Heuristic Search Planner Techniques

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Heuristic search planning techniques

Source

- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. Chapter 10. (3rd Edition). Ed. Pearsons
- Dana Nau's slides for Automated Planning. Licensed under License https://creativecommons.org/licenses/by-nc-sa/2.0/
- Jorg Hoffmann. FF: The Fast-Forward Planning System. AAAI'01

Outline

- **Introduction**
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- FF

Introduction

- Heuristic Search Planners (HSP) transform planning problems into heuristic search problems extracting heuristics functions, rather than enter them by hand
- Problems
  - Number of explored nodes is very high
  - Heuristic calculation in each step
- Examples of heuristics:
  - HSP: additive (h^{add})
  - FF: delete-relaxation

Outline

- Introduction
- **Heuristics**
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- FF

Heuristics

- Returns an estimate h(s) of the minimum cost h*(s) of getting from the state s to a goal state
- If the algorithm always finds optimal solutions, then the heuristic function will be admissible
- Best way of producing them: RELAXATION
  - Weakening some of the constraints
- They can be domain-specific or domain-independent
  - Max-Cost and Additive Cost
  - Delete-Relaxation
  - Landmarks

Outline

- Introduction
- Heuristics
  - **Max-cost and additive cost**
  - Delete-Relaxation
  - Landmarks
- FF

Max-cost heuristic

- The max-cost of a set of literals $g = \{g_1,\ldots,g_k\}$ is defined recursively as the largest max-cost of each g_i individually
- Computation of h^{max} can be visualized as an And/Or search going backward from g
- Admissible
- Planning: a goal (i.e., a set of literals such as g or the preconditions of an action) can be reached by achieving just one of the goal's literals, namely, the one that is the most expensive to achieve

Additive heuristic

- H^{add} is similar to h^{max} but adds the costs of each set of literals rather than taking their maximum
- Not admissible (better in practice)

Heuristic calculation: example (I)

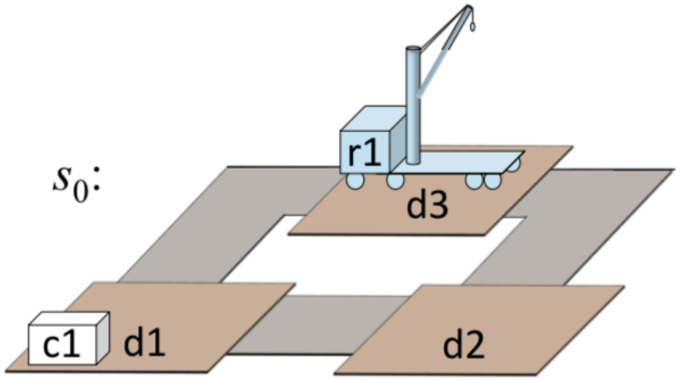
$$B = Robots \cup Docks \cup Containers \cup \{\mathsf{nil}\};$$
$$Robots = \{\mathsf{r}_1\};$$
$$Docks = \{\mathsf{d}_1, \mathsf{d}_2, \mathsf{d}_3\};$$
$$Containers = \{\mathsf{c}_1\}.$$

$\mathsf{load}(r, c, l)$
pre: $\mathsf{cargo}(r) = \mathsf{nil}$, $\mathsf{loc}(c) = l$, $\mathsf{loc}(r) = l$
eff: $\mathsf{cargo}(r) \leftarrow c$, $\mathsf{loc}(c) \leftarrow r$
cost: 1

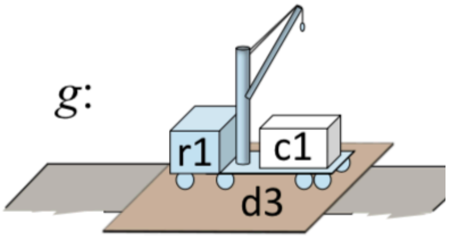
$\mathsf{unload}(r, c, l)$
pre: $\mathsf{cargo}(r) = c$, $\mathsf{loc}(r) = l$
eff: $\mathsf{cargo}(r) \leftarrow \mathsf{nil}$, $\mathsf{loc}(c) \leftarrow l$
cost: 1

$\mathsf{move}(r, d, e)$
pre: $\mathsf{loc}(r) = d$
eff: $\mathsf{loc}(r) \leftarrow e$
cost: 1

s_0:



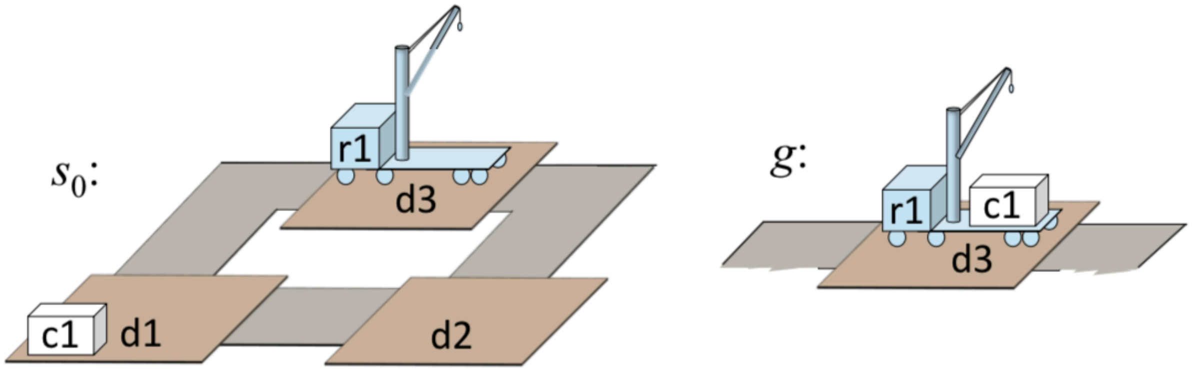
g:



$$s_0 = \{\mathsf{loc(r1)} = \mathsf{d3}, \mathsf{cargo(r1)} = \mathsf{nil}, \mathsf{loc(c1)} = \mathsf{d1}\};$$
$$g = \{\mathsf{loc(r1)} = \mathsf{d3}, \mathsf{loc(c1)} = \mathsf{r1}\}.$$

Heuristic calculation: example (II)

- Actions?
  - a1 = move (r1, d3, d1)
  - a2 = move(r1, d3, d2)

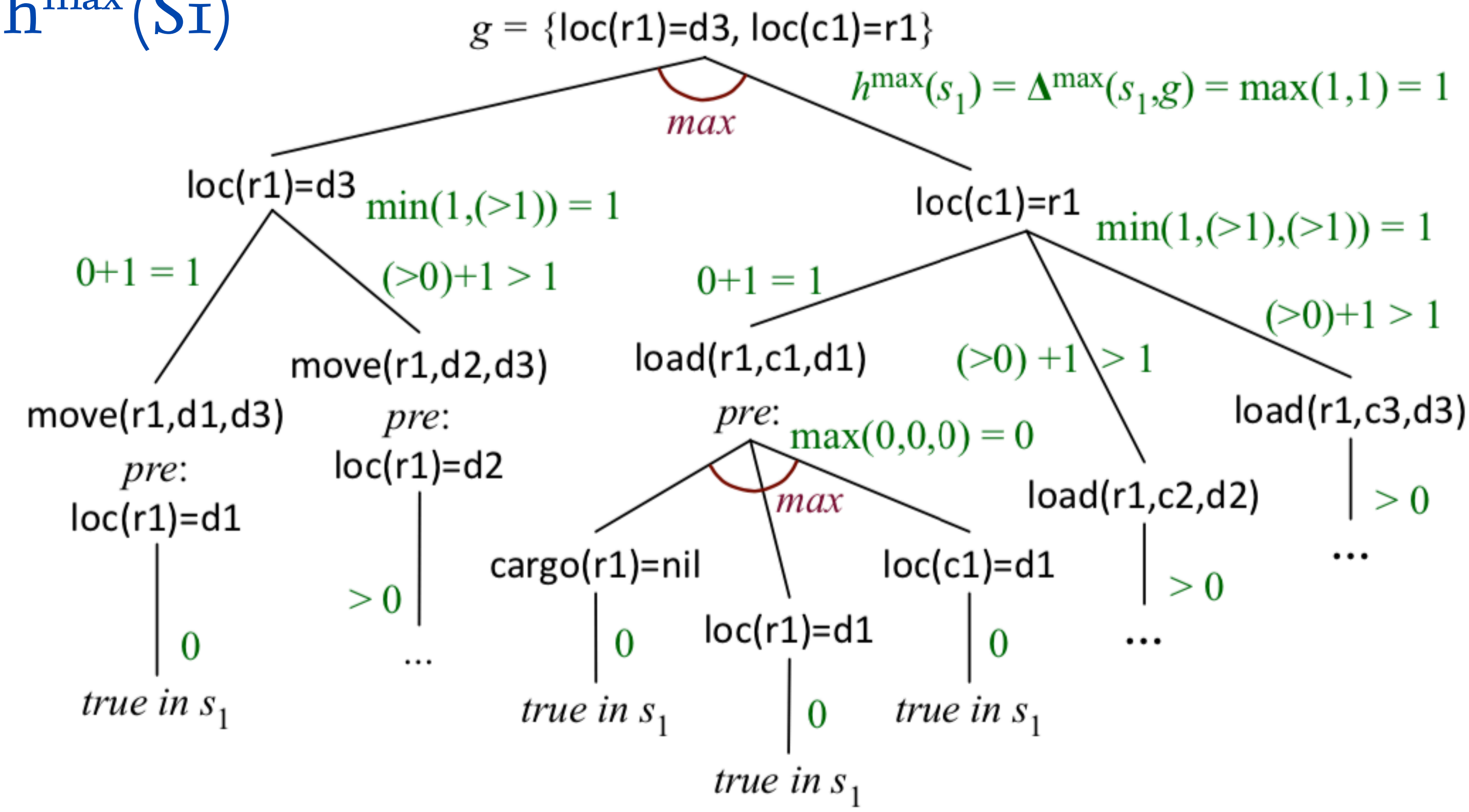


s1 = γ(s0, a1) = {loc(r1) = d1, cargo(r1) = nil, loc(c1) = d1}

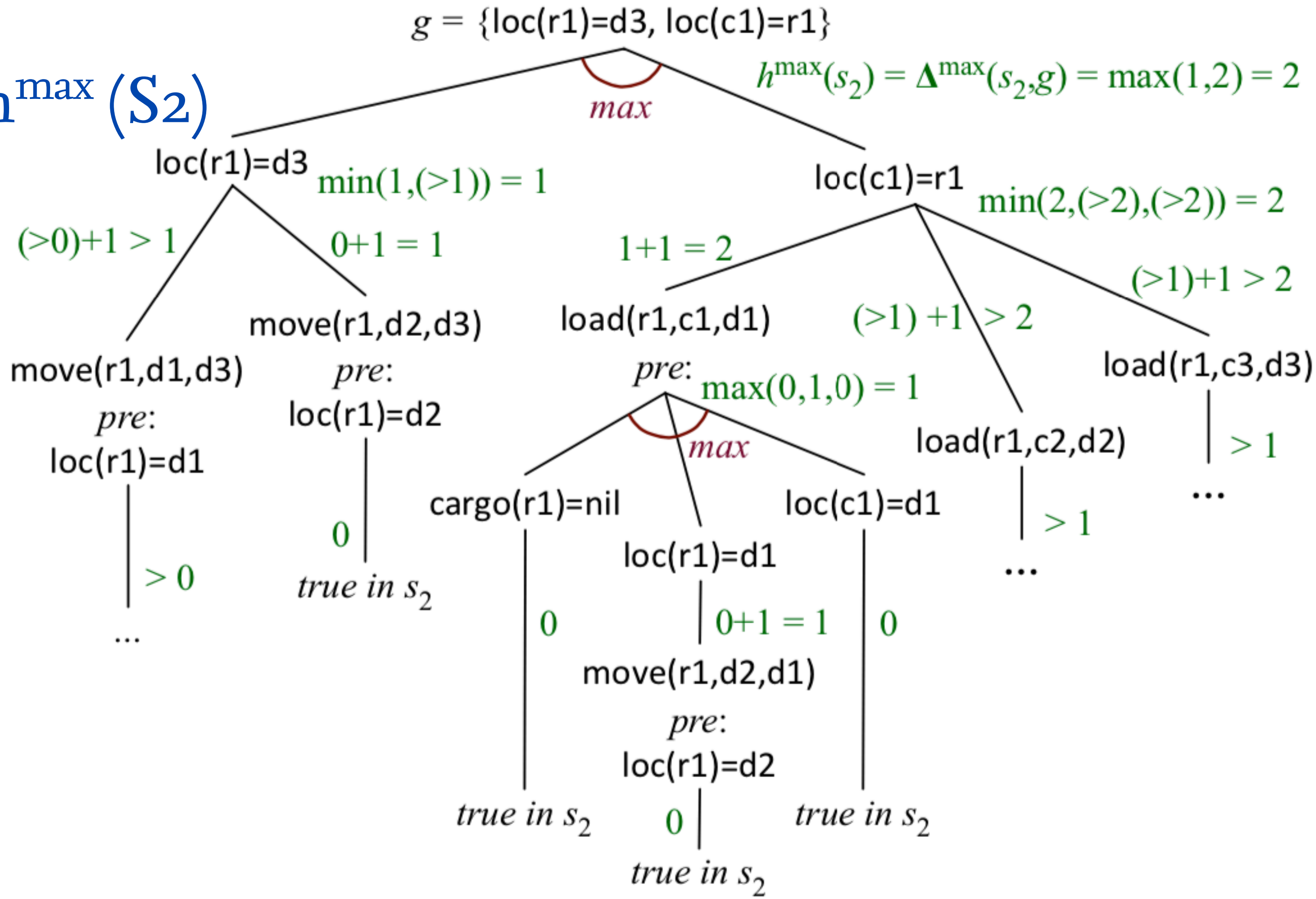
s2 = γ(s0, a2) = {loc(r1) = d2, cargo(r1) = nil, loc(c1) = d1}

Can you calculate the value of h^{add} and h^{max}?

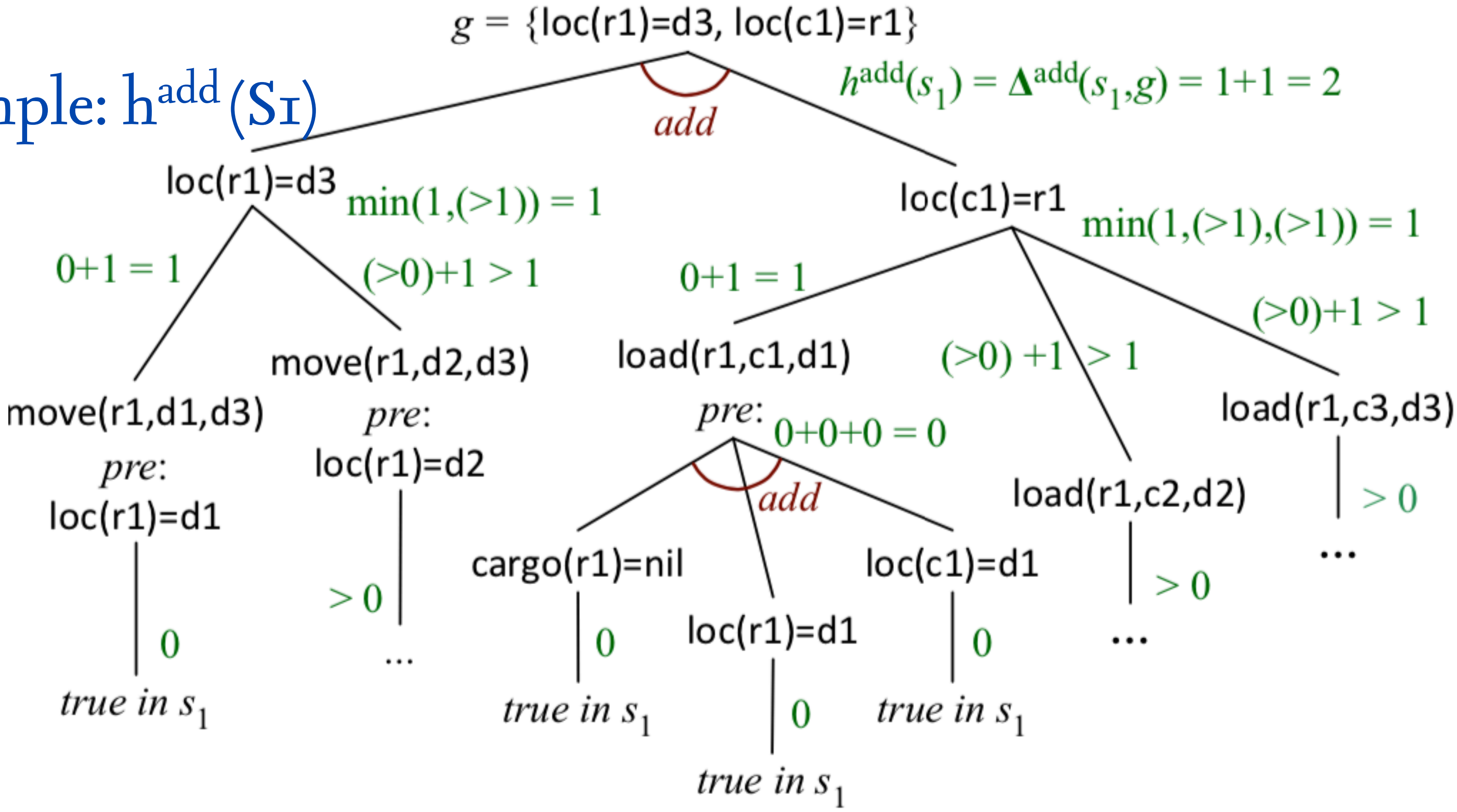
Example: $h^{max}(s_1)$



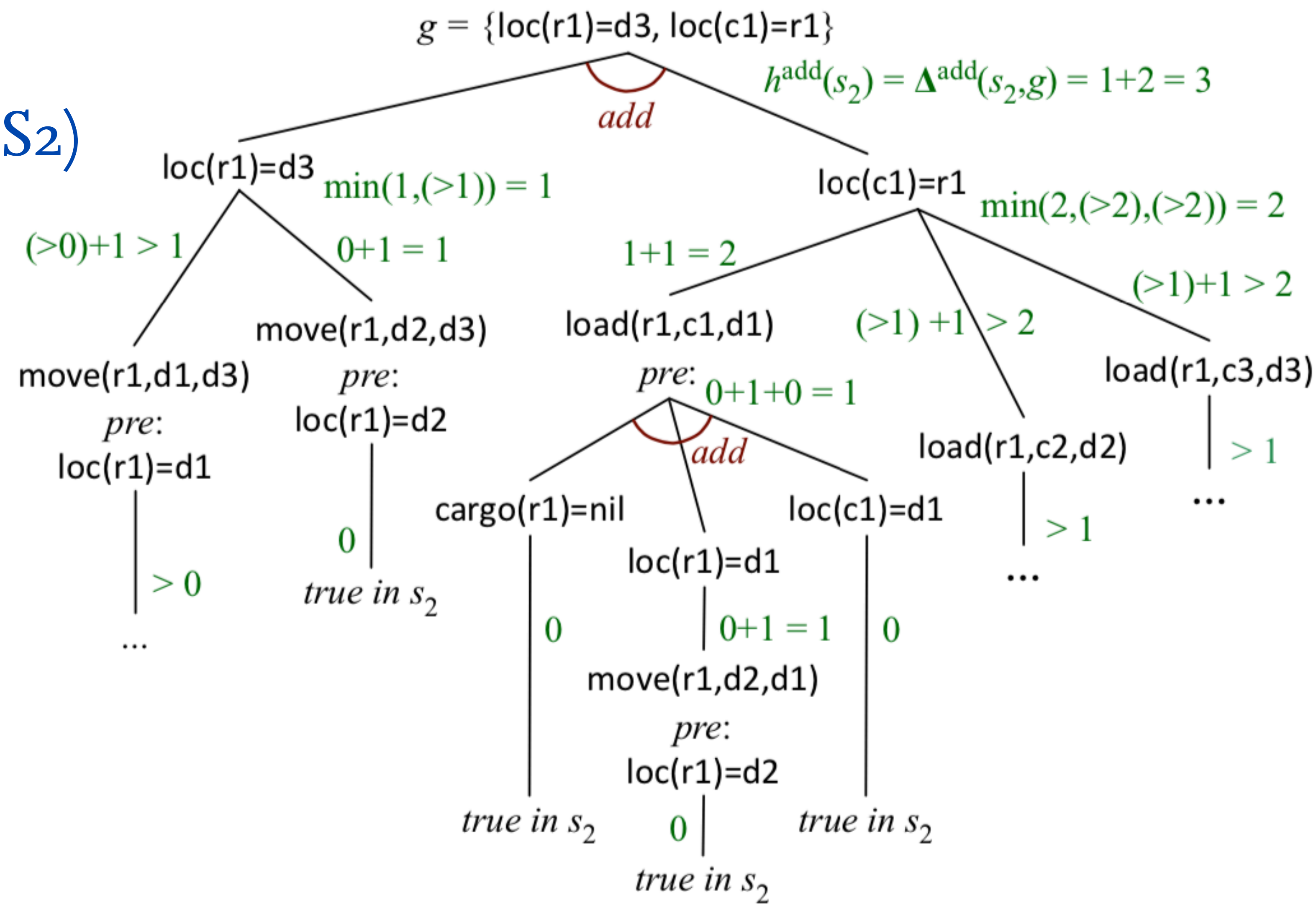
Example: h^{max} (S_2)



Example: $h^{add}(s_1)$



Example: h^{add} (S_2)



Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - **Delete-Relaxation**
  - Landmarks
- FF

Delete-relaxation

- h^+: applies new actions that never removes old atoms from a state (simply adds new ones)
- Admissible
- Use in combination with a relaxed GP

Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - **Landmarks**
- FF

Landmarks

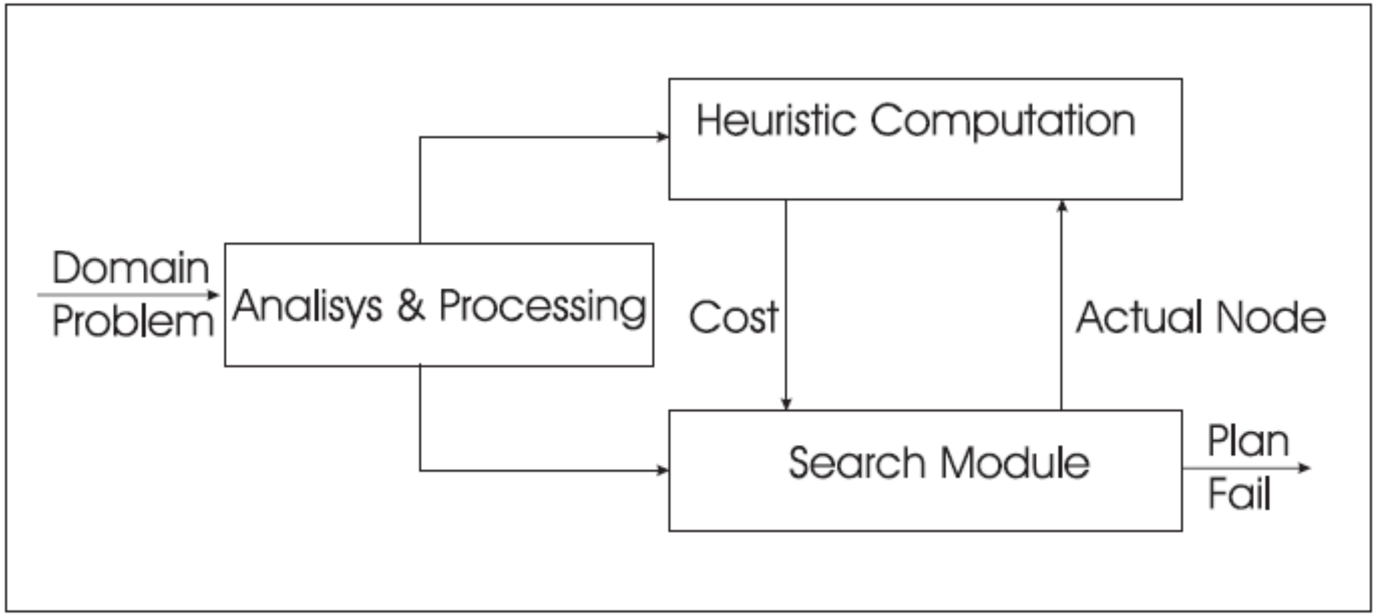
- To compute $h^{sl}(s1)$, we count the number of landmarks between s1 and g
- If we start in s1, then every solution plan must include a state in which cargo(r1) = c1
  - This is the only landmark for s1
  - $h^{sl}(s1) = 1$
- If we start in state s2, then the landmark computation will find two landmarks:
  - cargo(s1) = c1
  - loc(r1) = d1
  - $h^{sl}(s2) = 2$

Outline

- Introduction
- Heuristics
  - Max-cost and additive cost
  - Delete-Relaxation
  - Landmarks
- **FF: Fast-Forward**

FF (I)

- General architecture

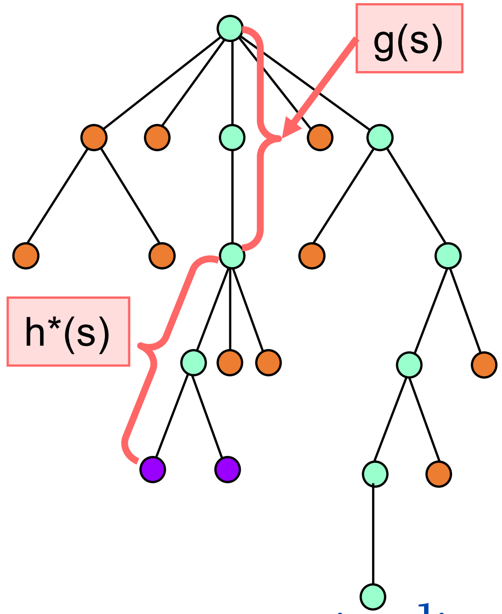


FF (II)

- **The Analysis & Processing module.** Analyze and process all the information from the domain and the initial state (table or vector with all the possible operators instantiated)
- **The Heuristic Computation module**. Computes the cost of applying a determined node in the search process
- **The Search module**. Depends on the heuristic computation, uses a search algorithm or a combination of them

FF search: A*

- For every state s, let
  - $g(s)$ = cost of the path from s_0 to s
  - $h^*(s)$ = least cost of all paths from s to goal nodes
  - $f^*(s) = g(s) + h^*(s)$ = least cost of all paths from s_0 to goal nodes that go through s
- Suppose $h(s)$ is an estimate of $h^*(s)$
  - Let $f(s) = g(s) + h(s)$
  - h is *admissible* if for every state s, $0 \leq h(s) \leq h^*(s)$
  - If h is admissible then f is a lower bound on f^* and A^* guarantees optimality
- In combination with Hill Climbing



FF: h

Move (x,y)
- **Pre:** in(x), opened
- **Add:** in(y)
- **Del:** in(x)

Open
- **Pre:** closed
- **Add:** opened
- **Del:** closed

Close
- **Pre:** opened
- **Add:** closed
- **Del:** opened

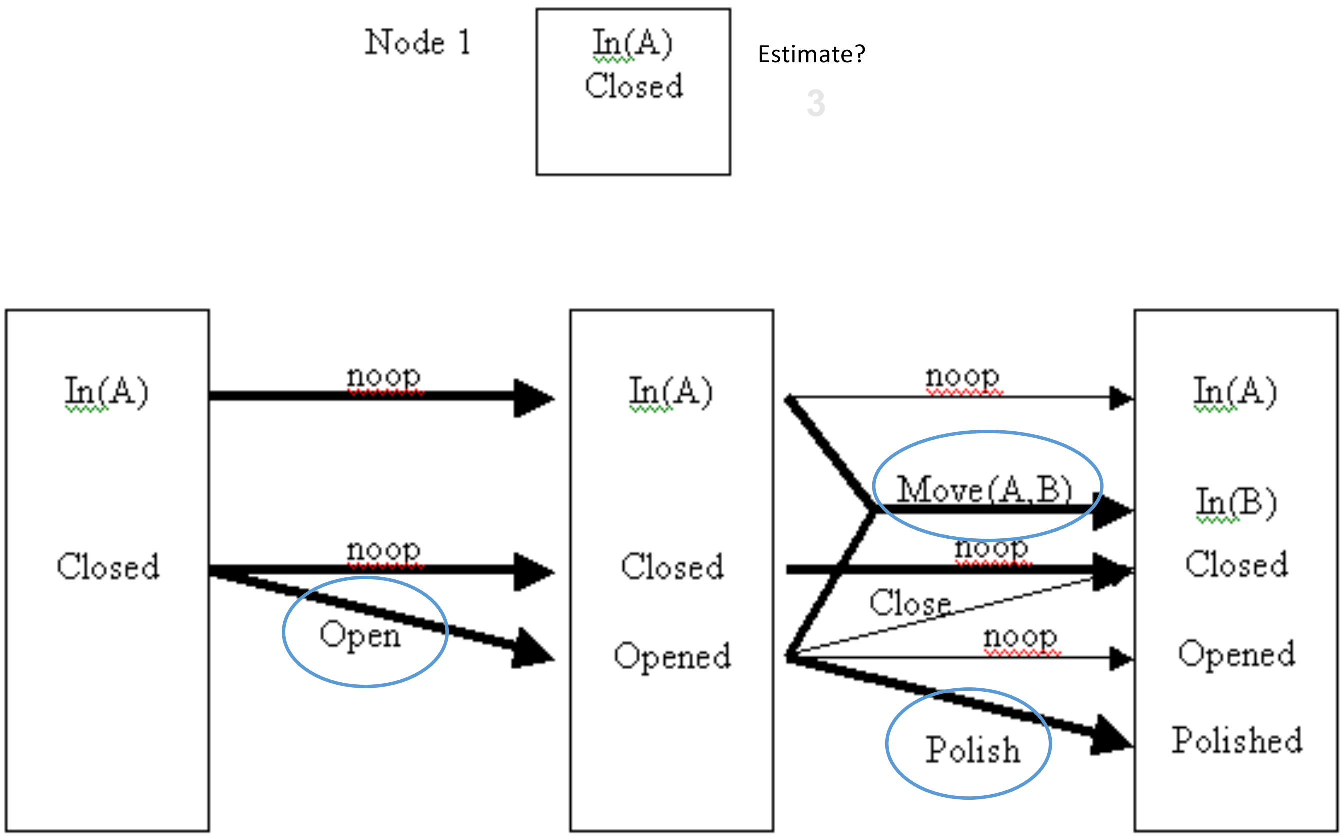
Polish
- **Pre:** opened
- **Add:** polished
- **Del:**

In(A)
Closed

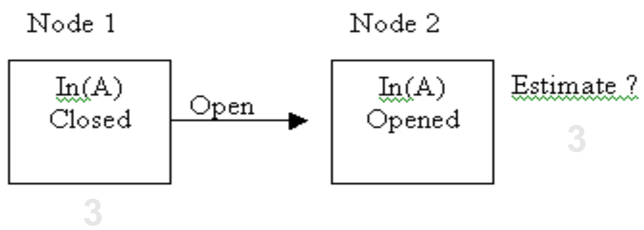
$\longrightarrow$

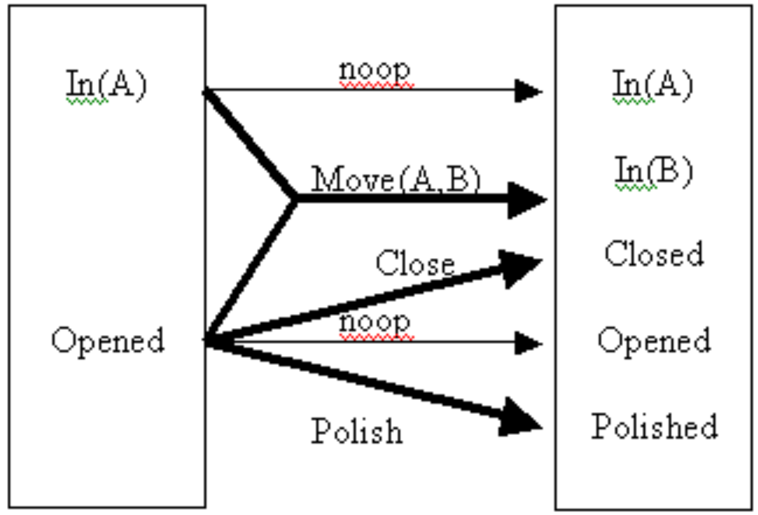
In(B)
Closed
Polished

FF: h

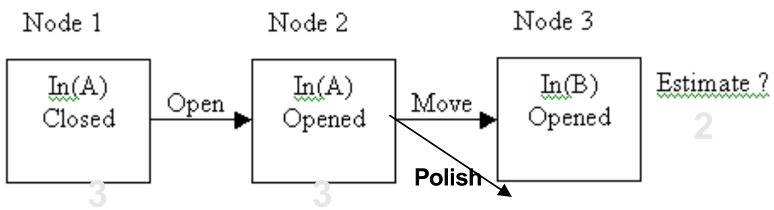

Node 1
In(A)
Closed
Estimate?
3
In(A)
Closed
noop
noop
Open
In(A)
Closed
Opened
noop
Move(A,B)
noop
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

FF: h

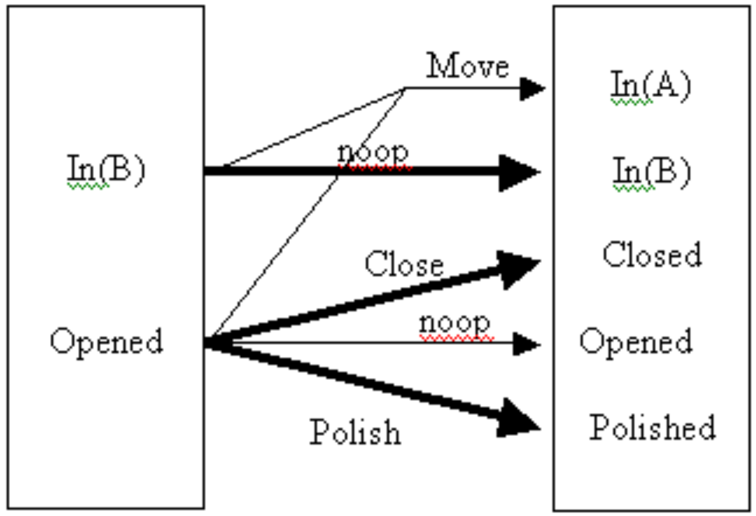

Node 1
Node 2
In(A)
Closed
Open
In(A)
Opened
Estimate ?
3
3


In(A)
Opened
noop
Move(A,B)
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

FF: h

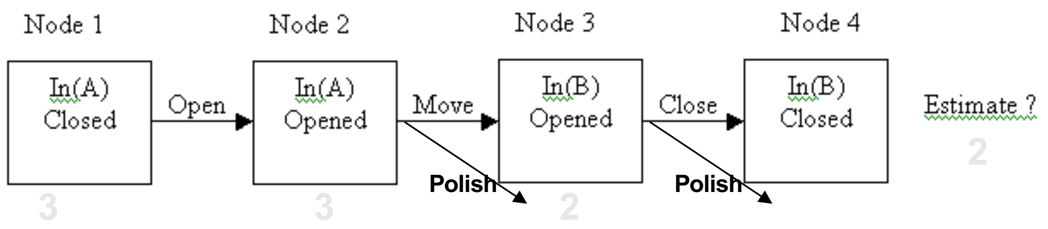


Node 1
Node 2
Node 3
In(A)
Closed
Open
In(A)
Opened
Move
Polish
In(B)
Opened
Estimate ?
2
3
3

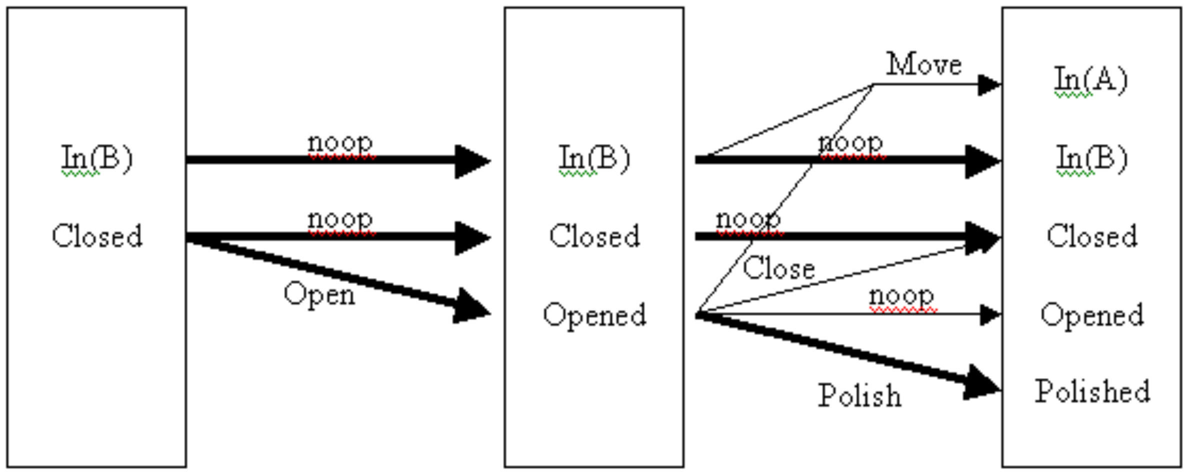


In(B)
Opened
Move
noop
Close
noop
Polish
In(A)
In(B)
Closed
Opened
Polished

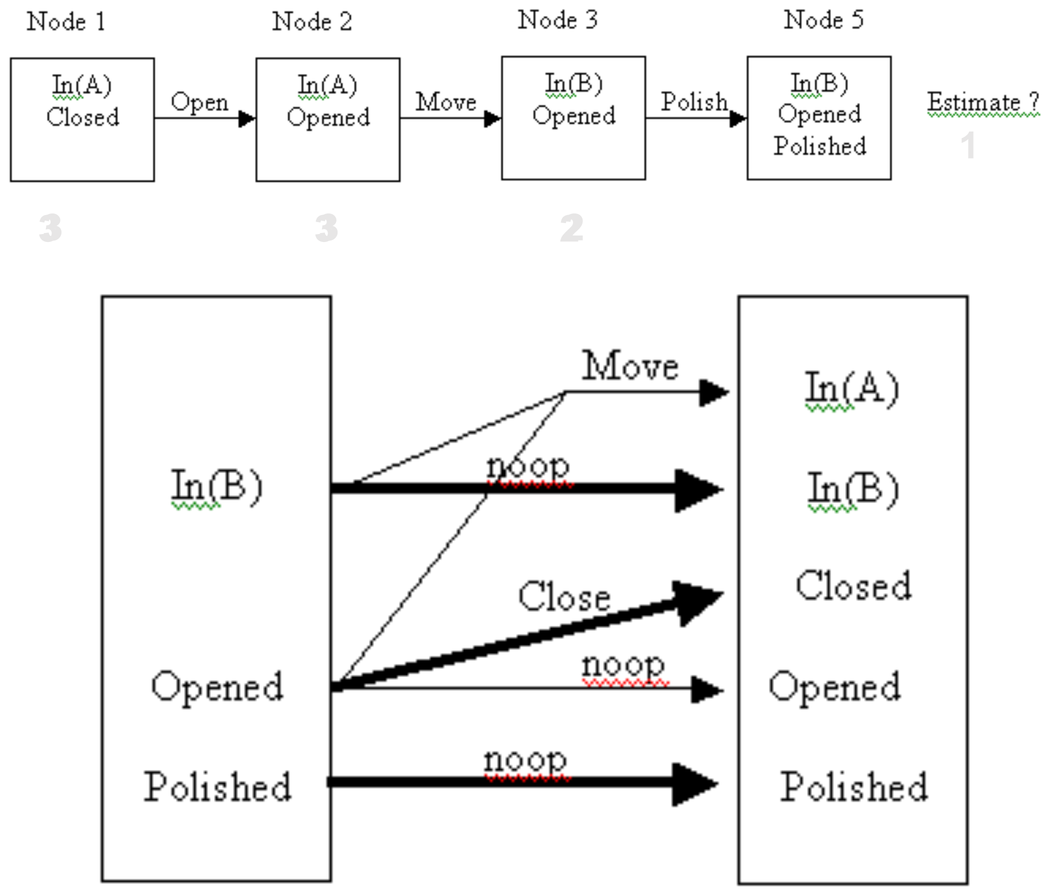
FF: h



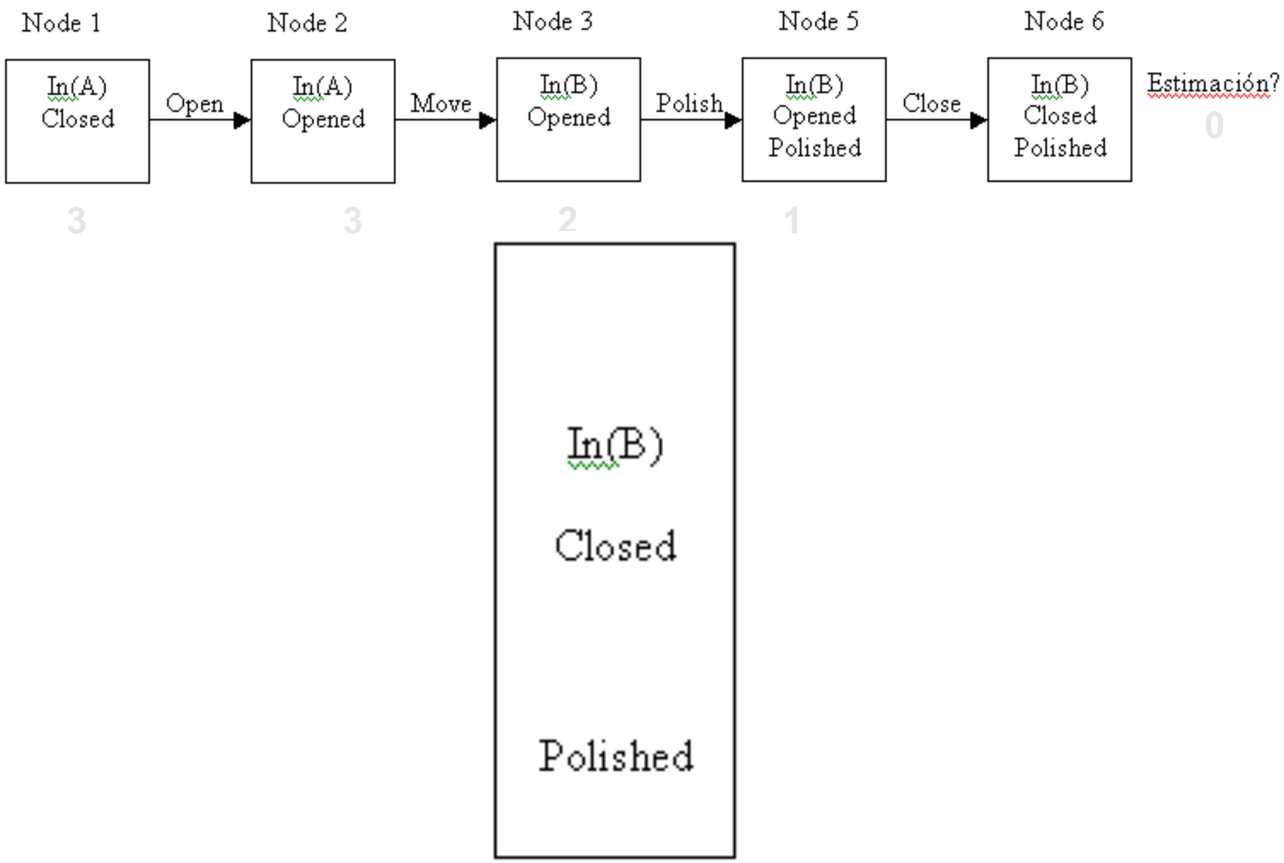
It does not improve the previous result, try Polish



FF: h


Node 1
In(A)
Closed
Open
Node 2
In(A)
Opened
Move
Node 3
In(B)
Opened
Polish
Node 5
In(B)
Opened
Polished
Estimate ?
1
3
3
2
In(B)
Opened
Polished
Move
noop
Close
noop
noop
In(A)
In(B)
Closed
Opened
Polished

FF: h



Node 1
In(A)
Closed
Open
Node 2
In(A)
Opened
Move
Node 3
In(B)
Opened
Polish
Node 5
In(B)
Opened
Polished
Close
Node 6
In(B)
Closed
Polished
Estimación?
0
3
3
2
1
In(B)
Closed
Polished