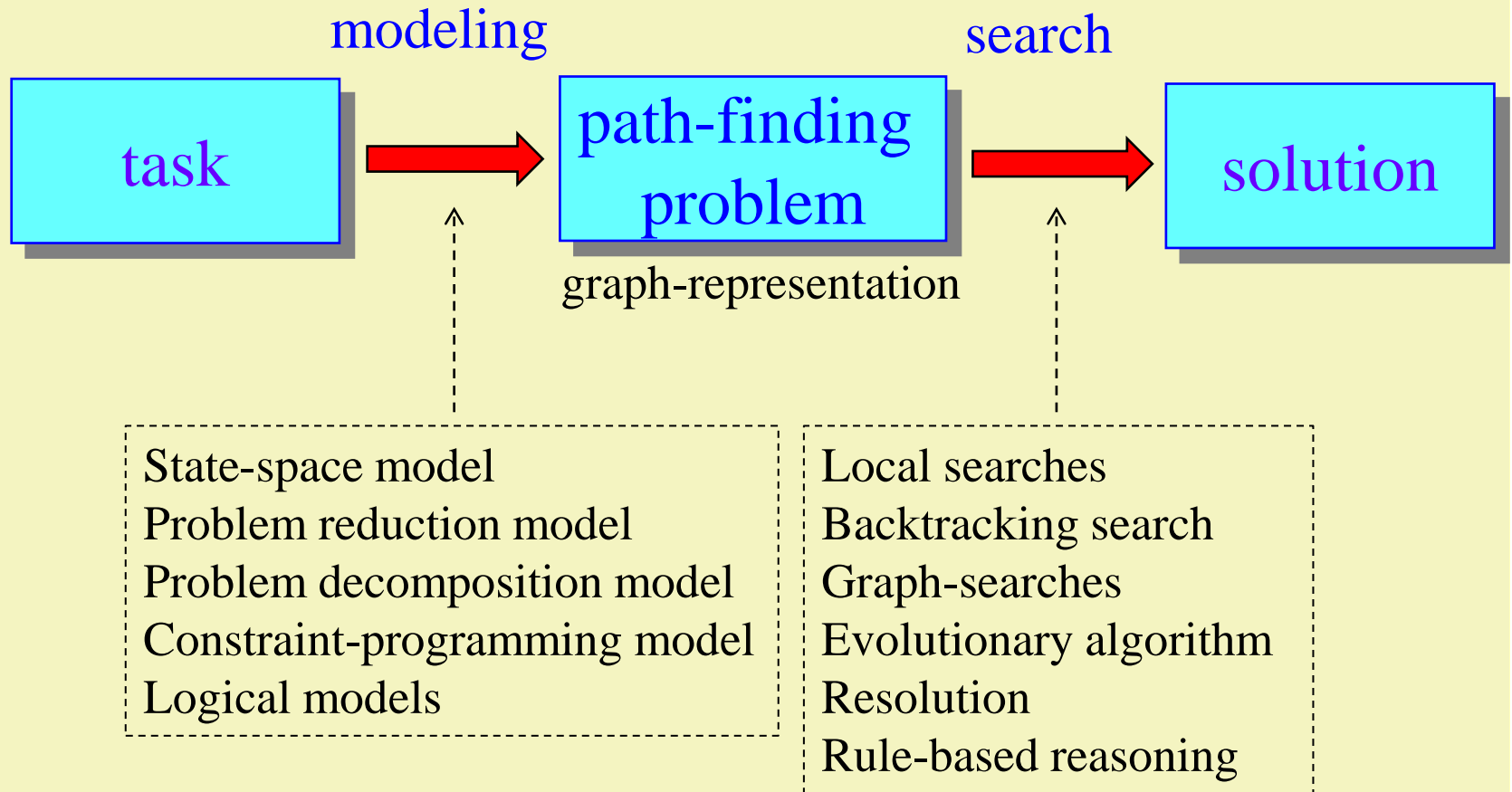


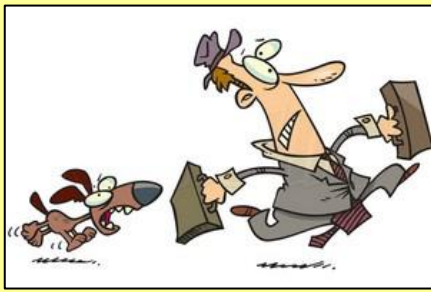
STATE-SPACE MODEL

Modeling & Search



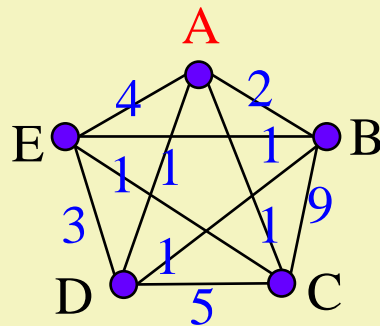
What does the modeling focus on?

- ❑ *Problem space*: contains the possible answers.
- ❑ *Goal*: finding correct answers (solutions).
- ❑ *Ideas that help the search*:
 - Restricting the problem space: feasible answers
 - Selecting an initial item of the problem space.
 - Defining neighborhood relationships between the items of the problem space:
 - It helps to go through the problem space systematically.
 - Ranking the currently available items of the problem space during its traversal



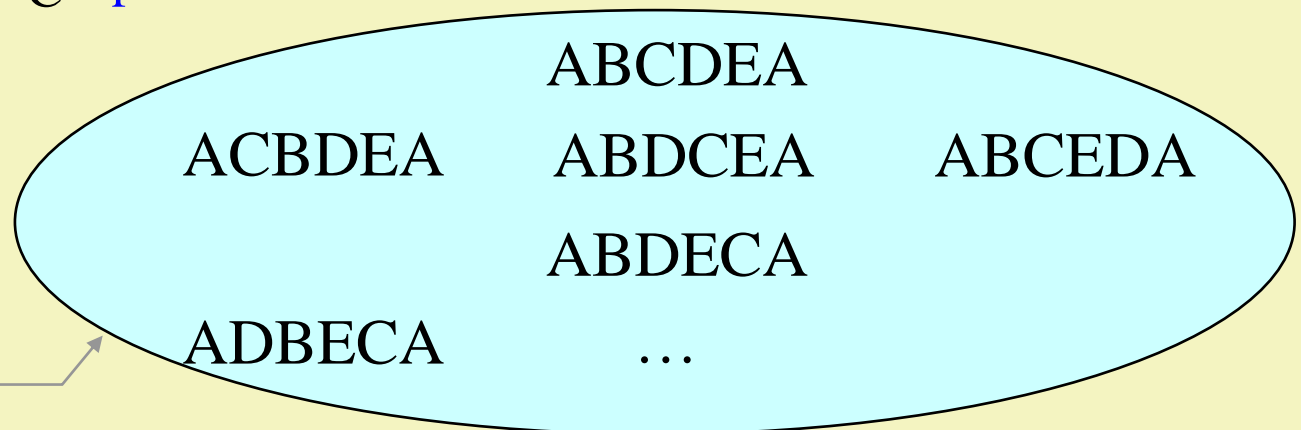
Travelling salesman problem

The traveling salesman must visit every city in his territory exactly once and then return home covering the optimal total cost. (n cities and cost of each pair of cities are known)



possible solutions:

n	$(n-1)!$
5	24
50	$6 \cdot 10^{62}$

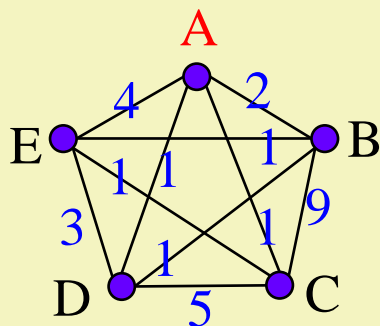


problem space



Travelling salesman problem

The traveling salesman must visit every city in his territory exactly once and then return home covering the optimal total cost. (n cities and cost of each pair of cities are known)



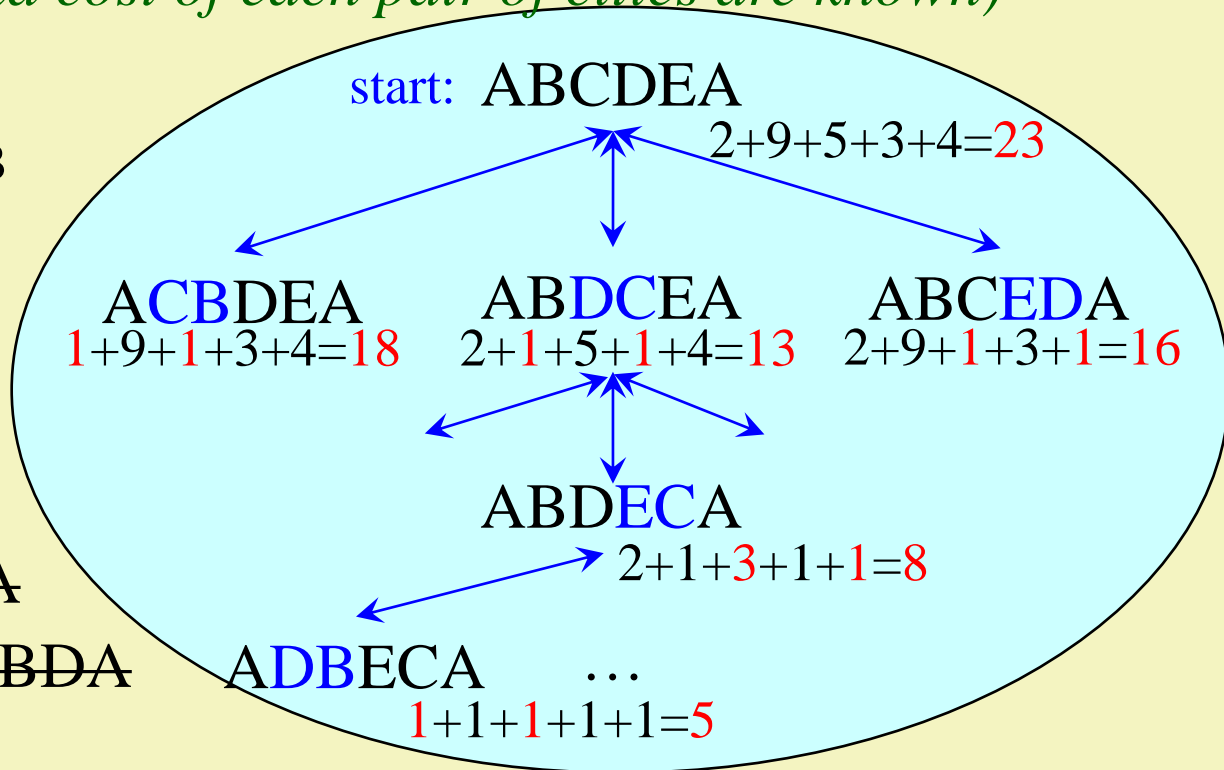
~~AEDCBA~~

~~AEDBCA~~

~~ACEDBA~~

~~ACEBDA~~

unnecessary



Path-finding problems

- Using an appropriate model, the problem space of a path-finding problem can be treated as an **arc-weighted, directed graph**, where the items of the problem space are represented by either nodes or paths. This graph might be infinite but the number of the **outgoing arcs of each node is always finite**, and there is a **positive constant lower bound (δ) on the cost of the edges (δ -graph)**.
- In order to solve these problems either a special **goal node** or a **path driving from a start node to any goal node** must be found. (Sometimes the optimal path is needed.)

Graph notations 1.

- nodes, arcs $N, A \subseteq N \times N$ (infinite)
- arc from n to m $(n, m) \in A \ (n, m \in N)$
- children of n $\Gamma(n) = \{m \in N \mid \exists (n, m) \in A\}$
- parents of n $\pi(n) \in \Pi(n) = \{m \in N \mid \exists (m, n) \in A\}$
- directed graph $R = (N, A)$
- finite outgoing arcs $|\Gamma(n)| < \infty \ (\forall n \in N)$
- cost of arc $c: A \rightarrow \mathbb{R}$
- $c(\delta \in \mathbb{R}^+)$ $c(n, m) \geq \delta > 0 \ \forall (n, m) \in A$
- δ -graph directed, arc-weighted, δ -property, finite outgoing arcs from a node

Graph notations 2.

- directed path

It is correct in spite of infinite number of paths since we have δ -graphs.

If no path, it is infinite.

- length of a path
- cost of a path
- optimal cost
- optimal path

$$\begin{aligned}\alpha &= (n, n_1), (n_1, n_2), \dots, (n_{k-1}, m) \\ &= \langle n, n_1, n_2, \dots, n_{k-1}, m \rangle\end{aligned}$$

$$n \rightarrow^\alpha m, n \rightarrow m, n \rightarrow M \quad (M \subseteq N)$$

$$\{n \rightarrow m\}, \{n \rightarrow M\} \quad (M \subseteq N)$$

$$|\alpha|$$

$$c^\alpha(n, m) := \sum_i c(n_{i-1}, n_i)$$

$$c^*(n, m) := \min_{\alpha \in \{n \rightarrow m\}} c^\alpha(n, m)$$

$$c^*(n, M) := \min_{\alpha \in \{n \rightarrow M\}} c^\alpha(n, m)$$

$$n \rightarrow^* m := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow m\} \}$$

$$n \rightarrow^* M := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow M\} \}$$

Definition of graph-representation

- All path-finding problems can be described with a graph-representation. It is a triple (R, s, T) where
 - $R=(N, A, c)$ is a δ -graph (representation graph)
 - $s \in N$ is the start node
 - $T \subseteq N$ is the set of goal nodes.
- Solution of the problem:
 - finding a goal node: $t \in T$
 - finding a path $s \rightarrow T$ or an optimal path $s \rightarrow^* T$
 - directed path from s to one of the nodes of T
 - the cheapest directed path from s to one of the nodes of T

Search

- ❑ Only special path-finding algorithms can find solution paths in a huge graph.
 - It **starts** from the start node that is the first current node.
 - In each step it **selects** a new current node among the children of the previous current node(s) in **non-deterministic** way.
 - It can **store** the subpart of the discovered part of the representation graph.
 - It **stops** when it detects the goal node.

Search-system

Procedure *Search-system*

1. **DATA** := *initial value*
 2. **while** \neg *termination condition*(**DATA**) **loop**
 3. **SELECT** **R** **FROM** *rules* that can be applied
 4. **DATA** := **R**(**DATA**)
 5. **endloop**
- end**

Search-system

Procedure *Search-system*

1. **DATA** \leftarrow *initial value*
 2. **while** \neg *termination condition*(**DATA**) **loop**
 3. **SELECT** **R** **FROM** *rules* that can be applied
 4. **DATA** \leftarrow **R**(**DATA**)
 5. **endloop**
- end**

global workspace

stores the part of knowledge acquired that is useful to preserve
(*initial value, termination condition*)

searching rules

can change the content of the workspace
(*precondition, effect*)

control strategy

selects an appropriate rule
(*general principle + heuristics*)

Examination of search-system

- ❑ soundness
 - the answer is correct if the search terminates
- ❑ completeness
 - the search guarantees the solution if there exists a solution
- ❑ optimality
 - the search gives optimal solution
- ❑ time complexity
 - number of the iterations and running time of one iteration
- ❑ space complexity
 - size of the workspace

Concept of state-space model

- *State-space*: set of states, where one state is a collection of values belonging to the data (objects) that are needed to describe the problem
 - the state-space can be defined as a subset of a base-set by a so-called *invariant statement*.
- *Operators*: map from the state-space to the state-space
 - step from a state to another state
 - defined with its *precondition* and *effect*
- *Initial state(s)* or its description (*initial condition*)
- *Final state(s)* or its description (*goal condition*)

Graph-representation of state-space model

❑ State-space model

- state
- effect of an operator on a state
- cost of an operator
- initial state
- final state

State-graph

node

directed arc

cost of arc

start node

goal node

❑ Graph-representation:

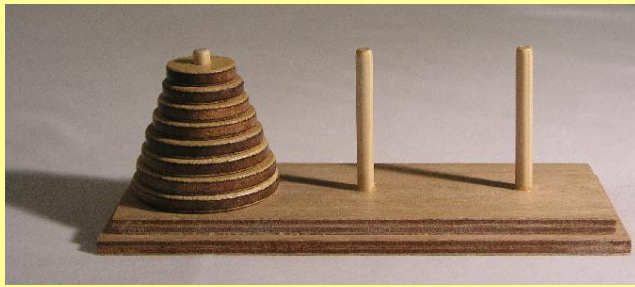
state-graph, start node, goal nodes

- sequence of operators
- solution

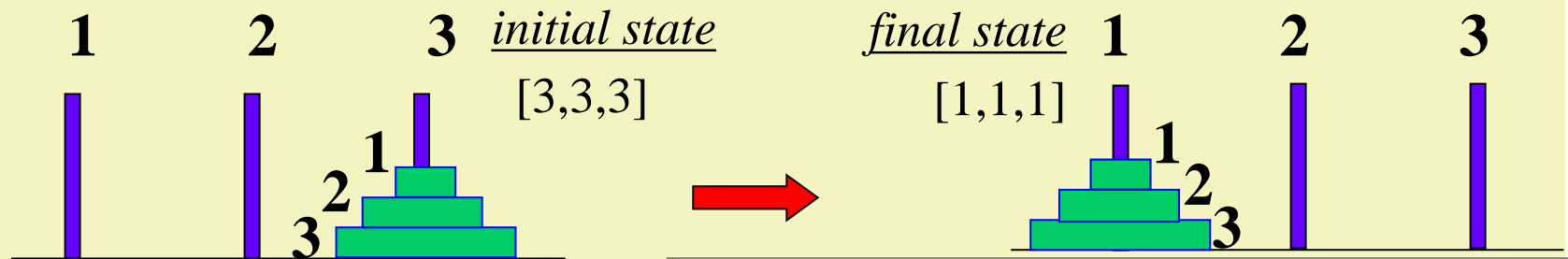
representation graph,
 δ -graph

directed path

directed path
from start to goal



Hanoi tower problem



State-space: $ST = \{1,2,3\}^n$

set of all possible n length sequences (arrays) where the elements may be 1, 2 or 3.

Operator: $Move(from, to): ST \rightarrow ST \quad from, to \in \{1,2,3\}$

IF '*from*' and '*to*' are valid and different pegs

and there is a disc on '*from*'

and '*to*' is either empty or its upper disc is greater than the disc we want to move (this is the upper disc on '*from*')

THEN *this*[the upper disc on '*from*'] := *to*

start

[3,3,3]

State-graph

Possible solutions are the paths driving from start

[2,3,3]

[1,3,3]

[2,1,3]

[1,2,3]

[1,1,3]

[2,2,3]

[3,1,3]

[3,2,3]

[1,1,2]

[2,2,1]

[3,1,2]

[2,1,2]

[1,2,1]

[3,2,1]

[3,2,2]

[2,3,2]

[1,3,1]

[3,1,1]

[2,2,2]

[1,1,1]

goal

[1,2,2]

[1,3,2]

[3,3,2]

[3,3,1]

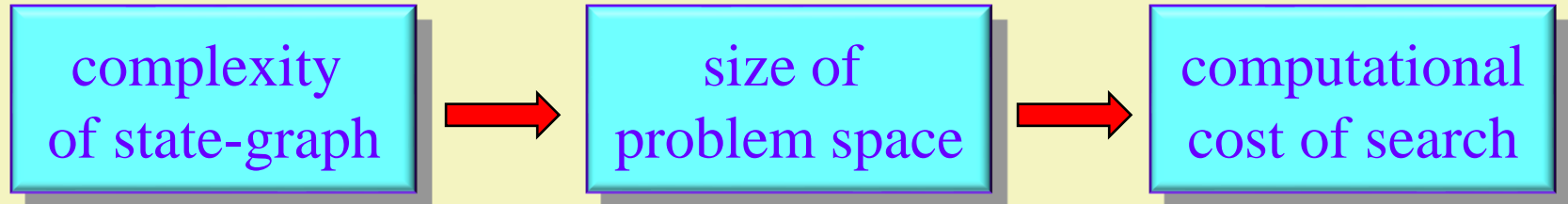
[2,3,1]

[2,1,1]

State-space vs. problem space

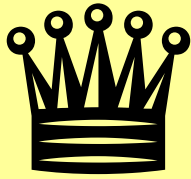
- ❑ The elements of the problem space can be symbolized with the paths driving from the start node in the state-graph.
- ❑ There is a **very close relationship** between the state-space and the problem space, but the state-space is **not identical** to the problem space.
 - In many cases (just in the Hanoi tower problem) the elements of the problem space are not the states (nodes) but the sequences of operators (paths driving from the start node) and some of them are the solutions.
 - Sometimes the solution might be only one state but a sequence of the operators (path) is needed to achieve it.

Complexity of state-graph



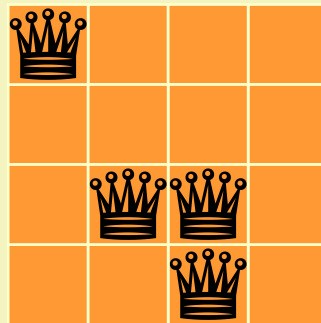
□ number of paths driving from the start depends on

- number of nodes and arcs
- **branching factor**: average number of outgoing arcs
- frequency of the **cycles** and diversity of their length

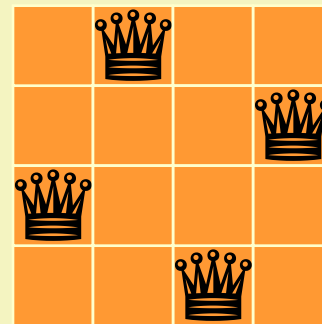


n-queens problem 1.


general state




final state



State-space: $ST = \{ \text{queen}, _ \}^{n \times n}$

two dimensional array ($n \times n$ matrix)
where its elements may be  or $_$

invariant: number of queens () = n

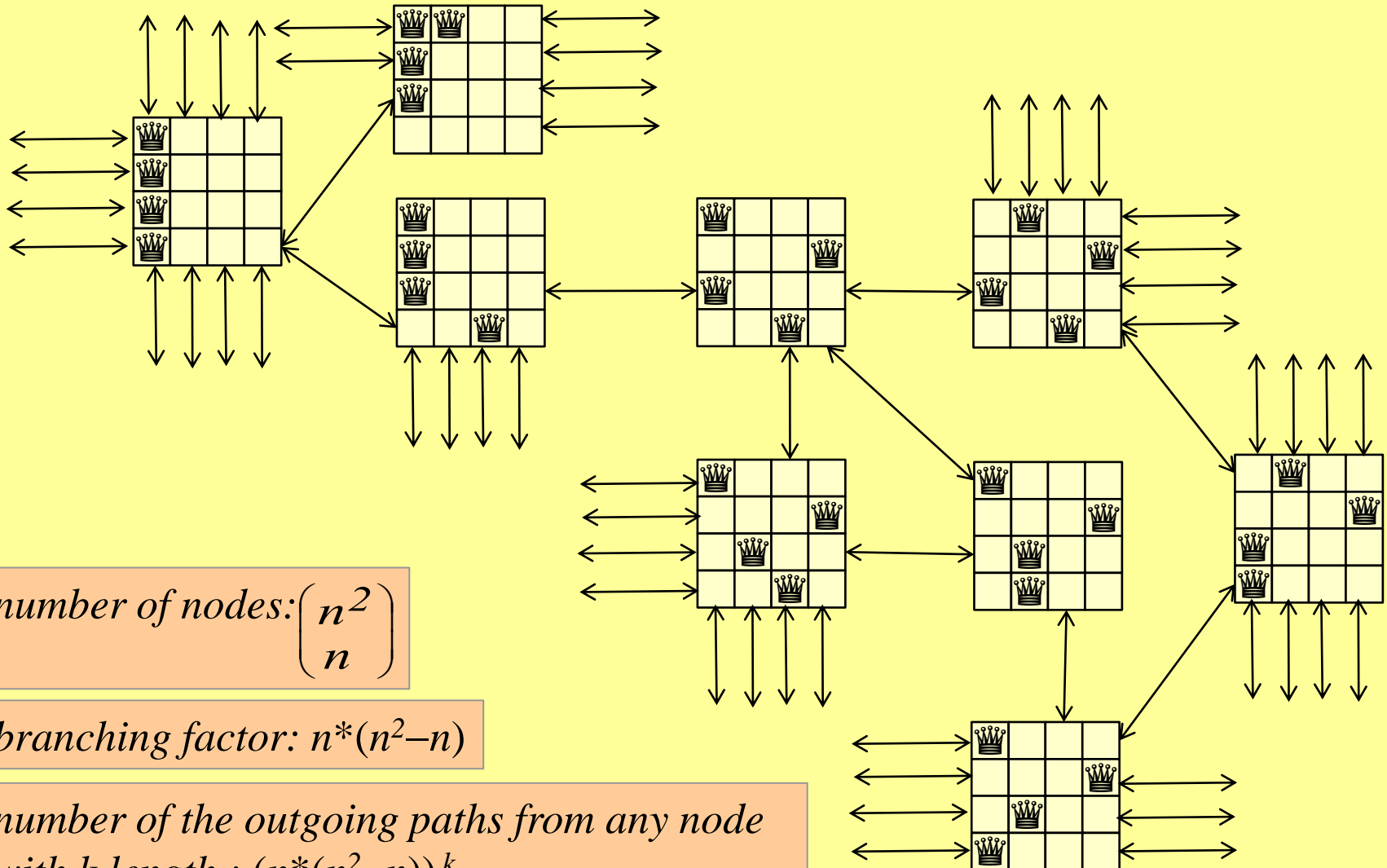
Operator: $\text{Change}(x,y,u,v): ST \rightarrow ST \quad x,y,u,v \in [1..n] \quad (\text{this}:ST)$

IF $1 \leq x,y,u,v \leq n$ and $\text{this}[x,y] = \text{queen}$ and $\text{this}[u,v] = _$

THEN $\text{this}[x,y] \leftrightarrow \text{this}[u,v]$

swap

State-graph



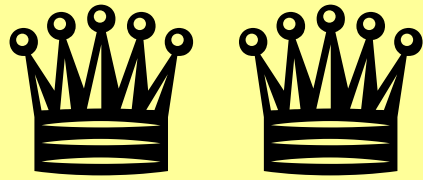
number of nodes: $\binom{n^2}{n}$

branching factor: $n \cdot (n^2 - n)$

number of the outgoing paths from any node
with k length : $(n \cdot (n^2 - n))^k$

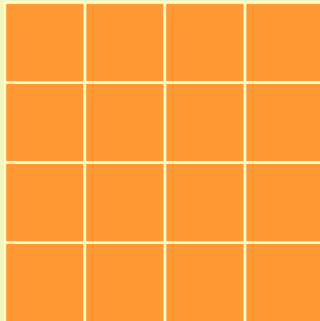
Reduce the problem space

- ❑ A problem may have several models:
the best model = the smallest problem space
 - In the previous representation the size of the problem space is huge.
 - Expand the state space with the states containing less queens than n , and use a new operator: put a new queen on the board (the initial state is the empty board).
 - The state-graph can be further reduced with limiting the precondition of the new operator (decreasing the branching factor):
 - Put the queens on the board row by row.
 - A new queen is never put on the board if it would be under attack.

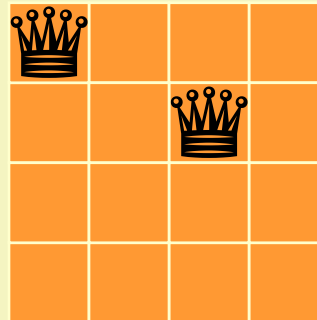


n-queens problem 2.

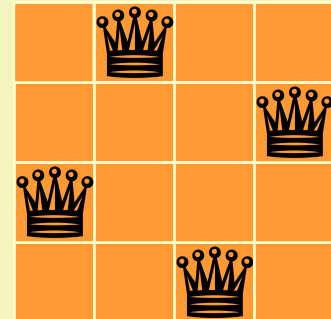
initial state



general state



final state



State-space: $ST = \{ \text{queen}, _ \}^{n \times n}$

invariant: number of queens (queen) $\leq n$ and
only in the first few rows can be found one-one queen

Operator: $Put(col): ST \rightarrow ST$ $col \in [1..n]$ (this: ST)

IF $1 \leq col \leq n$ and number of queens $< n$ and there is no attack

THEN $this[row, col] := \text{queen}$ where „row” is the next empty row

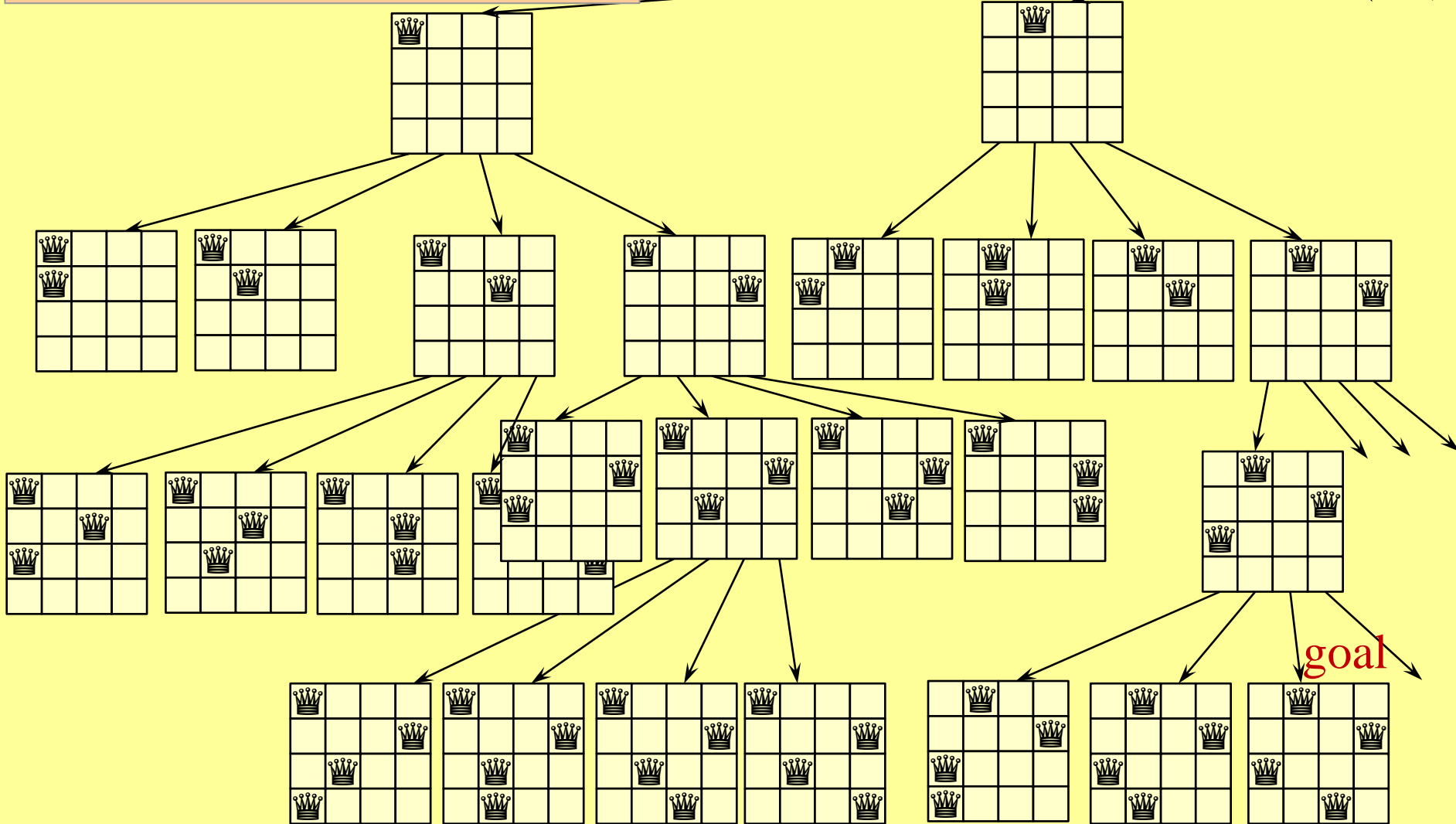
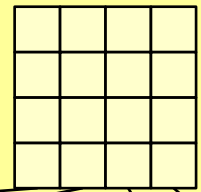
number of nodes $< (n^{n+1}-1)/(n-1)$

branching factor: n

number of possible solutions $< n^n$

State-graph

start

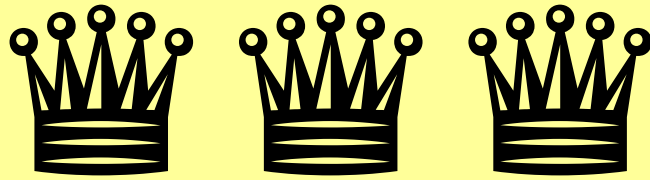


Gregorics Tibor

Artificial intelligence

Computational cost of the operator

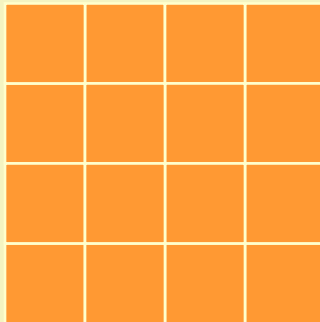
- ❑ The computational complexity of an operator can be reduced if the states are completed with extra information that are maintained by the operator itself.
- ❑ For example
 - The position of the next empty row can be stored in a state. It may be increased after placing a new queen instead of computing it over and over.
 - To avoid the attacks on the chessboard the empty squares that are under attack (not free) might be annotated in order to check easily whether a queen is allowed to place on that square. In this way there will be three kinds of squares: free, under attack and occupied by queen.



n-queens problem 3.

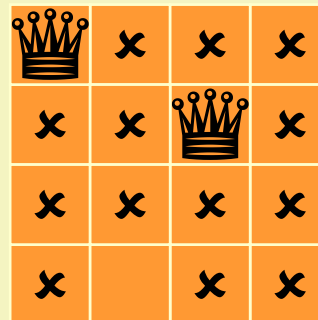
initial state:

next_row = 1



general state:

next_row = 3



final state:

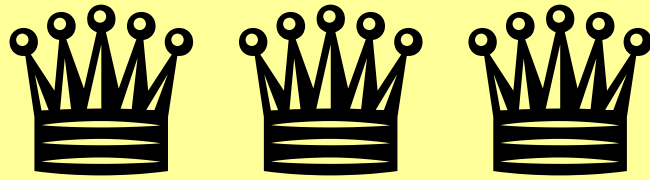
next_row = 5



State-space: $ST = \text{rec}(\text{board}: \{ \text{crown}, \text{x}, _ \}^{n \times n}, \text{next_row}: \mathbb{N})$

invariant: only *next_row-1* queens are on the board
in its first *next_row-1* rows one by one,
 $\text{next_row} \leq n+1$,
no attacks,

notations: **x** denotes the empty square under attack
_ denotes the free square



n-queens problem 3.

Operator:

Put(*col*): $ST \rightarrow ST$

$col \in [1..n]$ (*this*: ST)

IF $1 \leq col \leq n$ and $this.next_row \leq n$
and $this.board[this.next_row, col] = _$

*time complexity
of precondition
is constant*

THEN $this.board[this.next_row, col] := \text{crown}$

$\forall i \in [this.next_row + 1..n]:$

$this.board[i, col] := \times$

$this.board[i, i - this.next_row + col] := \times$

$this.board[i, this.next_row + col - i] := \times$

$this.next_row := this.next_row + 1$

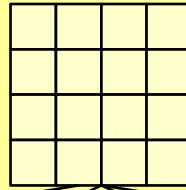
*time complexity
of effect is linear*

Initial: $this.board$ is empty, $this.next_row := 1$

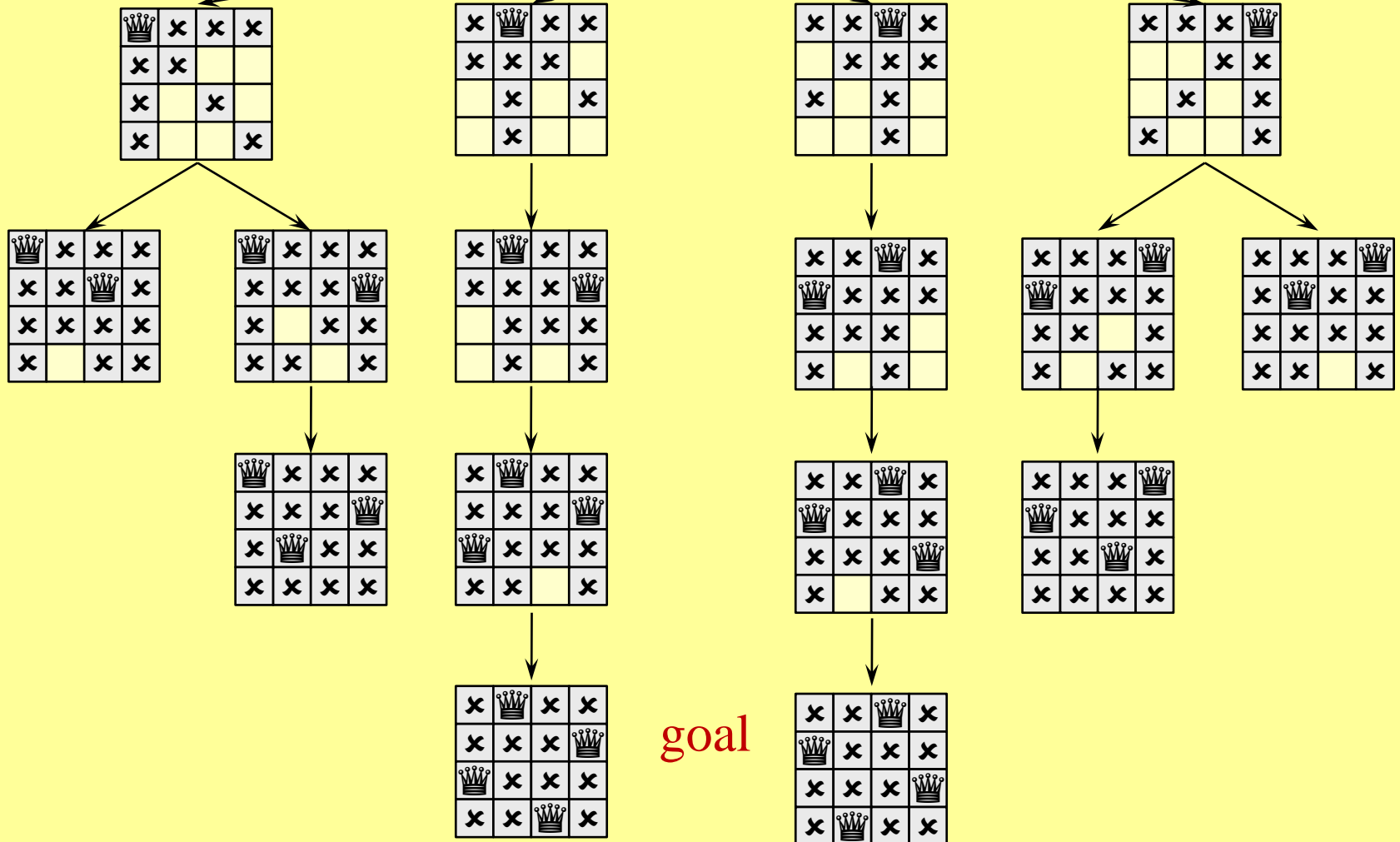
Final: $this.next_row = n + 1$

goal condition becomes very simple

start

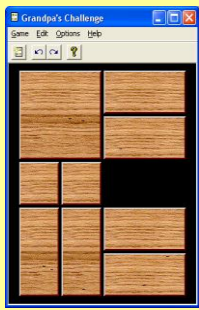


state-graph



Gregorics Tibor

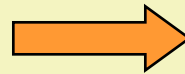
Artificial intelligence



8-puzzle

initial state:

2	8	3
1	6	4
7		5



final state:

1	2	3
8		4
7	6	5

State-space: $ST = \text{rec}(\text{table}: \{0..8\}^{3 \times 3}, \text{empty}: \{1..3\} \times \{1..3\})$

invariant: the elements of the *table* is a permutation of $0..8$
empty gives the coordinates of the empty cell that is
denoted with 0

it is computed
coordinate
by coordinate

Operator: $\text{Move}(\text{dir}): ST \rightarrow ST$ (this: ST)

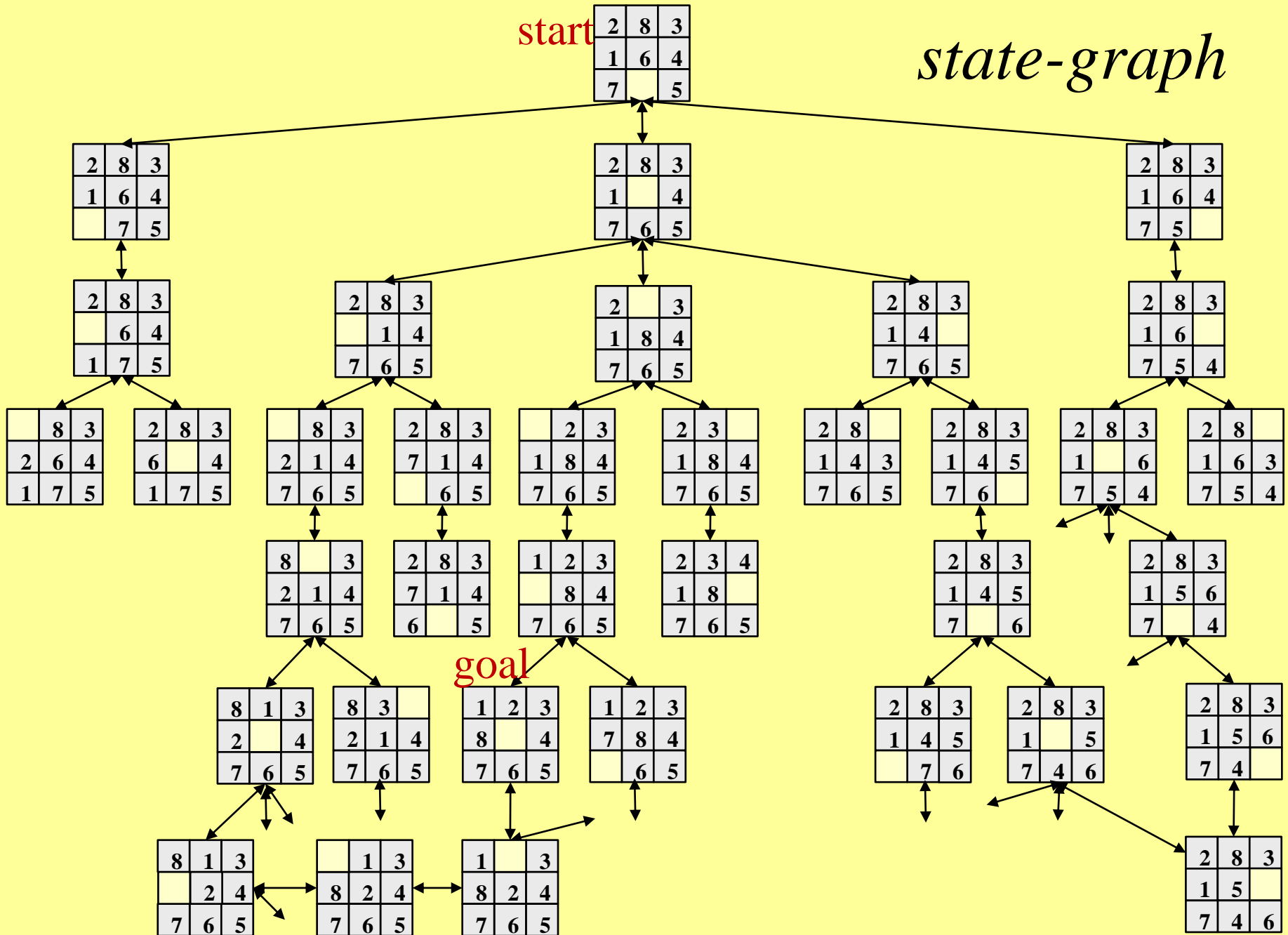
IF $\text{dir} \in \{(0,-1), (-1,0), (0,1), (1,0)\}$ and $(1,1) \leq \text{this.empty} + \text{dir} \leq (3,3)$

THEN $\text{this.table}[\text{this.empty}] \leftrightarrow \text{this.table}[\text{this.empty} + \text{dir}]$

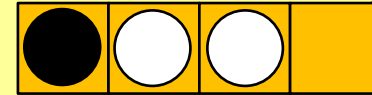
$\text{this.empty} := \text{this.empty} + \text{dir}$

start

state-graph



Black&White puzzle



There are n black and m white stones and one empty place in a linear frame with $n+m+1$ length. A stone can slide to the adjacent empty place or it can jump over one stone onto an empty place. Initially black stones precede the white stones. Let's reverse the order of black and white stones!

State-space: $ST = \text{rec}(s : \{B, W, _ \}^{n+m+1}, \text{pos} : [1.. n+m+1])$

invariant: pos is the index of the single empty place, the number of B is n , and the number of W is m

Operators: *MoveLeft, MoveRight, JumpLeft, JumpRight*

e.g.: $\text{MoveLeft} : ST \rightarrow ST$ (empty space is moved)

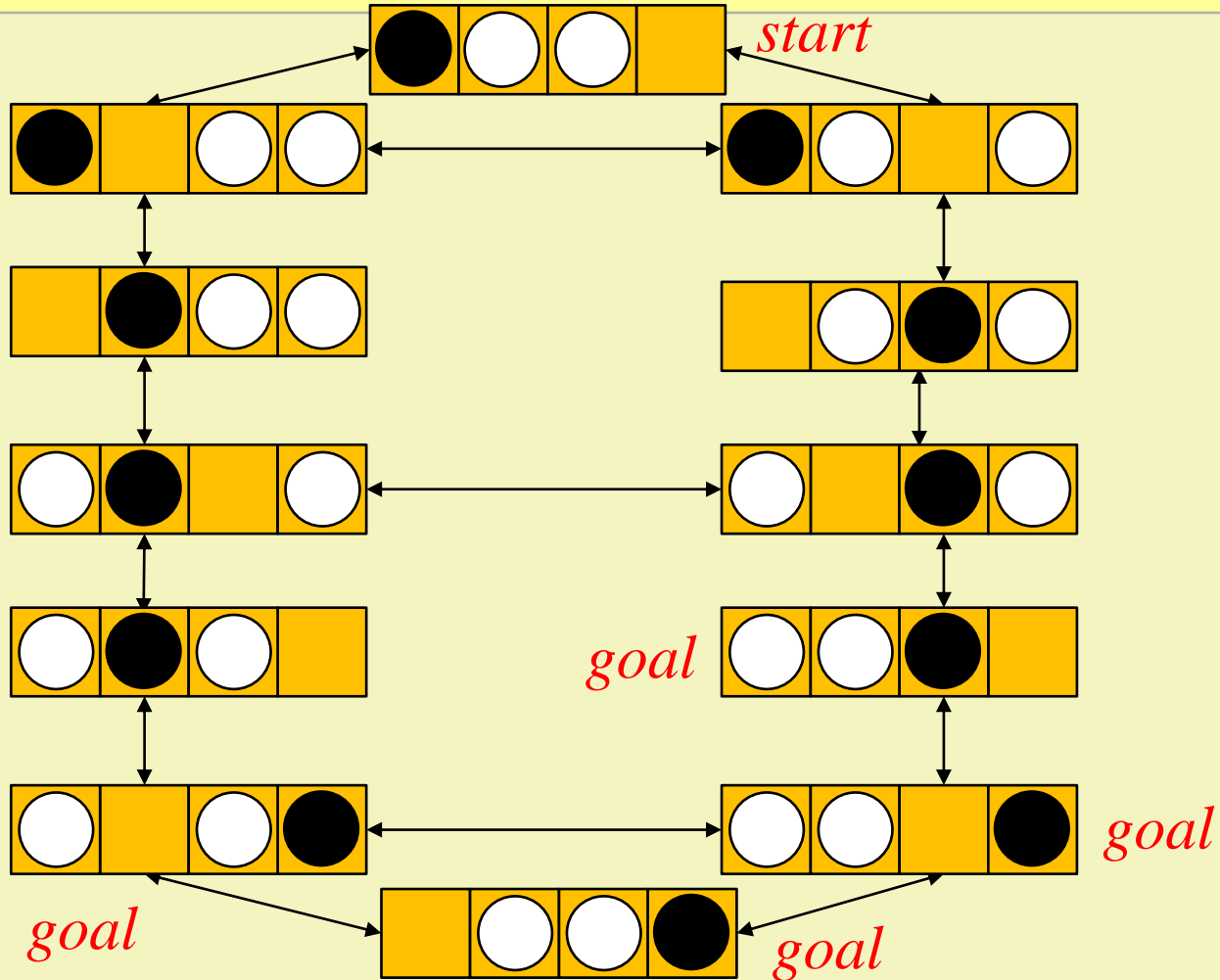
IF $\text{this.pos} \neq 1$ ($\text{this} : ST$)

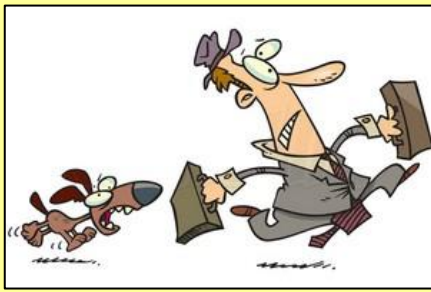
THEN $\text{this.s}[\text{this.pos}-1] \leftrightarrow \text{this.s}[\text{this.pos}] ; \text{this.pos} := \text{this.pos}-1$

Initial: $[B, \dots, B, W, \dots, W, _]$

Final: $\forall i, j \in [1.. n+m+1], i < j : \neg(\text{this.s}[i]=B \wedge \text{this.s}[j]=W)$

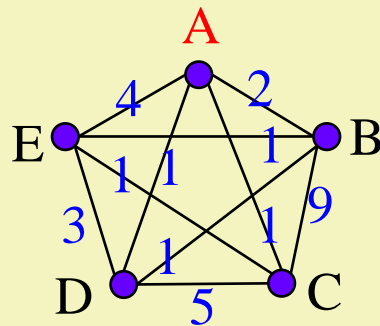
state-graph of Black&White puzzle





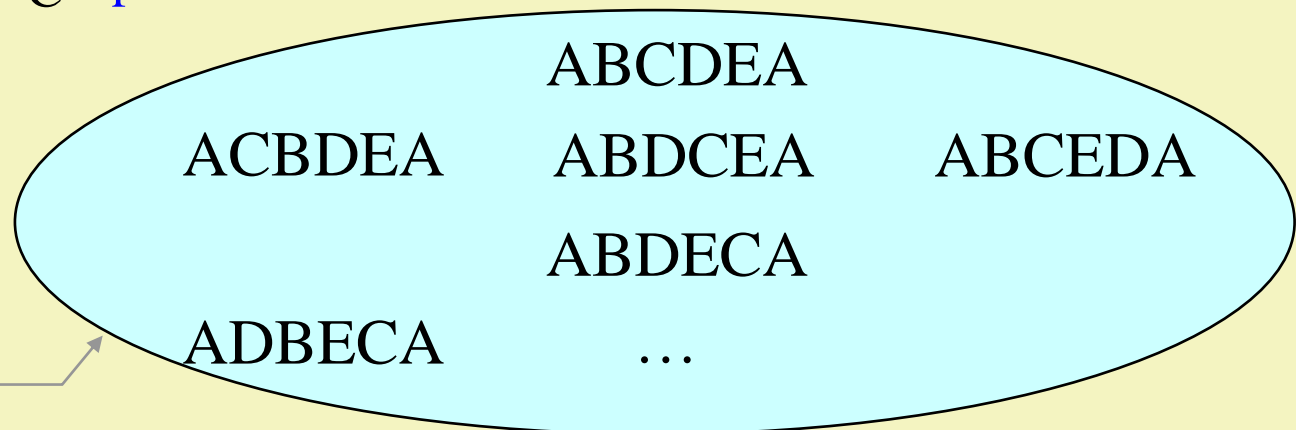
Travelling salesman problem

The traveling salesman must visit every city in his territory exactly once and then return home covering the optimal total cost. (n cities and cost of each pair of cities are known)



possible solutions:

n	$(n-1)!$
5	24
50	$6 \cdot 10^{62}$



problem space



Travelling salesman problem

The traveling salesman must visit every city in his territory exactly once and then return home (n cities and cost of each pair of cities are known) covering the optimal total cost.

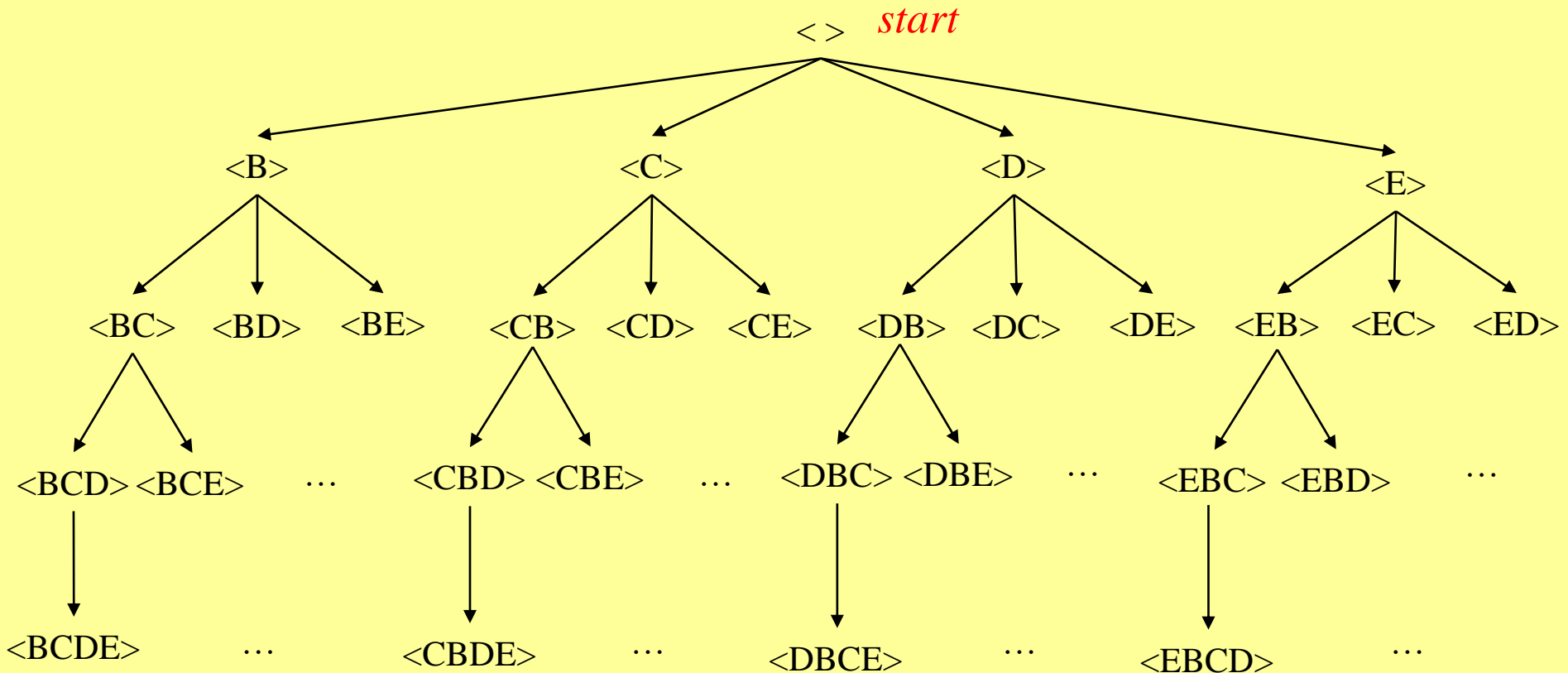
State-space: $ST = \{\text{cities}\}^*$ (set of finite sequences of cities
without home city)

Operator: $\text{Goto}(\text{city}): ST \rightarrow ST$ $\text{city} \in \{\text{cities}\}$
IF $\neg \text{this.contains}(\text{city})$ (this: ST)
THEN $\text{this.append}(\text{city})$

Initial state: $\langle \rangle$ (empty sequence)

Final state: $|\text{this}| = n - 1$ (length of this is n)

state-graph of travelling salesman



SAT – satisfiability problem

There is given a Boolean statement in CNF with n variables. Find a vector of truth assignments for the variables so that the formula be true.

*E.g.: $F(x_1, \dots, x_5) = (x_1 \vee \neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_5)$
possible solution: $x_1 = \text{true}, x_2 = \text{any}, x_3 = \text{any}, x_4 = \text{true}, x_5 = \text{true}$*

State-space: $ST = \mathbb{L}^n$

Operator: $\text{Change}(i): ST \rightarrow ST \quad i \in [1..n] \quad (\text{this} : ST)$
 $\text{this}[i] := \neg \text{this}[i]$

Initial state: arbitrary

Final state: $F(\text{this})$ is true

SAT – satisfiability problem 2.

There is given a Boolean statement in CNF with n variables. Find a vector of truth assignments for the variables so that the formula be true.

State-space: $ST = \text{rec}(t : \{\text{true}, \text{false}, \emptyset\}^n, i : \mathbb{N}, \text{count} : \mathbb{N})$

invariant: $0 \leq i \leq n, \forall j \in \{1 \dots i\}: t[j] \neq \emptyset$

count ($\leq C$) = the number of the clauses having true value

Operator:

True: $ST \rightarrow ST$

$i := i+1: \text{this}.t[i] := \text{true}$

$\text{update}(\text{this}.count)$

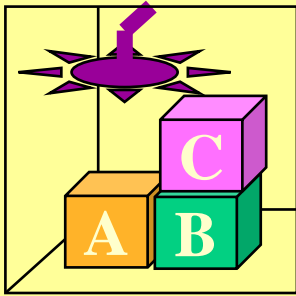
False: $ST \rightarrow ST \quad (\text{this} : ST)$

$i := i+1: \text{this}.t[i] := \text{false}$

$\text{update}(\text{this}.count)$

Initial state: $([\emptyset, \dots, \emptyset], 0, 0)$

Final state: $\text{this}.count = C$



Block world problem

There are some blocks (A,B,C,...). A robot arm can move the blocks: pickup, putdown, stack, unstack. Let's build a given formation!

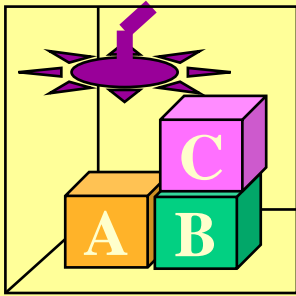
State-space: $ST = \text{set}(\text{ground literals})$

ground literals = { *ontable(A)*, *on(C,B)*, *clear(C)*, ... } where the following predicates occur : *ontable(x)*, *on(x,y)*, *clear(x)*, *handempty*, *holding(x)*

invariant: all states are consistent (e.g.: *on(C,B)* and *clear(B)* is impossible)

Initial: { *ontable(A)*, *clear(A)*, *ontable(B)*, *on(C,B)*, *clear(C)*, *handempty* }

Final: { *on(A,B)*, *on(B,C)* }



Operators of block world problem

Pickup(x): $ST \rightarrow ST$ $x \in \{A, B, C, \dots\}$ ($this : ST$)

IF $ontable(x), clear(x), handempty \in this$

THEN $this := this - \{ontable(x), clear(x), handempty\} \cup \{holding(x)\}$

Putdown(x): $ST \rightarrow ST$ $x \in \{A, B, C, \dots\}$ ($this : ST$)

IF $holding(x) \in this$

THEN $this := this - \{holding(x)\} \cup \{ontable(x), clear(x), handempty\}$

Stack(x, y): $ST \rightarrow ST$ $x, y \in \{A, B, C, \dots\}$ ($this : ST$)

IF $holding(x), clear(y) \in this$

THEN $this := this - \{holding(x), clear(y)\} \cup \{on(x, y), clear(x), handempty\}$

Unstack(x, y): $ST \rightarrow ST$ $x, y \in \{A, B, C, \dots\}$ ($this : ST$)

IF $on(x, y), clear(x), handempty \in this$

THEN $this := this - \{on(x, y), clear(x), handempty\} \cup \{holding(x), clear(y)\}$

state-graph

