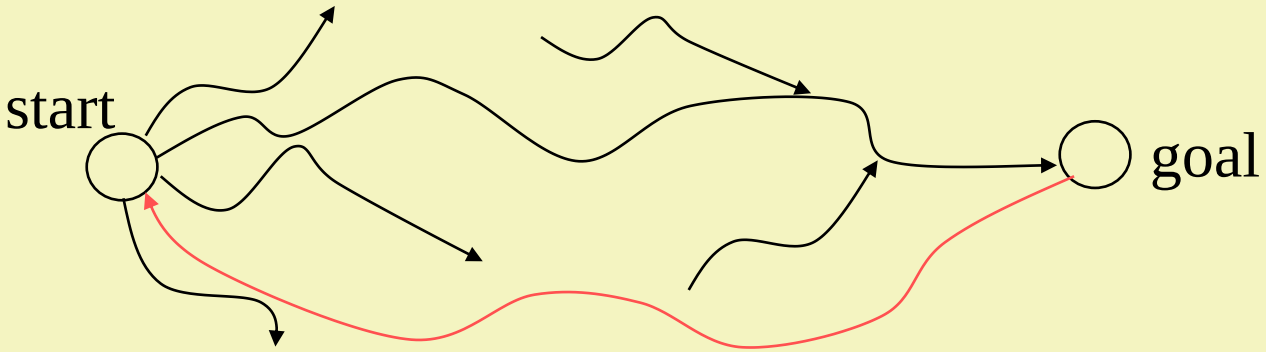
IV. Problem decomposition

1. Backward search
2. Problem decomposition
3. AND/OR graphs

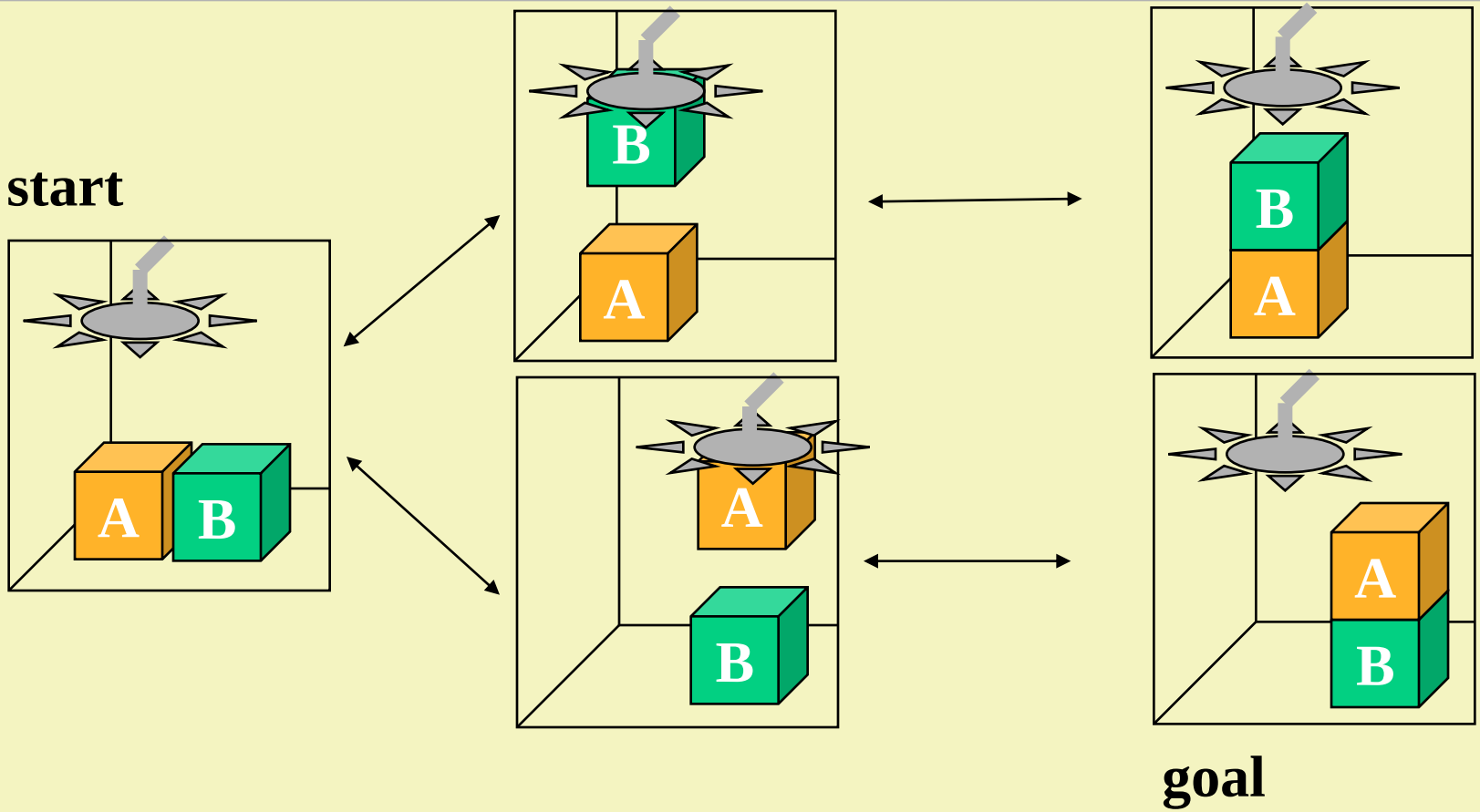
1. Backward search

- Sometimes the solution can be found easier if the search starts from the goal node and goes toward the start node.

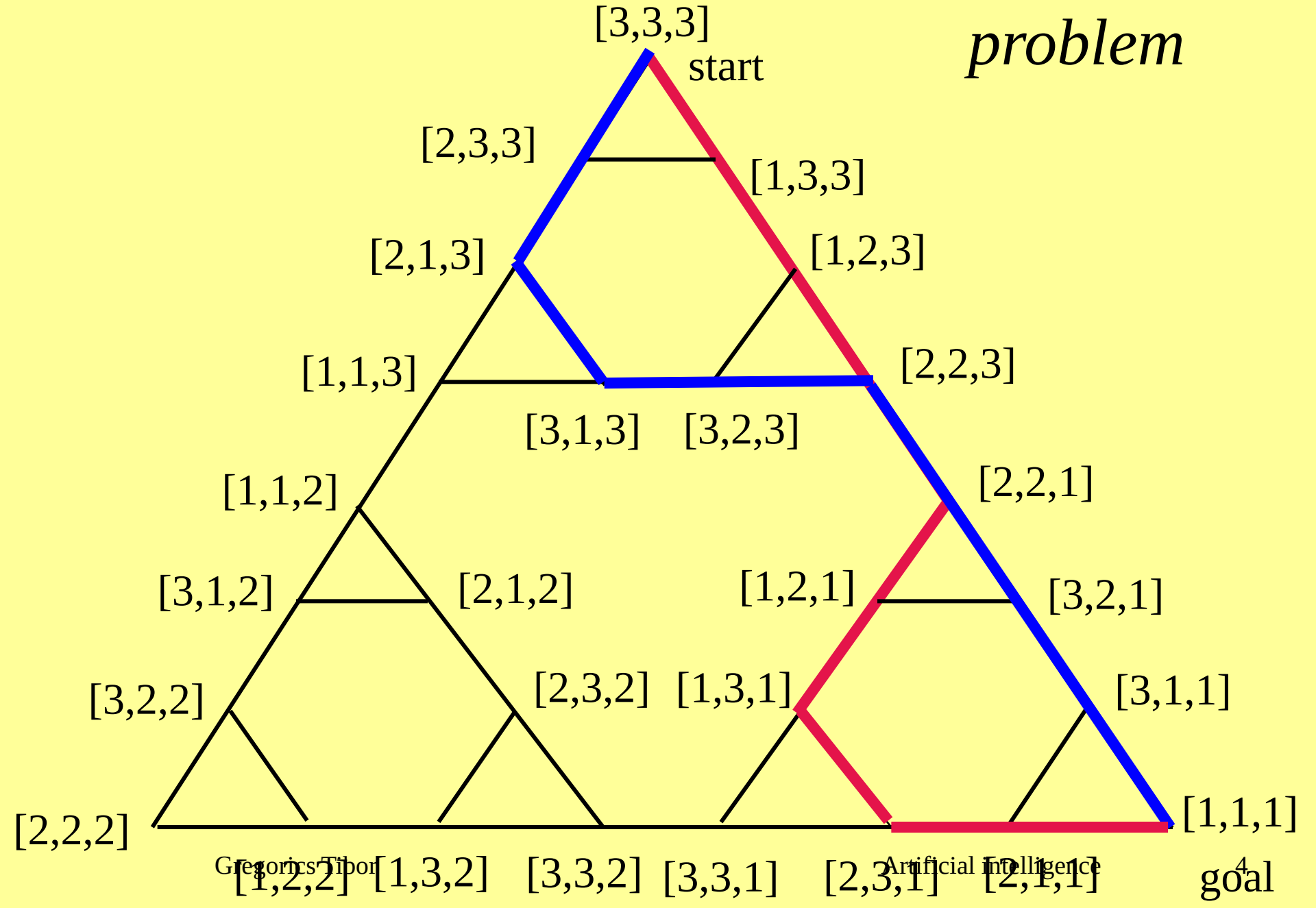


- In this case the path, that has been found, must be interpreted inversely. But it cannot be done always.

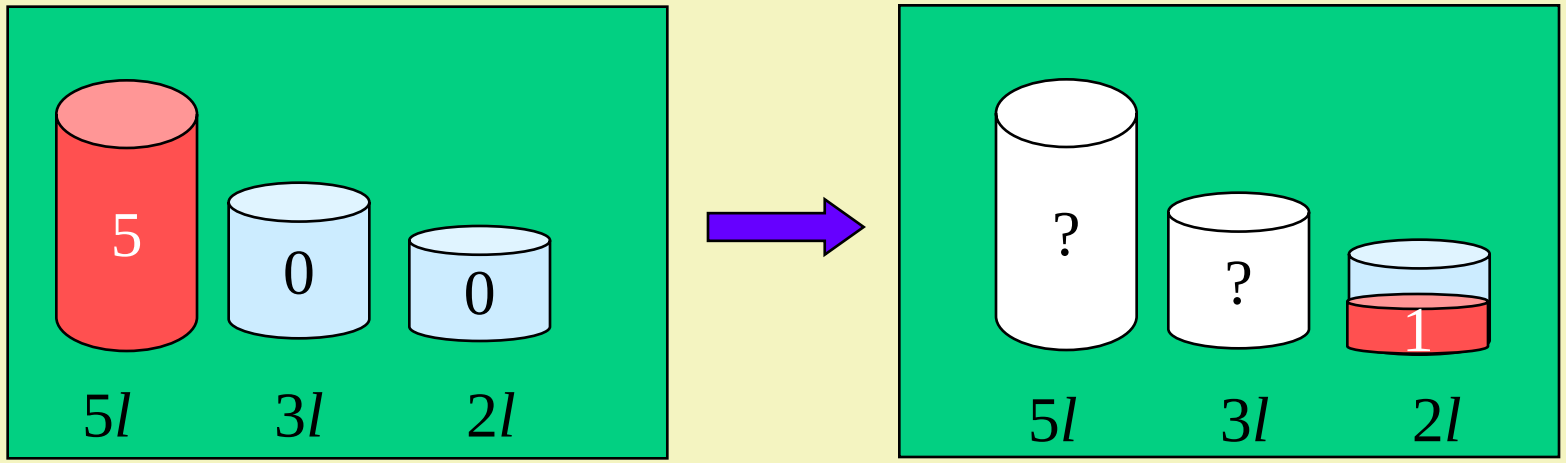
Block world problem



Bidirectional search on Hanoi tower problem



Why does not the backward search solve the jug's problem?



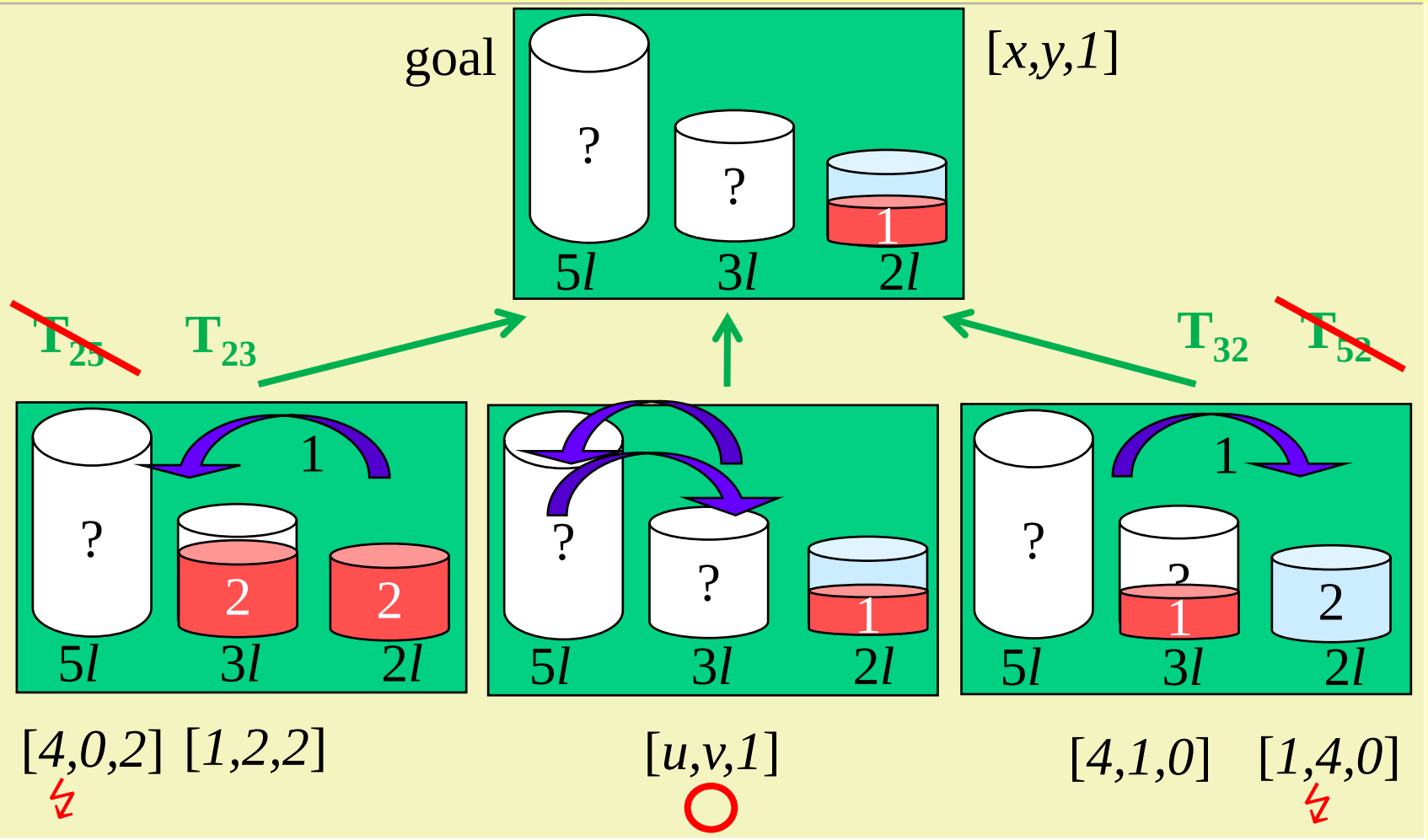
- Backward search finds the path $[4,0,1] \rightarrow [5,0,0]$ but it is not interpreted inversely.
- It is hard to select the goal state which may be the starting point of the backward search. For example the $[2,2,1]$ is not available from the start state.

Conditions of backward search

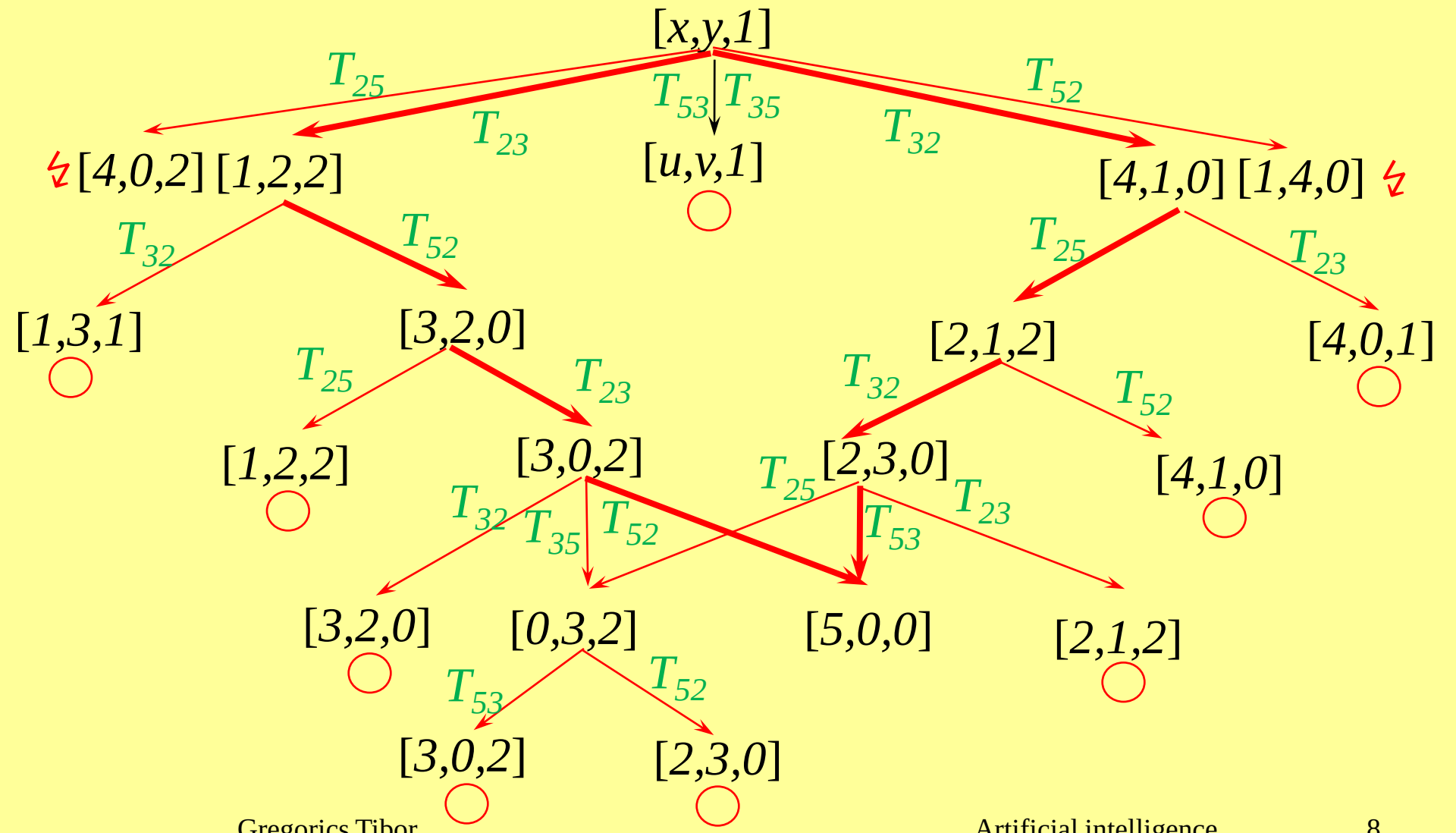
- All arcs have got backward pairs (at least on the path that is found from goal to start)
  - In case of using state space representation it means that all operators must have got inverse operators.
- We need a goal node from where the backward search can start.

What can we do if these conditions do not hold?

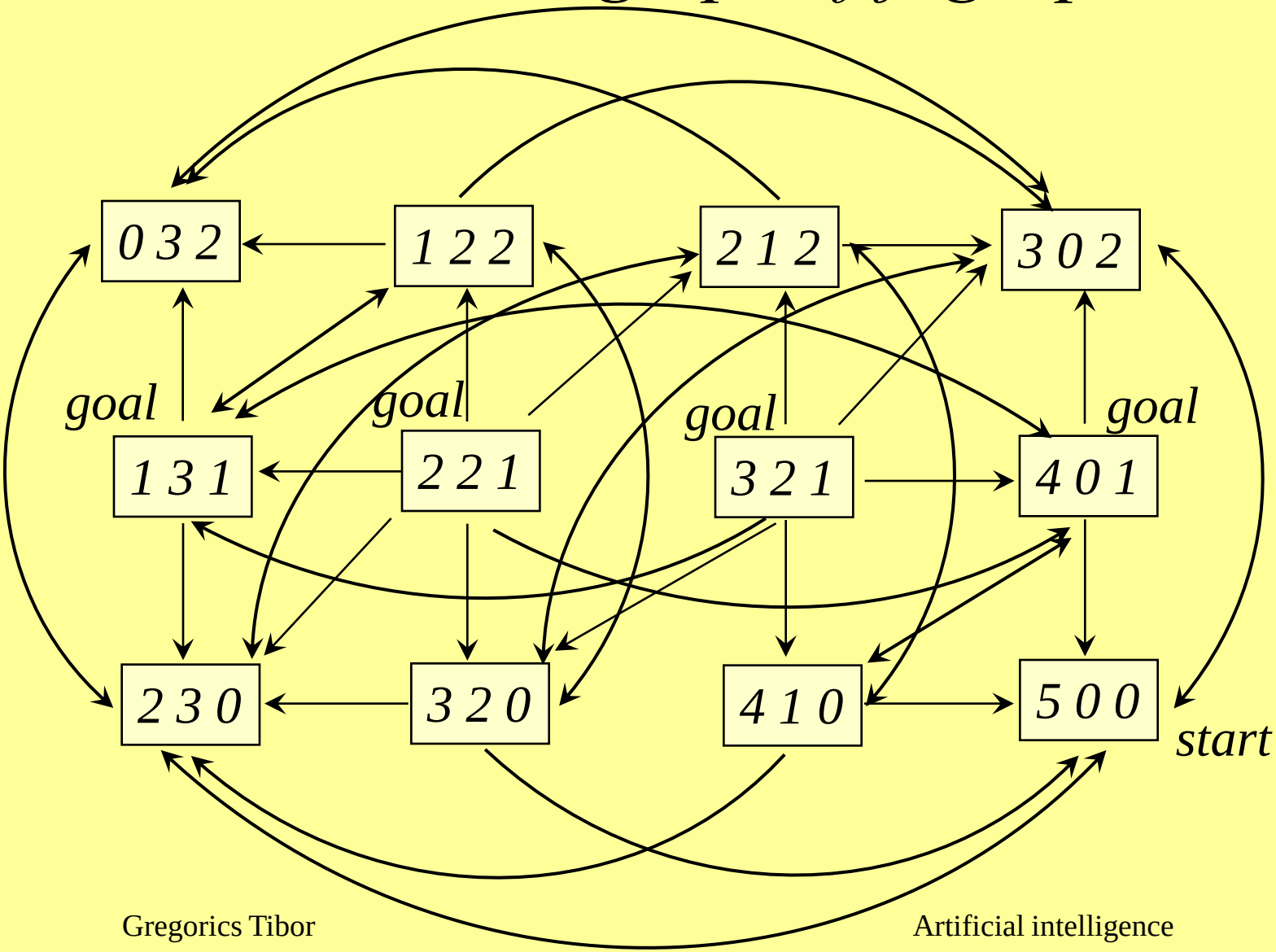
Reduction of jug's problem



Reduction graph of jug's problem



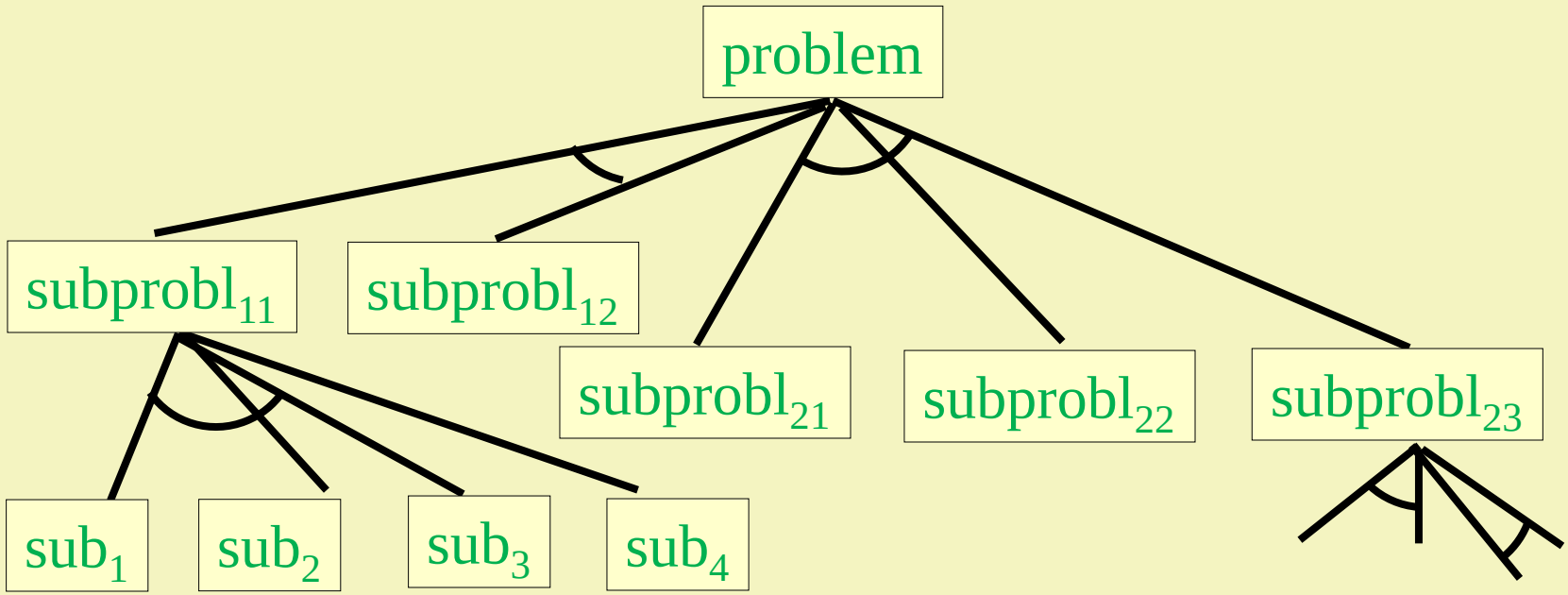
State graph of jug's problem



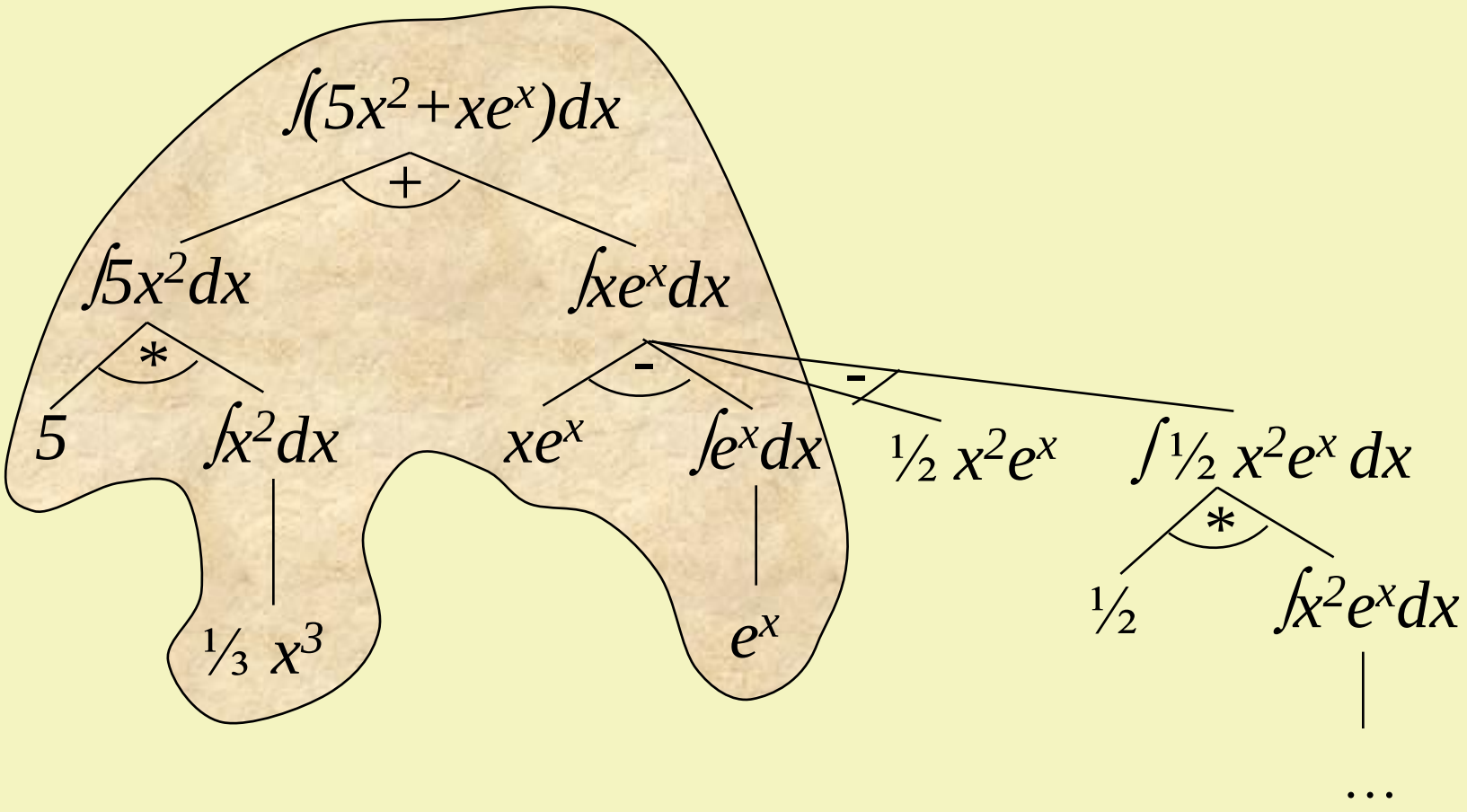
Representation with problem reduction

- The problem reduction is based on the operators $M{:}A \longrightarrow A$ of the state space representation:
  the reduction operator $R_M \subseteq 2^A \times 2^A$ can product the set of states (Y) from that the original operator M leads to the states of a given state-set (X): $(X,Y) \in R_M \Leftrightarrow \forall a \in Y{:}\ M(a) \in X$
  - Y may be „inconsistent”.
- Our aim is to find a path from the set of the goal states to the set of the start state. Reading this path backward we get the solution as a sequence of the operators of the original state space representation.

2. Problem decomposition



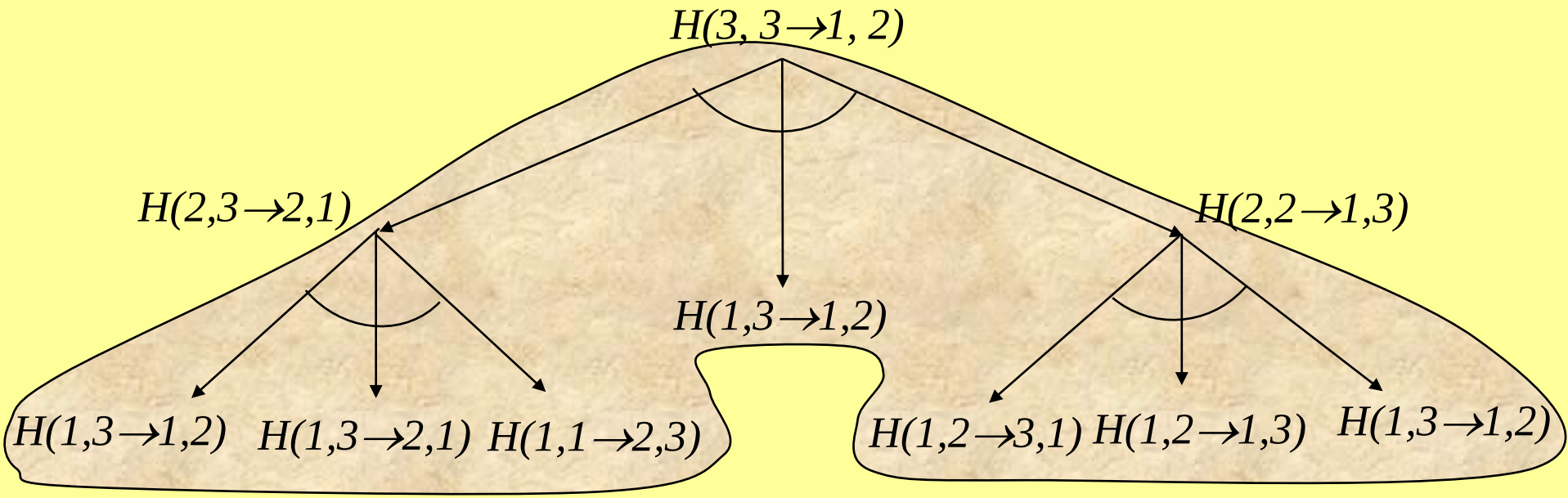
Symbolic integration



Hanoi tower problem

Instead of $H(n, i \rightarrow j, k)$ we have

$$H(n\text{-}1, i \rightarrow k, j)\ H(1, i \rightarrow j, k)\ H(n\text{-}1, k \rightarrow j, i)$$



Concept of problem decomposition

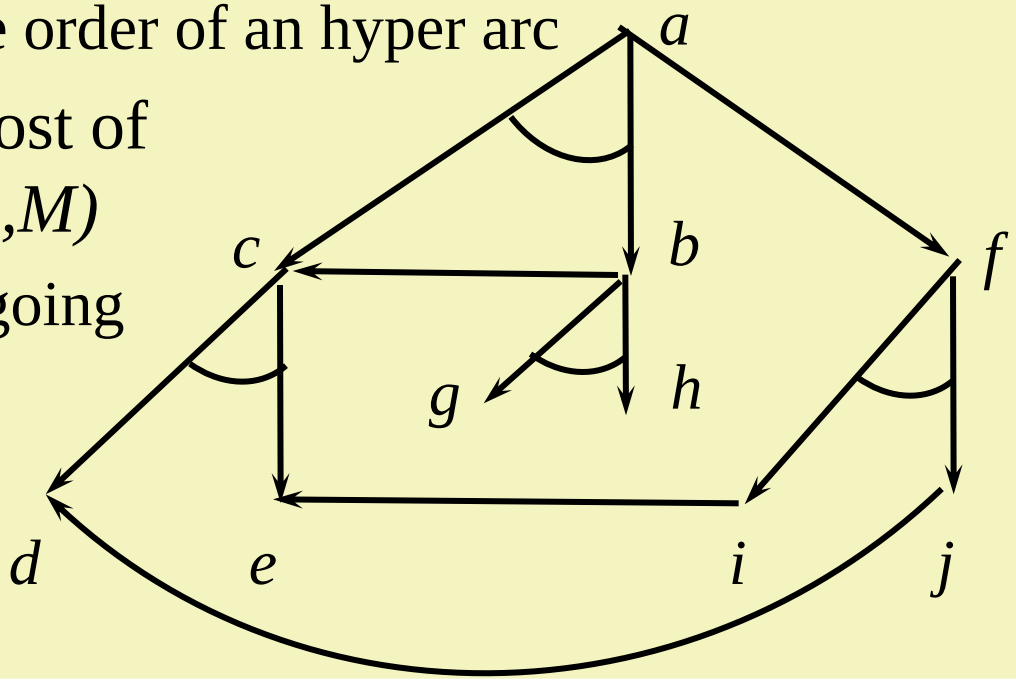
- Representation of decomposition contains:
  - general description of the subproblems,
  - original problem,
  - primitive (simple) problems that can be decided if they can be solved and their solution can be computed easy
  - decomposing operators :
    - D: *problem* $\rightarrow$ *problem*$^+$ and
      $D(p)=<p_1, \ldots , p_n>$

Graph representation

- The problem space of a problem can be described with not an ordinary δ-graph but a so-called AND/OR graph.
- The solution is not an ordinary path but a special subgraph: solution graph
  - A solution graph has no arc with „OR” relationship but it contains all arcs connected by „AND” so the solution graph gives an unequivocal direction from the node of the original problem to the nodes of primitive problems.

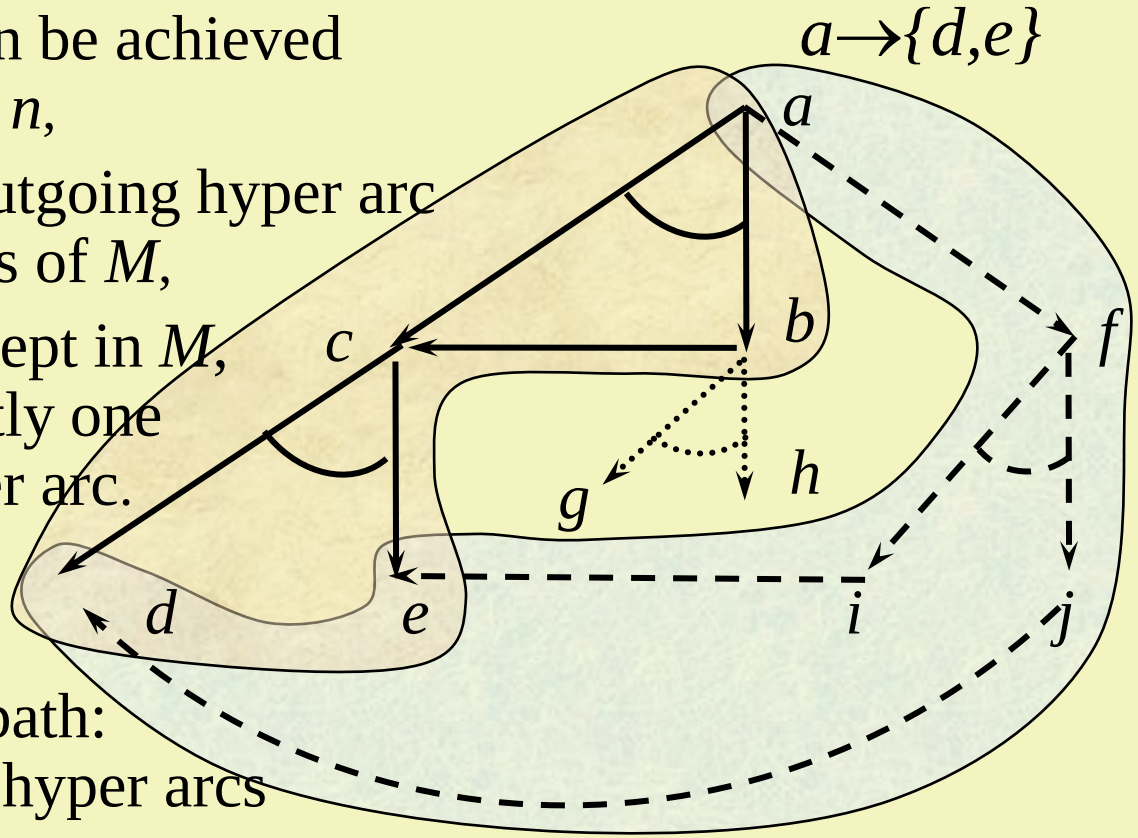
3. AND/OR graph

- Az $R=(N,A)$ is a arc-weighted directed hyper graph, where
  - N is the set of nodes,
  - $A \subseteq \{ (n,M) \in N \times 2^N \mid 0 \neq |M| < \infty \}$ is the set of hyper arcs, $|M|$ is the order of an 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
- $0 < \delta \leq c(n,M)$



Hyper path form the node n to the set of nodes M

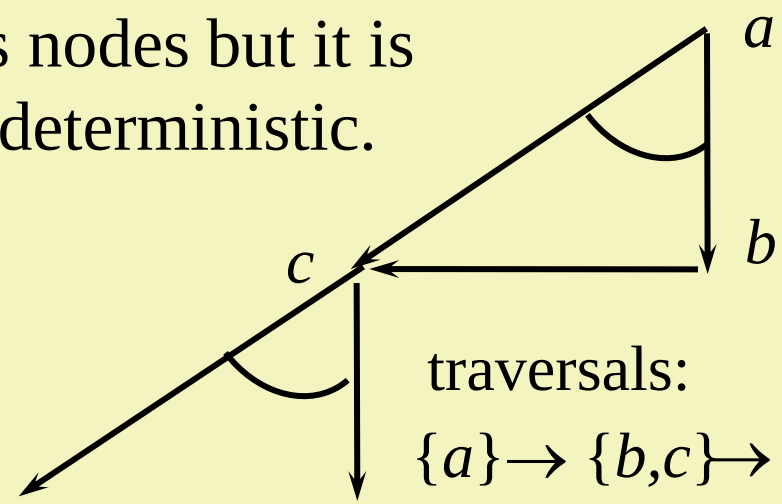
- It is a finite subgraph ($n^{\alpha} \rightarrow M$) of an AND/OR graph, where
  - every node can be achieved from the node *n*,
  - there are no outgoing hyper arc from the nodes of *M*,
  - the nodes, except in *M*, have got exactly one outgoing hyper arc.



- Length of hyper path: the number of its hyper arcs
- Cost of hyper path: the sum of the cost of its hyper arcs

Difference between the traversal of ordinary directed path and hyper path

- The traversal of an ordinary path is the sequence of the nodes of this path in order by its arcs. It is always deterministic.
- The traversal of an hyper path is the sequence of the sets of its nodes but it is non-deterministic.



traversals:

$\{a\} \rightarrow \{b,c\} \rightarrowtail \{c\} \rightarrow \{d,e\}$

$\{a\} \rightarrow \{b,c\} \rightarrow \{b,d,e\} \rightarrowtail \{c,d,e\} \rightarrow \{d,e\}$

Traversal of hyper path

- Traversal is the sequence of the sets of the nodes of the hyper path where
  - First set: $\{n\}$
  - The set C is followed by the set $C\text{-}\{k\}\cup K$ if there exists a hyper arc (k,K) in the hyper path where $k\in C$ but $k\notin M$

Remarks

- A traversal is treated as a sequence of all hyper arcs of the hyper path where the same hyper arc can be occurred several times.

1. The number of the occurrences of the hyper arc (k,K) of the hyper path $n \rightarrow M$ in the traversal of this hyper path is at most the number of the ordinary paths driving from n to k in this hyper path.
2. A hyper path including ordinary directed circle has no finite traversal.
3. A hyper path $n \rightarrow M$ without ordinary directed circle has finite number different finite traversal where the last set is M.

Graph representation of the problem decomposition

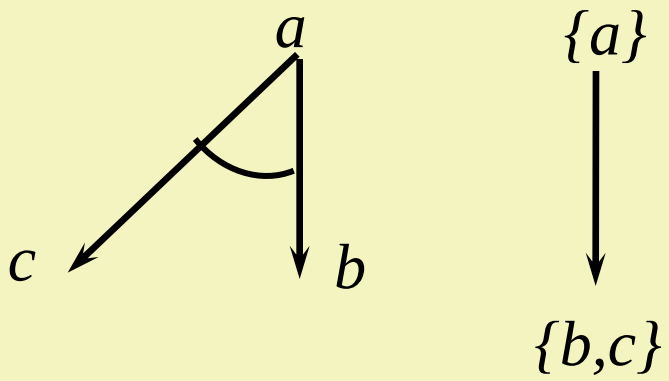
- The graph representation of a decomposition is the triple (R,s,T) where
  - $R=(N,A,c)$ is an AND/OR graph where
  - N denotes the subproblems,
  - A denotes the operators,
  - c gives the cost of the operators,
  - s means the initial problem,
  - T contains the simple problems.
- The solution of the problem is a hyper path $s \rightarrow M \subseteq T$ (named as solution graph) that does not contain circle.

Search in AND/OR graph

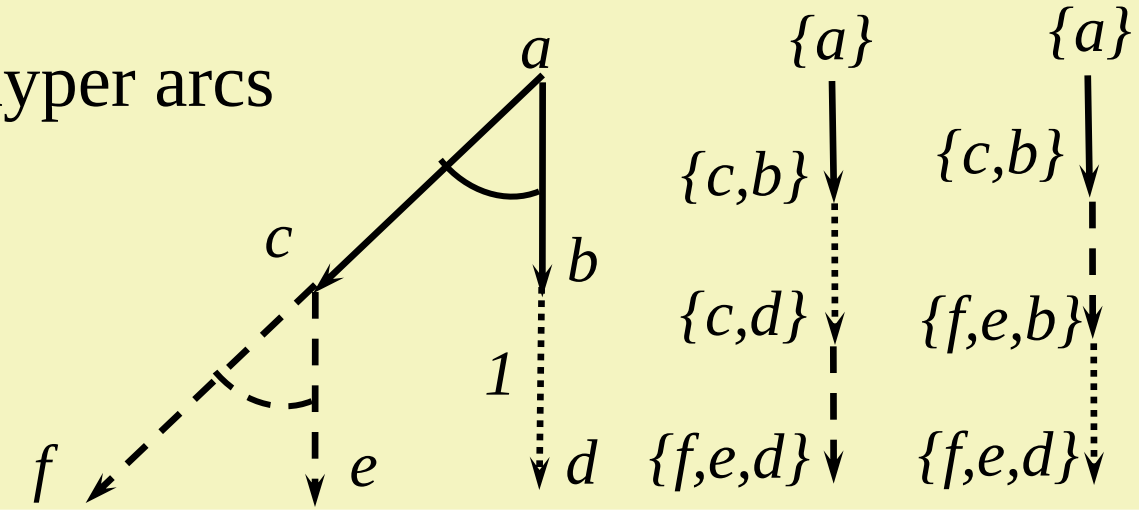
- Every AND/OR graph can be corresponded to an ordinary δ-graph where the solution paths are the traversals of the solution graphs of the AND/OR graph.
- That because the different search algorithms of the ordinary δ-graphs can be adapted onto the AND/OR graphs and they are able to find solution graph.

Transformation of hyper paths

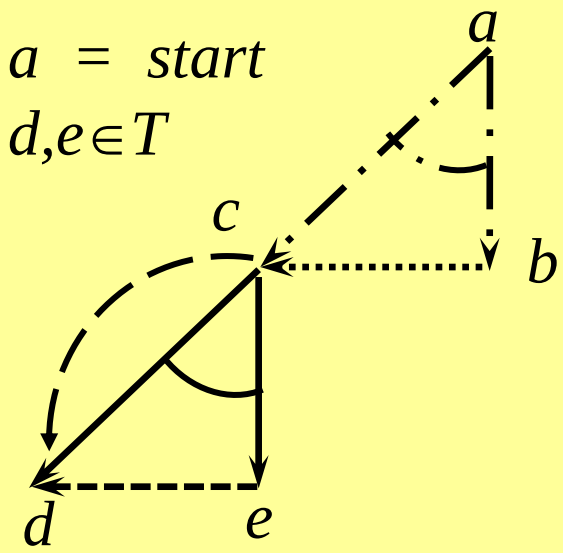
Case of one hyper arc
in a hyper path:



Case of several hyper arcs
in a hyper path

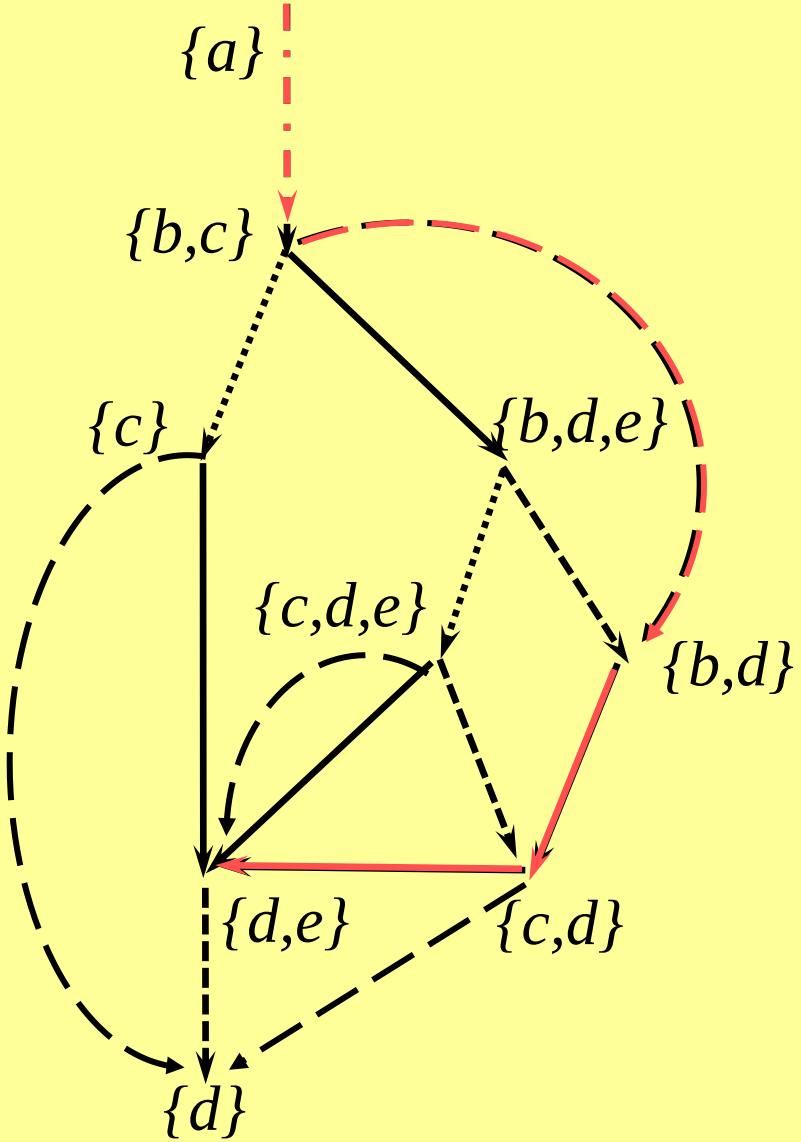


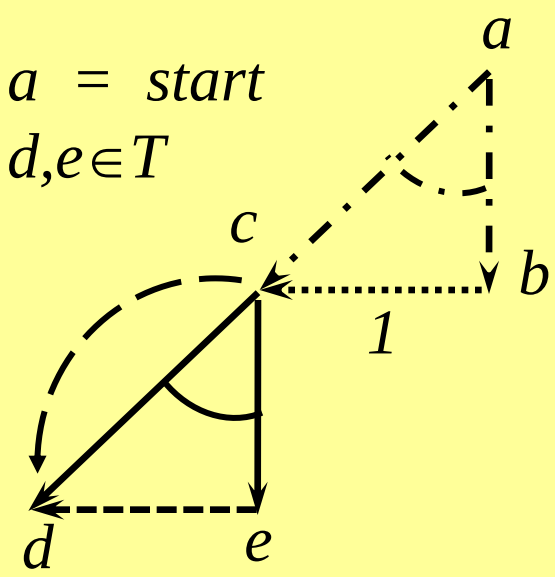
Transformation of AND/OR graph

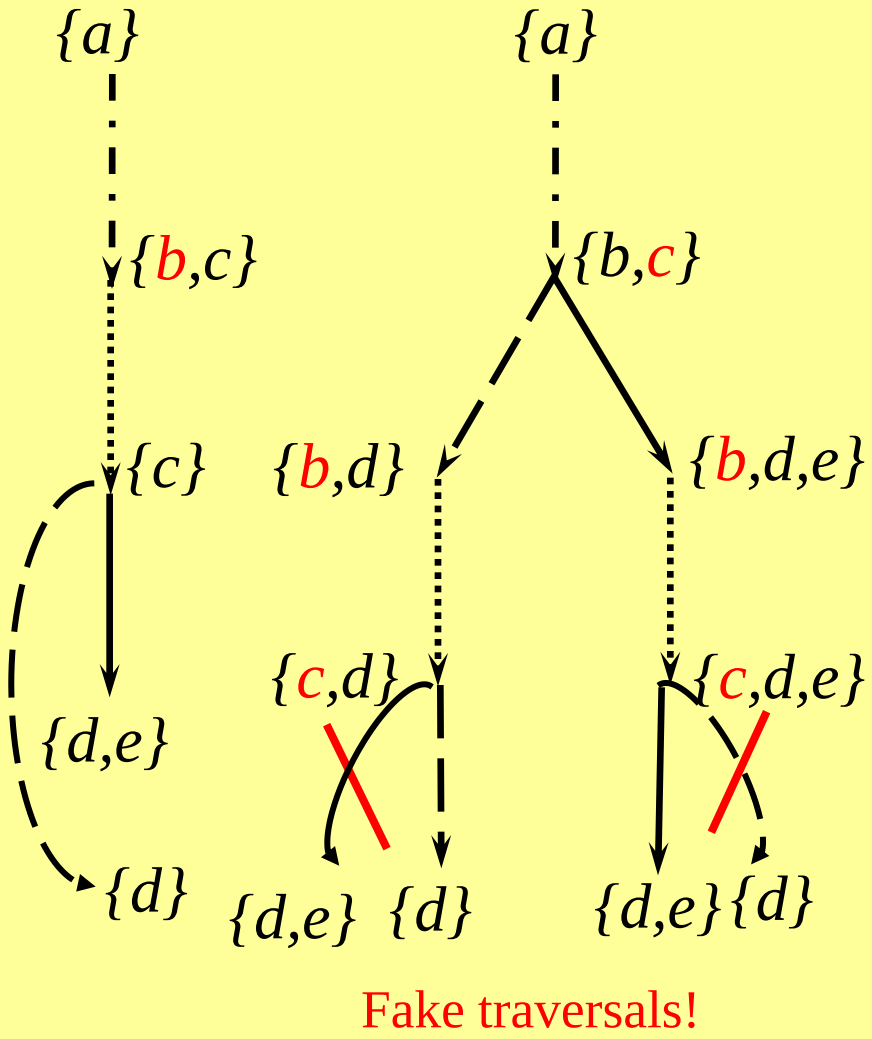


The traversals of the hyper paths outgoing from the start node are drawn up as ordinary paths.

No need all traversals of a hyper path!
Fake traversals (where the same node is substituted several times in different way) must be avoided!







1. Only one traversal of a hyper path is enough so in one step of the transformation it is enough to substitute only one node.

2. Only the hyper paths $s \rightarrow M \subseteq T$ are interested never step away from the goal nodes.

Fake traversals!

3. To avoid the fake traversals an ordinary tree is built and when a node is selected from the set C that node has been selected before (on the path driving from the start to C) then this node is substituted based on the same hyper arc that is used erlier.

Algorithm of transformation

1. Put the set {*start*} into a QUEUE as a start node of the ordinary graph.
2. If QUEUE is empty, then EXIT, otherwise a set C is pulled out from it and the successors of C are generated in the ordinary graph.
3. If C contains only goal nodes, then the set C is a goal in the ordinary graph without outgoing arcs.
4. Otherwise a non goal node k is selected from C.
   - If there exists an arc generated based on the hyper arc (k, K) on the path driving from {*start*} to C, then there will be only one successor of C, this is $C-\{k\}\cup K$.
   - Otherwise all outgoing hyper arc (k, K_i) generate successor $C-\{k\}\cup K_i$.
5. The successors of C are put into the QUEUE.
6. GOTO 2.

Theorem

1. After transformation every solution path of the ordinary graph represents one of the traversals of a solution graph of the AND/OR graph.
2. The transformation creates an ordinary solution path from one of the traversals of every solution graph of an AND/OR graph in finite steps.

- Remarks:
  - The transformed graph is a δ-graph
  - The transformation is built into the search algorithms.

Backtracking on AND/OR graph

```
Recursive procedure VL2(traversal) return solution
1.      C := tail(traversal)
2.      if  allgoal(C) then  return(nil) endif
3.      if length(traversal) ≥ limit then  return(fail) endif
4.      if C ∈ remain(traversal) then  return(fail) endif
5.      k := get-non-goalC)
6.      for  ∀ (k,K) ∈outgoing-hyper-arcs (k)  loop
7.          solution := VL2(concat(traversal, C-{k}∪K) )
8.          if solution ≠ fail then
9.              return(concat((C, C-{k}∪K), solution))  endif
10.     endloop
11.     return(fail)
end
```