Stavros Vassos, University of Athens, Greece stavrosv@di.uoa.gr May 2012

INTRODUCTION TO AI STRIPS PLANNING

.. and Applications to Video-games!

Course overview

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, **state-space search**
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS

STRIPS planning

- STRIPS! So why do we like this formalism?

STRIPS planning

- STRIPS! So why do we like this formalism?
  - Simple formalism for representing planning problems
  - Easy to compute applicable actions
    - Check whether the list of preconditions is a subset of the state description: PRECONDITIONS $\subseteq$ S
  - Easy to compute the successor states
    - Add the list of positive effects to the state description and remove the list of negative effects:
      S' = (S / NEGATIVE-EFFECTS) $\cup$ POSITIVE-EFFECTS
  - Easy to check if the goal is satisfied
    - Check whether the goal is a subset of the state description: G $\subseteq$ S

STRIPS planning

- STRIPS! So why do we like this formalism?
  - It can already describe difficult and complex problems (in more challenging domains than the example we saw)
  - …let's see how we can solve this kind of problems

STRIPS planning: state-based search

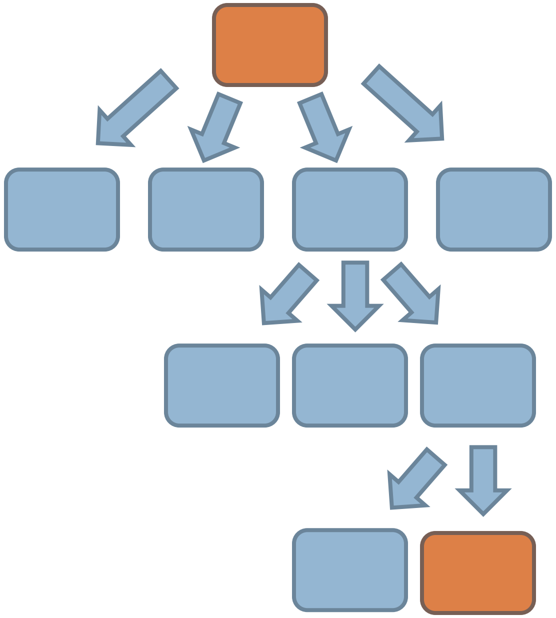
- Finding a solution to the planning problem following a state-based search
  - Init(On(A,Table) ∧ On(B,Table) ∧ …)
  - Goal(On(A,B) ∧ …)
  - Action(Move(b,x,y),

    PRECONDITIONS: On(b,x) ∧ …

    EFFECTS: On(b,y) ∧ …)
  - Action(MoveToTable(b,x),

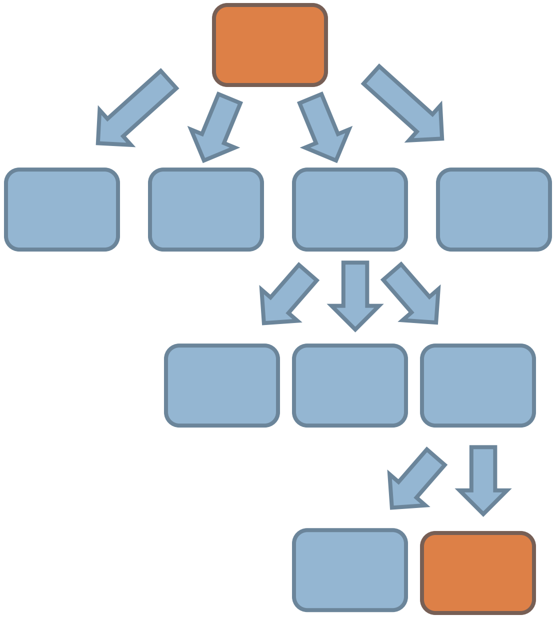
    PRECONDITIONS: On(b,x) ∧ …

    EFFECTS: On(b,Table) ∧ …)



STRIPS planning: state-based search

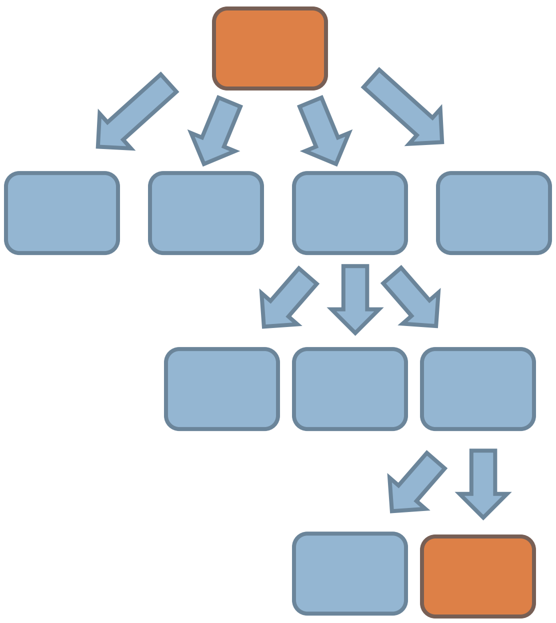
- Finding a solution to the planning problem following a state-based search
  - Init(where to start from)
  - Goal(when to stop searching)
  - Action(how to generate the "graph")



- Progression planning: forward state-based search
- Regression planning: backward state-based search

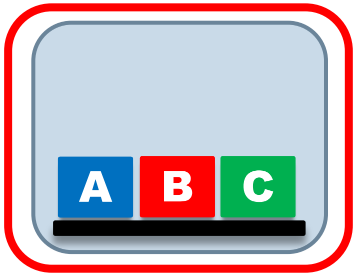
Progression planning

- Start from the initial state
- Check if the current state satisfies the goal
- Compute applicable actions to the current state
- Compute the successor states
- Pick one of the successor states as the current state
- Repeat until a solution is found or the state space is exhausted



Progression planning

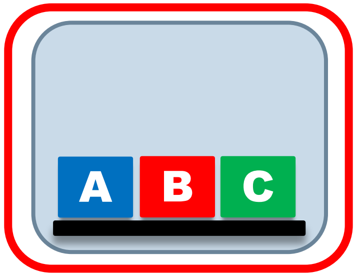
- Start from the initial state



On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

Progression planning

- Check if the current state satisfies the goal
  - No

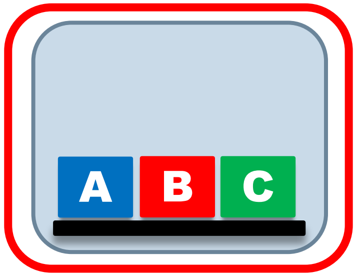


On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

On(A,B)
On(B,C)

Progression planning

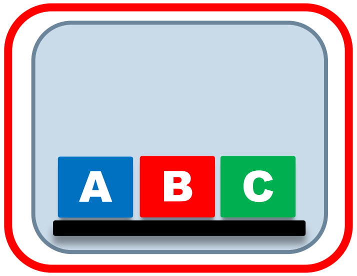
- Compute applicable actions to the current state
  - Action(Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))



On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

Progression planning

- Compute applicable actions to the current state
  - Action(Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))
    - Move(B,Table,C)
      Preconditions:
      - On(B,Table)
      - Clear(B)
      - Clear(C)

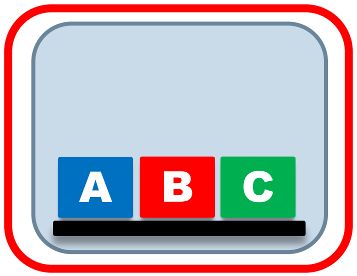


On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

    - Applicable action!

Progression planning

- Compute applicable actions to the current state
  - Action(Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))
    - Move(A,Table,B)
    - Move(A,Table,C)
    - Move(B,Table,A)
    - Move(B,Table,C)
    - Move(C,Table,A)
    - Move(C,Table,B)



On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

- All these are applicable actions!

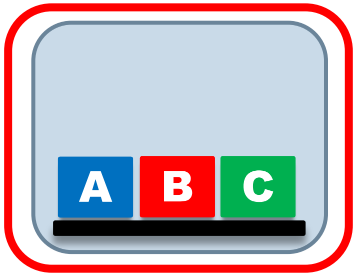
Progression planning

- Compute applicable actions to the current state
  - Action(Move(b,x,y),

    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),

    PRECONDITIONS: On(b,x) ∧ Clear(b))
    - Move(B,Table,B)

      Preconditions:
      - On(B,Table)
      - Clear(B)



On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

- This is also an applicable action!

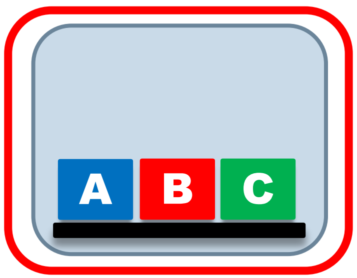
Progression planning

- Compute applicable actions to the current state
  - Action(Move(b,x,y),

    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),

    PRECONDITIONS: On(b,x) ∧ Clear(b))
    - MoveTT(B,Table)

      Preconditions:
      - On(B,Table)
      - Clear(B)

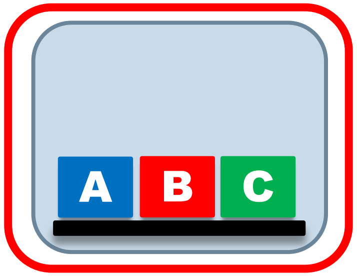


On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

- This is also an applicable action!

Progression planning

- Compute applicable actions to the current state
  - Action(Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action(MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))
    - Move(A,Table,A)
    - Move(B,Table,B)
    - Move(C,Table,C)
    - MoveTT(A,Table)
    - MoveTT(B,Table)
    - MoveTT(C,Table)

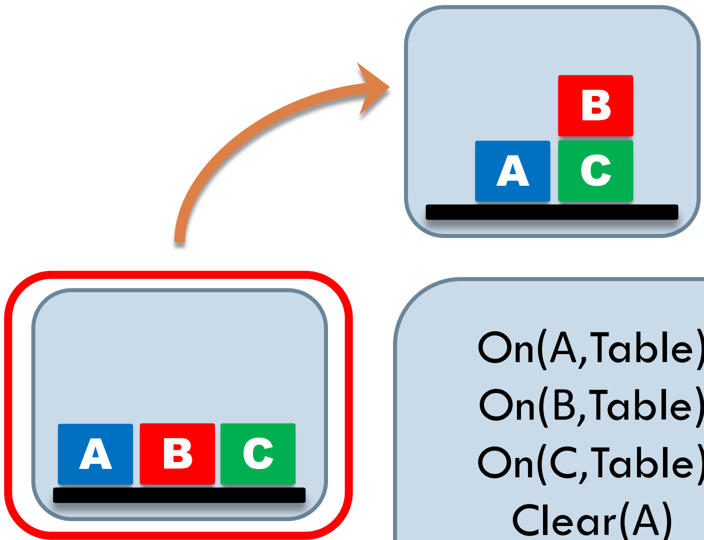


On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

- All these are applicable also actions!

Progression planning

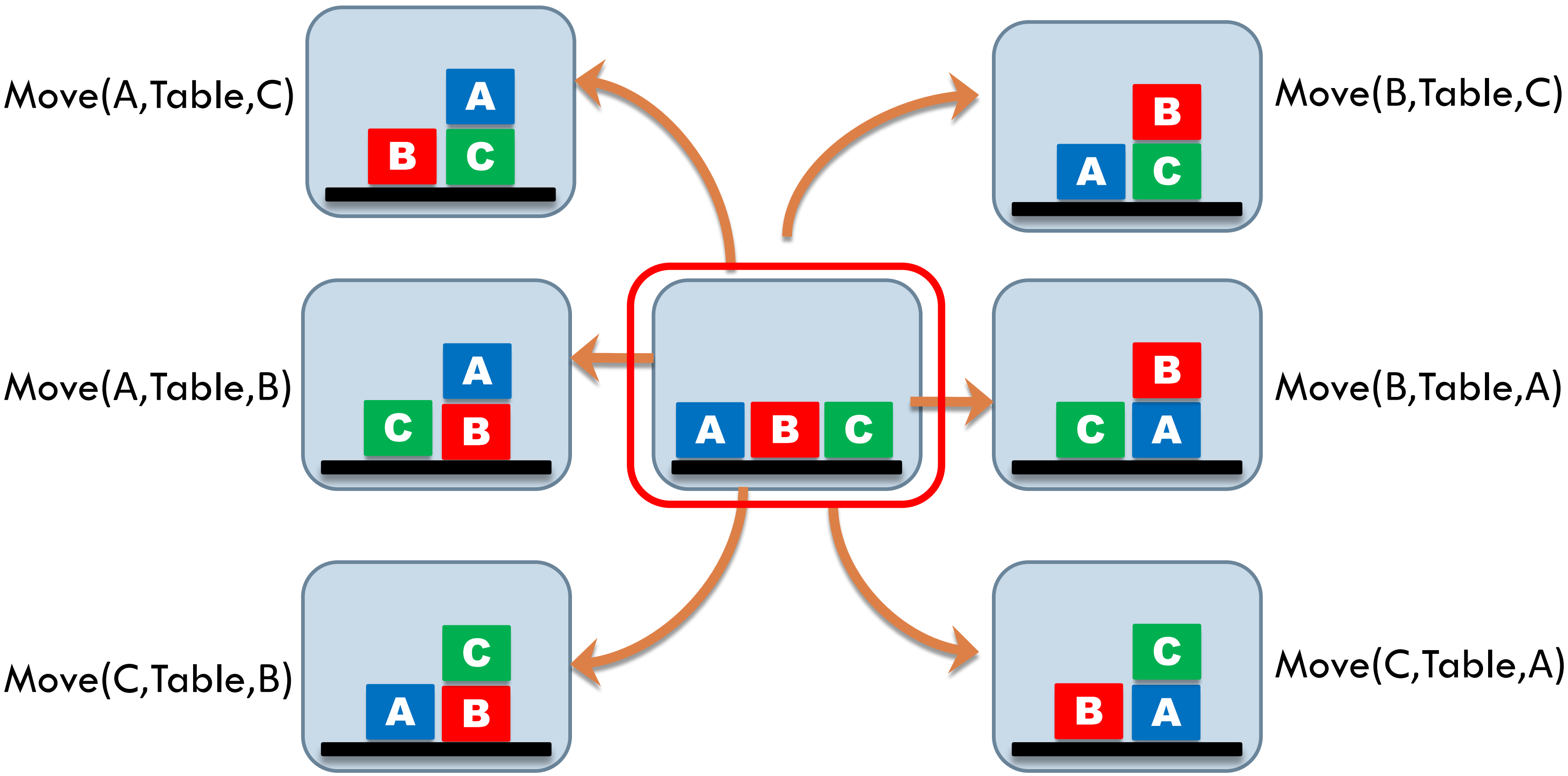
- Compute the successor states
  - Action(Move(b,x,y),
    EFFECTS: On(b,y) ∧
    Clear(x) ∧
    ¬On(b,x) ∧
    ¬Clear(y))
  - Move(B,Table,C)
    Effects:
    - On(B,C)
    - Clear(Table)
    - ¬ On(B,Table)
    - ¬Clear(C)



On(A,Table)
On(B,Table)
On(C,Table)
Clear(A)
Clear(B)
Clear(C)

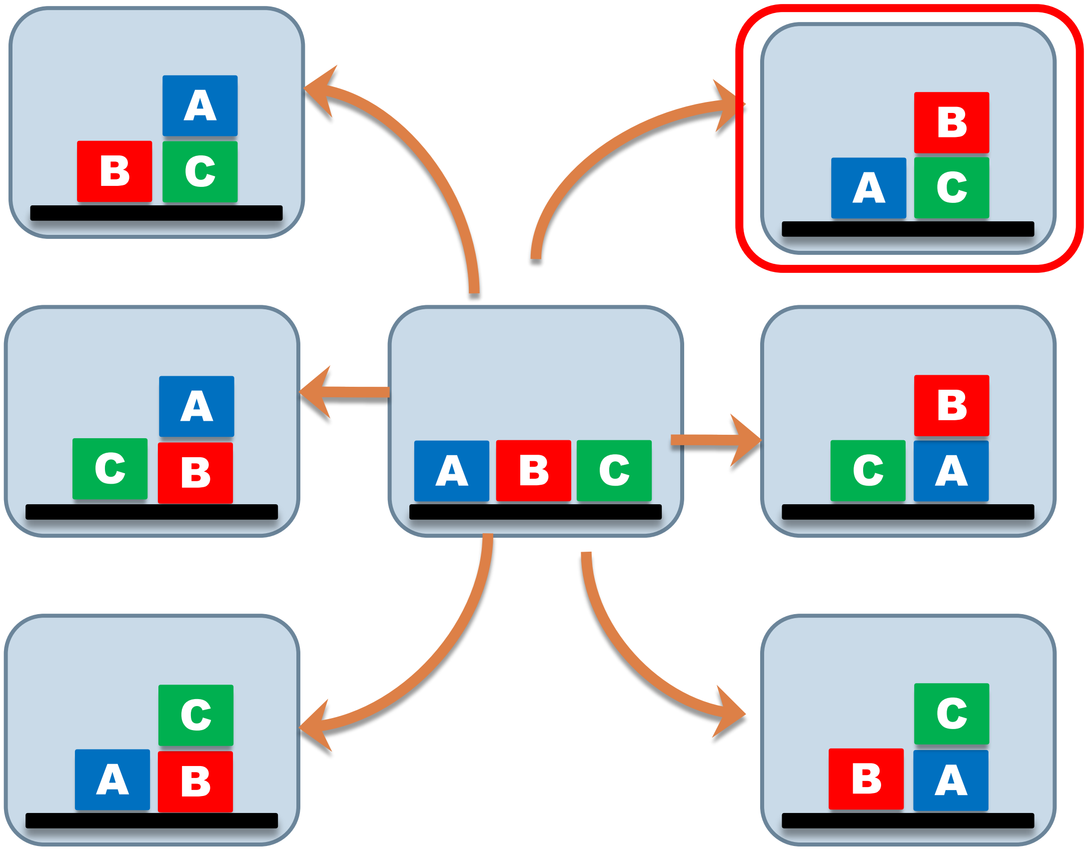
Progression planning

- Compute the successor states



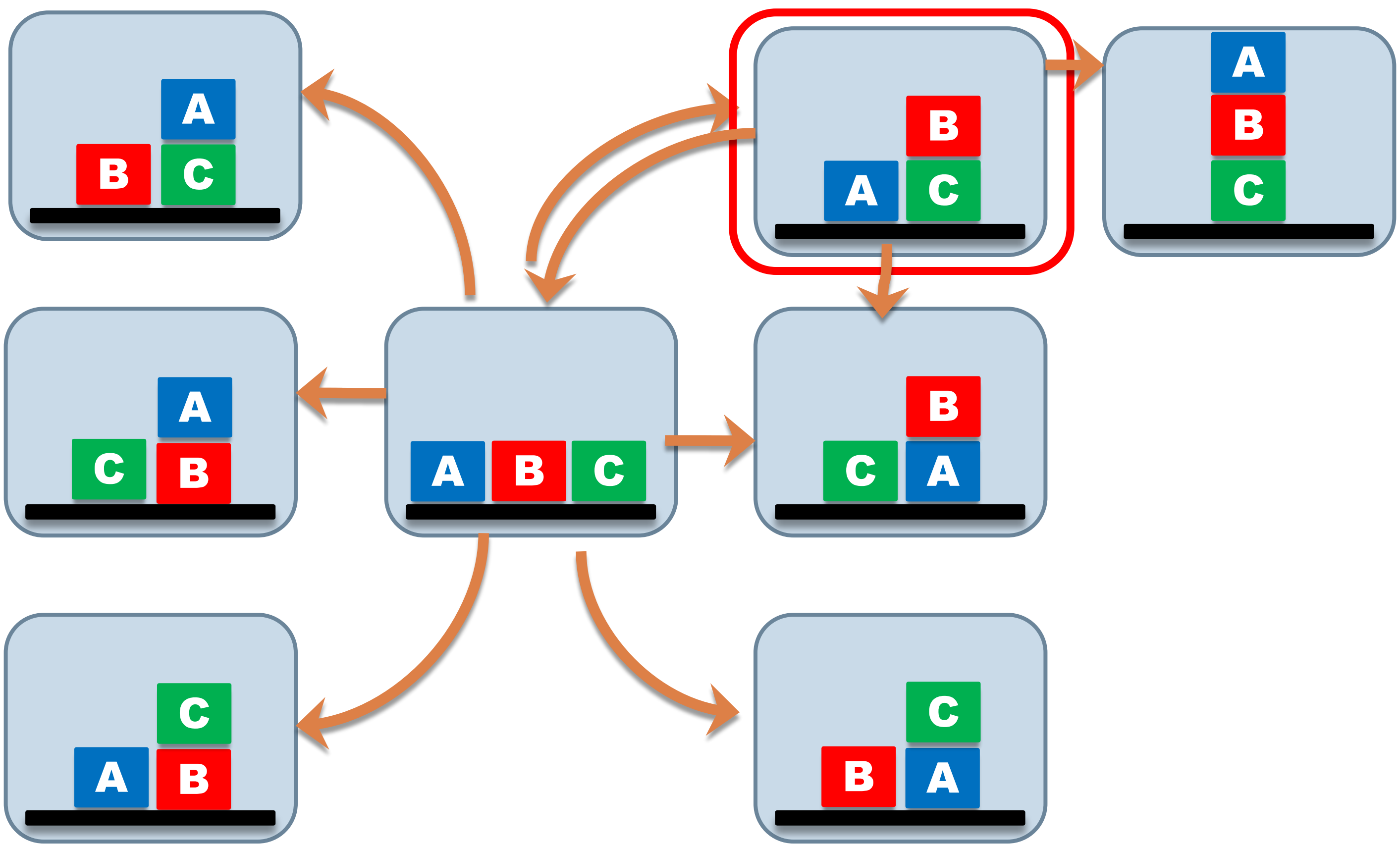
Progression planning

- Pick one of the successor states as current..



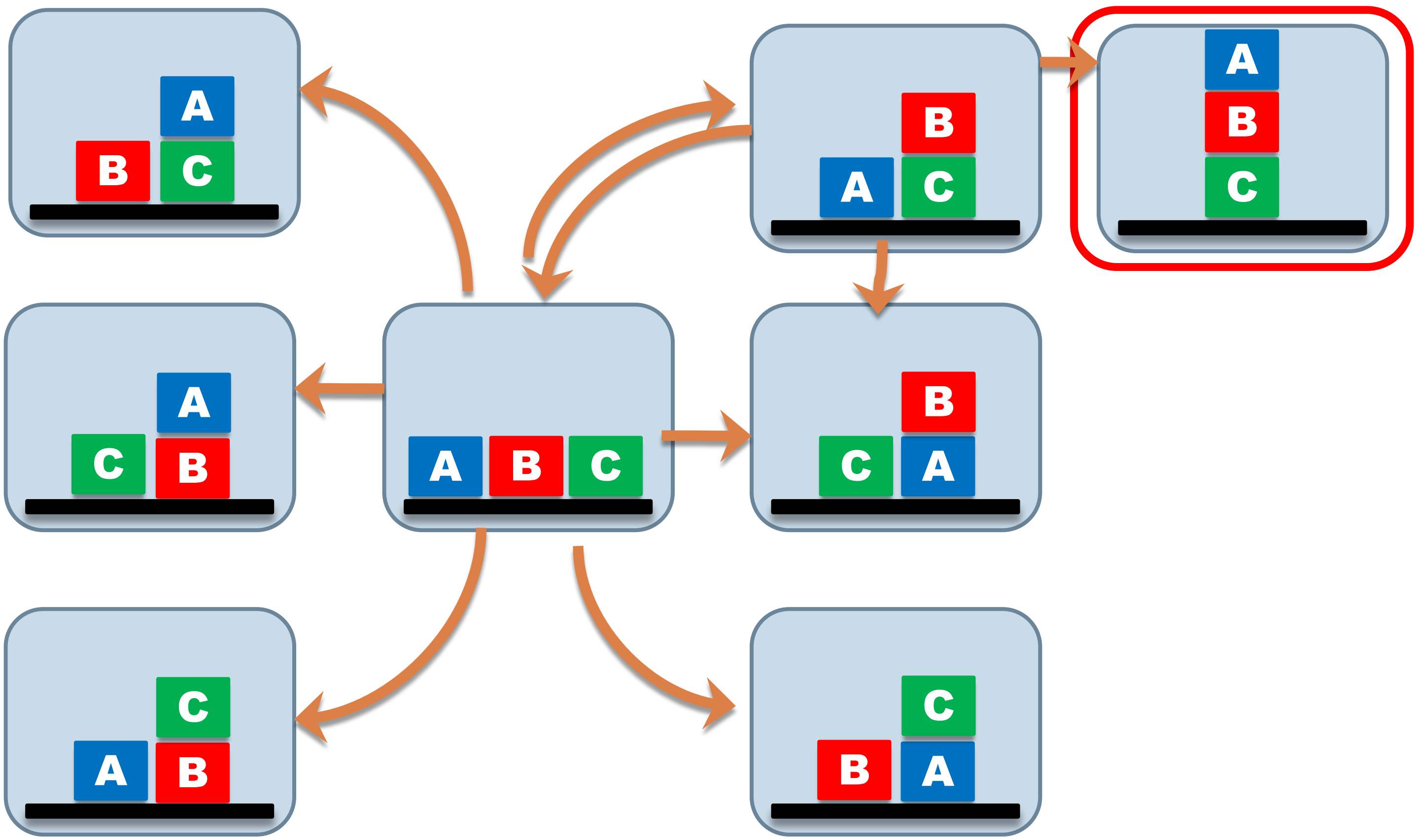
Progression planning

- ..and repeat!



Progression planning

- Pick one of the successor states as current



Progression planning

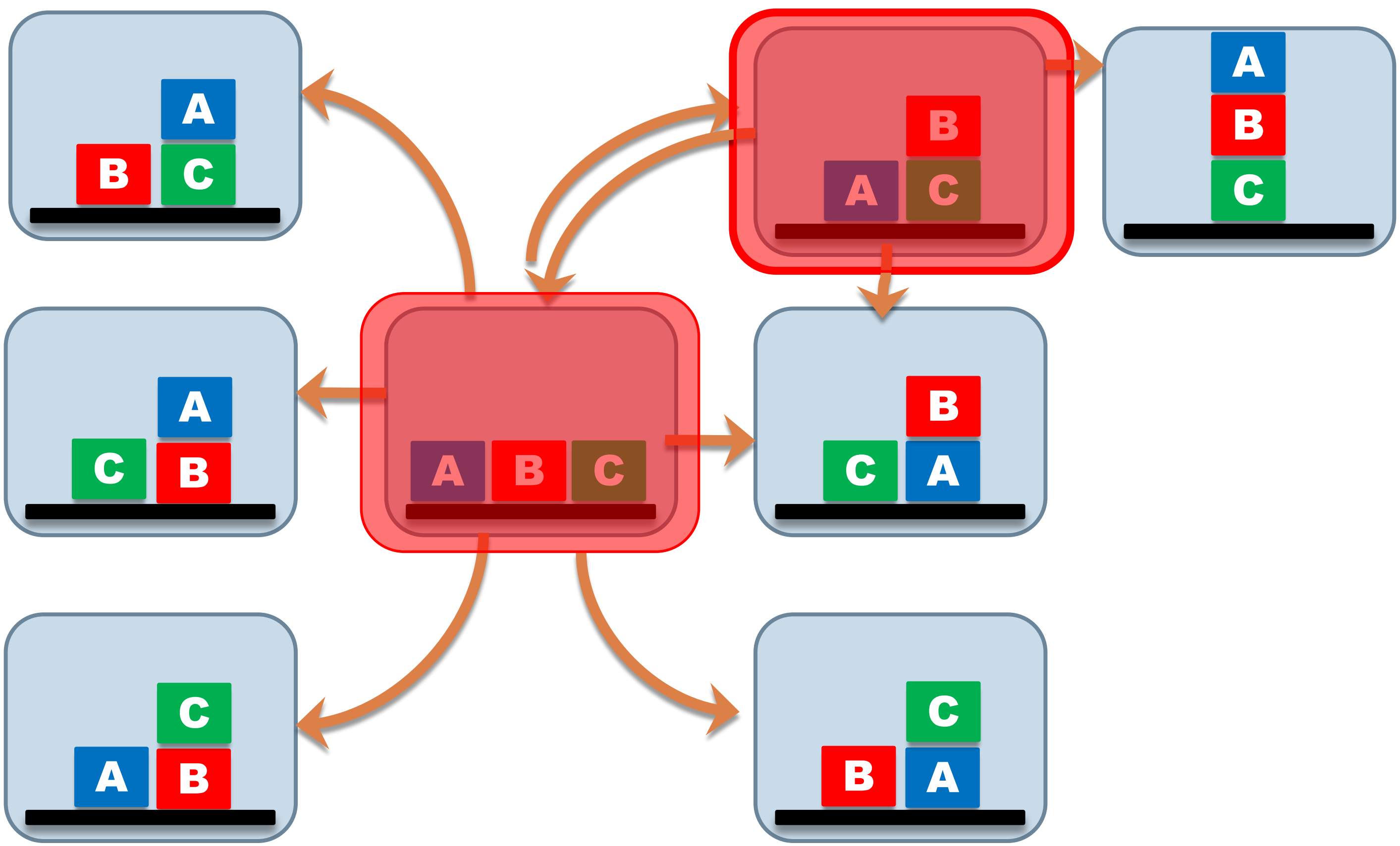
- Is it guaranteed that progressive planning will find a solution if one exists?

Progression planning

- Is it guaranteed that progressive planning will find a solution if one exists?
  - Given that the state-space is finite (ground atoms, no function symbols, finite number of constants)..
  - ..Yes! As long as we visit each state only once..

Progression planning

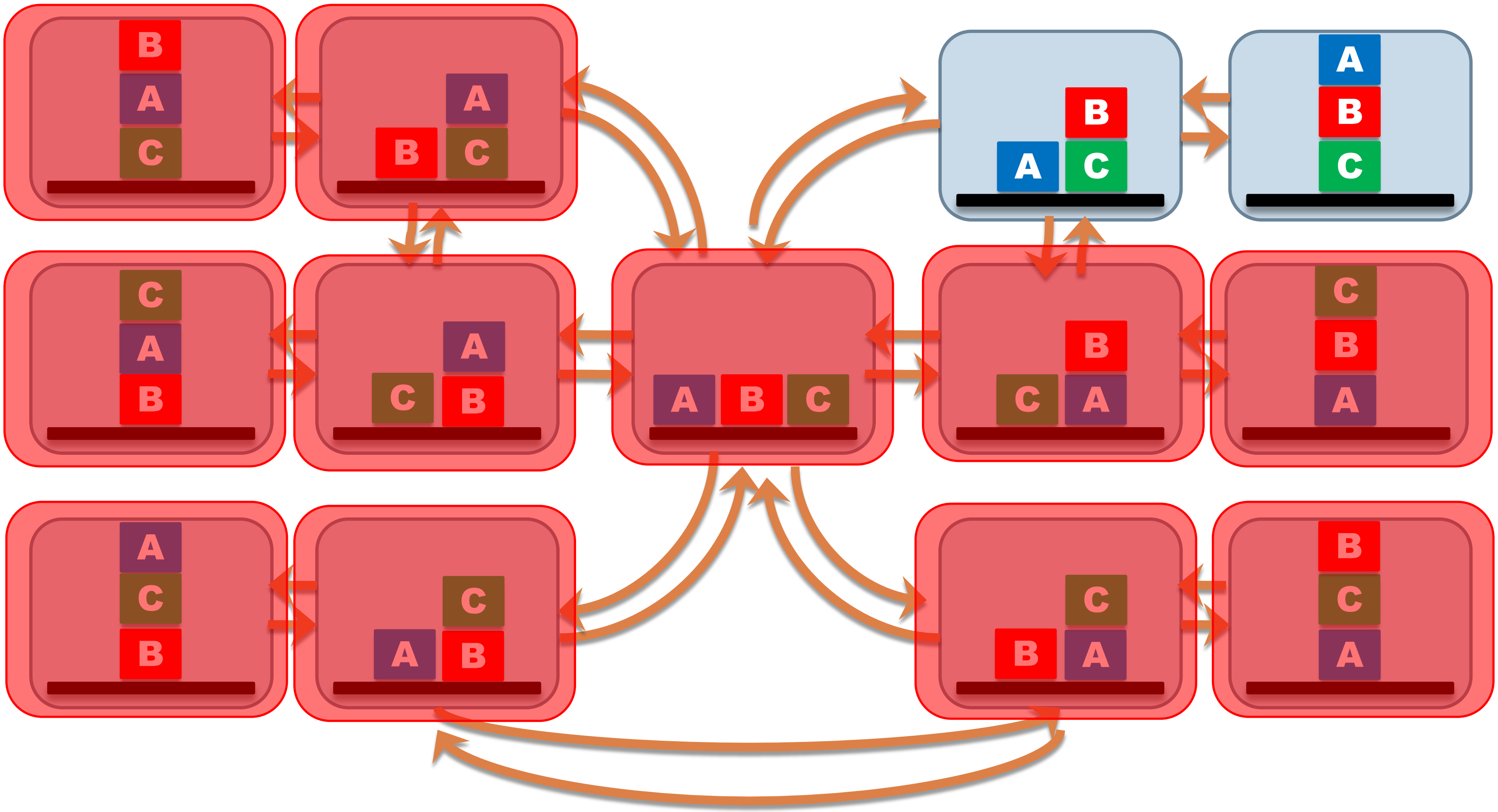
- Pick one of the **not-visited** successor states as current



Progression planning

- Is it guaranteed that progressive planning will find a solution if one exists?
  - Given that the state-space is finite (ground atoms, no function symbols, finite number of constants)..
  - ..Yes! As long as we visit each state only once..
- **But it may have to explore the whole state-space**

Progression planning



Progression planning

- Unlike this simple example, in many problems the state space is actually **huge.**
- Even this simple example becomes challenging if we consider **100 boxes** and **1000s of applicable actions** of the form Move(b,x,y) in each state.

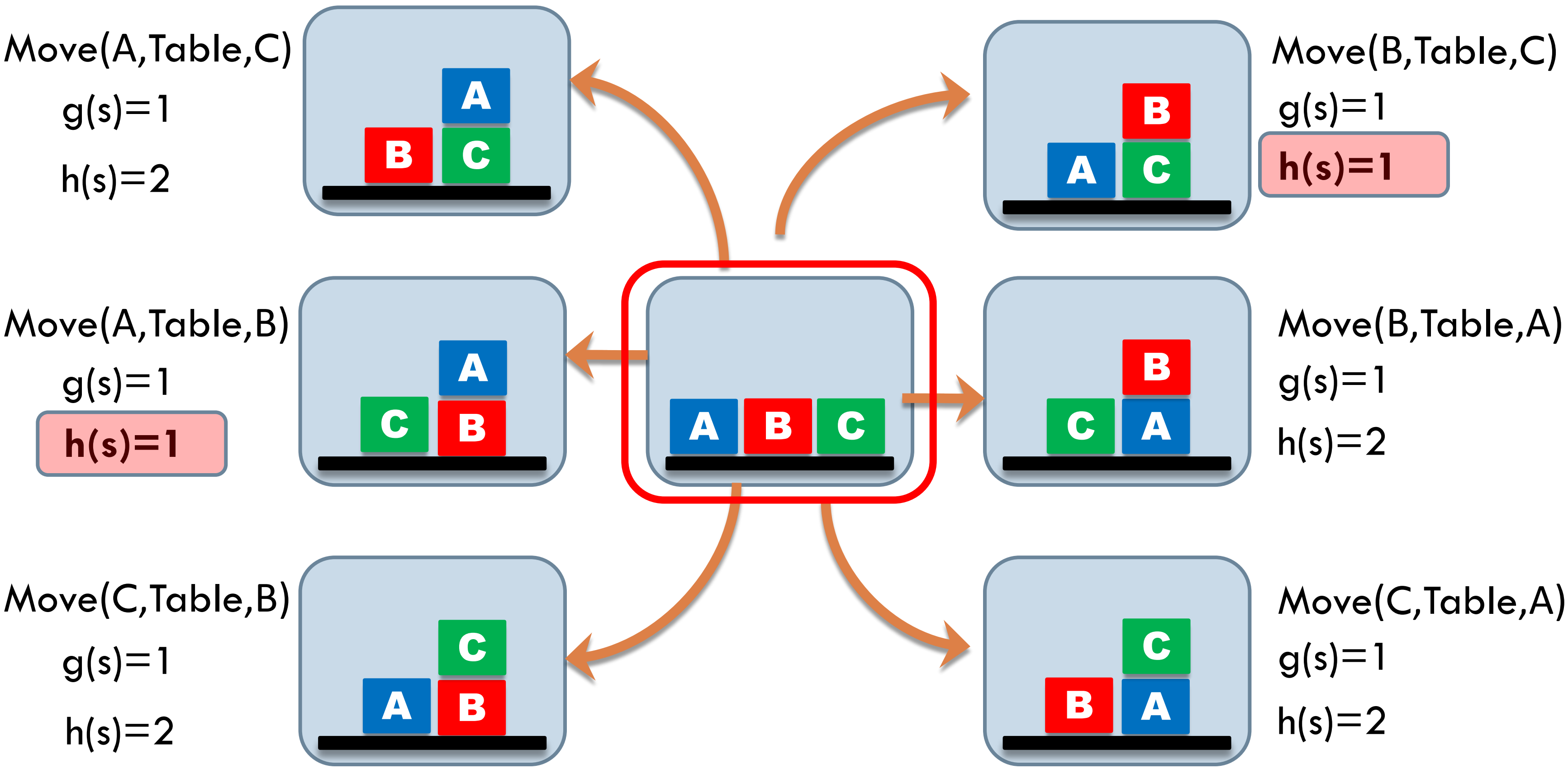
- Similar to search problems we can make use of heuristics that help progression planning pick the most promising states to investigate first.

Heuristics for progression planning

- A* search
- Evaluation function f(s) = g(s) + h(s)
  - g(s): the number of actions needed to reach state s from the initial state (accurate)
  - h(s): the number of actions needed to reach a goal state from state s (estimated)
- Use f(s) to order the successor states and pick the most promising one.

Heuristics for progression planning

- Consider a heuristic h(s) with the following behavior



Heuristics for progression planning

- So why is it different than the usual search problems?
  - E.g., in grid-based problems we could define h(s) using a "relaxed" distance metric such as Manhattan distance.

Heuristics for progression planning

- So why is it different than the usual search problems?
  - E.g., in grid-based problems we could define h(s) using a "relaxed" distance metric such as Manhattan distance.

- The action schemas provide valuable information that can be used to specify **domain independent** heuristic functions!

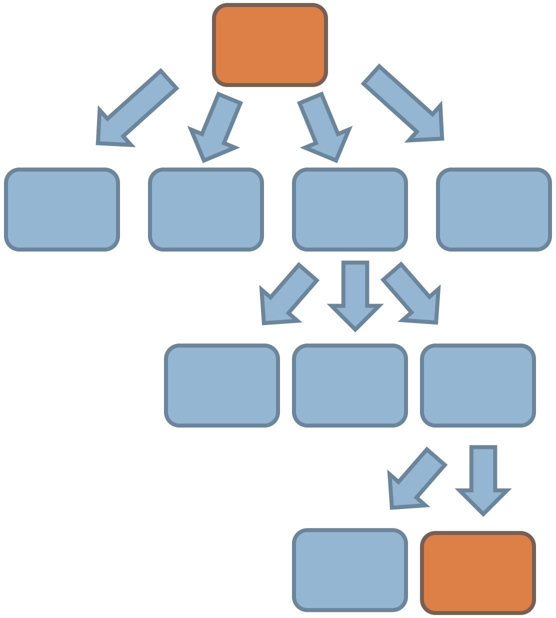
- *(Moreover they provide a logical specification of the problem that allows approaches for finding a solution that are different than search)

Heuristics for progression planning

- Empty list of preconditions
  - h(s) = number of actions needed to achieve the goal if we assume that all actions are always applicable
- Empty list of negative effects
  - h(s) = number of actions needed to achieve the goal if we disregard the negative effects of actions
- Planning graphs
- Simple example: h(s) = number of literals in the goal that are missing from s

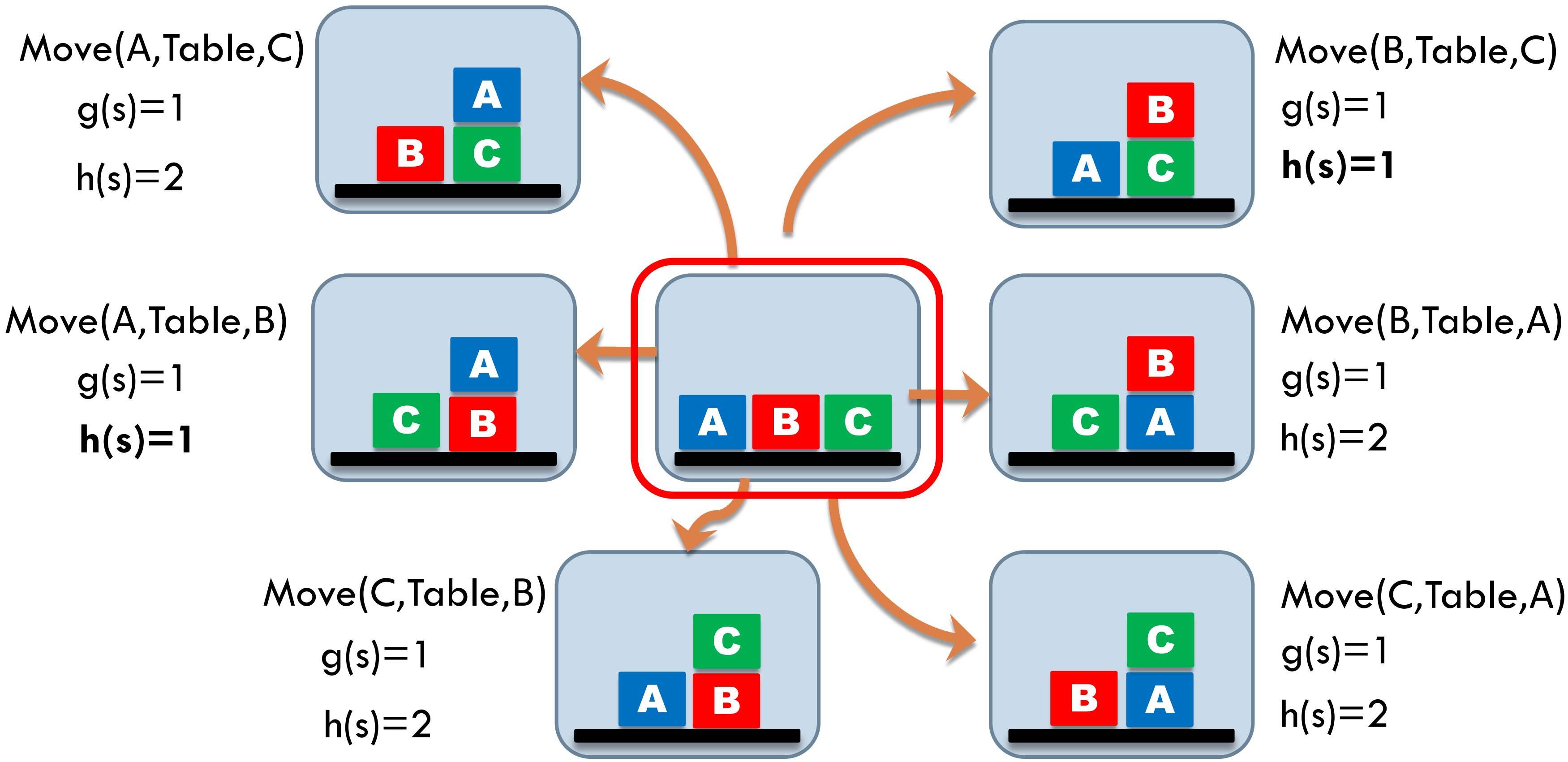
Heuristics for progression planning

- Start from the initial state
- Check if the current state satisfies the goal
- Compute applicable actions to the current state
- Compute the successor states
- Pick ~~one~~ **the most promising** of the successor states as the current state
- Repeat until a solution is found or the state space is exhausted



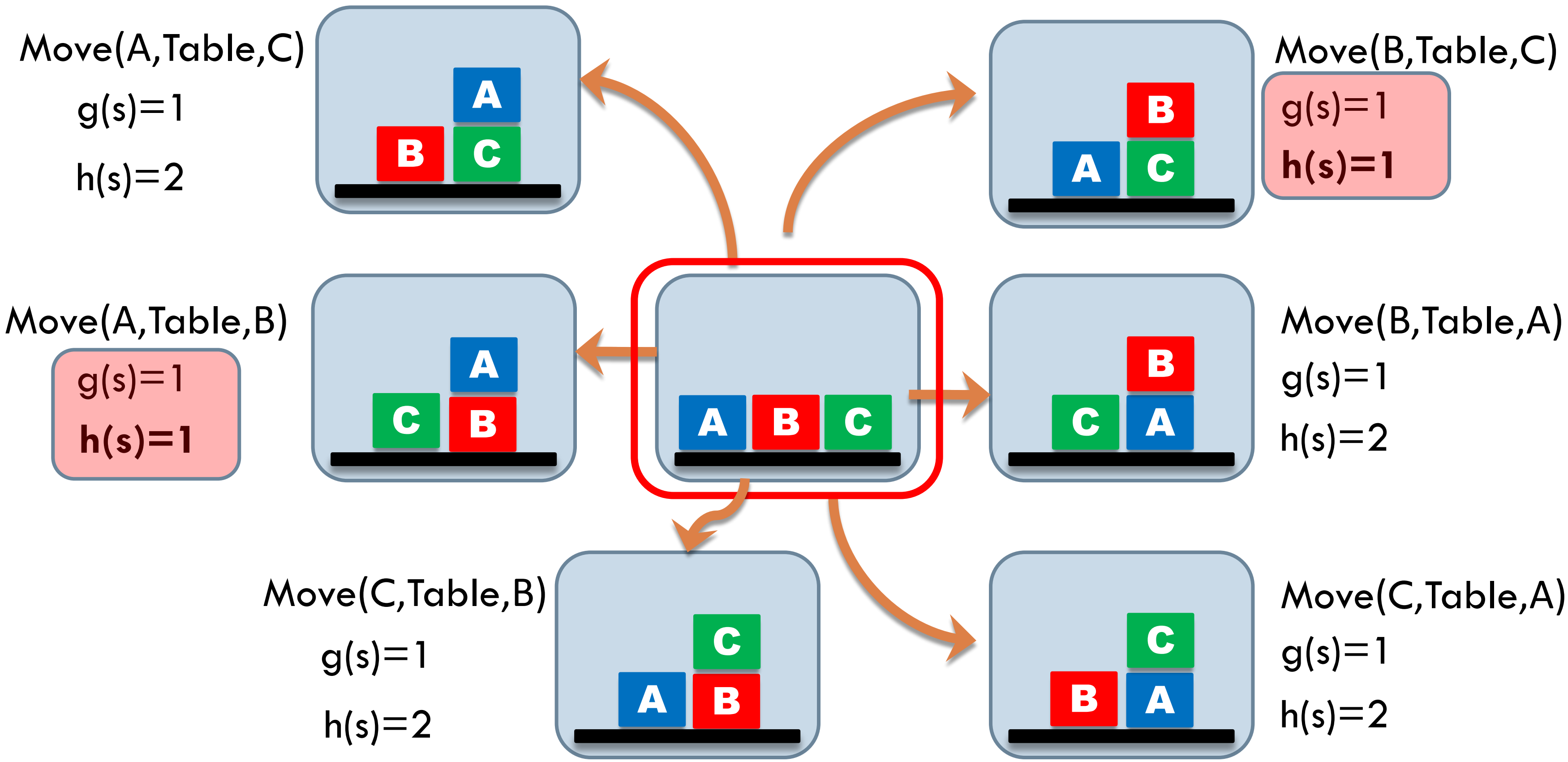
Heuristics for progression planning

- Compute the successor states



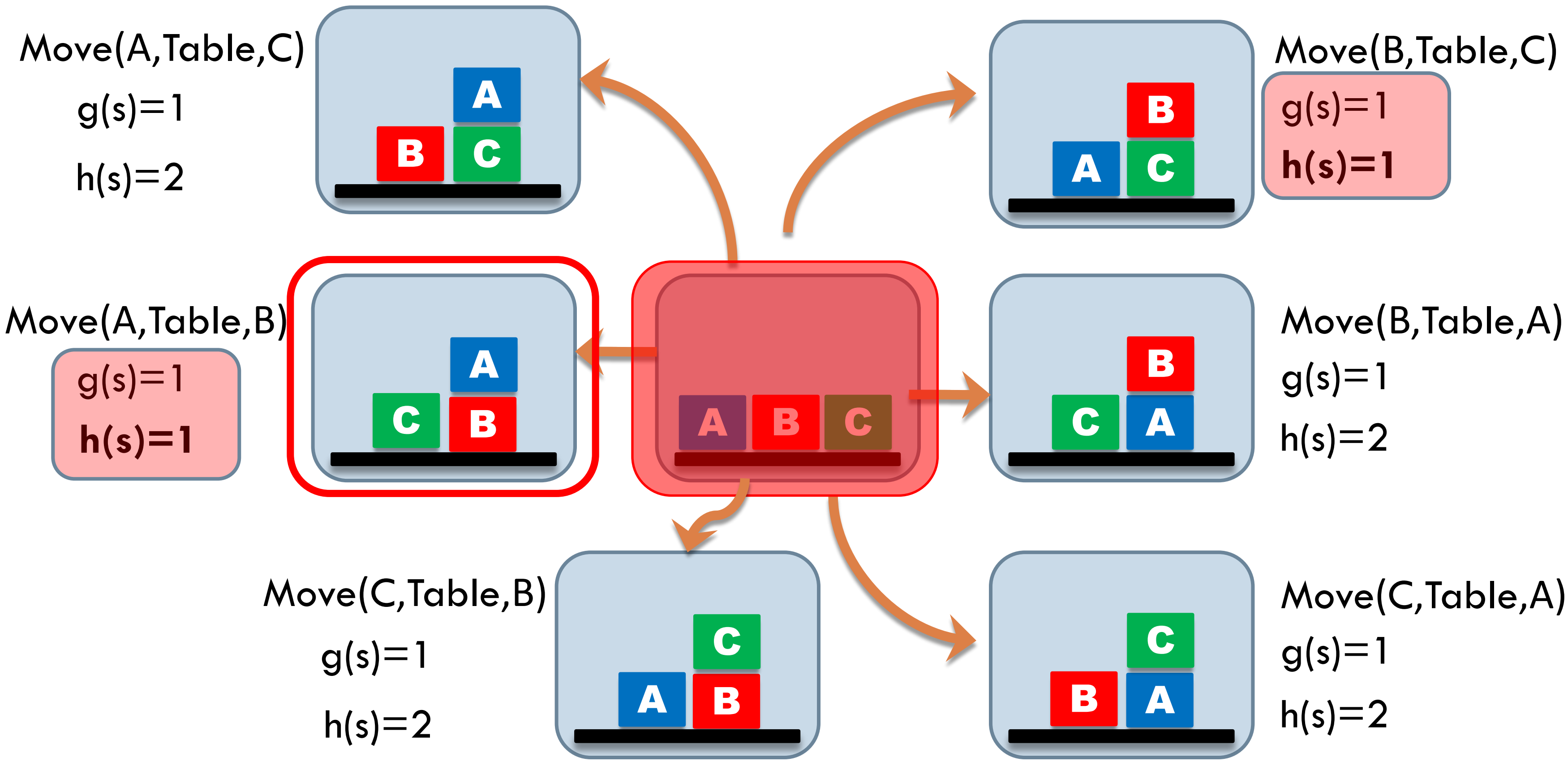
Heuristics for progression planning

- Compute the successor states



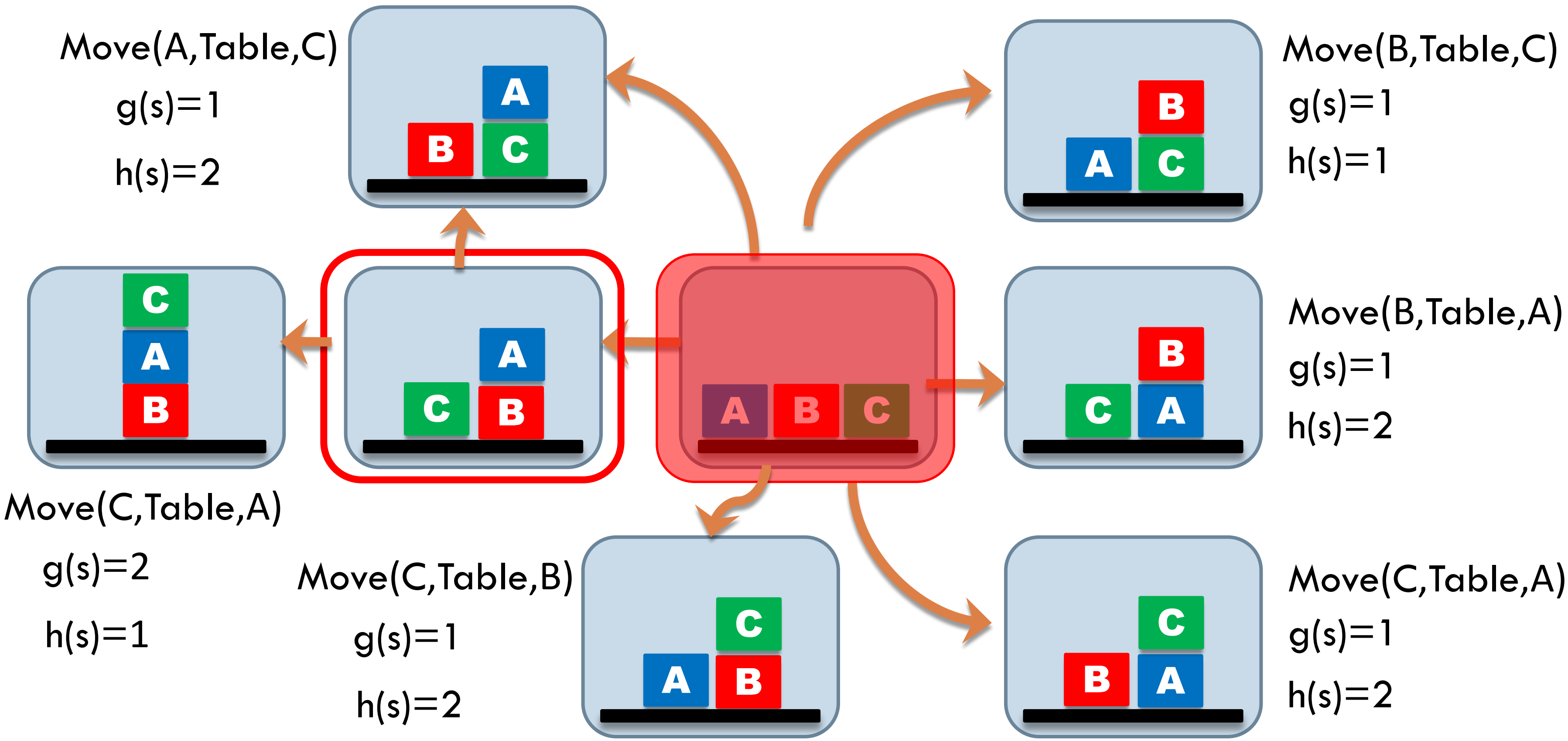
Heuristics for progression planning

- Pick the most promising successor state wrt f(s)



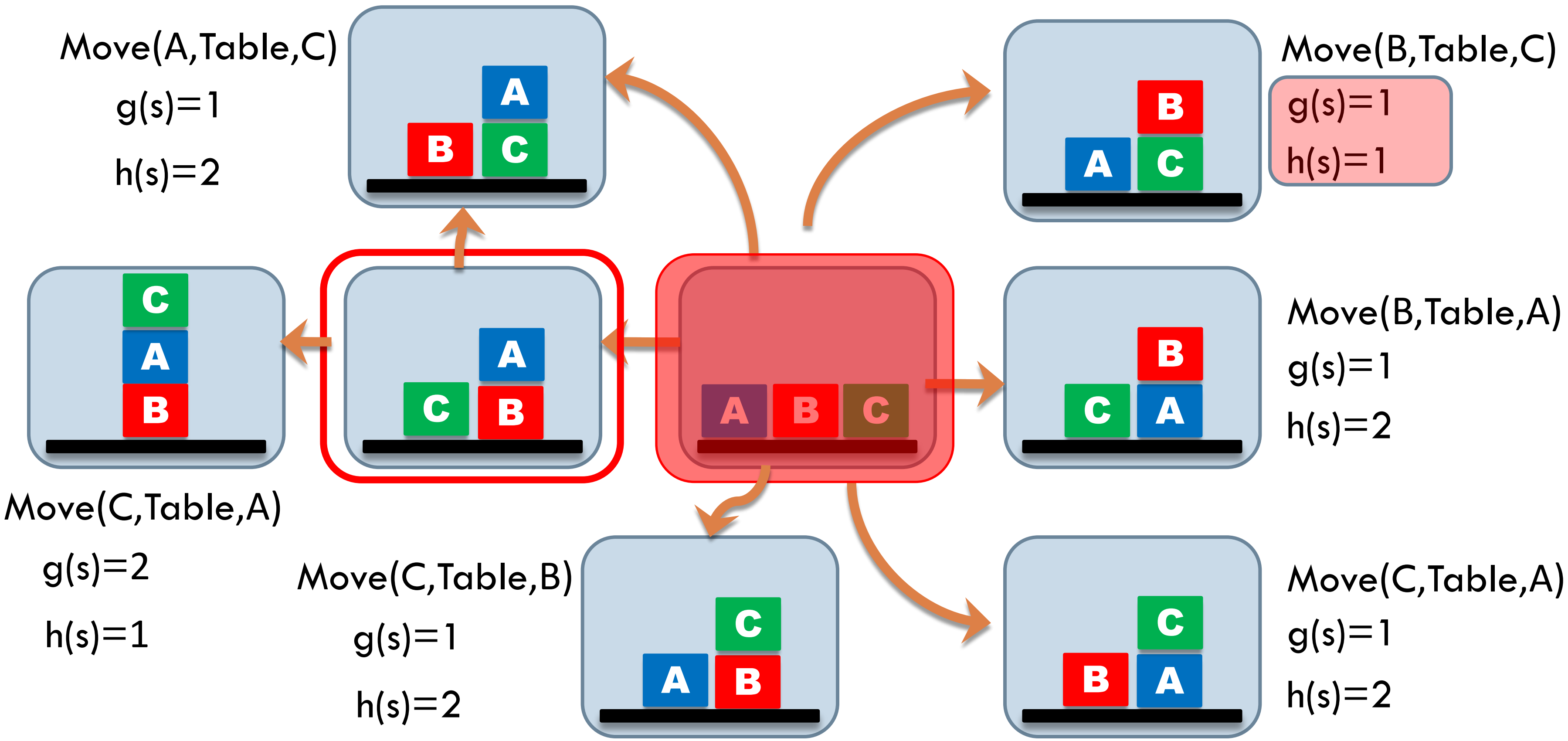
Heuristics for progression planning

- Compute the successor states



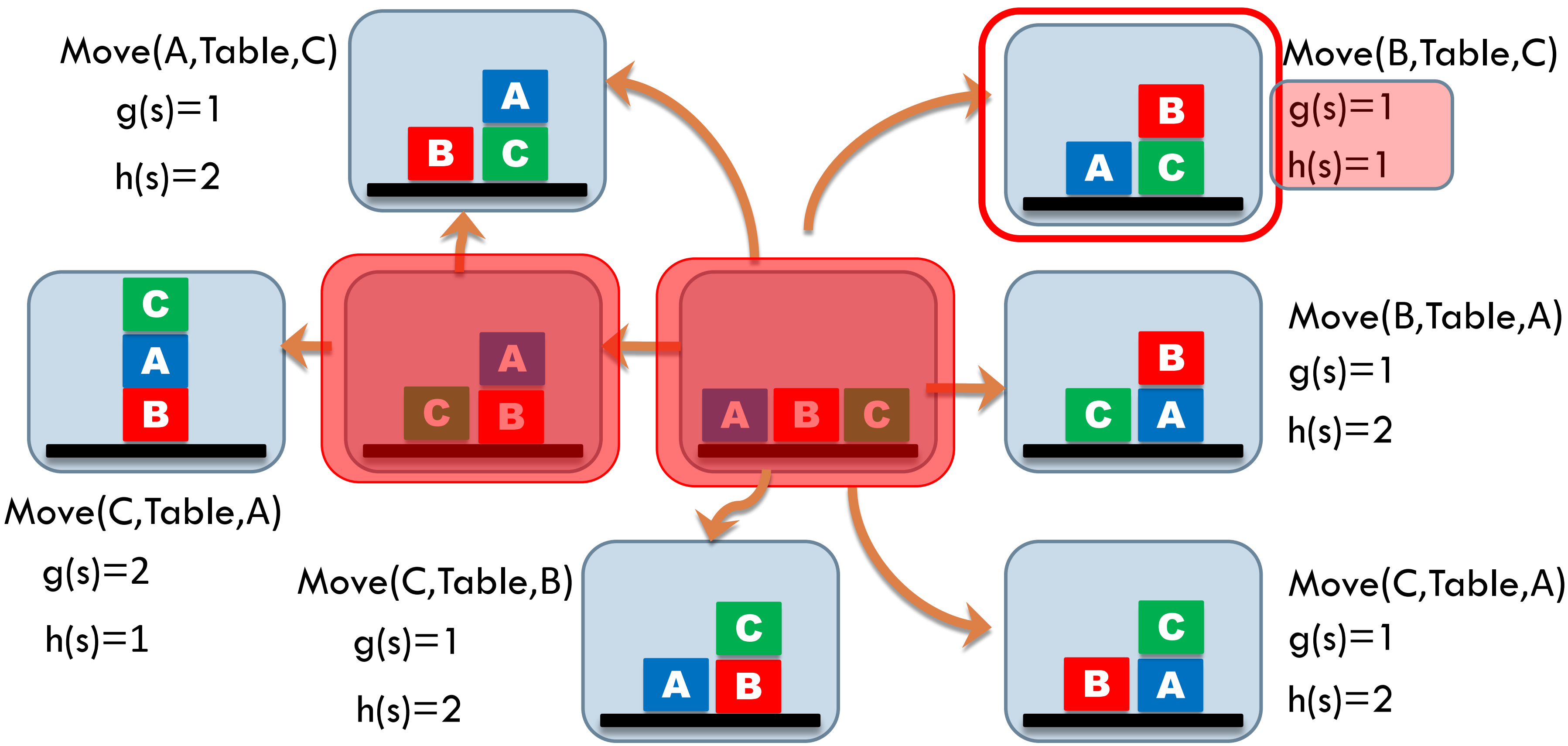
Heuristics for progression planning

- Compute the successor states



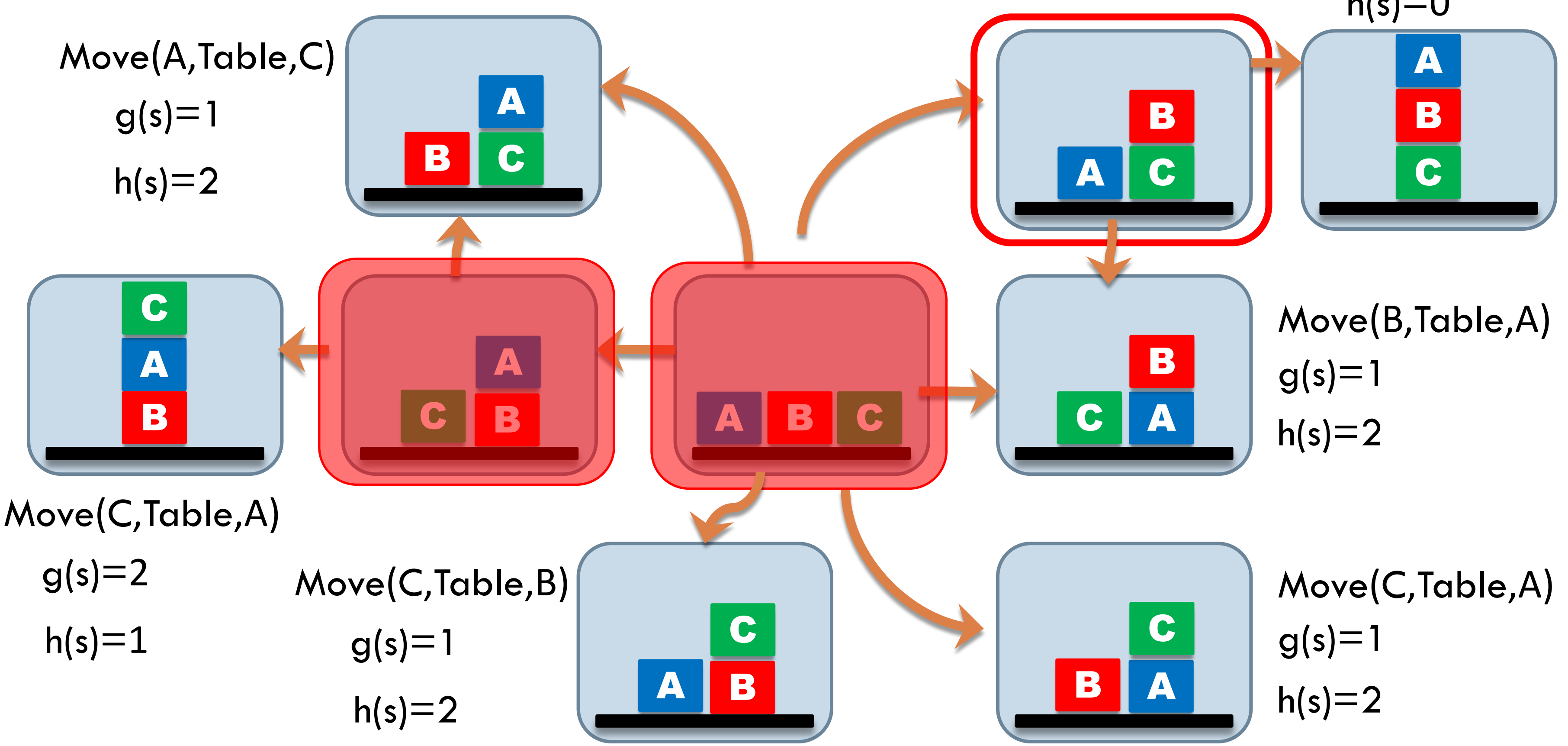
Heuristics for progression planning

- Pick the most promising successor state wrt f(s)



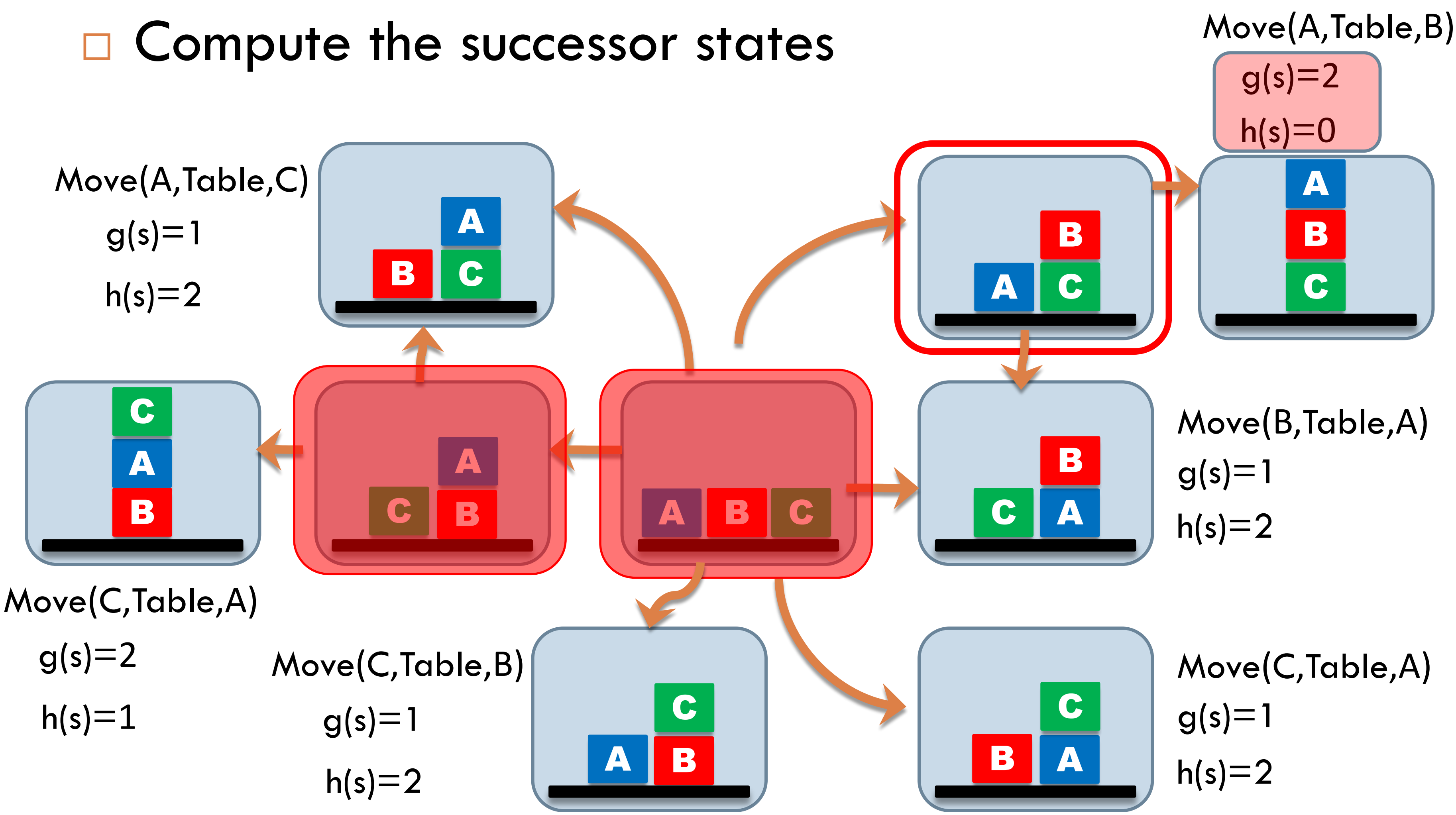
Heuristics for progression planning

- Compute the successor states



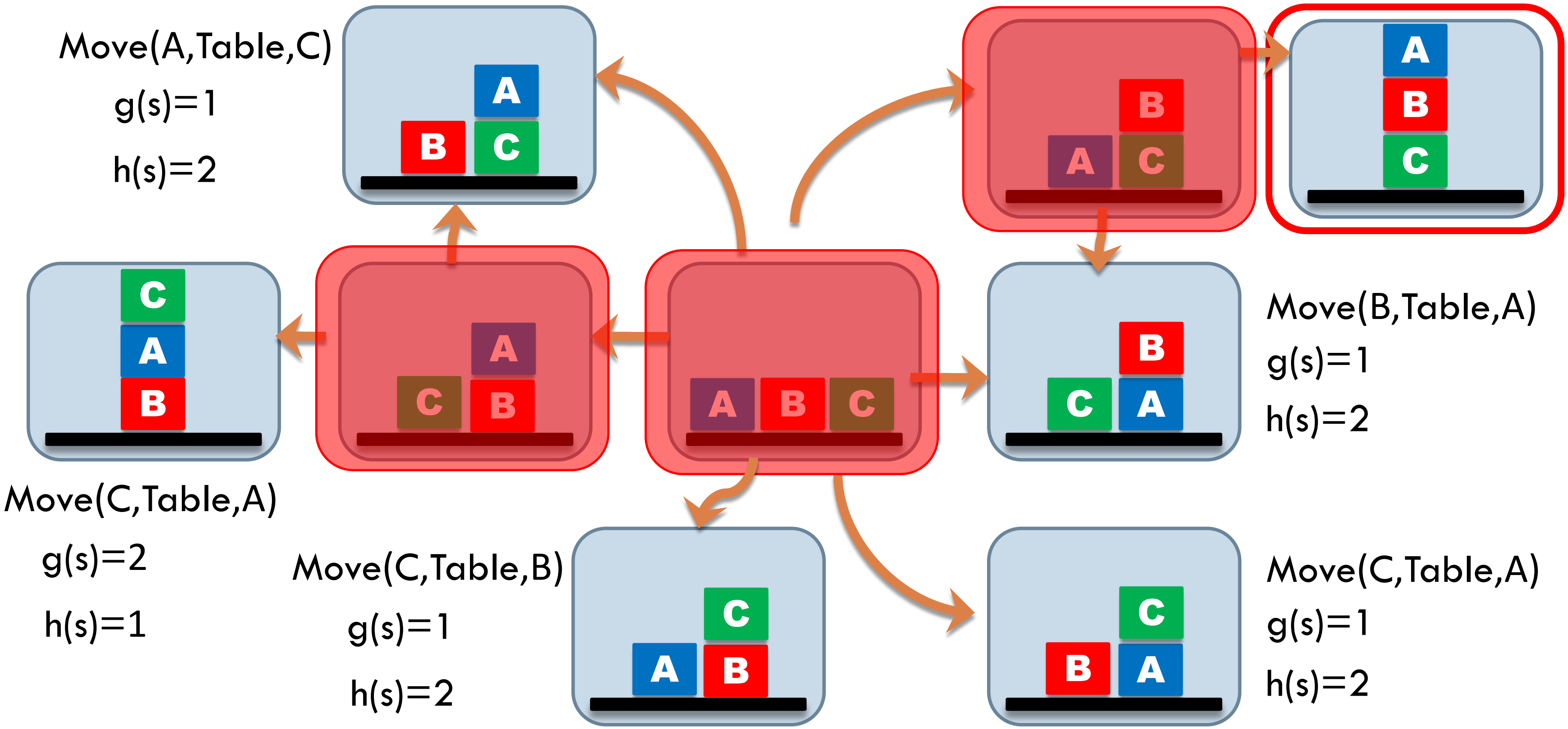
Heuristics for progression planning

- Compute the successor states



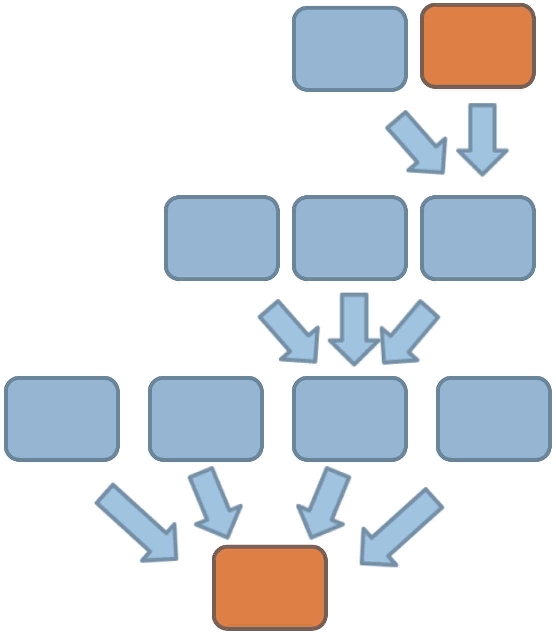
Heuristics for progression planning

- Pick the most promising successor state wrt f(s)



Regression planning

- Start from the **goal** as current goal
- Check if the **initial state** satisfies the current goal
- Compute the **relevant** and **consistent** actions for the current goal
- Compute the **predecessor** states
- Pick one of the **predecessor** states as the current goal
- Repeat until a solution is found or the state space is exhausted



Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - Formal language for specifying planning problems
  - Formal syntax similar to a programming language
  - Includes STRIPS and ADL, and many more features

  - Provides the ground for performing a direct comparison between planning techniques and evaluating against classes of problems

Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - International Planning Competition 1998 – today

- SAT Plan
- TL Plan
- FF
- BlackBox
- SHOP2
- TALPlanner
- …

Planning problems in PDDL, e.g., Blocks world, Storage, Trucks, …

Direct comparison between planning techniques! E.g., evaluation of heuristics, …

Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - We will see more of PDDL and off-the-shelve planners in Lecture 3

Next lecture

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS

Bibliography

- Material
  - Artificial Intelligence: A Modern Approach 2nd Ed. Stuart Russell, Peter Norvig. Prentice Hall, 2003 Section 11.2
- References
  - PDDL - The Planning Domain Definition Language. Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins. Technical report, Yale Center for Computational Vision and Control, TR-98-003, 1998.