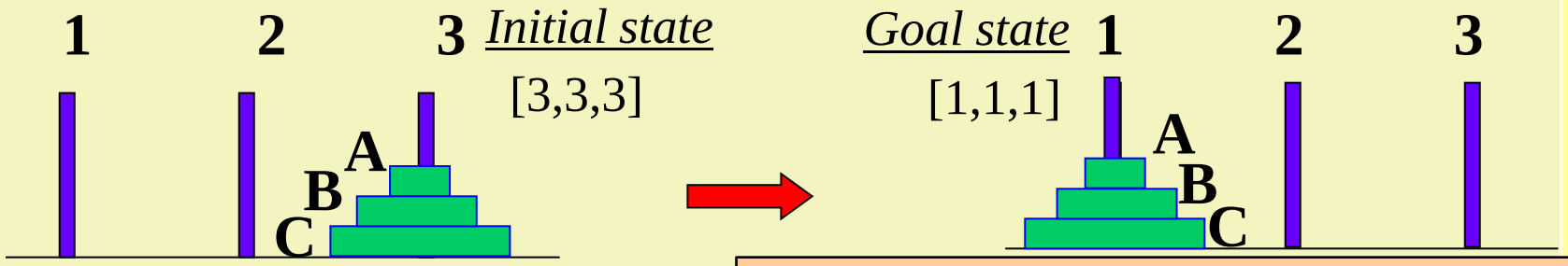
STATE-SPACE REPRESENTATION

Elements of state-space representation

- ***State-space***: set of possible values of the object standing in the focus of the problem
  - These values often have got a complex structure and must satisfy an invariant statement.
- ***Operators***: step from a state to another
  - They map from state-space to state-space so they have got precondition and effect
- ***Initial state(s)*** or its description with an initial condition
- ***Goal state(s)*** or its description with a goal condition

Hanoi tower problem



all possible *n* length sequences (an array) where the elements may be *1*, *2* or *3*.

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

Operator: ***Move****(from, to)*: ***SP*** → ***SP*** (*this*: ***SP***)

IF *'from'* and *'to'* are legal pegs

and there is a disc on *'from'*

and *'to'* is empty or the upper disc on *'to'* is greater than the disc (upper disc on *'from'*) that is moved

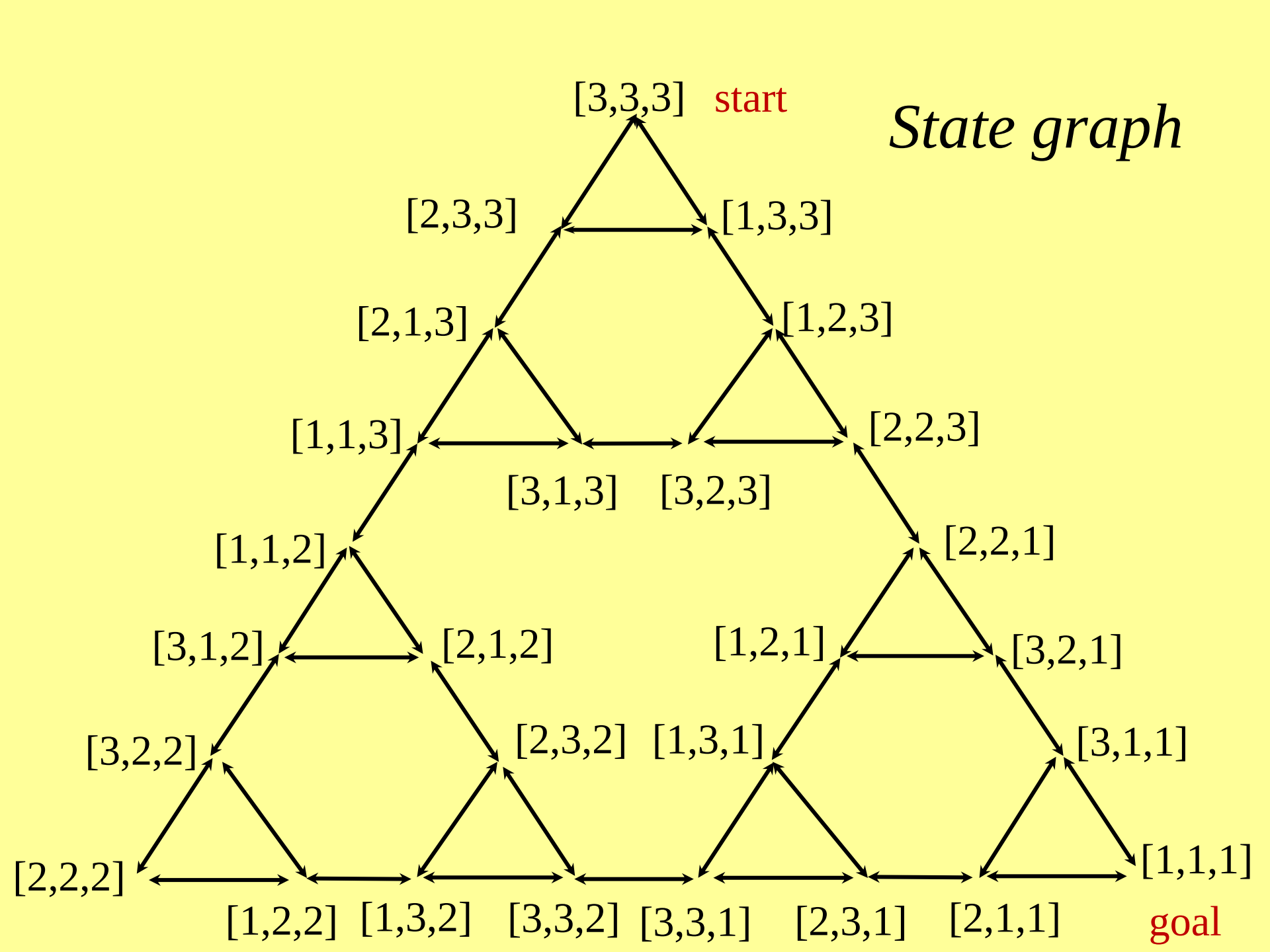
THEN *this*[the upper disc on *'from'*]:= *to*

Implementation

```
template <int n> class Hanoi {
    int a[n];
public :
    bool Move (int from, int to) {
        if (from<1 || from>3  || to<1 || to>3) return false;
        bool l1, l2; int i, j;
        l1,i=search i∈1..n (a[i]==from)  // disc i is wanted  to move
        l2,j=search j∈1..n (a[j]==to)       // onto disc j is wanted to put disc i
        if ( l1 && (¬l2 || i<j)  ){  t[i] = to;  return true; }
        else return false;
     }
     bool Goal() const  {  return  a== [1, … ,1] ; }
     void Init()  const   {  a = [3, … ,3] ; }
};
```

State graph of the state-space representation

state	node
effect of an operator	directed arc
cost of an operator	cost of arc
initial state	start node
goal states	goal nodes
sequence of operators	directed path

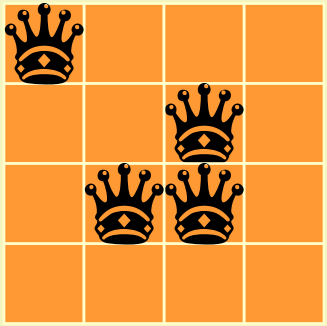

[3,3,3] start
State graph
[2,3,3]
[1,3,3]
[2,1,3]
[1,2,3]
[1,1,3]
[2,2,3]
[3,1,3]
[3,2,3]
[1,1,2]
[2,2,1]
[3,1,2]
[2,1,2]
[1,2,1]
[3,2,1]
[3,2,2]
[2,3,2]
[1,3,1]
[3,1,1]
[2,2,2]
[1,1,1]
[1,2,2]
[1,3,2]
[3,3,2]
[3,3,1]
[2,3,1]
[2,1,1]
goal

Remarks

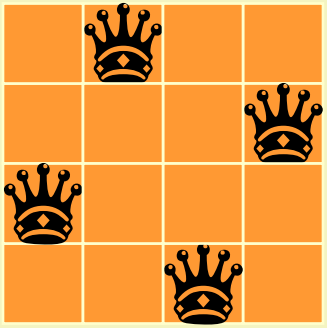
- ***Solution***: a directed path (sequence of operators)
- Computational cost of the finding path algorithm is determined by the complexity of the state graph
  - number of nodes (Hanoi: 3^n)
  - number of outgoing arcs from one node (Hanoi: at most *3*)
  - number of paths going from the start (Hanoi: number of the *k*-length paths without the *2*-length circles is 2^k)
  - length of the circles (Hanoi: 2, 3, 6, 7, 9, …)
- The state-space is not identical to the problem space.
  - In a path-finding problem the elements of the problem space are the paths (sequences of operators) and not the nodes (states).

n-queens problem 1.

general state



goal state



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

State-pace: ***SP*** = { ♕, _ }$^{n \times n}$

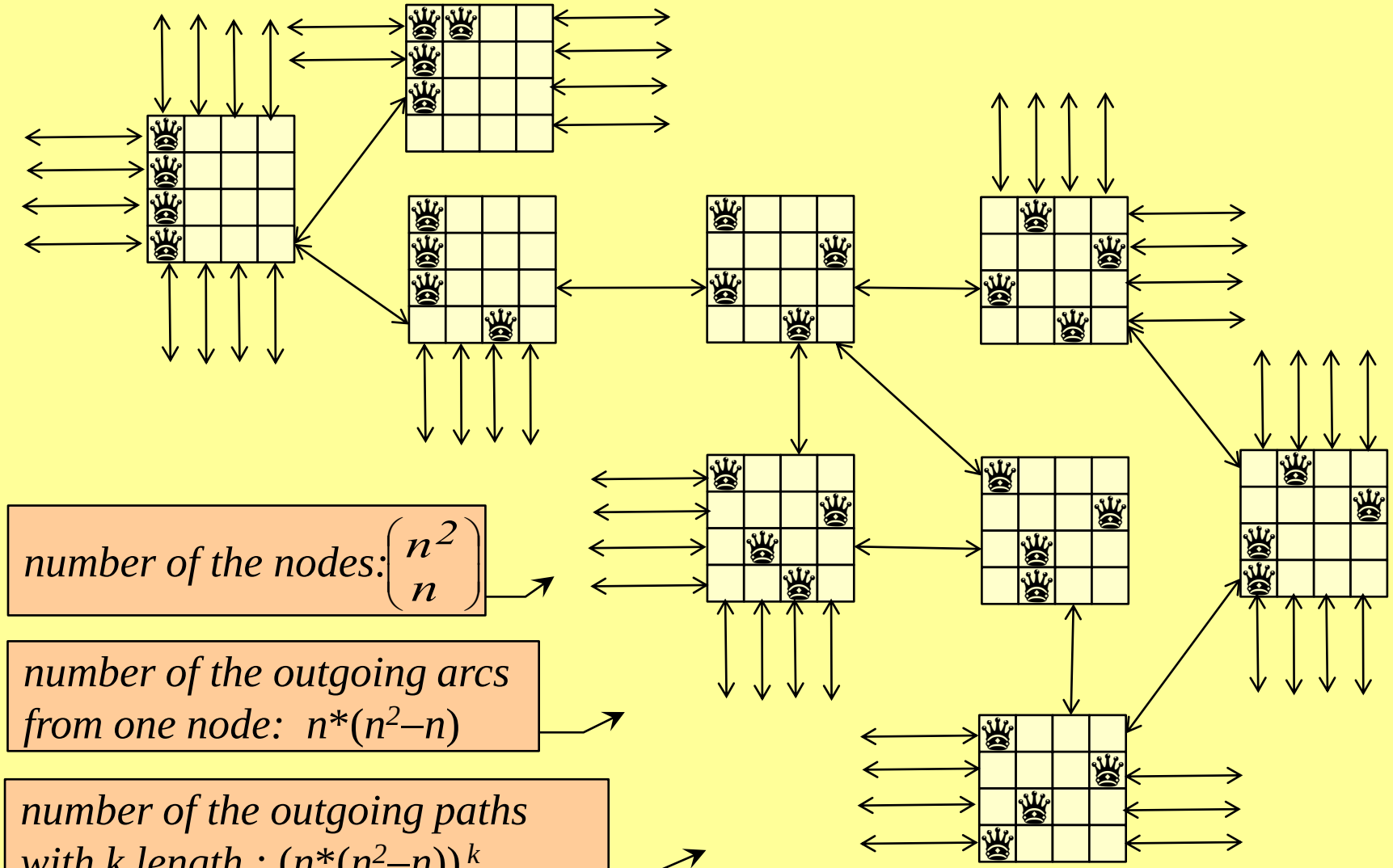
invariant: number of queens (♕) = n

Operator: ***Change***(x,y,u,v): ***SP*** → ***SP*** (*this*: ***SP***)

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

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

State graph



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

number of the outgoing arcs from one node: $n*(n^2-n)$

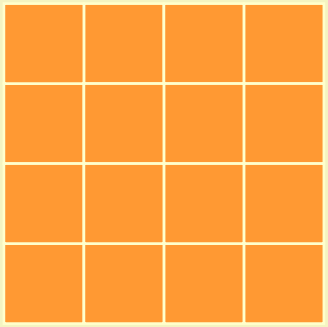
number of the outgoing paths with k length : $(n*(n^2-n))^k$

Reduce the problem space

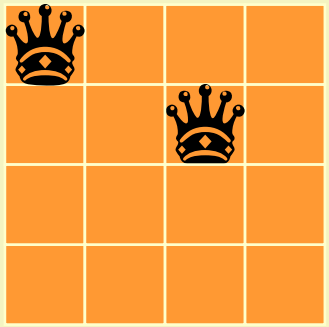
- A problem may have several models and the model with the smallest problem space is looked for
  - In the previous representation the size of the problem space (that is the number of the possible paths) is huge
  - If the state space were expanded with the states where the number of queens was less than *n*, and a new operator were used (put a new queen on the board), then the problem space (that contains the *n*-length paths) would be smaller than the previous one.
    Number of the *n*-length paths: $\binom{n^2}{n} \cdot n!$
  - The problem space can be further reduced with limiting the precondition of the operator:
    - Put the queens row by row. In this case the number of the *n*-length paths : n^n
    - Additionally, the size of the problem space will be decreased if a new queen is never put on the board containing an attack.

n-queens problem 2.

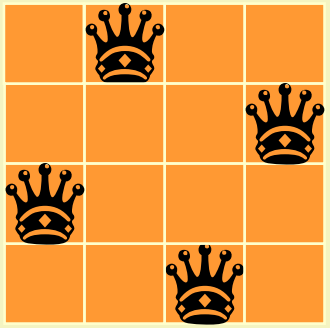
initial state



general state



goal state



State-space: ***SP*** = { ♕, _ }$^{n \times n}$

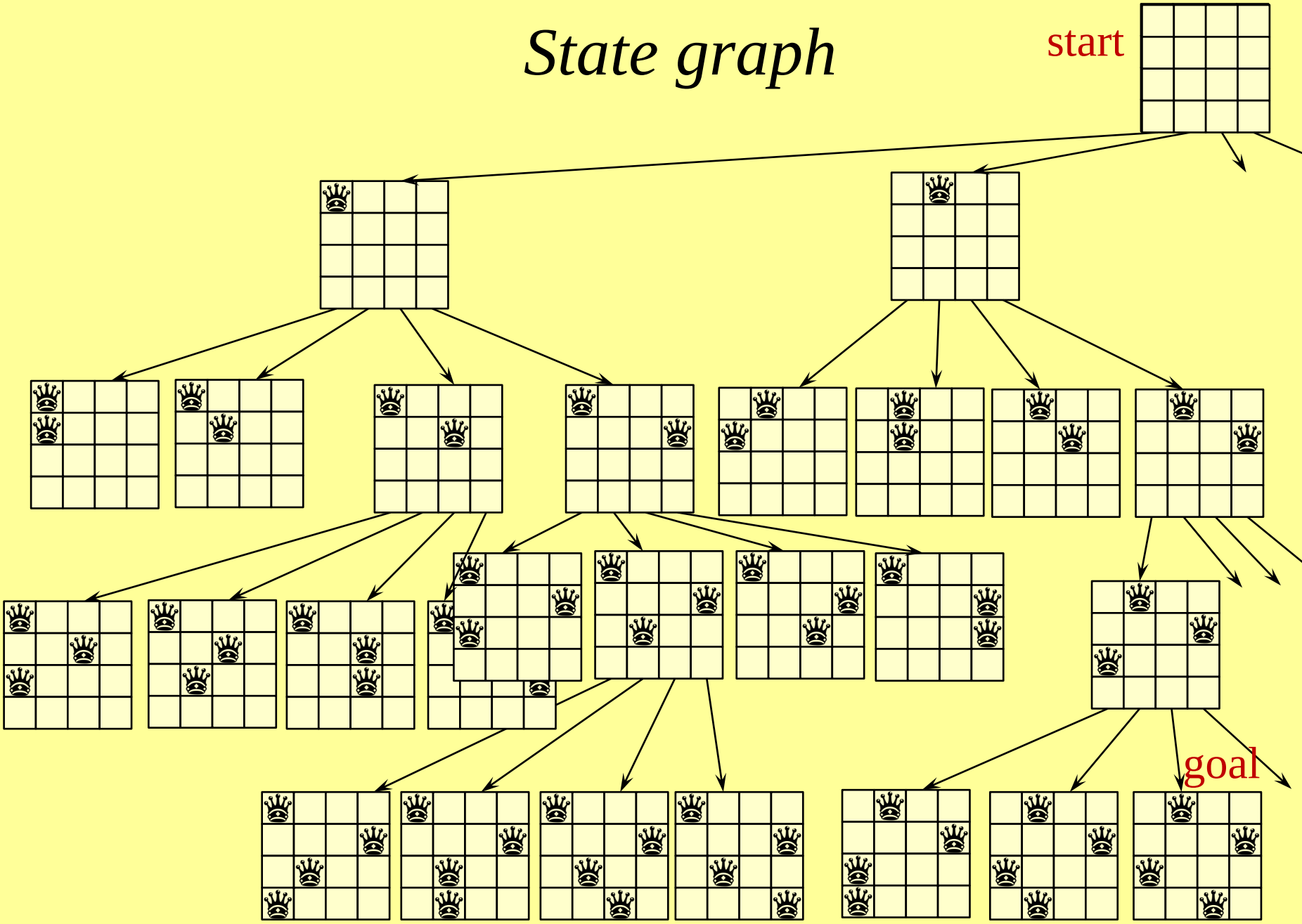
invariant: number of queens (♕) ≤ *n* and
only in the first few rows can be found one-one queen

Operator: ***Put***(*col*): ***SP*** → ***SP*** (*this*: ***SP***)

IF *1*≦ *col* ≦*n* and number of queens < *n* and no attacking
and *„row”* denotes the next empty row

THEN *this*[*row*,*col*] := ♕

State graph



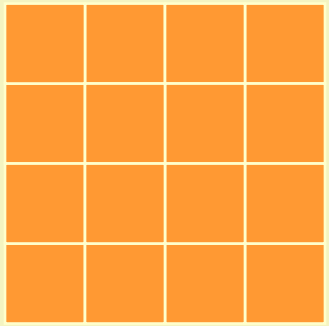
Computational cost of the operator

- The computational cost of the precondition of an operator can be reduced if
  - the state space (including its invariant) is narrowed down and
  - according to this the effects of the operators are modified
- For example
  - The position of the next empty row can be stored in the states instead of computing it.
  - The empty squares that are under attack can be annotated after placing a new queen.
  - Applying the previous idea it will be very simple to avoid all attacks on the board

n-queens problem 3.

initial state:

row = 1



general state:

row= 3

goal state:

row = 5

State-space: ***SP*** = *rec(t:* { ♕, × , _ }$^{n \times n}$, *row:* $\mathbb{N}$)

invariant: only *row–1* queens are on the board in the first *row–1* rows,

row ≦ *n+1*,

no attacking,

× denotes the empty square under attack

_ denotes the free square

n-queens problem 3.

Operator:

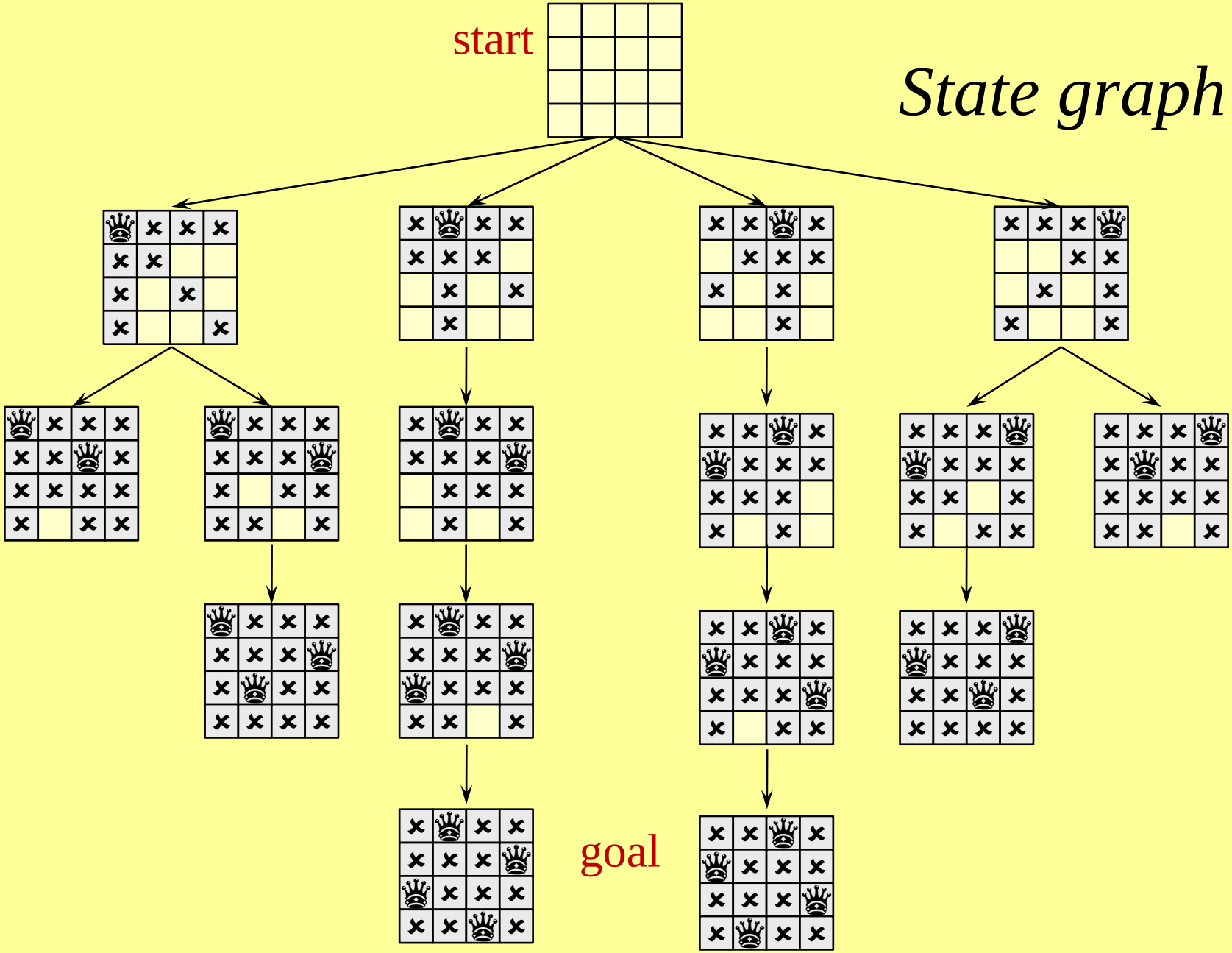
Put(*col*): ***SP*** → ***SP*** (*this*: ***SP***)

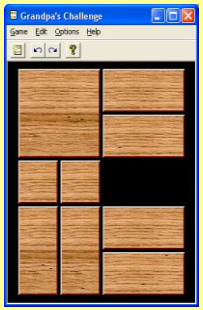
IF $1 \leqq col \leqq n$ and $this.row \leqq n$ and $this.t[this.row,col] = \_$
THEN $this.t[this.row, col] := ♛$
foreach corresponding i,j : $this.t[i,j] := ✖$
$this.row := this.row+1$

Initial: *this.t is empty, this.row:=1*

Goal: $this.row=n+1$

State graph





8-puzzle

initial state:

2	8	3
1	6	4
7		5

→

1	2	3
8		4
7	6	5

goal state:

State-space: ***SP*** $= rec(t:\{0..8\}^{3\times3}, e:\{1..3\}\times\{1..3\})$

invariant: the elements of the matrix is a permutation of *0 .. 8*

0 denotes the empty cell

e contains the coordinates of the empty cell

Operator: ***Move***(*dir*): ***SP*** → ***SP*** (*this*: ***SP***)

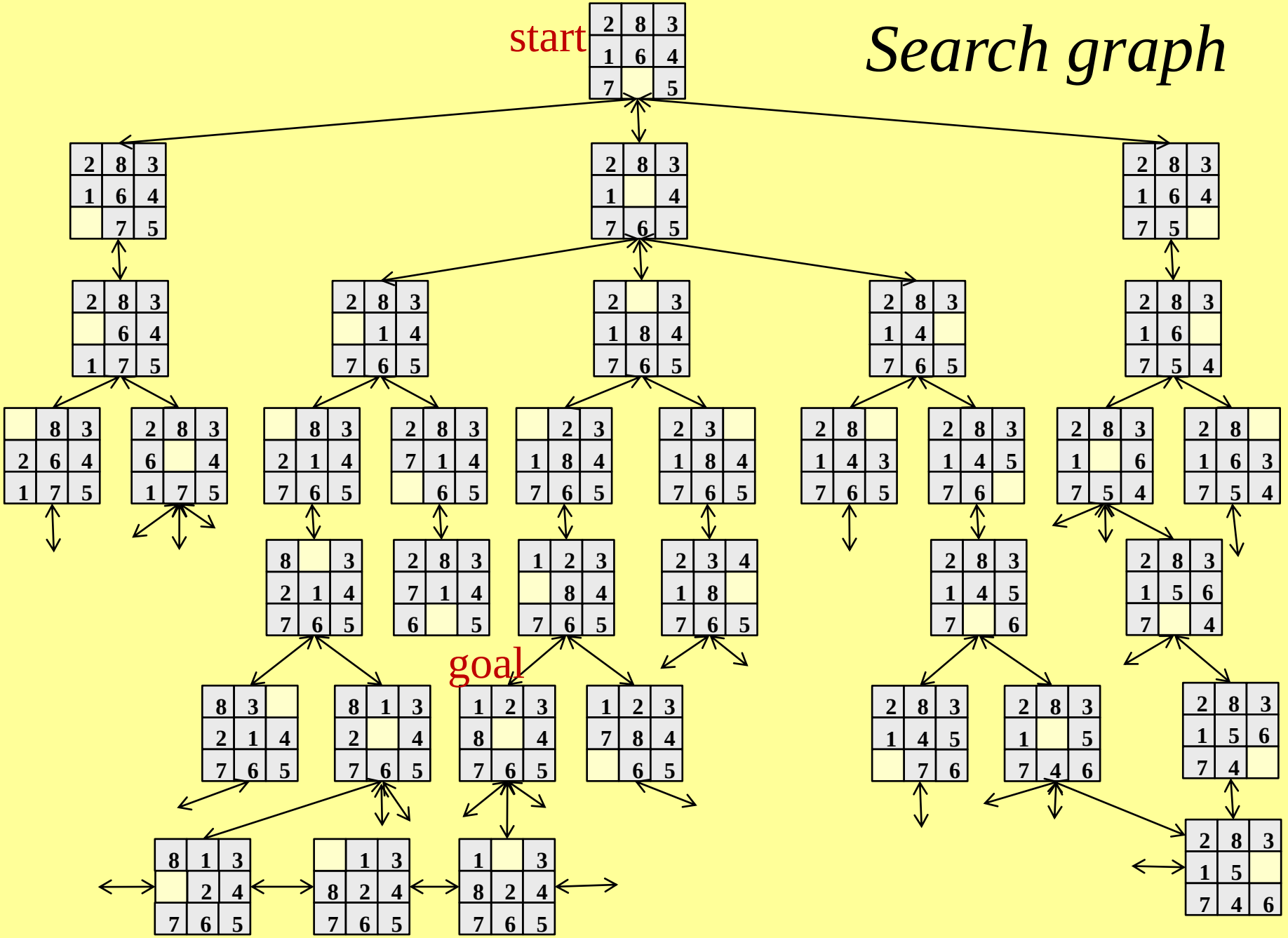
IF $dir \in \{(1,0),(0,1),(-1,0),(0,-1)\}$ and $1 \leqq this.e+dir \leqq 3$

THEN $this\ this.t[this.e] \leftrightarrow this.t[this.e+dir]$

$this.e := this.e+dir$

this.e+dir is computed coordinate by coordinate

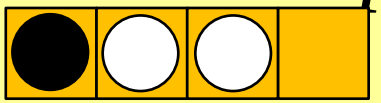
Search graph



Remarks

- A path finding algorithm can never see (or store) the total state graph. Moreover the part discovered can be viewed deformable. The causes of this is that when a successor of a state is generated
  - it may not be checked whether it is among the states generated earlier and a new node will be created for this. advantage: no circles; disadvantage: number of nodes becomes infinite
  - it may be useful to ignore the arc to the successor that is also the parent of the current state.

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 slided to the neighboring empty place or it can be jumped over one stone onto an empty place. Initially all black stones are on the left of the white stones and the empty is the last. Let's reverse the order of black and white stones!

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

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

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

e.g.: ***MoveLeft*** : ***SP*** $\rightarrow$ ***SP*** (empty space is moved)

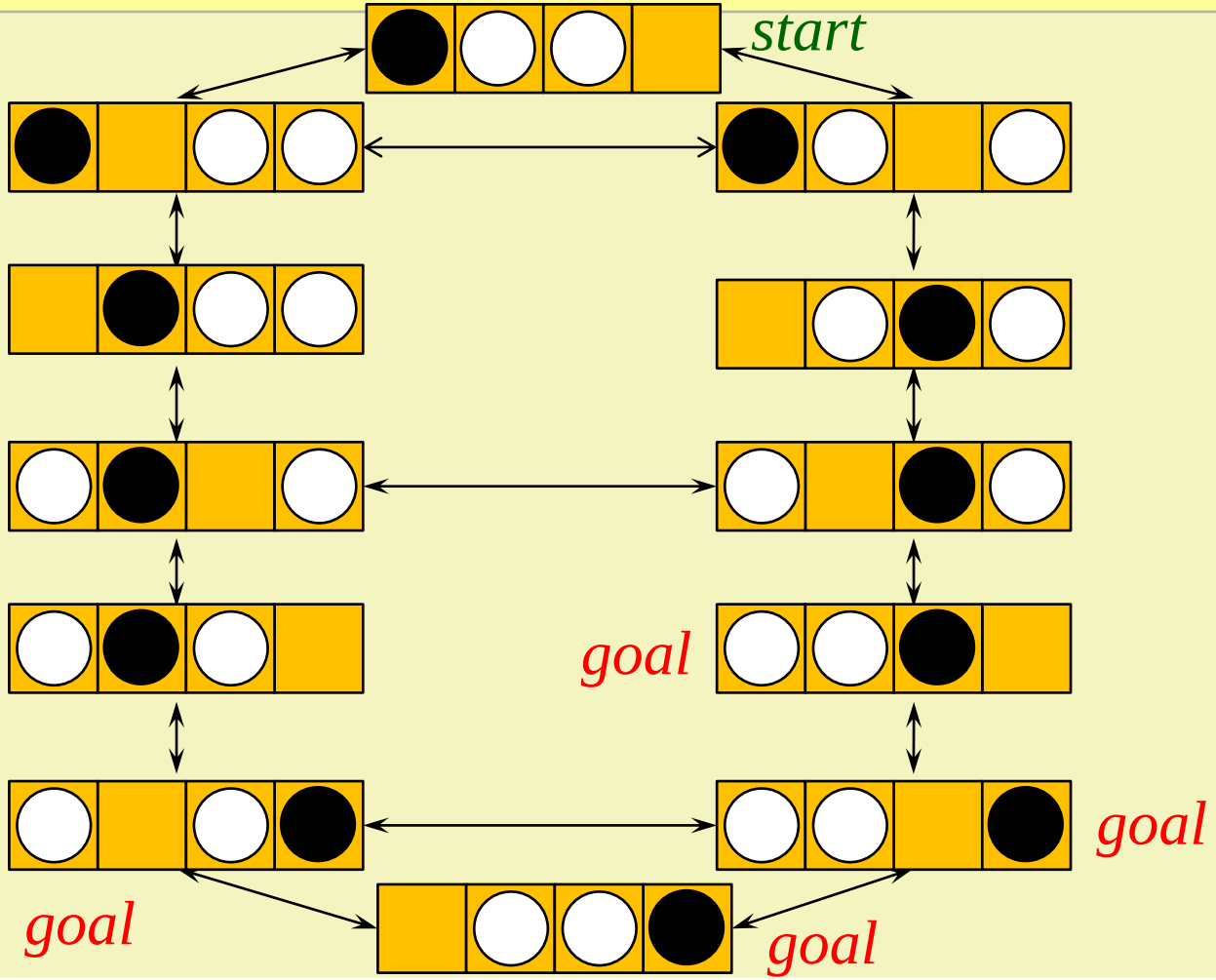
IF $this.pos \neq 1$ ($this$: ***Frame***)

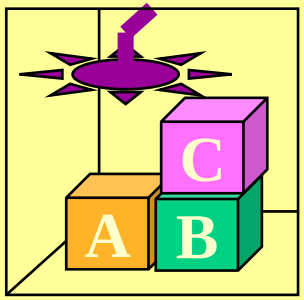
THEN $this.s[this.pos-1] \leftrightarrow this.s[this.pos]$; $this.pos := this.pos-1$

Initial: [*B, … , B, W, … , W, _*]

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

State graph of Black&White puzzle





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!

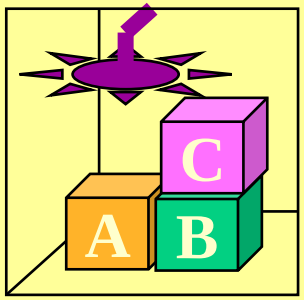
State-space: ***SP*** = *set*(*ground literals*)

ground literals={ontable(A), on(C,B), clear(C), … } where the following predicates occure : *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*

Goal: *on(A,B), on(B,C)*



Operators of block world problem

Pickup(*x*): ***SP*** → ***SP*** (*this* : ***SP***)

IF *ontable(x),clear(x),handempty* ∈ *this*

THEN *this* := *this* –{*ontable(x),clear(x),handempty* }∪{*holding(x)*}

Putdown(*x*): ***SP*** → ***SP*** (*this* : ***SP***)

IF *holding(x)*∈ *this*

THEN *this* := *this* –{*holding(x)*}∪{*ontable(x),clear(x),handempty*}

Stack(*x,y*): ***SP*** → ***SP*** (*this* : ***SP***)

IF *holding(x), clear(y)*∈ *this*

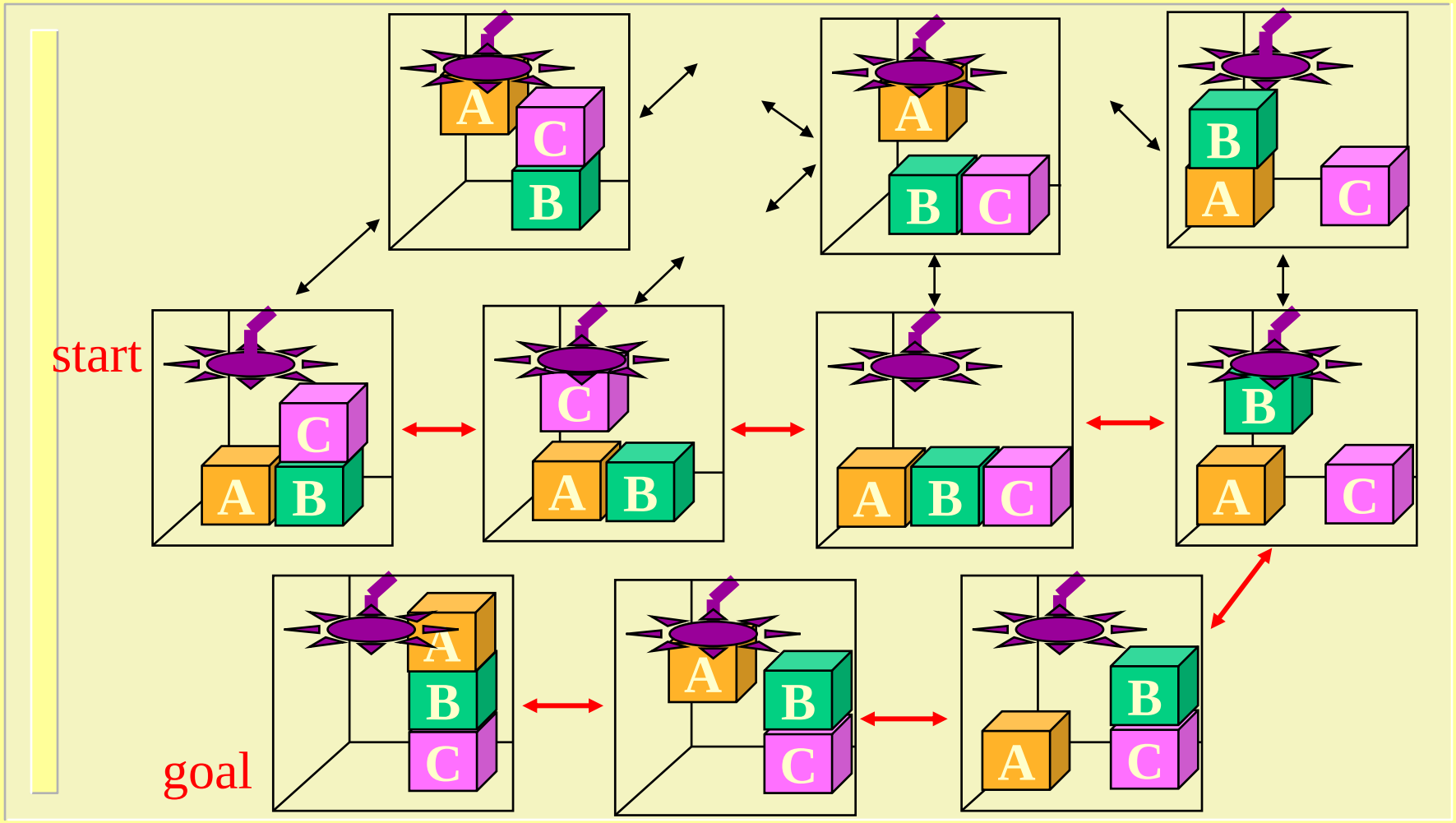
THEN *this* := *this* –{*holding(x), clear(y)*}∪{*on(x,y),clear(x),handempty*}

Unstack(*x,y*): ***SP*** → ***SP*** (*this* : ***SP***)

IF *on(x,y),clear(x),handempty* ∈ *this*

THEN *this* := *this* –{*on(x,y),clear(x),handempty*}∪{*holding(x), clear(y)*}

State graph



- ***Jug's problem***: Given a 5-liter jug filled with wine and empty 3-liter and 2-liter jugs. Let's obtain precisely 1 liter wine in the 2-liter jug.
- ***Missioner - cannibal 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 and in the boat.
- ***Satisfiability problem***: There is given a Boolean statement in CNF with *n* variables. Find a vector of truth assignments for all *n* variables so that the formula be true. E.g.: $F(x_1, \ldots, x_5) = (x_1 \vee \neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_5)$ possible solution: x_1=true, x_2=any, x_3=any, x_4=true, x_5=true
- ***Travelling salesman problem***: Covering the shortest distance, the traveling salesman must visit every city in his territory exactly once and then return home. (the *n* cities and all distances between them are known)

Jug's problem

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

State-space: $\mathbf{SP} = map(\mathbb{N} : \mathbb{N})$

invariant: this.Keys = { 5, 3, 2 } (this[5], this[3], this[2])

$\Sigma_{i \in [5,3,2]}\ this[i] = 5$

$\forall i \in [5,3,2]: this[i] \leq i$

Operator: $\mathbf{Fill}(i,j): \mathbf{SP} \rightarrow \mathbf{SP}$ $(this : \mathbf{SP})$

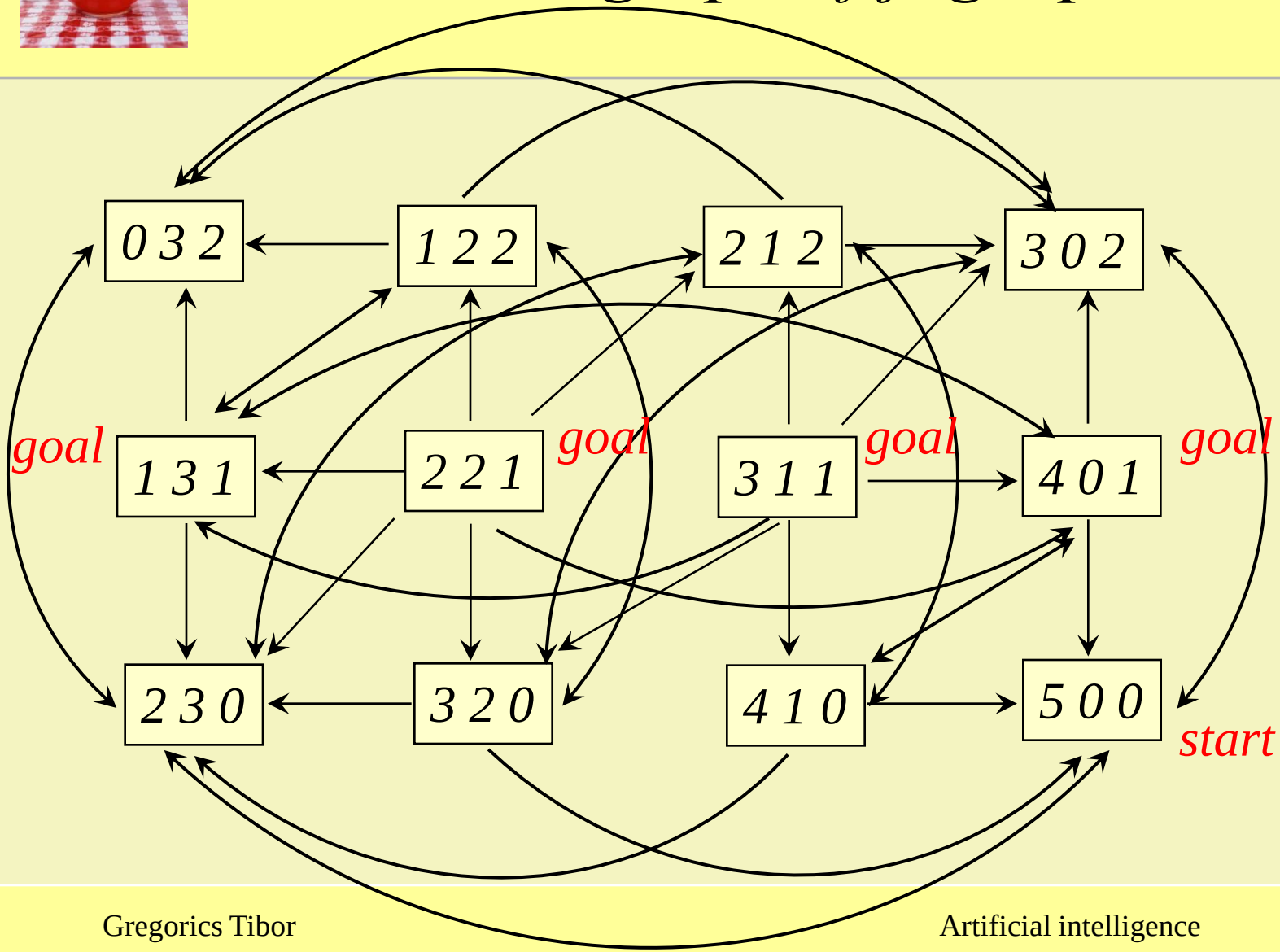
IF $i,j \in [5,3,2] \wedge i \neq j \wedge min(this[i], j - this[j]) > 0$

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

Initial: [5, 0, 0] (this[5]=5, this[3]=0, this[2]=0)

Goal: [x, y, 1] (this[2]=1)

State graph of jug's problem



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.

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

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

Initial state: $(n,n,true)$ *Goal state*: $(0,0,false)$

Operators: ***There***(x,y): ***SP*** $\rightarrow$ ***SP*** ***Back***(x,y): ***SP*** $\rightarrow$ ***SP*** (*this*: ***SP***)

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 $this.b := true$; $this.m := this.m+x$; $this.c := this.c+y$

SAT – satisfiability problem

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

E.g.: $F(x_1, \ldots, x_5) = (x_1 \vee \neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_5)$

possible solution: $x_1 = true, x_2 = any, x_3 = any, x_4 = true, x_5 = true$

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

Operator: ***Change***(i): $\boldsymbol{SP} \rightarrow \boldsymbol{SP}$ $(this : \boldsymbol{SP})$

$this[i] := \neg this[i]$

Initial state: *arbitrary*

Goal state: *F(this) is true*

SAT – satisfiability problem 2.

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

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

invariáns: $0 \le i \le n, \forall j \in \{1 \ldots i\}: t[j] \neq \emptyset$

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

Operator:

***True*: SP → SP**

i := i+1: *this.t*[*i*] := *true*

update(this.count)

***False*: SP → SP** (*this* : **SP**)

i := i+1: *this.t*[*i*] := *false*

update(this.count)

Initial state: *([∅, … , ∅] , 0, 0)*

Goal state: *this.count=C*

Travelling salesman problem

Covering the shortest distance, the traveling salesman must visit every city in his territory exactly once and then return home. (the n cities and all distances between them are known)

State-space: ***SP*** = {cities}*

Operator: ***Goto****(city)*: ***SP*** $\rightarrow$ ***SP***

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

THEN *this.append(city)*

Initial state: *empty*

Goal state: *length of this is n*