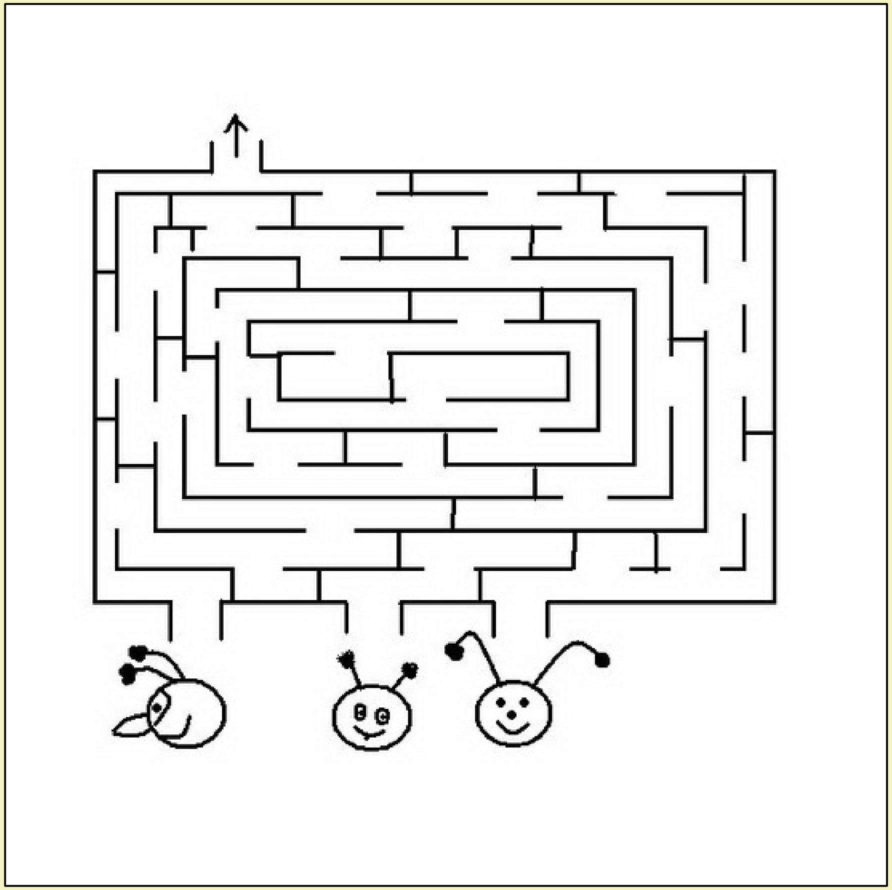
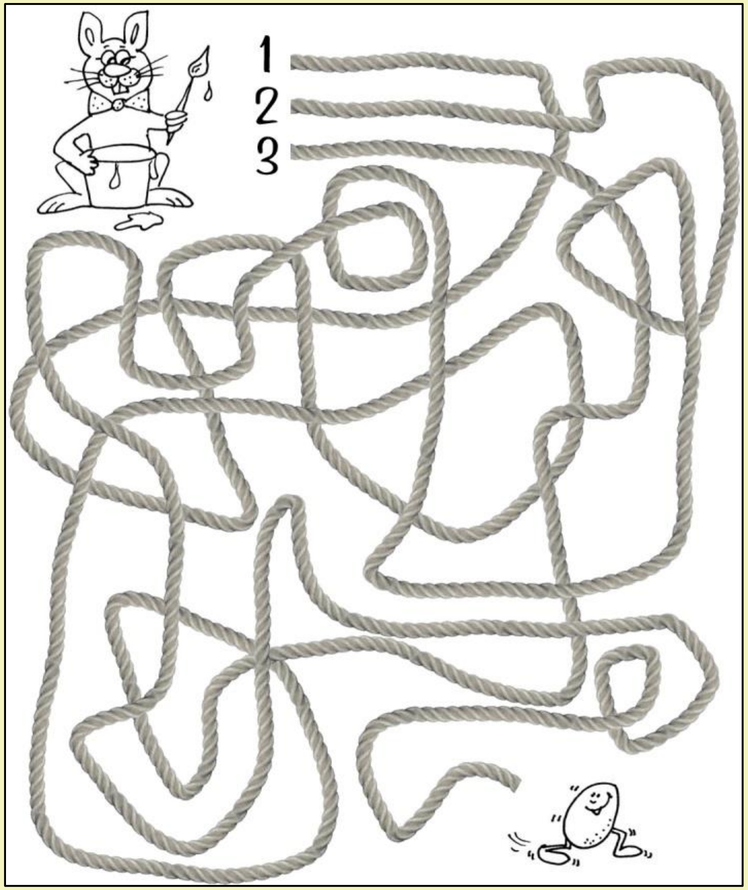
SEARCH
STARTING FROM THE GOAL

1. Backward search
2. Reduction
3. Decomposition

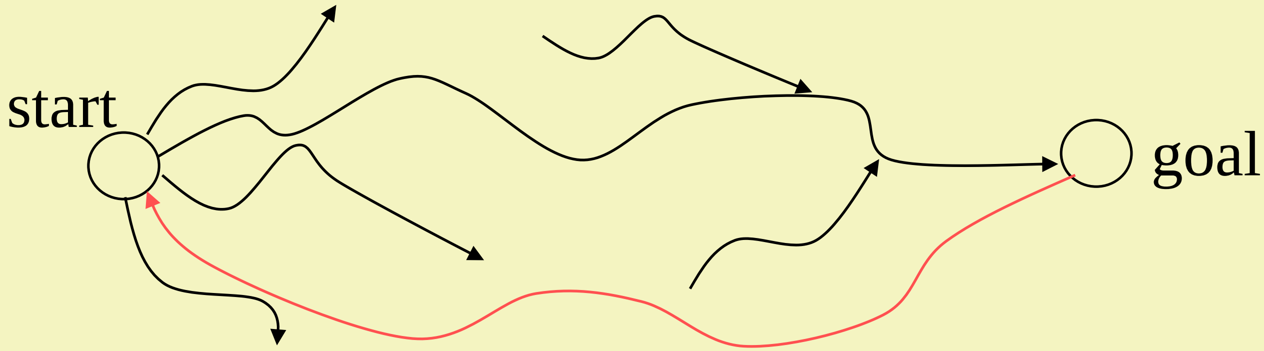
1. Backward search





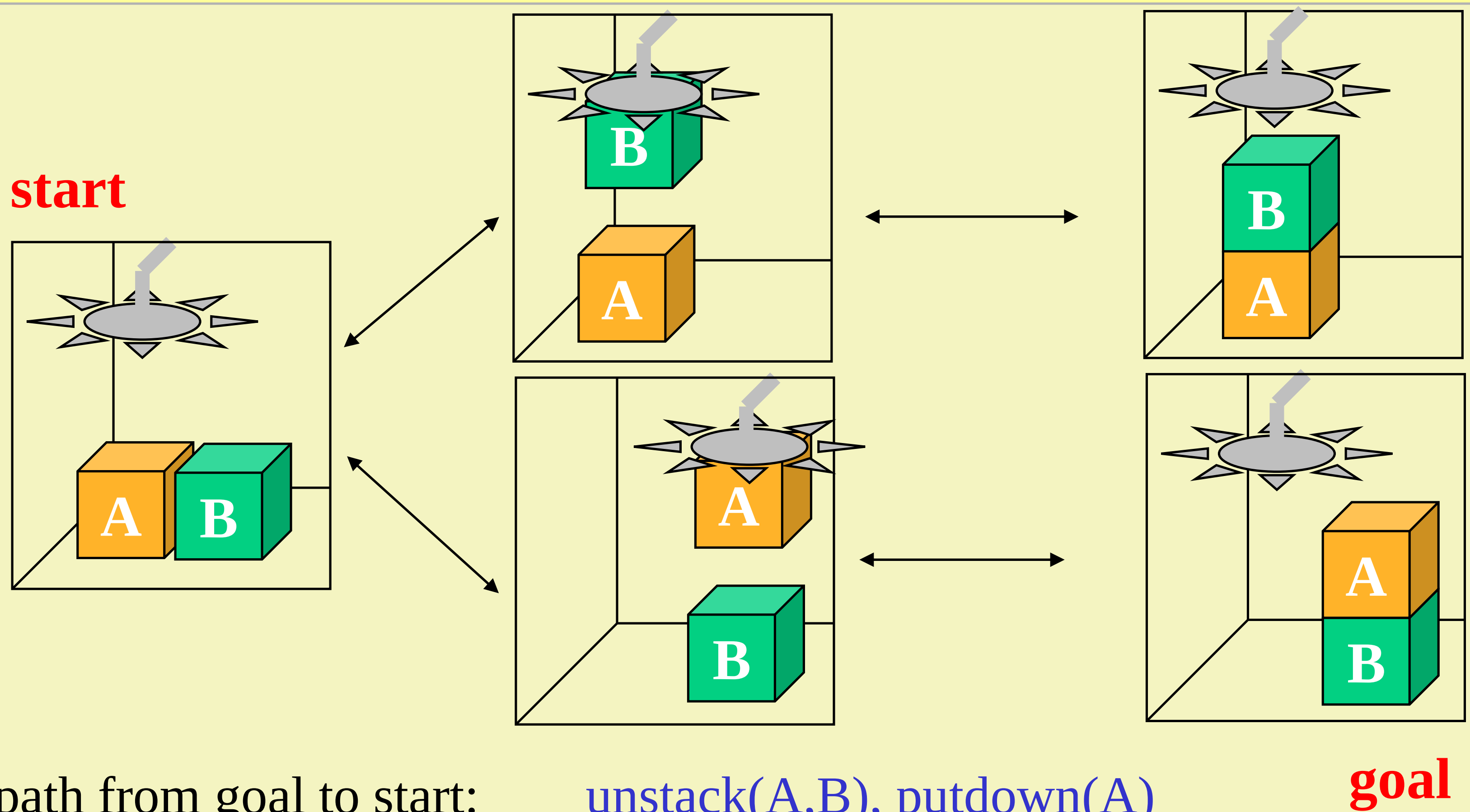
Solution of backward search

- If the representation graph shows fewer alternative paths looking from the goal than from the start, the problem may be solved faster by a backward search rather than a forward one.



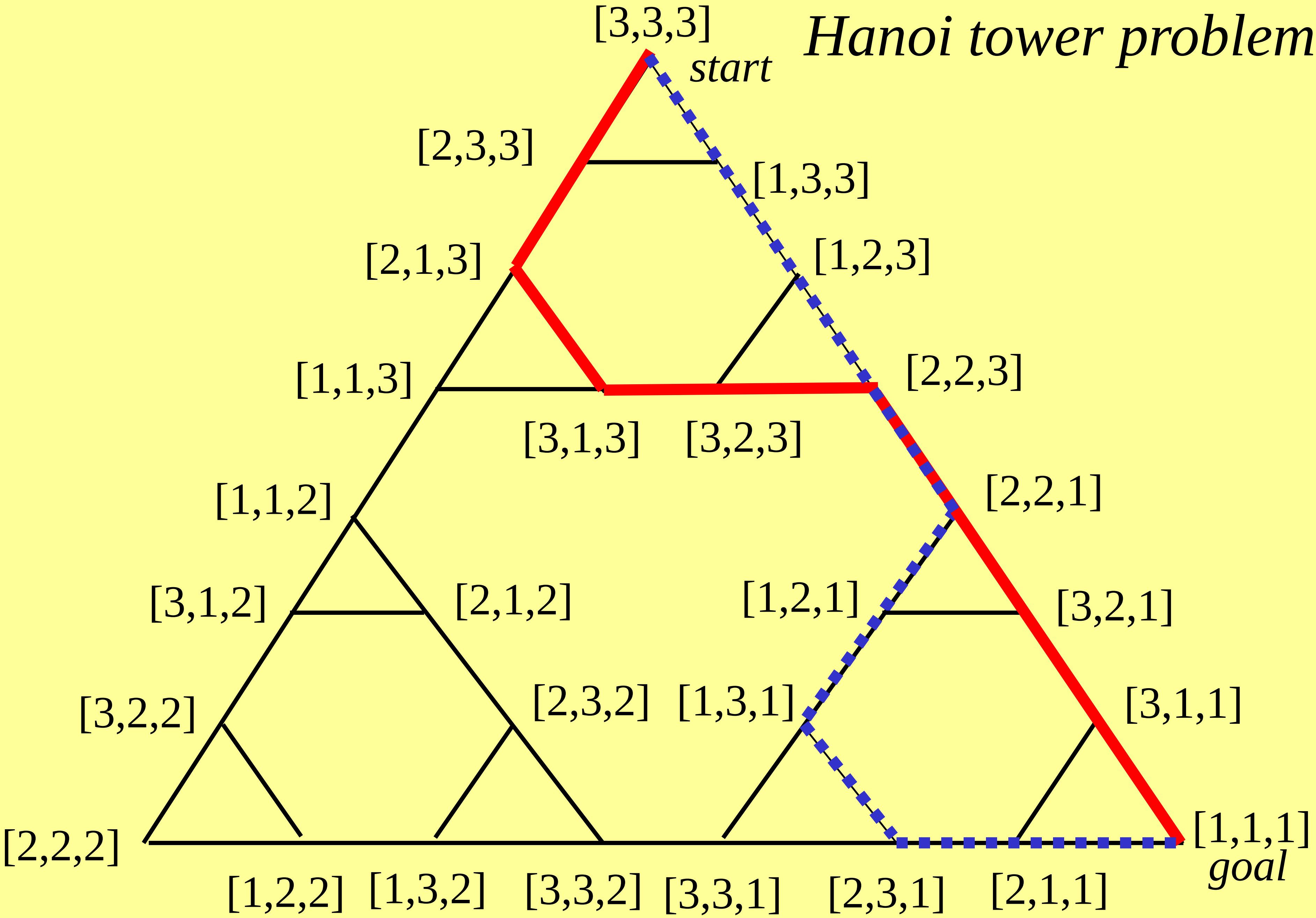
- When this backward search successfully terminates, it results in a path from the goal to the start. However this path is not the solution, we need the inverse of this path.

Block world problem

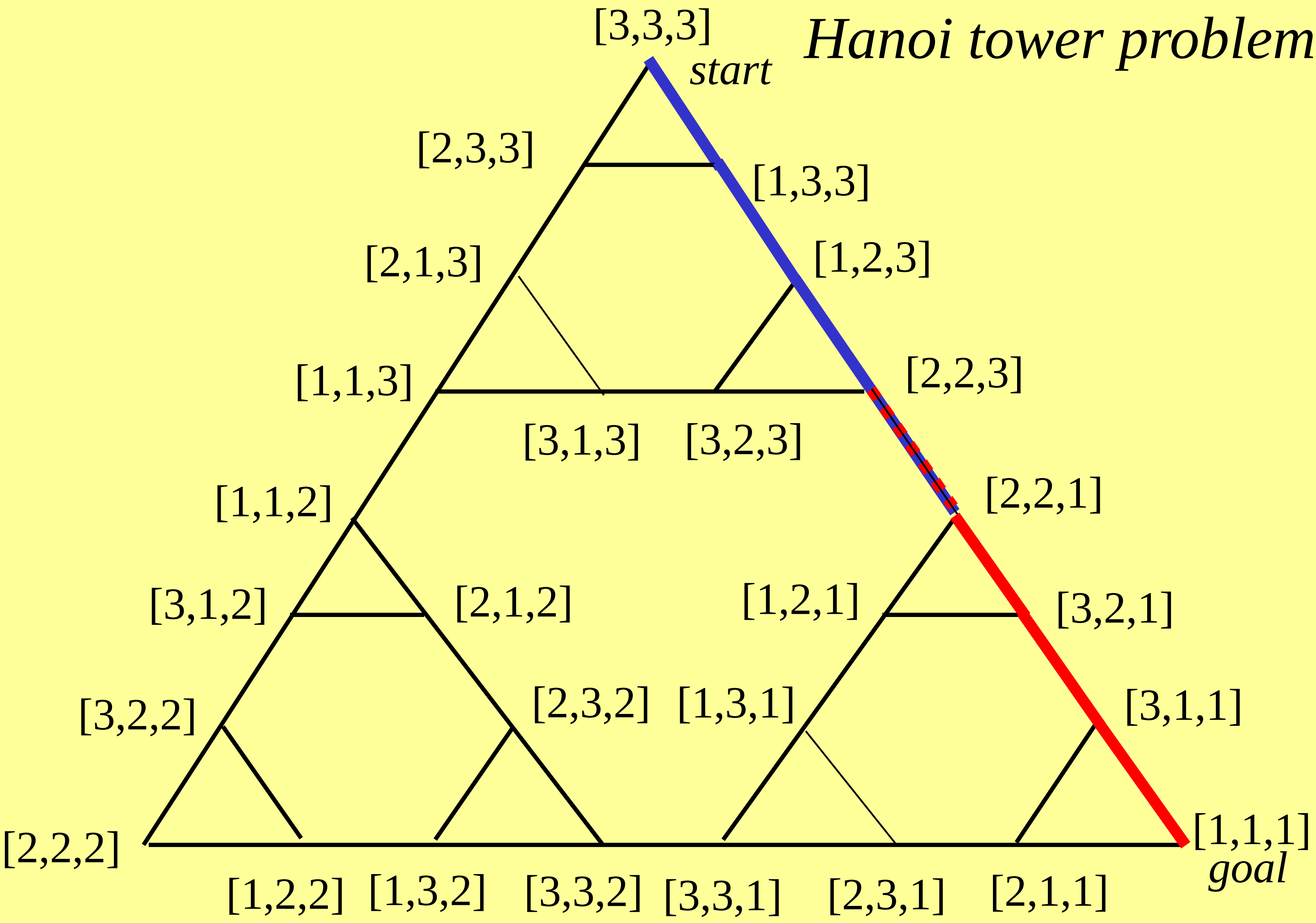


path from goal to start: unstack(A,B), putdown(A)
solution from start to go: pickup(A), stack(A,B)

Backward hill-climbing method over Hanoi tower problem



Bidirectional hill-climbing method on Hanoi tower problem



Jugs' problem and its state-space model

Given a 5-liter jug filled with wine and an empty 3-liter and a 2-liter jugs. Let's obtain precisely 1 liter wine in the 2-liter jug.

<u>State-space</u>: $\boldsymbol{SP} = map(key{:}\mathbb{N}, value{:}\mathbb{N})$ where $Keys=\{5,3,2\}$

invariant: $\Sigma_{i\in[5,3,2]}\ this[i] = 5$ and $\forall i\in[5,3,2]: this[i]\leq i$ $\quad$ ($this$: $\boldsymbol{SP}$)

<u>Initial</u>: $[5, 0, 0]$ — this[5]=5, this[3]=0, this[2]=0

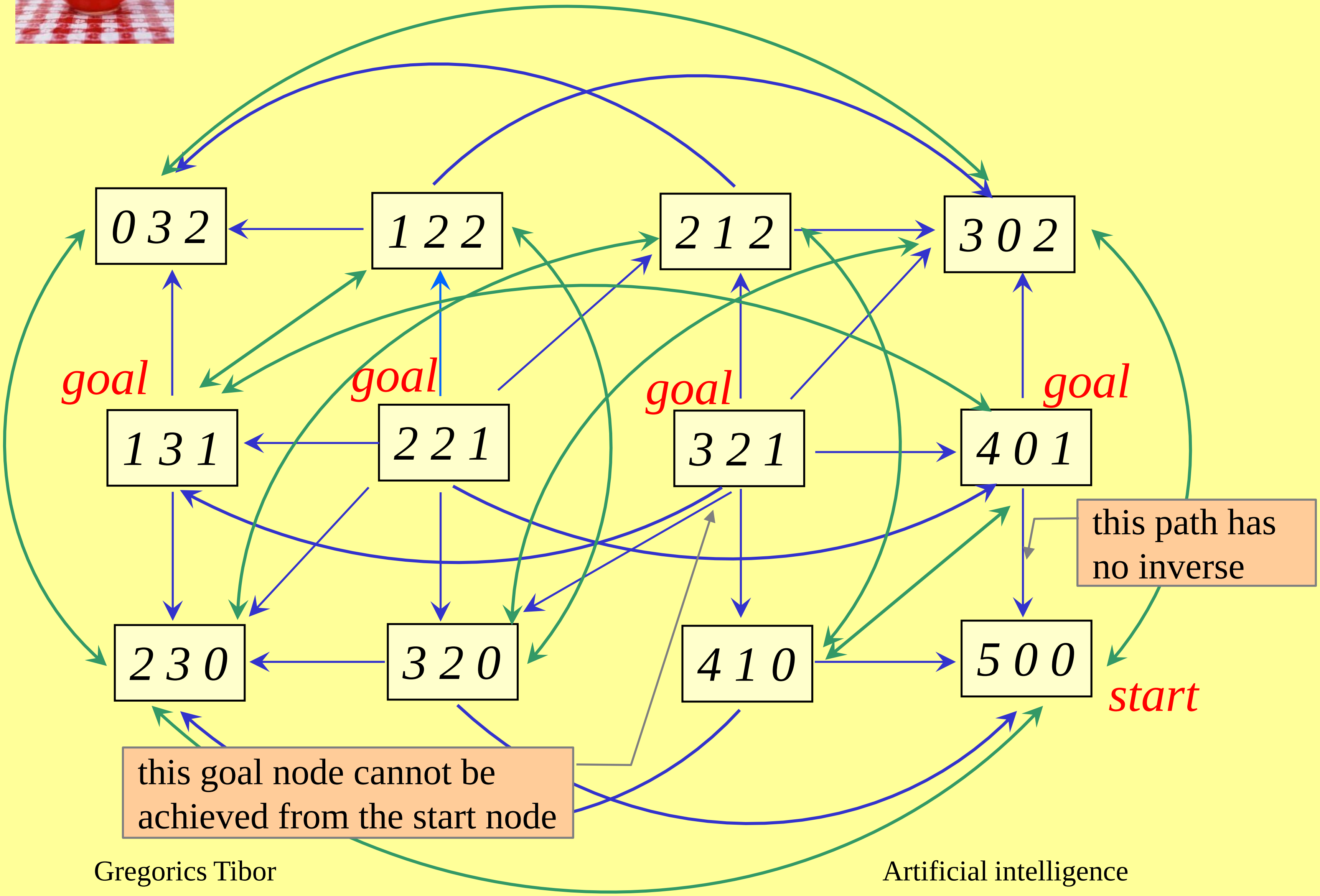
<u>Final</u>: $[x, y, 1]$ — this[2]=1

<u>Operator</u>: $\textbf{Decant}(i,j)$: $\boldsymbol{SP} \to \boldsymbol{SP}$

IF $\quad i,j\in\{5,3,2\}$ and $i\neq j$ and $min(this[i], j-this[j])>0$

THEN $\quad this[i],this[j] := this[i] - min(this[i], j-this[j]),$
$\quad\quad this[j] + min(this[i], j-this[j])$

State graph of jug's problem

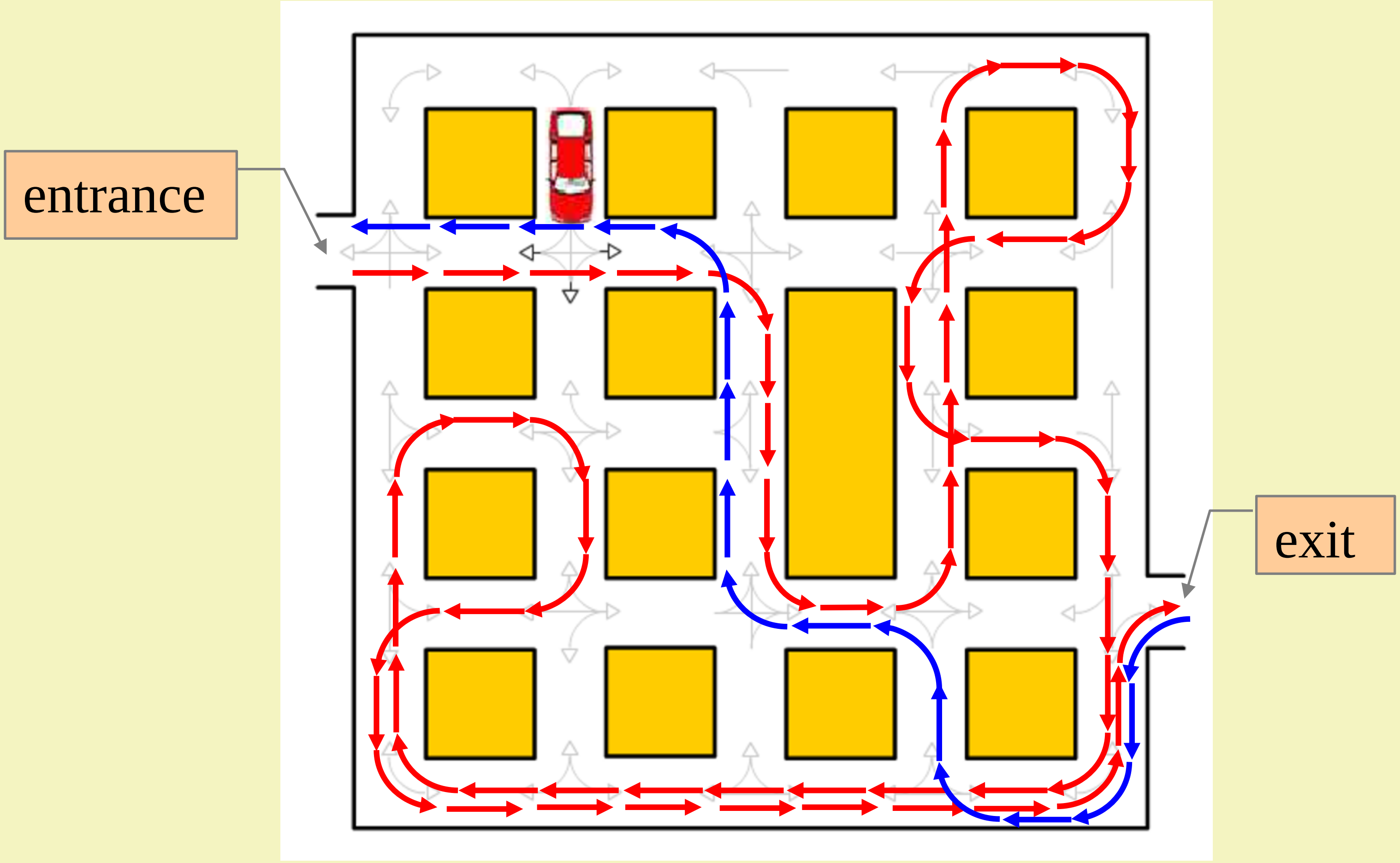


Preconditions of backward search

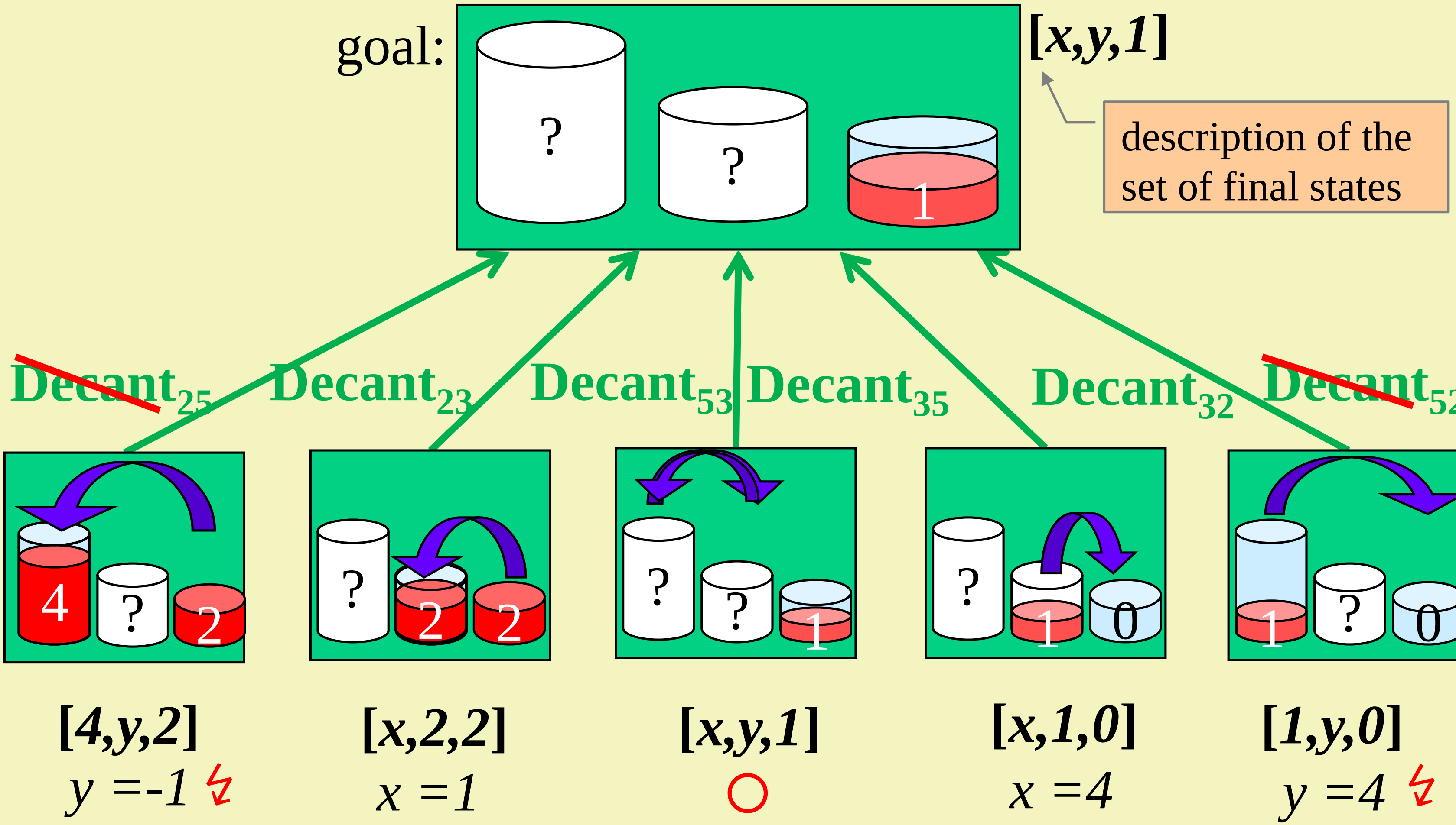
1. The arcs of the representation graph must be bidirectional (or at least the arcs of the path found from the goal to the start)
   - It means that in case of using state-space model, the operators must have got inverse.
2. The goal that can be achieved from the start must be known.

What can we do if one of these conditions does not hold but a backward like search is needed to apply?

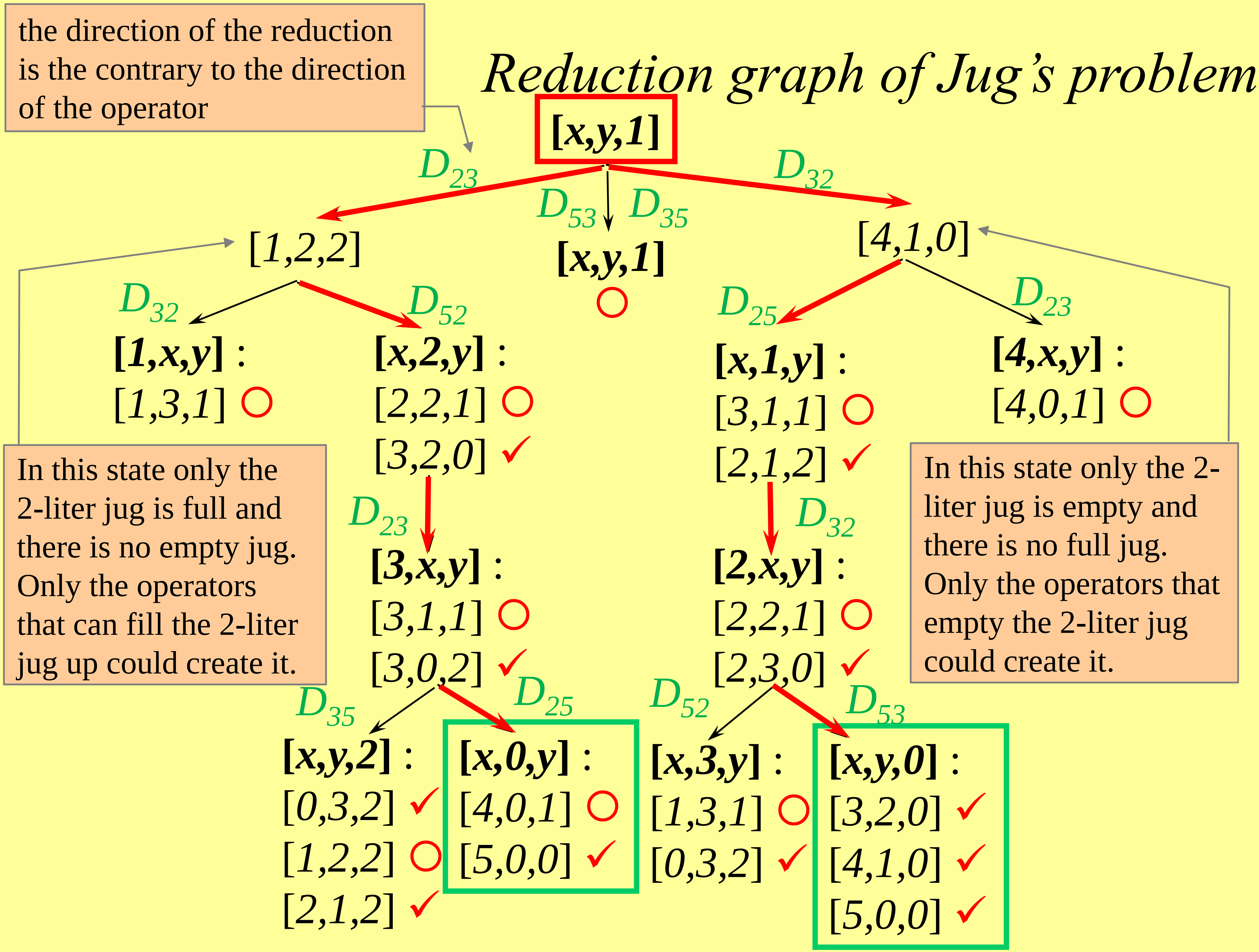
2. Problem reduction



Reduction of Jug's problem



Reduction graph of Jug's problem



Reduction step of Jug's problem

invariant: $\Sigma_{i\,\in[5,3,2]}$ $this[i] = 5$ and $\forall i\in[5,3,2]$: $this[i]\leq i$

Decant(i,j): ***SP*** $\rightarrow$ ***SP***

IF $i,j\in\{5,3,2\}$ and $i\neq j$ and $min(this[i], j-this[j])>0$

THEN $this[i] := this[i] - min(this[i], j-this[j])$

$this[j] := this[j] + min(this[i], j-this[j])$

power set of *SP*

the part of $\mathcal{D}_{\boldsymbol{Decant}(i,j)}$ referring to parameters i,j

$R_{\boldsymbol{Decant}(i,j)} : 2^{SP} \longrightarrow 2^{SP}$ where $i,j\in[5,3,2]$ and $i\neq j$

$\forall B \in 2^{SP}\backslash\emptyset$:

$R_{\boldsymbol{Decant}(i,j)}(B)= \{\ a\in SP\ |$

invariant of the state space model

$\Sigma_{i\,\in[5,3,2]}$ $a[i] = 5$ and $\forall i\in[5,3,2]$: $a[i] \leqq i$ and

$min(a[i], j-a[j])>0$ and

the part of $\mathcal{D}_{\boldsymbol{Decant}(i,j)}$ referring to *this* (now *this* is *a*)

$\forall b\in B : b[i]=a[i]-min(a[i], j-a[j])$ and

$b[j]=a[j]+min(a[i], j-a[j])$ and

$b[k]=a[k]$ (where $k\neq i$ és $k\neq j$) }

effect of $\boldsymbol{Decant}_{ij}$ where input: *a* (*this*), output: *b* (new *this*)

Problem reduction model

- There is given a state-space model: state-space (SP), invariant ($Inv{:}SP\longrightarrow\mathbb{L}$), initial and final states, operators ($M{:}SP\longrightarrow SP$).
- A subset of the states (2^{SP}) can be given with description.
- Each operator $M{:}SP\longrightarrow SP$ is corresponded to a reduction operator $R_M : 2^{SP}\longrightarrow 2^{SP}$
  - $\mathcal{D}_{\boldsymbol{R_M}} = \{B\in 2^{SP} \mid B \neq \emptyset \}$
  - $\forall B\in \mathcal{D}_{\boldsymbol{R_M}} : R_M(B) = \{ a\in SP \mid Inv(a) \text{ and } M(a)\in B \}$
    - if $R_M(B) = \emptyset$ then $R_M(B)$ is inconsistent.
- The final description gives a set of final states (usually all of them).
- The initial descriptions contain at least one initial state.

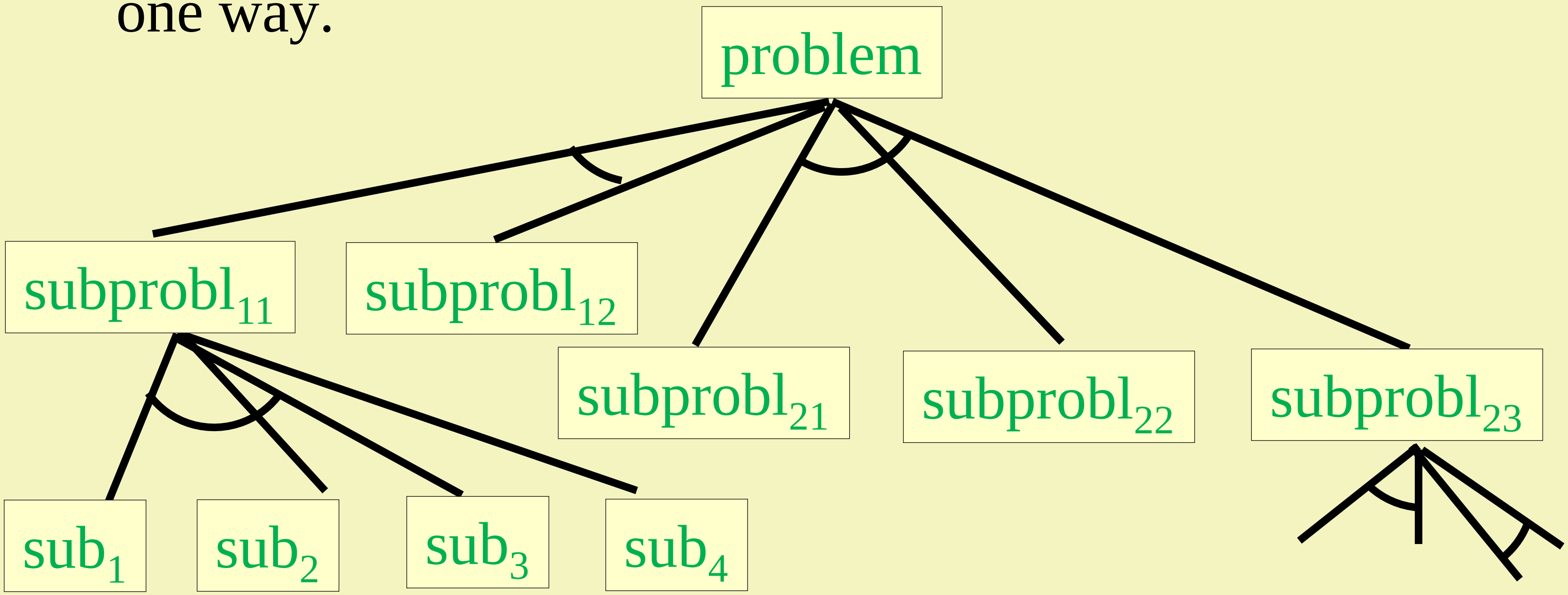
Remarks

- Our aim is to find the sequence of reduction operators that leads from the final description to any initial description. These operators are labelled by state-space operators.
- The solution is the reverse order of this sequence of labels.
- The problem reduction can be modeled with a directed graph

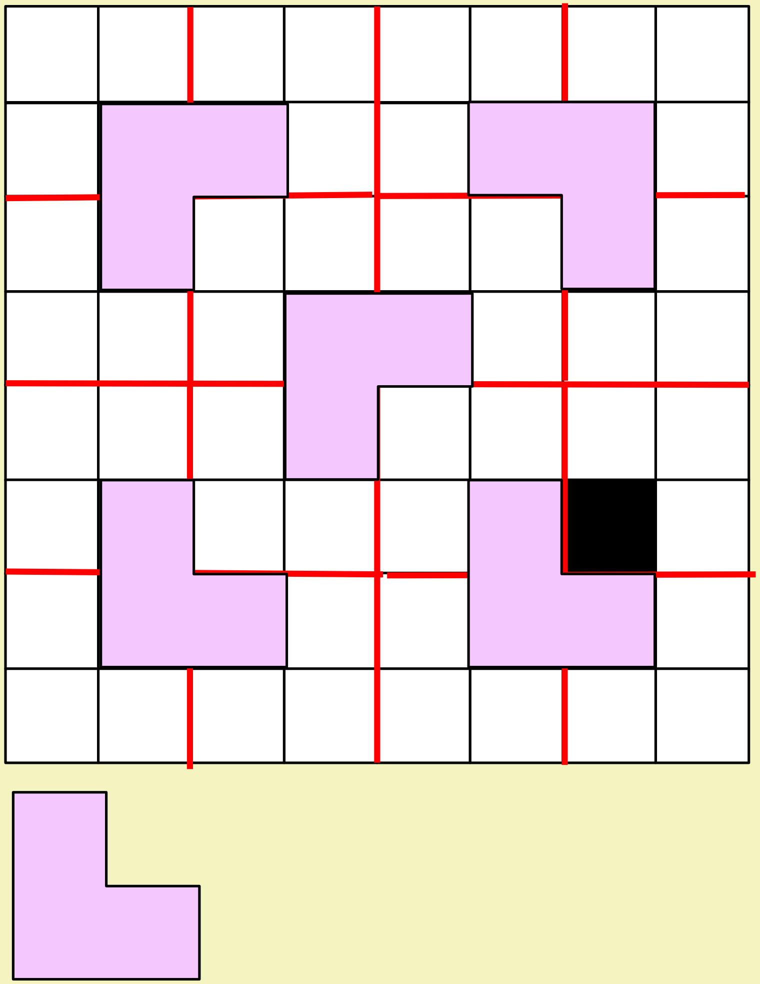
node	~	description (set of states)
arc labeled by an operator	~	reduction
start node	~	final description
goal node	~	initial description
path from start to goal	~	solution in reverse order

3. Problem decomposition

- During the problem decomposition one problem is divided into more subproblems, and these subproblems are further divided until easily solvable problems have been got.
- A problem might be divided into subproblems in not only one way.

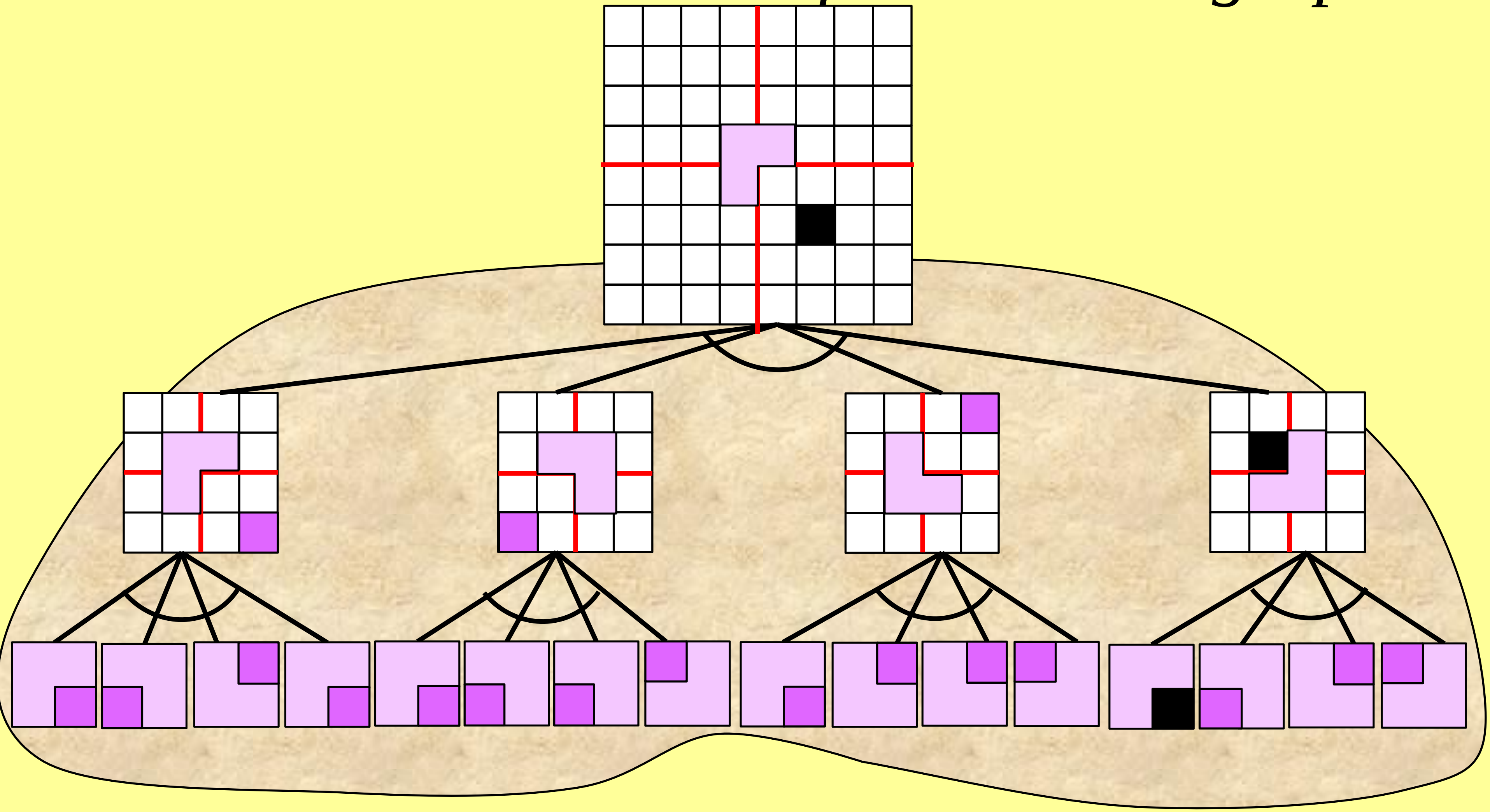


Covering the chess board



- **Problem description**:
  $2^n \times 2^n$ board with 1 hole
- **Original problem**:
  8×8 board with 1 hole
- **Primitive problem**:
  2×2 board with 1 hole
- **Operator**: *divides the board into 4 equal areas and takes an L shape tile in a such way that it would cover three squares: one from each area that does not contain the hole.*

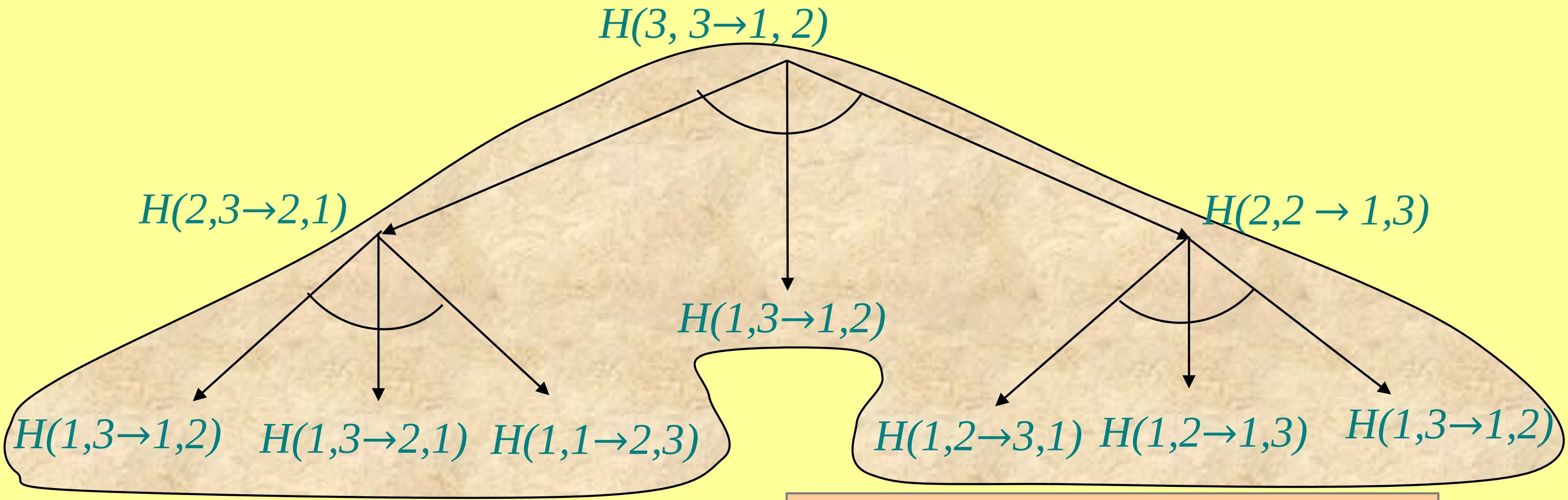
Representation graph



Solution graph: resolves the original problem to primitive problems

Solution: nodes of the solution tree show the placement of the L shaped tiles

Decomposition of Hanoi tower problem



- **Problem description** : $H(n, i\to j, k)$
- **Original problem** : $H(3, 3\to 1, 2)$
- **Primitive problem** : $H(1, i\to j, k)$
- **Decomposing operator** : $H(n, i\to j, k)$ is divided into $H(n-1, i\to k, j)$, $H(1, i\to j, k)$, $H(n-1, k\to j, i)$

n discs must be moved from the peg i to the peg j with the peg k.

easy to decide whether it is solvable

- **Solution graph**: tree resolving the original problem to primitive problems
- **Solution**: the leaves of the solution tree form left to right

Concept of problem decomposition

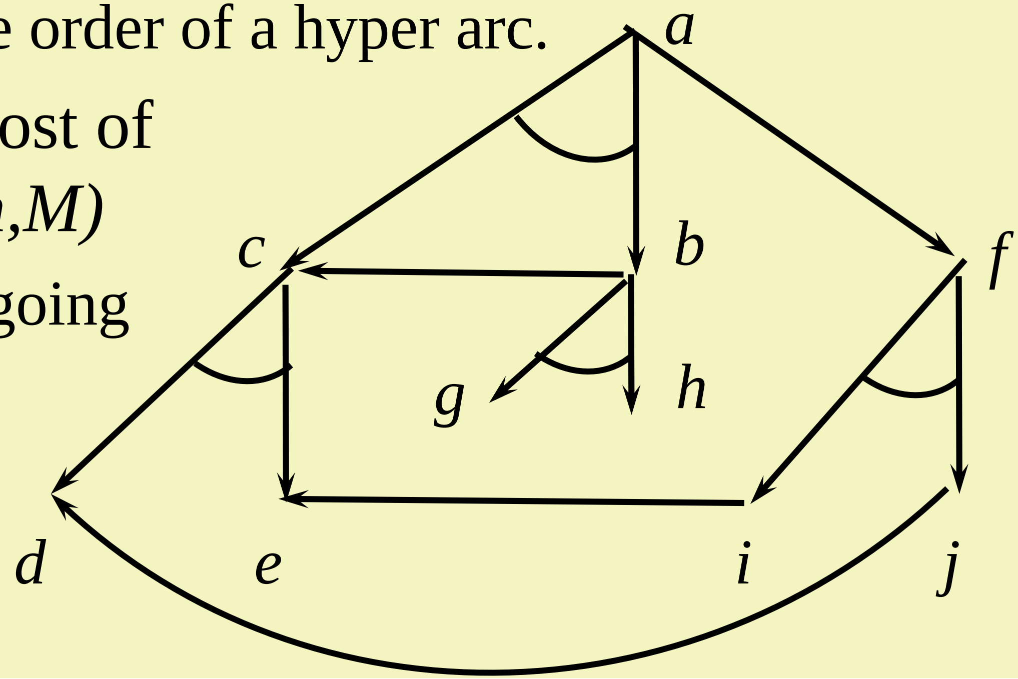
- The model of the problem decomposition contains:
  - general description of the subproblems,
  - description of the original problem,
  - description of the primitive problems (these are simple to decide whether they can be solved, and their solution can be computed easily),
  - the decomposing operators :
    - D: $problem \rightarrow problem^{+}$ and
      $D(p)=<p_1, \ldots , p_n>$

Representation graph of the problem decomposition model

- The decomposition model can be described with a so-called AND/OR graph ($R = (N,A)$) that is mostly a tree where
  - (N) the nodes represent the subproblems
  - (s) start node (root) is the original problem
  - (T) goal nodes (some leaves) are the solvable primitive problems
  - (A) a beam of arcs symbolizes the effect of a decomposing operator as it divides a problem into subproblems.
    - The arcs of the same beam are in an 'AND' connection; and there is 'OR' connection between the beams outgoing from the same node.

AND/OR graph

- R=(N,A) is an arc-weighted directed hyper graph where
  - N is the set of nodes,
  - $A \subseteq \{ (n,M) \in N \times N^{+} \mid 0 \neq |M| < \infty \}$ is the set of hyper arcs. $|M|$ is the order of a hyper arc.
  - $c(n,M)$ is the cost of the hyper arc (n,M)
- Number of the outgoing hyper arcs from one node is finite
- $0 < \delta \leq c(n,M)$

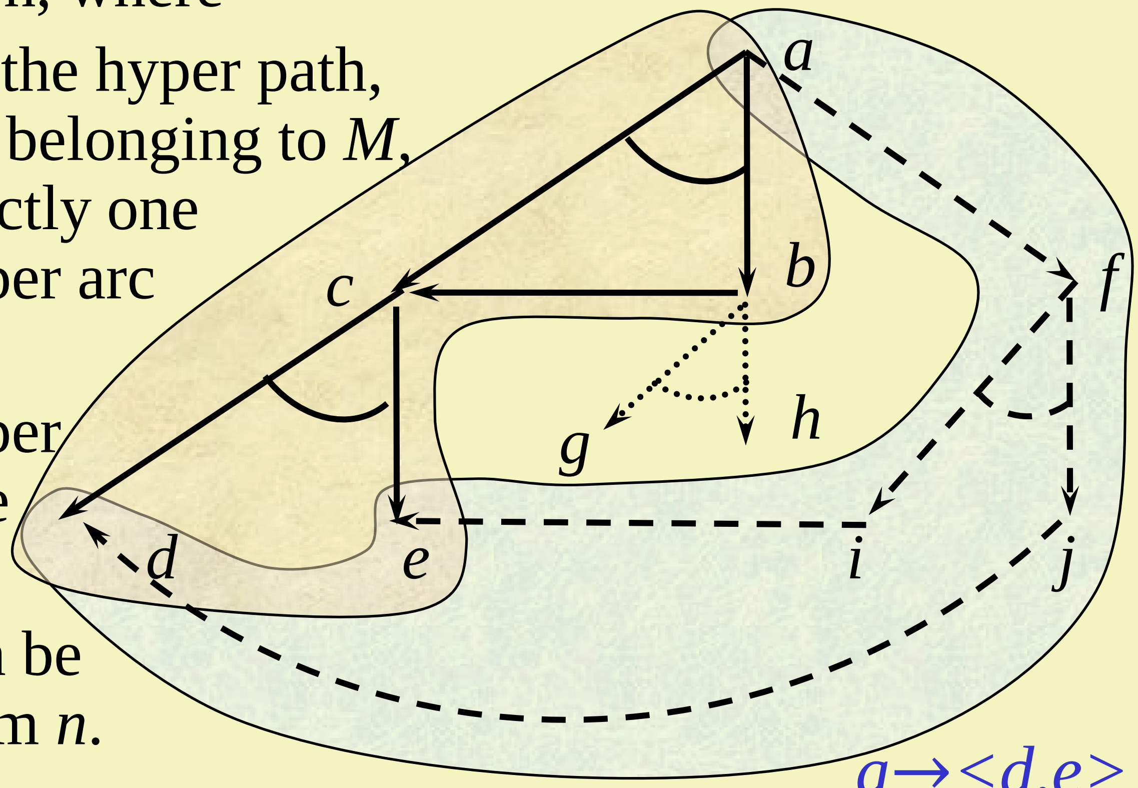


Solution graph

- The solution of a decomposition can be read from a special subgraph (solution graph) of the AND/OR graph that represents only one resolving way of the original problem to a sequence of solvable primitive problems.
  - In the solution graph each node can be achived from the start node via a path, and there is a path from each node to a goal node,
  - If an arc belongs to the solution graph, then all other arcs being in „AND" connection with that very arc also belong to it.
  - There are no „OR" connection between two arcs of the solution graph.

Hyper path
form a node to the sequence of nodes

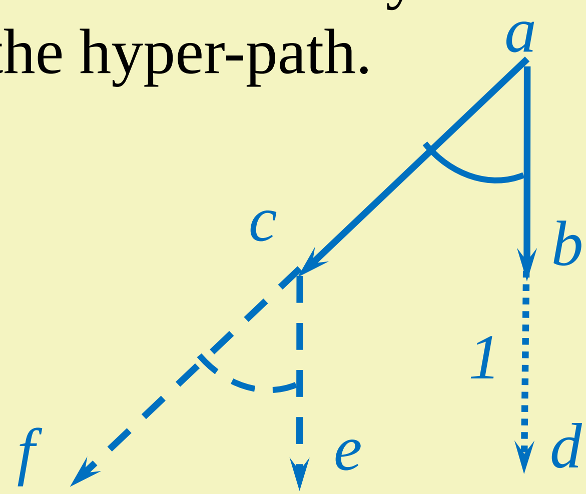
- The hyper path $n^{\alpha}\rightarrow M$ ($n\in N$, $M\in N^{+}$) is a finite subgraph of an AND/OR graph, where
  1. the nodes of the hyper path, except those belonging to M, have got exactly one outgoing hyper arc
  2. there are no outgoing hyper arcs from the nodes of M,
  3. all nodes can be achieved from n.

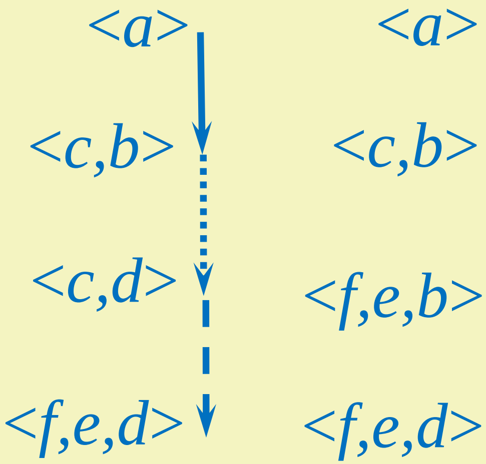


$a\rightarrow<d,e>$

Difference between paths and hyper paths

- The traversal of an ordinary directed path is the sequence of the nodes fitting the path. These nodes can be enumerated in order of the arcs of the path. This order is unequivocal.
- The traversal of a hyper path is also a sequence but its members are sequences of nodes and this traversal is non-deterministic: there may be several enumerations of the hyper arcs of the hyper-path.





Traversal of a hyper path

- The traversal of the hyper path $n \rightarrow M$ (that is the sequence of the sequences of nodes) can be generated as below:
  - the first sequence : $<n>$
  - The sequence C is followed by the sequence $C^{k \leftarrow K}$ (each occurrence of the node k is replaced with the sequence K) if the hyper path has got the hyper arc (k,K) where $k \in C$ but $k \notin M$.
- <u>Remark</u>:
  - A hyper path has got a finite number of finite length traversals.

Search in AND/OR graph

How can we find a solution graph in an AND/OR graph?

- Each AND/OR graph may be corresponded to a δ-graph where the paths driving from the start symbolize the traversals of the hyper paths driving from the start node of the AND/OR graph, and each ordinary solution path represents a traversal of a solution graph.
- This transformation must be built into the path finding algorithms: they gradually discover the traversals outgoing from the start node as ordinary paths of the corresponding δ-graph
- In this way the path finding algorithms over δ-graphs may be adapted to the AND/OR graphs to find solution graph.

Backtracking in AND/OR graph

Recursive procedure *VL2*(*traversal*) **return** *solution*

C := *tail*(*traversal*)
if *all_goal*(*C*) **then** **return**(*nil*) **endif**
if *length*(*traversal*) ≥ limit **then** **return**(*fail*) **endif**
if *C* ∈ *remain*(*traversal*) **then** **return**(*fail*) **endif**
k := *select_non_goal*(*C*)
for ∀(*k*,*K*)∈*outgoing_hyper_arcs*(*k*) **loop**
  solution := *VL2*(*concat*(*traversal*, $C^{k \leftarrow K}$))
  if *solution* ≠ *fail* **then**
    return(*concat* ((*k*,*K*), *solution*)) **endif**
endloop
return(*fail*)
end

Avoiding fake traversals: If the node *k* has been replaced earlier with an hyper arc in the current traversal, then this hyper arc be the only one (*k*,*K*) hyper arc that must be used here.