Curs 11

Probleme de IA şi rezolvarea lor

Cele 5 cerinţe în modelarea unei probleme de IA

- **Pasul 1** Diferenţiază problema generală de instanţele ei
- **Pasul 2** Recunoaşte o stare şi apreciază dimensiunea spaţiului stărilor
- **Pasul 3** Găseşte cea mai adecvată reprezentare a stărilor
- **Pasul 4** Reprezintă tranziţiile dintre stări
- **Pasul 5** Alege o strategie de control

Probleme de dimensiuni mici (*toy problems*)

- **Problema 8-puzzle**

  Există o tablă 3x3 pe care se găsesc 8 piese pătrate. La un moment dat o singură piesă se poate mişca cu o poziţie, pe orizontală sau verticală, în limitele cadrului tablei, în locul rămas liber. Se dă o configuraţie iniţială şi una finală a tablei. Trebuie să se găsească secvenţa de mutări care să aducă piesele din configuraţia iniţială în cea finală.

Probleme de dimensiuni mici

- **Problema misionarilor şi canibalilor**

  3 misionari şi 3 canibali se află la marginea unui râu, cu scopul de a trece pe celălalt mal. Ei au la dispoziţie o barcă de două persoane. Dacă la un moment dat, pe un mal sau pe celălalt numărul canibalilor întrece pe cel al misionarilor, misionarii sînt în pericol de a fi mâncaţi de canibali. Problema constă în a afla cum pot trece râul cele 6 persoane în deplină siguranţă.

Probleme de dimensiuni mici

- **Problema generării frazelor în limbaj natural**

Se dispune de o gramatică (un set de simboluri numiţi terminali, un set de simboluri numiţi neterminali, o colecţie de reguli, fiecare arătând cum poate fi expandată o categorie compusă în subcompuşi şi un simbol de start). Se doreşte generarea unei exprimări corecte gramatical.

Probleme de dimensiuni mici

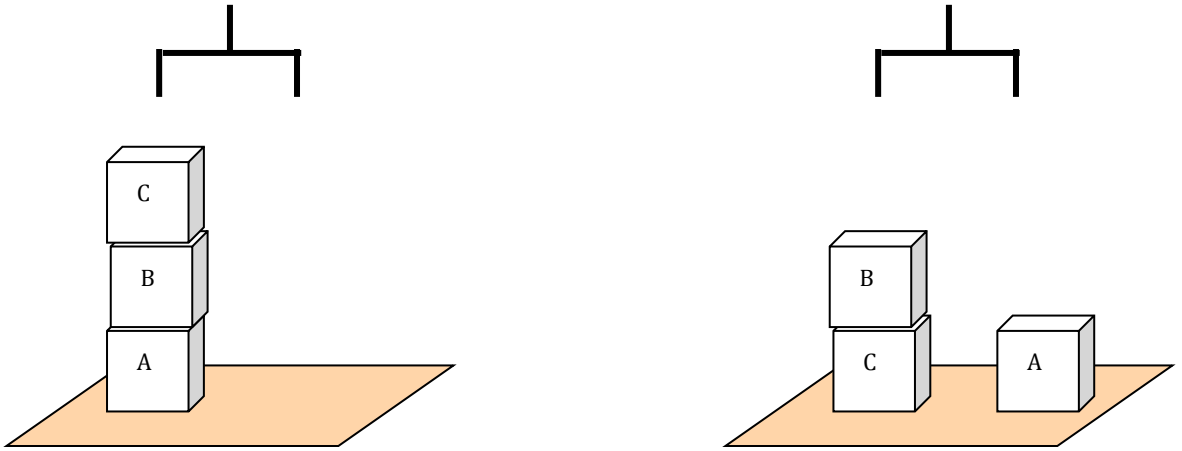
- **Problema maimuţei şi a bananei**

  O maimuţă este închisă într-o cuşcă în care se mai află o banană atârnată de tavan la o înălţime la care maimuţa nu poate ajunge şi, într-un colţ, o cutie. După un număr de încercări nereuşite de a apuca banana, maimuţa merge la cutie, o deplasează sub banană, se urcă pe cutie şi apucă banana. Se cere să se formalizeze maniera de raţionament a maimuţei.

Probleme de dimensiuni mici

- **Lumea cuburilor**

Un braț de robot trebuie să mute o stivă de cuburi, dintr-o aranjare inițială într-una finală.



starea iniţială

starea finală

Pasul 1

Problemă, instanţă de problemă

- 8-puzzle
  - formulată ca o instanţă de problemă
- Misionarii şi canibalii
  - formulată ca o instanţă de problemă
- Generarea frazelor
  - formulată ca o problemă
- Maimuţa şi banana
  - formulată ca o instanţă de problemă
- Lumea cuburilor
  - formulată ca o instanţă de problemă
- Alte exemple:
  - jocul de şah
  - condusul maşinii...

Pasul 1

Un exemplu de instanţă de problemă

- Generarea limbajului:

  G1 = {N1, T1, PROP, P1}, în care:

  N1 = {PROP, GN, GV, S, V} – o mulţime de neterminali cu semnificaţiile: propoziţie, grup nominal, grup verbal, substantiv şi verb;

  T1 = {*pisica*, *şoarecele*, *prinde*} – o mulţime de cuvinte;

  PROP1 = simbolul start al gramaticii, alegerea lui semnifică că ceea ce se doreşte să se obţină reprezintă propoziţii ale acestui mini-limbaj;

  P1 = {PROP := GN GV,

  GN := S,

  GV := V GN,

  S := *pisica*,

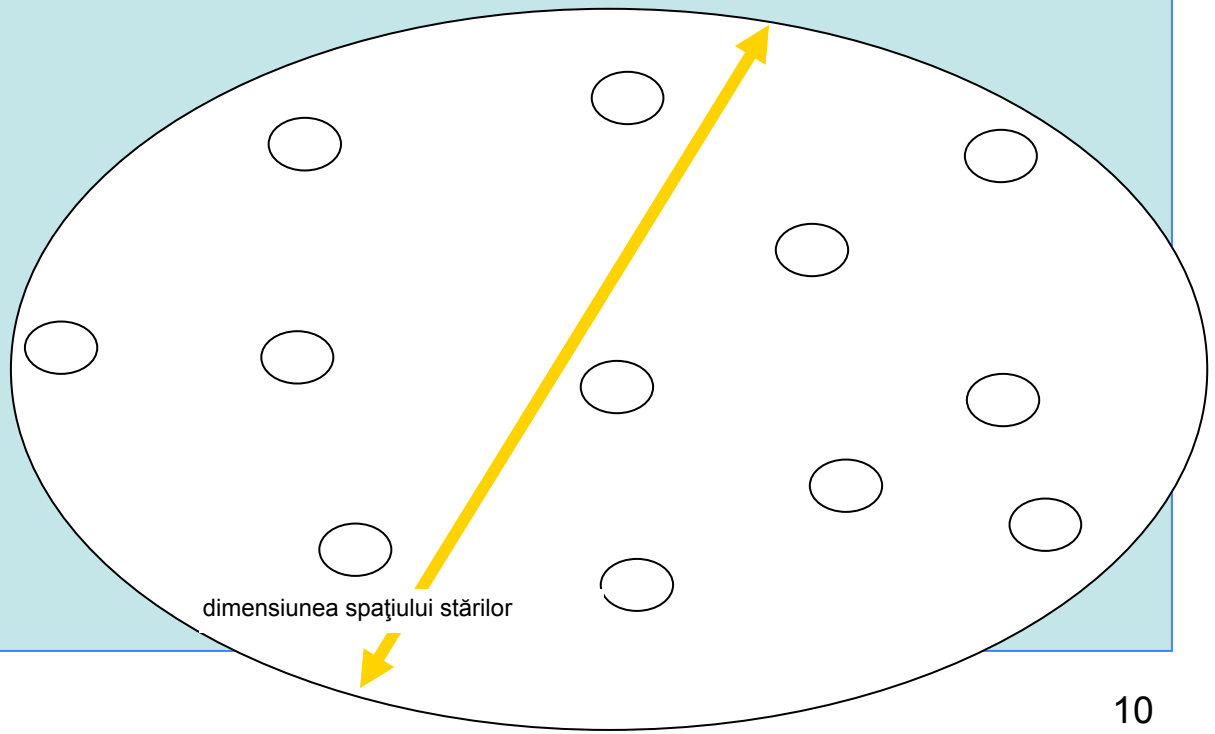
  S := *şoarecele*,

  V := *prinde*} – o listă de reguli de producție.

Pasul 2

Spaţiul problemei

- Stări, dimensiunea spaţiului



Pasul 2

Dimensiunea spaţiului stărilor

- Jocul de şah: 10^{120}

Pasul 2

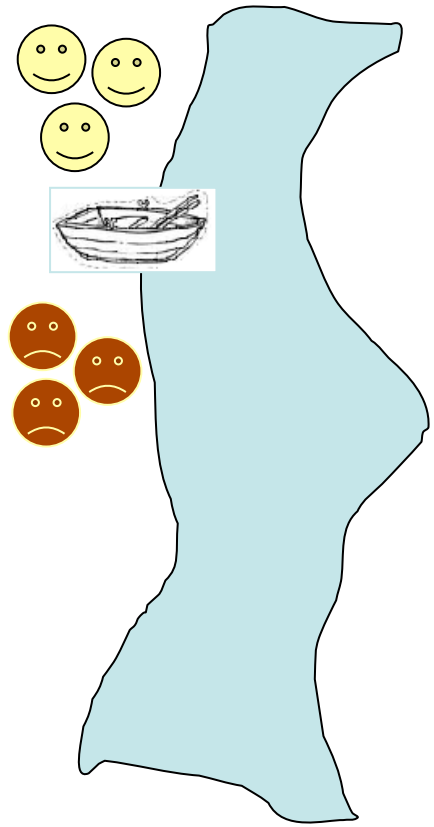
Dimensiunea spaţiului stărilor

- Jocul de şah: 10^{120}
- 8-puzzle: 9!

Pasul 2

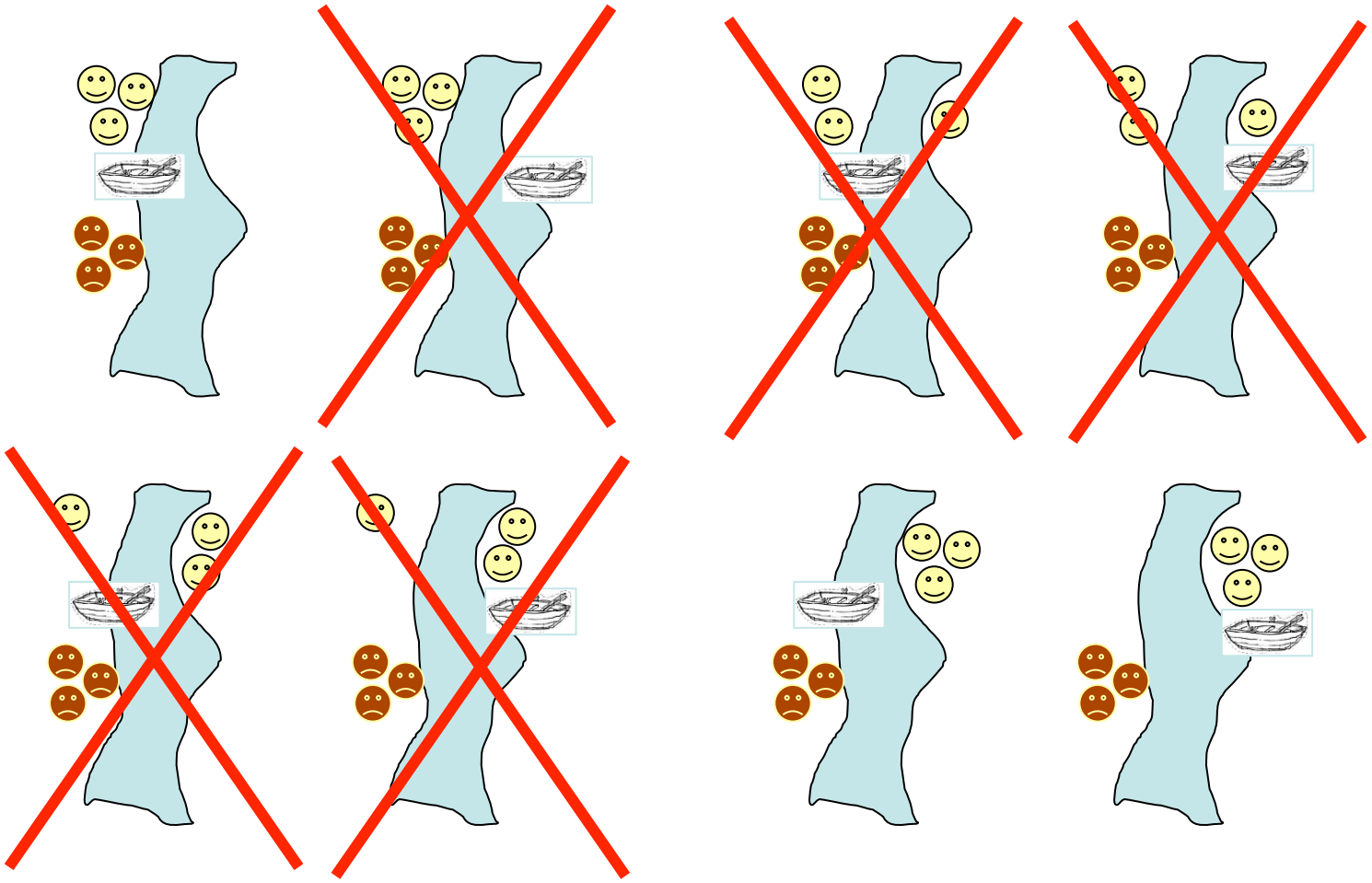
Dimensiunea spaţiului stărilor

- Jocul de şah: 10^{120}
- 8-puzzle: 9!
- misionari şi canibali:



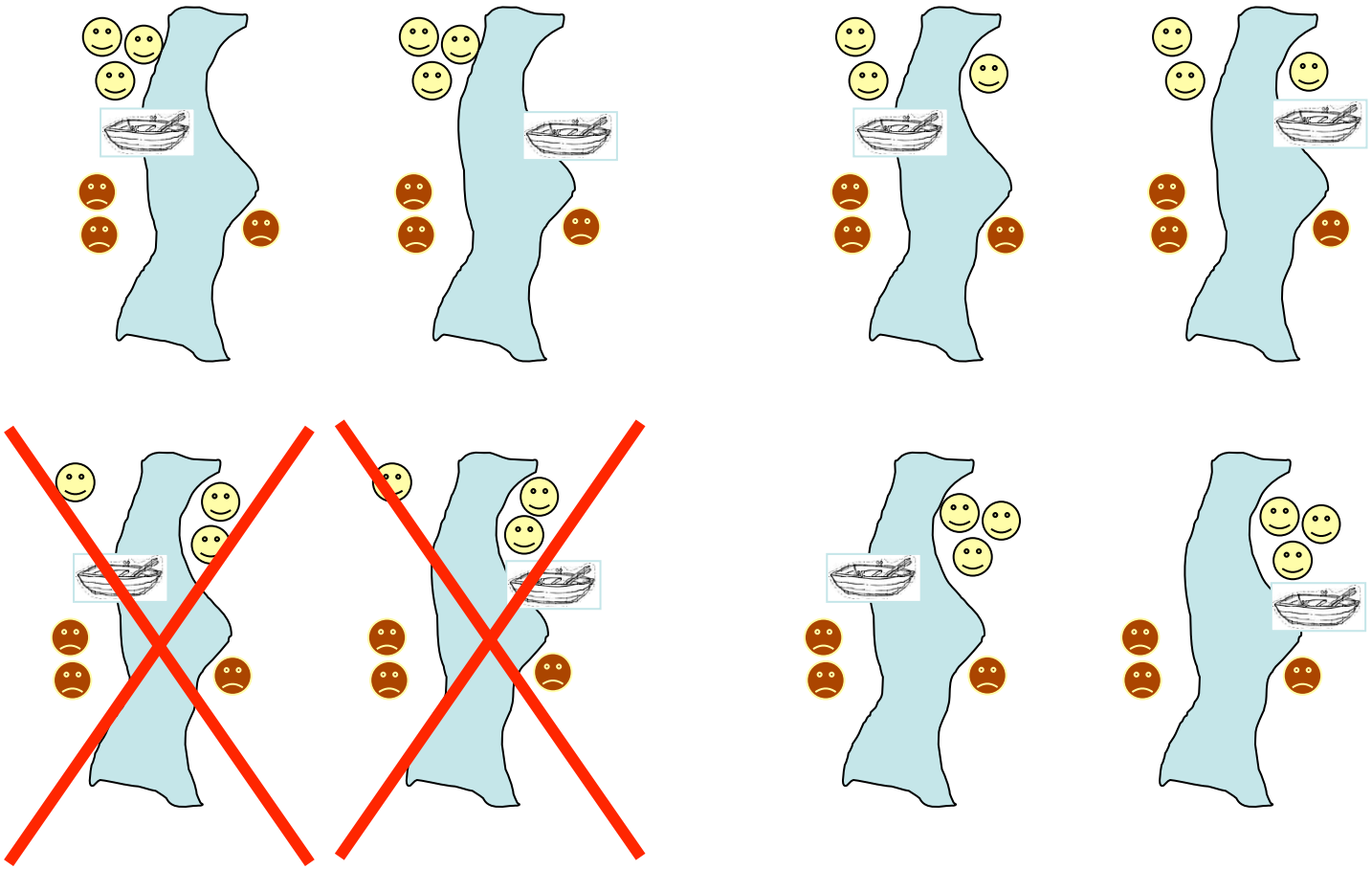
Pasul 2 Stări: misionari şi canibali

3 canibali în stânga



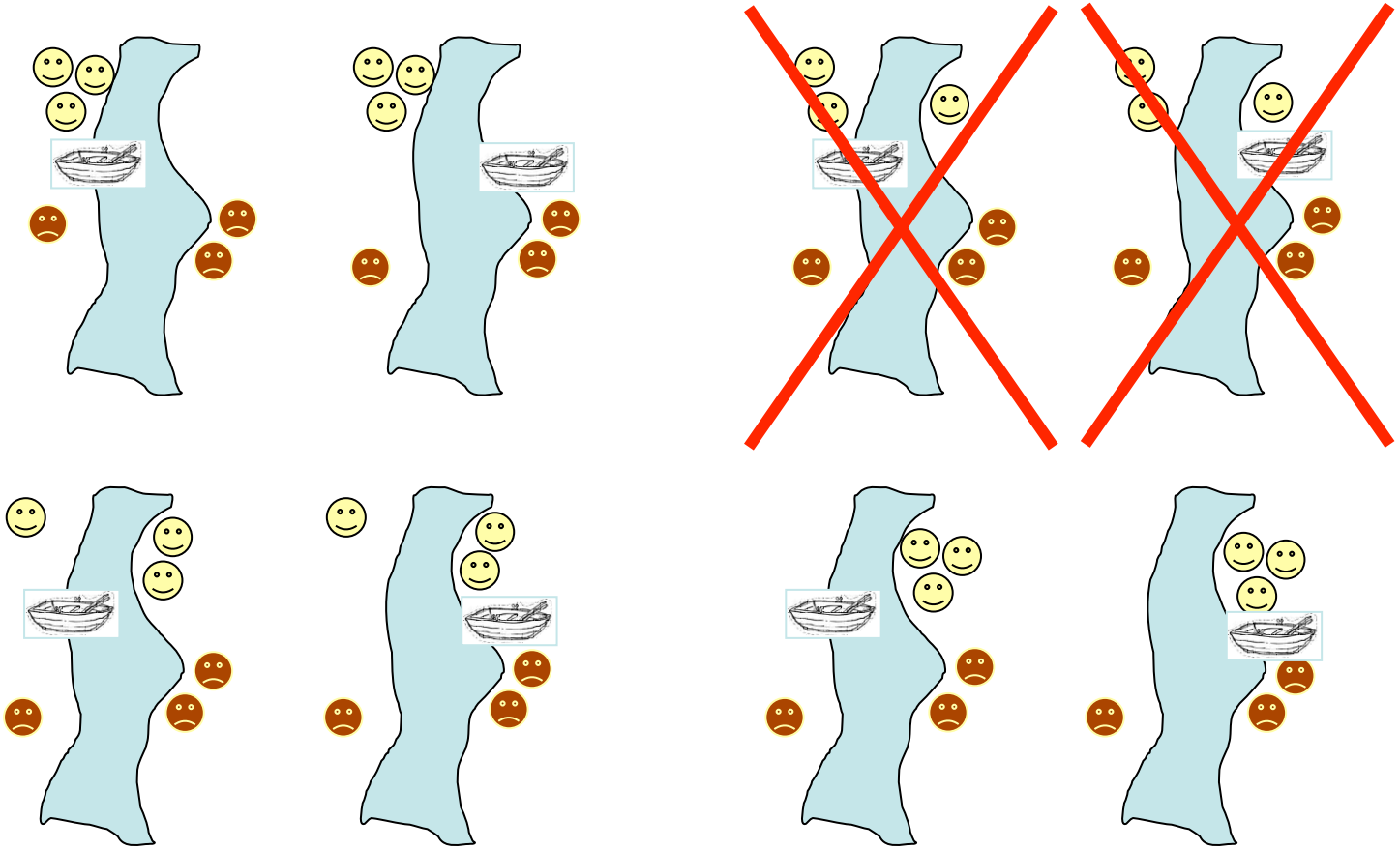
Pasul 2 Stări: misionari şi canibali

2 canibali în stânga



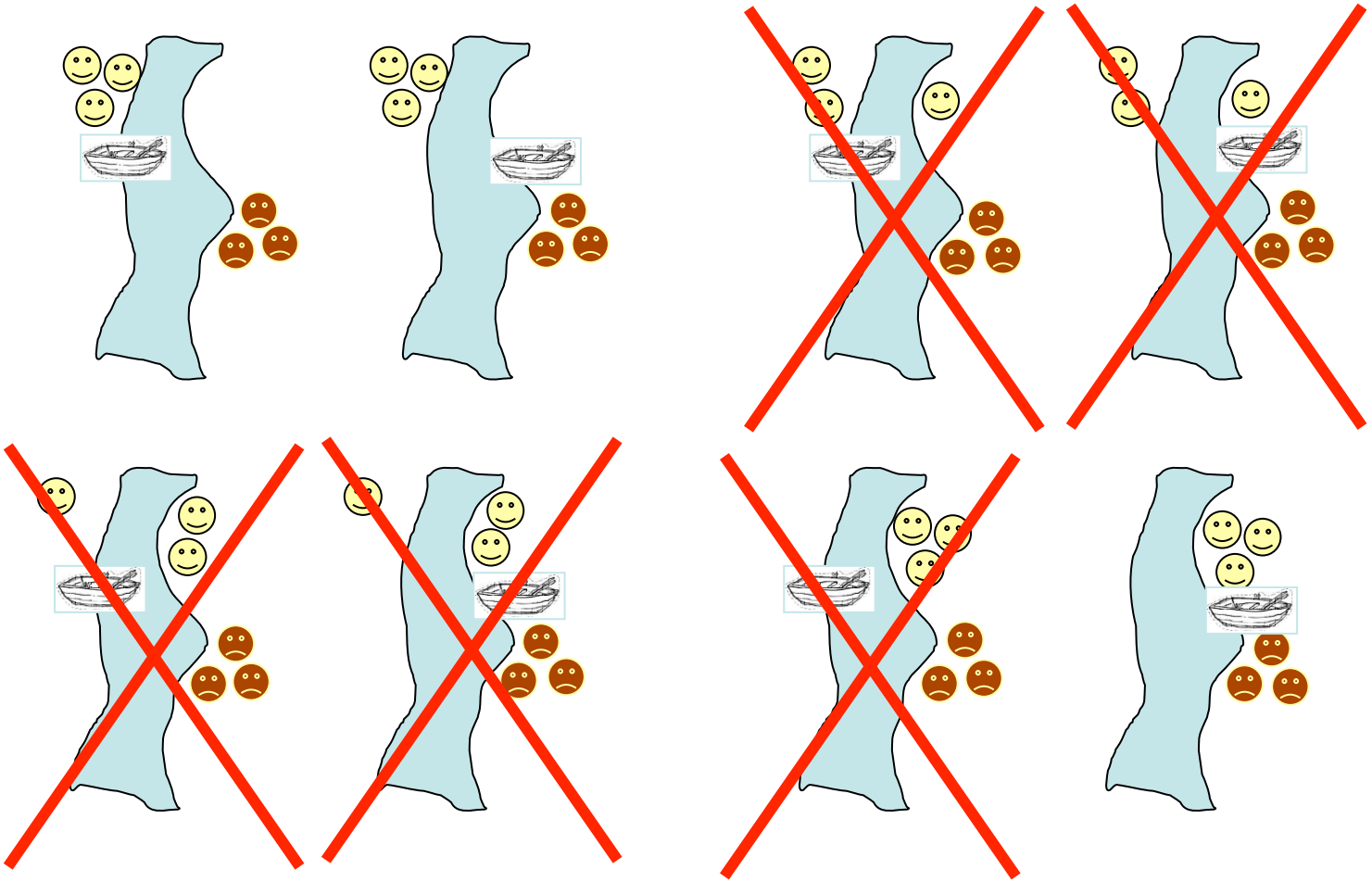
Pasul 2 Stări: misionari şi canibali

1 canibal în stânga



Pasul 2 Stări: misionari şi canibali

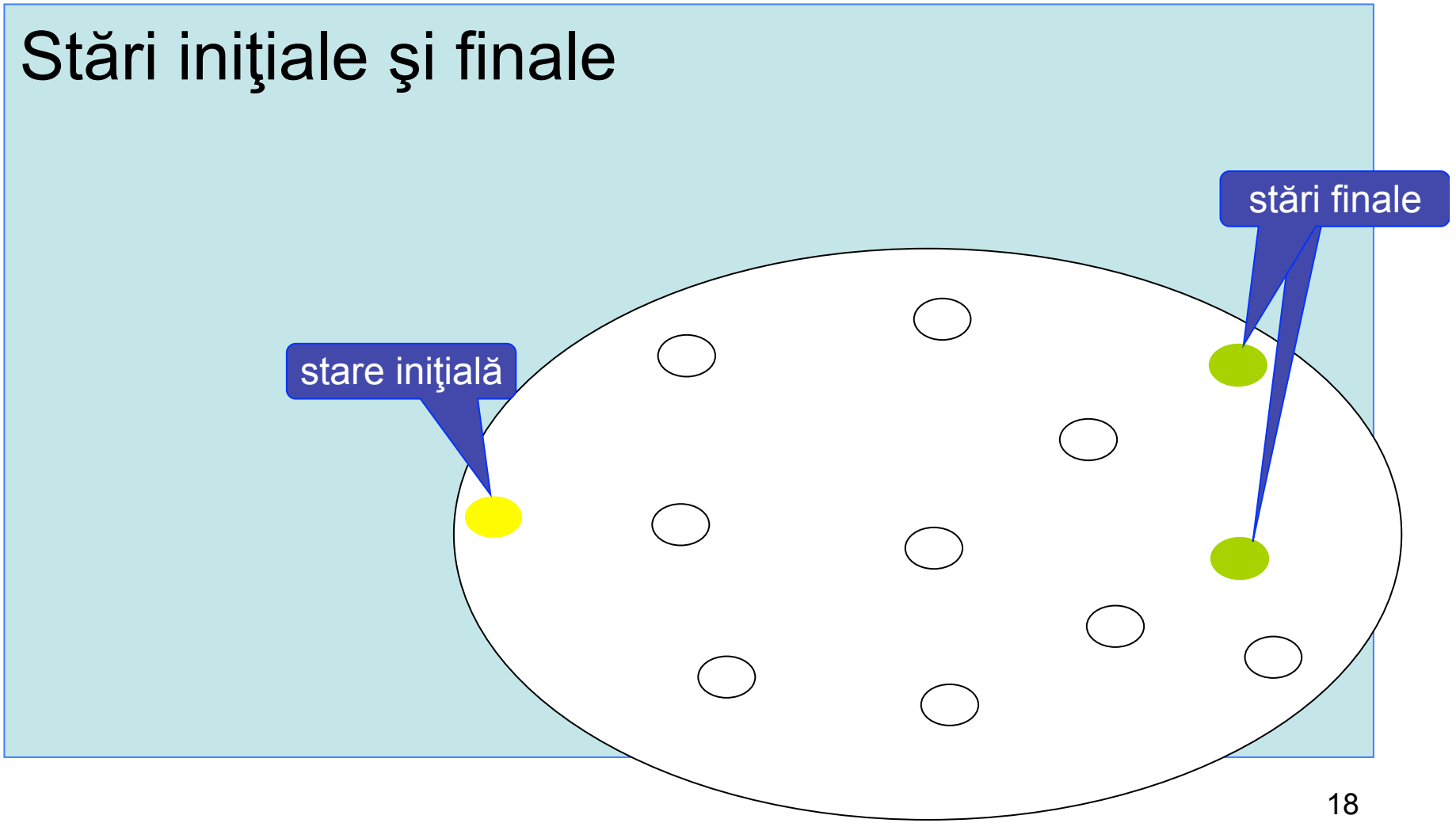
niciun canibal în stânga



Pasul 2

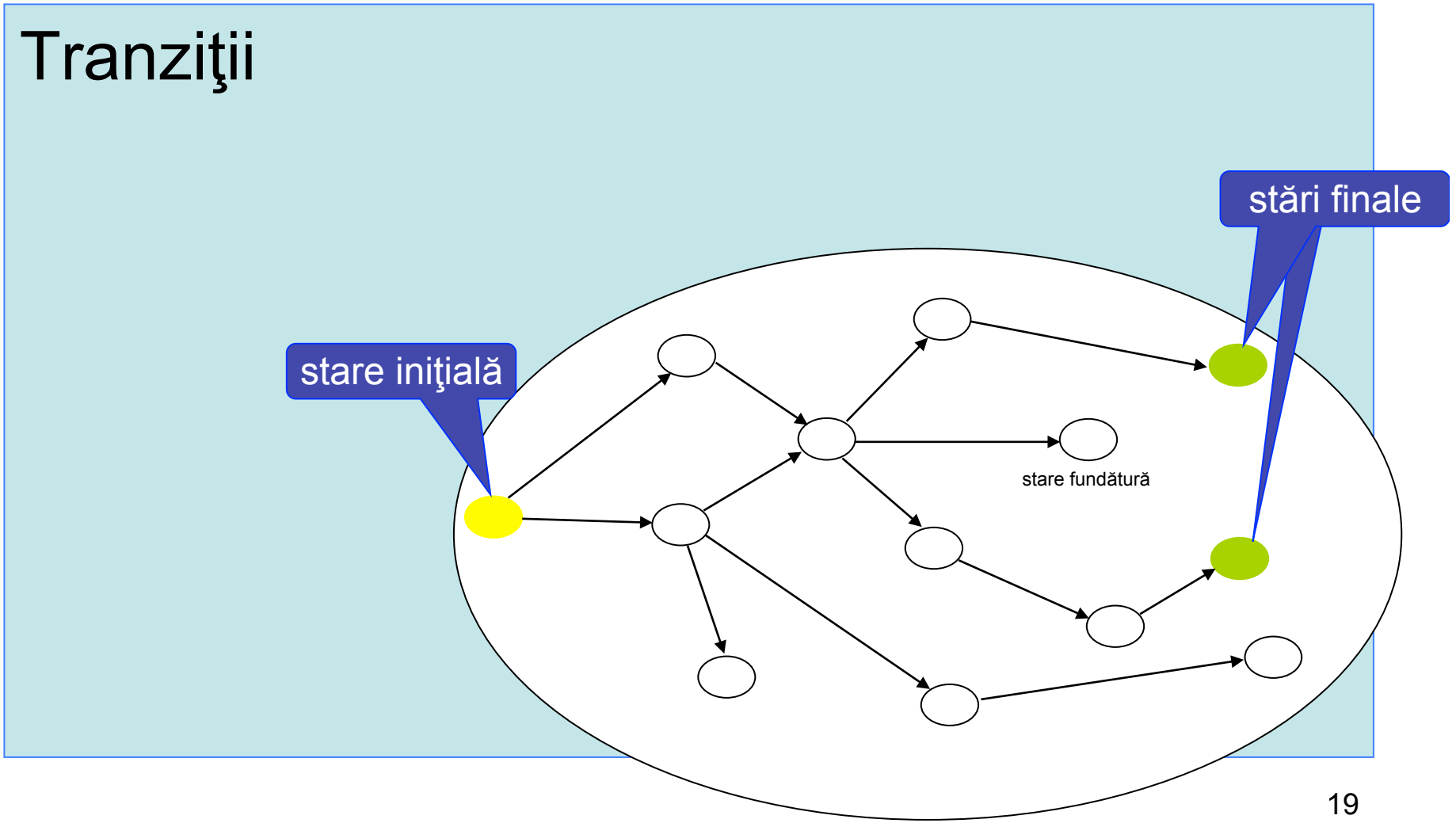
Stări, spaţiul stărilor, dimensiunea lui

Stări iniţiale şi finale



Pasul 2

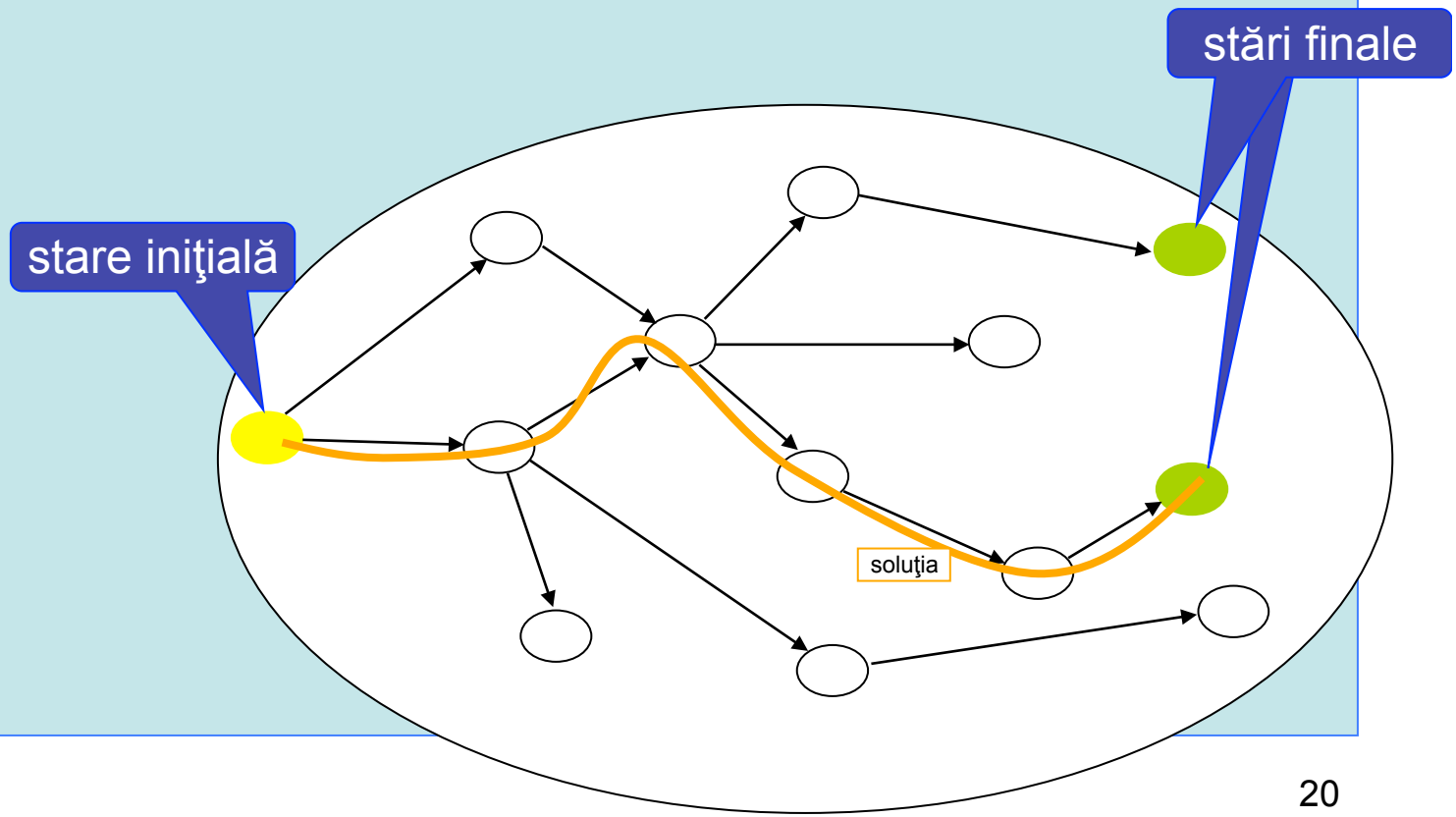
Stări, spaţiul stărilor, dimensiunea lui



Pasul 2

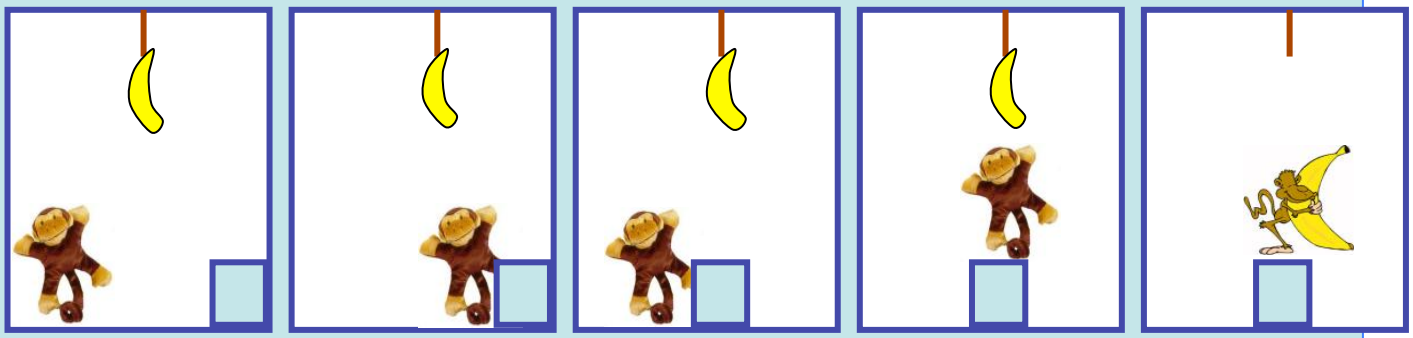
Stări, spaţiul stărilor, dimensiunea lui

Soluţia = un şir de tranziţii



Maimuţa şi banana

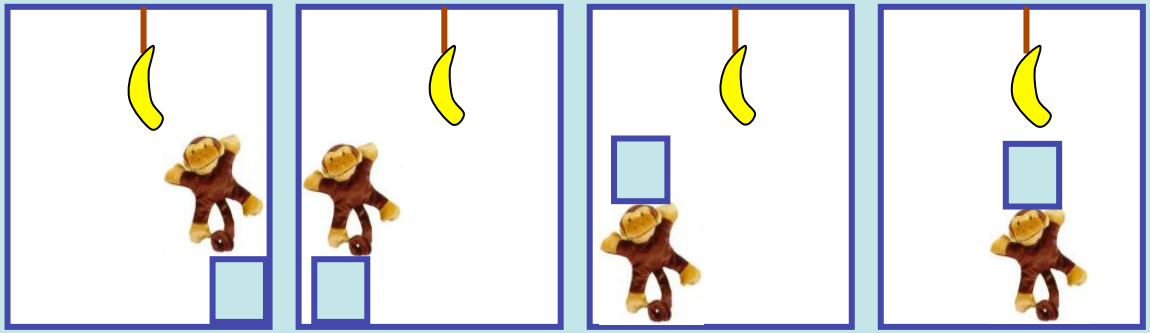
Soluţia = un şir de tranziţii



Pasul 2

Maimuţa şi banana

Alte stări posibile



Pasul 2

Pasul 3

Cum reprezentăm o stare?

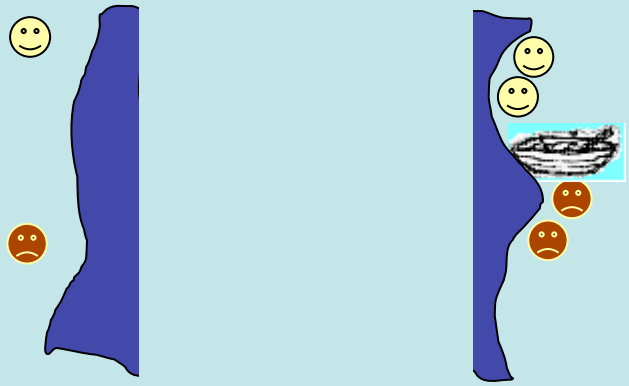
8-puzzle

o matrice 3x3

Pasul 3

Cum reprezentăm o stare?

Misionari şi canibali



un vector cu 3 poziţii: (c, m, b)

Pasul 3

Cum reprezentăm o stare?

Generarea frazelor

Pentru instanţa de problemă:

G1 = {N1, T1, S1, P1}

N1 = {PROP, GN, GV, S, V}

T1 = {*pisica*, *şoarecele*, *prinde*}

S1 = PROP

P1 = {PROP := GN GV,

GN := S,

GV := V GN,

S := *pisica*,

S := *şoarecele*,

V := *prinde*}

un şir de simboluri

Exemple de stări: PROP | GN GV | S GV | pisica GV | pisica V GN | pisica prinde GN | pisica prinde S | pisica prinde pisica

Pasul 3

Cum reprezentăm o stare?

Maimuţa şi banana

Relaţia maimuţă-cutie:
- **MC-departe** = Maimuţa se află departe de Cutie
- **MC-lângă** = Maimuţa se află lângă Cutie
- **MC-pe** = Maimuţa se afla pe Cutie
- **MC-sub** = Maimuţa de află sub Cutie

Relaţia Cutie – Banană:
- **CB-lateral** = Cutia este aşezată lateral faţă de Banană
- **CB-sub** = Cutia este aşezată sub Banană

Relaţia Maimuţa – Banană:
- **MB-departe** = Maimuţa se află departe de Banană
- **MB-aproape** = Maimuţa se află aproape de Banană
- **MB-ţine** = Maimuţa ţine Banana

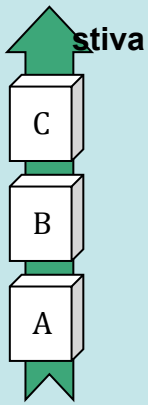
Starea iniţială: **MC-departe**, **CB-lateral**, **MB-departe**.
Starea finală: **MB-ţine**

o colecție de predicate

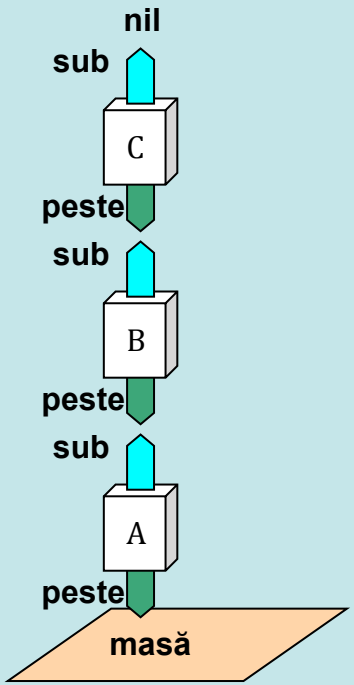
Pasul 3

Cum reprezentăm o stare?

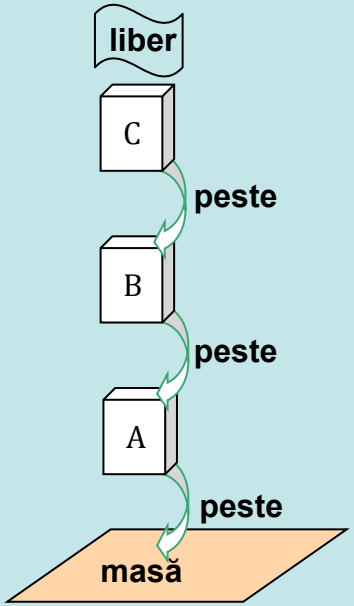
Lumea cuburilor



ordine definită global



ordine definită prin vecinătăţi locale



ordine definită prin relaţii dintre obiecte

Pasul 3

Cum reprezentăm o stare?

Lumea cuburilor – reprezentarea configurației brațului

mâna-liberă | **mâna-ţine**(X)

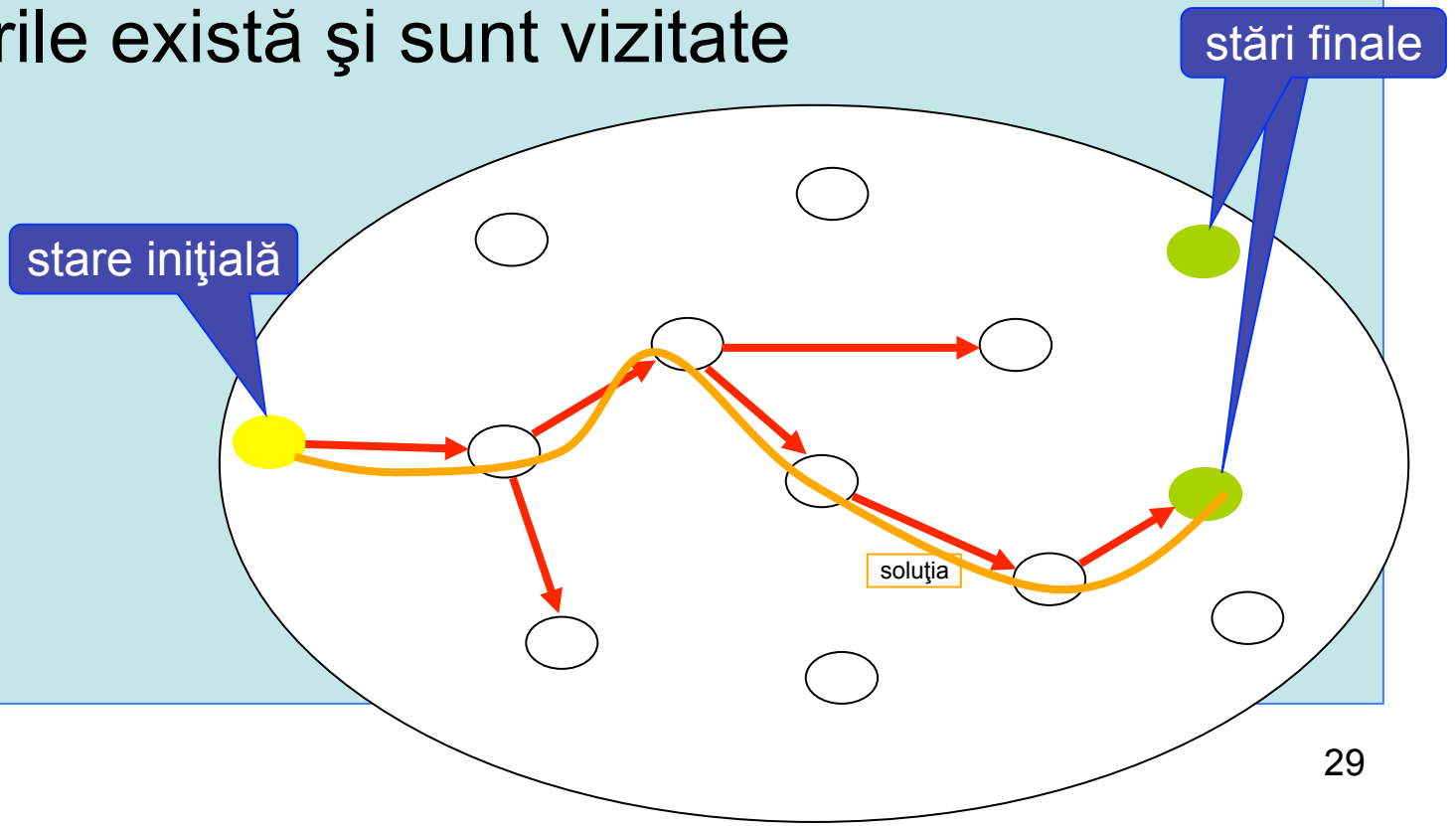
o colecție de predicate

Cum reprezentăm tranziţiile dintre stări

Pasul 4

Două moduri de a vedea o navigare în spaţiul stărilor:

– stările există şi sunt vizitate

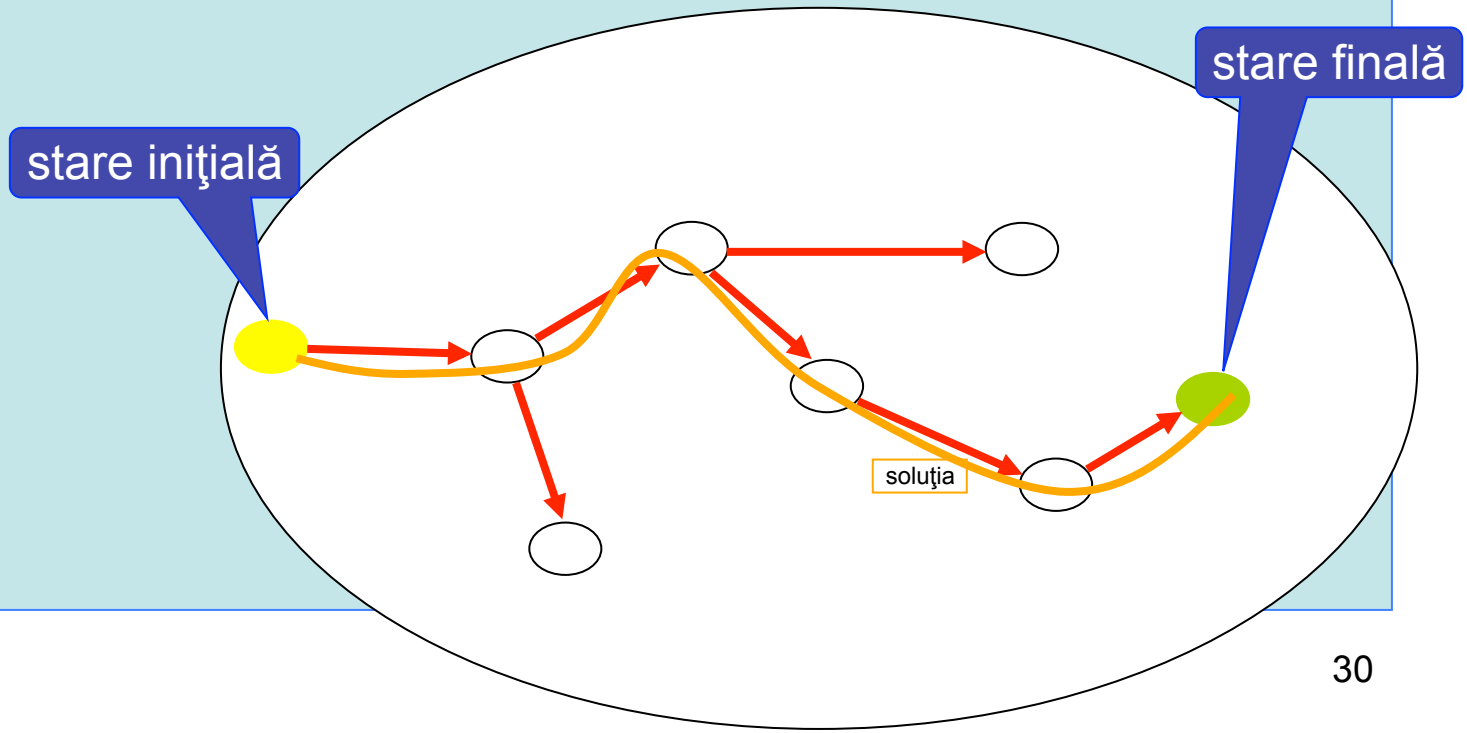


Cum reprezentăm tranziţiile dintre stări

Pasul 4

Două moduri de a vedea o navigare în spaţiul stărilor:

– stările sunt generate la momentul vizitării

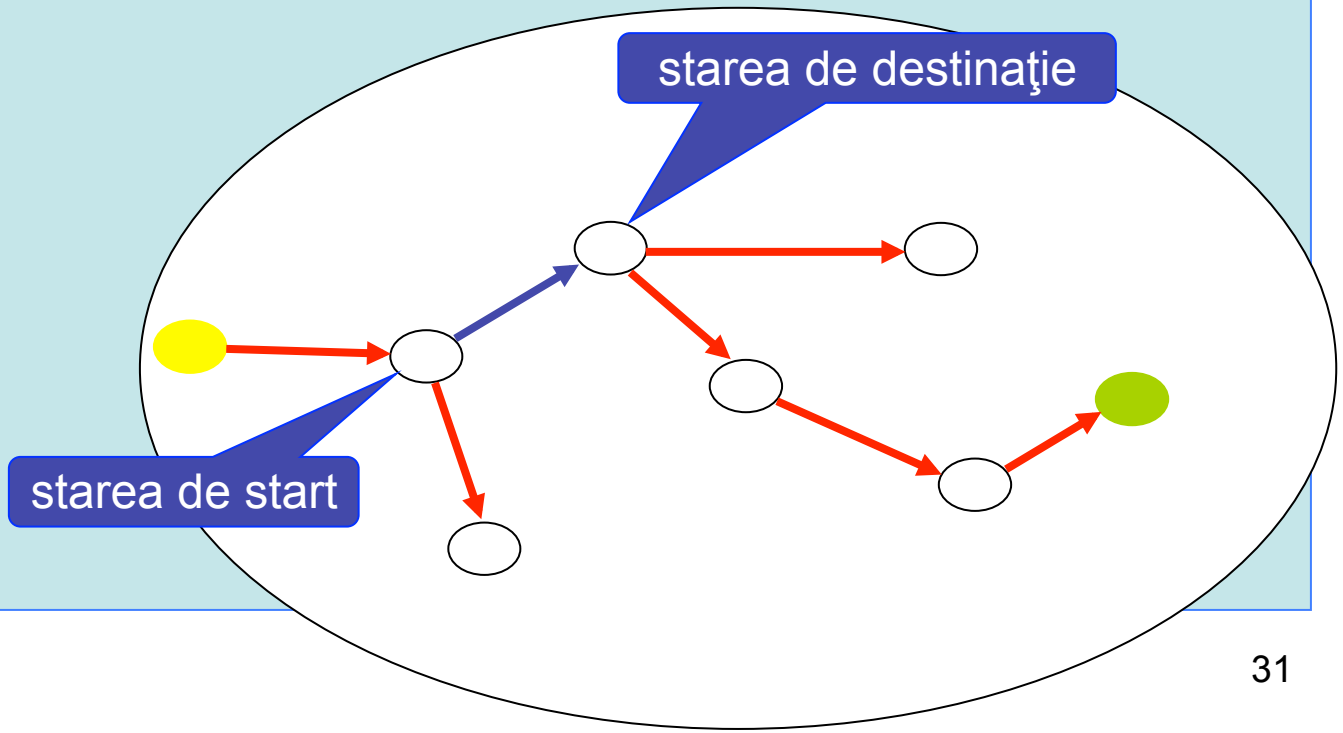


Cum reprezentăm tranziţiile dintre stări

Pasul 4

Un operator verifică condiţii şi produce transformări în stare

```
if <condiţii> then <acţiuni>
```



Cum reprezentăm tranziţiile dintre stări

Pasul 4

Şah:

regula salt-dublu-pion-din-a
DACĂ
pion în poziţia (a,2) şi
poziţia (a,3) e liberă şi
poziţia (a,4) e liberă
ATUNCI
mută pionul din poziţia (a,2) în poziţia (a,4)

8 reguli de acest fel...

Cum reprezentăm tranziţiile dintre stări

Pasul 4

Şah:

regula salt-dublu-pion(x)
DACĂ
pion în poziţia (x,2) şi
poziţia (x,3) e liberă şi
poziţia (x,4) e liberă
ATUNCI
mută pionul din poziţia (x,2) în poziţia (x,4)

O regulă de acest fel!

Două reguli, dacă parametrizez și jucătorul.

Cum reprezentăm tranziţiile dintre stări

Pasul 4

8-puzzle:

Regula mută-piesa-1-sus
DACĂ
piesa 1 nu e lipită de marginea de sus a tablei
şi poziţia de deasupra e liberă
ATUNCI
schimbă poziţia piesei 1 cu a căsuţei aflată deasupra ei

8 reguli de acest fel!
x 4 direcţii ➔ 32 reguli în total

Cum reprezentăm tranziţiile dintre stări

Pasul 4

8-puzzle:

Regula mută-blanc-sus
DACĂ
blancul nu e lipit de marginea de sus a tablei
ATUNCI
schimbă poziţia blancului cu a căsuţei aflată deasupra acestuia

O singură regulă de acest fel!

x 4 direcţii ➔ 4 reguli în total

Cum reprezentăm tranziţiile dintre stări

Pasul 4

Maimuţa şi banana:

aflată departe de cutie, maimuţa se aproprie de cutie:
apropie-MC:
dacă {MC-departe} atunci ŞTERGE{MC-departe}, ADAUGĂ{MC-lângă}

aflată lângă cutie, maimuţa se depărtează de cutie:
depărtează-MC:
dacă {MC-lângă} atunci ŞTERGE{MC-lângă}, ADAUGĂ{MC-departe}

aflată lângă cutie şi lateral faţă de banană, maimuţa trage cutia sub banană:
trage-sub-MCB:
dacă {MC-lângă, CB-lateral} atunci ŞTERGE {CB-lateral}, ADAUGĂ{CB-sub}

aflată lângă cutie şi sub banană, maimuţa trage cutia de sub banană: **trage-lateral-MCB**;
aflată lângă cutie, maimuţa se urcă pe ea: **urcă-MC**;
aflată pe cutie, maimuţa coboară de pe ea: **coboară-MC**;
aflată lângă cutie, maimuţa îşi urcă cutia deasupra capului: **urcă-pe-cap-MC**;
din postura în care maimuţa ţine cutia deasupra capului, maimuţa îşi dă jos cutia de pe cap: **coboară-de-pe-cap-MC**;
aflată pe cutie şi sub banană, maimuţa apucă banana: **apucă-MB**.

Cum reprezentăm tranziţiile dintre stări

Pasul 4

Sisteme de reguli STRIPS

stările reprezentate ca set de predicate (caracteristici)

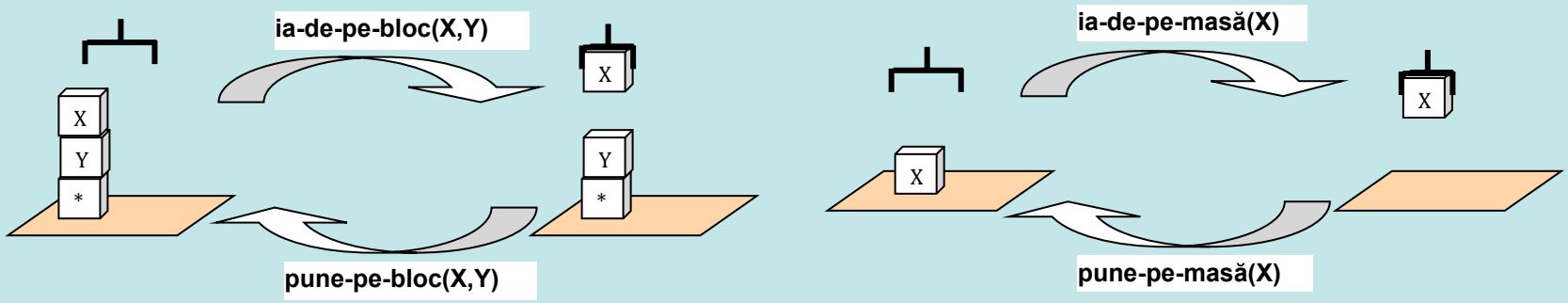
regulile:

if <lista-precondiţii> then <lista-ştergeri> <lista-adăugări>

Cum reprezentăm tranziţiile dintre stări

Pasul 4

Sisteme de reguli STRIPS în lumea cuburilor



Cum reprezentăm tranziţiile dintre stări

Pasul 4

Sisteme de reguli STRIPS în lumea cuburilor

ia-de-pe-bloc(X,Y):

dacă {peste(X, Y), liber(X), mâna-liberă} atunci ŞTERGE{peste(X, Y), liber(X), mâna-liberă} ADAUGĂ{liber(Y), mâna-ţine(X)}

ia-de-pe-masă(X):

dacă {peste(X, masă), liber(X), mâna-liberă} atunci ŞTERGE{peste(X, masă), liber(X), mâna-liberă} ADAUGĂ{mâna-ţine(X)}

pune-pe-bloc(X,Y):

dacă {mâna-ţine(X), liber(Y)} atunci ŞTERGE{mâna-ţine(X), liber(Y)} ADAUGĂ{peste(X, Y), liber(X), mâna-liberă}

pune-pe-masă(X):

dacă {mâna-ţine(X)} atunci ŞTERGE{mâna-ţine(X)} ADAUGĂ{peste(X, masă), liber(X), mâna-liberă}

Pasul 5

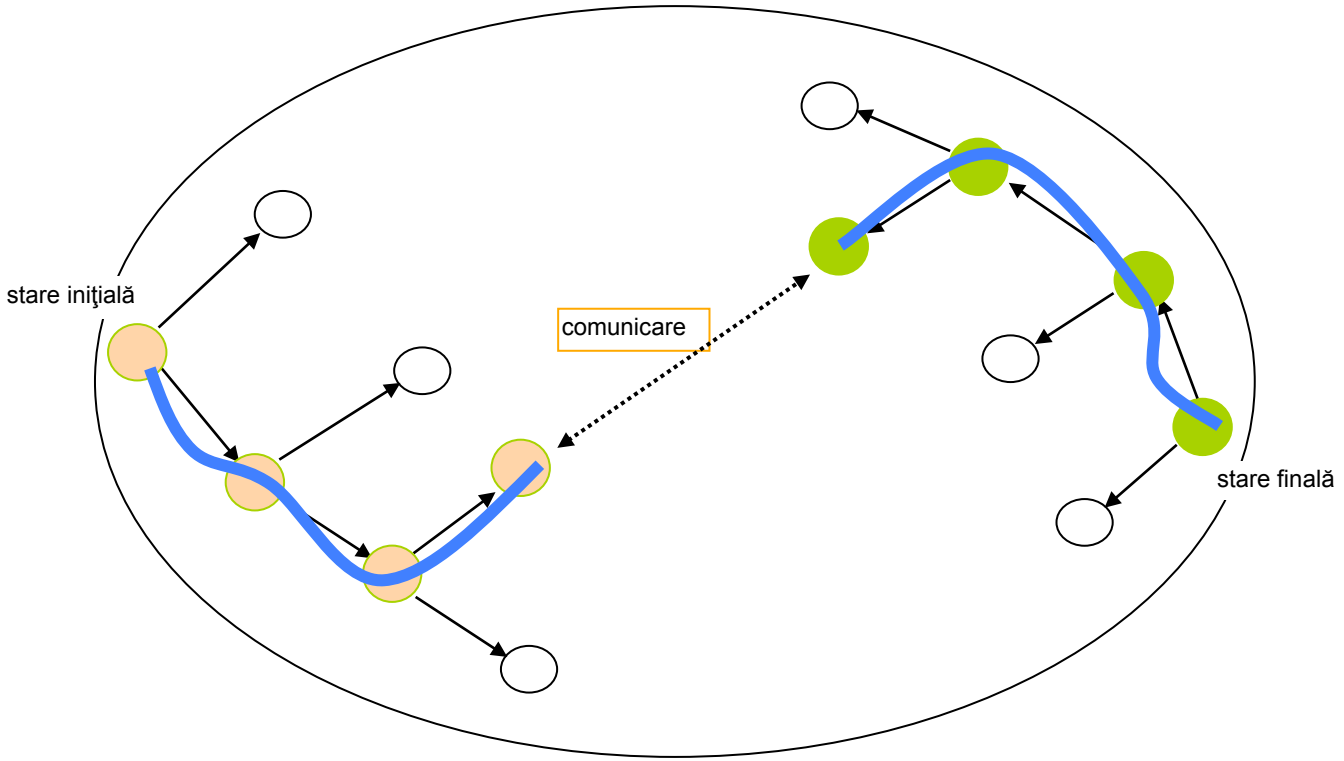
Căutarea soluției

- Algoritmi și euristici de căutare în spațiul stărilor
  - Strategii irevocabile
    - ascensională (*hill-climbing*)
  - Strategii tentative
    - ascensională cu revenire (*backtracking*)
  - Strategii exhaustive (*brute-force*)
    - generează-și-testează
    - întâi-în-adâncime (*depth-first*)
    - întâi-în-lărgime (*breadth-first*)
    - cel mai bun întâi (*best-first*)

Pasul 5

Căutare în spațiul stărilor

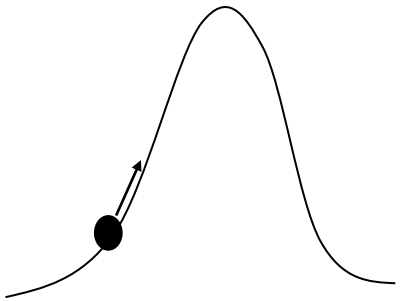
- Căutare bidirecţională sincronă



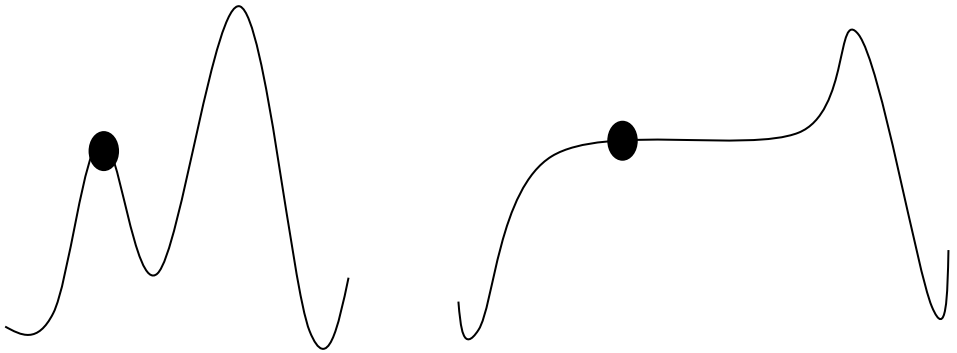
Pasul 5

Strategii irevocabile: *hill-climbing*

- Cale de întoarcere nu există
  - o funcție apreciază apropierea de soluție



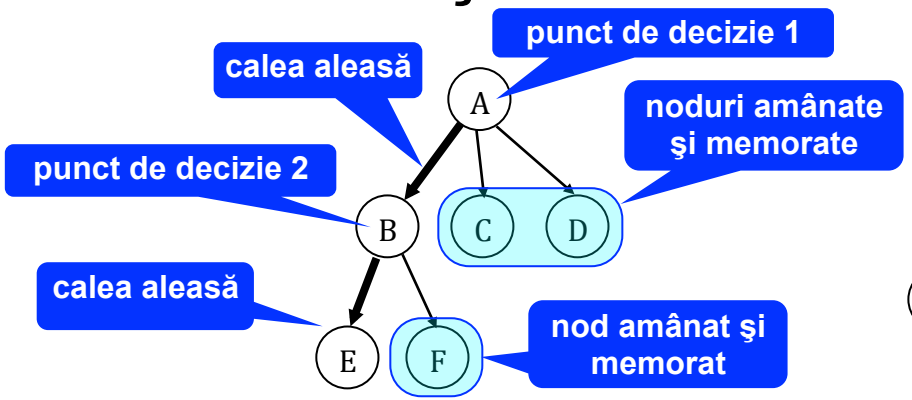
  - pericole: maxime locale, platouri



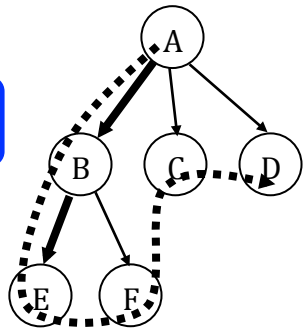
Pasul 5

Strategii tentative: *backtracking*

- Dacă o stare nu mai are succesori "iau urma îndărăt”
  - o memorie în care se plasează la fiecare pas stările vecine, diferite de cea în care se efectuează tranziţia



a. În fiecare punct în care se face o alegere, căile neexplorate se salvează



b. Când structura de salvare este o stivă, explorarea se face în ordinea întâi-în-adâncime

Pasul 5

Backtracking hill-climbing

```
function backtracking-hill-climbing(initial-state)
begin
  stack = initial-state;
  while (stack)
   { current-state = pop(stack);
     if (current-state e stare finală) return current-state;
     all-new-neighbour-states <- setul stărilor ce pot fi
       obţinute din current-state prin operatorii aplicabili ei);
     elimină din setul all-new-neighbour-states toate stările
       deja vizitate;
     if (all-new-neighbour-states ≠ ∅)
      { sortează all-new-neighbour-states în ordinea
          descrescătoare a valorilor funcţiei euristice;
        current-state <- prima stare clasată în
          all-new-neighbour-states);
        if (current-state e stare finală) return current-state;
        else
         { all-new-neighbour-states <- all-new-neighbour-states –
                                       {current-state};
           stack <- push(all-new-neighbour-states,stack);
         }
      }
   }
  return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

Pasul 5

- **Căutare întâi-în-adâncime (*depth-first search – DFS*)**
  - memoria: stivă

```
function depthFirstSearch(root)
begin
  stack <- push(root, ∅);
  while (stack not empty)
   { node <- pop(stack);
     if goal(node) then return node;
     else push(node's successors, stack);
   }
  return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

Pasul 5

- **Căutare întâi-în-lărgime (*breadth-first search – BFS*)**
  - memoria: coadă

```
function breadthFirstSearch(root)
begin
  queue <- in(root, ∅);
  while (queue not empty)
   { node <- out(queue);
     if goal(node) then return node;
     else in(node's successors, queue);
   }
  return FAIL;
end
```

Metode de căutare sistematică (*brute-force*)

Pasul 5

- **Căutare cel-mai bun-întâi (*best-first search*)**
  - memoria: listă; o funcție euristică de cost

```
function bestFirstSearch(root)
begin
  list <- include(root, ∅);
  while (list not empty)
   { node <- get-first(list);
     if goal(node) then return node;
     else
      { include(node's successors, list);
        sort list descending;
      }
   }
  return FAIL;
end
```

Pasul 5

Exemplu: *best-first search*