’s consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Finish

```
?- teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

```
?- trace, teaches(dr_fred, Course), studies(Student, Course).
        Call: teaches(dr_fred,_4060)
        Exit: teaches(dr_fred,history)
        Call: studies(_504,history)
        Fail: studies(_504,history)
        Redo: teaches(dr_fred,_508)
        Exit: teaches(dr_fred,english)
        Call: studies(_504,english)
        Exit: studies(alice,english)
Course = english,
Student = alice
        Redo: studies(_504,english)
        Exit: studies(angus,english)
Course = english,
Student = angus
        Redo: teaches(dr_fred,_502)
        Exit: teaches(dr_fred,drama)
        Call: studies(_498,drama)
        Exit: studies(amelia,drama)
Course = drama,
Student = amelia
```

Remember **trace** for debug

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let's consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = history

The cut is executed (i.e., True)
There are not Student
The second goal fails
No backtracking can be applied since the cut
NO VARIABLE BINDING

```
?- teaches(dr_fred, Course), !, studies(Student, Course).
false
```

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let's consider the following example:
NB: With respect to previous slide**, I add the fact highlighted in orange**

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, history).
6 studies(alice, english).
7 studies(angus, english).
8 studies(amelia, drama).
9 studies(alex, physics).
```

Course = history

The cut is executed (i.e., True)
There is one student that satisfies
The second goal (alice)
No backtracking can be applied since the cut
Finish

```
?- teaches(dr_fred, Course), !, studies(Student, Course).
```

Course = history,
Student = alice

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let's consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = history

There are not Student
The second goal fails
NB: We cannot get the cut
so **backtracking is possible**

```
?- teaches(dr_fred, Course), studies(Student, Course), !.
```

Course = english,
Student = alice

Question 2: Lab4

Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let's consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

Course = English

There is one Student alice
Cut goal is tried (succeed of course)
No backtracking can be applied since the cut
Finish

```
?- teaches(dr_fred, Course), studies(Student, Course), !.
```

Course = english,
Student = alice

Question 2: Lab4

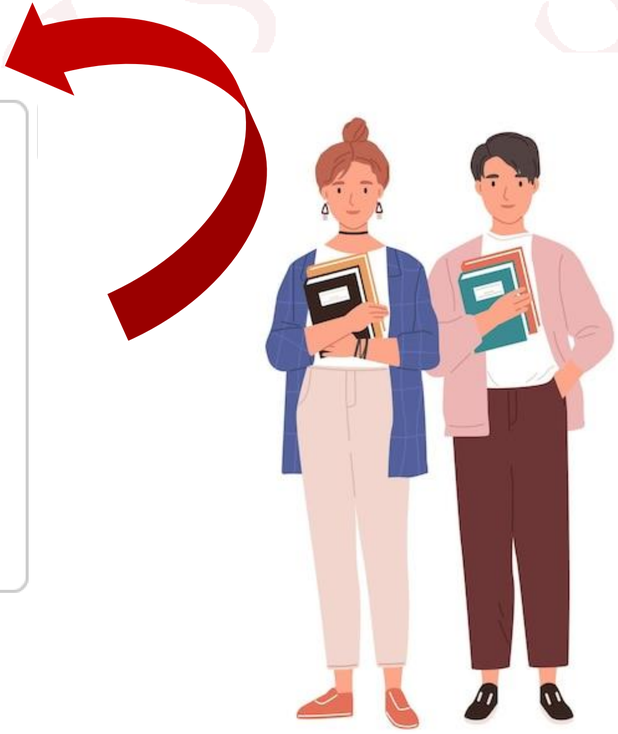
Come funziona il cut

The cut, in Prolog, is a goal, written as **!**, which always succeeds, but **cannot be backtracked** past.

Let's consider the following example:

```
1 teaches(dr_fred, history).
2 teaches(dr_fred, english).
3 teaches(dr_fred, drama).
4 teaches(dr_fiona, physics).
5 studies(alice, english).
6 studies(angus, english).
7 studies(amelia, drama).
8 studies(alex, physics).
```

the same solutions are found as if no cut was present, because it is **never necessary to backtrack past the cut** to find the next solution, so **backtracking is never inhibited**.



```
?- !, teaches(dr_fred, Course), studies(Student, Course).
```

Course = english,
Student = alice
Course = english,
Student = angus
Course = drama,
Student = amelia

Question 2: Lab4

Come funziona il cut

It constraints Prolog to choices made since the parent goal was called (the goal that used the clause containing the cut).

It cuts the alternatives left open for backtracking since the parent goal has been unified with the head of the rule used (including the choice to use that clause).

```
antibodies_owner(maria).
dance(linda).
sing(linda).
sing(gianni).
noproblems(vittorio).

healthy(X) :- happy(X).
healthy(X) :- antibodies_owner(X).
happy(X) :- sing(X), dance(X).
happy(X) :- noproblems(X).
```

```
?- healthy(X)
```

X = linda
X = vittorio
X = maria

Question 2: Lab4

Come funziona il cut

It constraints Prolog to choices made since the parent goal was called (the goal that used the clause containing the cut).

It cuts the alternatives left open for backtracking since the parent goal has been unified with the head of the rule used (including the choice to use that clause).

```
1  antibodies_owner(maria).
2  dance(linda).
3  sing(linda).
4  sing(gianni).
5  noproblems(vittorio).
6
7  healthy(X) :- happy(X).
8  healthy(X) :- antibodies_owner(X).
9  happy(X) :- sing(X), ! dance(X).
10 happy(X) :- noproblems(X).
```

```
?- healthy(X)
X = linda
X = maria
```

Question 2: Lab4

Come funziona il cut

It constraints Prolog to choices made since the parent goal was called (the goal that used the clause containing the cut).

It cuts the alternatives left open for backtracking since the parent goal has been unified with the head of the rule used (including the choice to use that clause).

```
1  antibodies_owner(maria).
2  dance(linda).
3  sing(linda).
4  sing(gianni).
5  noproblems(vittorio).
6
7  healthy(X) :- happy(X).
8  healthy(X) :- antibodies_owner(X).
9  happy(X) :- sing(X), dance(X), !.
10 happy(X) :- noproblems(X).
```

```
?- healthy(X)
X = linda
X = maria
```

Question 2: Lab4

Come funziona il cut

It constraints Prolog to choices made since the parent goal was called (the goal that used the clause containing the cut).

It cuts the alternatives left open for backtracking since the parent goal has been unified with the head of the rule used (including the choice to use that clause).

```
1  antibodies_owner(maria).
2  dance(linda).
3  sing(linda).
4  sing(gianni).
5  noproblems(vittorio).
6
7  healthy(X) :- happy(X).
8  healthy(X) :- antibodies_owner(X).
9  happy(X) :- !, sing(X), dance(X).
10 happy(X) :- noproblems(X).
```

```
?- healthy(X)
X = linda
X = maria
```

Question 2: Lab4

Come funziona il cut

It constraints Prolog to choices made since the parent goal was called (the goal that used the clause containing the cut).

It cuts the alternatives left open for backtracking since the parent goal has been unified with the head of the rule used (including the choice to use that clause).

```
1  antibodies_owner(maria).
2  dance(linda).
3  sing(linda).
4  sing(gianni).
5  noproblems(vittorio).
6
7  healthy(X) :- happy(X).
8  healthy(X) :- antibodies_owner(X).
9  happy(X) :- sing(X), dance(X).
10 happy(X) :- ! noproblems(X).
```

```
?- healthy(X)
X = linda
X = vittorio
X = maria
```

Question 3: Lab5

Non ho capito la sommatoria, in particolare su quale lettere deve essere operata. Deve essere eseguita su eventi che sono già conosciuti? Ad esempio: Avendo ~~P(A|B)= alpha*P(A|B)=~~ alpha * SOMM(B){P(A|B)}

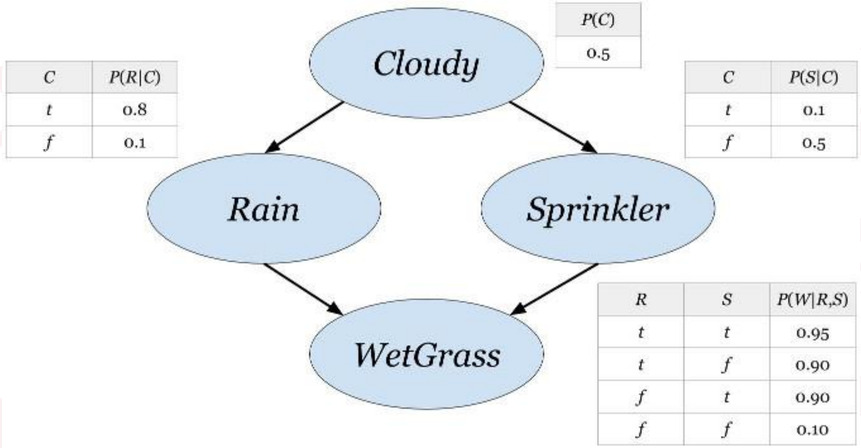
Pay attention the relation would be $P(A|B) = \alpha\, P(A,B)$

Example: Slide 12

$$P(W|c) = \alpha\, P(W,c)$$

α *normalization constant*

$$= \alpha \sum_{r,s} \boldsymbol{P}(W, c, r, s)$$



Question 3: Lab5

Non ho capito la sommatoria, in particolare su quale lettere deve essere operata. Deve essere eseguita su eventi che sono già conosciuti? Ad esempio: Avendo ~~P(A|B)= alpha*P(A|B)~~= alpha * SOMM(B){P(A|B)}

Then you should apply the chain rule to simplify the joint probability

Chain rule

- To simplify, let's start by decomposing the joint distribution in a product of conditional probabilities, a procedure called **chain rule**

$$
\begin{aligned}
P(x_1, \ldots, x_n) &= P(x_n | x_{n-1}, \ldots, x_1) P(x_{n-1}, \ldots, x_1) \\
&= P(x_n | x_{n-1}, \ldots, x_1) P(x_{n-1} | x_{n-2}, \ldots, x_1) P(x_{n-2}, \ldots, x_1) \\
&= \ldots \\
&= P(x_n | x_{n-1}, \ldots, x_1) P(x_{n-1} | x_{n-2}, \ldots, x_1) \ldots P(x_2 | x_1) P(x_1) \\
&= \mathbf{\Pi}_j P(x_j | x_1, \ldots, x_{j-1})
\end{aligned}
$$

- For example

$$P(x_1, x_2, x_3) = P(x_3 | x_1, x_2) P(x_2 | x_1) P(x_1)$$

Slide 5 by prof. Bellotto (Bayesian Networks (part 1))

Question 3: Lab5

Non ho capito la sommatoria, in particolare su quale lettere deve essere operata. Deve essere eseguita su eventi che sono già conosciuti? Ad esempio: Avendo ~~P(A|B)= alpha*P(A|B)~~= alpha * SOMM(B){P(A|B)}

Then you should apply the chain rule to simplify the joint probability

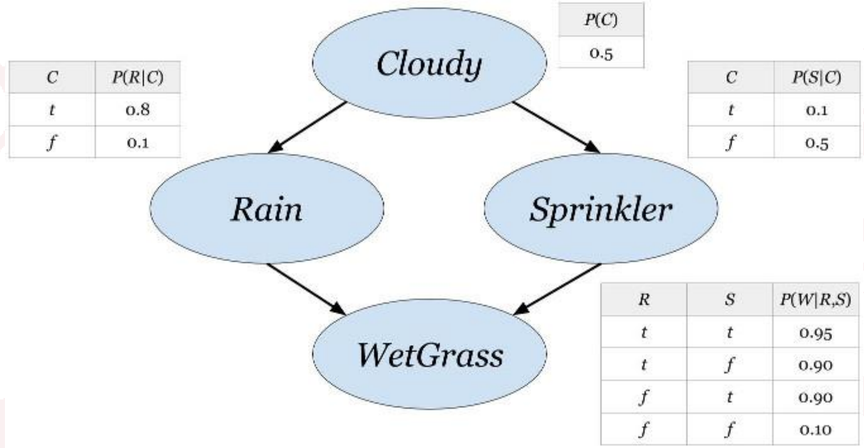
Example: Slide 12

$$\boldsymbol{P}(W|c) = \alpha\, \boldsymbol{P}(W,c) \qquad \alpha \text{ normalization constant}$$

$$= \alpha \sum_{r,s} \boldsymbol{P}(W, c, r, s)$$

$$= \alpha \sum_{r,s} \boldsymbol{P}(W|r,s)P(r|c)P(s|c)P(c)$$

$$= \alpha\, P(c) \sum_{r} P(r|c) \sum_{s} \boldsymbol{P}(W|r,s)P(s|c)$$



Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

```
# 'true' and 'false' indexes
t, f = 0, 1
```

$\mathbf{P}(W|R,S)$:

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
```

P(S|c) :

```
P_S_c = np.array([0.1, 0.9])
```

From Colab Lab5

Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

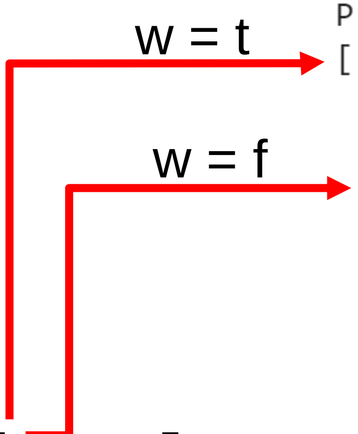
```
# 'true' and 'false' indexes
t, f = 0, 1
```

$\mathbf{P}(W|R,S)$:

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
print(P_W_RS)
```

```
P(w|¬r,s) =  0.9
[[[0.95 0.9 ]
  [0.9  0.1 ]]

 [[0.05 0.1 ]
  [0.1  0.9 ]]]
```



P_W_RS[:, :, :]

The order (t,f) is defined according to the indexes

From Colab Lab5

Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

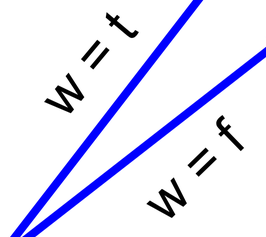
```
# 'true' and 'false' indexes
t, f = 0, 1
```

$\mathbf{P}(W|R,S)$:

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
print(P_W_RS)
```

```
P(w|¬r,s) =  0.9
[[[0.95 0.9 ]  r = t
  [0.9  0.1 ]] r = f

 [[0.05 0.1 ]  r = t
  [0.1  0.9 ]] r = f
```



P_W_RS[:, :, :]

The order (t,f) is defined according to the indexes

From Colab Lab5

Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

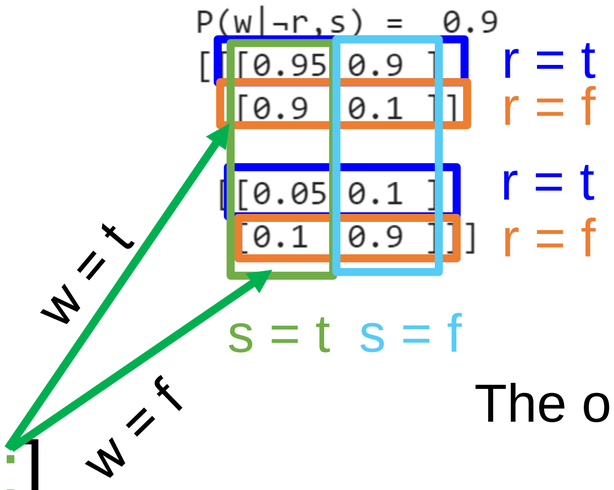
```
# 'true' and 'false' indexes
t, f = 0, 1
```

$\mathbf{P}(W|R,S)$:

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
print(P_W_RS)
```

```
P(w|¬r,s) =  0.9
[[[0.95 0.9 ]
  [0.9  0.1 ]]

 [[0.05 0.1 ]
  [0.1  0.9 ]]]
```



P_W_RS[:, :, :]

The order (t,f) is defined according to the indexes

From Colab Lab5

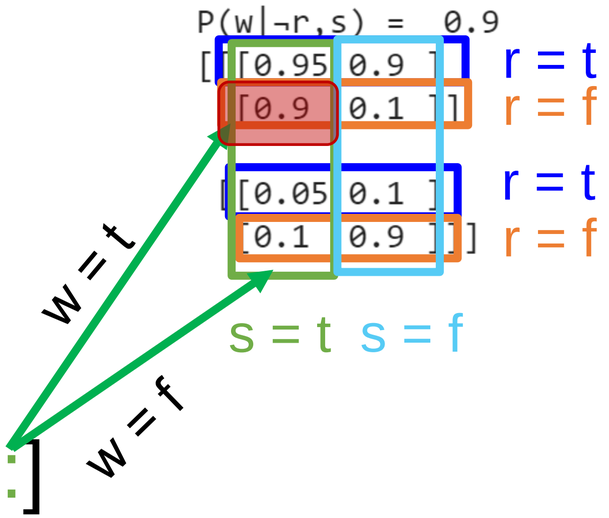
Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

```
# 'true' and 'false' indexes
t, f = 0, 1
```

$$\mathbf{P}(W|R,S):$$

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
print(P_W_RS)
```



P_W_RS[:, :, :]

```
P_W_RS = np.array([[[0.95, 0.90],[0.90, 0.10]],[[0.05, 0.10], [0.10, 0.90]]])
# this is a 2x2x2 matrix, the elements of which can be accessed as follows
print('P(w|¬r,s) = ', P_W_RS[t,f,t])
```

From Colab Lab5

Question 4: Lab5

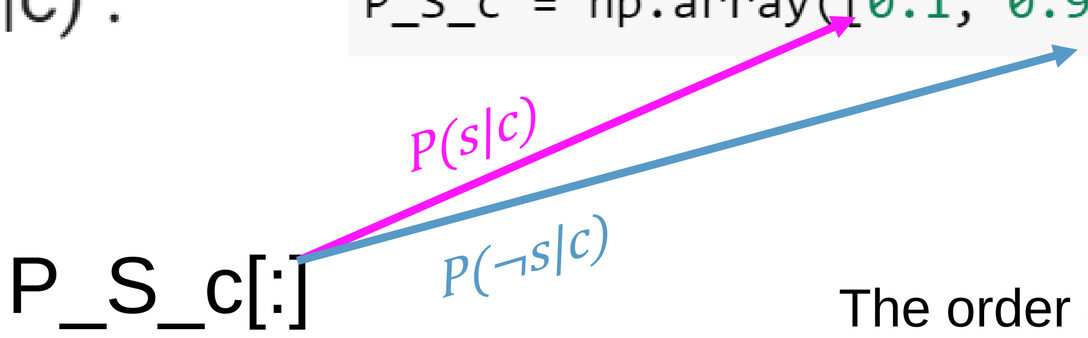
Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

```
# 'true' and 'false' indexes
t, f = 0, 1
```

Here it is a bit different than before.
We assume given c → c = t as the text required, so we exclude c=f)
We compute only the following for simplicity:

P(S|c) :

```
P_S_c = np.array([0.1, 0.9])
```



The order (t,f) is defined according to the indexes

From Colab Lab5

Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

```
# 'true' and 'false' indexes
t, f = 0, 1
```

The extended form is the following:

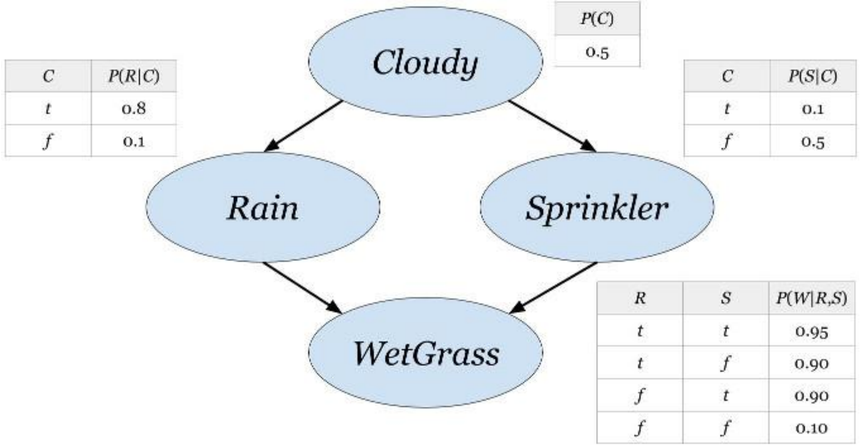
```
P_S_c = np.array([[0.1, 0.5], [0.9, 0.5]])
print(P_S_c)
print(P_S_c[:,t])
```

```
[[0.1 0.5]
 [0.9 0.5]]
[0.1 0.9]
```

s=t

s=f

P_S_c[:,:]



C	P(R|C)
t	0.8
f	0.1

P(C)
0.5

C	P(S|C)
t	0.1
f	0.5

R	S	P(W|R,S)
t	t	0.95
t	f	0.90
f	t	0.90
f	f	0.10

The order (t,f) is defined according to the indexes

From Colab Lab5

Question 4: Lab5

Perchè P(W|R,C) in python è Pwrc[-][-][-] mentre P(S|C) è Psc[-]

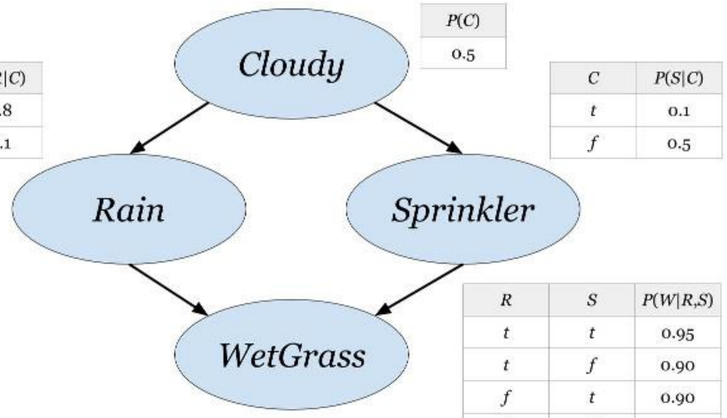
```
# 'true' and 'false' indexes
t, f = 0, 1
```

The extended form is the following:

```
P_S_c = np.array([[0.1, 0.5], [0.9, 0.5]])
print(P_S_c)
print(P_S_c[:,t])
```

```
  c=t   c=f
[[0.1 0.5]
 [0.9 0.5]]
[0.1 0.9]
```

P_S_c[:,:]



C	P(R|C)
t	0.8
f	0.1

P(C)
0.5

C	P(S|C)
t	0.1
f	0.5

R	S	P(W|R,S)
t	t	0.95
t	f	0.90
f	t	0.90
f	f	0.10

The order (t,f) is defined according to the indexes

From Colab Lab5

Question 5: Lab7

POTENTIAL() AND GETPOSTERIOR()

```
def getCuredObservedProba(m1,evs):
    evs0=dict(evs)
    evs1=dict(evs)
    evs0["Drug"]='Without'
    evs1["Drug"]='With'

    return gum.Potential().add(m1.variableFromName("Drug")).fillWith([
            gum.getPosterior(m1,target="Patient",evs=evs0)[1],
            gum.getPosterior(m1,target="Patient",evs=evs1)[1]
        ])

gnb.sideBySide(getCuredObservedProba(m1,{}),
               getCuredObservedProba(m1,{'Gender':'F'}),
               getCuredObservedProba(m1,{'Gender':'M'}),
               captions=["$P(Patient = Healed \mid Drug )$<br/>Taking $Drug$ is observed as efficient to cure",
                         "$P(Patient = Healed \mid Gender=F,Drug)$<br/>except if the $gender$ of the patient is female",
                         "$P(Patient = Healed \mid Gender=M,Drug)$<br/>... or male."])
```

http://webia.lip6.fr/~phw/aGrUM/docs/last/notebooks/potentials.ipynb.html

Potentials represent multi-dimensional arrays with random variables attached to each dimension

Drug	
Without	With
0.5000	0.5750

$P(Patient = Healed \mid Drug)$
Taking $Drug$ is observed as efficient to cure

Drug	
Without	With
0.8000	0.7000

$P(Patient = Healed \mid Gender=F,Drug)$
except if the $gender$ of the patient is female

Drug	
Without	With
0.4000	0.2000

$P(Patient = Healed \mid Gender=M,Drug)$
... or male.

A Potential function is a function that associates a non-negative value (or probability) with each possible assignment of values to a set of random variables. Potential functions are used to represent the local relationships between random variables in a graphical model. Specifically, a potential function is associated with each factor node in the graph, which typically corresponds to a set of random variables in the model.

Question 5: Lab7

POTENTIAL() AND GETPOSTERIOR()

pyAgrum.getPosterior(*model, *, target, evs=None*)

Compute the posterior of a single target (variable) in a BN given evidence

getPosterior uses a VariableElimination inference. If more than one target is needed with the same set of evidence or if the same target is needed with more than one set of evidence, this function is not relevant since it creates a new inference engine every time it is called.

Parameters
- **bn** (*pyAgrum.BayesNet or pyAgrum.MarkovRandomField*) – The probabilistic Graphical Model
- **target** (*string or int*) – variable name or id (forced keyword argument)
- **evs** (*Dict[name|id:val, name|id : List[val1, val2], ...]. (optional forced keyword argument)*) – the (hard and soft) evidence

Returns posterior (pyAgrum.Potential or other)

https://pyagrum.readthedocs.io/en/latest/functions.html

```
def getCuredObservedProba(m1,evs):
    evs0=dict(evs)
    evs1=dict(evs)
    evs0["Drug"]='Without'
    evs1["Drug"]='With'

    return gum.Potential().add(m1.variableFromName("Drug")).fillWith([
            gum.getPosterior(m1,target="Patient",evs=evs0)[1],
            gum.getPosterior(m1,target="Patient",evs=evs1)[1]
        ])

gnb.sideBySide(getCuredObservedProba(m1,{}),
               getCuredObservedProba(m1,{'Gender':'F'}),
               getCuredObservedProba(m1,{'Gender':'M'}),
               captions=["$P(Patient = Healed \mid Drug )$<br/>Taking $Drug$ is observed as efficient to cure",
                         "$P(Patient = Healed \mid Gender=F,Drug)$<br/>except if the $gender$ of the patient is female",
                         "$P(Patient = Healed \mid Gender=M,Drug)$<br/>... or male."])
```

Drug	
Without	With
0.5000	0.5750

$P(Patient = Healed \mid Drug)$
Taking $Drug$ is observed as efficient to cure

Drug	
Without	With
0.8000	0.7000

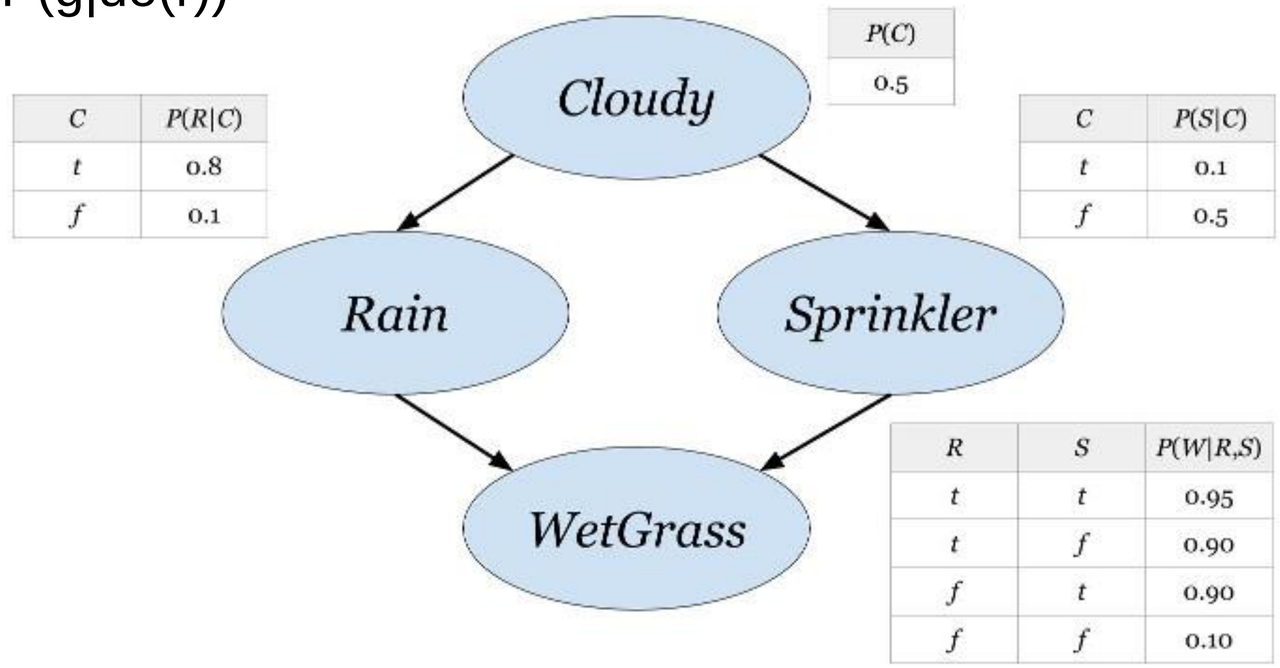
$P(Patient = Healed \mid Gender=F,Drug)$
except if the $gender$ of the patient is female

Drug	
Without	With
0.4000	0.2000

$P(Patient = Healed \mid Gender=M,Drug)$
... or male.

Question 6: Lab7

non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))



$P(C)$
0.5

C	$P(R\|C)$
t	0.8
f	0.1

C	$P(S\|C)$
t	0.1
f	0.5

R	S	$P(W\|R,S)$
t	t	0.95
t	f	0.90
f	t	0.90
f	f	0.10

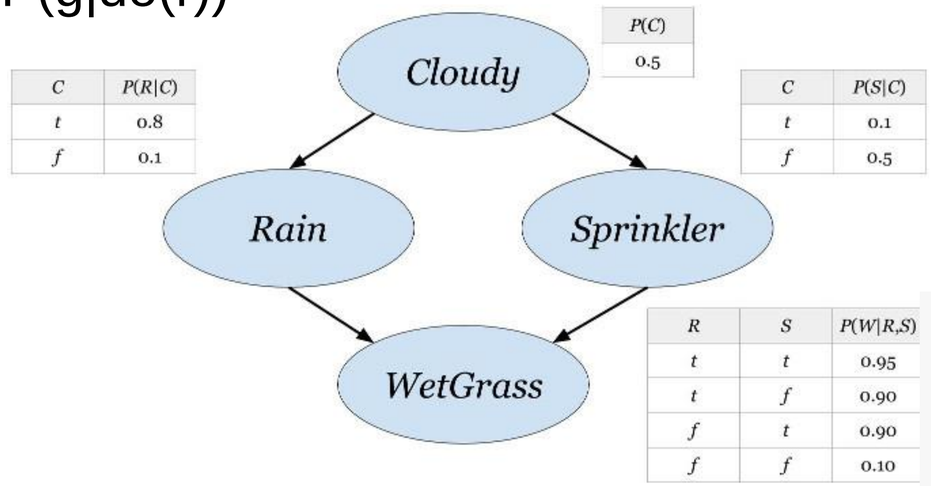
```
!pip install pyAgrum
from IPython.display import display, Math, Latex

import pyAgrum as gum
import pyAgrum.lib.notebook as gnb
import pyAgrum.causal as csl
import pyAgrum.causal.notebook as cslnb
```

Let's see the implementation of the Sprinkler example via PyAgrum

Question 6: Lab7

non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))



P(C)
0.5

C	P(R|C)
t	0.8
f	0.1

C	P(S|C)
t	0.1
f	0.5

R	S	P(W|R,S)
t	t	0.95
t	f	0.90
f	t	0.90
f	f	0.10

Let's see the implementation of the Sprinkler example via PyAgrum

```
# define the network
m1 = gum.fastBN("Cloudy{T|F}->Rain{T|F}->GrassWet{T|F}<-Sprinkler{T|F}<-Cloudy{T|F}")
m1.cpt("Cloudy")[:]=[0.5,0.5]
m1.cpt("Rain")[:]=[[0.8,0.2],  #Cloudy=T
                   [0.1,0.9]]  #Cloudy=F
m1.cpt("Sprinkler")[:]=[[0.1,0.9],  #Cloudy=T
                   [0.5,0.5]]  #Cloudy=F

m1.cpt("GrassWet")[{'Sprinkler':'T','Rain':'T'}]=[0.95,0.05]
m1.cpt("GrassWet")[{'Sprinkler':'T','Rain':'F'}]=[0.90,0.10]
m1.cpt("GrassWet")[{'Sprinkler':'F','Rain':'T'}]=[0.90,0.10]
m1.cpt("GrassWet")[{'Sprinkler':'F','Rain':'F'}]=[0.10,0.90]

gnb.flow.row(m1,m1.cpt("Cloudy"),m1.cpt("Rain"),m1.cpt("Sprinkler"), m1.cpt("GrassWet"))
```

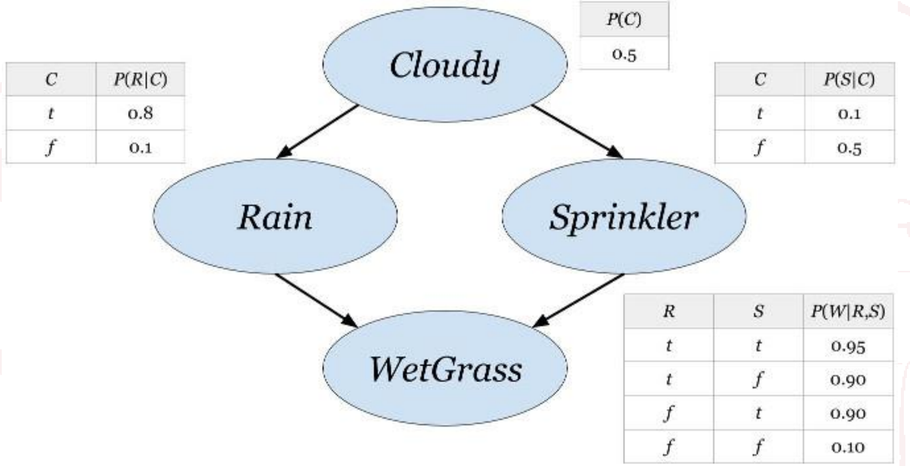
Question 6: Lab7

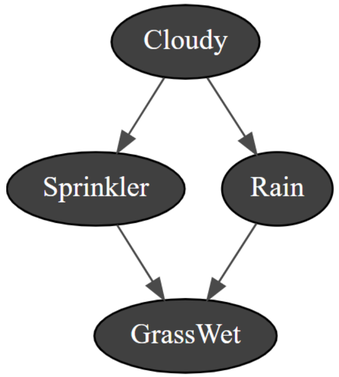
non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))

```
# define the network
m1 = gum.fastBN("Cloudy{T|F}->Rain{T|F}->GrassWet{T|F}<-Sprinkler{T|F}<-Cloudy{T|F}")
m1.cpt("Cloudy")[:]=[0.5,0.5]
m1.cpt("Rain")[:]=[[0.8,0.2],  #Cloudy=T
                   [0.1,0.9]]  #Cloudy=F
m1.cpt("Sprinkler")[:]=[[0.1,0.9],  #Cloudy=T
                   [0.5,0.5]]  #Cloudy=F

m1.cpt("GrassWet")[{'Sprinkler':'T','Rain':'T'}]=[0.95,0.05]
m1.cpt("GrassWet")[{'Sprinkler':'T','Rain':'F'}]=[0.90,0.10]
m1.cpt("GrassWet")[{'Sprinkler':'F','Rain':'T'}]=[0.90,0.10]
m1.cpt("GrassWet")[{'Sprinkler':'F','Rain':'F'}]=[0.10,0.90]

gnb.flow.row(m1,m1.cpt("Cloudy"),m1.cpt("Rain"),m1.cpt("Sprinkler"), m1.cpt("GrassWet"))
```





Cloudy	
T	F
0.5000	0.5000

	Rain	
Cloudy	T	F
T	0.8000	0.2000
F	0.1000	0.9000

	Sprinkler	
Cloudy	T	F
T	0.1000	0.9000
F	0.5000	0.5000

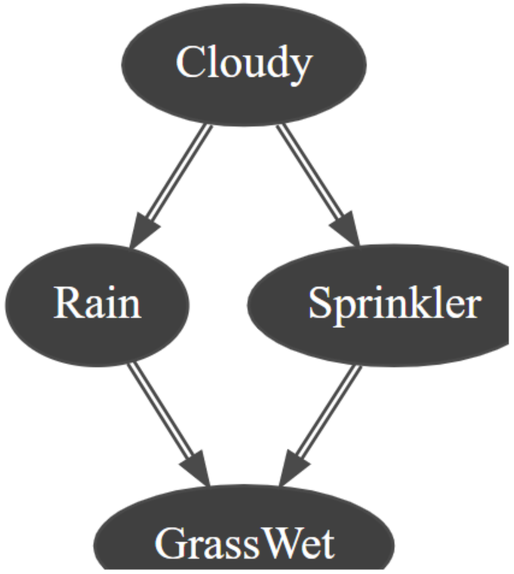
		GrassWet	
Sprinkler	Rain	T	F
T	T	0.9500	0.0500
	F	0.9000	0.1000
F	T	0.9000	0.1000
	F	0.1000	0.9000

Question 6: Lab7

non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))

Let's compute $P(g|do(r))$

```
# let's compute P(g | do(r))
d1 = csl.CausalModel(m1)
cslnb.showCausalImpact(d1, "GrassWet", doing="Rain",values={"Rain" : "T"})
prob= cslnb.getCausalImpact(d1, "GrassWet", doing="Rain",values={"Rain" : "T"})
print(prob)
```



Causal Model

Question 6: Lab7

non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))

Let's compute $P(g|do(r))$

```
# let's compute P(g | do(r))
d1 = csl.CausalModel(m1)
cslnb.showCausalImpact(d1, "GrassWet", doing="Rain",values={"Rain" : "T"})
prob= cslnb.getCausalImpact(d1, "GrassWet", doing="Rain",values={"Rain" : "T"})
print(prob)
```

pyAgrum.causal.notebook.getCausalImpact(*model, on, doing, knowing=None, values=None*)

return a HTML representing of the three values defining a causal impact : formula, value, explanation

Parameters
- **model** (*CausalModel*) – the causal model
- **on** (*str | Set[str]*) – the impacted variable(s)
- **doing** (*str | Set[str]*) – the interventions
- **knowing** (*str | Set[str]*) – the observations
- **values** (*Dict[str,int] default=None*) – value for certain variables

Return type HTML

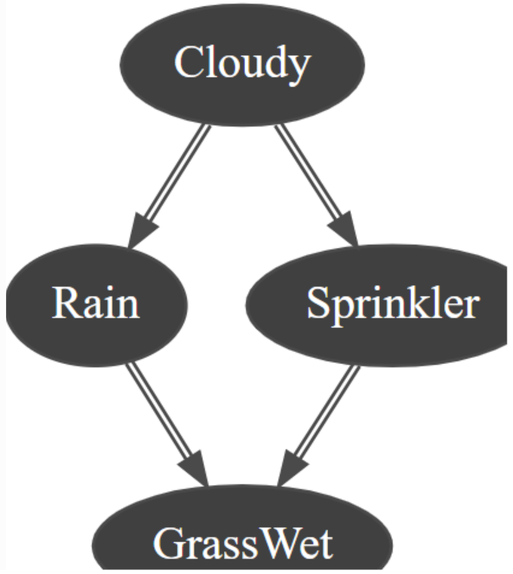
pyAgrum.causal.notebook.getCausalModel(*cm, size=None*)

return a HTML representing the causal model

Parameters
- **cm** (*CausalModel*) – the causal model
- **size** (*int|str*) – the size of the rendered graph

Returns the dot representation

Return type pydot.Dot



Causal Model

https://pyagrum.readthedocs.io/en/latest/CausalNotebook.html

Question 6: Lab7

non capisco come ottenere dei valori numerici utilizzando pyAgrum, ad esempio riesco ad ottenere correttamente la tabella di P(G|do(r)) ma non saprei come ottenere direttamente su python il valore P(g|do(r))

To extract data from <IPython.core.display.HTML object>, we can use **pandas**

```
from IPython.core.display import HTML
import pandas as pd

# assume html_object is an HTML string or an IPython.core.display.HTML object
if isinstance(prob, HTML):
    html_object = prob.data

# read the HTML data into a Pandas dataframe
df_list = pd.read_html(html_object)

# assume the first dataframe in the list contains the data you're interested in
df = df_list[0]
print(df)

# Exract the target value
print(df.loc[0][0])
```

```
  GrassWet
         T      F
0    0.915  0.085
0.915
```

GrassWet	
T	F
0.9150	0.0850

Questions