

# Query Answering

Cours confiné GraphIK 5

20 mai 2020

Dans les épisodes précédents:  
application de règle

$$p(X), \text{ not } q(X, Y), p(Y), \text{ not } q(Y, X) \rightarrow r(X)$$

$$F \left\{ \begin{array}{l} p(a) \\ q(a, b) \end{array} \right.$$



$r(a) ?$

$$G \left\{ \begin{array}{l} p(a) \\ q(b, a) \end{array} \right.$$



$r(a) ?$

# Dans les épisodes précédents: ASPERIX

$B^+, \text{not } B_1^-, \dots, \text{not } B_i^-, \dots, \text{not } B_k^- \rightarrow H$

$h$  ↗

IN	OUT	MBT
$F_k$	$O_k$	$M_k$

Si la règle est déclenchée par  $F_k$ , on va pouvoir l'évaluer.

On applique

On n'applique pas

IN	OUT	MBT
$F_k \cup h(H)$	$O_k \cup \{h(B_1^-), \dots, h(B_k^-)\}$	$M_k$

IN	OUT	MBT
$F_k$	$O_k$	$M_k \cup \{h(B^-)\}$

Ici  $h(H)$  et pas  $h^s(H)$   
car Skolem

Ici on rajoute  $k$  éléments,  
1 par corps négatif

Ici on rajoute 1 élément,  
disjonction des  $k$  corps négatifs

# Database Repairs

# Réparation (d'une base faits)

$K = \langle F, T \rangle$  //  $F$  grounded,  $T$ : règles exist. + contraintes

Une **réparation** de  $F$  relativement à  $T$  (plus court, de  $K$ ) est un sous ensemble maximal  $F'$  de  $F$  tq  $\langle F', T \rangle$  est consistant.

## Exemple 1

$F$ :  $p(a), q(a), r(a).$   
 $T$ :  $p(X), q(X) \rightarrow \perp$

2 réparations de  $K$ :

- $p(a), r(a).$
- $q(a), r(a).$

Si  $K$  est consistante,  $F$  est la seule réparation, sinon entre 0 et  $2^f$  réparations.

# Skolémisation et arrêt de la dérivation (1)

$$p(X, Y) \rightarrow p(X, Z). \\ p(a, b).$$

corps-skolemisation

$$p(X, Y) \rightarrow p(X, f(X, Y)). \\ p(a, b).$$
$$p(a, f(a, b)). \\ p(a, f(a, f(a, b))). \\ p(a, f(a, f(a, f(a, b)))). \\ \dots$$

« similaire » à oblivious chase

frontiere-skolemisation

$$p(X, Y) \rightarrow p(X, f(X)). \\ p(a, b).$$
$$p(a, f(a)). \\ p(a, f(a)).$$

Si corps-sk-chase s'arrete, alors fr-sk-chase s'arrete aussi.

- La réciproque n'est pas vraie
- Les 2 chases ont pour résultat un modèle universel

# Skolémisation et arrêt de la dérivation (2)

$$p(X, Y) \rightarrow p(X, Z), p(T, Y). \\ p(a, b).$$

frontiere-skolemisation

$$p(X, Y) \rightarrow p(X, f(X, Y)), p(g(X, Y), Y). \\ p(a, b).$$
$$p(a, f(a, b)), p(g(a, b), b).$$
$$p(a, f(a, f(a, b))), p(g(a, f(a, b)), f(a, b)). \\ p(g(a, b), f(g(a, b), b)), p(g(g(a, b), b), b).$$

...

pieces-skolemisation

$$p(X, Y) \rightarrow p(X, f(X)), p(g(Y), Y). \\ p(a, b).$$
$$p(a, f(a)), p(g(b), b). \\ p(a, f(a)), p(g(b), b). \\ p(a, f(a)), p(g(b), b).$$

Si fr-sk-chase s'arrete, alors pieces-sk-chase s'arrete aussi.

- La réciproque n'est pas vraie
- Les 2 chases ont pour résultat un modèle universel

# Skolémisation et résultat de la dérivation (1)

$p(X, Y) \rightarrow q(X, Z), r(Z, Z).$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X).$   
 $p(a, b), p(a, c).$

corps-skolemisation

$p(X, Y) \rightarrow q(X, f(X, Y)), r(f(X, Y), f(X, Y)).$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X).$   
 $p(a, b), p(a, c).$

frontiere-skolemisation

$p(X, Y) \rightarrow q(X, f(X)), r(f(X), f(X)).$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X).$   
 $p(a, b), p(a, c).$

$q(a, f(a, b)), r(f(a, b), f(a, b)).$   
 $q(a, f(a, c)), r(f(a, c), f(a, c)).$



$t(a).$

Le choix de la skolemisation  
change la sémantique: on  
n'obtient pas les mêmes  
modèles stables.

$q(a, f(a)), r(f(a), f(a)).$   
 $q(a, f(a)), r(f(a), f(a)).$





# Skolémisation et résultat de la dérivation (2)

$p(X, Y) \rightarrow q(X, Z), r(Z, Z), q(Y, U), r(U, U) .$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X) .$   
 $p(a, b), p(a, c) .$

frontiere-skolemisation

$p(X, Y) \rightarrow q(X, f(X, Y)), r(f(X, Y), f(X, Y)), q(Y, g(X, Y)), r(g(X, Y), g(X, Y)) .$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X) .$   
 $p(a, b), p(a, c) .$

...  
 $t(a) .$



pieces-skolemisation

$p(X, Y) \rightarrow q(X, f(X)), r(f(X), f(X)), q(Y, f(Y)), r(f(Y), f(Y)) .$   
 $q(X, Y), q(X, Z), \text{not } r(Y, Z) \rightarrow t(X) .$   
 $p(a, b), p(a, c) .$

....

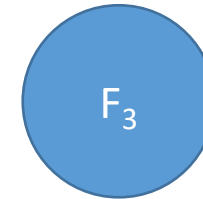
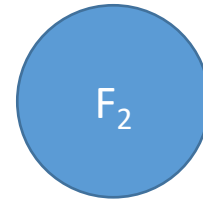
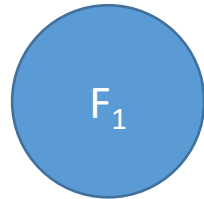
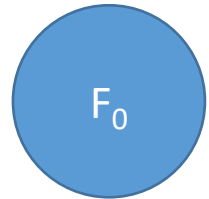


Le choix de la skolemisation change la sémantique: on n'obtient pas les mêmes modèles stables.

Sémantique(s) des autres chases

# Bonnes dérivations et bonnes branches

Corps-sk-chase / frontier-sk-chase / piece-sk-chase



Persistence

Complétude de la dérivation



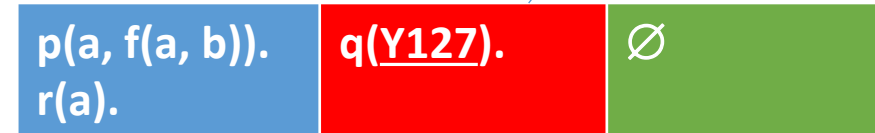
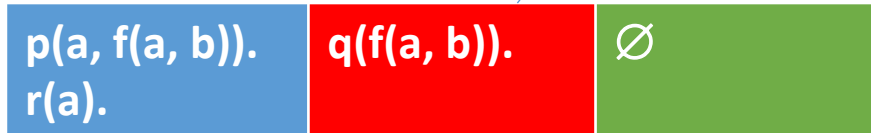
Satisfaction des OUT

Complétude de l'évaluation  
Satisfaction des MBT

Que se passe-t-il si on remplace notre « oblivious chase » (aka corps-sk-chase) par un autre chase?

# Facile: « attacher les variables »

$p(X, Y)$ , not  $q(Y) \rightarrow r(X)$ .



Version skolemisée

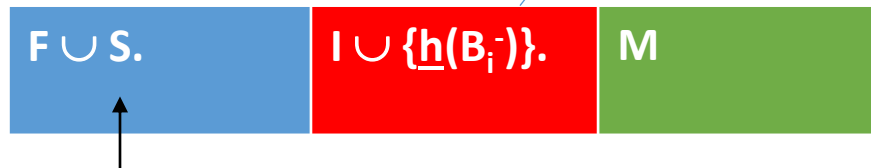
Version non-skolemisée

Implémentation:  $q(\underline{Y_{127}}) = q(Y) [Y \rightarrow Y_{127}]$

# Chases monotones: jusqu'ici tout va bien

Un chase est **monotone** si  $\forall i, F_i \subseteq F_{i+1}$

Tous les chases « habituellement » utilisés sont monotones à part le core-chase.



La monotonie intervient ici !!!

**Conclusion:** un chase monotone peut être utilisé syntaxiquement dans ASPERIX, on garde l'équivalence avec les bonnes dérivations.

**Bon attachement:** ici, tous les termes de I sont:

- Soit des constantes de F
- Soit des variables attachées à des termes de F
- Soit des variables « libres »

**Conservation du bon attachement:** les contraintes de I sont bien attachées dans F, donc bien attachées dans  $F \cup S$ . Les contraintes « rajoutées » sont obtenues à partir d'un homomorphisme h dans F. Les termes de I sont donc:

- Soit des constantes de F (image par h)
- Soit des variables attachées à des termes de F (image par h)
- Soit des variables « libres »

Les contraintes du nouveau sommet sont donc bien attachées dans F, et donc dans  $F \cup S$ .

# Le cas douloureux du core chase (1)

$p(a, X1), q(X1), p(a, X2), r(X2).$   $\text{not } s(\underline{X1}).$

$p(a, X1), q(X1), p(a, X2), r(X2).$   $\text{not } s(\underline{X2}).$   
 $q(X2).$

OUI MAIS...

$p(a, X1), q(X1), p(a, X2), r(X2).$   $\text{not } r(\underline{X1}).$

$p(a, X1), q(X1), p(a, X2), r(X2).$   $\text{not } r(\underline{X2}).$   
 $q(X2).$



$p(X, Y), r(Y) \rightarrow q(Y)$

La contrainte est maintenant attachée à une variable qui n'existe plus. Que faut-il faire?

- **Supprimer la contrainte** ? (berk, on perd de l'info...)
- La **renommer** ? Puisque X1 a été simplifié en X2, on renomme la contrainte  $\text{not } s(\underline{X2}).$

$p(X, Y), r(Y) \rightarrow q(Y)$

Ici nous avons procédé au renommage, mais ça a créé une violation du OUT. Pourtant, sans la simplification, ça n'aurait pas posé de problème...



# Le cas douloureux du core chase (2)

## **Propositions d'ébauches de solutions...**

V1/ Ne simplifier que si la simplification n'apporte pas de contradiction.

V2/ Considérer la contrainte comme une information. Dans ce cas, lors du renommage consécutif à la simplification, vérifier que la contrainte renommée est déjà présente (déductible des contraintes existantes).

Dans tous les cas, la contradiction (V1) ou la preuve (V2) peuvent apparaître plus tard dans la dérivation... Cela veut-il dire, par exemple pour la V2, qu'il faut prévoir des branching simplification / non simplification, et un champ MUST BE FALSE ?

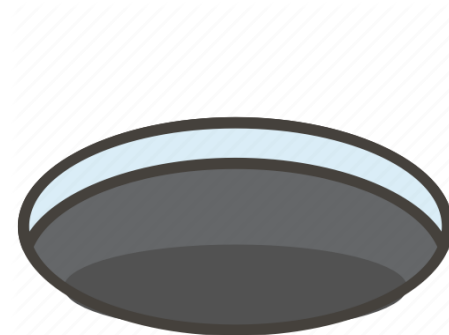
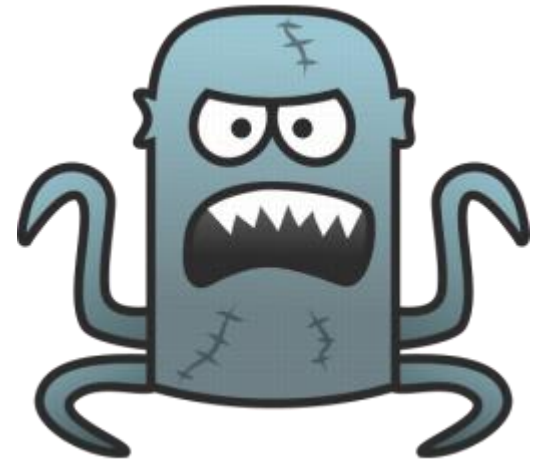
Ce mécanisme devra être résistant à l'ordre d'application des règles, caractéristique fondamentale d'ASPERIX.

Et, tant qu'on en parle, les modèles stables dépendent-ils de l'ordre d'application des règles au cours du restricted chase?

Et tout cela est-il bien utile?



Hunt for the Wumpus (1972)



# Le Monde du Wumpus





# WW: le moteur d'environnement

## %% INSTANCE DATA

```
#const maxarrow = 1.  
#const nbrows = 4.  
#const nbcolumns = 4.  
#const maxtime = 12. %% avant 15
```

```
time(1..maxtime).  
column(1..nbcolumns).  
row(1..nbrows).  
arrows(0..maxarrow).  
hasarrows(maxarrow, 0).  
livewumpus(0).  
livetreasure(0).
```

```
wumpus(1, 3).  
pit(3, 1).  
pit(3, 3).  
pit(4, 4).  
treasure(2, 3).  
hero(1, 1, 0).
```

## %% ENVIRONMENT

% connectivity

```
zone(X, Y) :- column(X), row(Y).  
north(X, Y, Z, T) :- zone(X, Y), zone(Z, T), X = Z, Y = T+1.  
south(Z, T, X, Y) :- north(X, Y, Z, T).  
east(X, Y, Z, T) :- zone(X, Y), zone(Z, T), X = Z+1, Y = T.  
west(Z, T, X, Y) :- east(X, Y, Z, T).  
connected(X, Y, Z, T) :- north(X, Y, Z, T).  
connected(X, Y, Z, T) :- south(X, Y, Z, T).  
connected(X, Y, Z, T) :- east(X, Y, Z, T).  
connected(X, Y, Z, T) :- west(X, Y, Z, T).
```

% actions

$\text{:- action(A1, D1, N), action(A2, D2, N), A1} \neq \text{A2.}$

$\text{:- action(A1, D1, N), action(A2, D2, N), D1} \neq \text{D2.}$

$\text{hero(X, Y, N+1) :- action(shoot, D, N), time(N+1), hero(X, Y, N).}$

$\text{shoot(D, N+1) :- action(shoot, D, N), hasarrows(K, N), K} > 0, \text{time(N+1).}$

$\text{shooting(N) :- shoot(D, N).}$

$\text{hasarrows(K-1, N) :- shoot(D, N), hasarrows(K, N-1), arrows(K-1).}$

$\text{hasarrows(K, N+1) :- hasarrows(K, N), time(N+1), not shooting(N+1).}$

$\text{headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(north, N), X1} = \text{X2, Y1} < \text{Y2.}$

$\text{headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(south, N), X1} = \text{X2, Y1} > \text{Y2.}$

$\text{headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(west, N), X1} < \text{X2, Y1} = \text{Y2.}$

$\text{headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(east, N), X1} > \text{X2, Y1} = \text{Y2.}$

$\text{scream(N) :- headshot(N), livewumpus(N-1).}$

$\text{livewumpus(N) :- livewumpus(N-1), time(N), hero(X, Y, N), not headshot(N).}$

$\text{hero(X, Y+1, N+1) :- hero(X, Y, N), action(move, north, N), time(N+1), zone(X, Y+1).}$

$\text{hero(X, Y, N+1) :- hero(X, Y, N), action(move, north, N), time(N+1), not zone(X, Y+1).}$

$\text{hero(X, Y-1, N+1) :- hero(X, Y, N), action(move, south, N), time(N+1), zone(X, Y-1).}$

$\text{hero(X, Y, N+1) :- hero(X, Y, N), action(move, south, N), time(N+1), not zone(X, Y-1).}$

$\text{hero(X+1, Y, N+1) :- hero(X, Y, N), action(move, east, N), time(N+1), zone(X+1, Y).}$

$\text{hero(X, Y, N+1) :- hero(X, Y, N), action(move, east, N), time(N+1), not zone(X+1, Y).}$

$\text{hero(X-1, Y, N+1) :- hero(X, Y, N), action(move, west, N), time(N+1), zone(X-1, Y).}$

$\text{hero(X, Y, N+1) :- hero(X, Y, N), action(move, west, N), time(N+1), not zone(X-1, Y).}$

$\text{picktreasure(N+1) :- hero(X, Y, N), treasure(X, Y), action(pick, U, N), livetreasure(N), time(N+1).}$

$\text{hero(X, Y, N+1) :- hero(X, Y, N), action(pick, U, N), time(N+1).}$

$\text{livetreasure(N+1) :- hero(X, Y, N+1), livetreasure(N), not picktreasure(N).}$

% perceptions

stench(Z, T) :- wumpus(X, Y), zone(Z, T), connected(X, Y, Z, T).

breeze(Z, T) :- pit(X, Y), zone(Z, T), connected(X, Y, Z, T).

glitter(X, Y):- treasure(X, Y).

feelstench(X, Y, N) :- hero(X, Y, N), stench(X, Y).

feelbreeze(X, Y, N) :- hero(X, Y, N), breeze(X, Y).

feelglitter(X, Y, N) :- hero(X, Y, N), glitter(X, Y), livetreasure(N).

% death of a hero

deadhero(X, Y, N) :- hero(X, Y, N), wumpus(X, Y), livewumpus(N).

deadhero(X, Y, N) :- hero(X, Y, N), pit(X, Y).


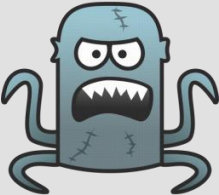




**deadhero(X, Y, maxtime) :- action(X, Y, maxtime).**

:- deadhero(X, Y, N).

% victory

victory(N) :- hero(X, Y, N), hero(X, Y, 0), not livetreasure(N).

# WW: test du moteur

4	stench		breeze	
3				breeze
2	stench		breeze	
1		breeze		breeze
	1	2	3	4

```
#include "wumpusengine.lp".
```

```
%% TESTING THE ENGINE
```

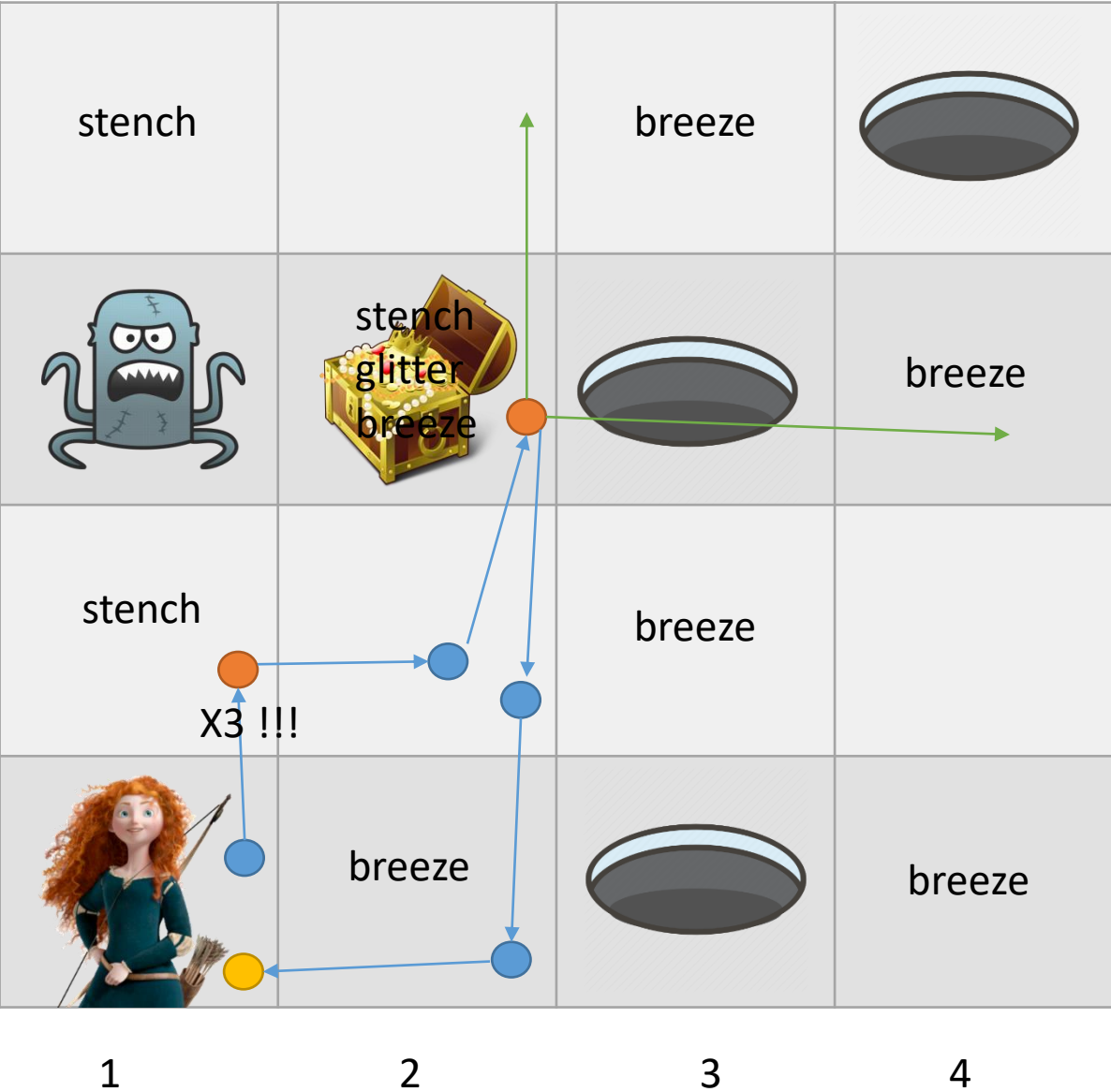
```
action(shoot, north, 0).
action(move, north, 1).
action(move, north, 2).
action(move, east, 3).
action(pick, treasure, 4).
action(move, west, 5).
action(move, south, 6).
action(move, south, 7).
```

```
#show hero/3.
#show action/3.
#show feelstench/3.
#show feelglitter/3.
#show feelbreeze/3.
#show scream/1.
#show victory/1.
```

```
clingo version 4.5.4
Reading from test4.lp
Solving...
Answer: 1
hero(1,1,0) action(shoot,north,0) action(move,north,1) action(move,north,2)
action(move,east,3) action(pick,treasure,4) action(move,west,5) action(move,south,6)
action(move,south,7) hero(1,1,1) hero(1,2,2) hero(1,3,3) hero(2,3,4) hero(2,3,5) hero(1,3,6)
hero(1,2,7) hero(1,1,8) scream(1) feelstench(1,2,2) feelstench(1,2,7) feelstench(2,3,4)
feelstench(2,3,5) feelbreeze(2,3,4) feelbreeze(2,3,5) feelglitter(2,3,4) feelglitter(2,3,5) victory(8)
SATISFIABLE
```

```
Models      : 1
Calls       : 1
Time        : 0.005s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time    : 0.000s
```

# WW: niveau 0



```
#include "wumpusengine.lp".
```

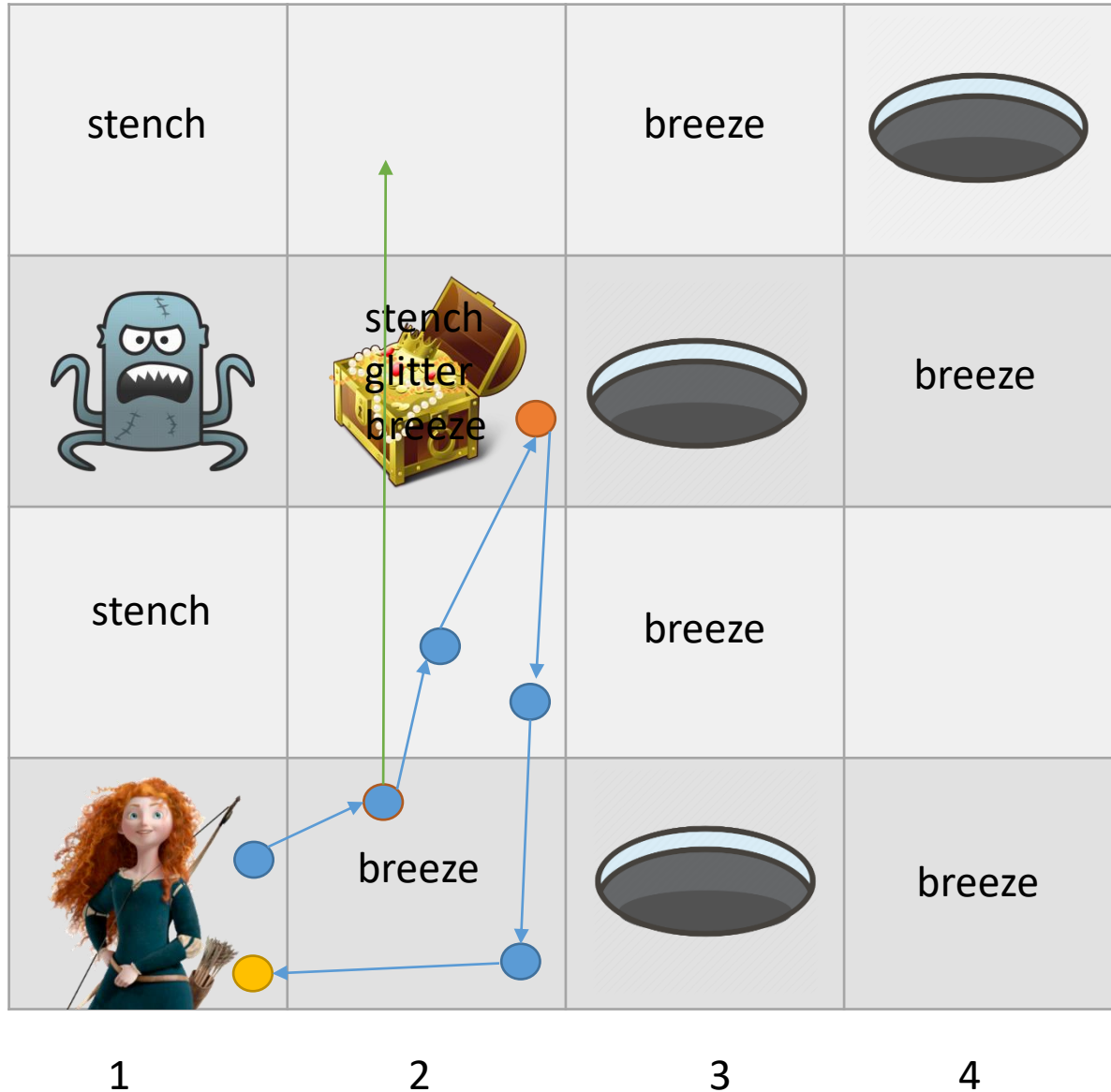
```
possaction(shoot; pick; move). %% macro utile  
possdirection(north; south; east; west).  
chose(0).
```

```
action(A, D, N) :- possaction(A), possdirection(D), chose(N),  
                    not otheraction(A, D, N).  
otheraction(A, C, N) :- possaction(A), possdirection(C),  
                        action(B, D, N), A != B.  
otheraction(A, C, N) :- possaction(A), possdirection(C),  
                        action(B, D, N), C != D.
```

```
chose(N+1) :- action(A, D, N), time(N+1), not victory(N+1).
```

```
clingo -n 1000000 wumpustest1.lp > file.txt  
Answer: 1000000  
hero(1,1,0) action(move,north,0) action(pick,west,1) action(pick,north,2) action(pick,west,3)  
action(move,east,4) action(move,north,5) action(pick,east,6) action(shoot,north,7)  
action(shoot,east,8) action(move,south,9) action(move,south,10) action(move,west,11)  
victory(12) hero(1,2,1) hero(1,2,2) hero(1,2,3) hero(1,2,4) hero(2,2,5) hero(2,3,6) hero(2,3,7)  
hero(2,3,8) hero(2,3,9) hero(2,2,10) hero(2,1,11) hero(1,1,12) feelstench(1,2,1) feelstench(1,2,2)  
feelstench(1,2,3) feelstench(1,2,4) feelstench(2,3,6) feelstench(2,3,7) feelstench(2,3,8)  
feelstench(2,3,9) feelbreeze(2,1,11) feelbreeze(2,3,6) feelbreeze(2,3,7) feelbreeze(2,3,8)  
feelbreeze(2,3,9) feelglitter(2,3,6) feelglitter(2,3,7)  
SATISFIABLE  
  
Models      : 1000000+  
Calls       : 1  
Time        : 115.062s (Solving: 115.05s 1st Model: 0.01s Unsat: 0.00s)  
CPU Time    : 112.969s
```

# WW: niveau 0+



```
#include "wumpusengine.lp".
```

```
possaction(shoot; pick; move). %% macro utile
possdirection(north; south; east; west).
chose(0).
```

```
action(A, D, N) :- possaction(A), possdirection(D), chose(N),
    not otheraction(A, D, N).
```

```
otheraction(A, C, N) :- possaction(A), possdirection(C),
    action(B, D, N), A != B.
```

```
otheraction(A, C, N) :- possaction(A), possdirection(C),
    action(B, D, N), C != D.
```

```
chose(N+1) :- action(A, D, N), time(N+1), not victory(N+1).
```

```
:- action(shoot, D, N), action(shoot, C, M), N < M.
```

```
:- action(pick, D, N), D != north.
```

```
:- action(pick, D, N), hero(X, Y, N), not feelglitter(X, Y, N).
```

```
clingo -n 1000000 wumpustest2.lp > file.txt
```

```
hero(1,1,0) action(move,east,0) action(shoot,north,1) action(move,north,2)
action(move,north,3) action(pick,north,4) action(move,south,5) action(move,south,6)
action(move,west,7) victory(8)
SATISFIABLE
```

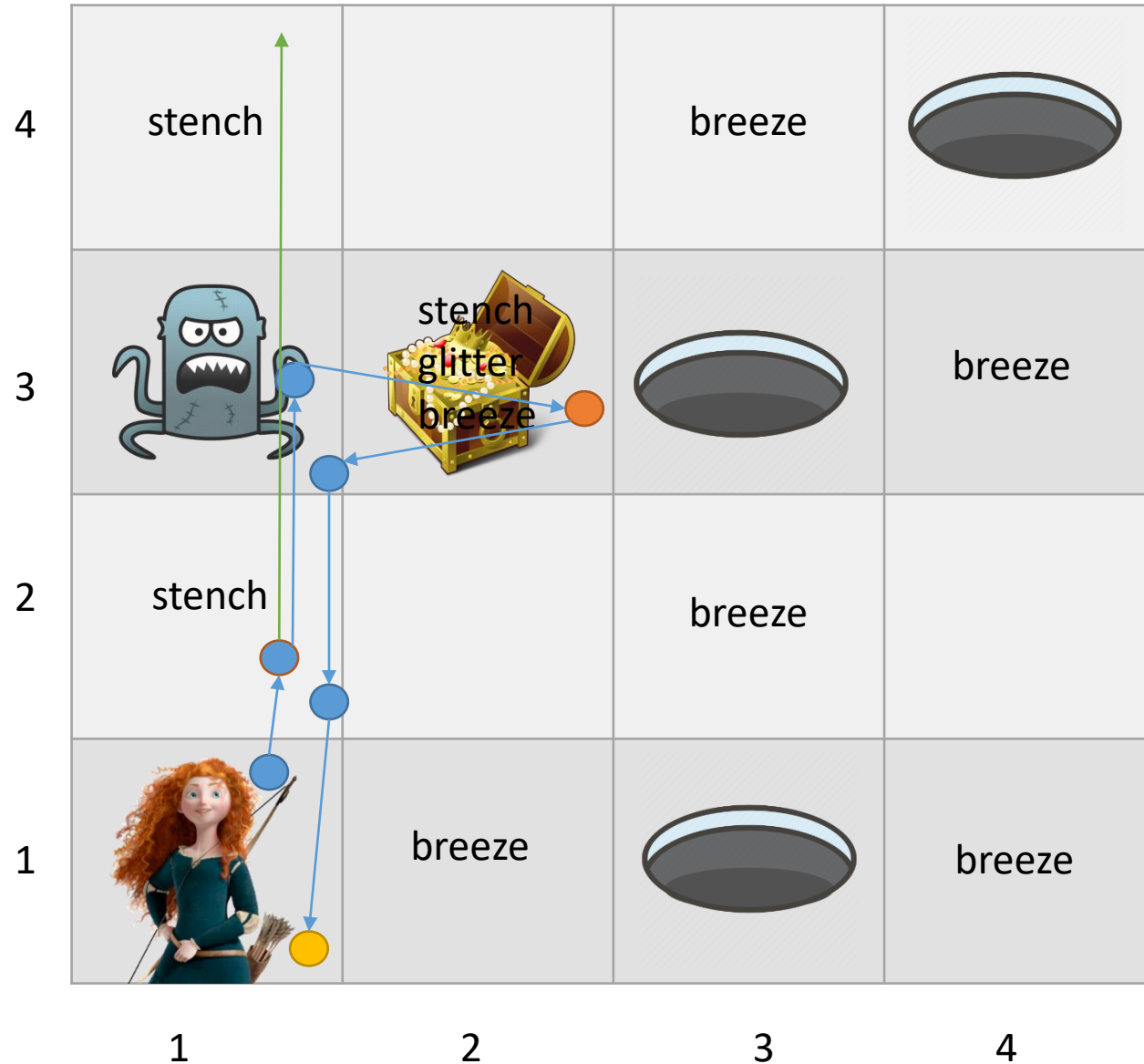
```
Models      : 132067
```

```
Calls       : 1
```

```
Time        : 14.786s (Solving: 14.77s 1st Model: 0.00s Unsat: 0.00s)
```

```
CPU Time    : 14.688s
```

# WW: niveau 0,5



%% on rajoute

```
pitsafe(Z, T, N) :- hero(X, Y, N), connected(X, Y, Z, T), not feelbreeze(X, Y, N).
pitsafe(X, Y, N+1) :- pitsafe(X, Y, N), time(N+1).
```

```
wumpussafe(Z, T, N) :-
```

```
    hero(X, Y, N), connected(X, Y, Z, T), not feelstench(X, Y, N).
```

```
wumpussafe(X, Y, N+1) :- wumpussafe(X, Y, N), time(N+1).
```

```
wumpussafe(X, Y, N) :- zone(X, Y), time(N), scream(M), M <= N.
```

```
safe(X, Y, N) :- wumpussafe(X, Y, N), pitsafe(X, Y, N).
```

```
:- action(move, north, N), hero(X, Y, N), not safe(X, Y+1, N).
```

```
:- action(move, south, N), hero(X, Y, N), not safe(X, Y-1, N).
```

```
:- action(move, east, N), hero(X, Y, N), not safe(X+1, Y, N).
```

```
:- action(move, west, N), hero(X, Y, N), not safe(X-1, Y, N).
```

**clingo -n 1000000 wumpustest3.lp > file.txt**

```
hero(1,1,0) action(move,north,0) action(shoot,north,1) action(move,north,2)
action(move,east,3) action(pick,north,4) action(move,west,5) action(move,south,6)
action(move,south,7) victory(8) hero(1,2,1) hero(1,2,2) hero(1,3,3) hero(2,3,4) hero(2,3,5)
hero(1,3,6) hero(1,2,7) hero(1,1,8) scream(2) feelstench(1,2,1) feelstench(1,2,2)
feelstench(1,2,7) feelstench(2,3,4) feelstench(2,3,5) feelbreeze(2,3,4) feelbreeze(2,3,5)
feelglitter(2,3,4) feelglitter(2,3,5)
SATISFIABLE
```

Models : 1898

Calls : 1

Time : 0.235s (Solving: 0.22s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.234s

# Pour plus tard...

Le hero ne risque plus de mourir... Mais il n'a aucune stratégie assurant qu'il trouvera une solution, si elle existe.

A FAIRE: V2 du wumpus engine.

- La position ne doit plus être donnée
- Bump quand il cogne un mur