STATE-SPACE MODEL

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)

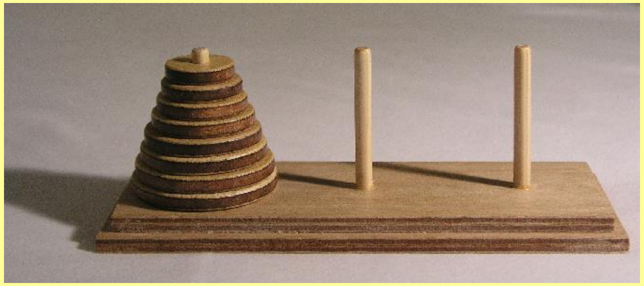
Graph-representation of state-space model

State-space model	Sate-graph
▪ state	node
▪ effect of an operator on a state	directed arc
▪ cost of an operator	cost of arc
▪ initial state	start node
▪ final state	goal node

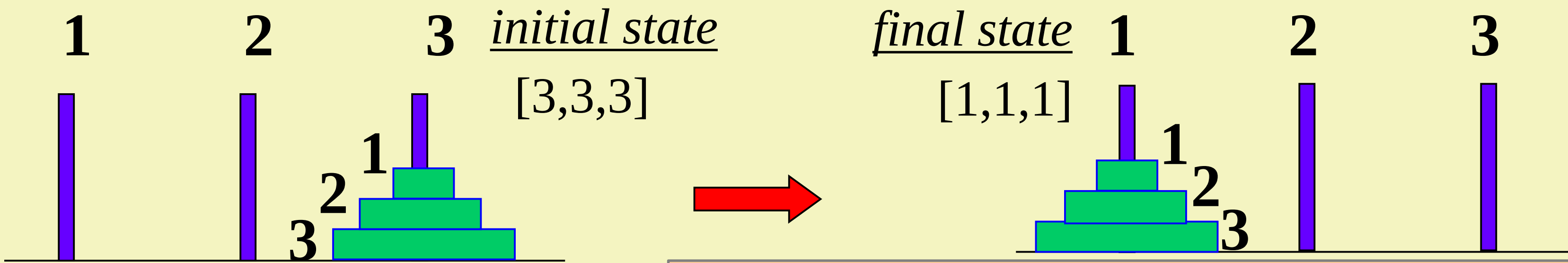
- Graph-representation:
  state-graph, start node, goal nodes

▪ sequence of operators	directed path
▪ solution	directed path from start to goal

representation graph, δ-graph



Hanoi tower problem



<u>*State-space*</u>: ***ST*** **=** $\{1,2,3\}^n$

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

<u>*Operator:*</u> ***Move****(from, to)*: ***ST*** $\rightarrow$ ***ST*** *from*, *to* $\in \{1,2,3\}$

IF *'from'* and *'to'* are <u>valid</u> and <u>different</u> pegs

and <u>there is</u> a disc on *'from'*

and *'to'* is either <u>empty</u> or its <u>upper disc is greater</u> than the disc is wanted to move (this is the upper disc on *'from'*)

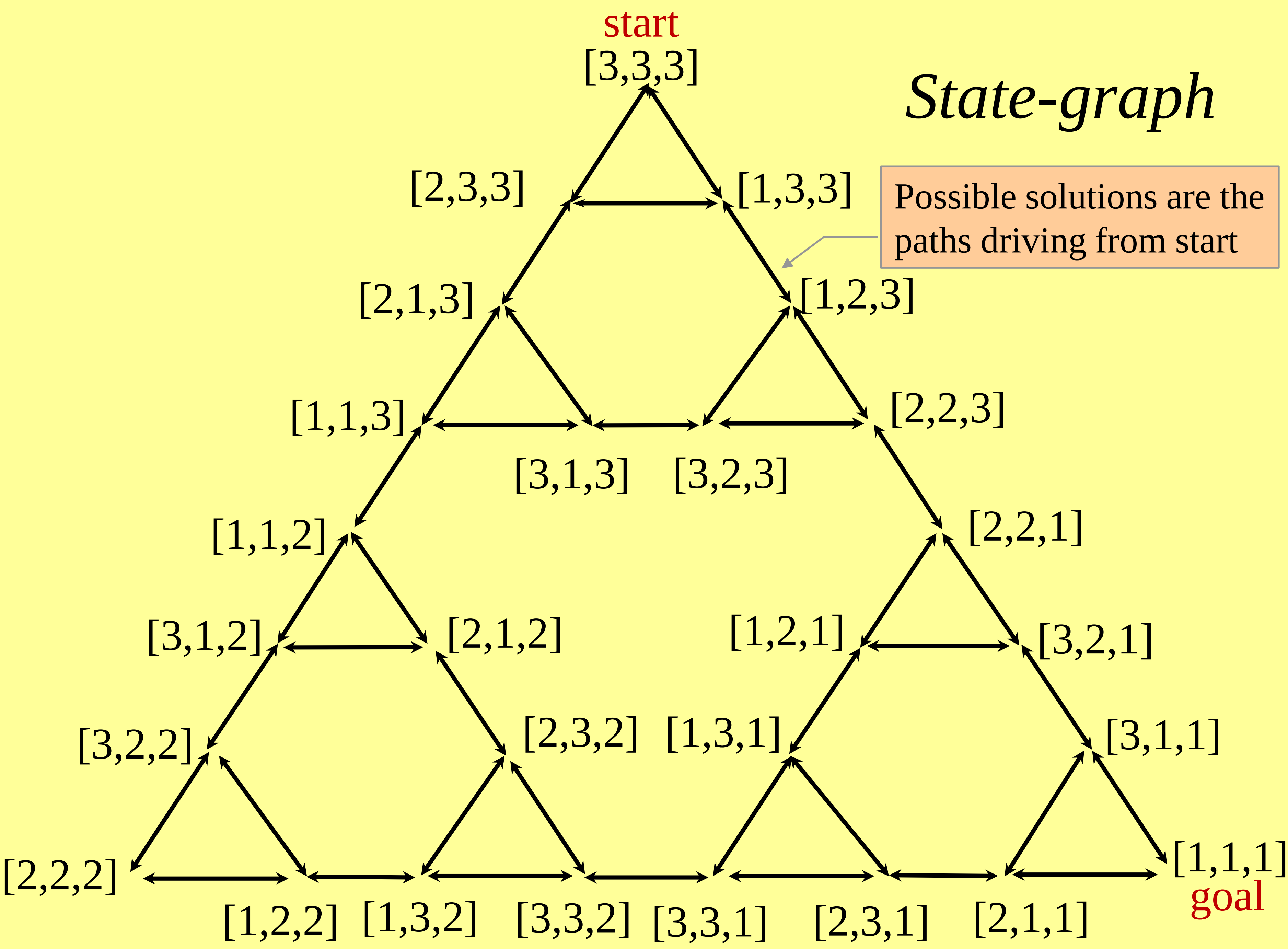
THEN *this*[the upper disc on *'from'*]:= *to*

this is the current state

Implementation

```
template <int  n = 3>  class Hanoi {
    int  _a[n];  // its elements are between 1 and  3
public :
    bool move (int from, int to) {
        if((from<1 || from>3 || to<1 || to>3) || (from==to)) return false;
        bool l1; int i; // l1~'from' is not empty, i~upper disc on 'from'
        for(l1=false, i=0; !l1 && i<n; ++i) l1 = _a[i]==from;
        if (!l1) return false;
        bool l2; int j;  // l2~'to' is not empty, j~upper disc on 'to'
        for(l2=false, j=0; !l2 && j<n; ++j) l2 = _a[j]==to;
        if(¬l2 || i<j){ _a[i] = to;  return true; } else return false;
    }
    bool final() const {
        for(int i=0; i<n; ++i) if(_a[i]!=1) return false;
        return true;
    }
    void init() { for(int i=0; i<n; ++i) _a[i] = 3; }
};
```

State-graph



Possible solutions are the paths driving from start

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.

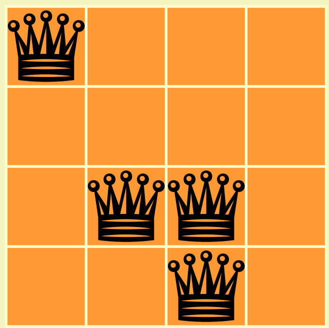
Complexity of state-graph

complexity of state-graph → size of problem space → computational cost of search

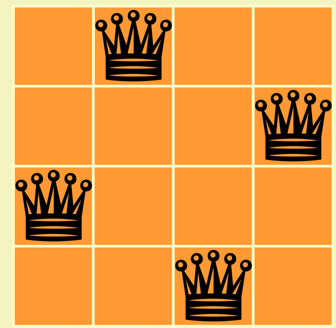
- number of paths driving from the start
  Hanoi: eliminating the *2*-length cycles, the number of at most *k*-length paths: $1+2+\ldots+2^k$, i.e. $2^{k+1}-1$
  - number of nodes and arcs
    Hanoi: 3^n nodes, $3 \cdot \dfrac{3^n-1}{2}$ arcs
  - branching factor: average number of outgoing arcs
    Hanoi: *3*
  - frequency of the cycles and diversity of their length
    Hanoi: 2, 3, 6, 7, 8, 9, …

n-queens problem 1.

general state



final state



State-pace: ***ST*** = {♛, _ }$^{n\times n}$

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

invariant: number of queens (♛) = n

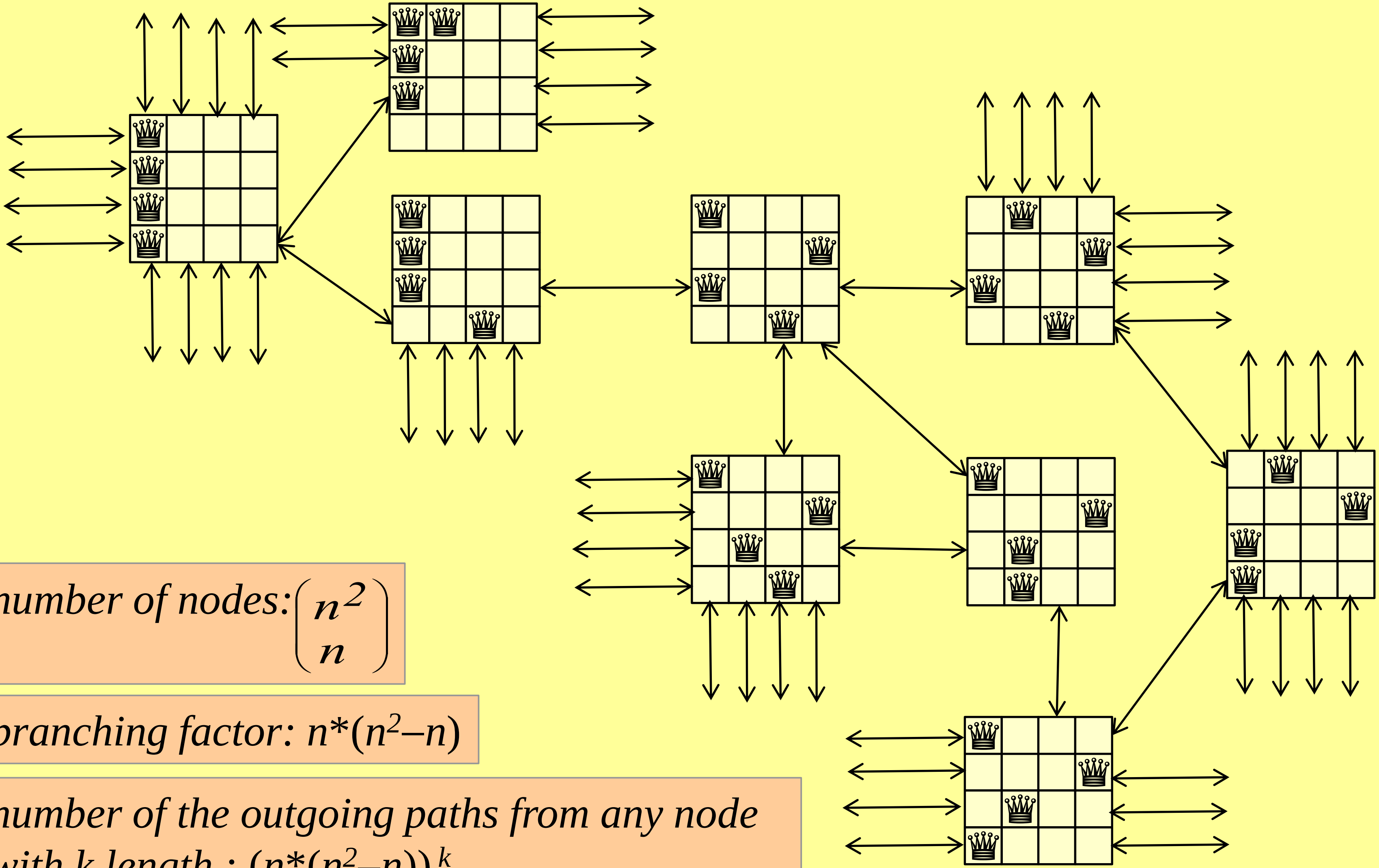
Operator: ***Change***(x,y,u,v): ***ST*** → ***ST*** $x,y,u,v\in[1..\ n]$ (*this*:***ST***)

IF $1\leqq x,y,u,v \leqq n$ and *this*$[x,y]$=♛ and *this*$[u,v]$=_

THEN *this*$[x,y] \leftrightarrow$ *this*$[u,v]$

swap

State-graph



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

branching factor: $n*(n^2-n)$

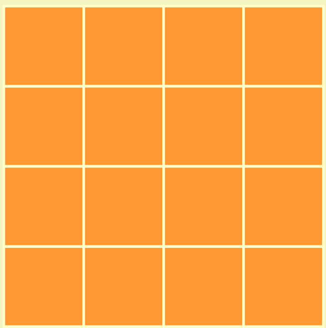
number of the outgoing paths from any node with k length : $(n*(n^2-n))^k$

Reduce the problem space

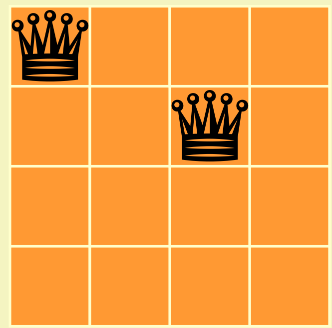
- A problem may have several models:
  the best model = the smallest problem space
  - In the previous representation the size of the problem space is huge.
  - Expand the state space with the states containing less queens than *n*, and use a new operator: put a new queen on the board (the initial state 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 containing attack.

n-queens problem 2.

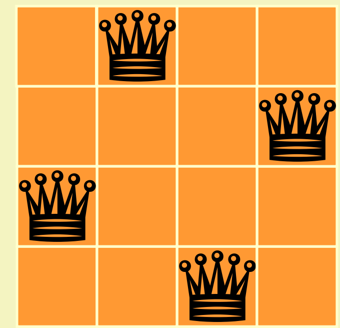
initial state



general state



final state



State-space: ***ST*** = { ♛, _ }$^{n \times n}$

invariant: number of queens (♛) ≤ *n* and

only in the first few rows can be found one-one queen

Operator: ***Put***(*col*): ***ST*** → ***ST*** *col*∈[*1.. n*] (*this*: ***ST***)

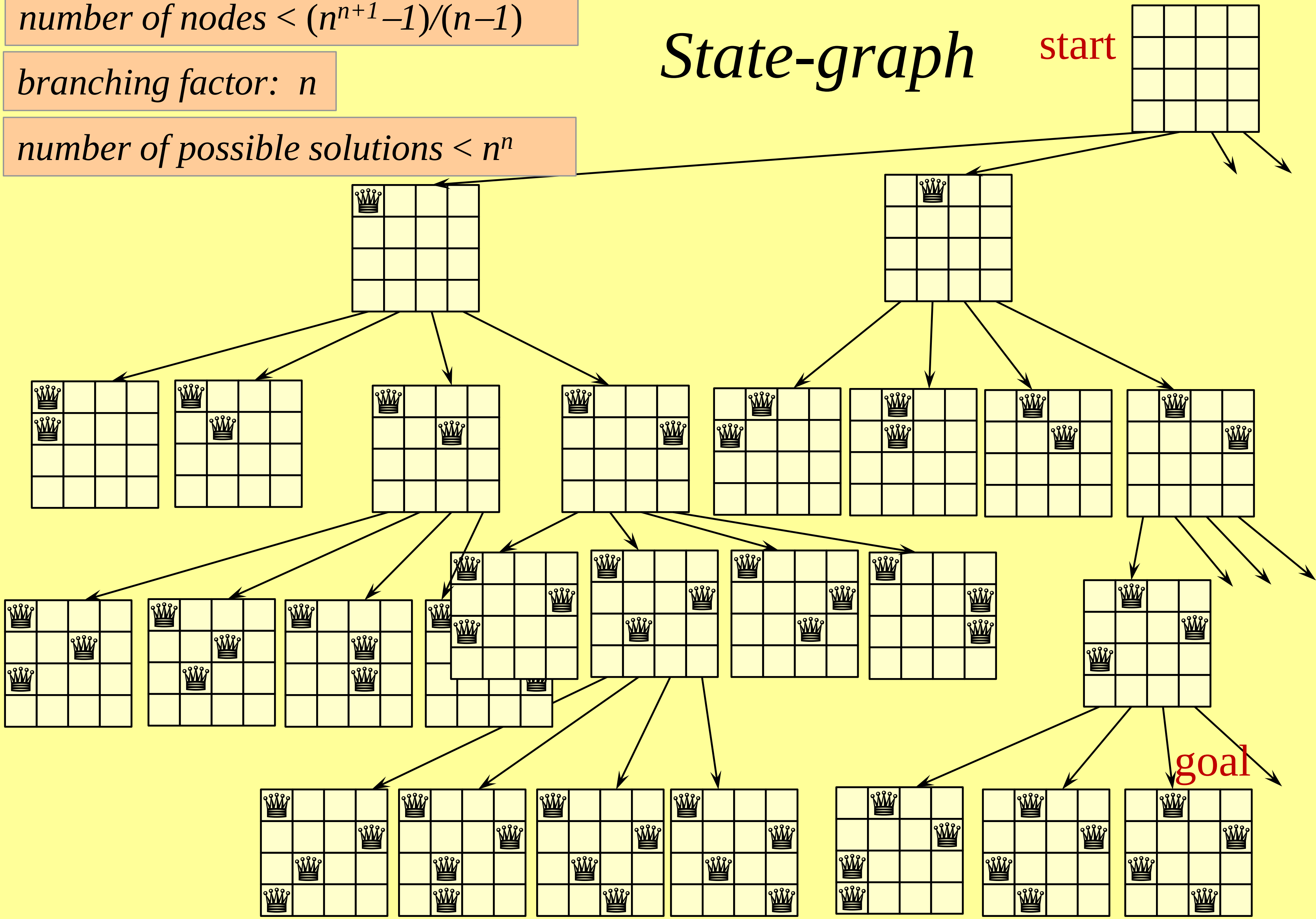
IF *1*≦ *col* ≦*n* and number of queens < *n* and there is no attack

THEN *this*[*row,col*] := ♛ 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



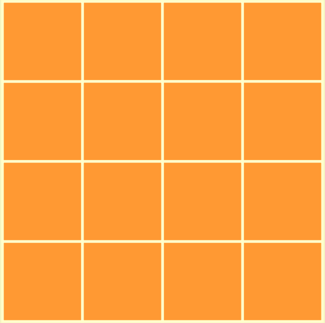
Computational cost of the operator

- 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.

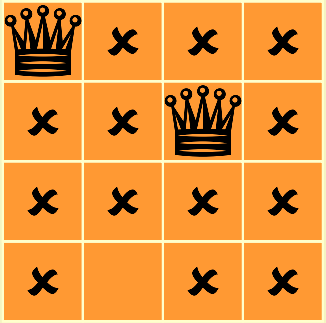
<u>*initial state*</u>*:*

next_row = 1



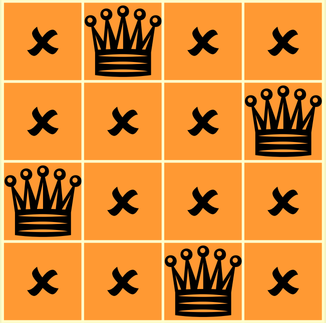
general state:

next_row = 3



<u>*final state*</u>*:*

next_row = 5



<u>*State-space*</u>*:* ***ST*** = *rec*(*board*: {♛, ×, _ }$^{n\times n}$, *next_row*: ℕ)

invariant: only *next_row–1* queens are on the board
in its first *next_row–1* rows one by one,
next_row ≦ *n+1*,
no attacks,
× denotes the empty square under attack
_ denotes the free square

n-queens problem 3.

<u>*Operator*</u>:

Put(*col*): ***ST*** → ***ST*** *col* ∈ [*1*.. *n*] (*this*: ***ST***)

IF *1* ≦ *col* ≦ *n* and *this. next_row* ≦ *n*

and *this.board*[*this. next_row*,*col*]= _

THEN *this.board*[*this.next_row*, *col*] := ♛

∀*i* ∈ [*this. next_row+1..n*]:

this.board[*i*,*col*]:= ×

this.board[*i*,*i*–*this.next_row+col*]:= ×

this.board[*i*, *this.next_row+col*–*i*]:= ×

this.next_row:= *this.next_row+1*

<u>*Initial*</u>: *this.board is empty*, *this.next_row*:=*1*

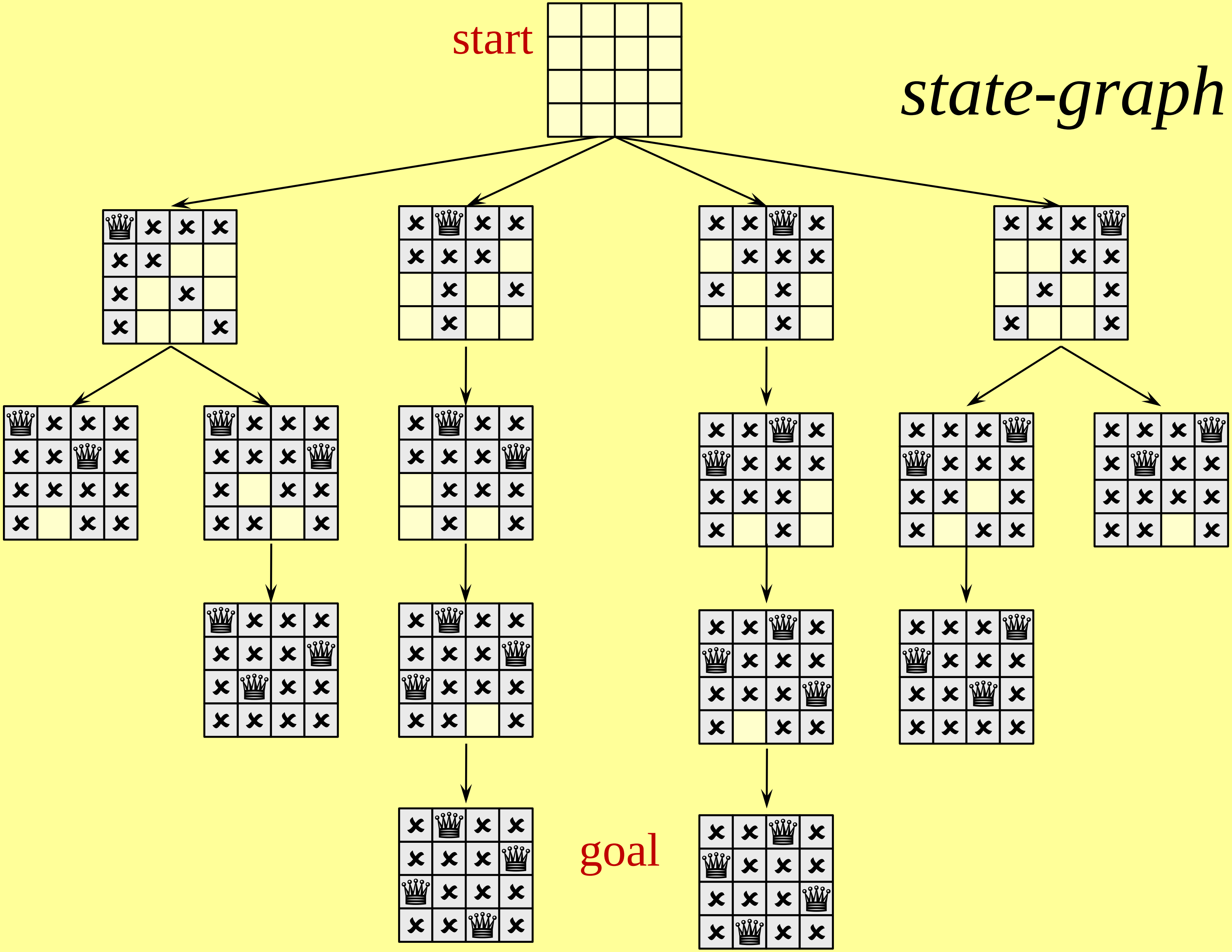
<u>*Final*</u>: *this.next_row*=*n+1*

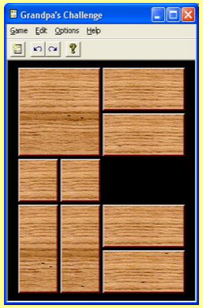
time complexity of precondition is constant

time complexity of effect is linear

goal condition becomes very simple

state-graph





8-puzzle

<u>initial state</u>:

2	8	3
1	6	4
7		5

<u>final state</u>:

1	2	3
8		4
7	6	5

<u>*State-space*</u>: ***ST*** $= rec(table\!:\!\{0..8\}^{3\times 3},\ empty\!:\!\{1..3\}\times\{1..3\})$

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

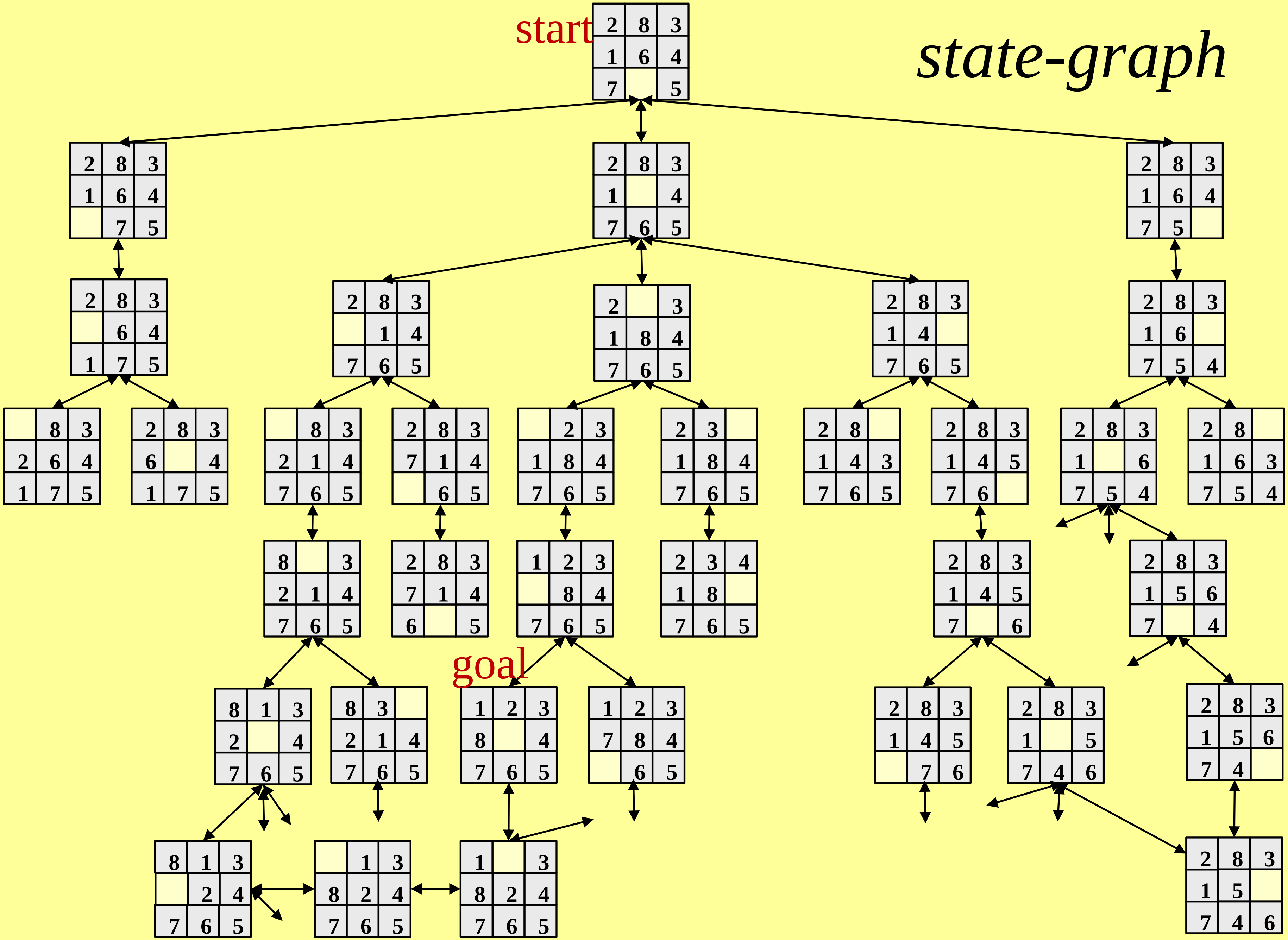
<u>*Operator*</u>: ***Move***(*dir*): ***ST*** → ***ST*** (*this*: ***ST***)

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

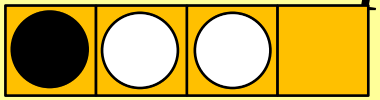
THEN $this.table[this.empty] \leftrightarrow this.table[this.empty+dir]$

$this.empty := this.empty+dir$

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 be slid to the adjacent empty place or it can be jumped 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: $\boldsymbol{ST} = rec(s : \{B, W, \_\}^{n+m+1}, pos : [1..\ n+m+1])$

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

Operators: ***MoveLeft, MoveRight, JumpLeft, JumpRight***

e.g.: ***MoveLeft*** : $\boldsymbol{ST} \to \boldsymbol{ST}$ (empty space is moved)

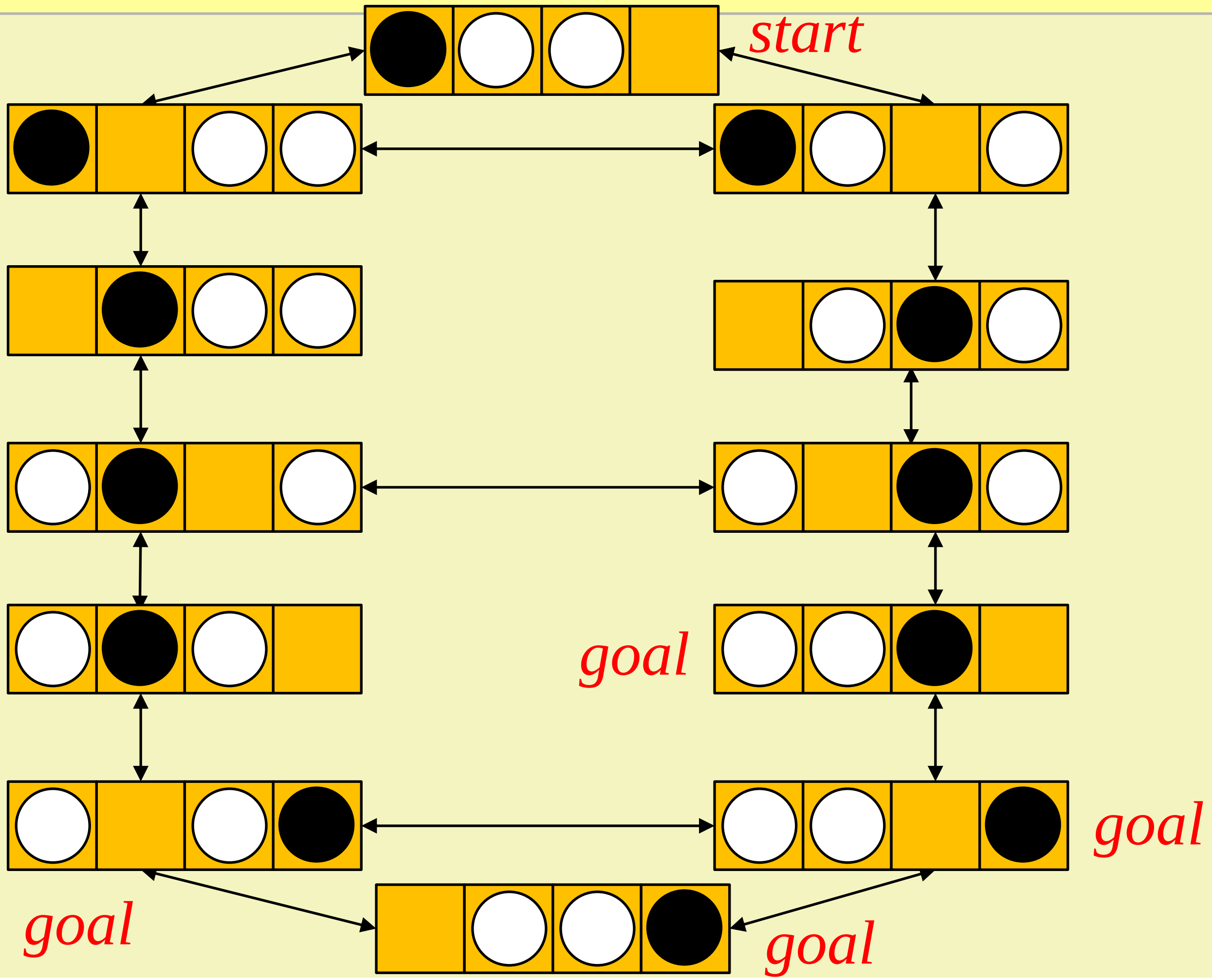
IF $this.pos \neq 1$ $(this : \boldsymbol{ST})$

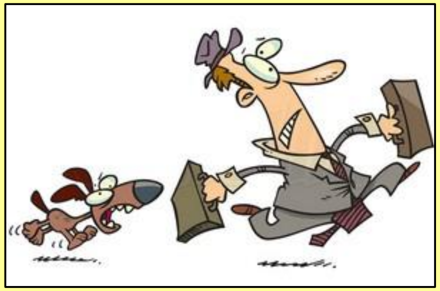
THEN $this.s[this.pos-1] \leftrightarrow this.s[this.pos]$; $this.pos := this.pos-1$

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

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

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 (n cities and cost of each pair of cities are known) covering the optimal total cost.

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

Operator: **Goto**(city): $\boldsymbol{ST} \rightarrow \boldsymbol{ST}$ $city \in \{cities\}$

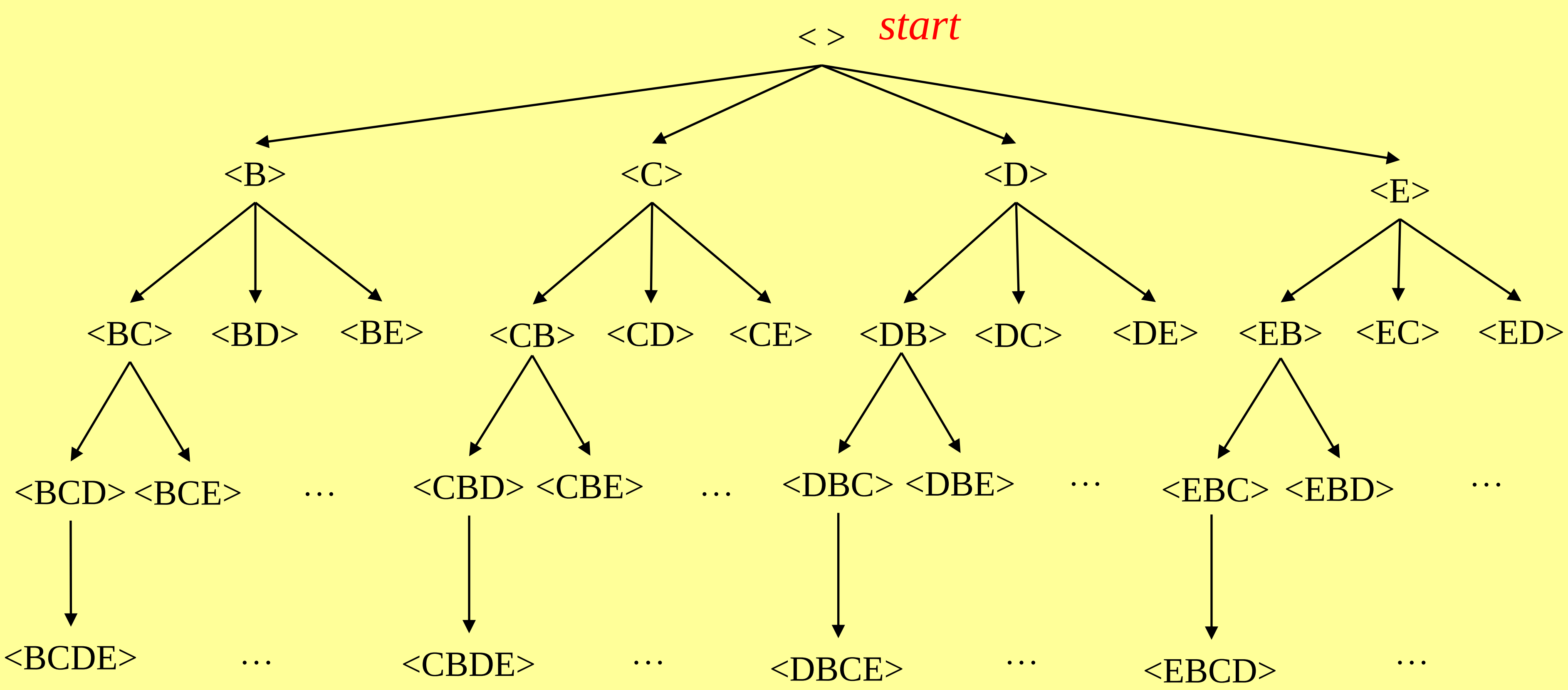
IF $\neg$ *this.contains(city)* (*this*: ***ST***)

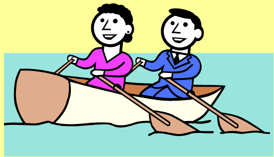
THEN *this.append(city)*

Initial state: < > *(empty sequence)*

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

state-graph of travelling salesman





Missionaries - cannibals problem

n missionaries and n cannibals want to cross a river in a boat that can hold h people in such a way that cannibals never outnumber missionaries on either side of the river or in the boat.

<u>State-space</u>: ***ST*** = $rec(m : [0..n], c : [0..n], b : \mathbb{L})$

invariant: no cannibalism, e.g. $I(m,c) \equiv m=c \lor m=0 \lor m=n$

<u>Initial state</u>: $(n,n,true)$

<u>Final state</u>: $(0,0,false)$

<u>Operators</u>: ***There***(x,y): ***ST*** → ***ST***   ***Back***(x,y): ***ST*** → ***ST*** ($x,y \in \mathbb{N}$, *this*: ***ST***)

IF $this.b$ and $0 \leqq x \leqq this.m$ and $0 \leqq y \leqq this.c$ and $0 < x+y \leqq h$ and $I(this.m-x,\ this.c-y)$
THEN $this.b := false$
$this.m := this.m-x$
$this.c := this.c-y$

IF $\neg this.b$ and $0 \leqq x \leqq n-this.m$ and $0 \leqq y \leqq n-this.c$ and $0 < x+y \leqq h$ and $I(this.m+x,\ this.c+y)$
THEN $this.b := true$
$this.m := this.m+x$
$this.c := this.c+y$

state-graph of missionaries - cannibals

m=3, n=3, h=2