Artificial Intelligence (CS6380)

Local search

Problems considered till now..

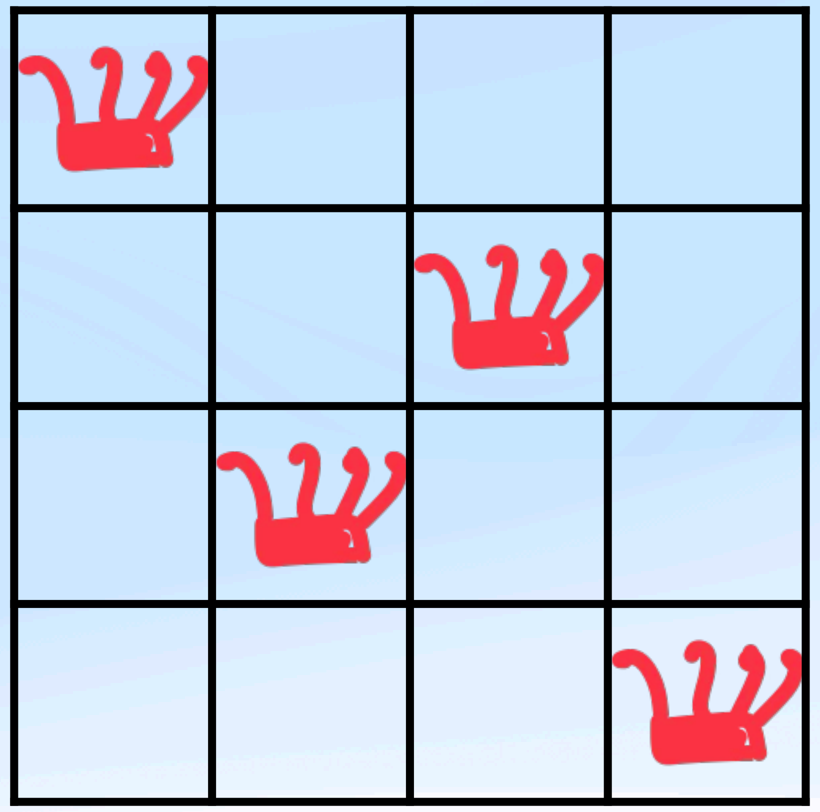
- **8 puzzle**
  - Move blank up, move blank right…
- **Finding route from CSE Dept. to IIT Main Gate**
  - Take right and go to biotech, take right and go to GC, take left and go to main gate
- **Man, Goat, Cabbage and Wolf**
  - Man takes goat to other side, comes back alone, takes cabbage to other side, ..
- **Knuth's 4 conjecture**
  - Floor(sqrt(…(4)…))

In each case solution is a sequence of actions.

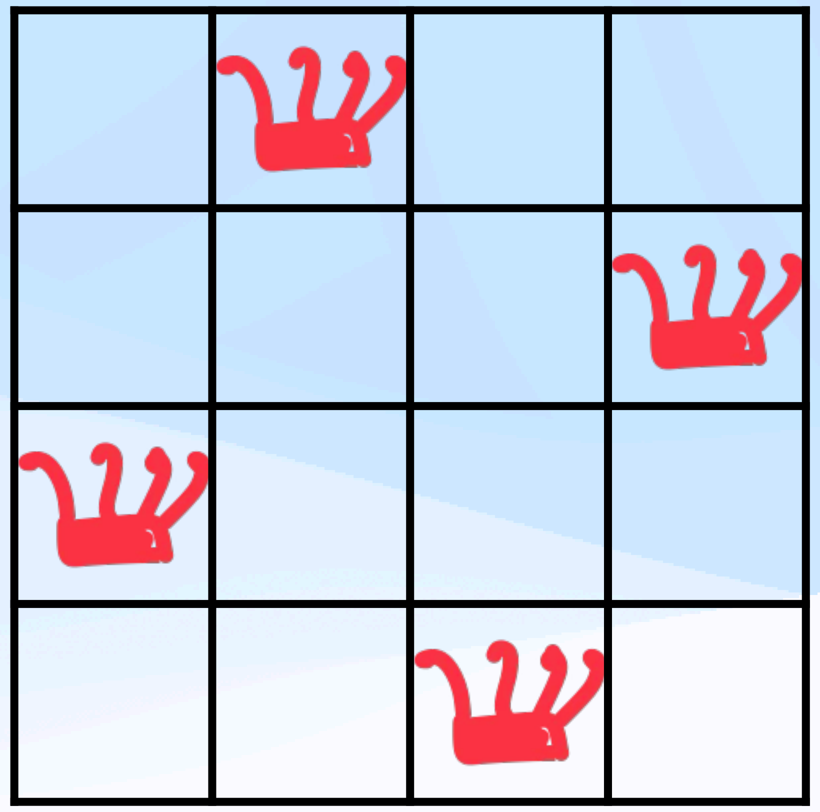
N Queens problem

N x N empty chess board

- Place N queens such that no queen is under attack by any other queen



Multiple queens under attack



No queen is under attack

Possible approaches:

- Start one queen at a time, place the next queen in a non-attacking position
- Start with a placement of all N queens, check if it is valid, else **perturb**.

Boolean Satisfiability

n boolean variables, m clauses

- Each variable can be assigned 0 or 1
- Goal: find an assignment, if possible, that satisfies the formula

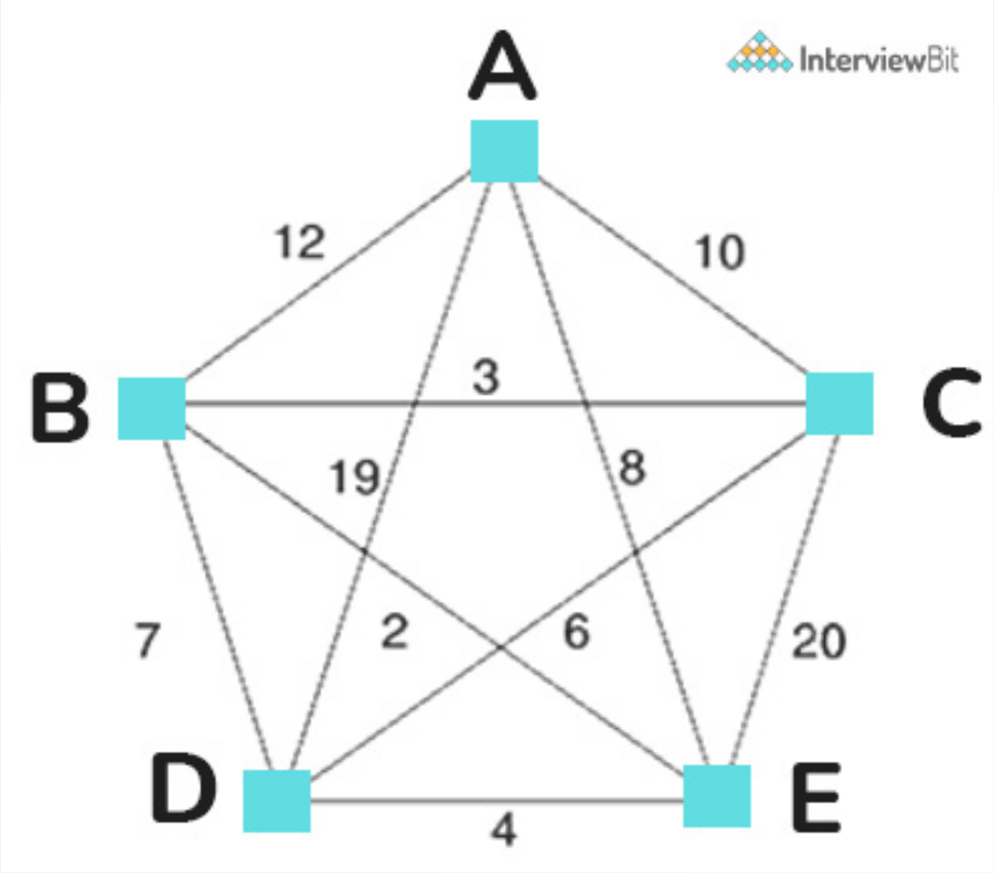
$$\phi_1 = (x_1 \vee x_2 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_3) \wedge x_5$$

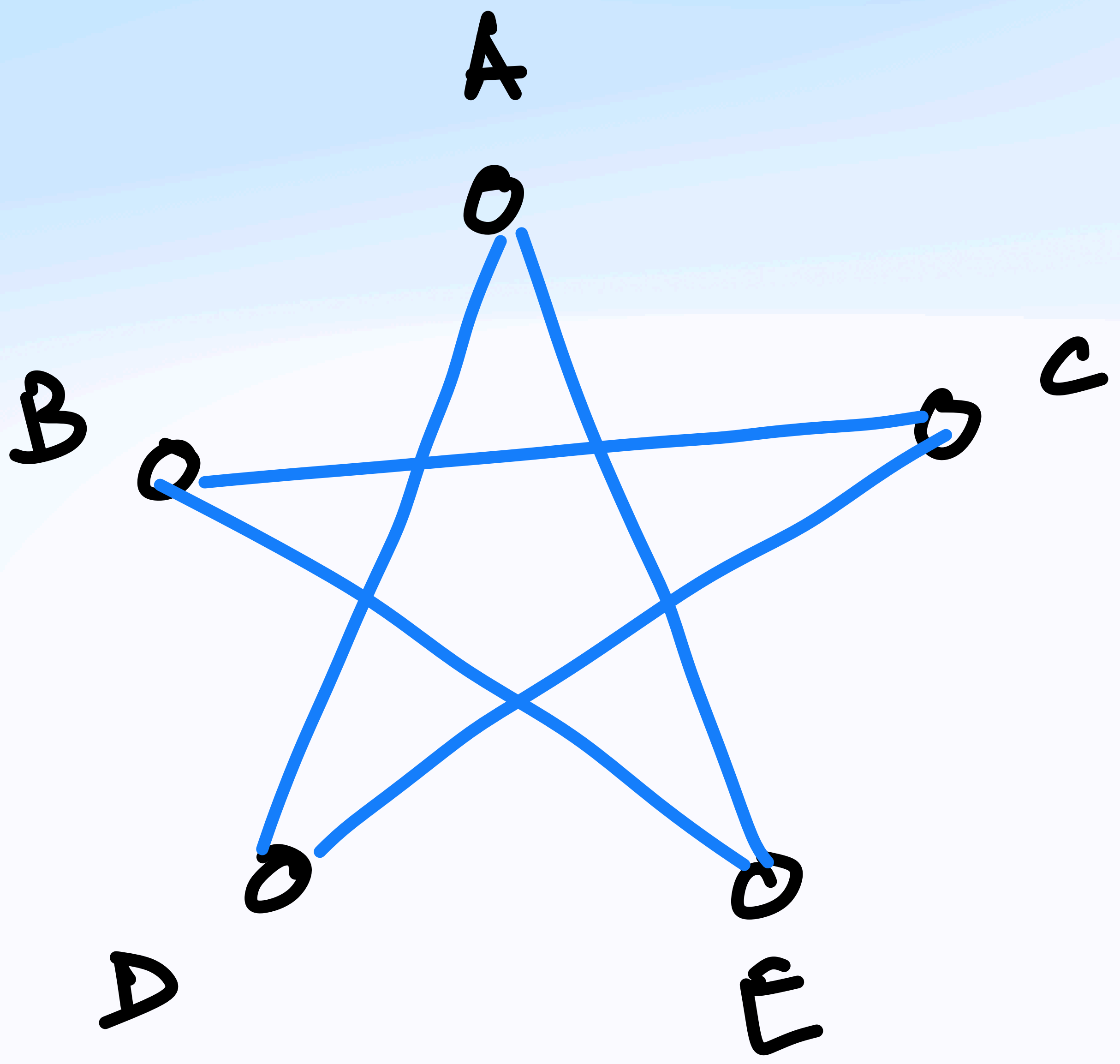
$$\phi_2 = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2)$$

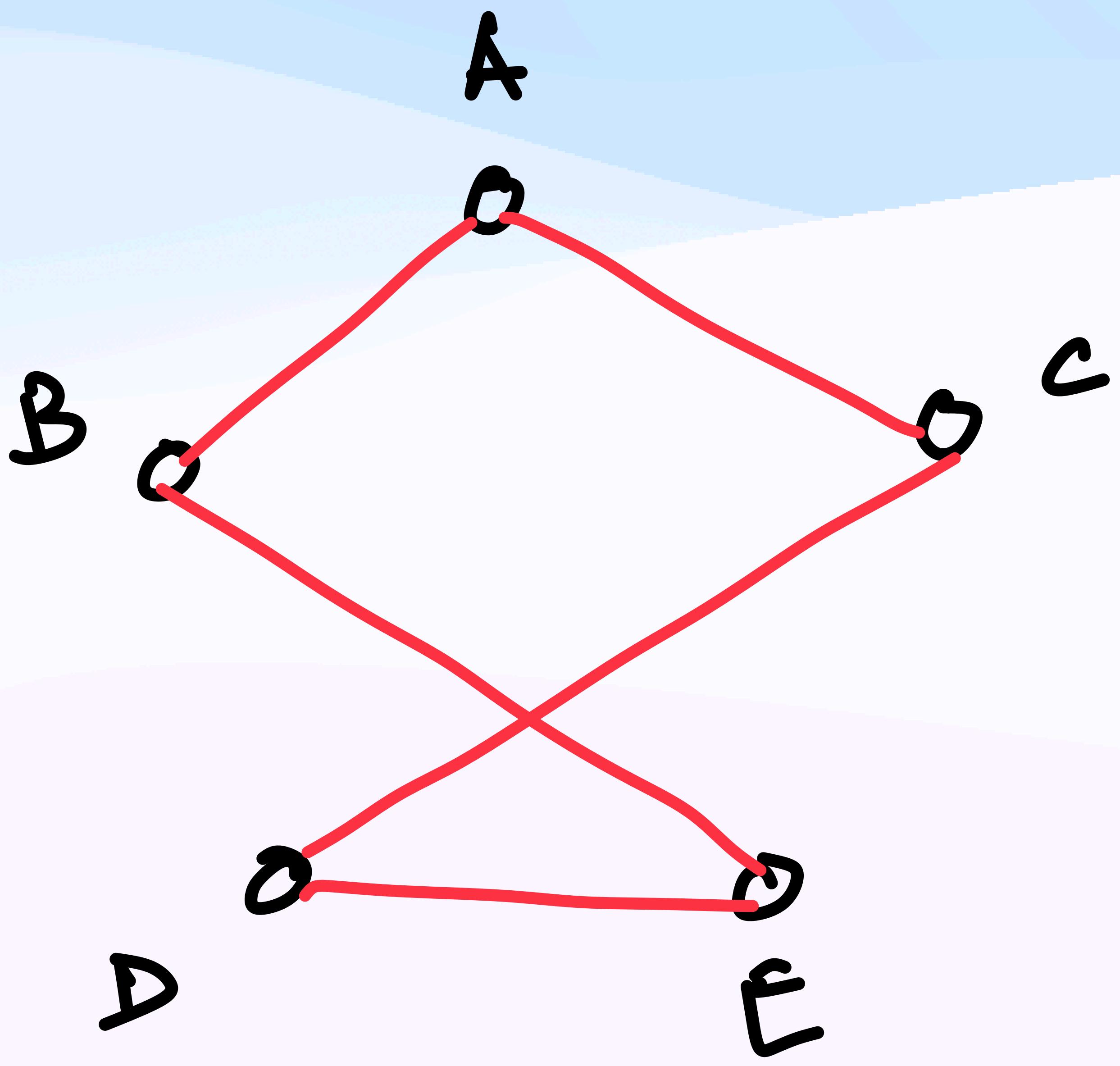
Travelling salesman

Complete graph on n vertices, non-negative edge cost

- Goal: find a tour of smallest cost







Local search

- Start with an initial state
- Operate by searching to neighbouring states
    - Does not keep track of reached states, hence is not systematic
    - May never explore portions of search space where solution resides
- Uses very less memory, often find reasonable solutions in infinite spaces
- Can also be used for optimisation problems

Hill climbing

Function Hill-Climbing (problem)

- Current = problem.initial state
- While true do
    - neighbour = highest value successor (current)
    - If value (neighbour) <= value (current) then return current
    - current = neighbour.
- End while

Where do we start?

What are the successors?

how do we evaluate a state?

Boolean Satisfiability

$$(b \lor \bar{c}) \land (c \lor \bar{d}) \land (\bar{b}) \land (\bar{a} \lor \bar{e}) \land (e \lor \bar{c}) \land (\bar{c} \lor \bar{d})$$

Take away: choice of start state matters.

Hill climbing : variants

Function Hill-Climbing (problem)

- Current = problem.initial state
- While true do
  - neighbour = highest value successor (current)
  - If value (neighbour) <= value (current) then return current
  - current = neighbour.
- End while

- **Local beam search** (keep k states rather than 1)
- **Stochastic hill climbing** (select one at random from the uphill moves)
- **Random restart hill climbing** (series of hill climbings from randomly generated starting states)
- **Simulated annealing** (allow downhill moves with some probability)

Variable neighbourhood hill climbing

Let **expand1, expand2, expand3**… be a sequence of denser and denser neighbourhood generation functions.

Main idea: use expand1 initially, get to a local optimum, if that is the goal state you are done,

Else use expand2 to find successors.

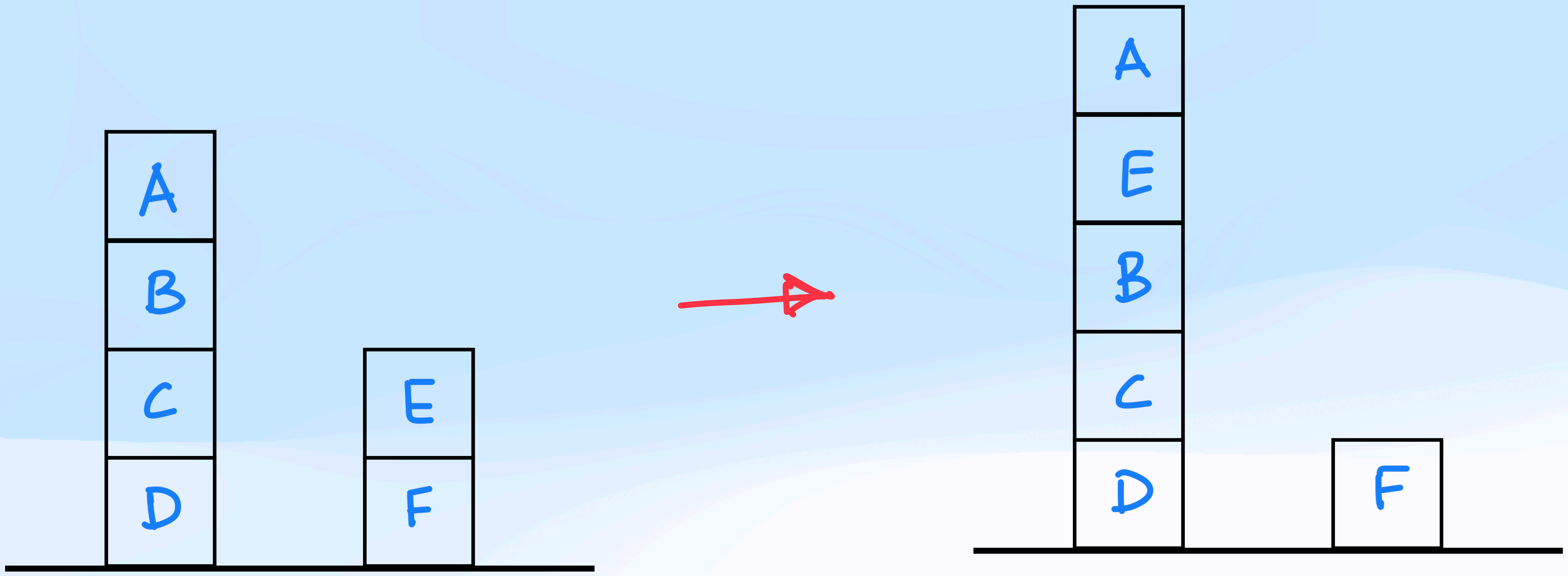
What is expand-k?

Function Hill-Climbing (start state, **expand-function f)**

- Current = start state
- While true do
  - neighbour = highest value successor (current) using **function f**
  - If value (neighbour) <= value (current) then return current
  - current = neighbour.
- End while

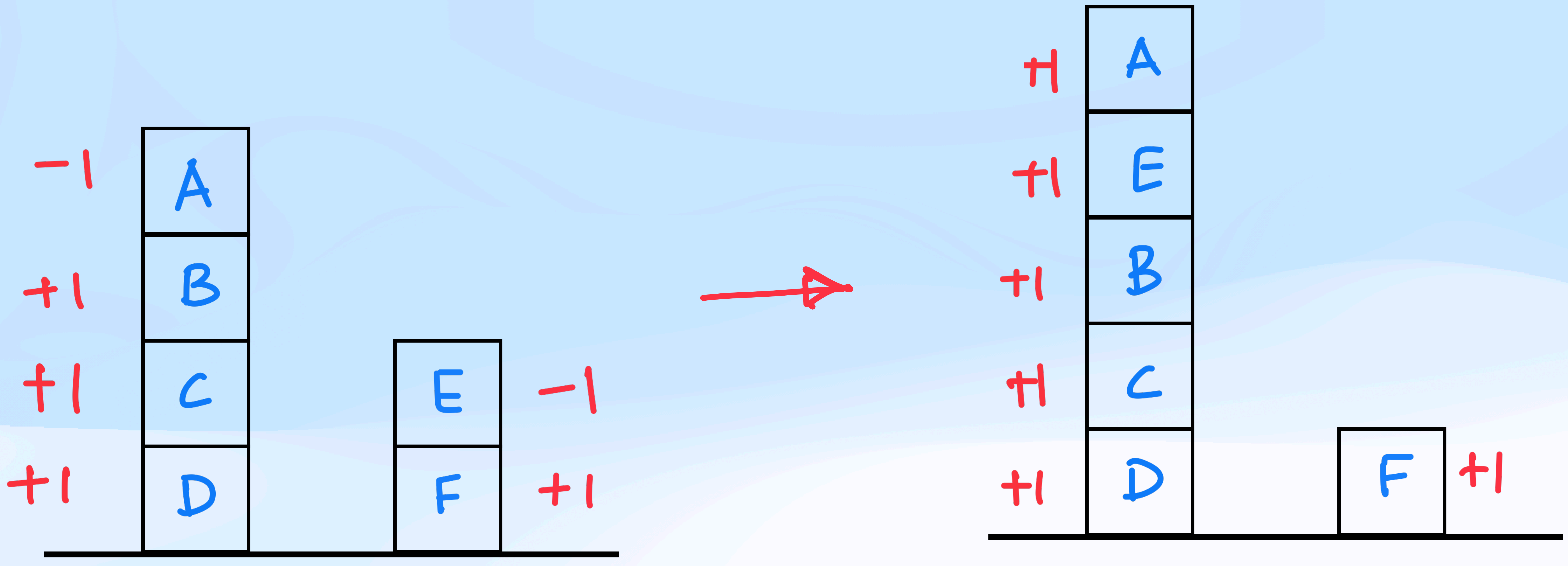
Take away: if the densest function spans the entire neighbourhood, the algorithm is complete, but resembles brute force. Key is to use denser function to get out of local maxima.

Blocks world domain



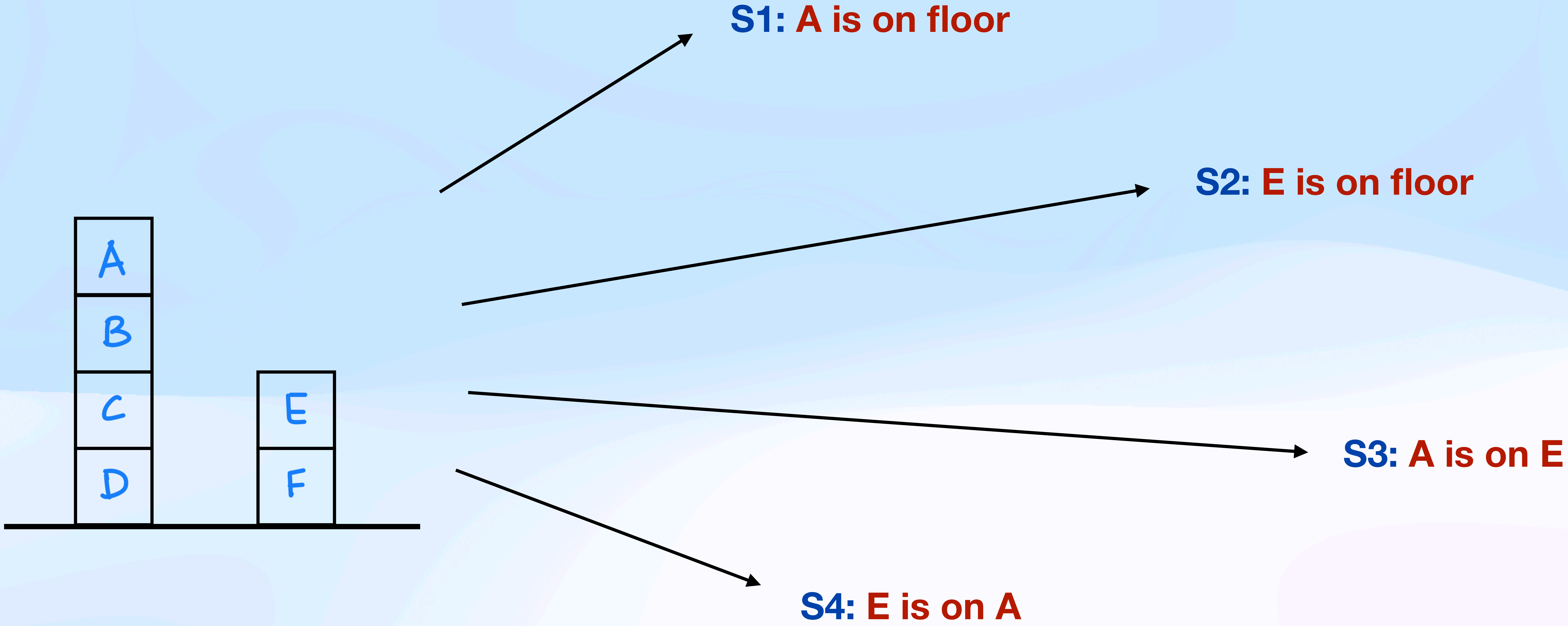
Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Blocks world domain



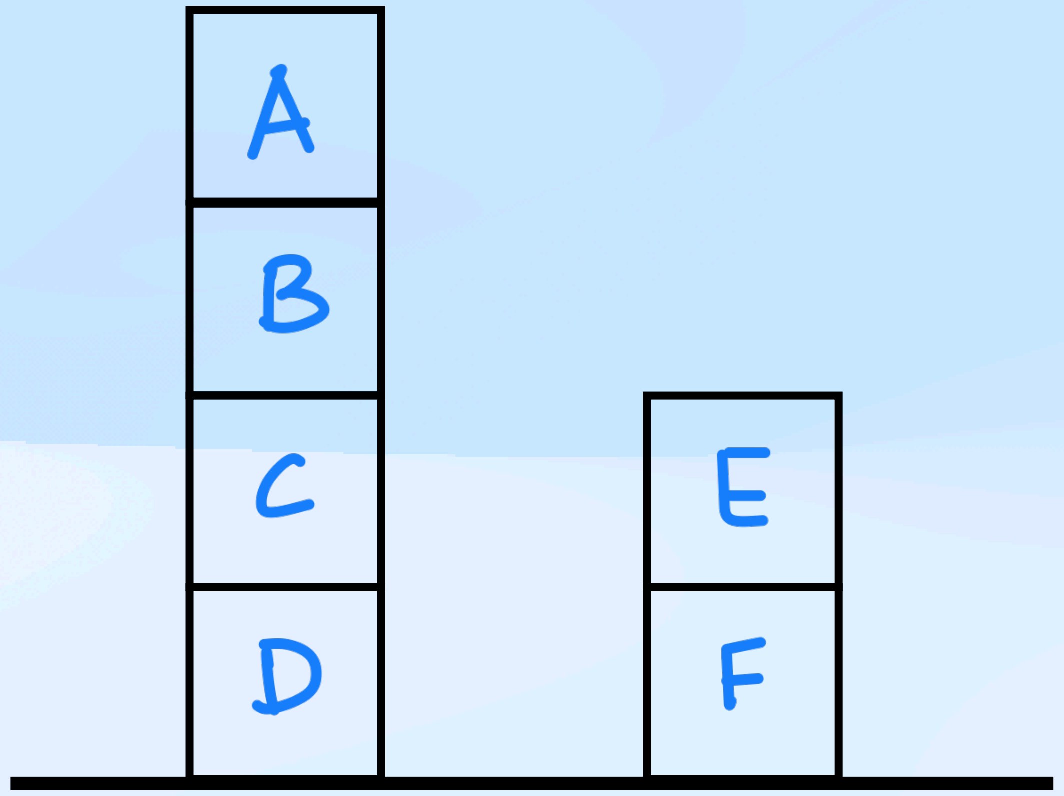
Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Blocks world domain



Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Blocks world domain



S1: A is on floor (2)

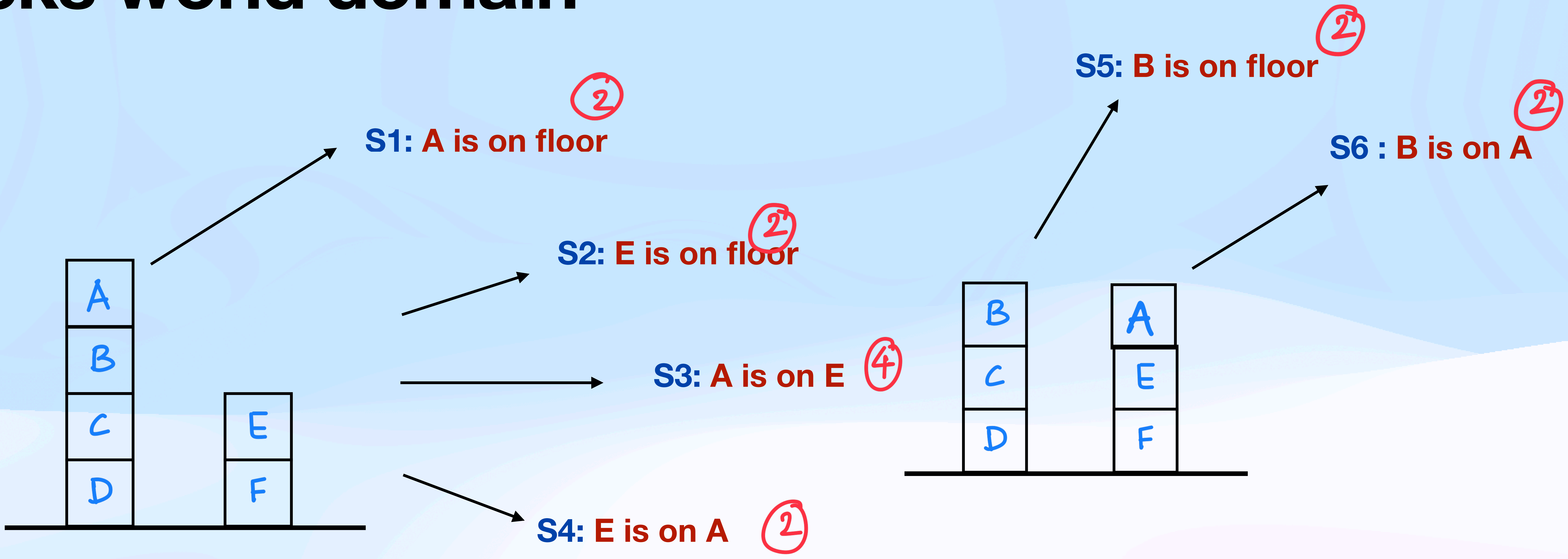
S2: E is on floor (2)

S3: A is on E (4)

S4: E is on A (2)

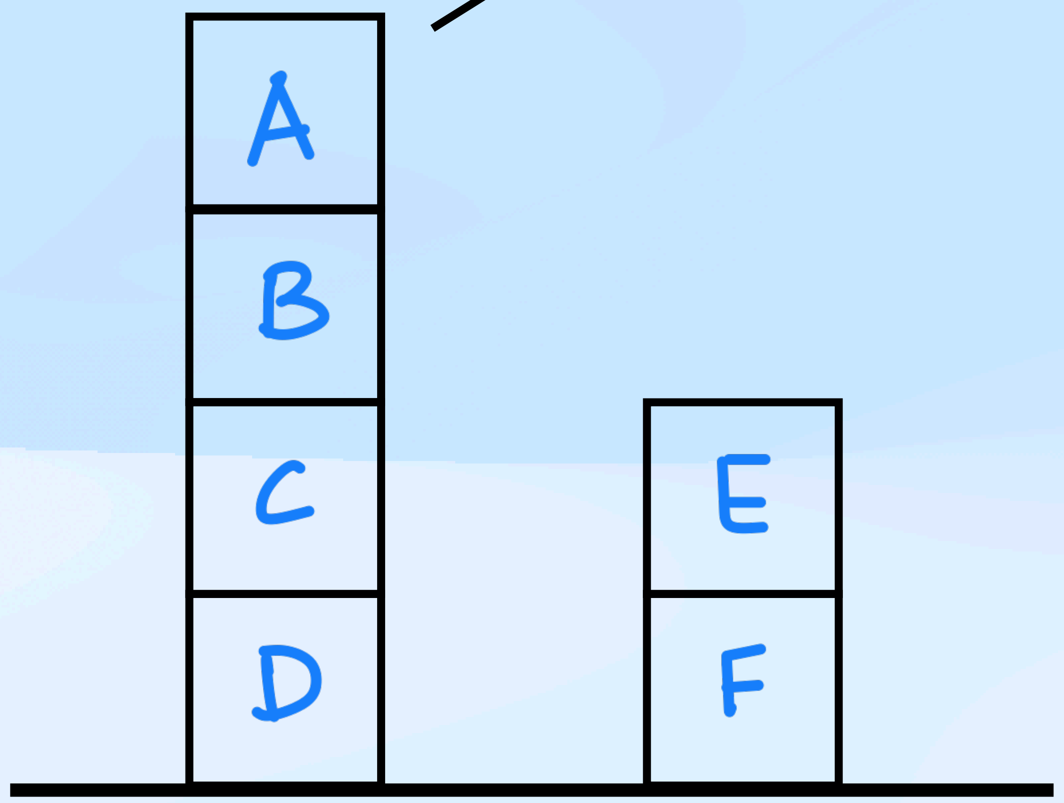
Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Blocks world domain



Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Blocks world domain



S1: A is on floor (2)

S2: E is on floor (2)

S3: A is on E (4)

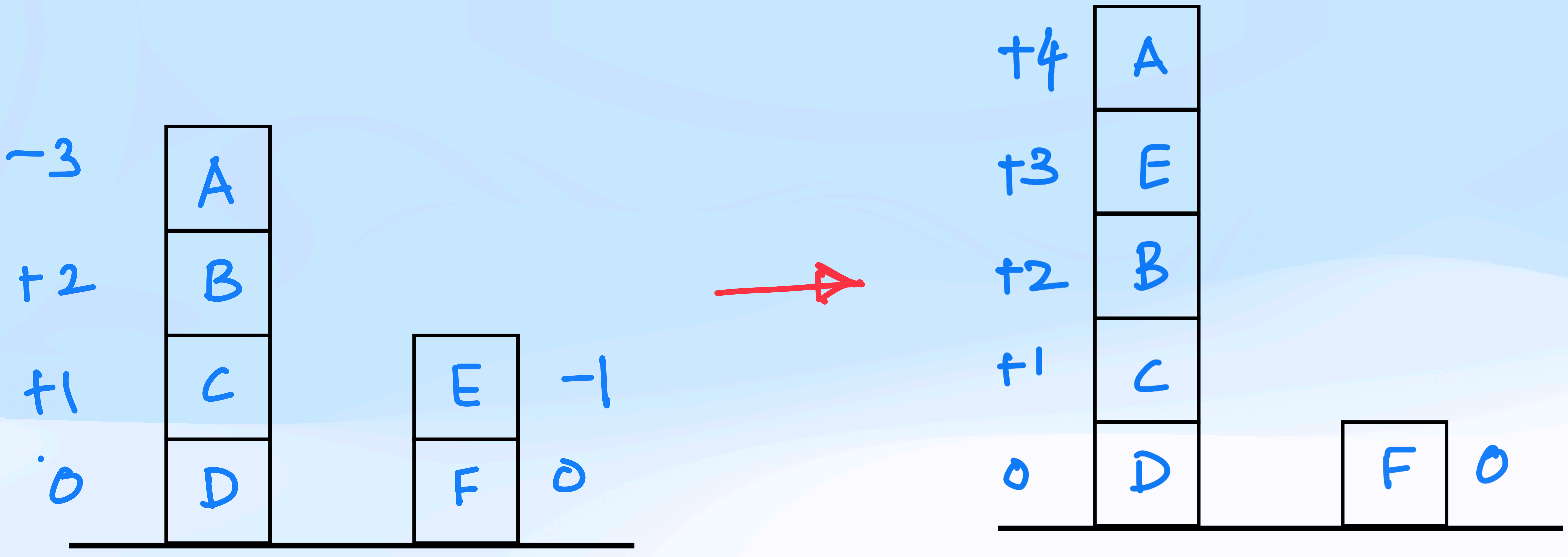
S4: E is on A (2)

Is our evaluation function good enough?

S1 seems a better state than others.

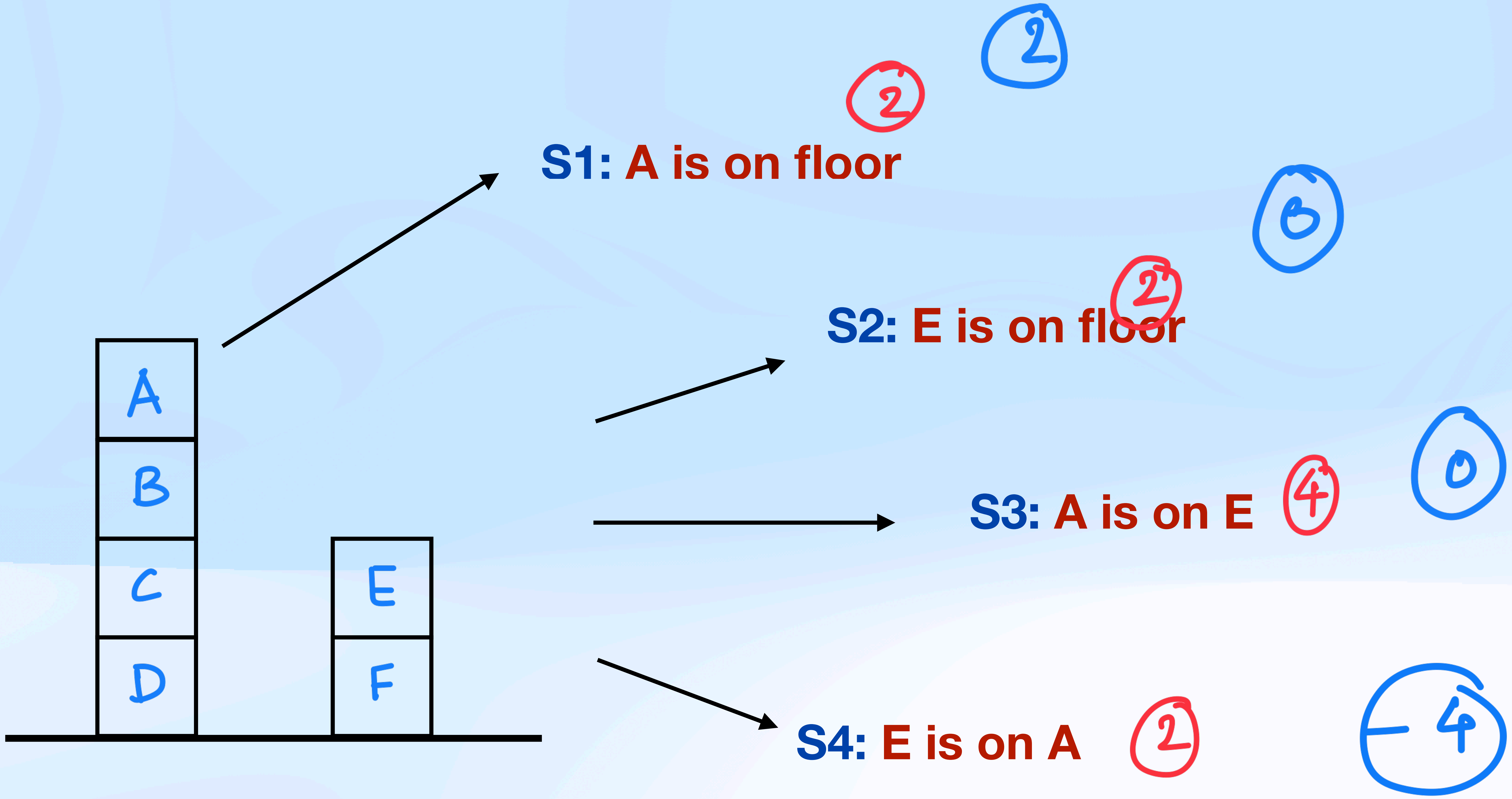
Val' (state) = for a block, if it is in correct configuration +1 for every block in configuration below it, else -1 for every block block in configuration

Blocks world domain



Val' (state) = for a block, if it is in correct configuration +1 for every block in configuration below it, else -1 for every block block in configuration

Blocks world domain



Take away: choice of evaluation function is crucial. The evaluation function should be discriminative at the same time efficient to compute.

Val (state) = 1 point for every block on correct block, -1 for every block block on incorrect block

Val' (state) = for a block, if it is in correct configuration +1 for every block in configuration below it, else -1 for every block block in configuration