Reasoning about Actions

Monkey and Bananas in ASP

Ira A. Fulton Schools of Engineering
Arizona State University

Objectives

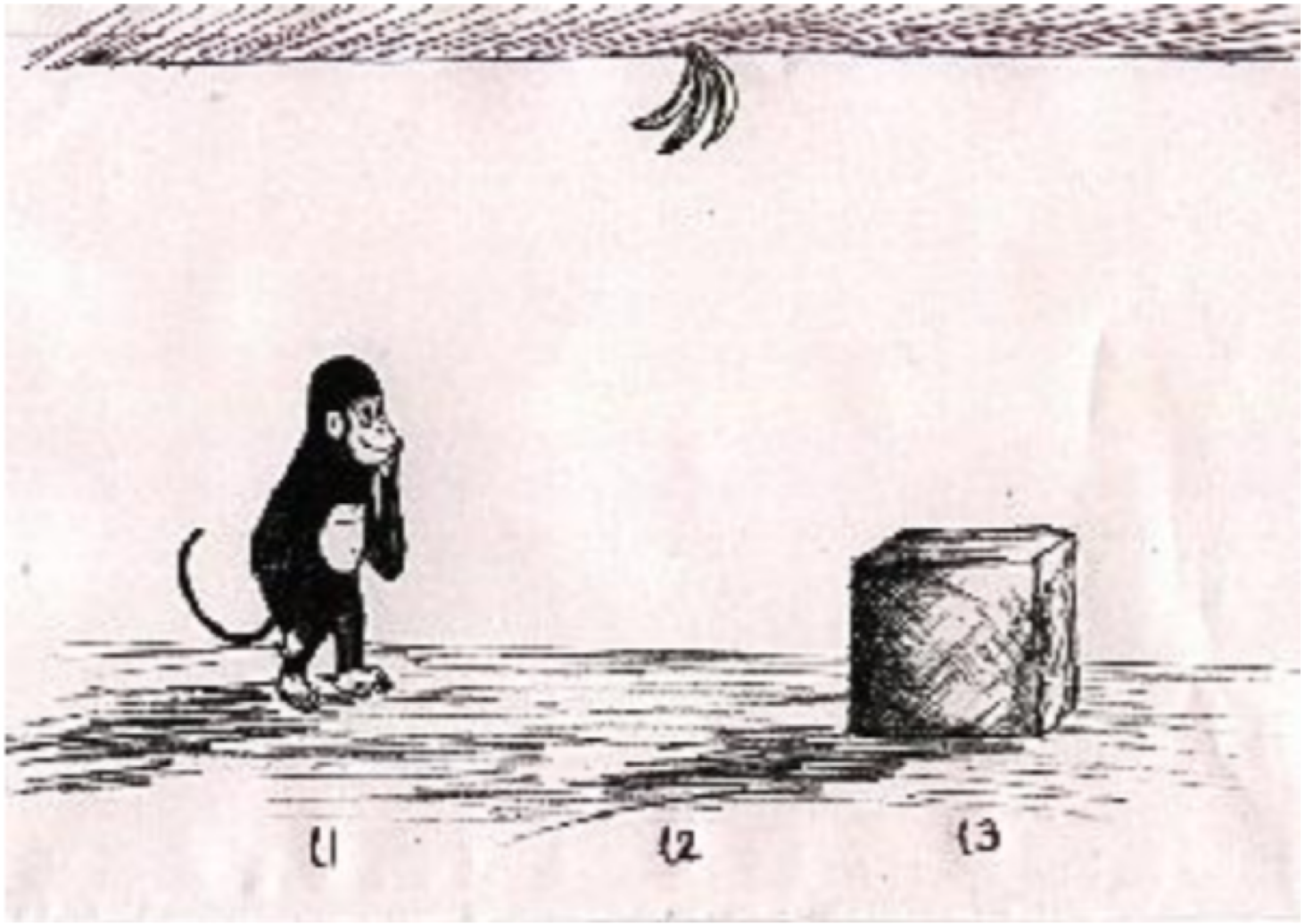
Objective

Model complex transition systems in answer set programming

Objective

Use KR tools for answering questions about prediction, postdiction, and planning problems

Monkey and Bananas



(Drawing by Esra Erdem)

States

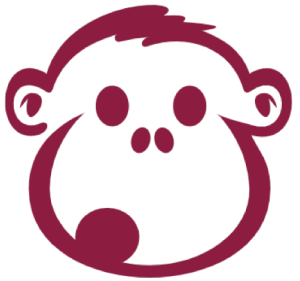
In Monkey and Bananas problem, a state can be described by specifying

- the current locations of the monkey, the bananas, and the box,
- whether or not the monkey is on the box, and
- whether or not the monkey has the bananas.

A state can be described by atoms

$$Loc(x, y)\ (x \in \{Monkey, Bananas, Box\},\quad y \in \{L1, L2, L3\}),$$

$$HasBananas, OnBox.$$

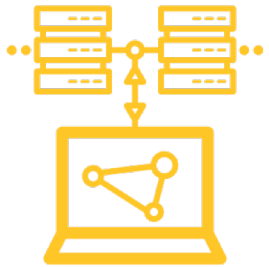


States, cont'd

The interpretations that satisfy

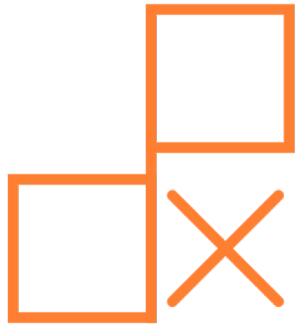
$$\exists y Loc(x, y),$$
$$\neg\exists y y_1 (Loc(x, y) \wedge Loc(x, y_1) \wedge y \neq y_1),$$
$$HasBananas \wedge Loc(Monkey, l) \rightarrow Loc(Bananas, l),$$
$$OnBox \wedge Loc(Monkey, l) \rightarrow Loc(Box, l)$$

represent the possible states of the system.



Representing Action Domain in ASP

- sort and object declaration
- state constraints
- effect and preconditions of actions
- `action constraints`
- `domain independent axioms`
  - `fluents are initially exogenous`
  - `uniqueness and existence of fluent values`
  - `actions are exogenous`
  - `Commonsense law of inertia`
- …



Representing MB in ASP (I)

```
%% sort and object declaration

boolean(t;f).
object(monkey;bananas;box).
location(l1;l2;l3).

%% state constraints

loc(bananas,L,T) :- hasBananas(t,T), loc(monkey,L,T).
loc(monkey,L,T) :- onBox(t,T), loc(box,L,T).
```

Representing MB in ASP (II)

```
%% effect and preconditions of actions

% walk
loc(monkey,L,T+1) :- walk(L,T).
:- walk(L,T), loc(monkey,L,T).
:- walk(L,T), onBox(t,T).

% pushBox
loc(box,L,T+1) :- pushBox(L,T).
loc(monkey,L,T+1) :- pushBox(L,T).
:- pushBox(L,T), loc(monkey,L,T).
:- pushBox(L,T), onBox(t,T).
:- pushBox(L,T), loc(monkey,L1,T), loc(box,L2,T), L1 != L2.
```

Representing MB in ASP (III)

```
% climbOn
onBox(t,T+1) :- climbOn(T).
:- climbOn(T), onBox(t,T).
:- climbOn(T), loc(monkey,L1,T), loc(box,L2,T), L1 != L2.

% climbOff
onBox(f,T+1) :- climbOff(T).
:- climbOff(T), onBox(f,T).

% graspBananas
hasBananas(t,T+1) :- graspBananas(T).
:- graspBananas(T), hasBananas(t,T).
:- graspBananas(T), onBox(f,T).
:- graspBananas(T), loc(monkey,L1,T), loc(bananas,L2,T),
   L1 != L2.
```

Representing MB in ASP (IV)

```
% disallow concurrent actions
:- walk(L,T), pushBox(L,T).
:- walk(L,T), climbOn(T).
:- pushBox(L,T), climbOn(T).
:- climbOff(T), graspBananas(T).
```

Representing MB in ASP (V)

```
%% domain independent axioms

% fluents are initially exogenous
1{hasBananas(BB,0):boolean(BB)}1.
1{onBox(BB,0):boolean(BB)}1.
1{loc(O,LL,0):location(LL)}1 :- object(O).

% uniqueness and existence of fluent values
:- not 1{loc(O,LL,T)}1, object(O), T = 1..m.
:- not 1{onBox(BB,T)}1, T = 1..m.
:- not 1{hasBananas(BB,T)}1, T = 1..m.
```

Representing MB in ASP (VI)

```
% actions are exogenous
{walk(L,T)} :- location(L), T = 0..m-1.
{pushBox(L,T)} :- location(L), T = 0..m-1.
{climbOn(T)} :- T = 0..m-1.
{climbOff(T)} :- T = 0..m-1.
{graspBananas(T)} :- T = 0..m-1.


% commonsense law of inertia
{hasBananas(B,T+1)} :- hasBananas(B,T), T=0..m-1.
{onBox(B,T+1)} :- onBox(B,T), T=0..m-1.
{loc(O,L,T+1)} :- loc(O,L,T), T=0..m-1.
```

Problems

How many states are there?

How many transitions are there?

State Constraints and Ramification Problem

If the monkey has the bananas, then there is a cause for the bananas to be at the same place where the monkey is:

```
loc(bananas,L,T) :- hasBananas(t,T), loc(monkey,L,T).
```

Q: Why is this rule redundant?

```
loc(bananas,L,T+1) :- hasBananas(t,T), walk(L,T).
```

In the presence of the first rule, the change in the location of the bananas is an indirect effect, or “ramification,” of walking (and of any other action that affects the location of the monkey)

State Constraints and Ramification Problem, Cont'd

If the monkey is on the box then there is a cause for the monkey to be at the same place where the box is:

```
loc(monkey,L,T) :- onBox(t,T), loc(box,L,T).
```

Question

Why don't we need the following rules for prohibiting concurrencies?

a) `:- walk(L,T), walk(L1,T), L!=L1.`

b) `:- walk(L,T), pushBox(L1,T), L!=L1.`

c) `:- walk(L,T), climbOff(T).`

d) `:- walk(L,T), graspBananas(T).`

e) `:- climbOn(T), climbOff(T).`

f) `:- climbOn(T), graspBananas(T).`

Planning Query

Find the shortest sequence of actions that would allow the monkey to have the bananas.

```
% File 'monkey-planning.lp'

% initial condition
:- not loc(monkey,l1,0).
:- not loc(bananas,l2,0).
:- not loc(box,l3,0).
:- not hasBananas(f,0).

% goal
:- not hasBananas(t,m).
```

```
$ clingo monkey.lp
monkey-planning.lp -c m=4

clingo version 5.3.0
Reading from monkey ...
Solving...
Answer: 1
pushBox(l2,1) walk(l3,0)
graspBananas(3)
climbOn(2) SATISFIABLE
```

Prediction Query

KB ⊨ F iff KB ∪ ¬F is unsat.

Initially, the monkey is at L1, the bananas are at L2, and the box is at L3. The monkey walks to L3 and then pushes the box to L2. Does it follow that in the resulting state the monkey, the bananas and the box are at the same location?

- This question can be formalized as follows: Determine if MB with m=2 conjoined with the negation of the conclusion has no stable models.

```
% File 'monkey-prediction.lp'
:- not loc(monkey,l1,0).
:- not loc(bananas,l2,0).
:- not loc(box,l3,0).
:- not walk(l3,0).
:- not pushBox(l2,1).

:- #count{LL:loc(monkey,LL,2),
          loc(bananas,LL,2),
          loc(box,LL,2)}=1
```

Postdiction Query

The monkey walked to location L3 and then pushed the box. Does it follow that the box was initially at L3?

```
% File 'monkey-postdiction.lp'

:- not walk(l3,0).
:- not 1{pushBox(LL,1):location(LL)}.

:- loc(box,l3,0).
```

Wrap-Up