@TAASP2023

UNIVERSITÀ DELLA CALABRIA
il Campus per eccellenza

I 💘 ASP, BUT…

Mario Alviano

Future Artificial Intelligence Research

PRODE
Probabilistic declarative process mining

CYBER CHALLENGE.IT

SERICS
SECURITY AND RIGHTS IN THE CYBERSPACE

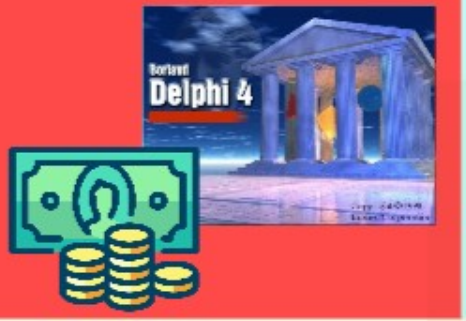
CYBER XCELERATOR
L'Acceleratore Cybertech della Rete Nazionale CDP

LAIA intelligenza artificiale
SISTEMA INTEGRATO DI LABORATORI PER L'AMBIENTE
SILA

1983

1990

1999

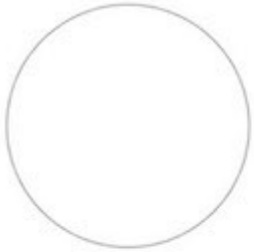


2002

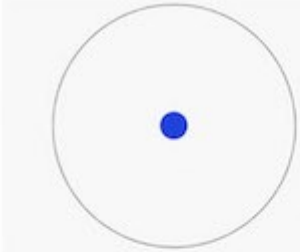
UNIVERSITÀ
DELLA
CALABRIA

il Campus per eccellenza

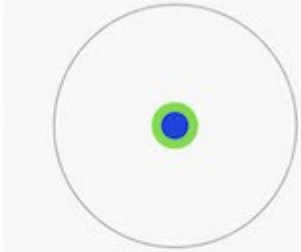
Imagine a circle that contains all of human knowledge:



By the time you finish elementary school, you know a little:



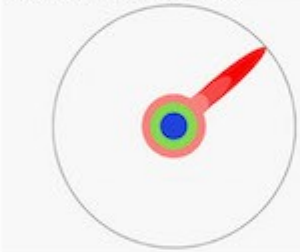
By the time you finish high school, you know a bit more:



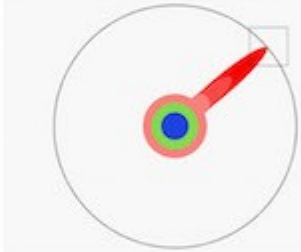
With a bachelor's degree, you gain a specialty:

A master's degree deepens that specialty:

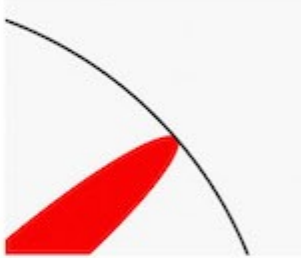
Reading research papers takes you to the edge of human knowledge:



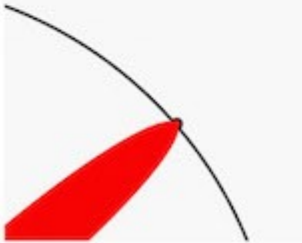
Once you're at the boundary, you focus:



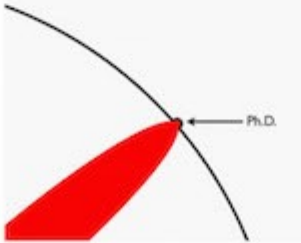
You push at the boundary for a few years:



Until one day, the boundary gives way:



And, that dent you've made is called a Ph.D.:



Of course, the world looks different to you now:

So, don't forget the bigger picture:

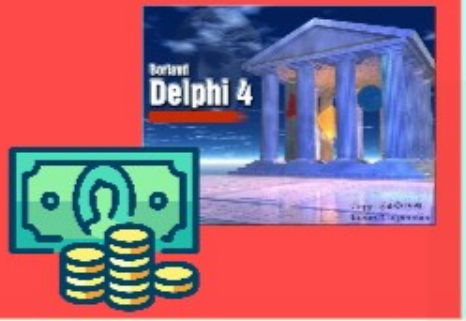
Keep pushing.

https://mymodernmet.com/phd-infographic-matt-might/

1983

1990

1999



2002

UNIVERSITÀ DELLA CALABRIA

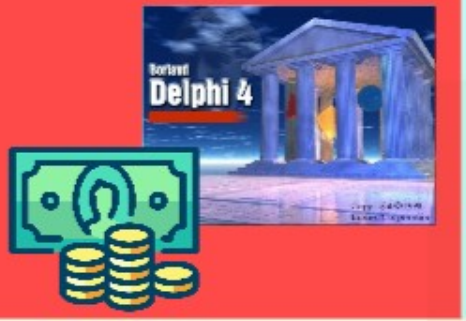
il Campus per eccellenza

2016

1983

1990

1999



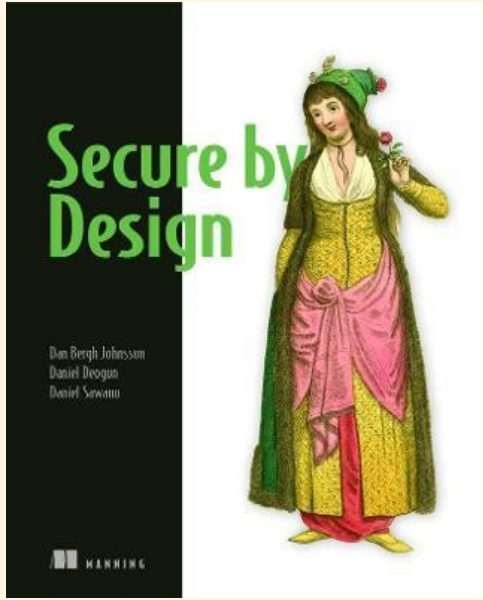
2002

UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

2016

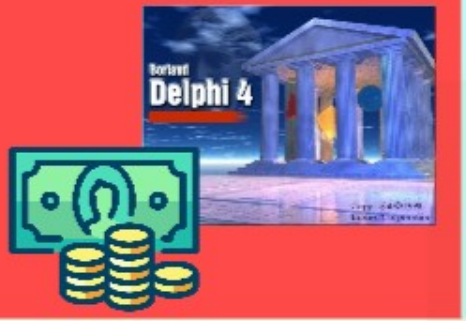
2019



1983

1990

1999



2002

UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

2016

2019

I learned **HOW** to code in 2019

(if I ever learned)

In the classroom **I teach secure coding**,
concealing my true identity

In my office I code in my beloved language,
<u>insecure</u> but fearless

I live two parallel realities

I live two parallel realities

Am I crazy? Or...

VALIDATE DATA IN CONSTRUCTION AND DON'T REPEAT YOURSELF

WE DON'T DO THAT HERE

Input often from other software, it's reliable (is it?)

Efficiency, efficiency, efficiency... don't add overhead (do we?)

ASP programs are short and understandable (are they?)

We usually have correctness proofs (really?)

ASP programs are always different (are they?)

Input often from other software, it’s reliable (is it?)

Input often from other software, it’s reliable (is it?)

Same assumption in the other software?

Trust boundaries?

Input often from other software, it's reliable (is it?)

Same assumption in the other software?

Is the encoding used somewhere else?

Efficiency, efficiency, efficiency...
don’t add overhead (do we?)

Who said overhead?

	Video Streaming	Solitaire
CLINGO grouding via Python interface, no answer set search	0.06 seconds	0.07 seconds
VALASP validation (includes grounding via CLINGO Python interface), no answer set search	0.18 seconds	0.13 seconds

Efficiency, efficiency, efficiency...
don't add overhead (do we?)

Who said overhead?

	Video Streaming	Solitaire
CLINGO grouding via Python interface, no answer set search	0.06 seconds	0.07 seconds
VALASP validation (includes grounding via CLINGO Python interface), no answer set search	0.18 seconds	0.13 seconds

Efficiency, efficiency, efficiency...
don't add overhead (do we?)

ASP programs are short and understandable (are they?)

Cause or effect?

Checco Zalone in “Cado dalle nubi”

ASP programs are short and understandable (are they?)

Cause or effect?

Checco Zalone in “Cado dalle nubi”

Are we short because we can’t be long?

For every rule we add, we have to (re)check all others

ASP programs are short and understandable (are they?)

Cause or effect?

Checco Zalone in “Cado dalle nubi”

Are we short because we can’t be long?

For every rule we add, we have to (re)check all others

ASP programs are short and understandable (are they?)

We usually have correctness proofs (really?)

Cause or effect?

Checco Zalone in “Cado dalle nubi”

Are we short because we can’t be long?

For every rule we add, we have to (re)check all others

ASP programs are short and understandable (are they?)

We usually have correctness proofs (really?)

Understandable but we need proofs?!?

ASP programs are always different (are they?)

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

ASP programs are always different (are they?)

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

ASP programs are always different (are they?)

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Ensure that a guessed tree encoded by node/1 and tree/2 is not a forest.

ASP programs are always different (are they?)

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Ensure that a guessed tree encoded by node/1 and tree/2 is not a forest.

```
reach(X) :- X = #min{Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

ASP programs are always different (are they?)

Code reusability?

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Ensure that a guessed tree encoded by node/1 and tree/2 is not a forest.

```
reach(X) :- X = #min{Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

ASP programs are always different (are they?)

We are ugly!

We are ugly!

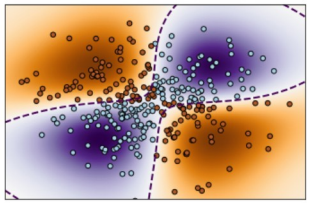
localhost:8889/notebooks/test01/jupyter_test.ipynb#

jupyter sklearn_test

File Edit View Insert Cell Kernel

Run

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

xx, yy = np.meshgrid(np.linspace(-3, 3, 500),
                     np.linspace(-3, 3, 500))

                    ⋮

plt.axis([-3, 3, -3, 3])
plt.show()
```

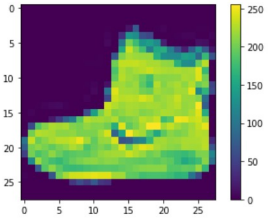


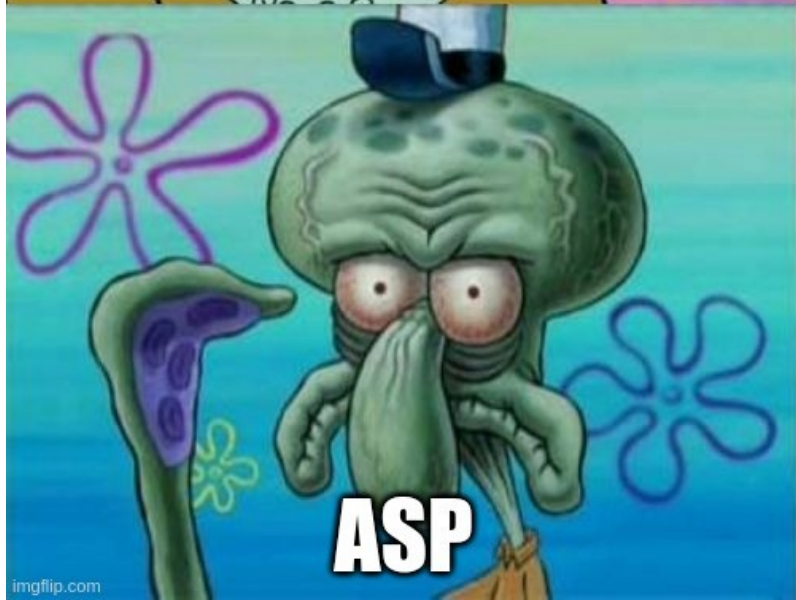
jupyter tensorflow_test

File Edit View Insert Cell Kernel

Run

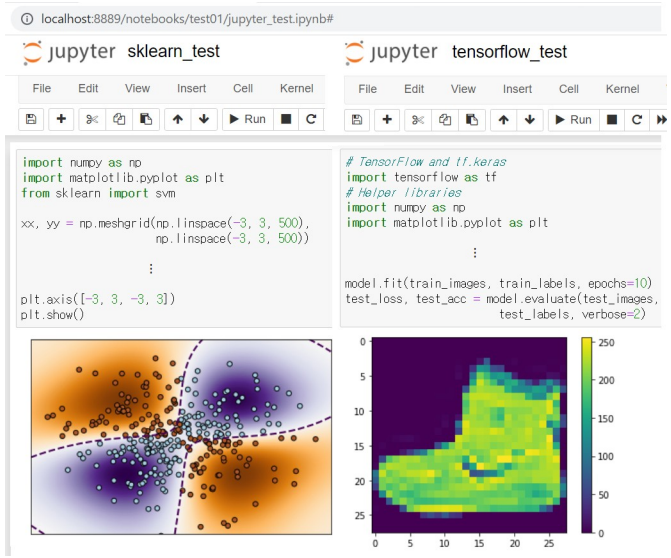
```
# TensorFlow and tf.keras
import tensorflow as tf
# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

                    ⋮

model.fit(train_images, train_labels, epochs=10)
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels, verbose=2)
```



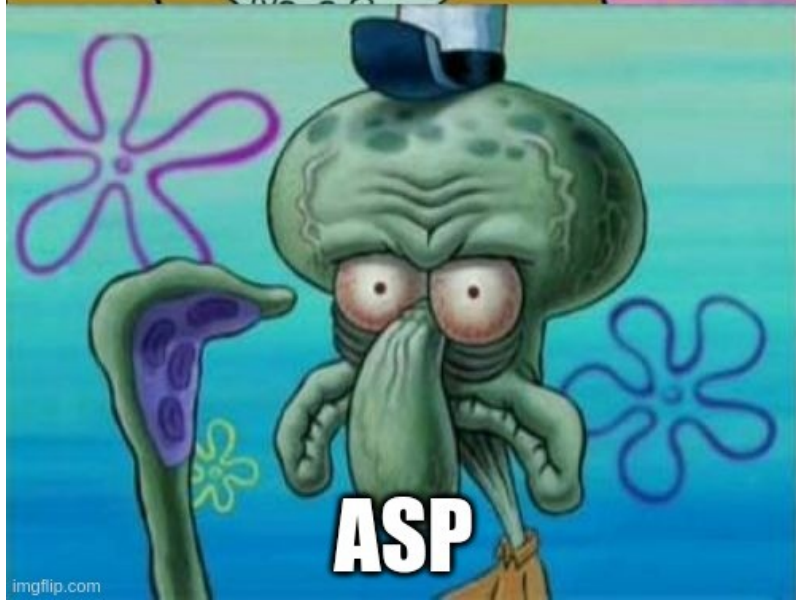


We are ugly!

This is how we are seen




```
$ clingo sudoku.asp
clingo version 5.4.0
Reading from sudoku.asp
Solving...
Answer: 1
assign((1,1),6) assign((1,3),9) assign((1,4),8) assign((1,6),7) assign((2,4),6)
assign((2,9),1) assign((3,2),3) assign((3,3),5) assign((3,6),2) assign((3,8),7)
assign((4,2),6) assign((4,3),8) assign((4,7),1) assign((4,9),2) assign((5,1),3)
assign((5,6),5) assign((6,4),2) assign((6,7),3) assign((6,8),6) assign((7,1),8)
assign((7,2),5) assign((7,3),4) assign((7,4),7) assign((7,5),2) assign((7,7),6)
assign((7,8),9) assign((8,4),5) assign((8,5),9) assign((8,9),8) assign((9,1),2)
assign((9,3),6) assign((9,4),4) assign((9,5),3) assign((9,7),7) assign((9,8),1)
assign((9,9),5) assign((1,2),2) assign((2,1),4) assign((2,2),8) assign((2,3),7)
assign((3,1),1) assign((1,5),1) assign((2,5),5) assign((2,6),3) assign((3,4),9)
assign((3,5),4) assign((1,8),3) assign((1,7),5) assign((1,9),4) assign((2,7),9)
assign((2,8),2) assign((3,7),8) assign((3,9),6) assign((4,1),9) assign((5,2),7)
assign((5,3),2) assign((6,1),5) assign((6,2),4) assign((6,3),1) assign((4,4),3)
assign((4,5),7) assign((4,6),4) assign((5,4),1) assign((5,5),6) assign((6,5),8)
assign((6,6),9) assign((4,8),5) assign((5,7),4) assign((5,8),8) assign((5,9),9)
assign((6,9),7) assign((8,1),7) assign((8,2),1) assign((8,3),3) assign((9,2),9)
assign((7,6),1) assign((8,6),6) assign((9,6),8) assign((7,9),3) assign((8,7),2)
assign((8,8),4)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.012s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.012s
```



New project scaffolding

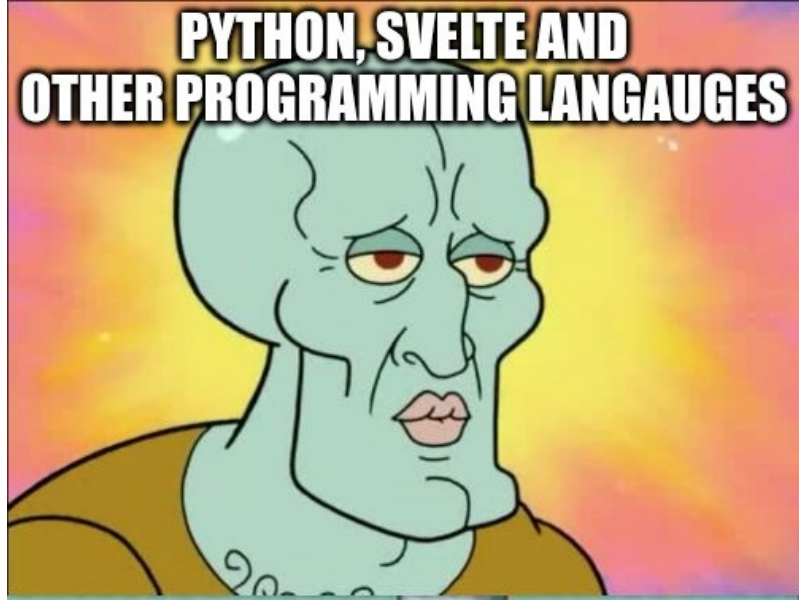
(we are ugly!)

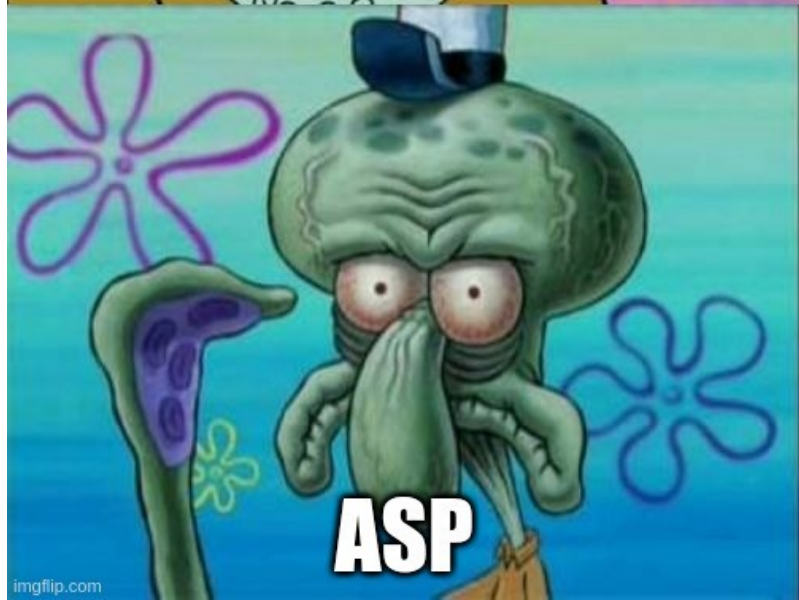
```
poetry new poetry-demo
```

then
poetry add this
poetry add that

npm add this
npm add that

```
npm create svelte@latest my-app
cd my-app
npm install
npm run dev
```





New project scaffolding

(we are ugly!)

```
poetry new poetry-demo
```

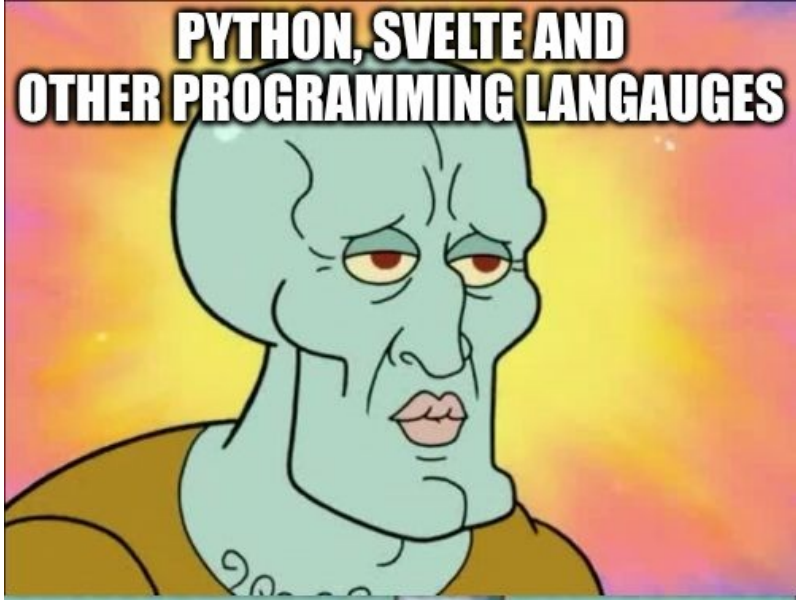
then
poetry add this
poetry add that

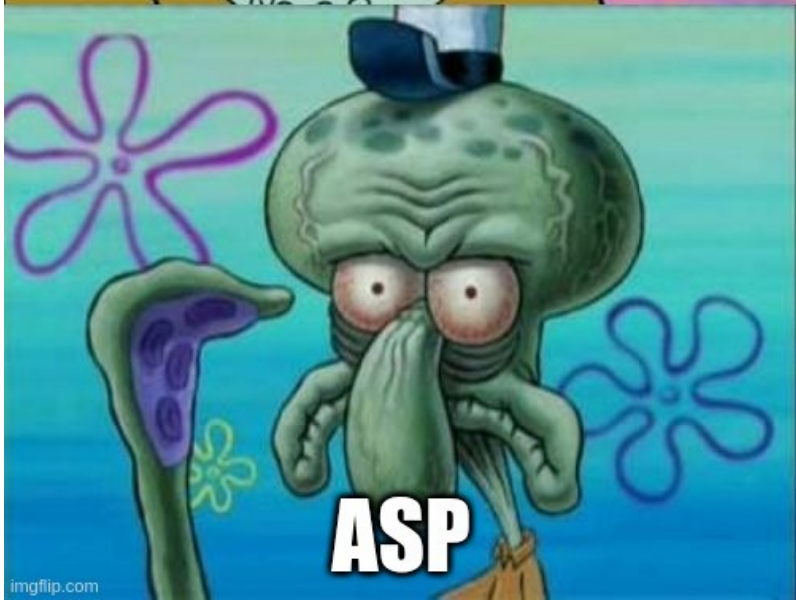
npm add this
npm add that

```
npm create svelte@latest my-app
cd my-app
npm install
npm run dev
```

```
$ vi sudoku.asp
```

and that's it!





I’m not the first to complain

Types, macros, templates…
many attempt to improve the language

I’m not the first to complain

Types, macros, templates…
many attempt to improve the language

Why didn’t they succeed in ASP?

I’m not the first to complain

Types, macros, templates…
many attempt to improve the language

Why didn’t they succeed in ASP?

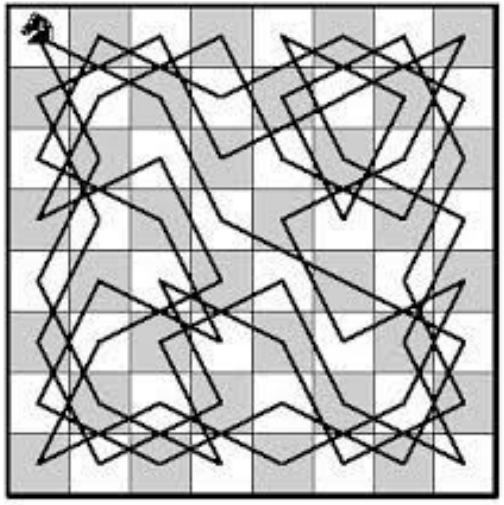
I'm not the first to complain

Types, macros, templates…
many attempt to improve the language

Why didn't they succeed in ASP?

Why should it be different now?

Let’s start with data validation



mat.unical.it/aspcomp20

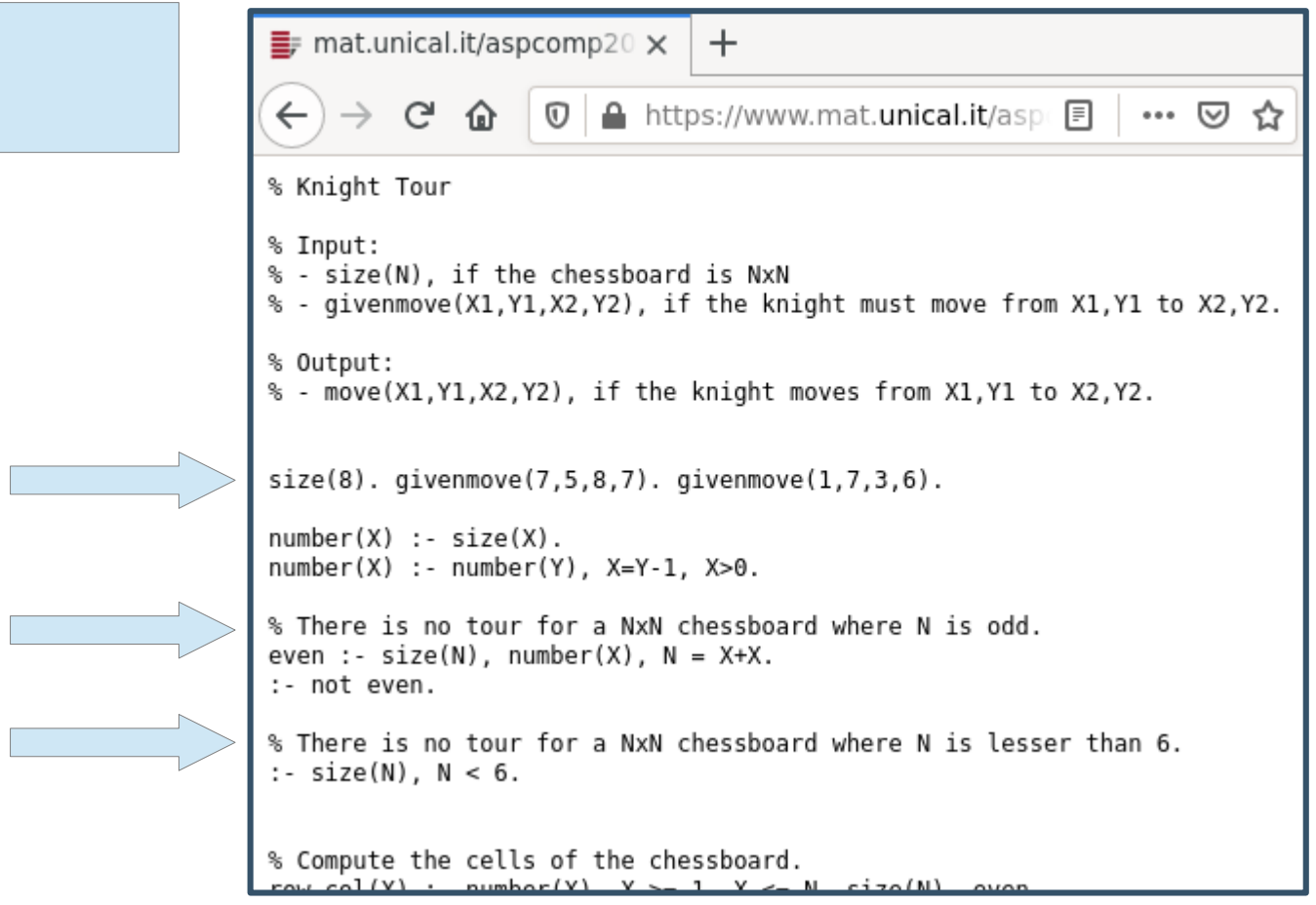
https://www.mat.unical.it/asp

```
% Knight Tour

% Input:
% - size(N), if the chessboard is NxN
% - givenmove(X1,Y1,X2,Y2), if the knight must move from X1,Y1 to X2,Y2.

% Output:
% - move(X1,Y1,X2,Y2), if the knight moves from X1,Y1 to X2,Y2.


size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).

number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
row_col(X) :- number(X), X >= 1, X <= N, size(N), even
```

https://www.mat.unical.it/aspcomp2011/files/KnightTour/knight_tour.enc.asp

SCAN ME

Wait, wait...


mat.unical.it/aspcomp20
https://www.mat.unical.it/asp
% Knight Tour
% Input:
% - size(N), if the chessboard is NxN
% - givenmove(X1,Y1,X2,Y2), if the knight must move from X1,Y1 to X2,Y2.
% Output:
% - move(X1,Y1,X2,Y2), if the knight moves from X1,Y1 to X2,Y2.
size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).
number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.
% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.
% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.
% Compute the cells of the chessboard.

Wait, wait...

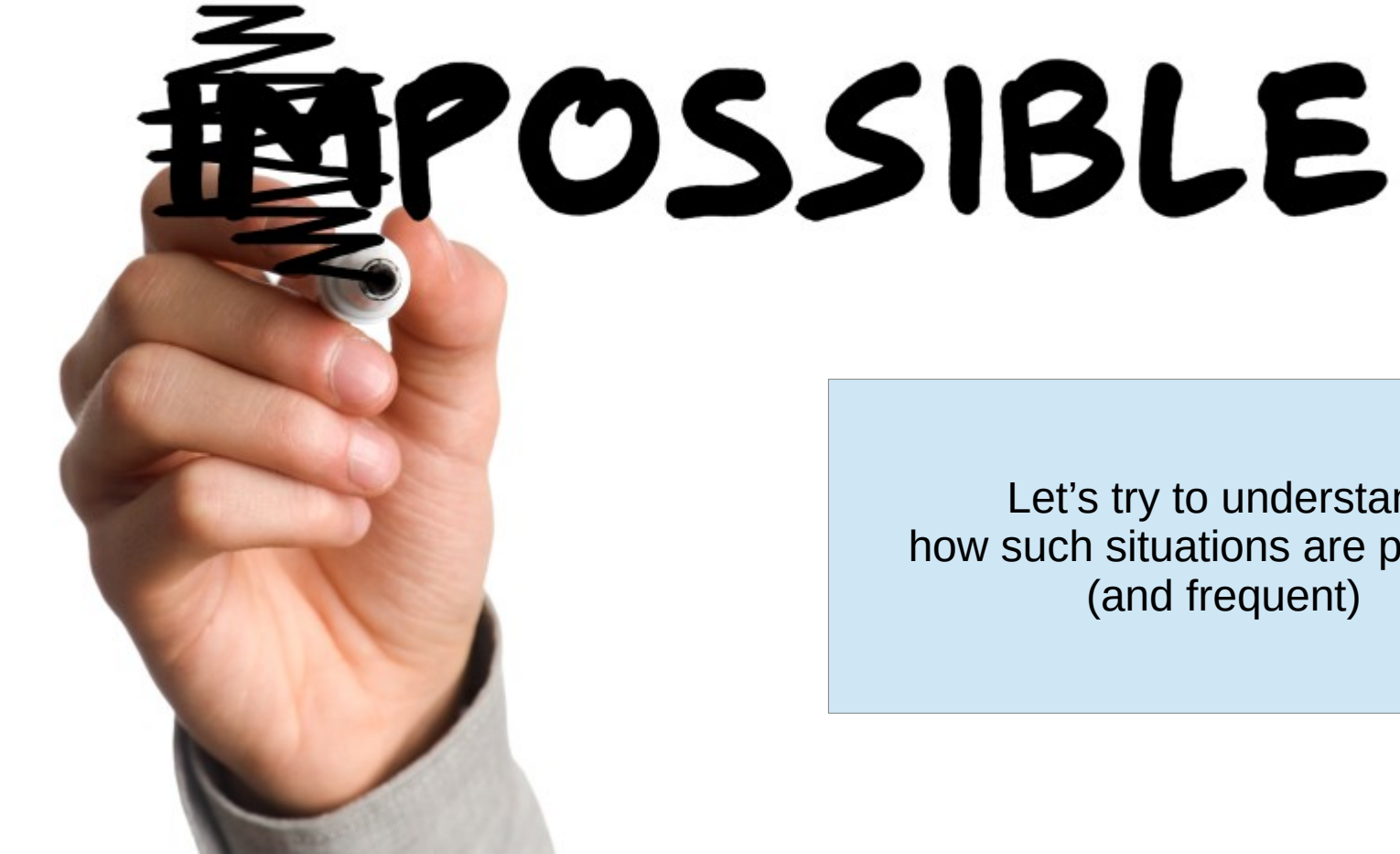
mat.unical.it/aspcomp20

https://www.mat.unical.it/asp

```
% Knight Tour

% Input:
% - size(N), if the chessboard is NxN
% - givenmove(X1,Y1,X2,Y2), if the knight must move from X1,Y1 to X2,Y2.

% Output:
% - move(X1,Y1,X2,Y2), if the knight moves from X1,Y1 to X2,Y2.


size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).

number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
```

size/1 and givenmove/4 had to be input predicates

So, making mistakes with declarative programming is



Let’s try to understand
how such situations are possible
(and frequent)

Open | knight_tour.enc.asp ~/tmp | Save

```
number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

even :- size(N), number(X), N = X+X.
:- not even.

:- size(N), N < 6.


row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
cell(X,Y) :- row_col(X), row_col(Y).


move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).

move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).


valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.


:- cell(X,Y), not exactlyOneMoveEntering(X,Y).
exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.

:- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.


reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
:- cell(X,Y), not reached(X,Y).
```

Dev writes an encoding using a text editor

Open | knight_tour.enc.asp ~/tmp | Save

```
1 number(X) :- size(X).
2 number(X) :- number(Y), X=Y-1, X>0.
3
4 even :- size(N), number(X), N = X+X.
5 :- not even.
6
7 :- size(N), N < 6.
8
9
10 row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
11 cell(X,Y) :- row_col(X), row_col(Y).
12
13
14 move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).
15
16 move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).
17
18
19 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
20 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
21 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
22 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
23 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
24 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
25 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
26 valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.
27
28
29 :- cell(X,Y), not exactlyOneMoveEntering(X,Y).
30 exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
31 atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
32 atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.
33
34 :- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
35 exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
36 atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
37 atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.
38
39
40 reached(X,Y) :- move(1,1,X,Y).
41 reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
42 :- cell(X,Y), not reached(X,Y).
```

Dev writes an encoding using a text editor

Dev knows that long complicated code must be commented

Open | *knight_tour.enc.asp ~/tmp | Save

```
number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
cell(X,Y) :- row_col(X), row_col(Y).


% Given moves must be done.
move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).

% Guess the other moves.
move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).


% Compute the valid moves from each cell.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.


% Exactly one move entering to each cell.
:- cell(X,Y), not exactlyOneMoveEntering(X,Y).
exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.

% Exactly one move leaving each cell.
:- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.


% Each cell must be reached by the knight.
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
:- cell(X,Y), not reached(X,Y).
```

Dev writes an encoding using a text editor

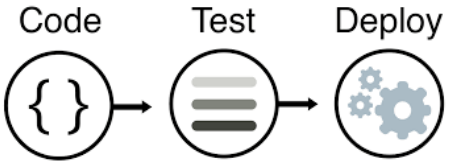
Dev knows that long complicated code must be commented

Open | *knight_tour.enc.asp ~/tmp | Save

```
number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
cell(X,Y) :- row_col(X), row_col(Y).


% Given moves must be done.
move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).

% Guess the other moves.
move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).


% Compute the valid moves from each cell.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.


% Exactly one move entering to each cell.
:- cell(X,Y), not exactlyOneMoveEntering(X,Y).
exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.

% Exactly one move leaving each cell.
:- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.


% Each cell must be reached by the knight.
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
:- cell(X,Y), not reached(X,Y).
```

Dev writes an encoding using a text editor

Dev knows that long complicated code must be commented



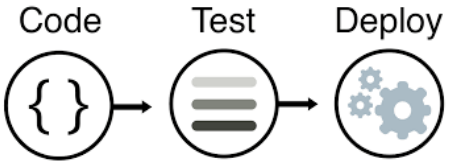
Dev tests the encoding in the laziest way, by adding input to the encoding

Open | *knight_tour.enc.asp ~/tmp | Save

```
size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).

number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
cell(X,Y) :- row_col(X), row_col(Y).


% Given moves must be done.
move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).

% Guess the other moves.
move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).


% Compute the valid moves from each cell.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.


% Exactly one move entering to each cell.
:- cell(X,Y), not exactlyOneMoveEntering(X,Y).
exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.

% Exactly one move leaving each cell.
:- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.


% Each cell must be reached by the knight.
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
:- cell(X,Y), not reached(X,Y).
```

Dev writes an encoding using a text editor

Dev knows that long complicated code must be commented



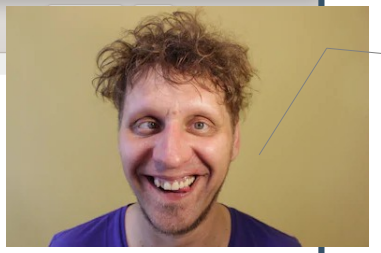
Dev tests the encoding in the laziest way, by adding input to the encoding

Open

*knight_tour.enc.asp

~/tmp

```
size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).

number(X) :- size(X).
number(X) :- number(Y), X=Y-1, X>0.

% There is no tour for a NxN chessboard where N is odd.
even :- size(N), number(X), N = X+X.
:- not even.

% There is no tour for a NxN chessboard where N is lesser than 6.
:- size(N), N < 6.


% Compute the cells of the chessboard.
row_col(X) :- number(X), X >= 1, X <= N, size(N), even.
cell(X,Y) :- row_col(X), row_col(Y).


% Given moves must be done.
move(X1,Y1,X2,Y2) :- givenmove(X1,Y1,X2,Y2).

% Guess the other moves.
move(X1,Y1,X2,Y2) | non_move(X1,Y1,X2,Y2):- valid(X1,Y1,X2,Y2).


% Compute the valid moves from each cell.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y1 = Y2+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+2, Y2 = Y1+1.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X1 = X2+1, Y2 = Y1+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y1 = Y2+2.
valid(X1,Y1,X2,Y2) :- cell(X1,Y1), cell(X2,Y2), X2 = X1+1, Y2 = Y1+2.


% Exactly one move entering to each cell.
:- cell(X,Y), not exactlyOneMoveEntering(X,Y).
exactlyOneMoveEntering(X,Y) :- move(X,Y,X1,Y1), not atLeastTwoMovesEntering(X,Y).
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), X1 != X2.
atLeastTwoMovesEntering(X,Y) :- move(X,Y,X1,Y1), move(X,Y,X2,Y2), Y1 != Y2.

% Exactly one move leaving each cell.
:- cell(X,Y), not exactlyOneMoveLeaving(X,Y).
exactlyOneMoveLeaving(X,Y) :- move(X1,Y1,X,Y), not atLeastTwoMovesLeaving(X,Y).
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), X1 != X2.
atLeastTwoMovesLeaving(X,Y) :- move(X1,Y1,X,Y), move(X2,Y2,X,Y), Y1 != Y2.


% Each cell must be reached by the knight.
reached(X,Y) :- move(1,1,X,Y).
reached(X2,Y2) :- reached(X1,Y1), move(X1,Y1,X2,Y2).
:- cell(X,Y), not reached(X,Y).
```

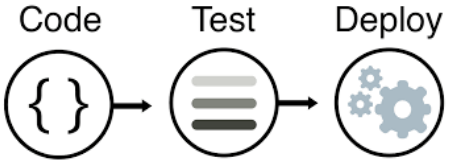


"I will delete it later"

It will stay there forever!

Dev writes an encoding using a text editor

Dev knows that long complicated code must be commented



Dev tests the encoding in the laziest way, by adding input to the encoding

Input and output “documentation”

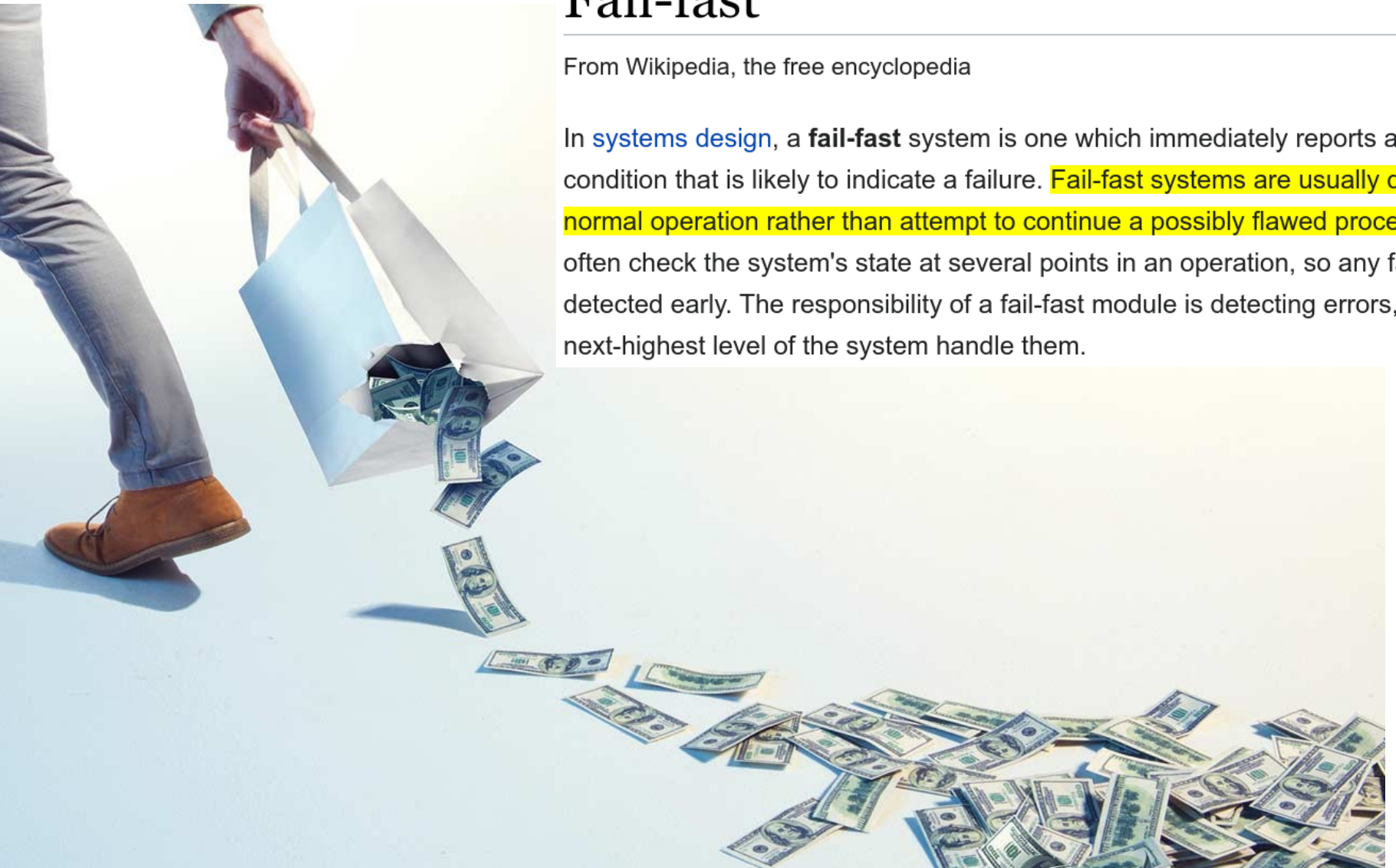
```
1 % Knight Tour
2
3 % Input:
4 % - size(N), if the chessboard is NxN
5 % - givenmove(X1,Y1,X2,Y2), if the knight must move from X1,Y1 to X2,Y2.
6
7 % Output:
8 % - move(X1,Y1,X2,Y2), if the knight moves from X1,Y1 to X2,Y2.
```

Much more than this that is not stated!

- We expect exactly one instance of size(N), with N positive
- Possibly, we may want to enforce N >= 6 and even as an input requirement
- Upper bound for N? Do we expect the encoding to be used for N = 1000?

- In givenmove(X1,Y1,X2,Y2) those (X1,Y1) and (X2,Y2) are cells of a grid
- It’s not stated, it’s not checked! The encoding may work on wrong input data

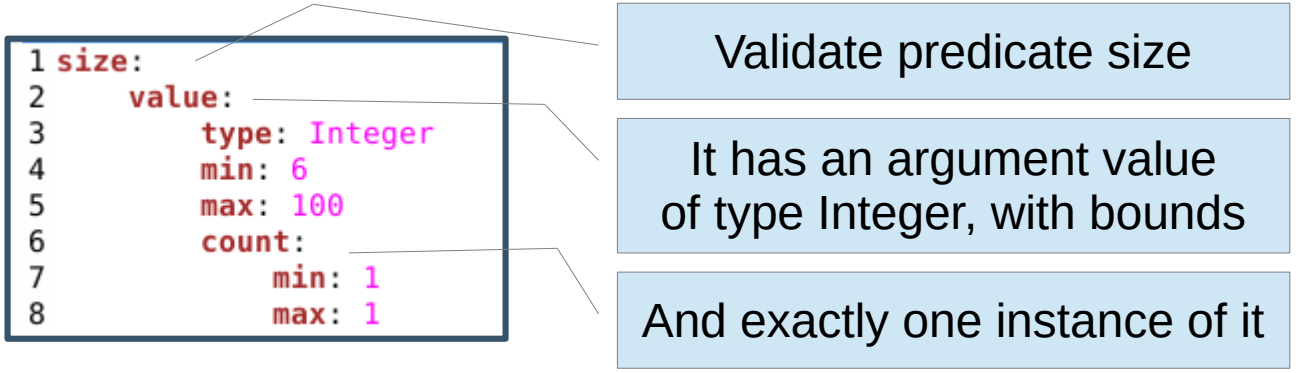
- Similar for move/4



Fail-fast

From Wikipedia, the free encyclopedia

In systems design, a **fail-fast** system is one which immediately reports at its interface any condition that is likely to indicate a failure. Fail-fast systems are usually designed to stop normal operation rather than attempt to continue a possibly flawed process. Such designs often check the system's state at several points in an operation, so any failures can be detected early. The responsibility of a fail-fast module is detecting errors, then letting the next-highest level of the system handle them.


size:
value:
type: Integer
min: 6
max: 100
count:
min: 1
max: 1
Validate predicate size
It has an argument value of type Integer, with bounds
And exactly one instance of it

```yaml
size:
    value:
        type: Integer
        min: 6
        max: 100
        count:
            min: 1
            max: 1
```

Validate predicate size

It has an argument value of type Integer, with bounds

And exactly one instance of it

```
$ cat knight_tour.instance.2.asp
size(5). givenmove(7,5,8,7). givenmove(1,7,3,6).
```

```
$ python -m valasp knight_tour.0.yaml knight_tour.enc-revised.asp knight_tour.instance.2.asp
VALIDATION FAILED
=================
Invalid instance of size:
    in constructor of size
  with error: Should be >= 6, but received 5 in atom/term 5
=================
```

```
$ cat knight_tour.instance.3.asp
size(eight). givenmove(7,5,8,7). givenmove(1,7,3,6).
```

```
$ python -m valasp knight_tour.0.yaml knight_tour.enc-revised.asp knight_tour.instance.3.asp
VALIDATION FAILED
=================
Invalid instance of size:
    in constructor of size
  with error: expecting clingo.SymbolType.Number, but received eight in atom/term eight
=================
```

```yaml
size:
    value:
        type: Integer
        min: 6
        max: 100
        count:
            min: 1
            max: 1
```

```
$ cat knight_tour.instance.1.asp
size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).
size(6).
```

```
$ python -m valasp knight_tour.0.yaml knight_tour.enc-revised.asp knight_tour.instance.1.asp
VALIDATION FAILED
=================
count of value in predicate size may exceed 1
=================
```

```
$ cat knight_tour.instance.0.asp
size(8). givenmove(7,5,8,7). givenmove(1,7,3,6).
```

```
$ python -m valasp knight_tour.0.yaml knight_tour.enc-revised.asp knight_tour.instance.0.asp
ALL VALID!
==========
Answer: size(8) number(8) number(7) number(6) number(5) number(4) number(3) number(2) number(1
,8) cell(7,8) cell(6,8) cell(5,8) cell(4,8) cell(3,8) cell(2,8) cell(1,8) cell(8,7) cell(7,7)
,6) cell(4,6) cell(3,6) cell(2,6) cell(1,6) cell(8,5) cell(7,5) cell(6,5) cell(5,5) cell(4,5)
,4) cell(1,4) cell(8,3) cell(7,3) cell(6,3) cell(5,3) cell(4,3) cell(3,3) cell(2,3) cell(1,3)
```

How does it work

YAML converted to valasp Python code

```
@context.valasp(validate_predicate=True, with_fun=valasp.domain.primitive_types.Fun.FORWARD_IMPLICIT, auto_blacklist=True)
class Size:
        value: Integer
        def __post_init__(self):
                if self.value < 6: raise ValueError(f"Should be >= 6, but received {self.value}")
                if self.value > 100: raise ValueError(f"Should be <= 100, but received {self.value}")
                self.__class__.count_of_value += 1
                if self.value % 2 != 0:
                    raise ValueError('Size must be an even number')
                self.__class__.value = self.value



        @classmethod
        def before_grounding_init_count_value(cls): cls.count_of_value = 0
        @classmethod
        def after_grounding_check_count_value(cls):
                if cls.count_of_value > 1: raise ValueError('count of value in predicate size may exceed 1')
                if cls.count_of_value < 1: raise ValueError('count of value in predicate size cannot reach 1')
```

How does it work

YAML converted to valasp Python code

from here

```
@context.valasp(validate_predicate=True, with_fun=valasp.domain.primitive_types.Fun.FORWARD_IMPLICIT, auto_blacklist=True)
class Size:
        value: Integer
        def __post_init__(self):
                if self.value < 6: raise ValueError(f"Should be >= 6, but received {self.value}")
                if self.value > 100: raise ValueError(f"Should be <= 100, but received {self.value}")
                self.__class__.count_of_value += 1
                if self.value % 2 != 0:
                    raise ValueError('Size must be an even number')
                self.__class__.value = self.value



        @classmethod
        def before_grounding_init_count_value(cls): cls.count_of_value = 0
        @classmethod
        def after_grounding_check_count_value(cls):
                if cls.count_of_value > 1: raise ValueError('count of value in predicate size may exceed 1')
                if cls.count_of_value < 1: raise ValueError('count of value in predicate size cannot reach 1')
```

injected constraints

```
:- pred(X1), @valasp_validate_pred(X1) != 1.

:- pred(X1,...,Xn), @valasp_validate_pred(pred(X1,...,Xn)) != 1.
```

How does it work

YAML converted to valasp Python code

from here

```
@context.valasp(validate_predicate=True, with_fun=valasp.domain.primitive_types.Fun.FORWARD_IMPLICIT, auto_blacklist=True)
class Size:
        value: Integer
        def __post_init__(self):
                if self.value < 6: raise ValueError(f"Should be >= 6, but received {self.value}")
                if self.value > 100: raise ValueError(f"Should be <= 100, but received {self.value}")
                self.__class__.count_of_value += 1
                if self.value % 2 != 0:
                    raise ValueError('Size must be an even number')
                self.__class__.value = self.value



        @classmethod
        def before_grounding_init_count_value(cls): cls.count_of_value = 0
        @classmethod
        def after_grounding_check_count_value(cls):
                if cls.count_of_value > 1: raise ValueError('count of value in predicate size may exceed 1')
                if cls.count_of_value < 1: raise ValueError('count of value in predicate size cannot reach 1')
```

injected constraints

```
:- pred(X1), @valasp_validate_pred(X1) != 1.

:- pred(X1,...,Xn), @valasp_validate_pred(pred(X1,...,Xn)) != 1.
```

validation trigger

```
def valasp_validate_pred(value):
    Pred(value)
    return 1
```

Let's discuss reusability

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Ensure that a guessed tree encoded by node/1 and tree/2 is not a forest.

```
reach(X) :- X = #min{Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Methodology (?) so far

You have or search for this

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Methodology (?) so far

You have or search for this

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

vi ninja time

```
:%s/vertex/node/gc
:%s/edge/tree/gc
```

Methodology (?) so far

You have or search for this

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

vi ninja time

```
:%s/vertex/node/gc
:%s/edge/tree/gc
```

```
reach(X) :- X = #min{Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Methodology (?) so far

You have or search for this

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

vi ninja time

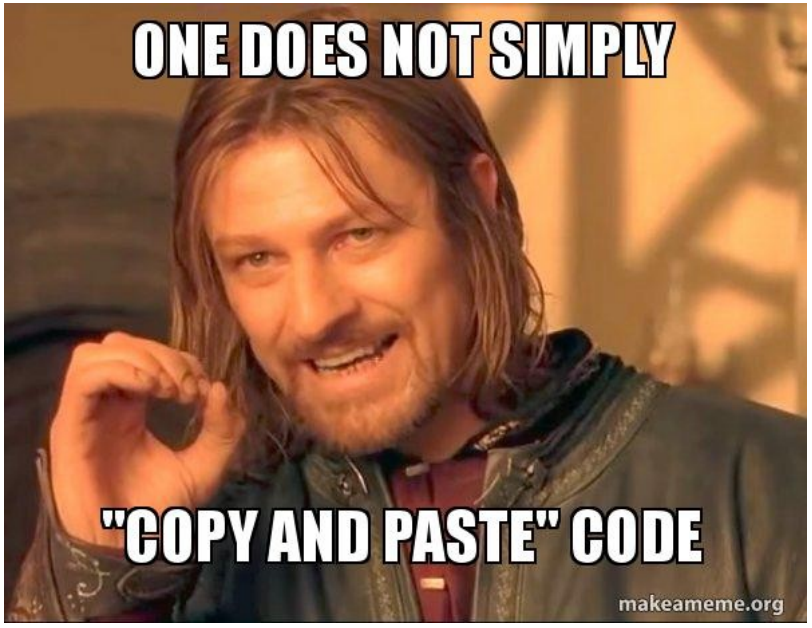
```
:%s/vertex/node/gc
:%s/edge/tree/gc
```


```
reach(X) :- X = #min{Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```


WHAT THE H*LL ARE YOU
TALKING ABOUT?!
NETFLIX

So wrong!

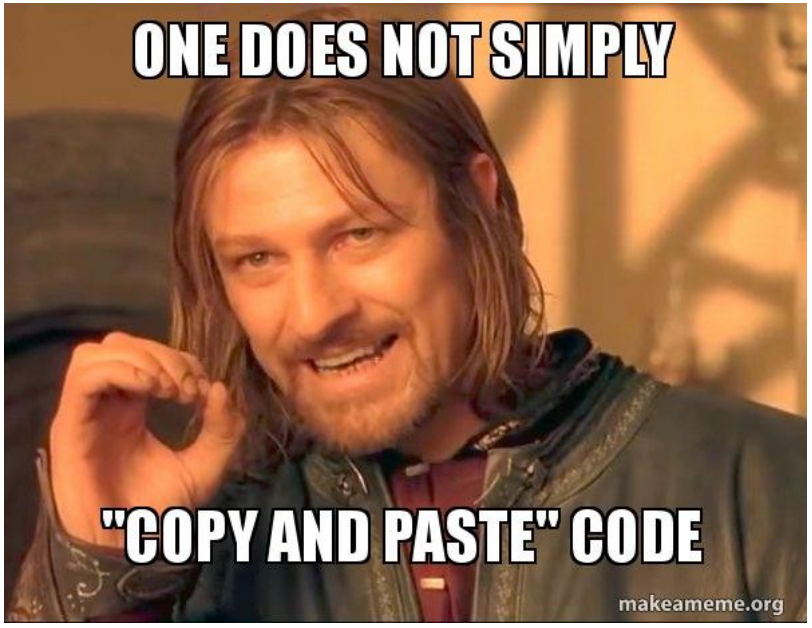
If you copy&paste code,
you must check it,
LINE BY LINE!



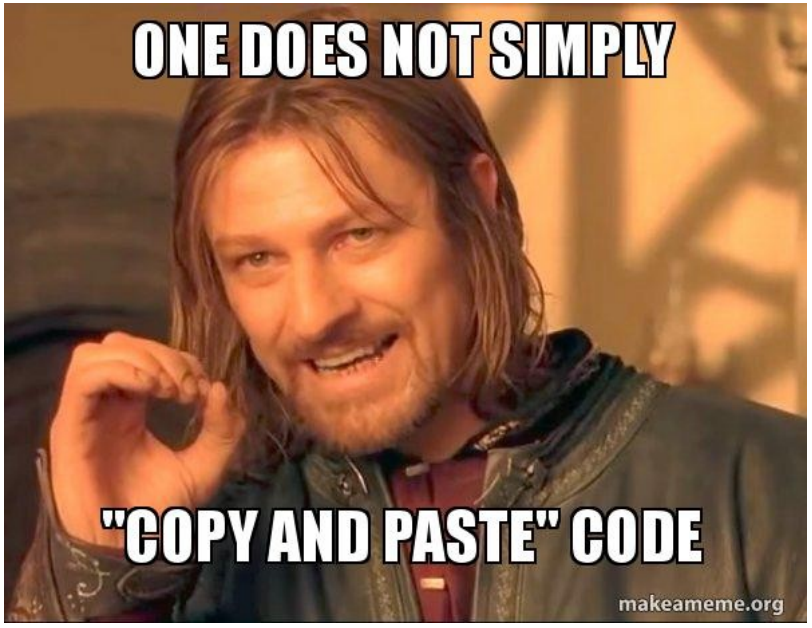
So wrong!

If you copy&paste code,
you must check it,
LINE BY LINE!

Can you reuse someone else's code?
Do you understand it?



So wrong!



If you copy&paste code,
you must check it,
LINE BY LINE!

Can you reuse someone else's code?
Do you understand it?

Can you reuse your own code after some days?

Invariants

Something that does not change or vary.

Invariants

Something that does not change or vary.

It will never be negative

```c
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

Invariants

Something that does not change or vary.

It will never be negative

It returns a non negative integer

```c
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

Invariants

Something that does not change or vary.

It will never be negative

If you refer res outside this block, it is a different variable

It returns a non negative integer

```c
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

Invariants

Something that does not change or vary.

It will never be negative

If you refer res outside this block, it is a different variable

It returns a non negative integer

```c
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

It returns n! under the assumption that n < 13

Invariants

Something that does not change or vary.

It will never be negative

If you refer res outside this block, it is a different variable

It returns a non negative integer

```c
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

It returns n! under the assumption that n < 13

Yes, but don't forget to check n < 13 before calling it, or...

Invariants

It will never be negative

Using a library for exceptions

```
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  if(n > 12) {
    throw (UPPER_BOUND_EXCEEDED, n);
  }
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

n < 13
(no overflow is possible)

It returns n!
(or raises an error)

Invariants

It will never be negative

Using a library for exceptions

```
#include <stdint.h>

uint32_t factorial(uint32_t n) {
  if(n > 12) {
    throw (UPPER_BOUND_EXCEEDED, n);
  }
  uint32_t res = 1;
  for(uint32_t i = 1; i <= n; i++) {
    res *= i;
  }
  return res;
}
```

n < 13
(no overflow is possible)

It returns n!
(or raises an error)

Now we have another nice invariant encoded in our code!

Invariants (in ASP?)

Something that does not change or vary.

Non-monotonic logic

10 languages

From Wikipedia, the free encyclopedia

A **non-monotonic logic** is a formal logic whose conclusion relation is not monotonic. In other words, non-monotonic logics are devised to capture and represent defeasible inferences (cf. defeasible reasoning), i.e., a kind of inference in which reasoners draw tentative conclusions, enabling reasoners to retract their conclusion(s) based on further evidence.[1] Most studied formal logics have a

Invariants (in ASP?)

Something that does not change or vary.

Non-monotonic logic

10 languages

From Wikipedia, the free encyclopedia

A **non-monotonic logic** is a formal logic whose conclusion relation is not monotonic. In other words, non-monotonic logics are devised to capture and represent defeasible inferences (cf. defeasible reasoning), i.e., a kind of inference in which reasoners draw tentative conclusions, enabling reasoners to retract their conclusion(s) based on further evidence.[1] Most studied formal logics have a

Not monotonic... how much can be guaranteed to not change in ASP?

Example

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Example

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Are you sure this predicate is not in the head of other rules?

Example

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Are you sure this predicate is not in the head of other rules?

What about this?

Example

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Are you sure this predicate is not in the head of other rules?

What about this?

And this?

Testing (with no invariant?)

Ensure that a guessed undirected graph encoded by vertex/1 and edge/2 is connected.

```
reach(X) :- X = #min{Y : vertex(X)}.
reach(Y) :- reach(X), edge(X,Y).
:- vertex(X), not reach(X).
```

Are you sure this predicate is not in the head of other rules?

What about this?

And this?

Is there anything that really makes sense to test here?

Can you team work?

Yes, if this is your definition of team working!

Can you team work?

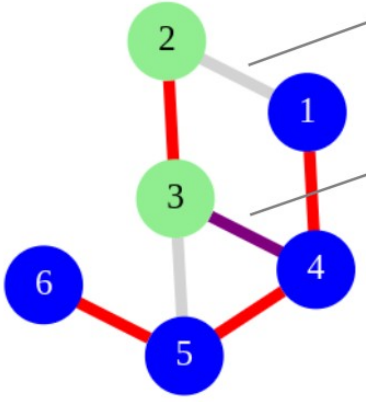
if this is your definition of team working!

Can you team work?

Problem (part of a bigger project)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.



Edges (1,2) and (3,5) of the graph are not part of the spanning tree

Closing here has impact |2-4| = 2

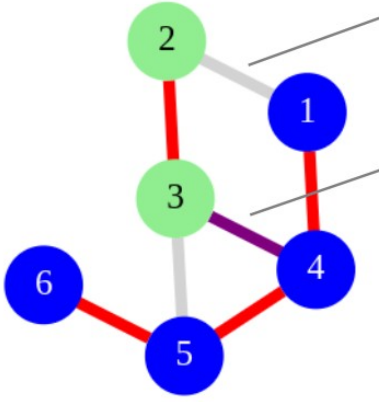
```
node(1..6).
link(1,4).  link(1,2).
link(2,3).  link(3,4).
link(3,5).  link(4,5).
link(5,6).
```

Can you team work?

Problem (part of a bigger project)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.



Edges (1,2) and (3,5) of the graph are not part of the spanning tree

Closing here has impact |2-4| = 2

```
node(1..6).
link(1,4).  link(1,2).
link(2,3).  link(3,4).
link(3,5).  link(4,5).
link(5,6).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

Undirected graph, let me close link/2 symmetrically

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

Undirected graph, let me close link/2 symmetrically

Let me guess (# of nodes – 1) links

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

Undirected graph, let me close link/2 symmetrically

Let me guess (# of nodes – 1) links

Close symmetrically also tree/2

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

Undirected graph, let me close link/2 symmetrically

Let me guess (# of nodes – 1) links

Close symmetrically also tree/2

Check that the tree is not a forest

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Alpha (let's say me)

Given a graph representing road segments, we are interested in finding a spanning tree to build a highway network.

Undirected graph, let me close link/2 symmetrically

Let me guess (# of nodes – 1) links

Close symmetrically also tree/2

Check that the tree is not a forest

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

Team Bravo (let's say a friend of mine)

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Team Bravo (let's say a friend of mine)

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.

I will receive tree/2 from Alpha, and I will guess the closed segment

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Team Bravo (let's say a friend of mine)

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.

I will receive tree/2 from Alpha, and I will guess the closed segment

Everything else stays open

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Team Bravo (let's say a friend of mine)

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.

I will receive tree/2 from Alpha, and I will guess the closed segment

Everything else stays open

Let's measure the impact

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Team Bravo (let's say a friend of mine)

For each such network proposal, we want to understand the impact of closing every single road segment in terms of the resulting tree-size-difference between connected points.

I will receive tree/2 from Alpha, and I will guess the closed segment

Everything else stays open

Let's measure the impact

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

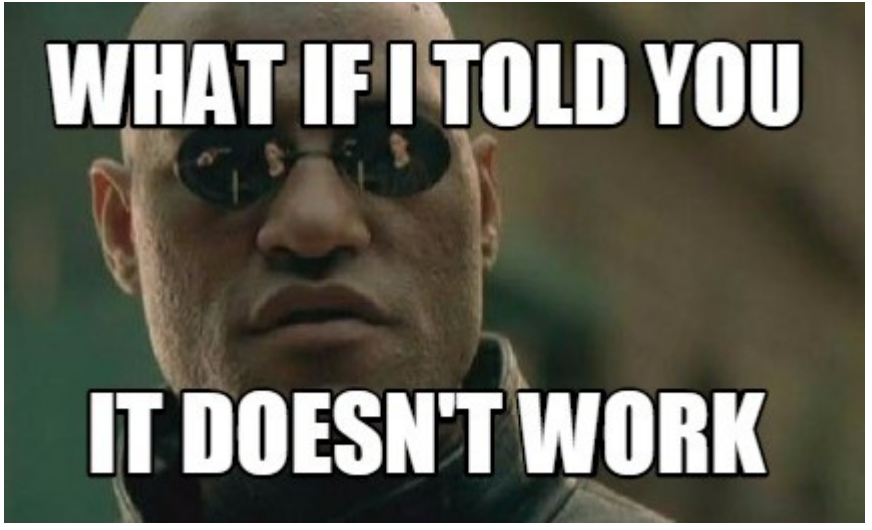
```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```

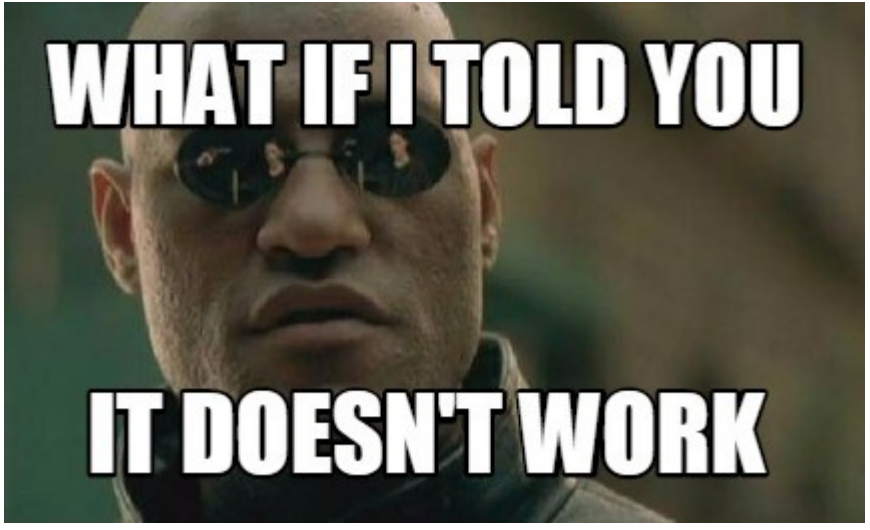
```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```



```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

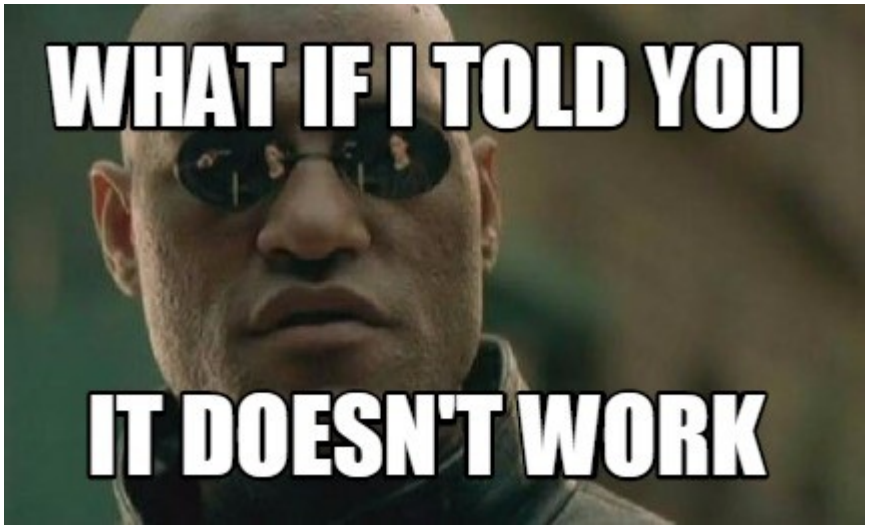
```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```



reach/2: same name, different meanings

```
link(X,Y) :- link(Y,X).
{tree(X,Y) : link(X,Y), X < Y} = C-1
          :- C = #count {X : node(X)}.
tree(X,Y) :- tree(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), tree(X,Y).
:- node(X), not reach(X).
```

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
     reach(Z); -1,Z : node(Z), not reach(Z)}.
```



reach/2: same name, different meanings

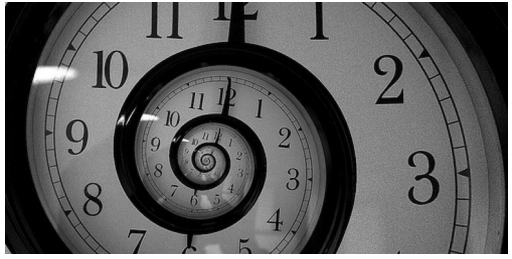
What about tree/2?

We also need a notion of locality!

Fix it (with a TTD*)

Fix it (with a TTD*)

*TTD Time Travel Device



Step 1. Go back in time

Fix it (with a TTD*)

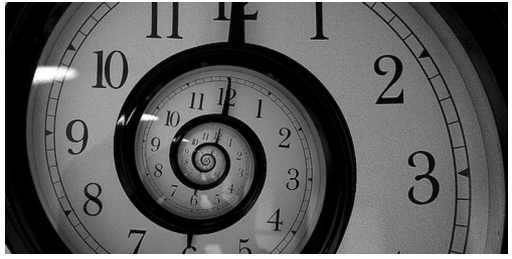
*TTD Time Travel Device

Step 1. Go back in time

Step 2. Explain to myself-v1 the need for renaming

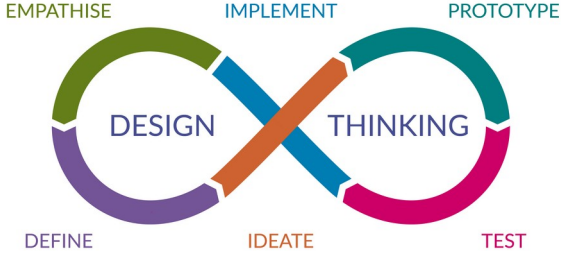
Fix it (with a TTD*)

*TTD Time Travel Device



Step 1. Go back in time

Step 2. Explain to myself-v1 the need for renaming

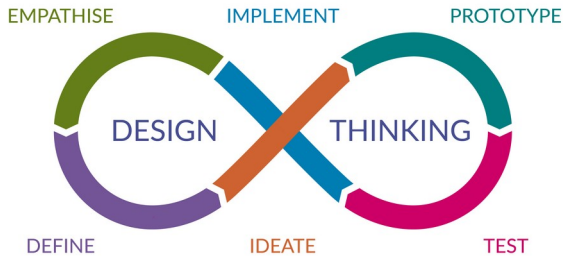


Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

Fix it (with a TTD*)

*TTD Time Travel Device

```
__template__("symmetric closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Y) :- relation(Y,X).
__end__.

__template__("reachable nodes").
    reach(X) :- start(X).
    reach(Y) :- reach(X), link(X,Y).
__end__.
```



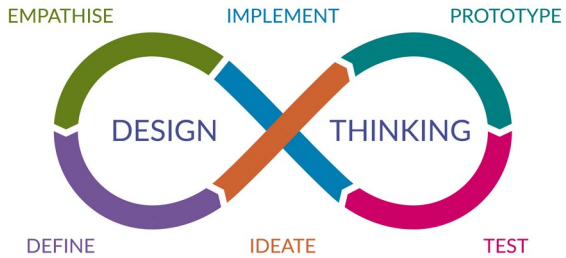
Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

Fix it (with a TTD*)

*TTD Time Travel Device

```
__template__("symmetric closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Y) :- relation(Y,X).
__end__.

__template__("reachable nodes").
    reach(X) :- start(X).
    reach(Y) :- reach(X), link(X,Y).
__end__.
```

```
__template__("connected graph").
    __start(X) :- X = #min{Y : node(Y)}.
    __apply_template__("reachable nodes", (start, __start), (reach, __reach)).
    :- node(X), not __reach(X).
__end__.
```



Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

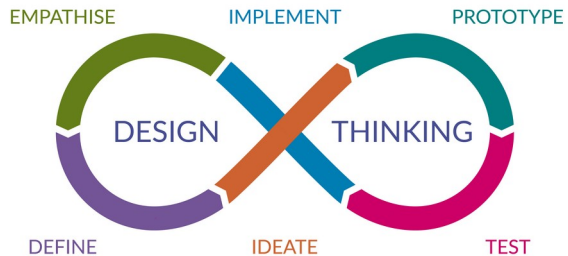
Fix it (with a TTD*)

*TTD Time Travel Device

```
__template__("symmetric closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Y) :- relation(Y,X).
__end__.

__template__("reachable nodes").
    reach(X) :- start(X).
    reach(Y) :- reach(X), link(X,Y).
__end__.
```

```
__template__("connected graph").
    __start(X) :- X = #min{Y : node(Y)}.
    __apply_template__("reachable nodes", (start, __start), (reach, __reach)).
    :- node(X), not __reach(X).
__end__.
```

```
__template__("spanning tree of undirected graph").
    {tree(X,Y) : link(X,Y), X < Y} = C - 1 :- C = #count{X : node(X)}.
    __apply_template__("symmetric closure", (relation, tree), (closure, __tree)).
    __apply_template__("connected graph", (link, __tree)).
__end__.
```



Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

Fix it (with a TTD*)

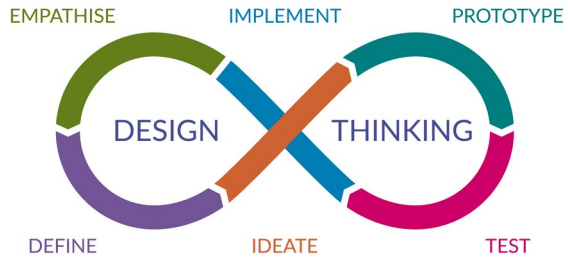
*TTD Time Travel Device

```
__template__("symmetric closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Y) :- relation(Y,X).
__end__.

__template__("reachable nodes").
    reach(X) :- start(X).
    reach(Y) :- reach(X), link(X,Y).
__end__.
```

```
__template__("connected graph").
    __start(X) :- X = #min{Y : node(Y)}.
    __apply_template__("reachable nodes", (start, __start), (reach, __reach)).
    :- node(X), not __reach(X).
__end__.
```

```
__template__("spanning tree of undirected graph").
    {tree(X,Y) : link(X,Y), X < Y} = C - 1 :- C = #count{X : node(X)}.
    __apply_template__("symmetric closure", (relation, tree), (closure, __tree)).
    __apply_template__("connected graph", (link, __tree)).
__end__.
```

```
__apply_template__("symmetric closure", (relation, link), (closure, link)).
__apply_template__("spanning tree of undirected graph").
```

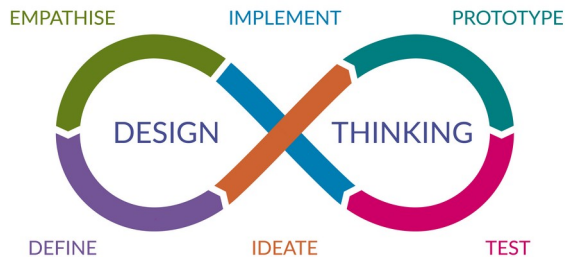


Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

Fix it (with a TTD*)

*TTD Time Travel Device

```
link(X,Y) :- link(X,Y).
link(X,Y) :- link(Y,X).
(C-1) = { tree(X,Y): link(X,Y), X < Y } :- C = #count { X: node(X) }.
__tree_03cf88a4_...(X,Y) :- tree(X,Y).
__tree_03cf88a4_...(X,Y) :- tree(Y,X).
__start_b72a4bf4_..._03cf88a4_...(X) :- X = #min { Y: node(Y) }.
__reach_b72a4bf4_..._03cf88a4_...(X) :- __start_b72a4bf4_..._03cf88a4_...(X).
__reach_b72a4bf4_..._03cf88a4_...(Y) :- __reach_b72a4bf4_..._03cf88a4_...(X); __tree_03cf88a4_...(X,Y).
:- node(X); not __reach_b72a4bf4_..._03cf88a4_...(X).
```



Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

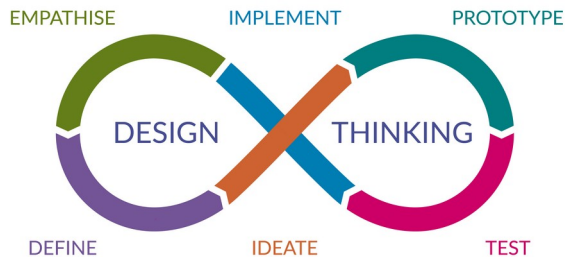
Step 4. Transpile the code, and send it to Bravo

Fix it (with a TTD*)

*TTD Time Travel Device

No idea what Bravo will use, but they will not use this name! (an invariant)

```
link(X,Y) :- link(X,Y).
link(X,Y) :- link(Y,X).
(C-1) = { tree(X,Y): link(X,Y), X < Y } :- C = #count { X: node(X) }.
__tree_03cf88a4_...(X,Y) :- tree(X,Y).
__tree_03cf88a4_...(X,Y) :- tree(Y,X).
__start_b72a4bf4_..._03cf88a4_...(X) :- X = #min { Y: node(Y) }.
__reach_b72a4bf4_..._03cf88a4_...(X) :- __start_b72a4bf4_..._03cf88a4_...(X).
__reach_b72a4bf4_..._03cf88a4_...(Y) :- __reach_b72a4bf4_..._03cf88a4_...(X); __tree_03cf88a4_...(X,Y).
:- node(X); not __reach_b72a4bf4_..._03cf88a4_...(X).
```



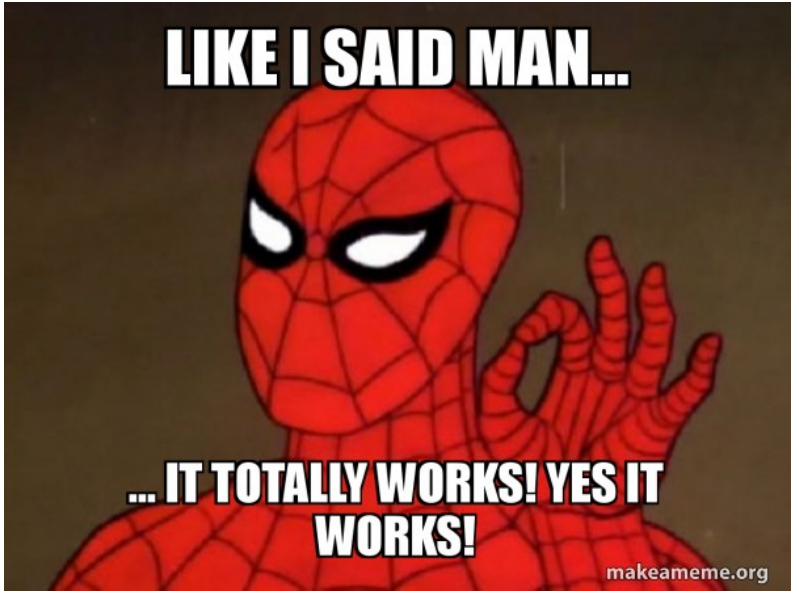
Step 3. Work on templates: instantiated by predicate renaming; locality by UUID

Step 4. Transpile the code, and send it to Bravo

```
link(X,Y) :- link(X,Y).
link(X,Y) :- link(Y,X).
(C-1) = { tree(X,Y): link(X,Y), X < Y } :- C = #count { X: node(X) }.
__tree_03cf88a4_...(X,Y) :- tree(X,Y).
__tree_03cf88a4_...(X,Y) :- tree(Y,X).
__start_b72a4bf4_..._03cf88a4_...(X) :- X = #min { Y: node(Y) }.
__reach_b72a4bf4_..._03cf88a4_...(X) :- __start_b72a4bf4_..._03cf88a4_...(X).
__reach_b72a4bf4_..._03cf88a4_...(Y) :- __reach_b72a4bf4_..._03cf88a4_...(X); __tree_03cf88a4_...(X,Y).
:- node(X); not __reach_b72a4bf4_..._03cf88a4_...(X).
```

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Doesn't clash!

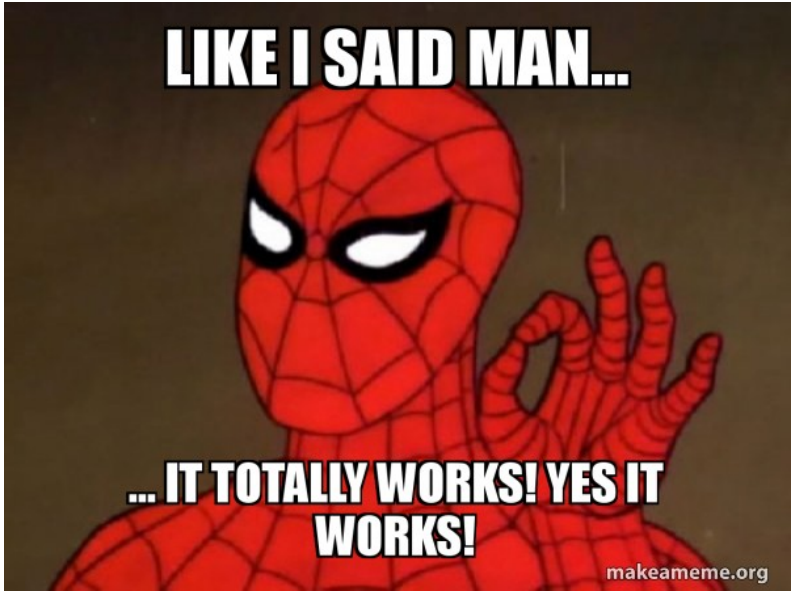


```
link(X,Y) :- link(X,Y).
link(X,Y) :- link(Y,X).
(C-1) = { tree(X,Y): link(X,Y), X < Y } :- C = #count { X: node(X) }.
__tree_03cf88a4_...(X,Y) :- tree(X,Y).
__tree_03cf88a4_...(X,Y) :- tree(Y,X).
__start_b72a4bf4_..._03cf88a4_...(X) :- X = #min { Y: node(Y) }.
__reach_b72a4bf4_..._03cf88a4_...(X) :- __start_b72a4bf4_..._03cf88a4_...(X).
__reach_b72a4bf4_..._03cf88a4_...(Y) :- __reach_b72a4bf4_..._03cf88a4_...(X); __tree_03cf88a4_...(X,Y).
:- node(X); not __reach_b72a4bf4_..._03cf88a4_...(X).
```

```
{out(X,Y) : tree(X,Y)} = 1.
in(X,Y) :- tree(X,Y), not out(X,Y).
in(X,Y) :- in(Y,X).
reach(X) :- X = #min {Y : node(Y)}.
reach(Y) :- reach(X), in(X,Y).
impact(X,Y,|C|) :- out(X,Y), C = #sum{1,Z :
      reach(Z); -1,Z : node(Z), not reach(Z)}.
```

Doesn't clash!

Don't be like Team Bravo, don't write again and again the same rules!



Testing? Test small units

Testing? Test small units

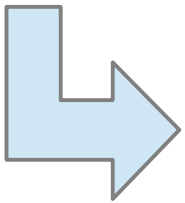
Templates perhaps?

```
def test_validate_in_all_models_transitive_closure():
    program = Template.expand_program(SymbolicProgram.parse("""
__apply_template__("@dumbo/transitive closure", (relation, link), (closure, link)).

link(a,b).
link(b,c).
    """))

    validate_in_all_models(
        program=program,
        true_atoms=Model.of_atoms("link(a,b) link(b,c) link(a,c)".split()),
    )

    with pytest.raises(ValueError):
        validate_in_all_models(program=program, true_atoms=Model.of_atoms("link(a,a)".split()))

    with pytest.raises(ValueError):
        validate_in_all_models(
            program=program,
            false_atoms=Model.of_atoms("link(a,a)".split()),
        )
```

```
__template__("@dumbo/exact copy (arity 2)").
    output(X0,X1) :- input(X0,X1).
    :- output(X0,X1), not input(X0,X1).
__end__.

__template__("@dumbo/transitive closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Z) :- closure(X,Y), relation(Y,Z).
__end__.

__template__("@dumbo/transitive closure guaranteed").
    __apply_template__("@dumbo/transitive closure", (closure, __closure)).
    __apply_template__("@dumbo/exact copy (arity 2)", (input, __closure), (output, closure)).
__end__.
```

Not a lot of extra code.
Easily switch from one version to the other (e.g. to debug).

```
__template__("@dumbo/exact copy (arity 2)").
    output(X0,X1) :- input(X0,X1).
    :- output(X0,X1), not input(X0,X1).
__end__.

__template__("@dumbo/transitive closure").
    closure(X,Y) :- relation(X,Y).
    closure(X,Z) :- closure(X,Y), relation(Y,Z).
__end__.

__template__("@dumbo/transitive closure guaranteed").
    __apply_template__("@dumbo/transitive closure", (closure, __closure)).
    __apply_template__("@dumbo/exact copy (arity 2)", (input, __closure), (output, closure)).
__end__.
```



```
__template__("@dumbo/transitive closure guaranteed").
    __closure(X,Y) :- relation(X,Y).
    __closure(X,Z) :- __closure(X,Y); relation(Y,Z).
    closure(X0,X1) :- __closure(X0,X1).
    :- closure(X0,X1); not __closure(X0,X1).
__end__.
```

Compute on local predicates, then copy on global predicates. This way the template gets an I/O-flow.

Not a lot of extra code.
Easily switch from one version to the other (e.g. to debug).

Simple scaffold

main.py

```python
import webbrowser

from dumbo_asp.primitives import SymbolicProgram, Template, Model
from dumbo_asp.queries import pack_asp_chef_url

program = SymbolicProgram.parse("""
node(1..6).
link(1,2).
link(1,4).
link(2,3).
link(3,4).
link(3,5).
link(4,5).
link(5,6).

__apply_template__("@dumbo/symmetric closure", (relation, link), (closure, link)).
__apply_template__("@dumbo/spanning tree of undirected graph").
""")
program = Template.expand_program(program)
print(program)

model = Model.of_program(program).filter(when=lambda atom: atom.predicate_name == "tree")
print("\n\nModel:", model)

webbrowser.open(
    pack_asp_chef_url(
        "http://localhost:5188/#eJytU8mSmzAU/CWBh0lxyCHjBQsbuQw2Wm4gYiMQSxVm89fnyTOZ2PecVK2
        [model]
    )
)
```

Share content with ASP Chef
click to open

```
link(X,Y) :- link(X,Y).
link(X,Y) :- link(Y,X).
(C-1) = { tree(X,Y): link(X,Y), X < Y } :- C = #count { X: node(X) }.
__tree_90b873c6_b4e8_49b0_a430_fbb25e963621(X,Y) :- tree(X,Y).
__tree_90b873c6_b4e8_49b0_a430_fbb25e963621(X,Y) :- tree(Y,X).
__start_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(X) :- X = #min { Y: node(Y) }.
__reach_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(X) :- __start_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(X).
__reach_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(Y) :- __reach_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(X); __tree_90b873c6_b4e8_49b0_a430_fbb25e963621(X,Y).
#false :- node(X); not __reach_c574f8d5_af4c_4b39_bdf2_24d3b174b6a1_90b873c6_b4e8_49b0_a430_fbb25e963621(X).


Model: tree(1,2) tree(1,4) tree(2,3) tree(4,5) tree(5,6)
```

Expand templates

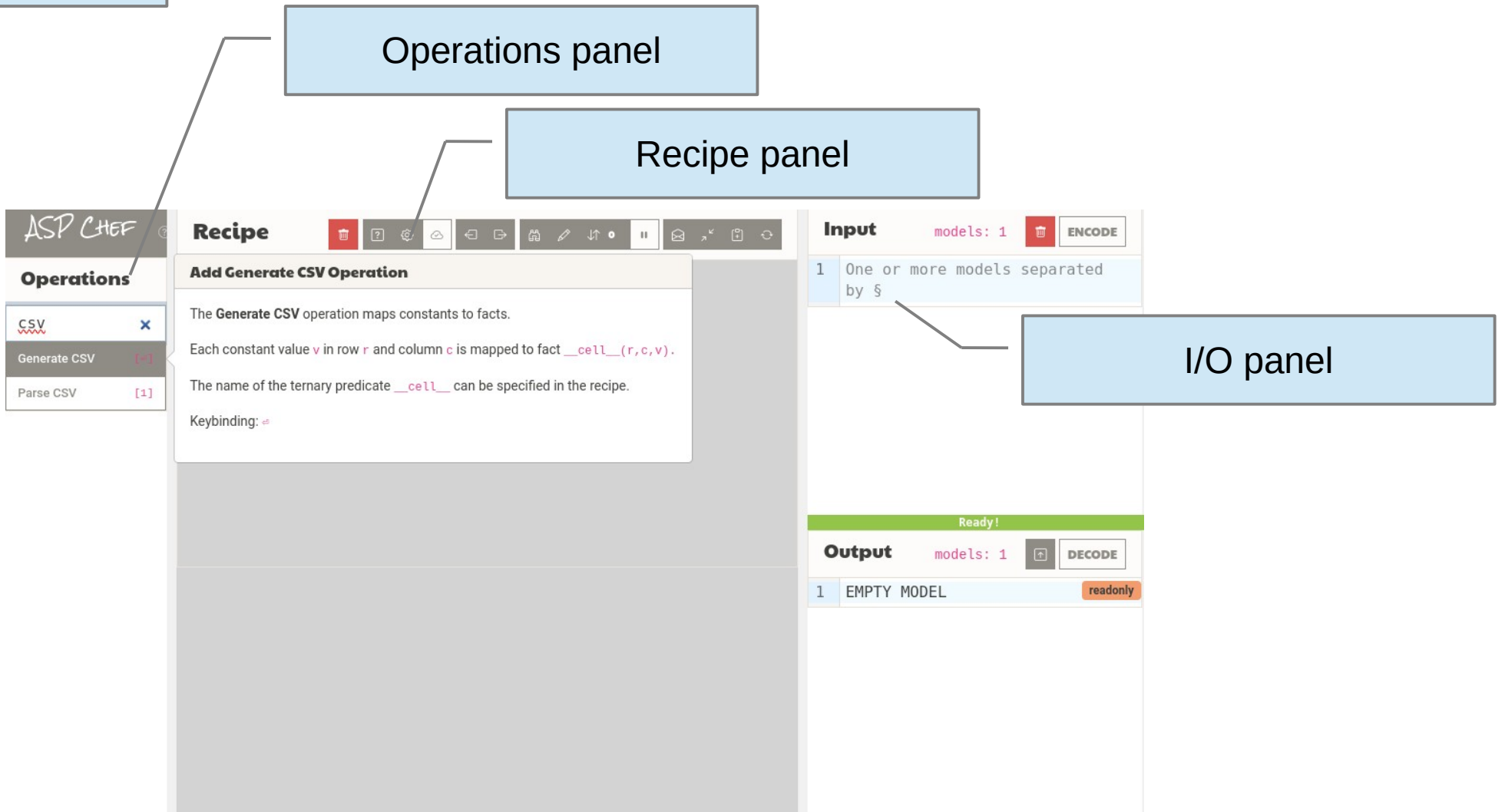
Visualize with ASP Chef!

```
$ poetry new templates-example
$ poetry add dumbo-asp
```

Result
click to open


NOT UGLY
WONDERBAR
BUT WONDERFUL
imgflip.com

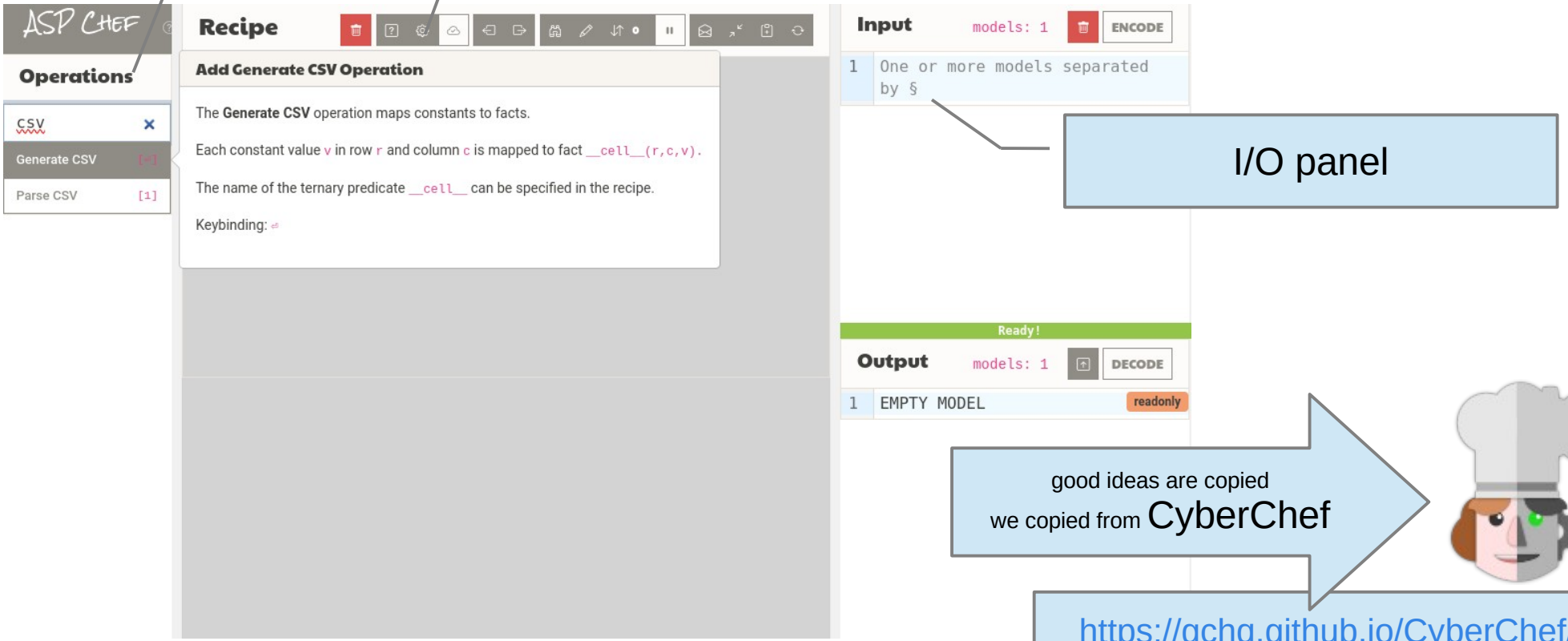
ASP Chef


Operations panel
Recipe panel
I/O panel
ASP CHEF
Operations
CSV
Generate CSV [1]
Parse CSV [1]
Recipe
Add Generate CSV Operation
The Generate CSV operation maps constants to facts.
Each constant value v in row r and column c is mapped to fact __cell__(r,c,v).
The name of the ternary predicate __cell__ can be specified in the recipe.
Keybinding:
Input
models: 1
ENCODE
1
One or more models separated by §
Ready!
Output
models: 1
DECODE
1
EMPTY MODEL
readonly

ASP Chef

Operations panel

Recipe panel



I/O panel

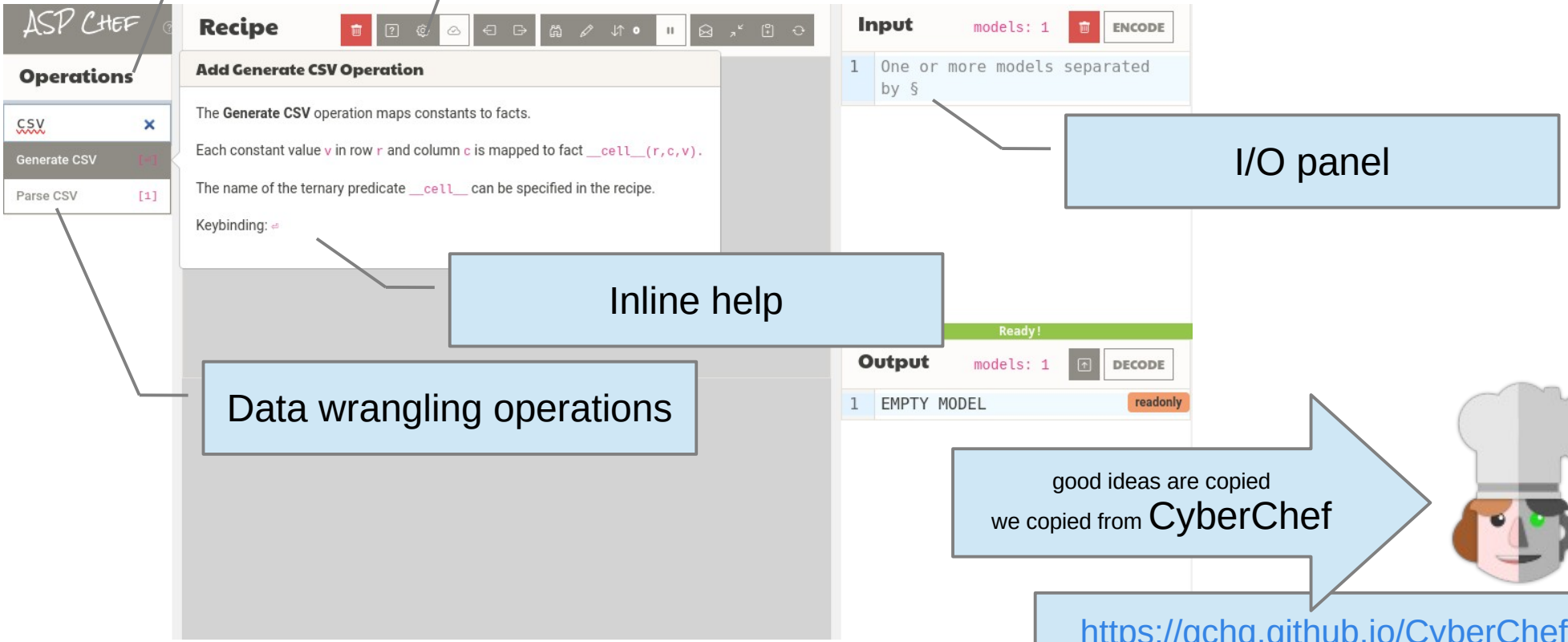
good ideas are copied
we copied from CyberChef

https://gchq.github.io/CyberChef/

ASP Chef

Operations panel

Recipe panel



I/O panel

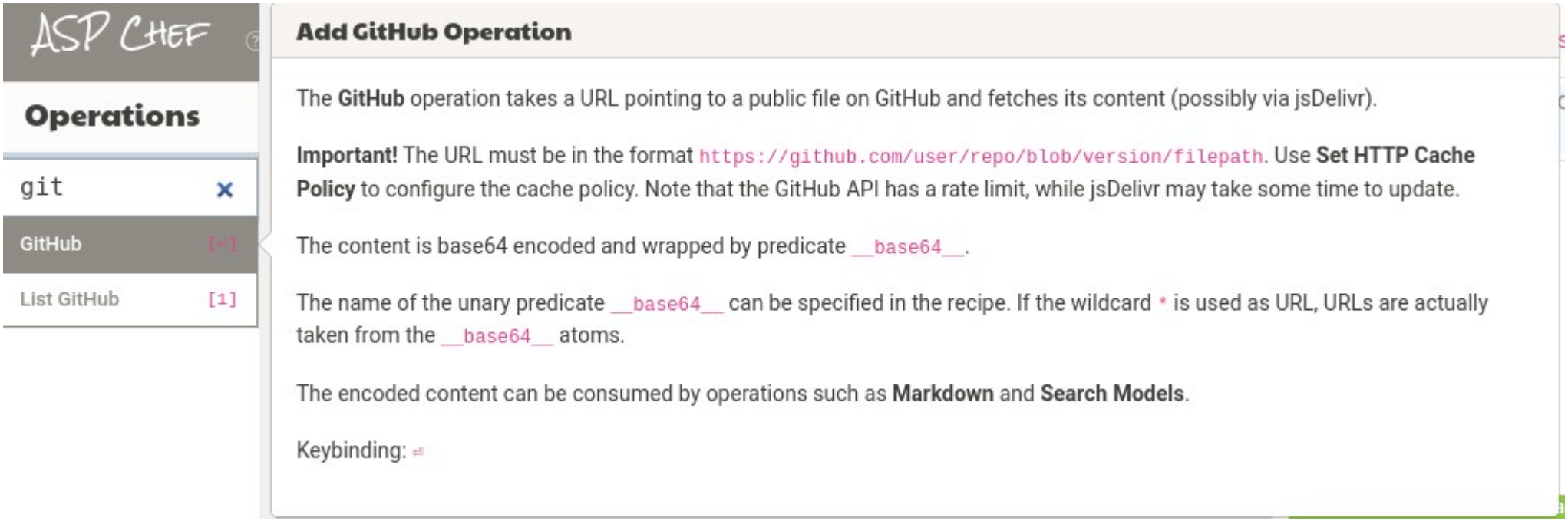
Inline help

Data wrangling operations

good ideas are copied
we copied from CyberChef

https://gchq.github.io/CyberChef/

Load external content



ASP CHEF

Operations

mar

Markdown [-]

Add Markdown Operation

The **Markdown** operation shows the markdown encoded content in each model in input. Latex math expressions are supported; e.g., `\\(x = 4\\)` or `\\[x = 4\\]`.

Models can be queried with the mustache syntax `{{ program with #show directives }}` (or `{{= (terms) : conjunctive_query }}` as a shortcut for `{{ #show (terms) : conjunctive_query. }}`).

Output can be ordered via the varadics predicate `sort`, specifying the indices of the terms to use (positive for ascending, negative for descending).

The separator of the obtained substitutions can be specified with `separator("\n")`. Similarly, `term_separator/1`, `prefix/1` and `suffix/1` can be used to customize the print of each obtained substitution.

Tables can be specified by the varadics predicates `th` and `tr`. Alignment of columns (by default left) can be specified in `th` by terms `left("column header")`, `center("col")`, `right("col")`. Alternatively, `matrix/3` can be used to produce a table by specifying values for each cell. Row 0 can be used to provide header cells. Columns are indexed by 1.

Ordered and unordered lists can be specified by the varadics predicates `ol` and `ul`.

Predicates `png/1`, `gif/1` and `jpeg/1` can be used to show a Base64-encoded PNG image.

Predicate `base64/1` decodes Base64-encoded content.

Predicate `qrcode/1` (and links `[...](qrcode)`) are shown as QR-codes.

The input is echoed in output.

Keybinding: ⏎

Produce HTML snippets (querying the model)

ASP CHEF

Operations

serv

Server

Add Server Operation

The **Server** operation asks a remote or local server to process the input.

Input and output must be arrays of models. Additional options can be sent to the server; it is up to the server to interpret such options. For example, a server may implement the *Search Models* operation and accept options like a predicate to decode, the number of models to compute and so on.

Note that the result is cached (as for many other operations). In order to fetch new data even if the input didn't change, add an *Invalidate Cache* operation.

Keybinding:

Attach clingraph or ASPECT

As yet another example, a single graph can be obtained by clingraph via the following server:

```python
import base64
import os
import tempfile

from clingraph.orm import Factbase
from clingraph.graphviz import compute_graphs, render
from IPython.display import Image
from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware


app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5188", "https://asp-chef.alviano.net"],
    allow_credentials=False,
    allow_methods=["POST"],
    allow_headers=["*"],
)


@app.post("/")
async def process(request: Request):
    json = await request.json()
    input_part = json["input_part"]
    decoded_input = json["decoded_input"]
    options = json["options"].strip()
    predicate = options if options else "png"

    fb = Factbase()
    program = '\n'.join([f"{atom}." for atom in input_part]) + '\n'.join(decoded_input)
    fb.add_fact_string(program)
    graphs = compute_graphs(fb)

    with tempfile.NamedTemporaryFile(mode="w", suffix=".png") as tmp_file:
        directory, filename = os.path.split(tmp_file.name)
        render(graphs, directory=directory, format="png", name_format=os.path.splitext(filename)[0])
        encoded = base64.b64encode(open(tmp_file.name, "rb").read())

    return {"models" : [[f"{predicate}(\"{encoded.decode()}\")"]]}
```

The same approach is suitable to obtain a graph with ASPECT (here using *graph mode*):

```python
import base64
import os
import subprocess
import tempfile

from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware


DIRNAME = os.path.dirname(os.path.realpath(__file__))

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:5188", "https://asp-chef.alviano.net"],
    allow_credentials=False,
    allow_methods=["POST"],
    allow_headers=["*"],
)


@app.post("/")
async def process(request: Request):
    json = await request.json()
    input_part = json["input_part"]
    decoded_input = json["decoded_input"]
    options = json["options"].strip()
    predicate = options if options else "png"

    program = '\n'.join([f"{atom}." for atom in input_part]) + '\n'.join(decoded_input)

    with tempfile.TemporaryDirectory() as tmp_dir:
        with open(f"{tmp_dir}/program.lp", "w") as program_file:
            program_file.write(program)
        subprocess.run(["java", "-jar", f"{DIRNAME}/ASPECT.jar", "--graph", "program.lp"], cwd=tmp_d
        subprocess.run(["pdftocairo", "aspect_out_program1.pdf", "-png"], cwd=tmp_dir)
        encoded = base64.b64encode(open(f"{tmp_dir}/aspect_out_program1-1.png", "rb").read())

    return {"models" : [[f"{predicate}(\"{encoded.decode()}\")"]]}
```

Baking delay (250ms)

250

Remote clingo

http://localhost:8000

GitHub API Token (set it only on trusted browsers)

••

GitHub short links configuration

username	repository
/	wise@unical

CLOSE

Shorten URLs and store them in your GitHub repos

Nice effects

click to open

Recipe

ASP Chef

(get this list on https://asp-chef.alviano.net/s)

Examples

- ASP Chef QR Code Generator
- Marketplace logistics

Tutorials

- Basic usage
- Fortress
- Fighting with the gang of Billy the Kid
- Aquarium

https://asp-chef.alviano.net/s

Summing up

I dream of an ecosystem that

- eases **data validation**
- enables **code reusability**
- leads to the development of **libraries**
- lets us use code even if we **don’t understand** it (yes!)
- ignites the spark of **Test-Driven Development**
- simplifies resource **sharing**
- **connects** products of different groups

Summing up

I dream of an ecosystem that

- eases **data validation**
- enables **code reusability**
- leads to the development of **libraries**
- lets us use code even if we **don’t understand** it (yes!)
- ignites the spark of **Test-Driven Development**
- simplifies resource **sharing**
- **connects** products of different groups

An ecosystem that lets us SHINE!

Questions