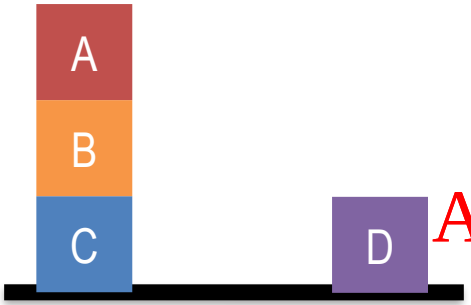
Planning

Lesson Preview

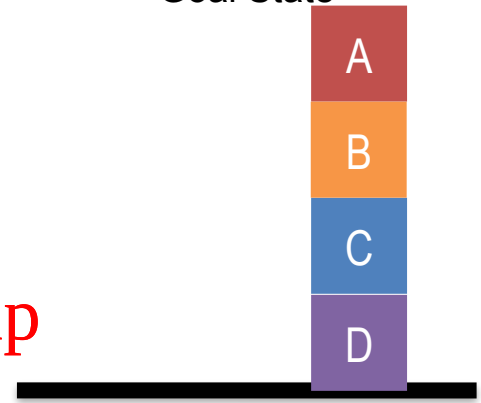
- States, goals, and operators
- Conflicts in planning
- Partial-order planning
- Hierarchical task networks

Current State

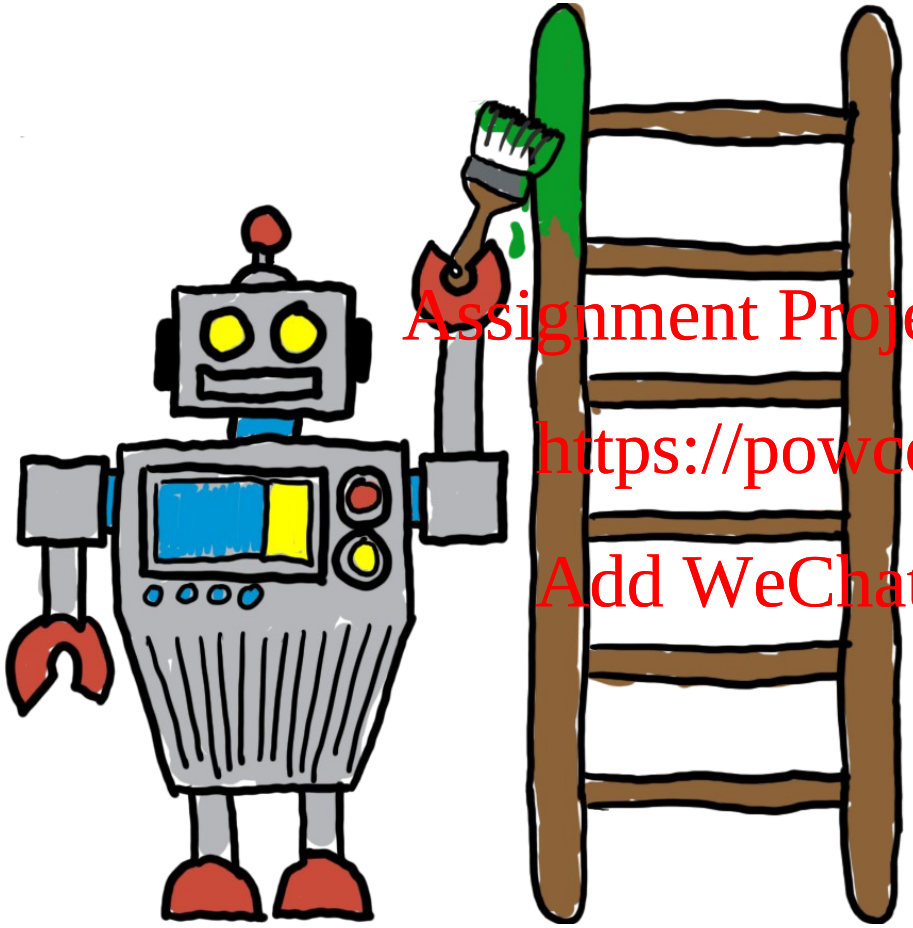


A on B
B on C
C on Table
D on Table
Δ = 1

Goal State



A on B
B on C
C on D
D on Table



Goals:

The ceiling is painted and the ladder is painted

In propositional logic:

`Painted(Ceiling)`

Goals:

The ceiling is painted and the ladder is painted

In propositional logic:

How would we represent second part of the goal state?


```
Painted(Ceiling)
Painted(Ladder)
```


Goal State:

How would we represent the goal state as a conjunction?

```
Painted(Ceiling) ∧
Painted(Ladder)
```

Initial State:

On(Robot, Floor) ∧
Dry(Ladder) ∧
Dry(Ceiling)

Goal State:

Painted(Ceiling) ∧
Painted(Ladder)

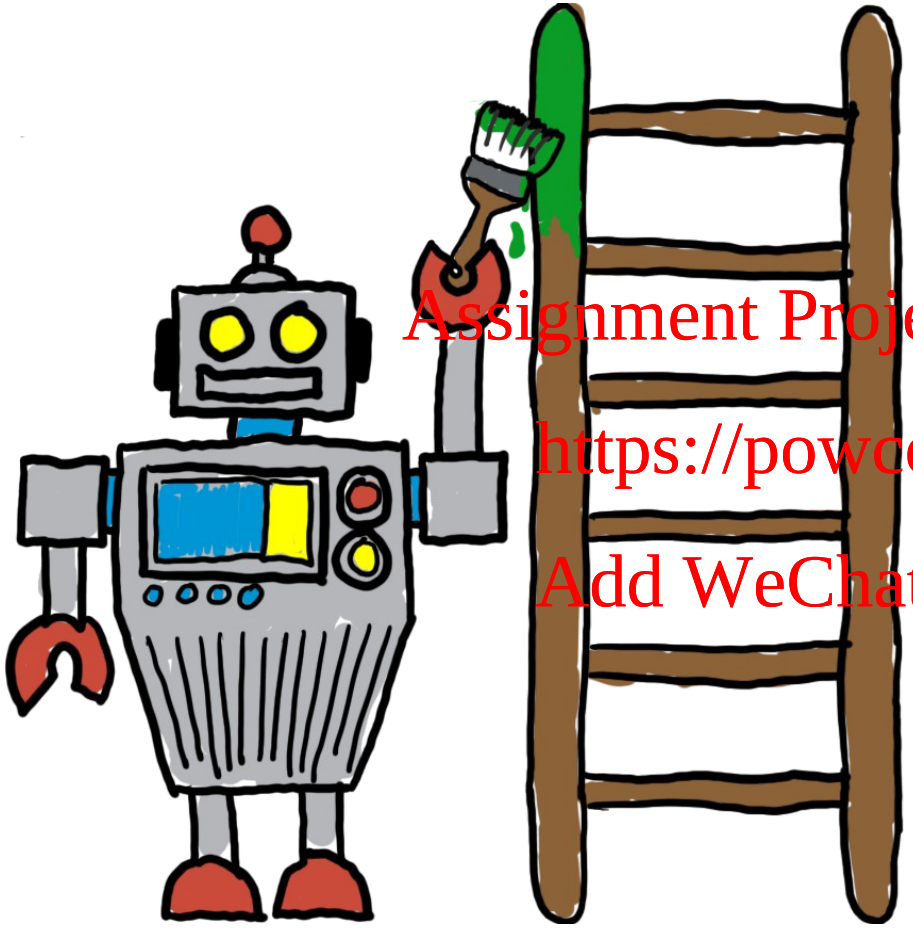
Initial State:

On(Robot, Floor) ∧
Dry(Ladder) ∧
Dry(Ceiling)

State:

Painted(Ceiling) ∧
On(Robot, Ladder)

How would we represent a state where the robot is on the ladder and the ceiling is painted?



Operator:
Climb ladder

In propositional logic:


```
climb-ladder:
    Precondition:
        On(Robot, Floor)
        ∧
        Dry(Ladder)
    Postcondition:
        On(Robot,
Ladder)
```

climb-ladder:

Precondition:

On(Robot, Floor) ∧ Dry(Ladder)

Postcondition:

On(Robot, Ladder)

descend-ladder:

Precondition:

On(Robot, Ladder) ∧ Dry(Ladder)

paint-ceiling:

Precondition:

On(Robot, Ladder)

Postcondition:

Painted(Ceiling) ∧ ¬Dry(Ceiling)

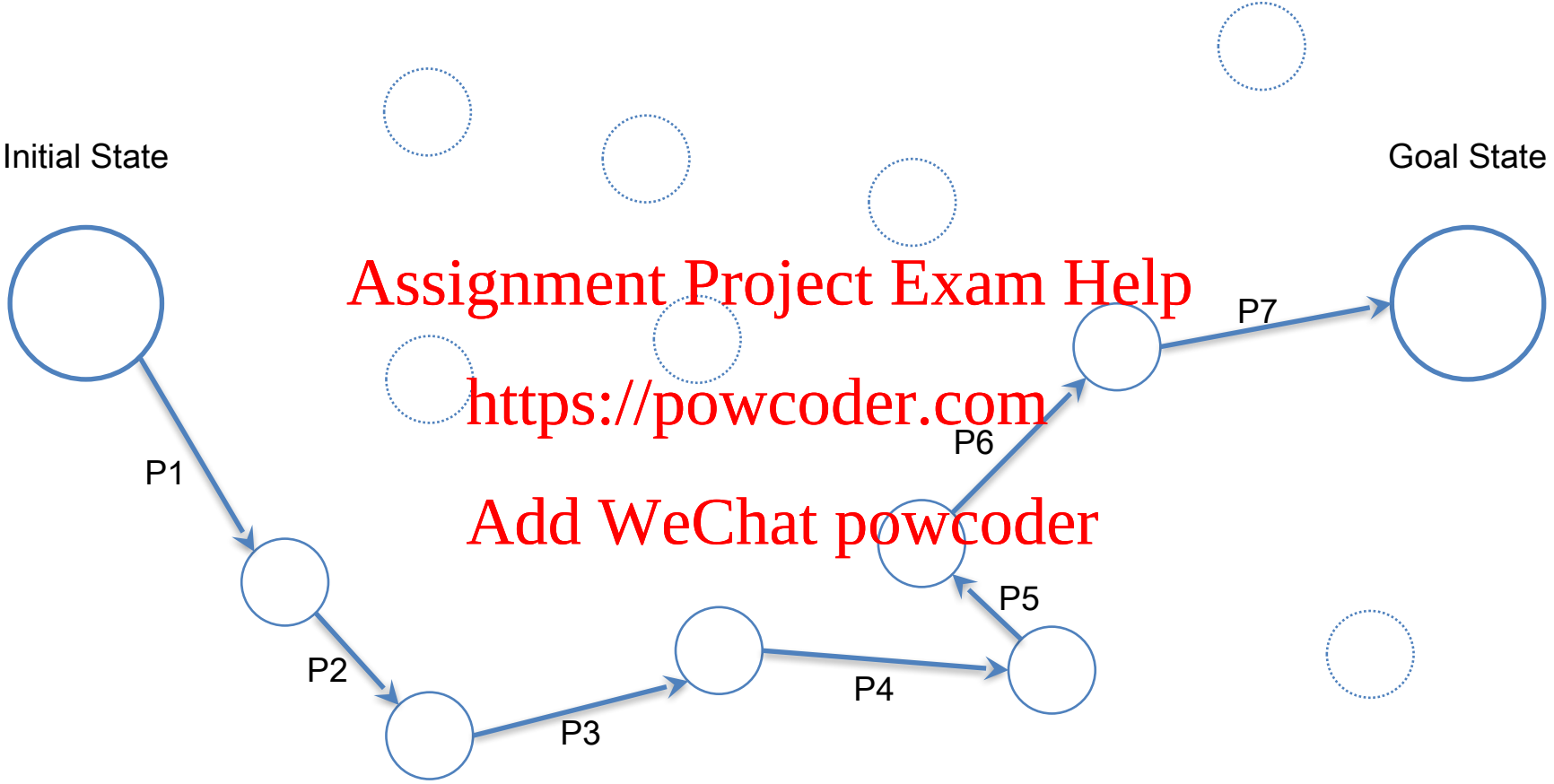
paint-ladder:

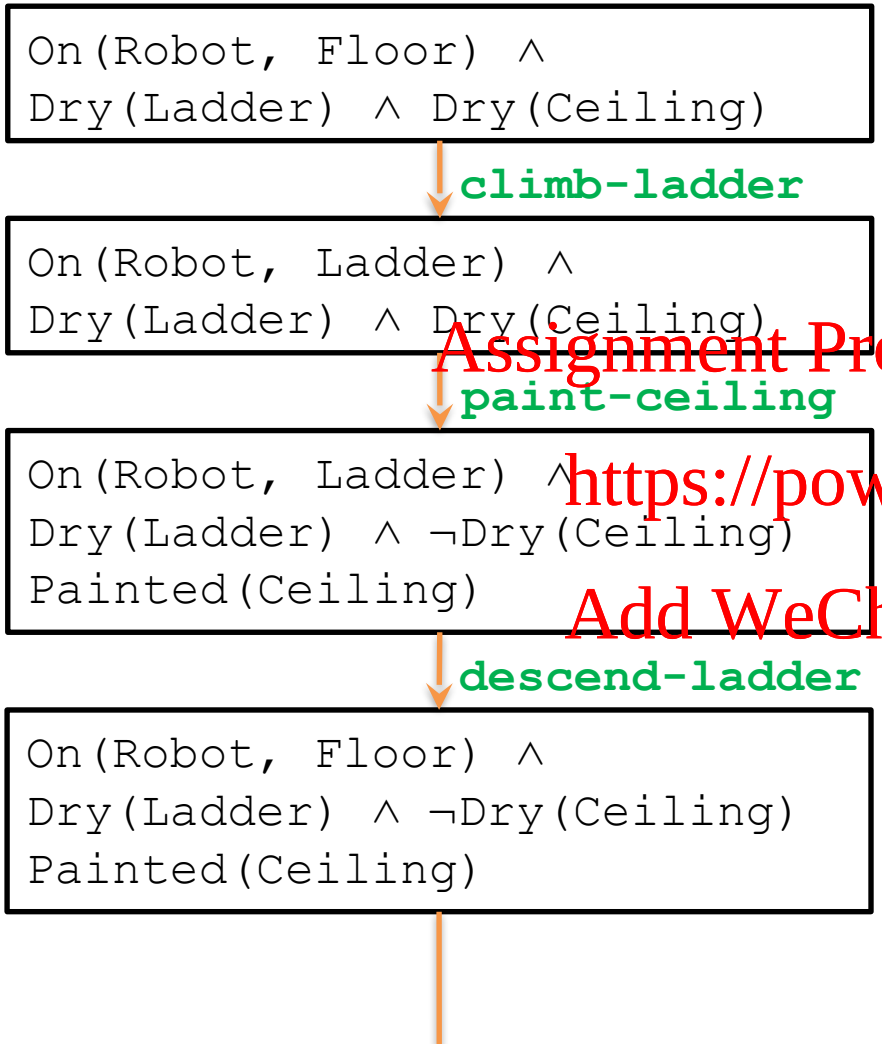
Precondition:

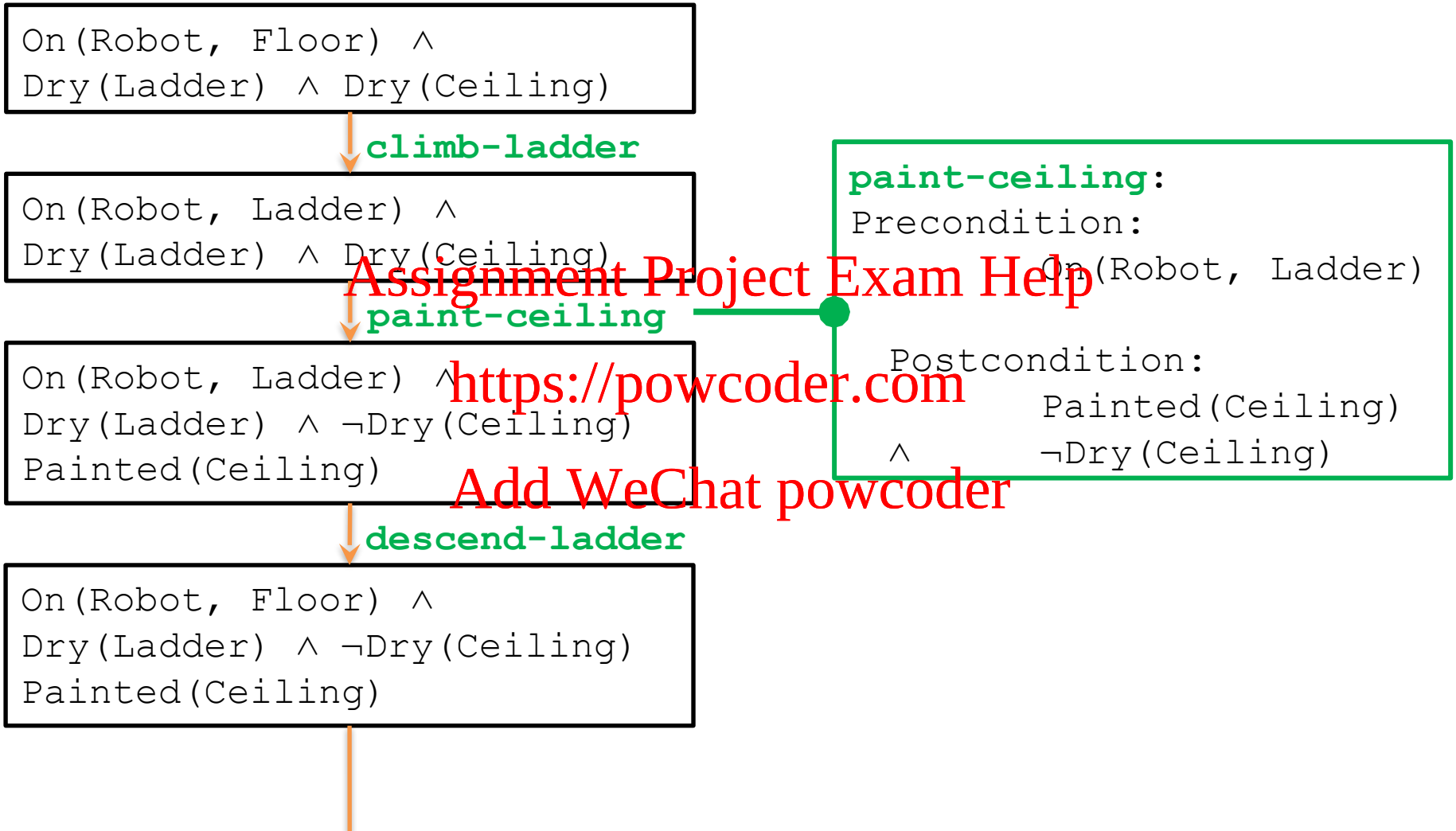
On(Robot, Floor)

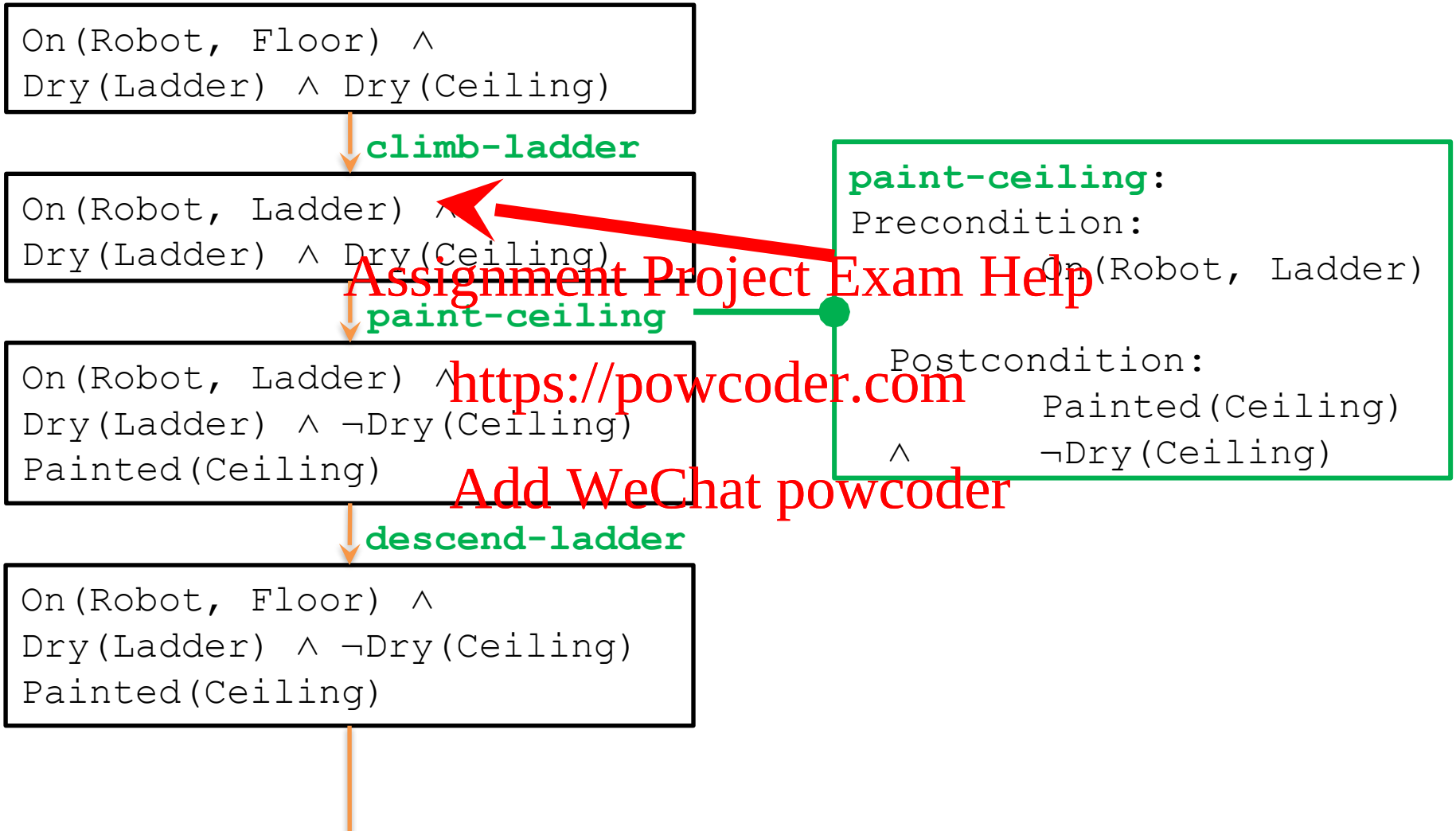
Postcondition:

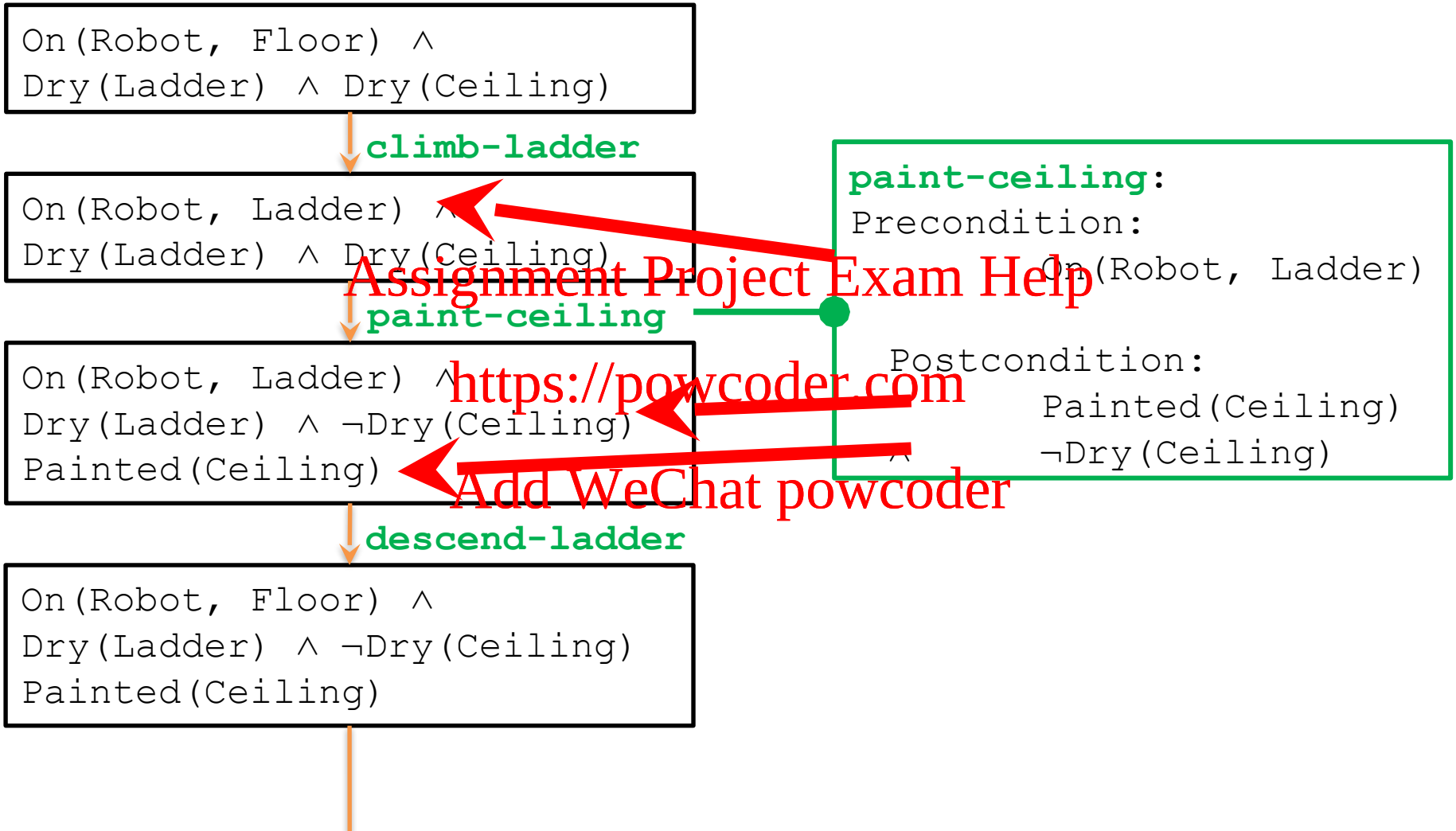
Painted(Ladder)











Goal State:

Painted(Ladder) ∧
Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

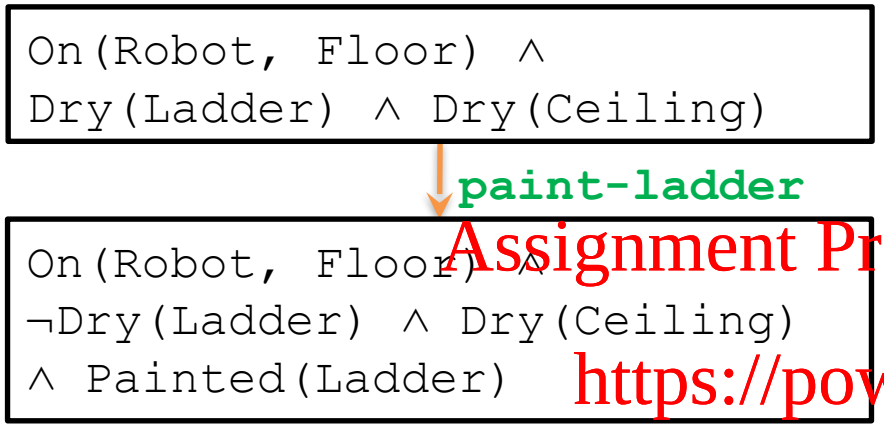
paint-ladder:
Precondition:
On(Robot, Floor)

Postcondition:
Painted(Ladder) ∧
¬Dry(Ladder)

climb-ladder:
Precondition:
On(Robot, Floor)
∧
Dry(Ladder)

Postcondition:
On(Robot, Ladder)

Goal: Painted(Ladder)



Goal: Painted(Ladder)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

Goal: Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

climb-ladder

On(Robot, Ladder) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ceiling

On(Robot, Ladder) ∧
Dry(Ladder) ∧ ¬Dry(Ceiling)
∧ Painted(Ceiling)

Detecting Conflicts

For each precondition in current plan:

If precondition for an operator in the current plan is clobbered by a state in another plan:

Promote current plan above other plan

Goal: Painted(Ladder)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

Goal: Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

climb-ladder

On(Robot, Ladder) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ceiling

On(Robot, Ladder) ∧
Dry(Ladder) ∧ ¬Dry(Ceiling)
∧ Painted(Ceiling)

Goal: Painted(Ladder)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

Goal: Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

climb-ladder

On(Robot, Ladder) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ceiling

On(Robot, Ladder) ∧
Dry(Ladder) ∧ ¬Dry(Ceiling)
∧ Painted(Ceiling)

climb-ladder:
Precondition:
On(Robot, Floor)
∧
Dry(Ladder)

Postcondition:
On(Robot, Ladder)

Goal: Painted(Ladder)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

climb-ladder:
Precondition:
On(Robot, Floor)
∧
Dry(Ladder)

Postcondition:
On(Robot, Ladder)

Goal: Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

climb-ladder

On(Robot, Ladder) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ceiling

On(Robot, Ladder) ∧
Dry(Ladder) ∧ ¬Dry(Ceiling)
∧ Painted(Ceiling)

Goal: Painted(Ladder)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ladder

On(Robot, Floor) ∧
¬Dry(Ladder) ∧ Dry(Ceiling)
∧ Painted(Ladder)

Goal: Painted(Ceiling)

On(Robot, Floor) ∧
Dry(Ladder) ∧ Dry(Ceiling)

climb-ladder

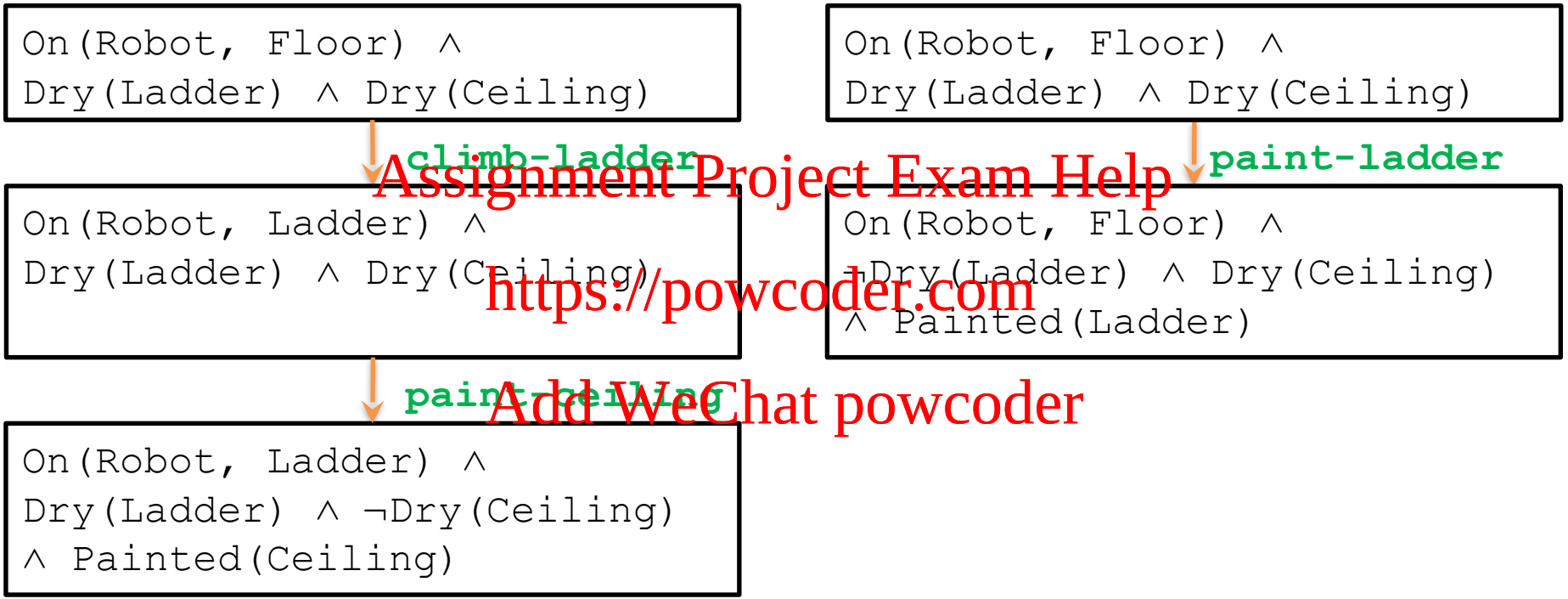
On(Robot, Ladder) ∧
Dry(Ladder) ∧ Dry(Ceiling)

paint-ceiling

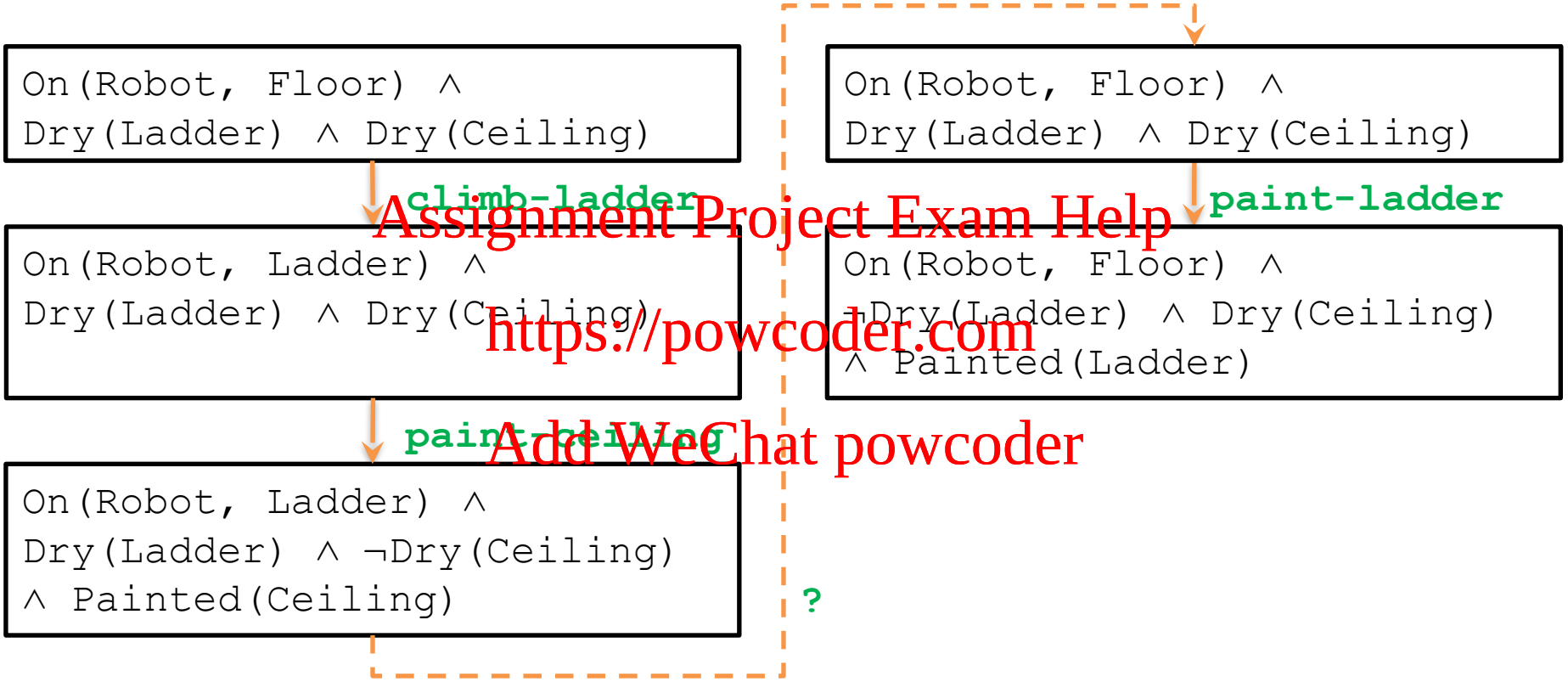
On(Robot, Ladder) ∧
Dry(Ladder) ∧ ¬Dry(Ceiling)
∧ Painted(Ceiling)

Result: Promote
Painted(Ceiling) above
Painted(Ladder)

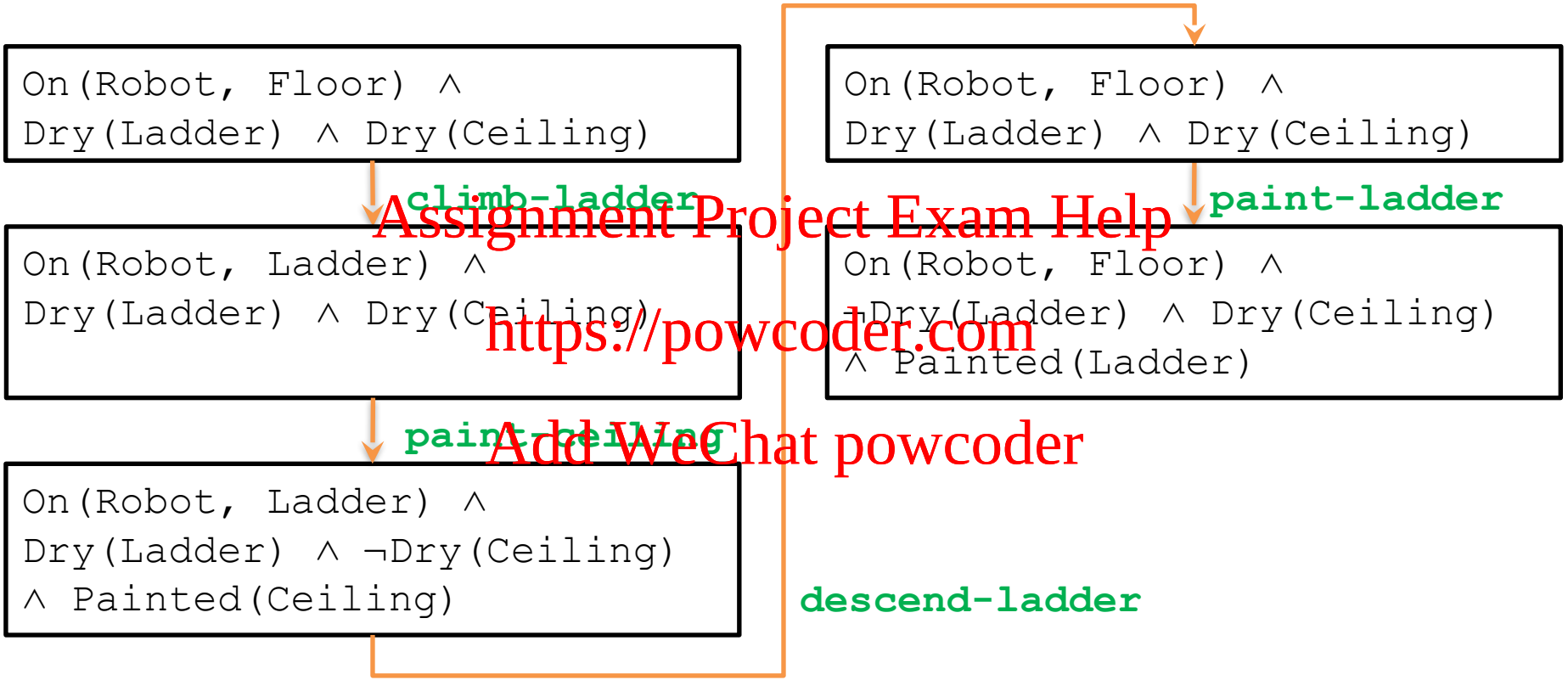
Final Plan



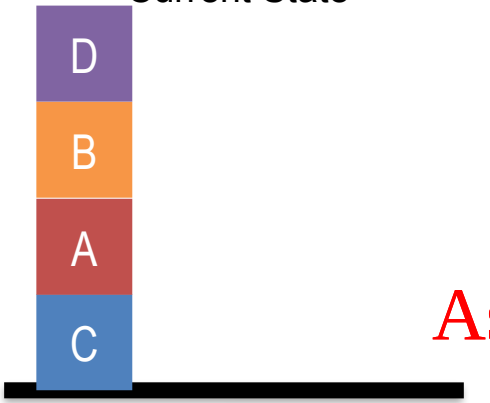
Final Plan



Final Plan

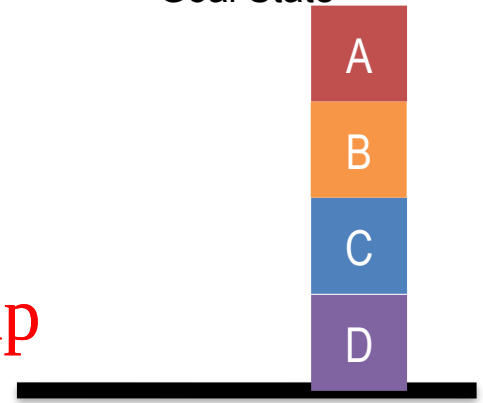


Current State




```
On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)
```

Goal State




```
On(A, B) ∧
On(B, C) ∧
On(C, D) ∧
On(D, Table)
```


Write the current and goal states in propositional logic.

Syntax:

D is on B → `On(D, B)`

Top of D is clear → `Clear(D)`

Current State


```
On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)
```

Write the pre- and post-conditions for the two Move operators.

Operator:
Move block x to block y

```
Move(x, y)
Precondition:
    Clear(x) ∧
Clear(y)
Postcondition:
    On(x, y)
```

Operator:
Move block x to table

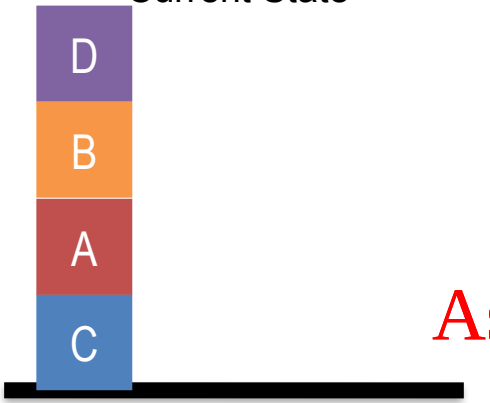
```
Move(x, Table):
Precondition:
    Clear(x)
Postcondition:
    On(x, Table)
```

Syntax:
D is on B → `On(D, B)`
Top of D is clear → `Clear(D)`

Goal State


```
On(A, B) ∧
On(B, C) ∧
On(C, D)
```

Current State




```
On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)
```

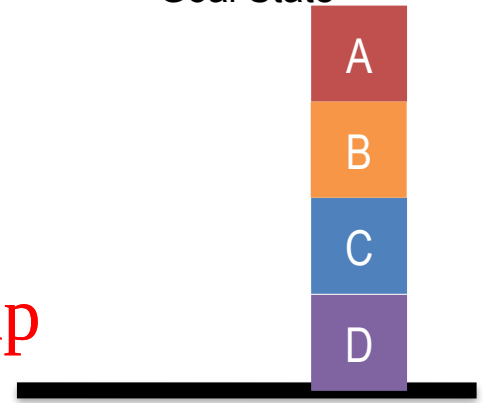
Write plans for accomplishing each goal.

```
On(A, B):
        Move(D, Table)
        Move(B, Table)
        Move(A, B)

On(B, C):
        Move(D, Table)
        Move(B, Table)
        Move(A, Table)
        Move(B, C)

On(C, D):
        Move(D, Table)
        Move(B, Table)
        Move(A, Table)
        Move(C, D)
```


Goal State



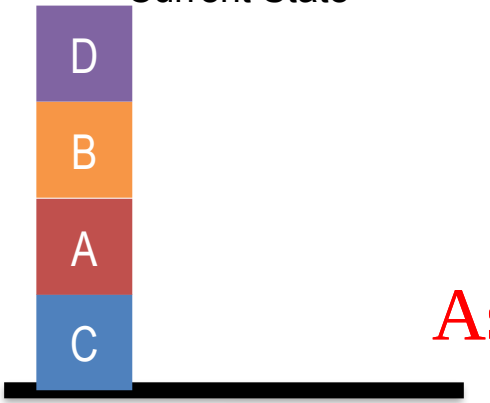

```
On(A, B) ∧
On(B, C) ∧
On(C, D)
```

D is on B → `On(D, B)`

Top of D is clear → `Clear(D)`

Put D on Table → `Move(D, Table)`

Current State



On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)

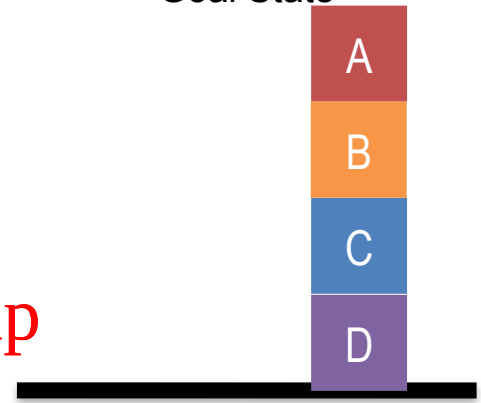
Use partial-order planning to order the plans.

On(A, B):
- Move(D, Tab 1
- Move(B, Table)
- Move(A, B)

On(B, C):
- Move(D, Tab 2
- Move(B, Table)
- Move(A, Table)
- Move(B, C)

On(C, D):
- Move(D, Tab 3
- Move(B, Table)
- Move(A, Table)
- Move(C, D)

Goal State



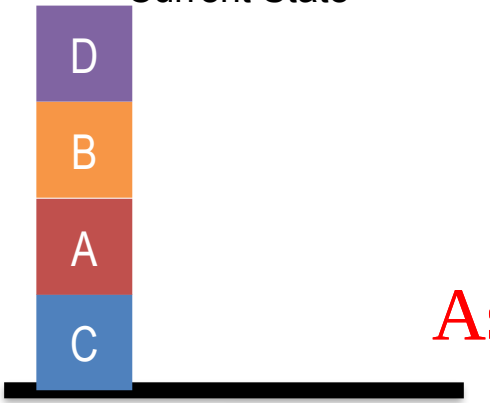
On(A, B) ∧
On(B, C) ∧
On(C, D)

D is on B → On(D, B)

Top of D is clear → Clear(D)

Put D on Table → Move(D, Table)

Current State



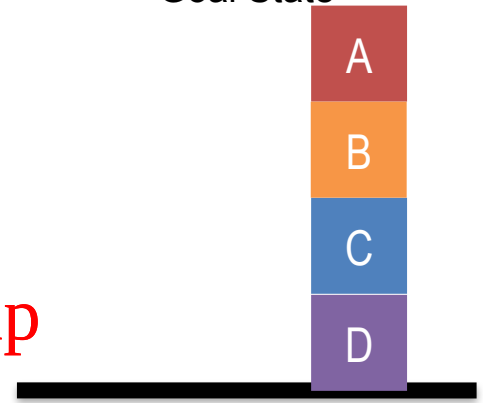
On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)

Write a final plan for converting the current state to the goal state.

Final Plan:

Move(D, Table)
Move(B, Table)
Move(A, Table)
Move(C, D)
Move(B, C)
Move(A, B)

Goal State



On(A, B) ∧
On(B, C) ∧
On(C, D)

D is on B → On(D, B)

Top of D is clear → Clear(D)

Put D on Table → Move(D, Table)

Current State

Final Plan:

```
Move(D, Table)
Move(B, Table)
Move(A, Table)
Move(C, D)
Move(B, C)
Move(A, B)
Move(H, Table)
Move(F, Table)
Move(G, Table)
Move(G, H)
Move(F, G)
Move(E, F)
Move(K, Table)
Move(L, Table)
Move(J, Table)
Move(K, L)
Move(J, K)
Move(I, J)
Move(O, Table)
Move(N, Table)
Move(P, Table)
Move(O, P)
Move(N, O)
Move(M, N)
```

Goal State

Current State


```
On(D, B) ∧
On(B, A) ∧
On(A, C) ∧
On(C, Table)
```

Final Plan:

```
Move(D, Table)
Move(B, Table)
Move(A, Table)
Move(C, D)
Move(B, C)
Move(A, B)
```

unstack

stack-ascending

Goal State


```
On(A, B) ∧
On(B, C) ∧
On(C, D)
```

D is on B → `On(D, B)`

Top of D is clear → `Clear(D)`

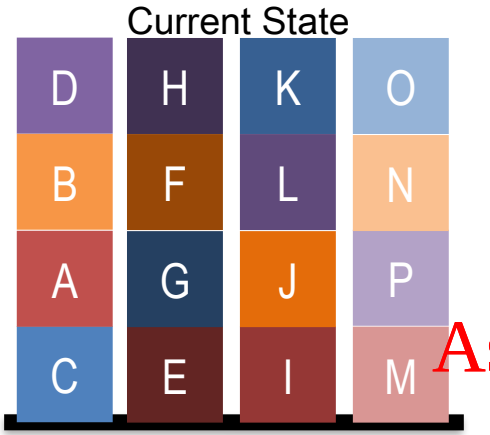
Put D on Table → `Move(D, Table)`

unstack:

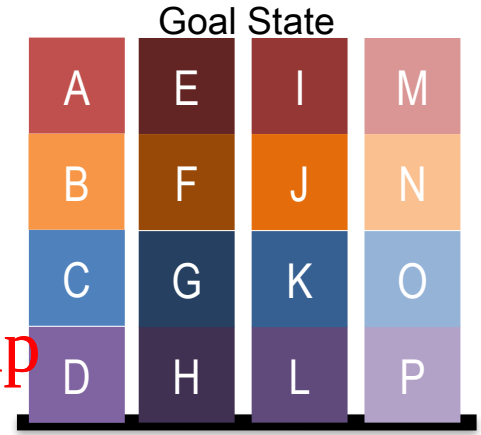
Precondition:
On(w, x) ∧
On(x, y) ∧
On(y, z) ∧
On(z, Table) ∧
Postcondition:
On(w, Table) ∧
On(x, Table) ∧
On(y, Table) ∧
On(z, Table)
Method:
Move(w, Table)
Move(x, Table)
Move(y, Table)

stack-ascending:

Precondition:
On(a, Table) ∧
On(b, Table) ∧
On(c, Table) ∧
On(d, Table)
Postcondition:
On(a, b) ∧
On(b, c) ∧
On(c, d) ∧
On(d, Table)
Method:
Move(c, d)
Move(b, c)
Move(a, b)




```
Final Plan:
        unstack(D, B, A, C)
        stack-ascending(A, B,
C, D)
        unstack(H, F, G, E)
        stack-ascending(E, F,
G, H)
        unstack(K, I, J, L)
        stack-ascending(I, J,
K, L)
        unstack(O, N, P, M)
        stack-ascending(M, N,
O, P)
```



Assignment

How would you use planning to address Raven's progressive matrices?

To recap…

- Planning in propositional logic
- Goal conflicts
- Partial-order planning for conflict avoidance
- Hierarchical task networks