

Modéliser avec les modèles stables

HAI 9331 – Partie 3

Jean-François Baget – baget@lirmm.fr

2022

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 { $p(a)$
 $q(a, b)$



$r(a) ?$

G { $p(a)$
 $q(b, a)$



$r(a) ?$

Dans les épisodes précédents:
dérivations persistantes et complètes

$(R_1) \quad p(X), \text{ not } q(X) \rightarrow r(X)$

$(R_2) \quad p(X) \rightarrow q(X)$

$p(a)$

$p(a) \xrightarrow[\text{q(a) absent}]{\text{application } R_1,} p(a), r(a) \xrightarrow[\text{q(a) apparaît}]{\text{application } R_2,} p(a), r(a), q(a)$

On voudrait des applications
persistantes dans la dérivation: toute
application devrait rester applicable.

$p(a) \xrightarrow[\text{q(a) absent}]{\text{application } R_1,} p(a), r(a)$
 R_2 est applicable,
mais on ferme les yeux

On voudrait des dérivations
complètes.

$p(a) \xrightarrow{\text{application } R_2} p(a), r(a)$
 R_1 n'est plus applicable,
on a bien fini

Bonne dérivation : persistante et
complète.

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

Dans les épisodes précédents: « 3-coloration »

$$v(X, Y) \rightarrow v(Y, X)$$

$$s(X), \text{ not } c(X, b), \text{ not } c(X, y) \rightarrow c(X, r)$$

$$s(X), \text{ not } c(X, b), \text{ not } c(X, r) \rightarrow c(X, y)$$

$$s(X), \text{ not } c(X, r), \text{ not } c(X, y) \rightarrow c(X, b)$$

$$v(X, Y), c(X, Z), c(Y, Z) \rightarrow \perp$$

Correction: « k-coloration »

$\text{voisin}(X, Y) \rightarrow \text{voisin}(Y, X)$

$\text{sommet}(X), \text{not } \text{autrecoleur}(X, C) \rightarrow \text{couleur}(X, C)$

$\text{sommet}(X), \text{couleur}(C), \text{couleur}(X, D), C \neq D \rightarrow \text{autrecoleur}(X, C)$

$\text{voisin}(X, Y), \text{couleur}(X, Z), \text{couleur}(Y, Z) \rightarrow \perp$

Avec clingo

<https://sourceforge.net/projects/potassco/files/clingo/4.5.4/>

```
% ==== DATA
```

```
sommet(1..6).
```

```
voisin(1, 2). voisin(1, 6). voisin(2, 3). voisin(2, 6).
```

```
voisin(3, 4). voisin(3, 5). voisin(3, 6). voisin(4, 5).
```

```
couleur(bleu). couleur(rouge). couleur(jaune).
```

```
% ===== PROGRAM
```

```
voisin(Y, X) :- voisin(X, Y).
```

```
colore(X, C) :- sommet(X), couleur(C), not autrecouleur(X, C).
```

```
autrecouleur(X, C) :- sommet(X), couleur(C), colore(X, D), C != D.
```

```
err() :- voisin(X, Y), colore(X, C), colore(Y, C), not err().
```

```
C:\Users\baget\clingo-4.5.4-win64> ./clingo -n 1000 testcol.lp > testcol.txt
```

```
clingo version 4.5.4
```

```
Reading from testcol.lp
```

```
Solving...
```

```
Answer: 1
```

```
voisin(1,2) voisin(1,6) voisin(2,3) voisin(2,6) voisin(3,4) voisin(3,5) voisin(3,6) voisin(4,5)
couleur(bleu) couleur(rouge) couleur(jaune) sommet(1) sommet(2) sommet(3) sommet(4)
sommet(5) sommet(6) voisin(2,1) voisin(6,1) voisin(3,2) voisin(6,2) voisin(4,3) voisin(5,3) voisin(6,3)
voisin(5,4) colore(1,bleu) autrecouleur(2,bleu) colore(3,bleu) autrecouleur(4,bleu)
autrecouleur(5,bleu) autrecouleur(6,bleu) autrecouleur(1,rouge) autrecouleur(2,rouge)
autrecouleur(3,rouge) autrecouleur(4,rouge) colore(5,rouge) colore(6,rouge) autrecouleur(1,jaune)
colore(2,jaune) autrecouleur(3,jaune) colore(4,jaune) autrecouleur(5,jaune) autrecouleur(6,jaune)
```

...

```
Answer: 12
```

```
voisin(1,2) voisin(1,6) voisin(2,3) voisin(2,6) voisin(3,4) voisin(3,5) voisin(3,6) voisin(4,5)
couleur(bleu) couleur(rouge) couleur(jaune) sommet(1) sommet(2) sommet(3) sommet(4)
sommet(5) sommet(6) voisin(2,1) voisin(6,1) voisin(3,2) voisin(6,2) voisin(4,3) voisin(5,3) voisin(6,3)
voisin(5,4) autrecouleur(1,bleu) colore(2,bleu) autrecouleur(3,bleu) autrecouleur(4,bleu)
colore(5,bleu) autrecouleur(6,bleu) autrecouleur(1,rouge) autrecouleur(2,rouge)
autrecouleur(3,rouge) colore(4,rouge) autrecouleur(5,rouge) colore(6,rouge) colore(1,jaune)
autrecouleur(2,jaune) colore(3,jaune) autrecouleur(4,jaune) autrecouleur(5,jaune)
autrecouleur(6,jaune)
```

```
SATISFIABLE
```

```
Models      : 12
```

```
Calls       : 1
```

```
Time        : 0.004s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
```

```
CPU Time    : 0.000s
```



Sommet(1..6) veut dire sommet(1). Sommet(2).
En ASP, la virgule en tete (et faits) veut dire OU.



Le fermier, le loup, la chèvre et le chou

FLCC: les règles du jeu



conditions initiales

berge(est), berge(ouest), oppose(est, ouest), oppose(ouest, est).
passager(loup), passager(chevre), passager(chou).
predateur(loup, chevre), predateur(chevre, chou).
position(fermier, ouest, 0).
position(loup, ouest, 0).
position(chevre, ouest, 0).
Position(chou, ouest, 0).

on gagne si tous les passagers sont de l'autre coté

position(loup, est, N), position(chevre, est, N),
position(chou, est, N) → gagne(N).

on perd si le fermier ne protège pas la proie du prédateur

position(P, B, N), position(C, B, N), predateur(P, C),
position(fermier, A, N), oppose(B, A) → perdu(N).

on ne veut pas des modeles stables ou on a perdu

perdu(N) → ⊥.

FLCC: choix de l'action



choisir le passager a faire traverser (ou personne)

soit le fermier fait traverser un passager, soit il est seul

$\text{position}(\text{fermier}, B, N), \text{not } \text{traversedeux}(N), \text{not } \text{gagne}(N) \rightarrow \text{traverseseul}(N).$

$\text{position}(\text{fermier}, B, N), \text{passager}(P), \text{position}(P, B, N), \text{not } \text{traverseseul}(N), \text{not } \text{gagne}(N) \rightarrow \text{traversedeux}(N).$

si il fait traverser quelqu'un, alors il faut choisir

$\text{traversedeux}(N), \text{position}(\text{fermier}, B, N), \text{passager}(X), \text{position}(X, B, N), \text{not } \text{autretransport}(X, N) \rightarrow \text{transporte}(X, N).$

$\text{position}(X, B, N), \text{transporte}(Y, N), X \neq Y \rightarrow \text{autretransport}(X, N).$

FLCC: mise a jour

mise a jour des informations a l'étape N + 1

pour le fermier

$\text{position}(\text{fermier}, B, N), \text{oppose}(B, C), \text{not } \text{gagne}(N) \rightarrow \text{position}(\text{fermier}, C, N+1).$

pour le transporté

$\text{transporte}(X, N), \text{position}(X, B, N), \text{oppose}(B, C) \rightarrow \text{position}(X, C, N+1).$

pour les autres

$\text{position}(X, B, N), \text{passager}(X), \text{not } \text{transporte}(X, N), \text{not } \text{gagne}(N) \rightarrow \text{position}(X, B, N+1).$



FLCC: branches finies

empêcher la même configuration à des étapes différentes



$\text{position}(X, B, N), \text{position}(X, C, N), \text{oppose}(B, C), N < M \rightarrow \text{change}(N, M).$

$\text{position}(X, B, N), \text{position}(X, B, M), N < M, \text{not } \text{change}(N, M) \rightarrow \text{redondant}().$

$\text{redondant}() \rightarrow \perp.$

Que se passerait-il si on entrait ce programme sous ASPERIX?

Que se passe-t-il si on entre ce programme sous CLINGO?

FLCC: adaptation pour clingo

temps(0..10).

berge(est). berge(ouest). oppose(est, ouest). oppose(ouest, est).
passager(loup). passager(chevre). passager(chou).
position(loup, ouest, 0). position(chevre, ouest, 0). position(chou, ouest, 0).
position(fermier, ouest, 0).

predateur(loup, chevre). predateur(chevre, chou).

gagne(N) :- position(loup, est, N), position(chevre, est, N), position(chou, est, N).

perdu(N) :- position(X, B, N), position(Y, B, N), predateur(X, Y), position(fermier, C, N), oppose(B, C).
paux() :- perdu(N), not paux().

traverseseul(N) :- position(fermier, B, N), not traversedeux(N), not gagne(N).
traversedeux(N) :- position(fermier, B, N), passager(Y), position(Y, B, N), not traverseseul(N), not gagne(N).

transporte(X, N) :- traversedeux(N), position(X, B, N), position(fermier, B, N), passager(X), not autretransport(X, N).
autretransport(X, N) :- position(X, B, N), transporte(Y, N), X != Y.

position(X, C, N+1) :- transporte(X, N), position(X, B, N), oppose(B, C), temps(N+1).
position(X, B, N+1) :- position(X, B, N), passager(X), not transporte(X, N), not gagne(N), temps(N+1).
position(fermier, C, N+1) :- position(fermier, B, N), oppose(B, C), not gagne(N), temps(N+1).

change(N, M) :- position(X, B, N), position(X, C, M), oppose(B, C), N < M.
redondant() :- position(X, B, N), position(X, B, M), N < M, not change(N, M).
raux() :- redondant(), not raux().

clingo version 4.5.4

Reading from test.lp

Solving...

Answer: 1

berge(est) berge(ouest) oppose(est,ouest) oppose(ouest,est) passager(loup) passager(chevre) passager(chou) position(loup,ouest,0) position(chevre,ouest,0) position(chou,ouest,0)
position(fermier,ouest,0) predateur(loup,chevre) predateur(chevre,chou) temps(0) temps(1) temps(2) temps(3) temps(4) temps(5) temps(6) temps(7) temps(8) temps(9) temps(10)
traversedeux(0) position(fermier,est,1) position(loup,ouest,1) **transporte(chevre,0)** position(chou,ouest,1) autretransport(loup,0) autretransport(chou,0) position(loup,ouest,2)
position(chou,ouest,2) position(fermier,ouest,2) position(chevre,est,1) traversedeux(2) position(loup,ouest,3) **transporte(chou,2)** position(fermier,est,3) **traverseseul(1)** position(chevre,est,2)
position(loup,ouest,4) position(chevre,ouest,4) **transporte(chevre,3)** position(fermier,ouest,4) autretransport(loup,2) autretransport(chevre,2) position(chevre,est,3) position(chou,est,3)
traversedeux(4) **transporte(loup,4)** position(chevre,ouest,5) position(fermier,est,5) traversedeux(3) position(chou,est,4) position(chevre,ouest,6) position(fermier,ouest,6)
autretransport(chevre,4) autretransport(chou,4) traversedeux(6) position(loup,est,5) position(chou,est,5) **transporte(chevre,6)** position(fermier,est,7) autretransport(loup,3)
autretransport(chou,3) autretransport(loup,6) autretransport(chou,6) **traverseseul(5)** position(loup,est,6) position(chou,est,6) gagne(7) **position(loup,est,7)** **position(chevre,est,7)**
position(chou,est,7) autretransport(fermier,0) autretransport(fermier,2) autretransport(fermier,3) autretransport(fermier,4) autretransport(fermier,6) change(1,2) change(1,3) change(2,3)
change(1,4) change(2,4) change(3,4) change(1,5) change(2,5) change(3,5) change(4,5) change(1,6) change(2,6) change(3,6) change(4,6) change(5,6) change(1,7) change(2,7) change(3,7)
change(4,7) change(5,7) change(6,7) change(0,1) change(0,3) change(0,2) change(0,5) change(0,4) change(0,7) change(0,6)

Answer: 2

berge(est) berge(ouest) oppose(est,ouest) oppose(ouest,est) passager(loup) passager(chevre) passager(chou) position(loup,ouest,0) position(chevre,ouest,0) position(chou,ouest,0)
position(fermier,ouest,0) predateur(loup,chevre) predateur(chevre,chou) temps(0) temps(1) temps(2) temps(3) temps(4) temps(5) temps(6) temps(7) temps(8) temps(9) temps(10)
traversedeux(0) position(fermier,est,1) position(loup,ouest,1) **transporte(chevre,0)** position(chou,ouest,1) autretransport(loup,0) autretransport(chou,0) position(loup,ouest,2)
position(chou,ouest,2) position(fermier,ouest,2) position(chevre,est,1) traversedeux(2) **transporte(loup,2)** position(chou,ouest,3) position(fermier,est,3) **traverseseul(1)** position(chevre,est,2)
position(chevre,ouest,4) **transporte(chevre,3)** position(chou,ouest,4) position(fermier,ouest,4) autretransport(chevre,2) autretransport(chou,2) position(loup,est,3) position(chevre,est,3)
traversedeux(4) position(chevre,ouest,5) **transporte(chou,4)** position(fermier,est,5) traversedeux(3) position(loup,est,4) position(chevre,ouest,6) position(fermier,ouest,6)
autretransport(loup,4) autretransport(chevre,4) traversedeux(6) position(loup,est,5) position(chou,est,5) **transporte(chevre,6)** position(fermier,est,7) autretransport(loup,3)
autretransport(chou,3) autretransport(loup,6) autretransport(chou,6) **traverseseul(5)** position(loup,est,6) position(chou,est,6) gagne(7) **position(loup,est,7)** **position(chevre,est,7)**
position(chou,est,7) autretransport(fermier,0) autretransport(fermier,2) autretransport(fermier,3) autretransport(fermier,4) autretransport(fermier,6) change(1,2) change(1,3) change(2,3)
change(1,4) change(2,4) change(3,4) change(1,5) change(2,5) change(3,5) change(4,5) change(1,6) change(2,6) change(3,6) change(4,6) change(5,6) change(1,7) change(2,7) change(3,7)
change(4,7) change(5,7) change(6,7) change(0,1) change(0,3) change(0,2) change(0,5) change(0,4) change(0,7) change(0,6)

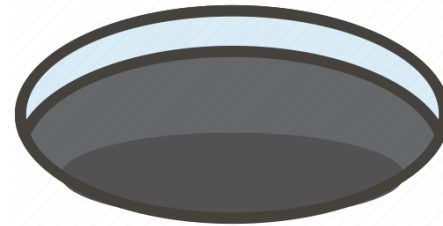
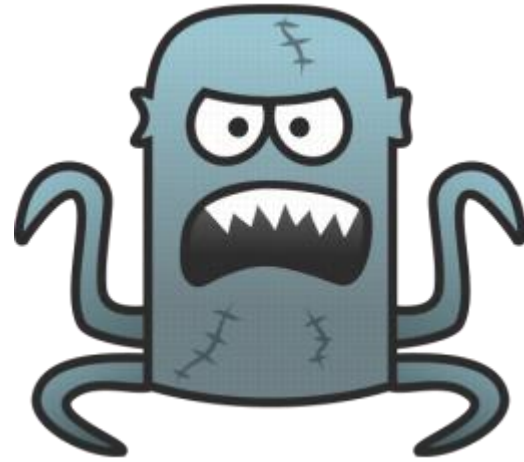
SATISFIABLE

Models : 2

Calls : 1

Time : 0.010s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)

CPU Time : 0.016s



Le Monde du Wumpus


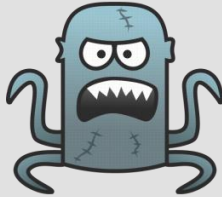






WW: le problème

Problem Statement:

*The Wumpus world is a cave with 16 rooms (4×4). Each room is connected to others through walkways (no rooms are connected diagonally). The knowledge-based agent starts from Room[1, 1]. The cave has – some **pits**, a **treasure** and a beast named **Wumpus**. The Wumpus can not move but eats the one who enters its room. If the agent enters the pit, it gets stuck there. The goal of the agent is to take the treasure and come out of the cave. The agent is rewarded, when the goal conditions are met. The agent is penalized, when it falls into a pit or being eaten by the Wumpus.*

Some elements support the agent to explore the cave, like -The wumpus's adjacent rooms are stenchy. -The agent is given one arrow which it can use to kill the wumpus when facing it (Wumpus screams when it is killed). – The adjacent rooms of the room with pits are filled with breeze. -The treasure room is always glittery.

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

WW: Principe de codage



action(A, M, N).

Règles d'environnement

wumpus(1, 3).
treasure(2, 3).



connected(X, Y, Z, T).

Règles de l'agent

safe(X, Y, N).



position(X, Y, N).
feelstench(X, Y, N).

WW: le moteur d'environnement

%% INSTANCE DATA

```
#const maxarrow = 1.  
#const nbrows = 4.  
#const nbcolumns = 4.  
#const maxtime = 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

:- action(A1, D1, N), action(A2, D2, N), A1 != A2.

:- action(A1, D1, N), action(A2, D2, N), D1 != D2.

hero(X, Y, N+1) :- action(shoot, D, N), time(N+1), hero(X, Y, N).

shoot(D, N+1) :- action(shoot, D, N), hasarrows(K, N), K > 0, time(N+1).

shooting(N) :- shoot(D, N).

hasarrows(K-1, N) :- shoot(D, N), hasarrows(K, N-1), arrows(K-1).

hasarrows(K, N+1) :- hasarrows(K, N), time(N+1), not shooting(N+1).

headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(north, N), X1 = X2, Y1 < Y2.

headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(south, N), X1 = X2, Y1 > Y2.

headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(west, N), X1 < X2, Y1 = Y2.

headshot(N) :- hero(X1, Y1, N), wumpus(X2, Y2), shoot(east, N), X1 > X2, Y1 = Y2.

scream(N) :- headshot(N), livewumpus(N-1).

livewumpus(N) :- livewumpus(N-1), time(N), hero(X, Y, N), not headshot(N).

hero(X, Y+1, N+1) :- hero(X, Y, N), action(move, north, N), time(N+1), zone(X, Y+1).

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

hero(X, Y-1, N+1) :- hero(X, Y, N), action(move, south, N), time(N+1), zone(X, Y-1).

hero(X, Y, N+1) :- hero(X, Y, N), action(move, south, N), time(N+1), not zone(X, Y-1).

hero(X+1, Y, N+1) :- hero(X, Y, N), action(move, east, N), time(N+1), zone(X+1, Y).

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

hero(X-1, Y, N+1) :- hero(X, Y, N), action(move, west, N), time(N+1), zone(X-1, Y).

hero(X, Y, N+1) :- hero(X, Y, N), action(move, west, N), time(N+1), not zone(X-1, Y).

picktreasure(N+1) :- hero(X, Y, N), treasure(X, Y), action(pick, U, N), livetreasure(N), time(N+1).

hero(X, Y, N+1) :- hero(X, Y, N), action(pick, U, N), time(N+1).

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).


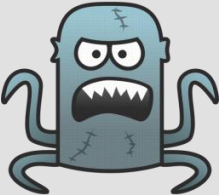




deadhero(X, Y, N) :- hero(X, Y, N), pit(X, Y).

:- deadhero(X, Y, N).

% victory

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

WW: test du moteur

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

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

```
%% TESTING THE ENGINE
```

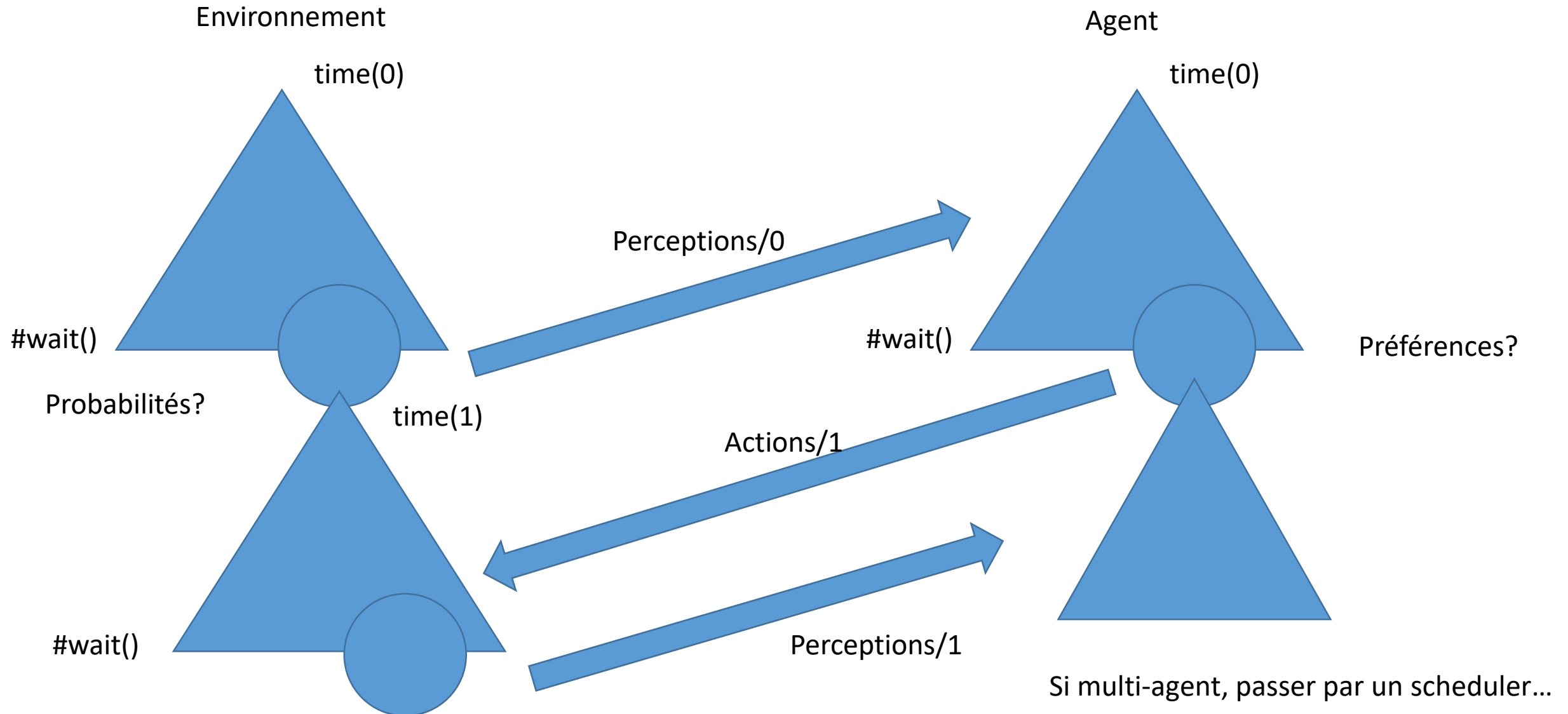
```
action(shoot, north, 0).  
action(move, north, 1).  
action(move, north, 2).  
action(move, east, 3).  
action(pick, treasure, 4).  
action(move, west, 5).  
action(move, south, 6).  
action(move, south, 7).
```

```
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
```

Idées pour une architecture multi-solver



Appel à participation en forme de conclusion

- FLCC: un exemple de résolution de problème
 - Même idée que coloration, on génère tout et on filtre
 - Introduction d'une gestion du temps discrétisé
 - Frame problem: ce qui n'a pas bougé reste vrai
- WW: l'agent n'a qu'une vue partielle de l'environnement
 - On a vu uniquement la gestion de l'environnement
 - Ebauche d'une architecture ASP/SMA (suite vieille discussion Madalina)
 - On n'a pas écrit le raisonnement de l'agent
- Appel à participation: la gestion de l'agent
 - Niveau 0: l'agent choisit au hasard ce qu'il va faire, et le solver vire ce qui ne marche pas
 - Niveau 1: l'agent ne réalise que des actions « safe »
 - Niveau 2: l'agent se donne des objectifs, explorer (par ex DFS), se positionne pour tuer Wumpus si exploration safe incomplete, re-explore si wumpus mort, si le trésor est trouvé revenir au départ. Ici, chaque modèle stable est une sol. optimale.