

Curs 9

Probleme de IA și rezolvarea lor

Cele 5 cerințe în modelarea unei probleme de IA

Pasul 1

Diferențiază problema generală de instanțele ei

Pasul 2

Recunoaște o stare și apreciază dimensiunea spațiului stărilor

Pasul 3

Găsește cea mai adecvată reprezentare a stărilor

Pasul 4

Reprezintă tranzițiile 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 mișca cu o poziție, pe orizontală sau verticală, în limitele cadrului tablei, în locul rămas liber. Se dă o configurație inițială și una finală a tablei. Trebuie să se găsească secvența de mutări care să aducă piesele din configurația inițială în cea finală.

Probleme de dimensiuni mici

- **Problema misionarilor și canibalilor**

3 misionari și 3 canibali se află la marginea unui râu, cu scopul de a trece pe celălalt mal. Ei au la dispoziție 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 mâncați de canibali. Problema constă în a afla cum pot trece râul cele 6 persoane în deplină siguranță.

Probleme de dimensiuni mici

- **Problema generării frazelor în limbaj natural**

Se dispune de o gramatică (un set de simboluri numiți terminali, un set de simboluri numiți neterminali, o colecție de reguli, fiecare arătând cum poate fi expandată o categorie compusă în subcompuși și un simbol de start). Se dorește generarea unei exprimări corecte gramatical.

Probleme de dimensiuni mici

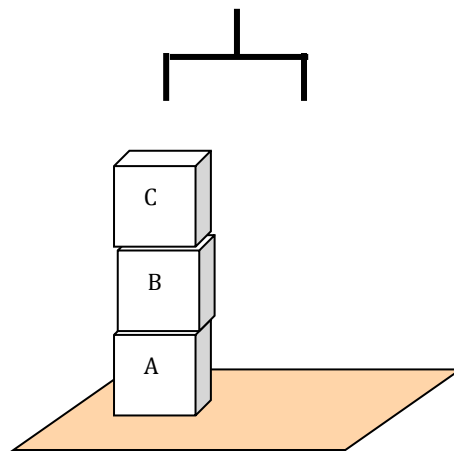
- **Problema maimuței și a bananei**

O maimuță este închisă într-o cușcă în care se mai află o banană atârnată de tavan la o înălțime la care maimuța nu poate ajunge și, într-un colț, o cutie. După un număr de încercări nereușite de a apuca banana, maimuța merge la cutie, o deplasează sub banană, se urcă pe cutie și apucă banana. Se cere să se formalizeze maniera de raționament a maimuței.

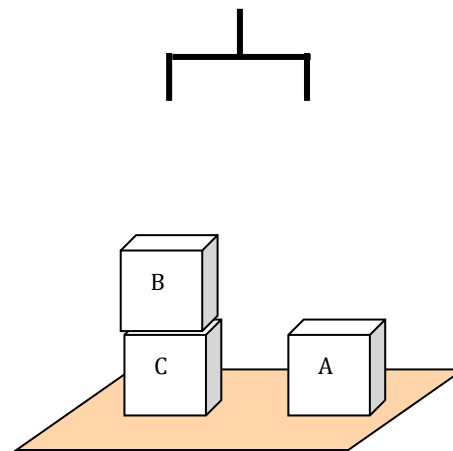
Probleme de dimensiuni mici

- **Lumea cuburilor**

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



starea inițială



starea finală

Problemă, instanță de problemă

- 8-puzzle
 - formulată ca o instanță de problemă
- Misionarii și canibalii
 - formulată ca o instanță de problemă
- Generarea frazelor
 - formulată ca o problemă
- Maimuța și banana
 - formulată ca o instanță de problemă
- Lumea cuburilor
 - formulată ca o instanță de problemă
- Alte exemple:
 - jocul de șah
 - condusul mașinii...

Un exemplu de instanță de problemă

- Generarea limbajului:

$G1 = \{N1, T1, PROP, P1\}$, în care:

$N1 = \{PROP, GN, GV, S, V\}$ – o mulțime de neterminali cu semnificațiile:
propoziție, grup nominal, grup verbal, substantiv și verb;

$T1 = \{pisica, șoarecele, prinde\}$ – o mulțime de cuvinte;

$PROP1$ = simbolul start al gramaticii, alegerea lui semnifică că ceea ce se dorește să se obțină reprezintă propoziții ale acestui mini-limbaj;

$P1 = \{PROP := GN GV,$

$GN := S,$

$GV := V GN,$

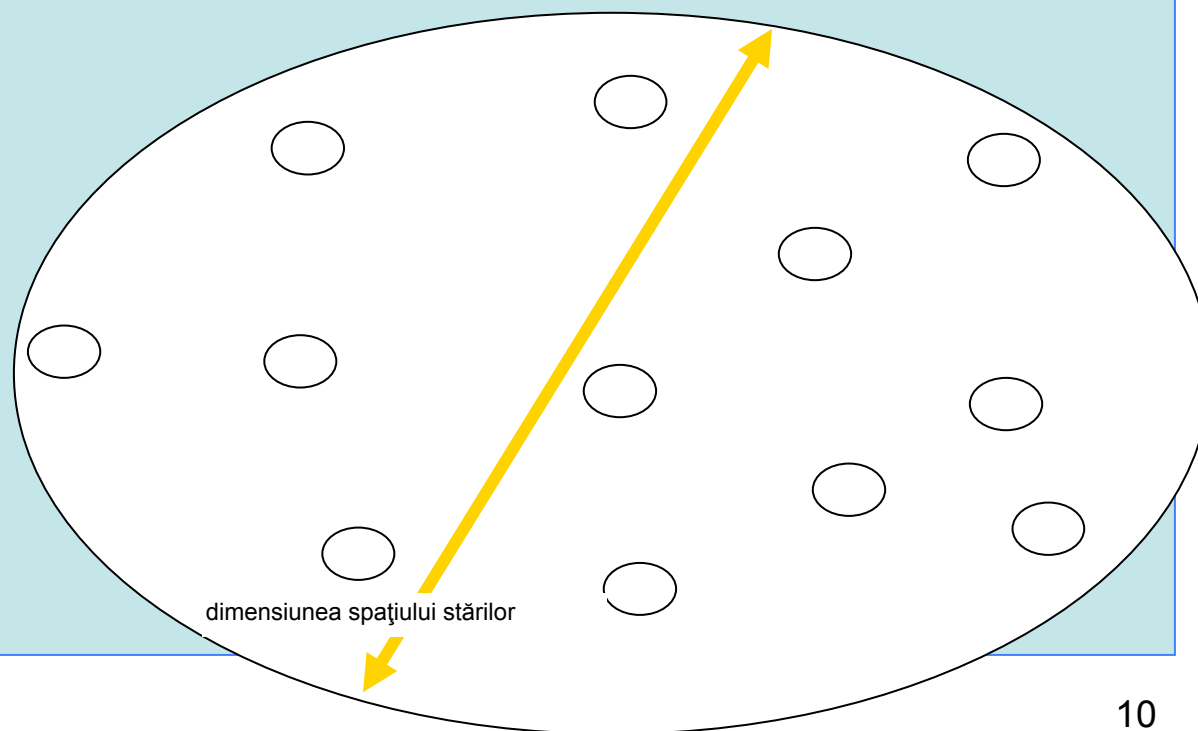
$S := pisica,$

$S := șoarecele,$

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

Spațiul problemei

- Stări, dimensiunea spațiului



Dimensiunea spațiului stărilor

- Jocul de șah: 10^{120}



Pasul 2

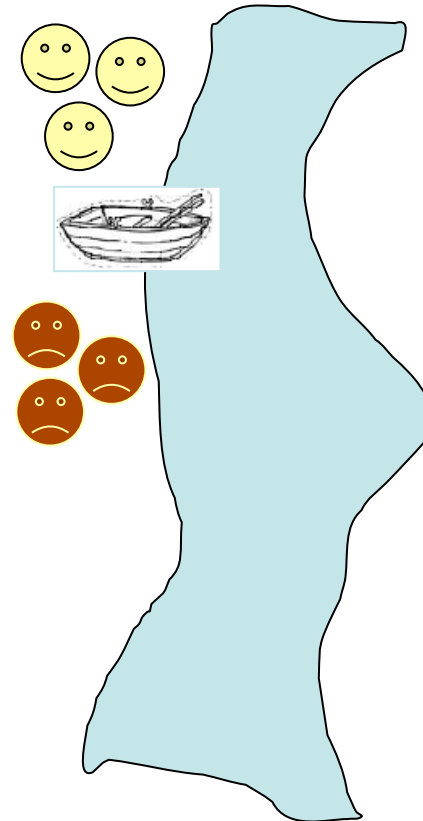
Dimensiunea spațiului stărilor

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



Dimensiunea spațiului stărilor

- Jocul de șah: 10^{120}
- 8-puzzle: $9!$
- misionari și canibali:



Pasul 2

Stări: misionari și canibali

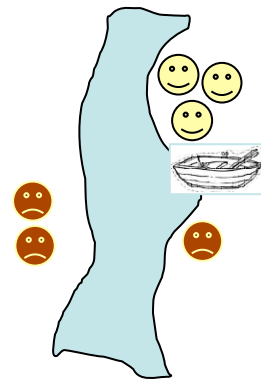
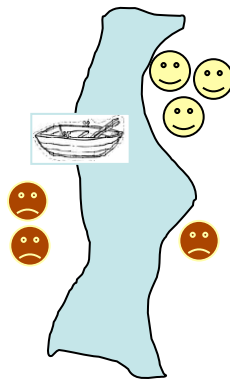
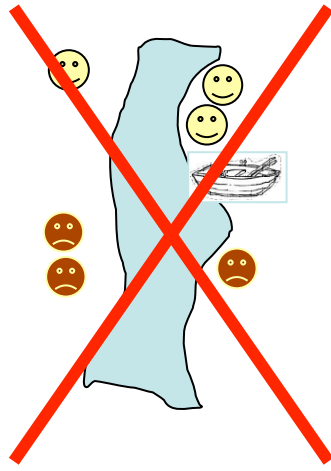
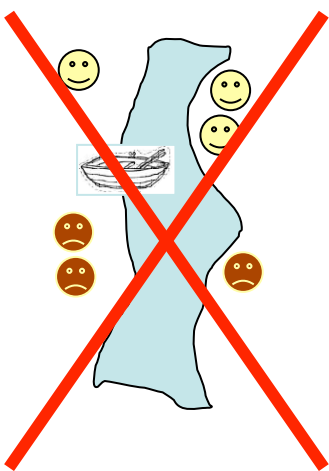
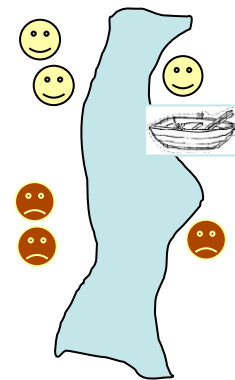
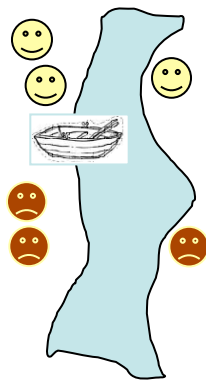
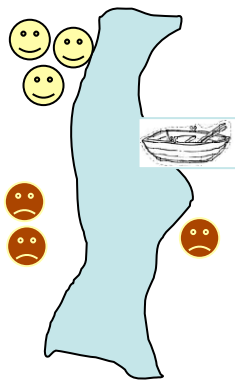
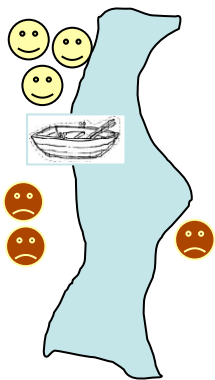
3 canibali în stânga



Pasul 2

Stări: misionari și canibali

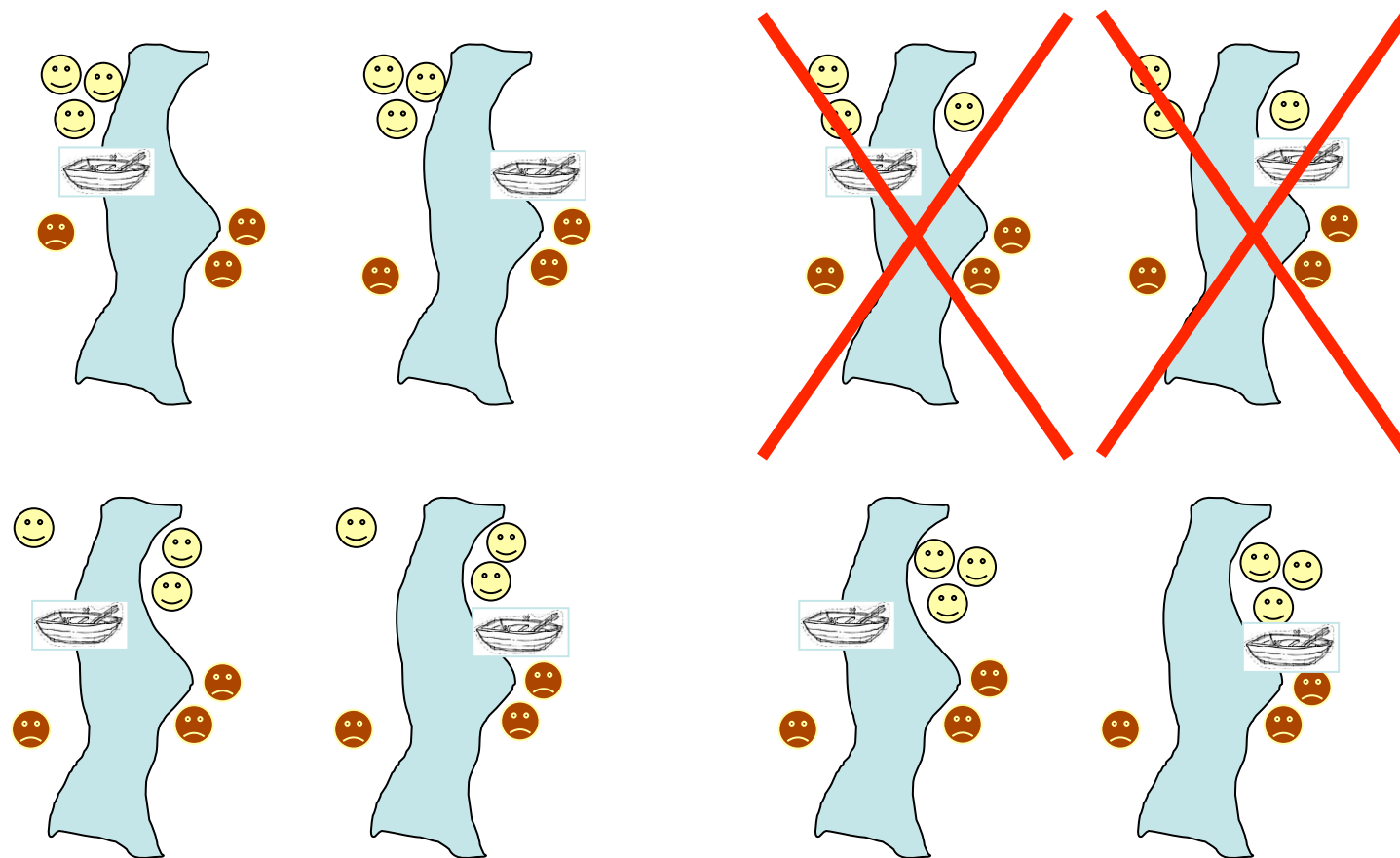
2 canibali în stânga



Pasul 2

Stări: misionari și canibali

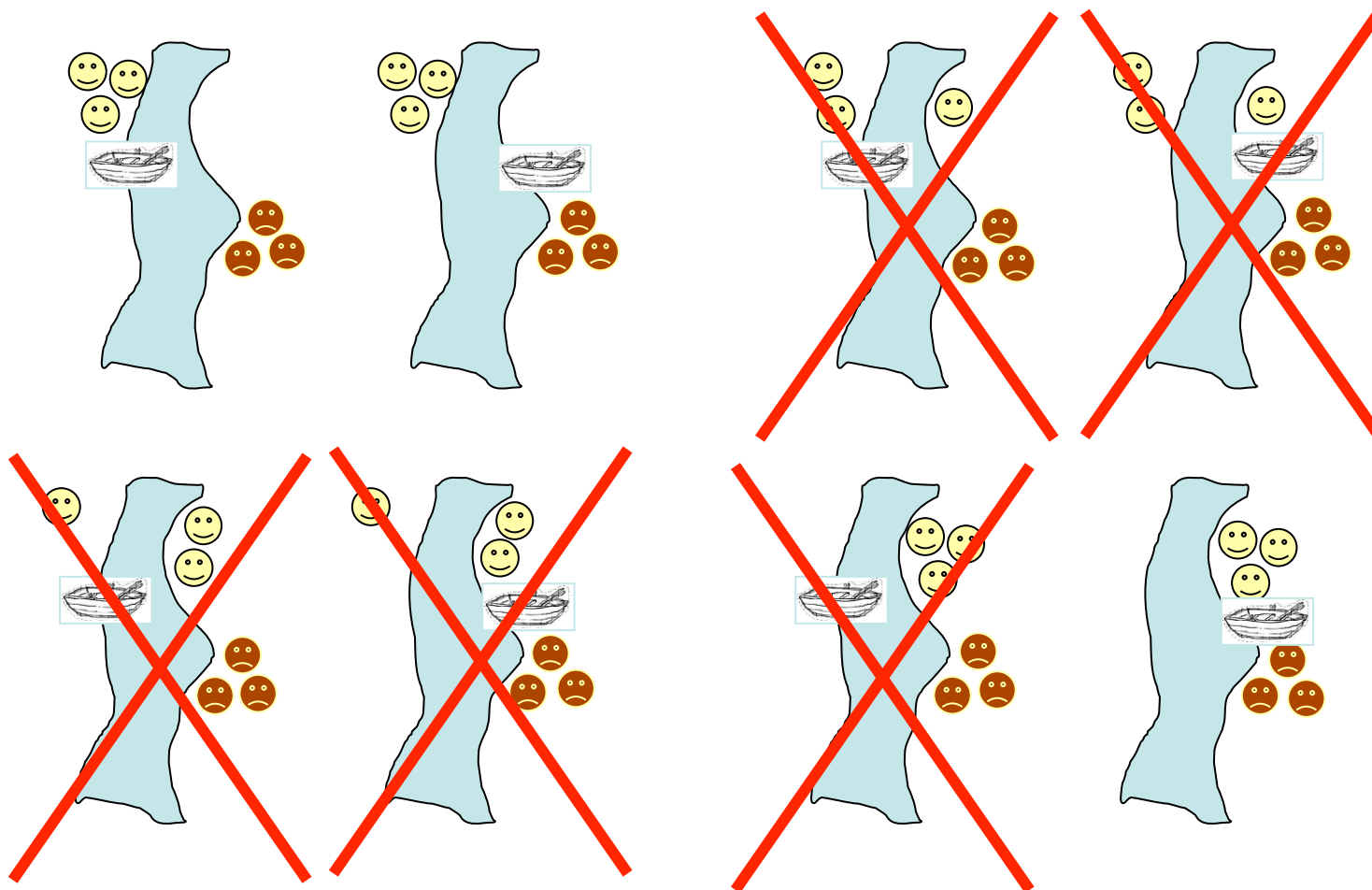
1 canibal în stânga



Pasul 2

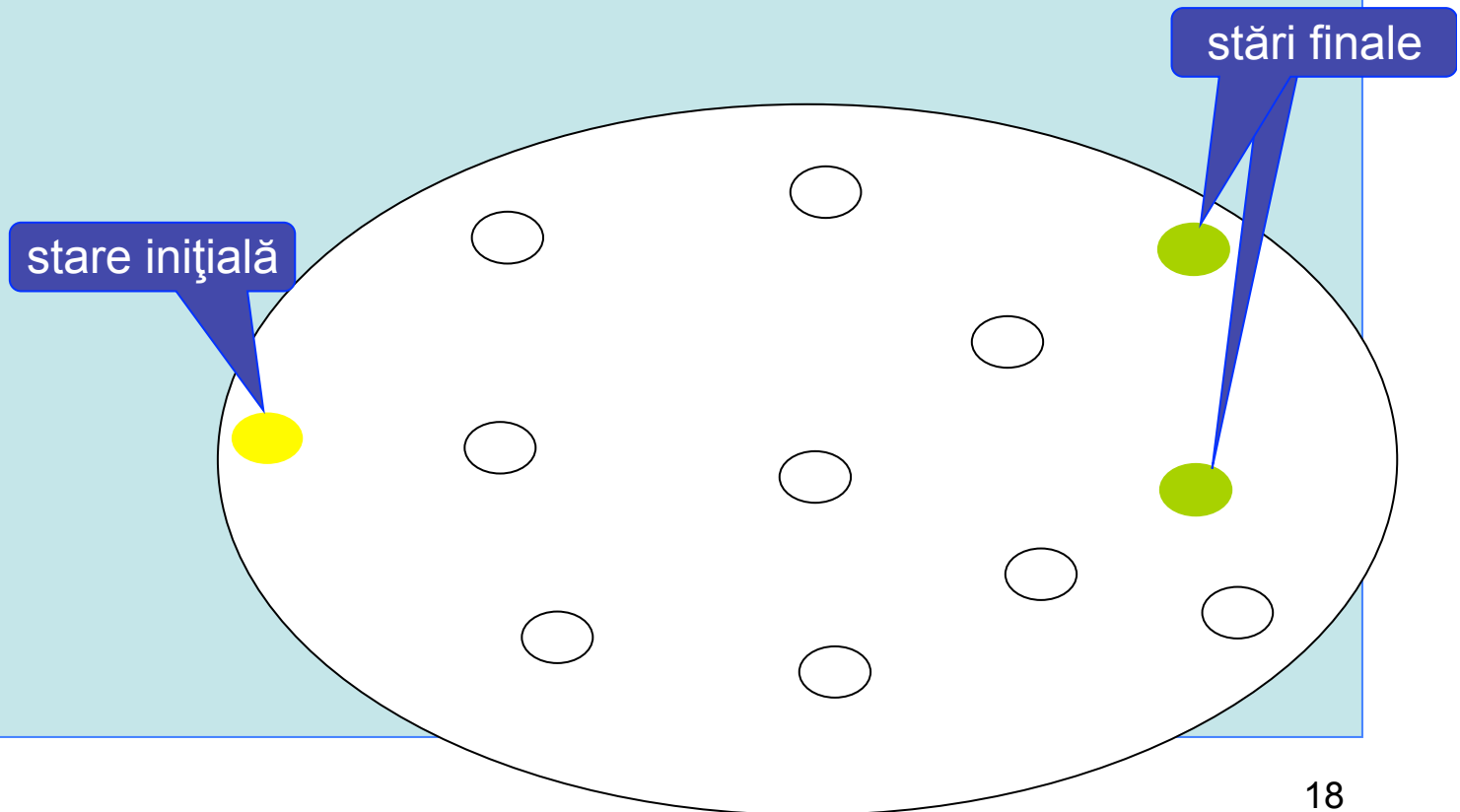
Stări: misionari și canibali

niciun canibal în stânga



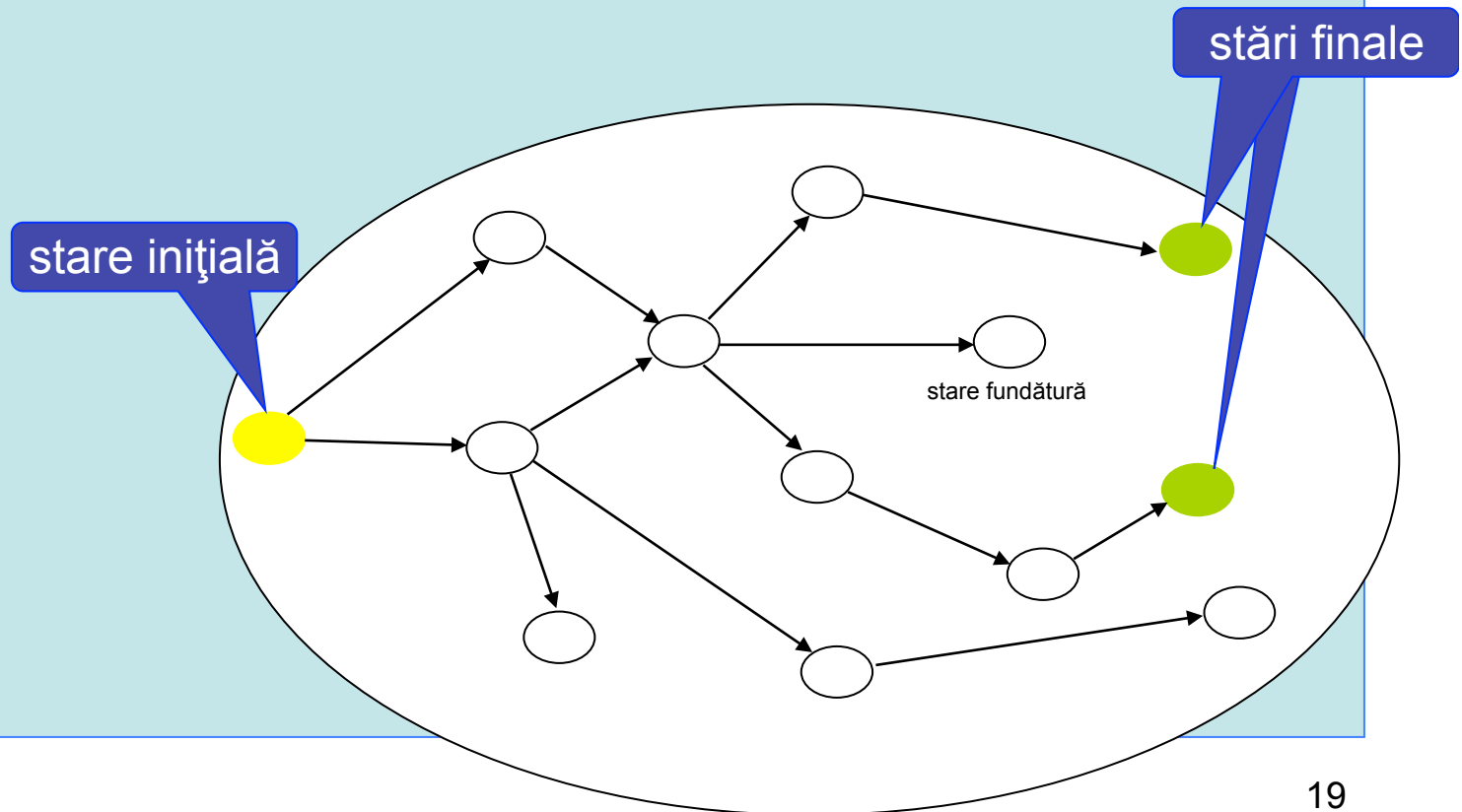
Stări, spațiul stărilor, dimensiunea lui

Stări inițiale și finale



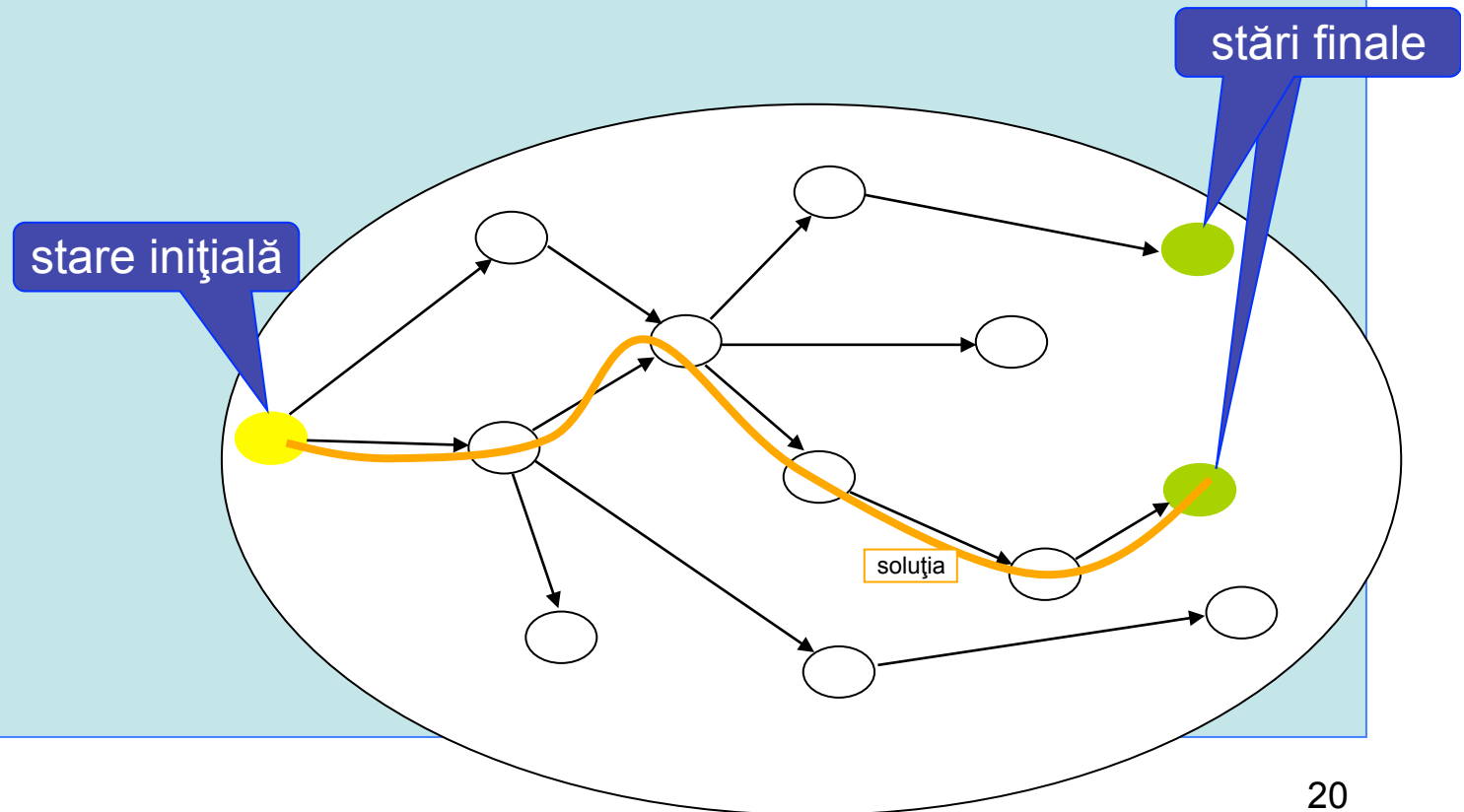
Stări, spațiul stărilor, dimensiunea lui

Tranziții



Stări, spațiul stărilor, dimensiunea lui

Soluția = un șir de tranziții

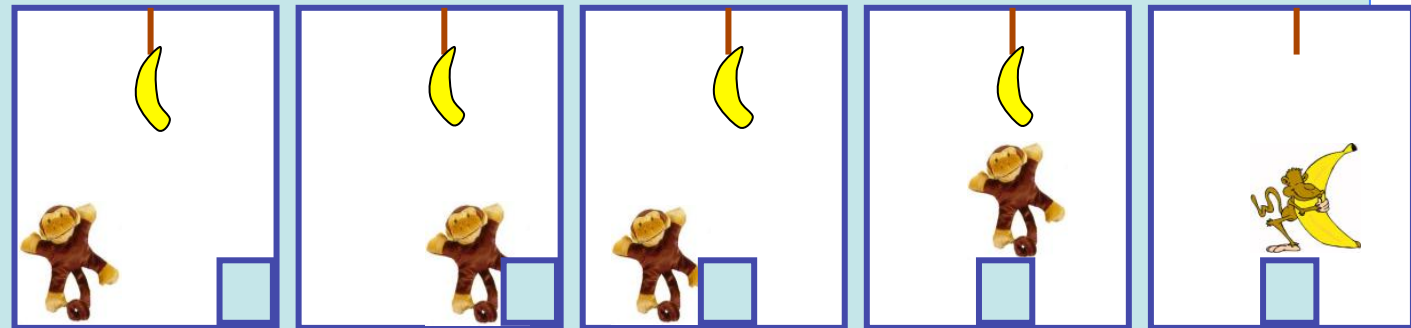




Maimuța și banana



Soluția = un șir de tranziții

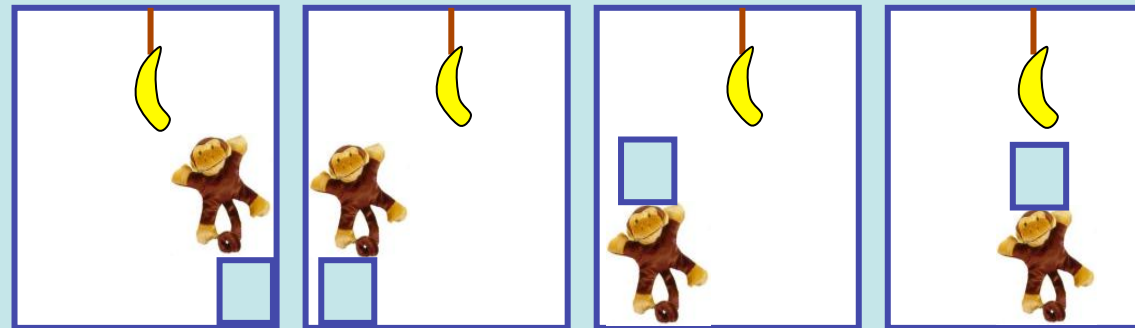




Maimuța și banana



Alte stări posibile



Cum reprezentăm o stare?

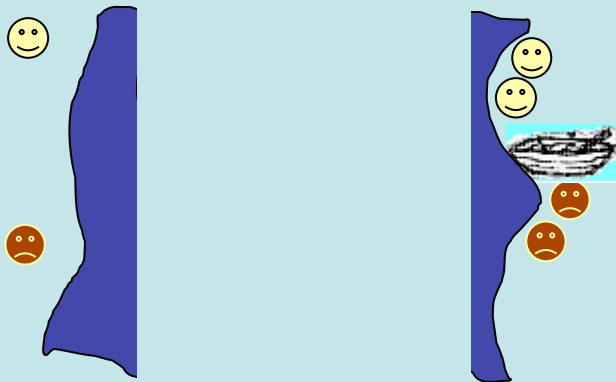
8-puzzle



o matrice 3x3

Cum reprezentăm o stare?

Misionari și canibali



un vector cu 3 poziții: (c, m, b)

Cum reprezentăm o stare?

Generarea frazelor

Pentru instanța de problemă:

$G1 = \{N1, T1, S1, P1\}$

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

$T1 = \{pisica, șoarecele, prinde\}$

$S1 = PROP$

$P1 = \{PROP := GN GV,$

$GN := S,$

$GV := V GN,$

$S := pisica,$

$S := șoarecele,$

$V := prinde\}$

un șir de simboluri

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

pisica prinde S

pisica prinde pisica

Cum reprezentăm o stare?

Maimuța și banana

Relația maimuță-cutie:

MC-departe = Maimuța se află departe de Cutie

MC-lângă = Maimuța se află lângă Cutie

MC-pe = Maimuța se afla pe Cutie

MC-sub = Maimuța de află sub Cutie

Relația Cutie – Banană:

CB-lateral = Cutia este așezată lateral față de Banană

CB-sub = Cutia este așezată sub Banană

Relația Maimuța – Banană:

MB-departe = Maimuța se află departe de Banană

MB-aproape = Maimuța se află aproape de Banană

MB-ține = Maimuța ține Banana

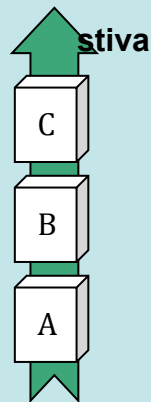
Starea inițială: **MC-departe**, **CB-lateral**, **MB-departe**.

Starea finală: **MB-ține**

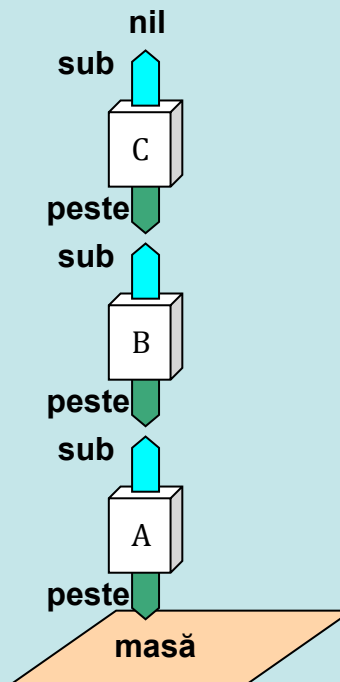
o colecție de predicate

Cum reprezentăm o stare?

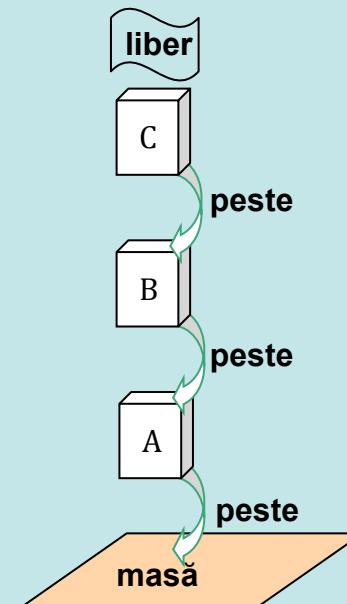
Lumea cuburilor



ordine definită global



ordine definită prin vecinătăți locale



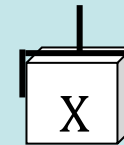
ordine definită prin relații dintre obiecte

Cum reprezentăm o stare?

Lumea cuburilor – reprezentarea configurației brațului



mâna-liberă



mâna-ține(X)

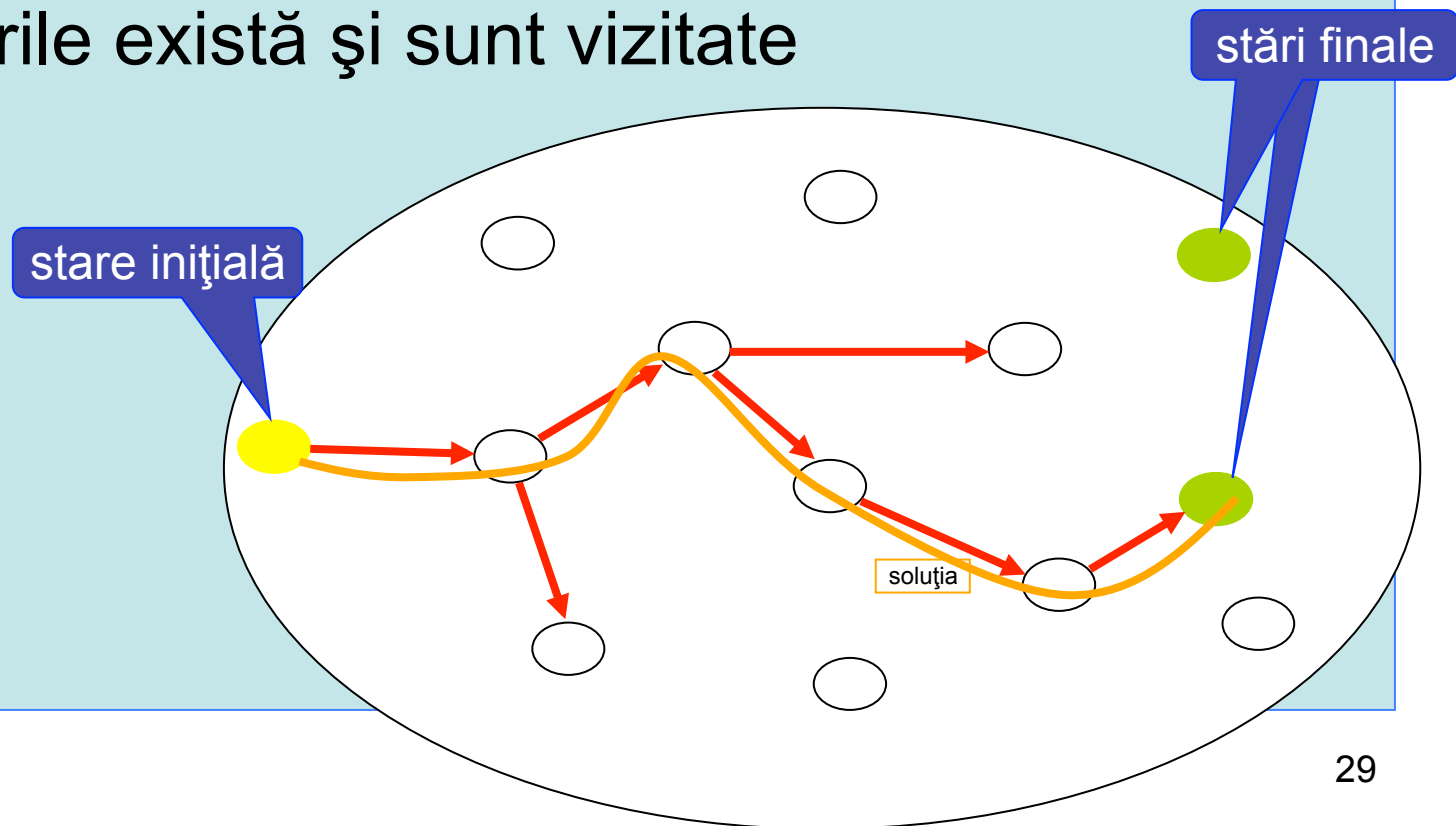
o colecție de predicate

Cum reprezentăm tranzițiile dintre stări

Pasul 4

Două moduri de a vedea o navigare în
spațiul stărilor:

- stările există și sunt vizitate

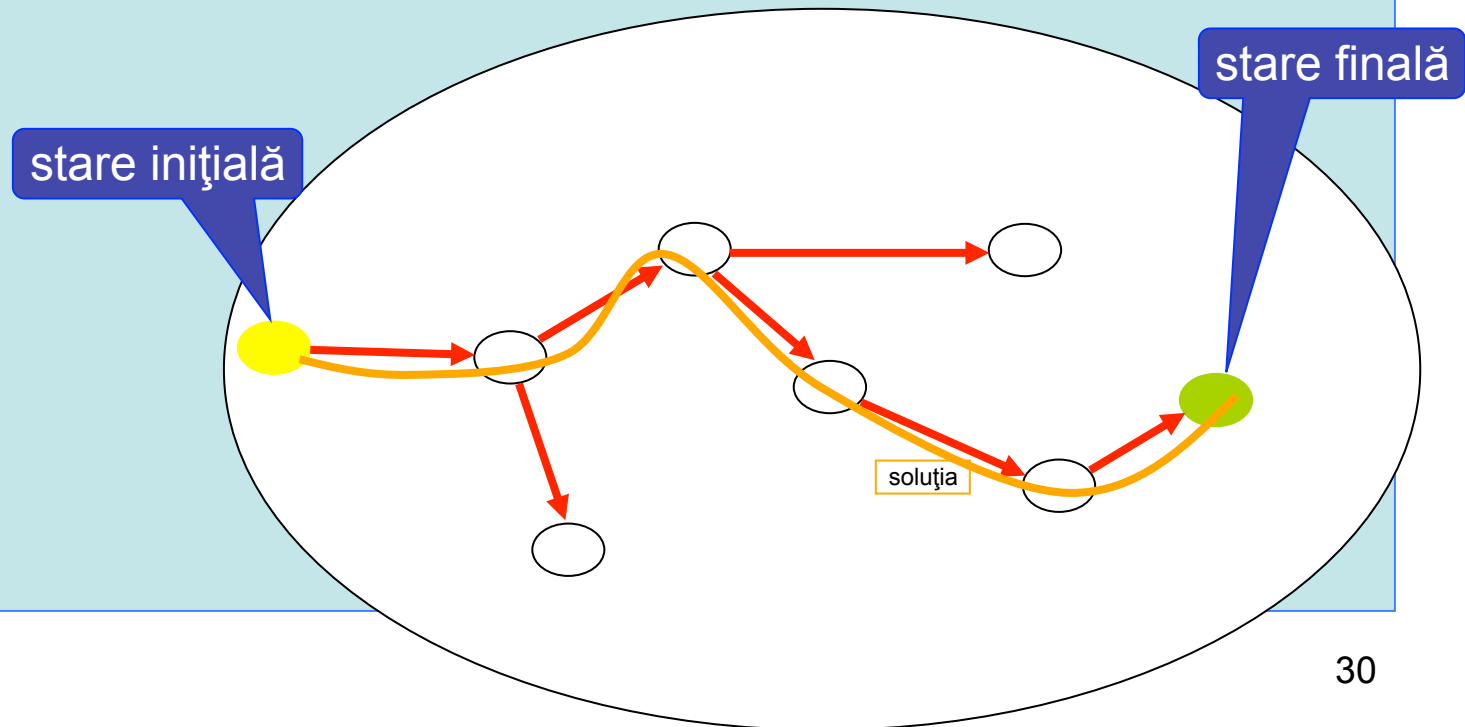


Cum reprezentăm tranzițiile dintre stări

Pasul 4

Două moduri de a vedea o navigare în
spațiul stărilor:

- stările sunt generate la momentul vizitării

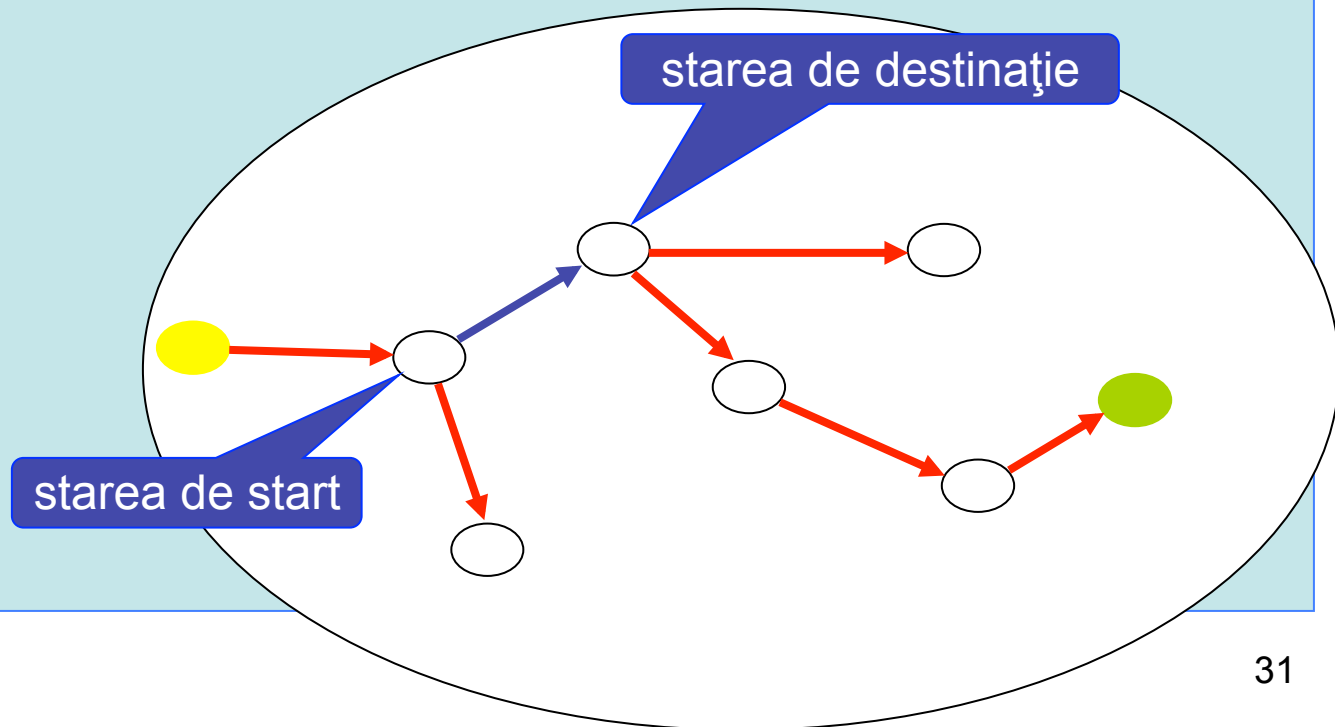


Cum reprezentăm tranzițiile dintre stări

Pasul 4

Un operator verifică condiții și produce transformări în stare

```
if <condiții> then <acțiuni>
```



Cum reprezentăm tranzițiile dintre stări

Pasul 4

Șah:

regula salt-dublu-pion-din-a

DACĂ

pion în poziția (a,2) și

poziția (a,3) e liberă și

poziția (a,4) e liberă

ATUNCI

mută pionul din poziția (a,2) în poziția (a,4)

8 reguli de acest fel...

Cum reprezentăm tranzițiile dintre stări

Pasul 4

Șah:

regula salt-dublu-pion(x)

DACĂ

pion în poziția (x,2) și
poziția (x,3) e liberă și
poziția (x,4) e liberă

ATUNCI

mută pionul din poziția (x,2) în poziția (x,4)

O regulă de acest fel!

Două reguli, dacă parametrizez și jucătorul.

Cum reprezentăm tranzițiile dintre stări

Pasul 4

8-puzzle:

Regula mută-piesa-1-sus

DACĂ

piesa 1 nu e lipită de marginea de sus a tablei
și poziția de deasupra e liberă

ATUNCI

schimbă poziția piesei 1 cu a căsuței aflată deasupra ei

8 reguli de acest fel!

x 4 direcții → 32 reguli în total

Cum reprezentăm tranzițiile dintre stări

Pasul 4

8-puzzle:

Regula mută-blanc-sus

DACĂ

blancul nu e lipit de marginea de sus a tablei

ATUNCI

schimbă poziția blancului cu a căsuței aflată deasupra acestuia

O singură regulă de acest fel!

x 4 direcții → 4 reguli în total

Cum reprezentăm tranzițiile dintre stări

Pasul 4

Maimuța și banana:

aflată departe de cutie, maimuța se apropie de cutie:

apropie-MC:

dacă {MC-departe} atunci ȘTERGE{MC-departe}, ADAUGĂ{MC-lângă}

aflată lângă cutie, maimuța se depărtează de cutie:

depărtează-MC:

dacă {MC-lângă} atunci ȘTERGE{MC-lângă}, ADAUGĂ{MC-departe}

aflată lângă cutie și lateral față de banană, maimuța trage cutia sub banană:

trage-sub-MCB:

dacă {MC-lângă, CB-lateral} atunci ȘTERGE {CB-lateral}, ADAUGĂ{CB-sub}

aflată lângă cutie și sub banană, maimuța trage cutia de sub banană: **trage-lateral-MCB;**

aflată lângă cutie, maimuța se urcă pe ea: **urcă-MC;**

aflată pe cutie, maimuța coboară de pe ea: **coboară-MC;**

aflată lângă cutie, maimuța își urcă cutia deasupra capului: **urcă-pe-cap-MC;**

din postura în care maimuța ține cutia deasupra capului, maimuța își dă jos cutia de pe cap:

coboară-de-pe-cap-MC;

aflată pe cutie și sub banană, maimuța apucă banana: **apucă-MB.**

Cum reprezentăm tranzițiile dintre stări

Pasul 4

Sisteme de reguli STRIPS

stările reprezentate ca set de predicate
(caracteristici)

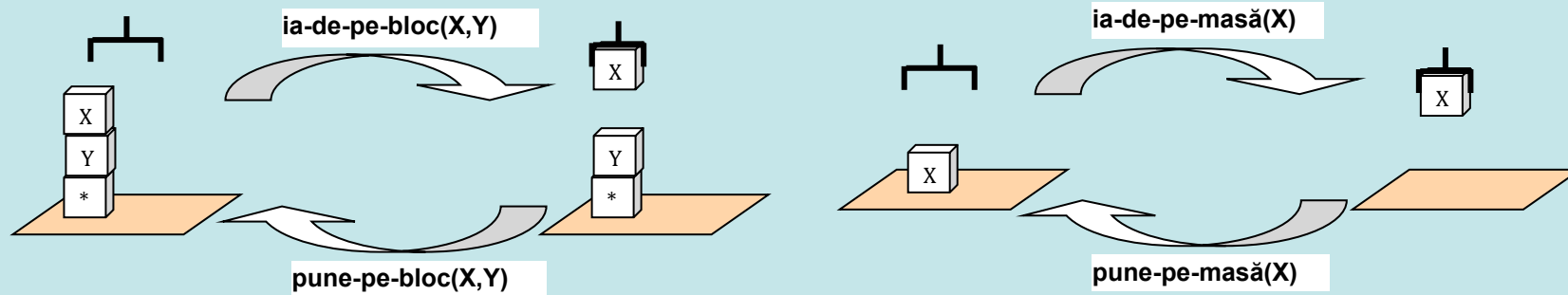
regulile:

```
if <lista-precondiții> then <lista-ștergeri>  
    <lista-adăugări>
```

Cum reprezentăm tranzițiile dintre stări

Pasul 4

Sisteme de reguli STRIPS în lumea cuburilor



Cum reprezentăm tranzițiile 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 ȘTERGE{peste(X, Y), liber(X), mâna-liberă} ADAUGĂ{liber(Y), mâna-ține(X)}

ia-de-pe-masă(X):

dacă {peste(X, masă), liber(X), mâna-liberă} atunci ȘTERGE{peste(X, masă), liber(X), mâna-liberă} ADAUGĂ{mâna-ține(X)}

pune-pe-bloc(X,Y):

dacă {mâna-ține(X), liber(Y)} atunci ȘTERGE{mâna-ține(X), liber(Y)} ADAUGĂ{peste(X, Y), liber(X), mâna-liberă}

pune-pe-masă(X):

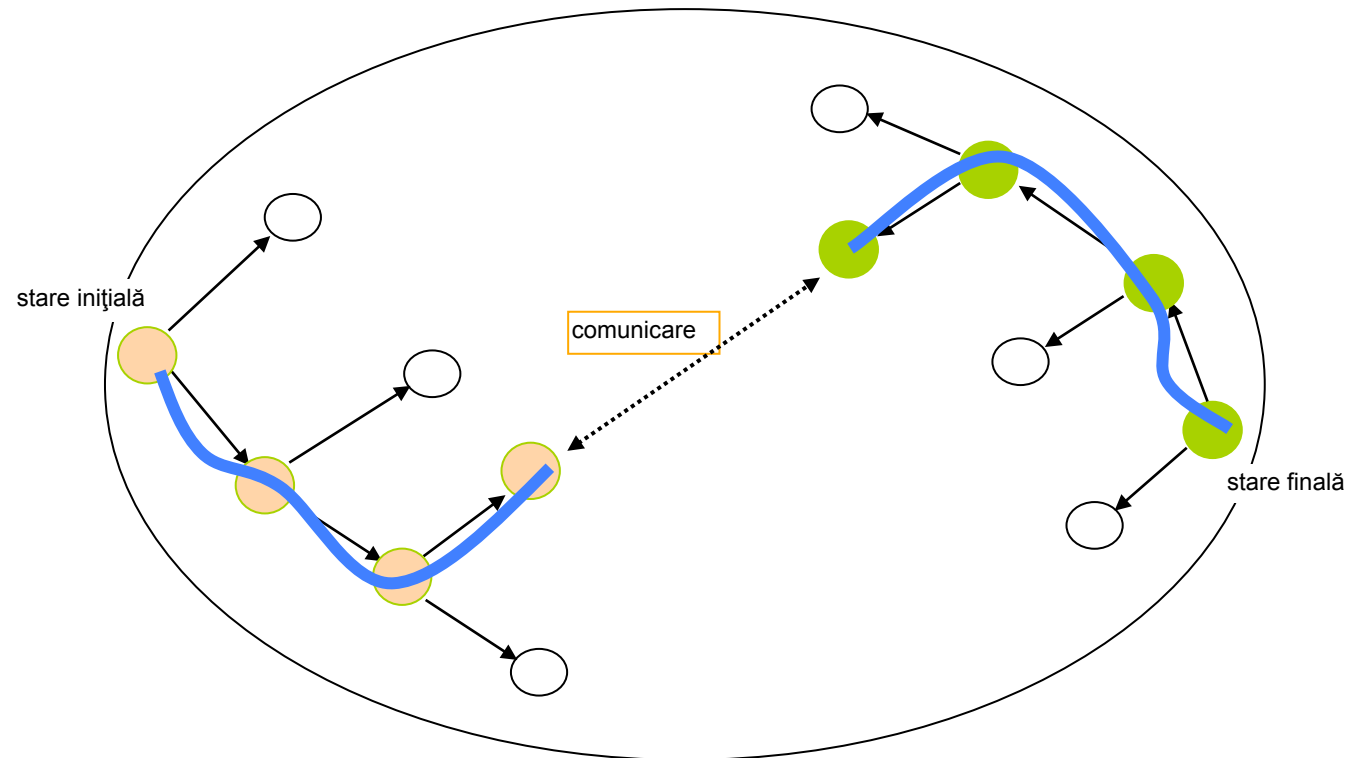
dacă {mâna-ține(X)} atunci ȘTERGE{mâna-ține(X)} ADAUGĂ{peste(X, masă), liber(X), mâna-liberă}

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*)

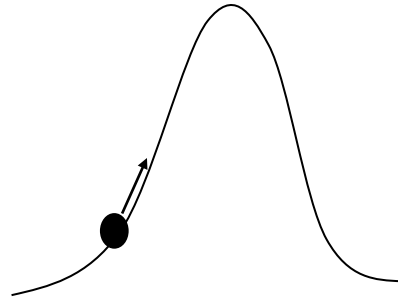
Căutare în spațiul stărilor

- Căutare bidirecțională sincronă

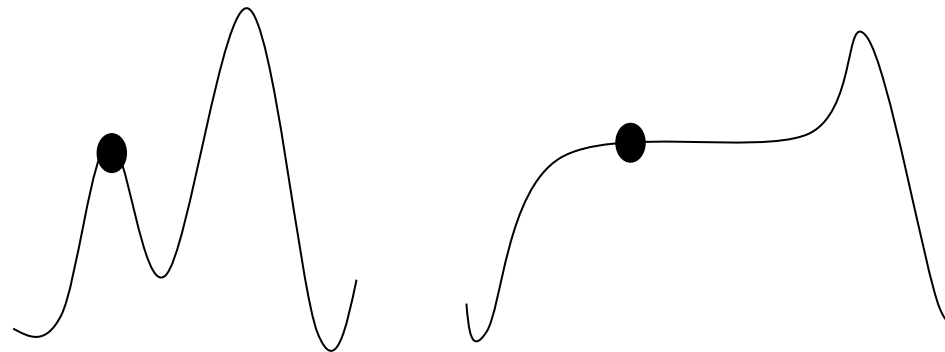


Strategii irevocabile: *hill-climbing*

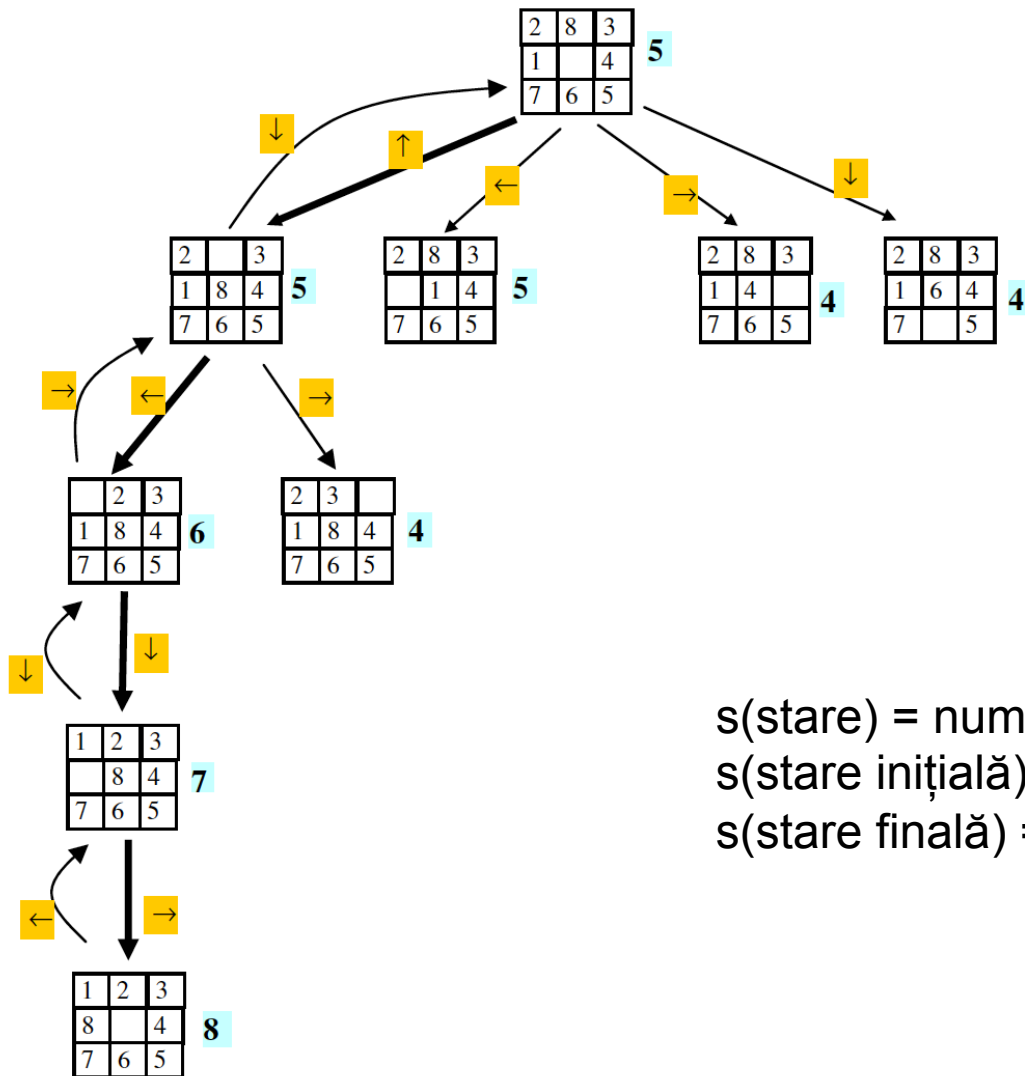
- Cale de întoarcere nu există
 - o funcție apreciază apropierea de soluție



- pericole: maxime locale, platouri



8-puzzle: o parte a arborelui de căutare în *hill-climbing*



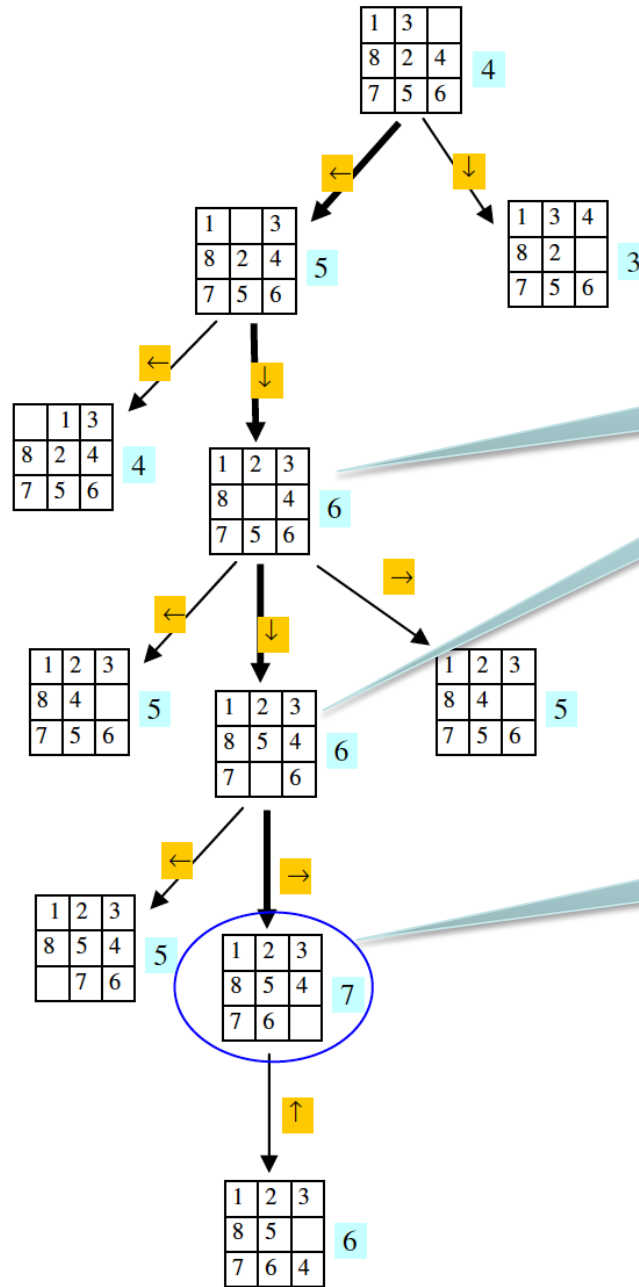
Starea inițială



Starea finală

$s(\text{stare}) = \text{numărul de piese aflate în poziții finale}$
 $s(\text{stare inițială}) = 5$
 $s(\text{stare finală}) = 8.$

8-puzzle: căutare întreruptă de un maxim local



platou

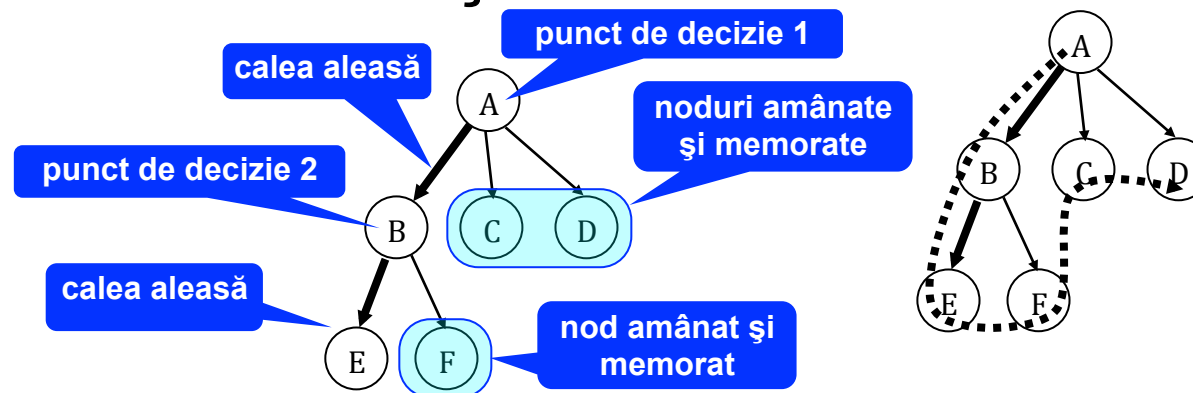
maxim local

Hill climbing

```
procedure hill-climbing(initial-state)  
begin  
  current-state <- initial-state;  
  while(current-state) {  
    if (current-state e stare finală) return current-state;  
    all-new-neighbour-states <- setul stărilor ce pot fi obținute din  
current-state prin operatorii aplicabili ei;  
    all-new-neighbour-states <- all-new-neighbour-states minus toate  
stările deja vizitate;  
    sortează all-new-neighbour-states în ordinea descrescătoare a  
valorilor funcției de cost;  
    all-new-neighbour-states <- all-new-neighbour-states minus stările  
de valoare mai mică decât a lui current-state;  
    if (all-new-neighbour-states ≠ ∅) current-state <- prima clasată  
în all-new-neighbour-states);  
    else return FAIL;  
  }  
  return fail;  
end
```

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 tranziția



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

Backtracking hill-climbing

```
procedure backtracking-hill-climbing(initial-state)  
begin  
  heap <- initial-state ° ∅;  
  solution <- ∅;  
  while(heap) {  
    current-state <- first(stack);  
    heap <- rest(heap);  
    solution <- solution ° current-state;  
    if (current-state e stare finală) return solution;  
    all-new-neighbour-states <- setul stărilor ce pot fi  
    obținute din current-state prin operatorii aplicabili ei;  
    all-new-neighbour-states <- all-new-neighbour-states \  
    toate stările deja vizitate;  
    heap <- heap ° all-new-neighbour-states;  
    sort heap descendent după valorile funcției cost;  
    heap <- heap \  
    stările de valori mai mici decât a lui  
    current-state;  
  }  
  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,  $\emptyset$ ); solution <-  $\emptyset$ ;
  while (stack not empty)
  { node <- pop(stack);
    solution <- solution  $\circ$  node;
    if goal(node) then return solution;
    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, ∅); solution <- ∅;
  while (queue not empty)
  { node <- out(queue);
    solution <- solution ° node;
    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, ∅); solution <- ∅;
  while (list not empty)
  { node <- get-first(list);
    solution <- solution ° node;
    if goal(node) then return solution;
    else
      { include(node's successors, list);
        sort list descending;
      }
  }
  return FAIL;
end
```

Exemplu: *best-first search*