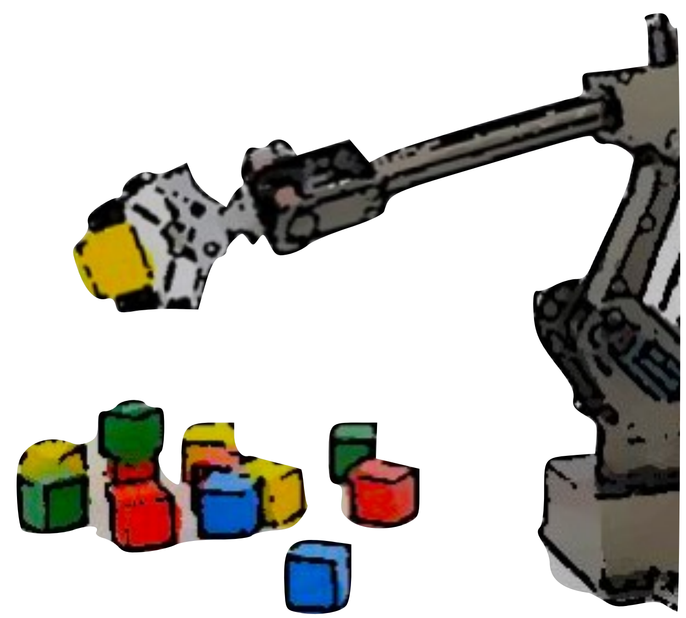
Ve492: Introduction to Artificial Intelligence
Classical Planning

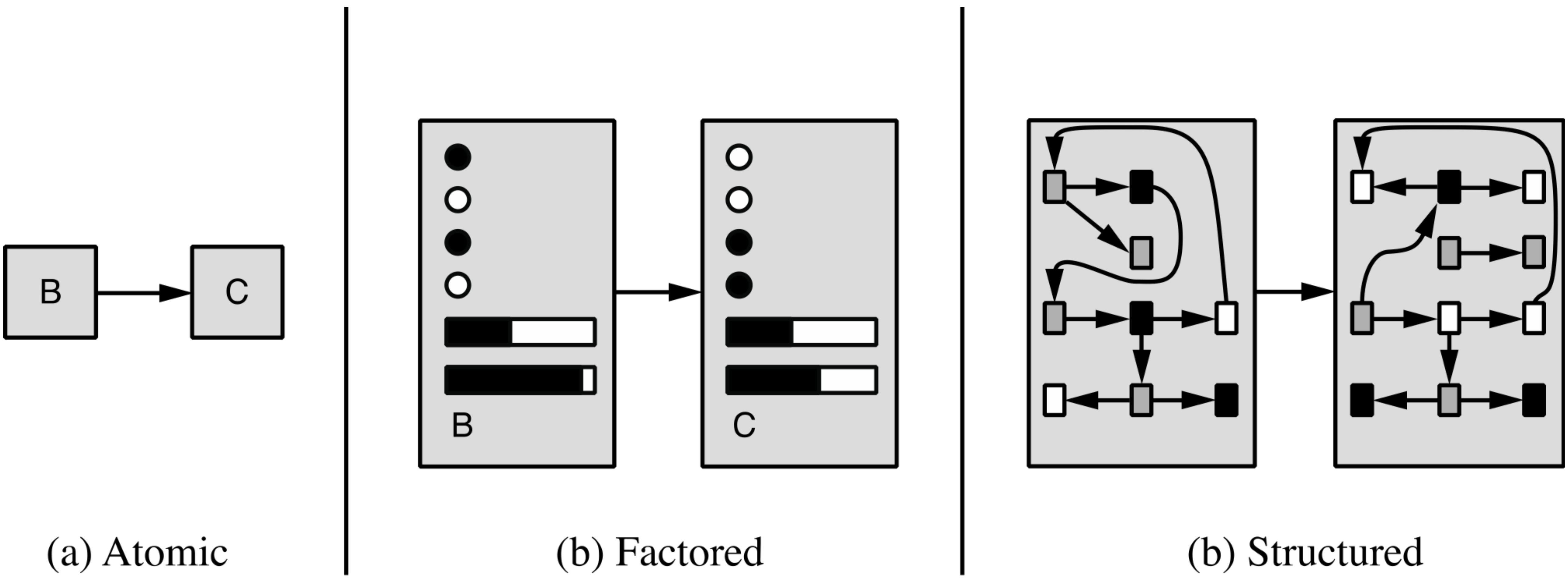


Paul Weng

UM-SJTU Joint Institute

Slides adapted from CMU, AIMA, UM

Spectrum of representations



(a) Atomic

(b) Factored

(b) Structured

Search, game-playing MDP, RL

CSPs, planning, propositional logic, Bayes nets, neural nets, RL with function approx.

First-order logic, databases, probabilistic programs

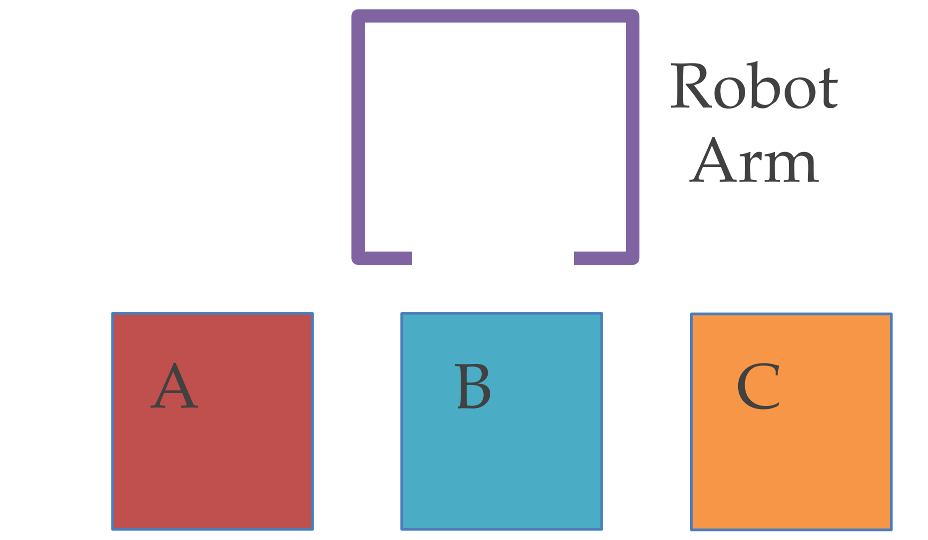
Robot Block Stacking

Start state: A, B, C on table

Goal: Block B on C and C on A

Plan: ?

Modeling Block Stacking States



Start state: A, B, C on table

Goal: Block B on C and C on A

Plan: ?

Goals in the World

Goal States

completely specified

Goal Statements

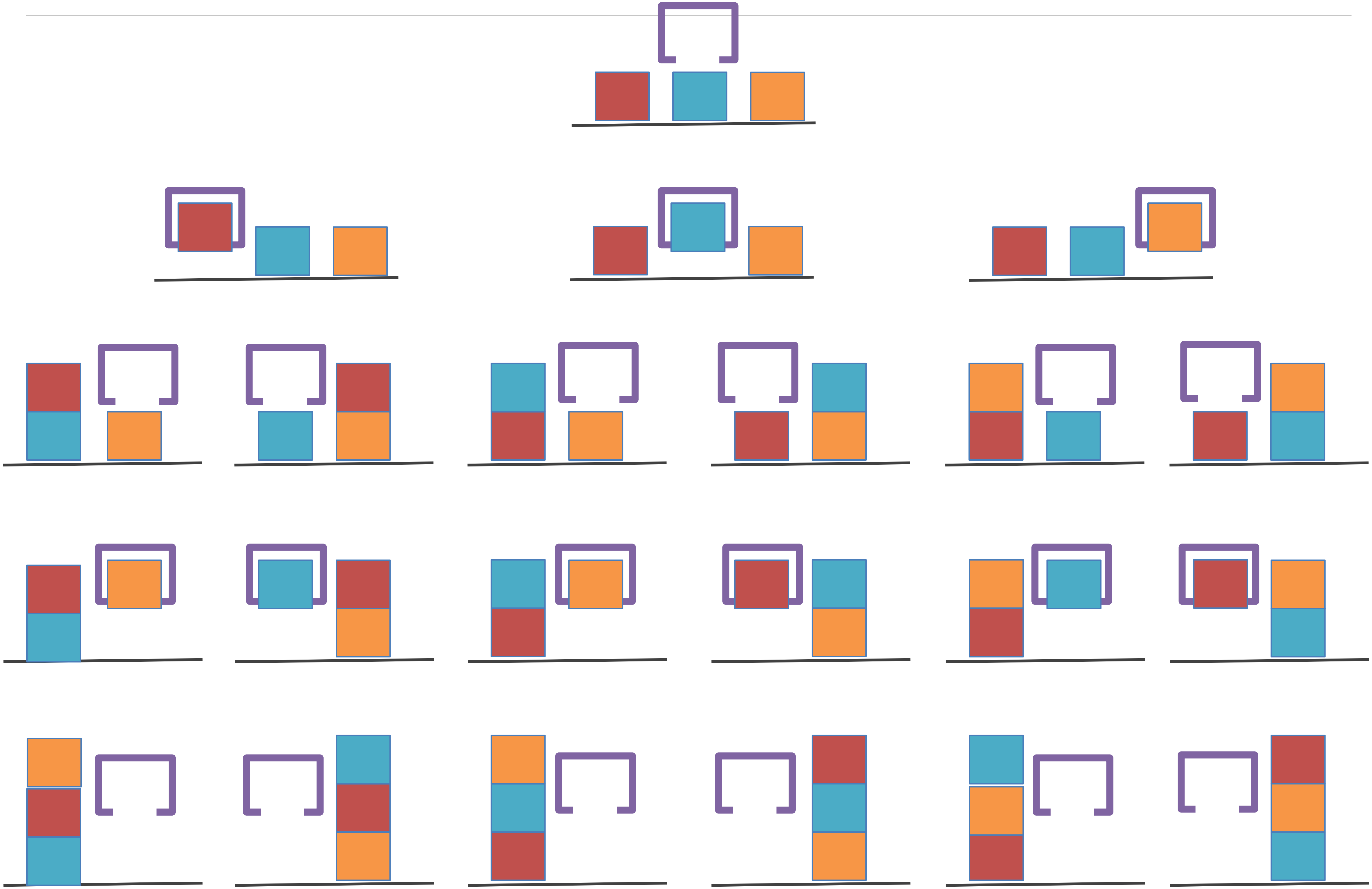
partially specified

Preference models

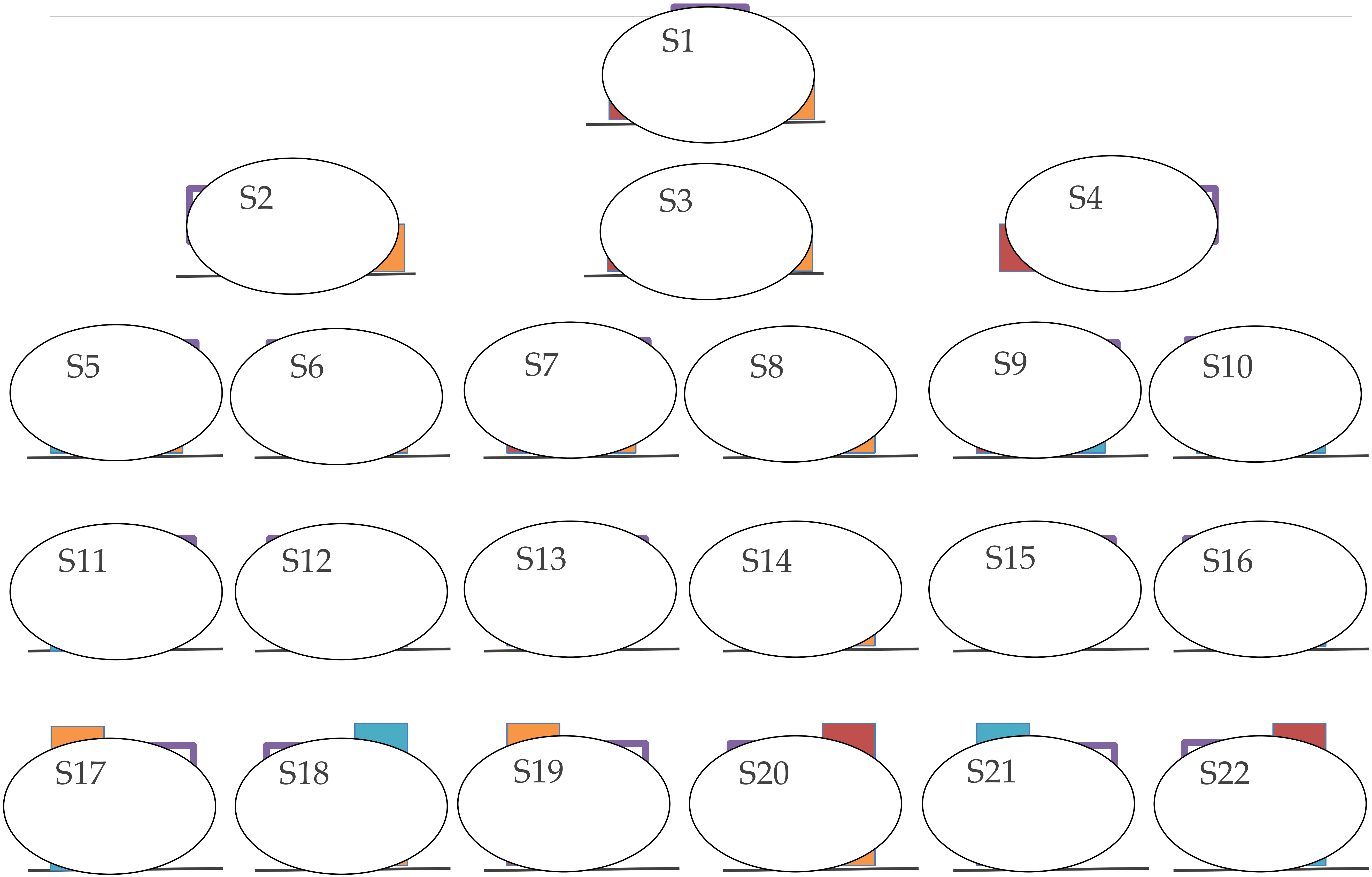
objective function

Increasing Generality

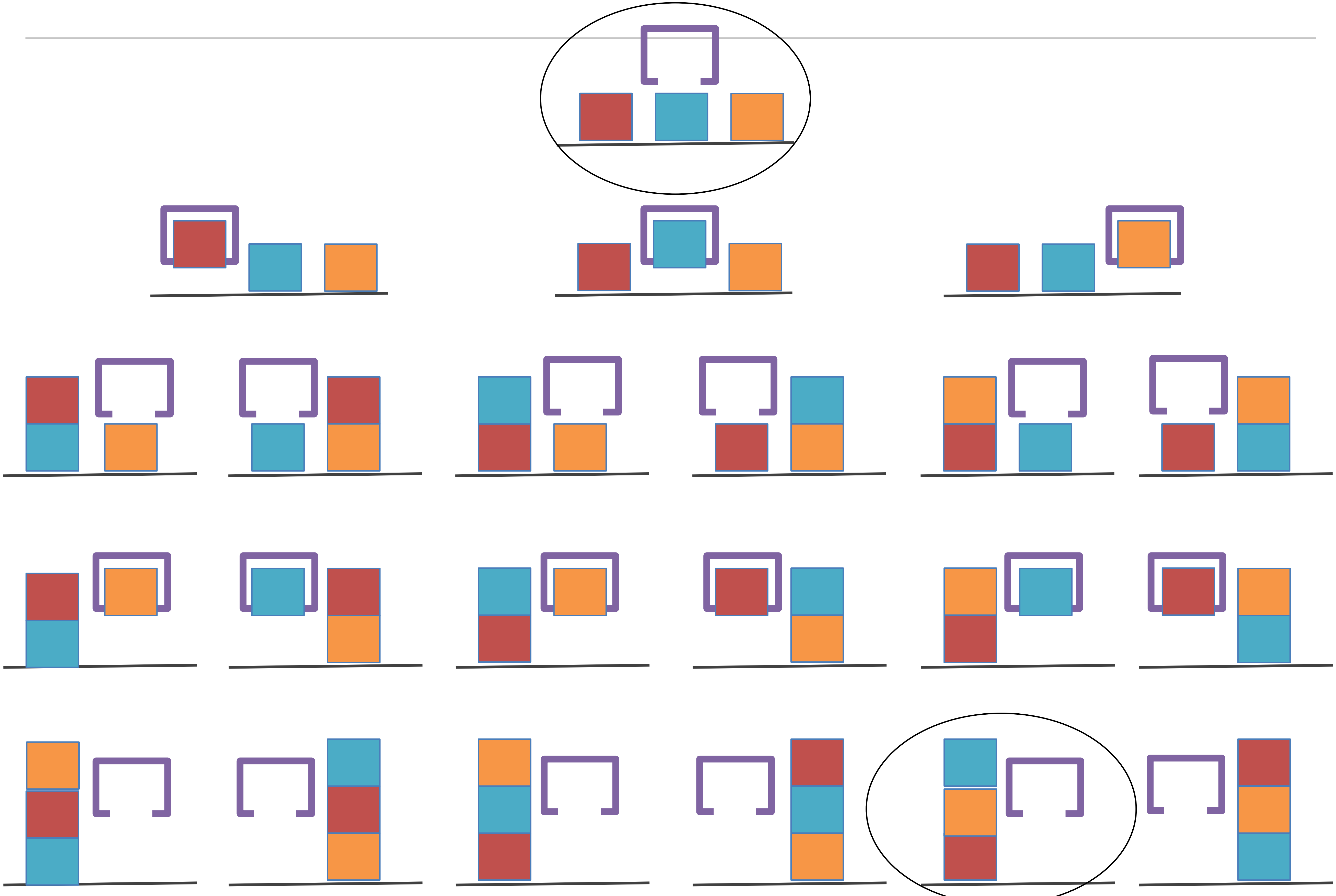
Block Stacking States



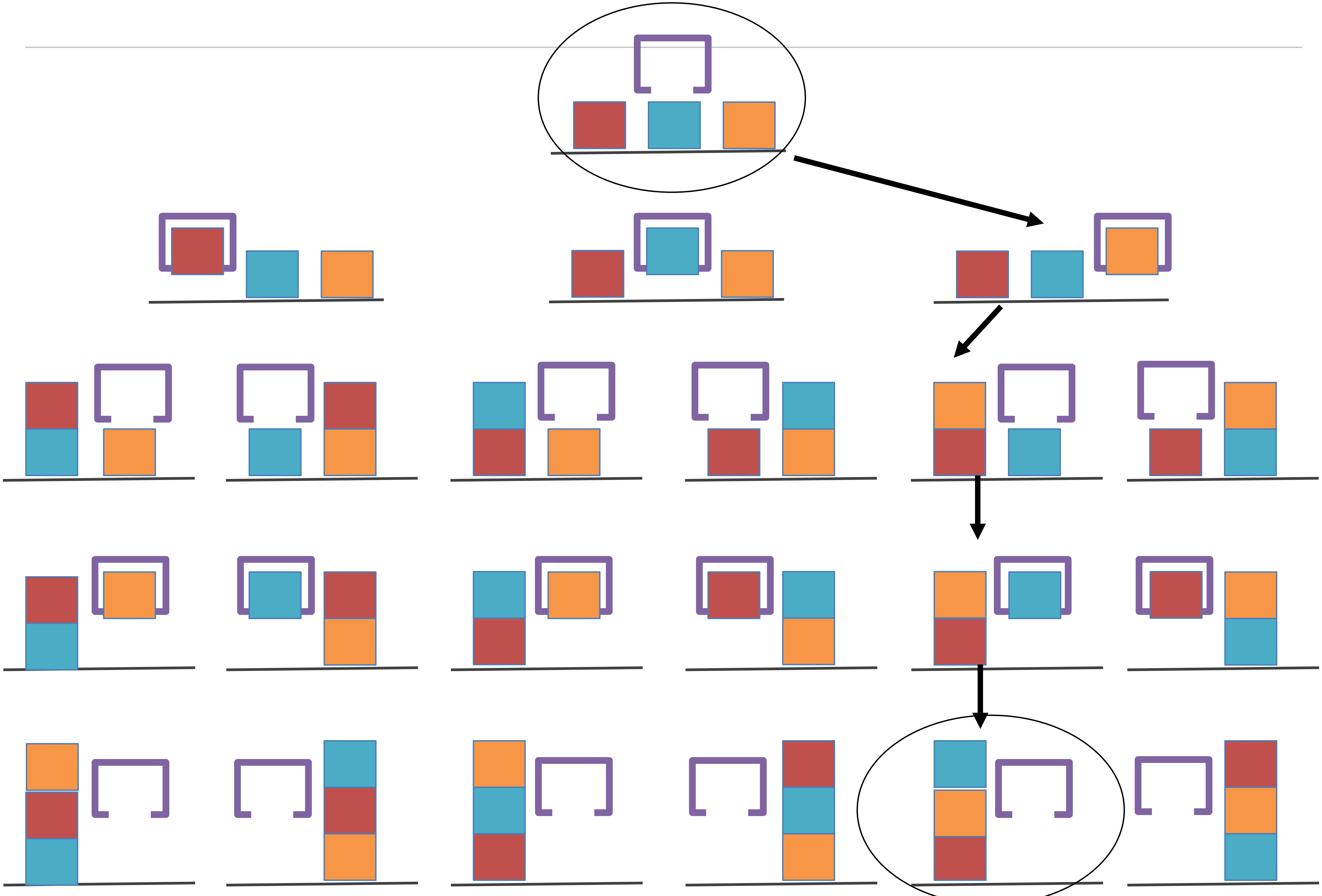
States are Atomic



Initial and Goal States



Plan from Initial to Goal State



Goals in the World

Goal States

completely specified

Goal Statements

partially specified

Preference models

objective function

Increasing Generality

BFS, DFS, A*

Goals in the World

Goal States

completely specified

Goal Statements

partially specified

Preference models

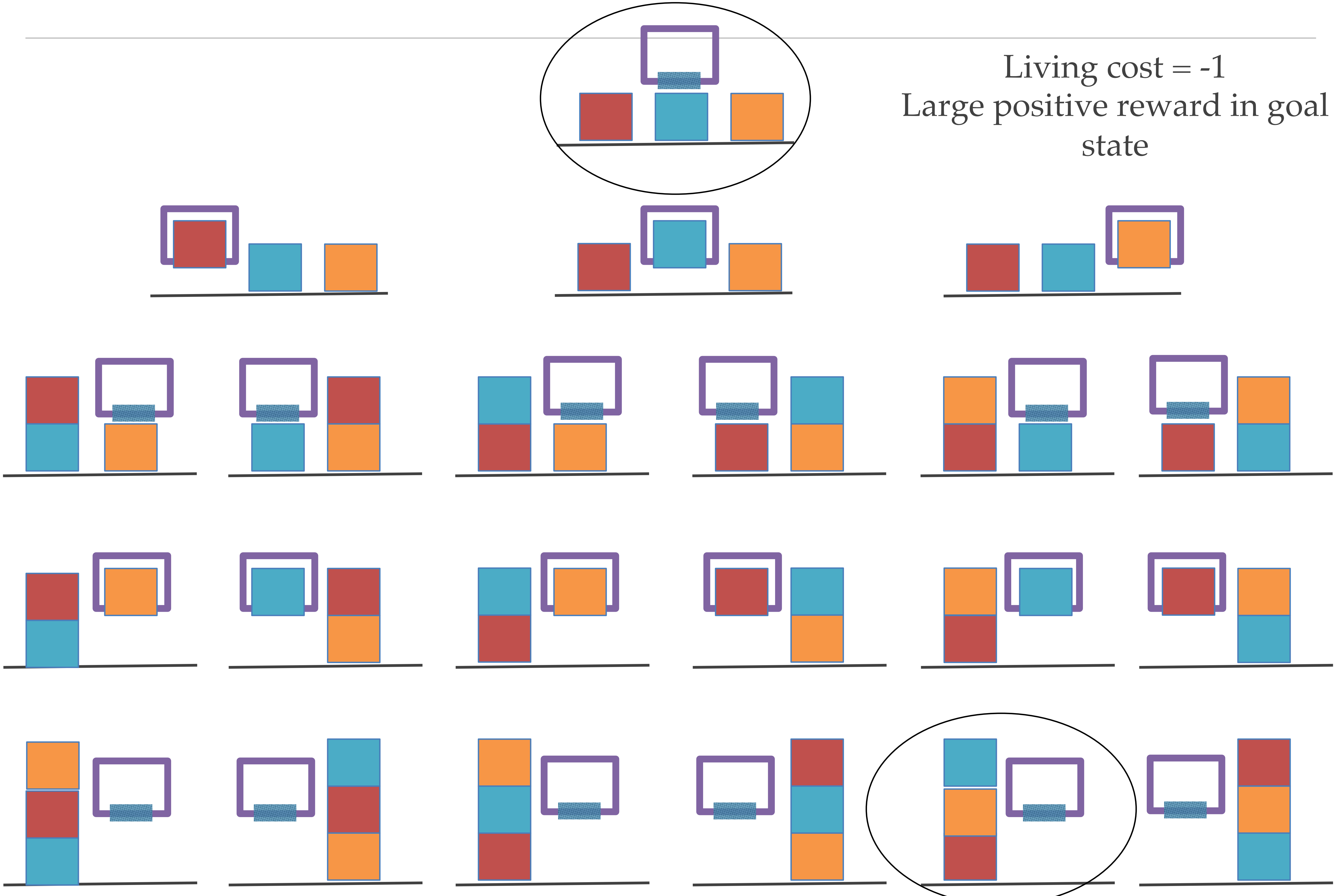
objective function

Increasing Generality

BFS, DFS, A*

Reward Function

Living cost = -1
Large positive reward in goal state



Goals in the World

Goal States

completely specified

Goal Statements

partially specified

Preference models

objective function

Increasing Generality

BFS, DFS, A*

A*, MDP, RL

Goals in the World

Goal States

completely specified

Goal Statements

partially specified

Preference models

objective function

Increasing Generality

BFS, DFS, A*

?

A*, MDP, RL

Goals in the World

Goal States
completely specified

Goal Statements
partially specified

Preference models
objective function

Increasing Generality

BFS, DFS, A*

Logic, CSP

A*, MDP, RL

Logical Agents

Create a Knowledge Base (KB)

- Symbols – each is true or false

TELL KB

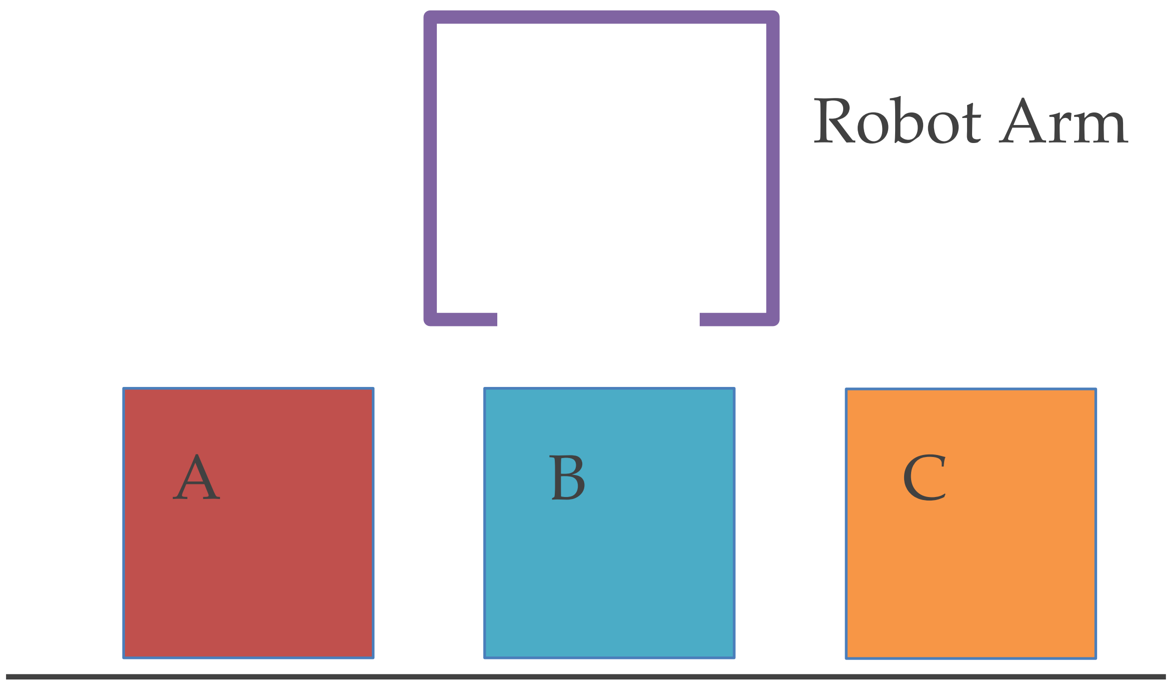
- Initial state and a priori knowledge
- Domain knowledge and "Physics" of the domain

Logical Agents with PL

Create a Knowledge Base

Symbols

A-on-Table
B-on-Table
C-on-Table
Hand-Empty
A-on-B
A-on-C
B-on-C
A-In-Hand
B-In-Hand
C-In-Hand
B-on-A
C-on-A
C-on-B
Pick_A, Pick_B, Pick_C
Put_on_Table_A, Put_on_Table_B, Put_on_Table_C
Anything missing?

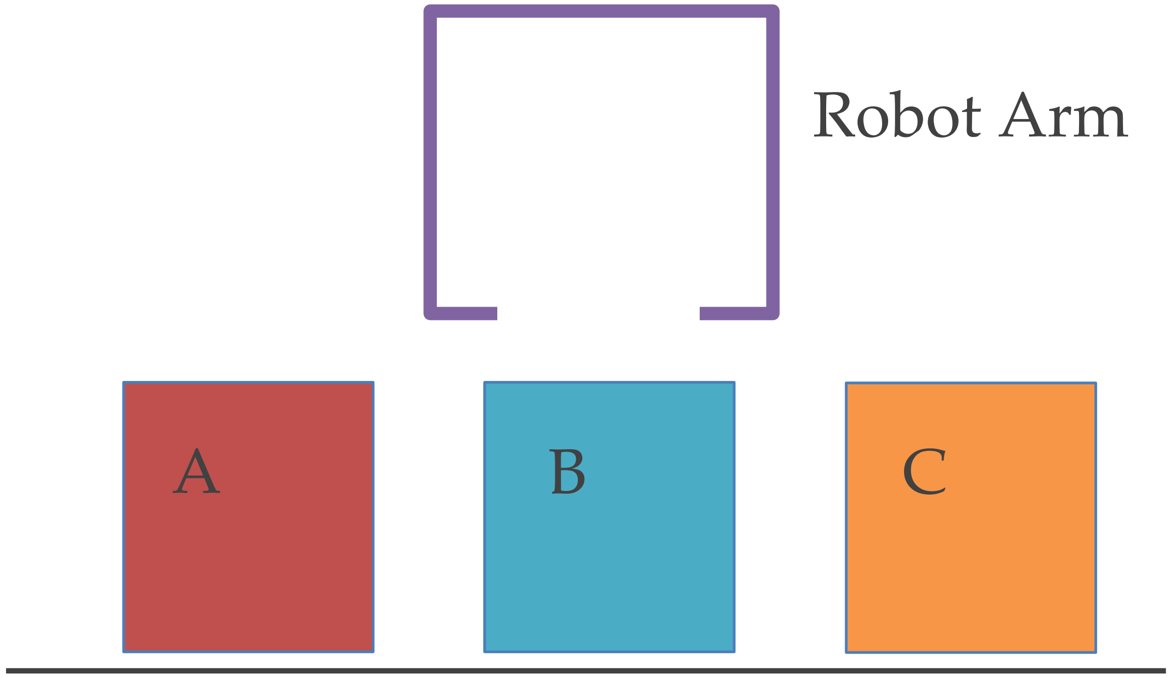


Logical Agents with PL

Create a Knowledge Base

Symbols

A-on-Table[t]   A-In-Hand[t]
B-on-Table[t]   B-In-Hand[t]
C-on-Table[t]   C-In-Hand[t]
Hand-Empty[t]
A-on-B[t]   B-on-A[t]
A-on-C[t]   C-on-A[t]
B-on-C[t]   C-on-B[t]
Pick_A[t], Pick_B[t], Pick_C[t]
Put_on_Table_A[t], Put_on_Table_B[t], Put_on_Table_C[t]
…



Logical Agents with PL

Create a Knowledge Base

Symbols

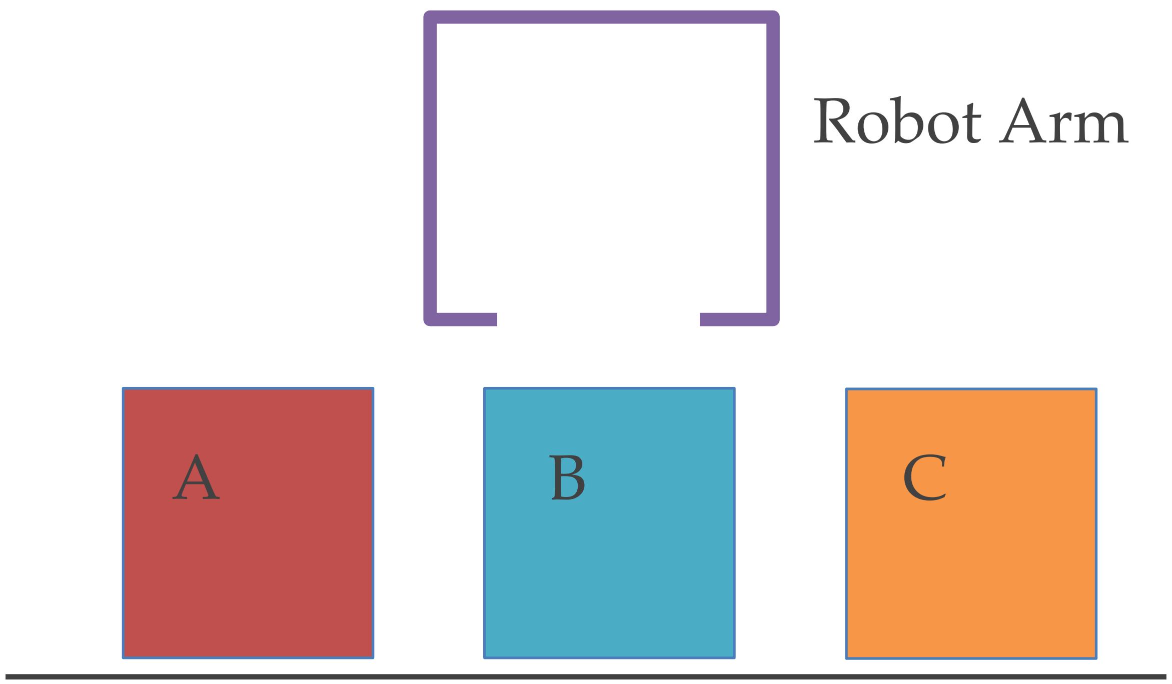
A-on-Table[0]
B-on-Table[0]
C-on-Table[0]
Hand-Empty[0]

A-In-Hand[t]
B-In-Hand[t]
C-In-Hand[t]

A-on-B[t]
A-on-C[t]
B-on-C[t]

B-on-A[t]
C-on-A[t]
C-on-B[t]

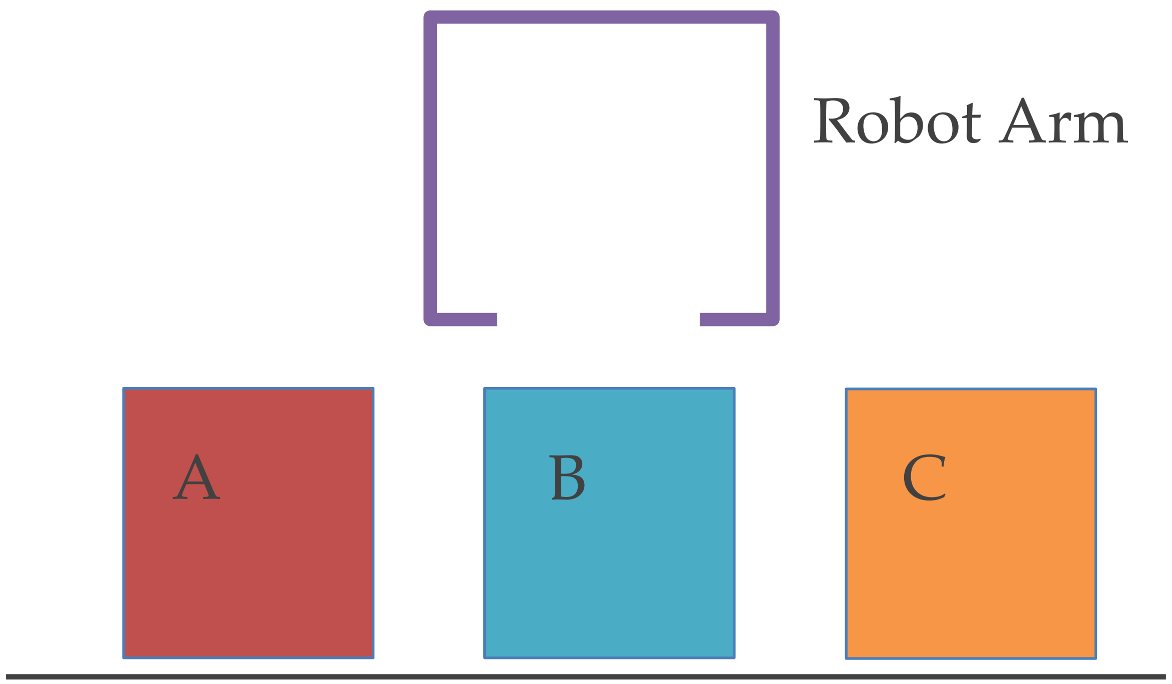
Pick_A[t], Pick_B[t], Pick_C[t]
Put_on_Table_A[t], Put_on_Table_B[t], Put_on_Table_C[t]
…



Logical Agents with PL

Create a Knowledge Base
Implications

When is A-on-Table[t] true or false?
Use successor-state axioms!



Logical Agents

Create a Knowledge Base
Implications

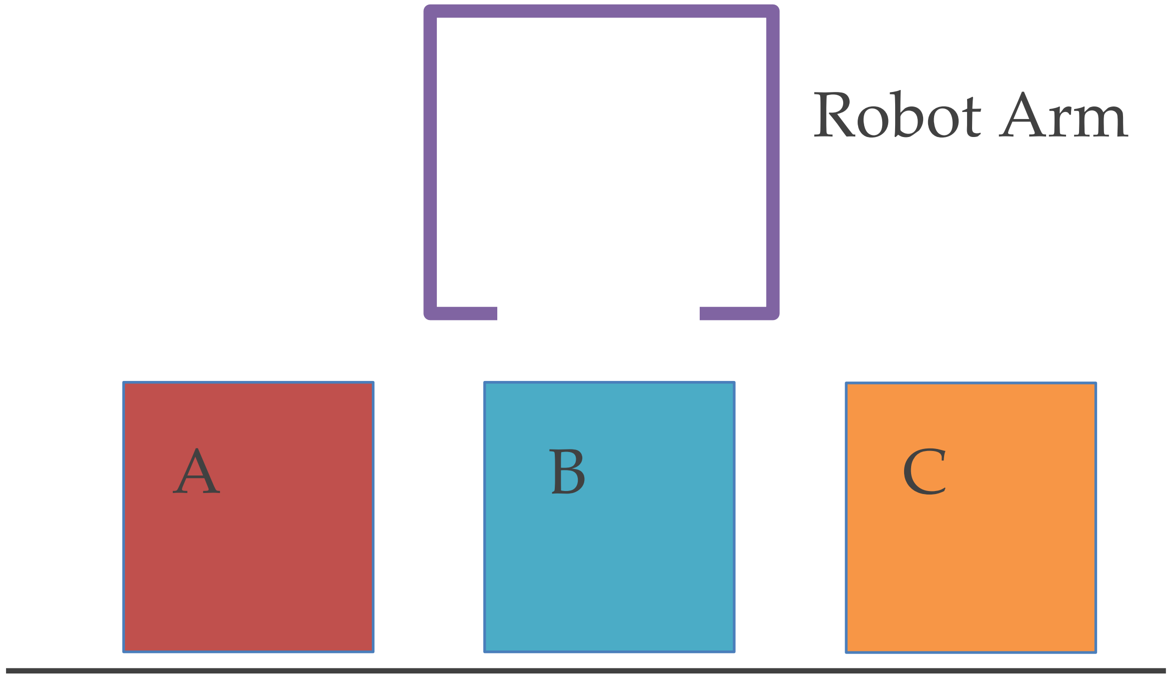
A-on-Table[t]

$\Leftrightarrow$

(A-on-Table[t-1] $\wedge \neg$ Pick_A[t-1])

$\vee$

(A-in-Hand[t-1] $\wedge$ Put-on-Table[t-1])



Logical Agents

Create a Knowledge Base (KB)

Symbols – each is true or false

TELL KB

Initial state and a priori knowledge

Domain knowledge and “Physics” of the domain

ASK whether KB entails a query

e.g., B-on-C[t] $\wedge$ C-on-A[t]?

Partially-Specified Goal

We didn't specify what all goal symbols needed to be, only some

There are potentially many possible world (i.e., goal states) that could satisfy this goal

We only need to search for one satisfying assignment of variables

Challenges of Logic Planning

We need symbols for each time step (even with FOL)

A state (i.e., model or possible world) is fully-specified by the list of all literals that are true in this model

Actions (e.g., picking a block) are represented with successor-state axioms

Easy to incorrectly specify an axiom (e.g., Hand-Empty[t])

So many symbols means it is hard to debug

Classical Planning (with STRIPS)

Also partially-specified goals

Create a Knowledge Base
- Using STRIPS language
  - Predicates for describing states
  - Operators for describing actions

Using KB, find plan to reach any goal state
- Goal = conjunction of predicates

Predicates

Propositional Logic

A-on-Table, A-In-Hand,
B-on-Table, B-In-Hand,
C-on-Table, C-In-Hand,
Hand-Empty,
A-on-B, B-on-A,
A-on-C, C-on-A,
B-on-C, C-on-B
Clear-A, Clear-B,
Clear-C
…

STRIPS

Constants: A, B, C
Predicates:
- In-Hand(A)
- On-Table(B)
- On-Block(B,C)
- HandEmpty()
- Clear(A)
- …

STRIPS

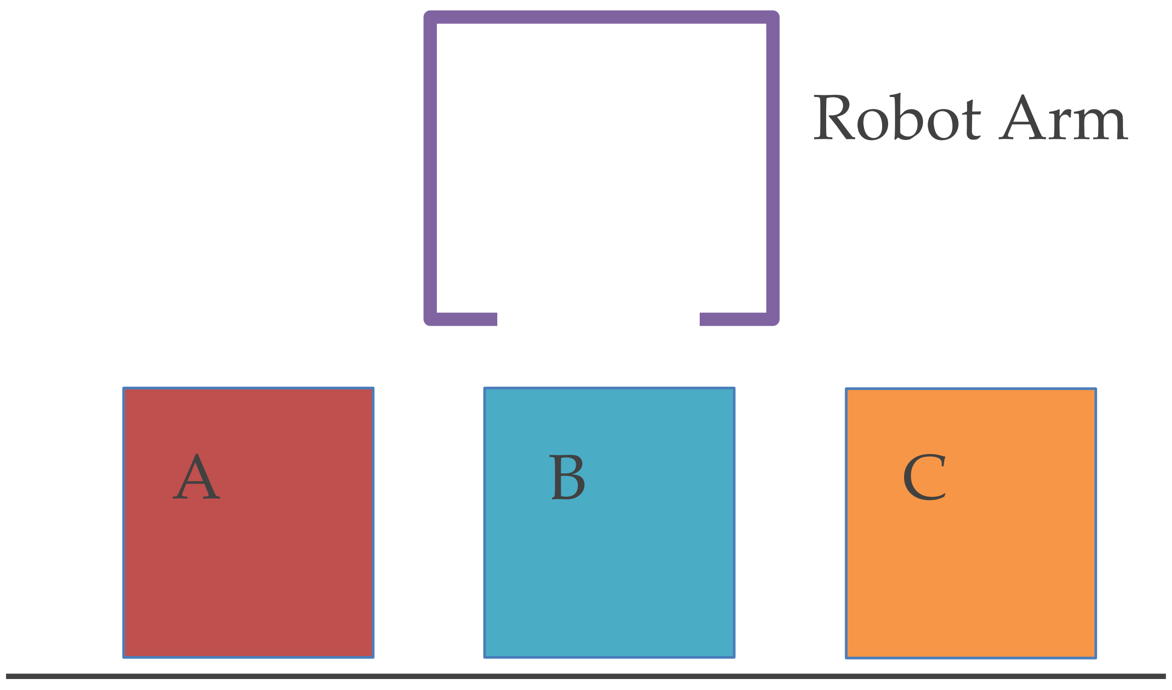
- STRIPS inspired by FOL
- In STRIPS
  - No functions!
  - States are represented by conjunctions of positive predicates
    - If a predicate doesn't appear in this conjunction, it is false
  - Predicates don't depend on time
  - Actions don't need any predicates, they are represented as operators
- Trade-off between expressivity of language and efficiency of solving algorithms

How to Describe this State?

Instances: A, B, C

Propositions:

1) In-Hand(A)
2) In-Hand(B)
3) In-Hand(C)
4) On-Table(A)
5) On-Table(B)
6) On-Table(C)
7) On-Block(B,C)
8) On-Block(A,B)
9) HandEmpty()

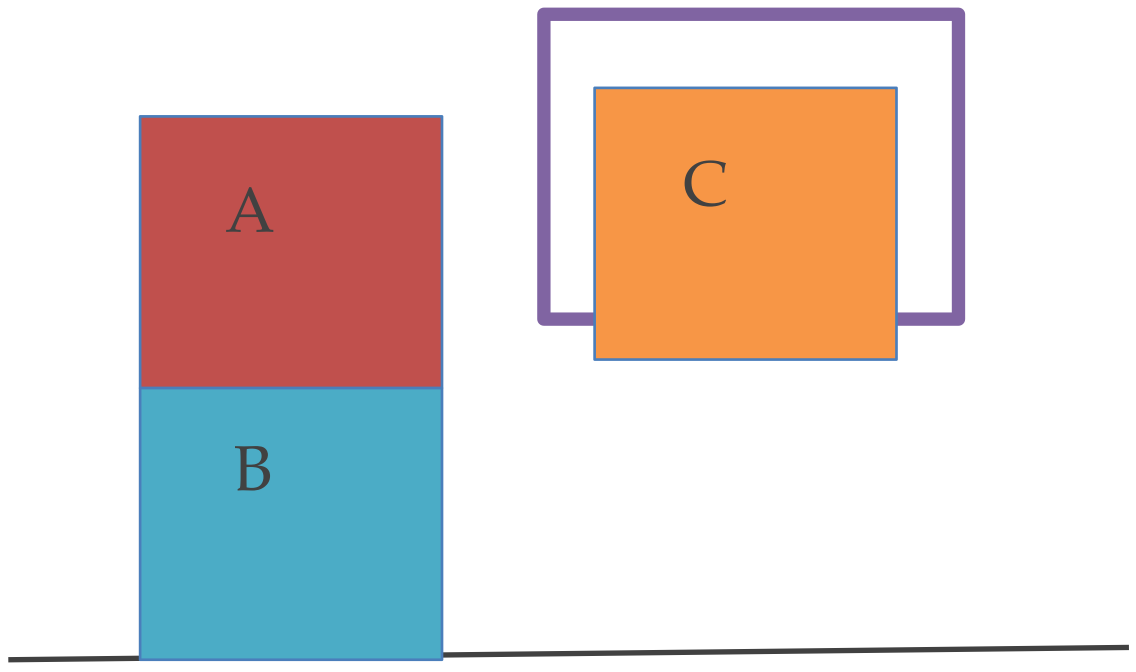


Quiz: Check all that apply

Instances: A, B, C

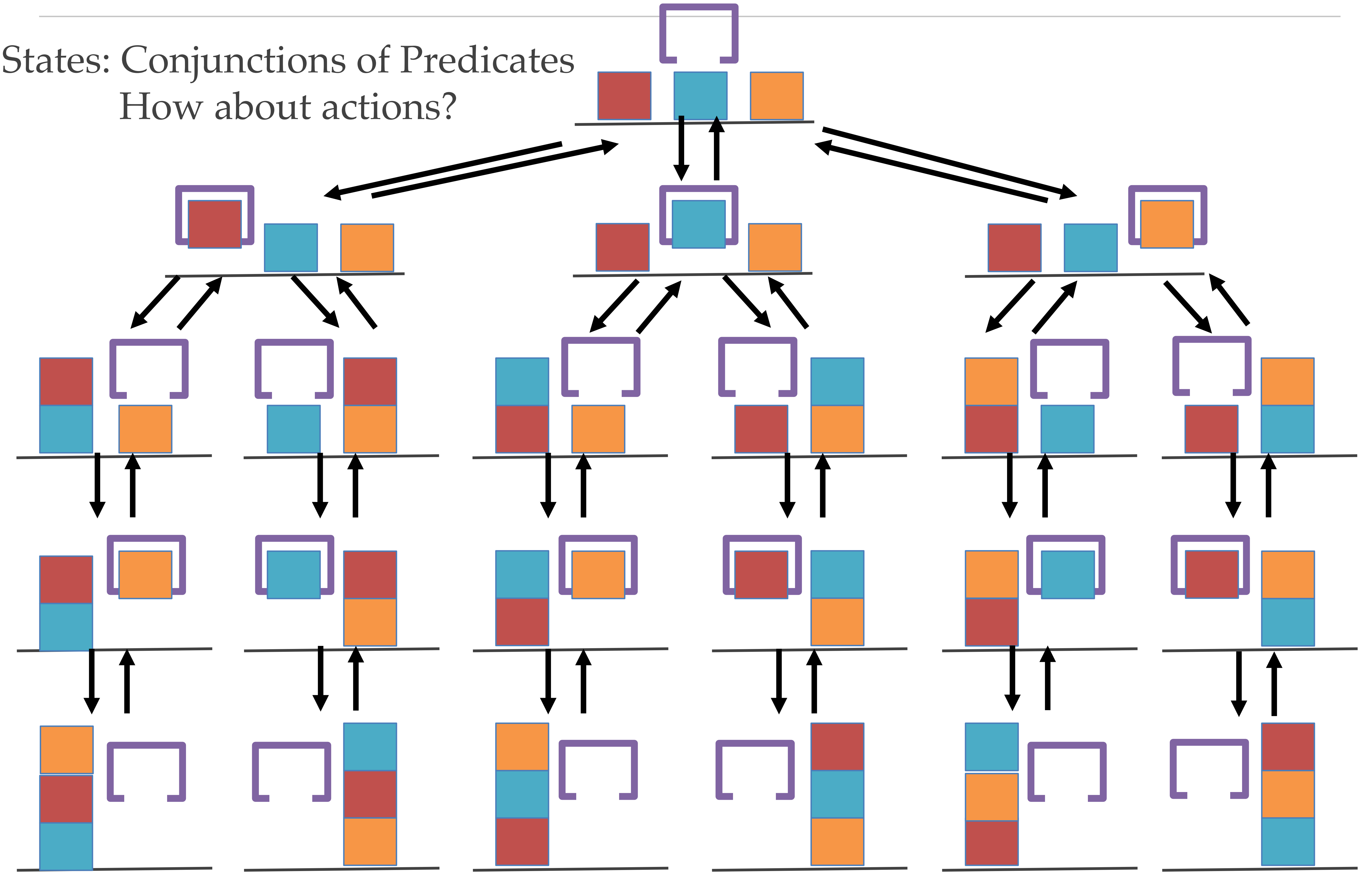
Propositions:

1) In-Hand(A)
2) In-Hand(B)
3) In-Hand(C)
4) On-Table(A)
5) On-Table(B)
6) On-Table(C)
7) On-Block(B,C)
8) On-Block(A,B)
9) HandEmpty()



Block Stacking

States: Conjunctions of Predicates
How about actions?



Operators

Actions can be applied only if some conditions are met

- Represented as conjunctions of positive predicates

Actions change the state of the world

- Represented as effects that add / delete predicates

Operator = precondition, delete list, add list

Operators can have parameters and are given names

- e.g., pick-up(o), put-down(o)

Actions for Block Stacking

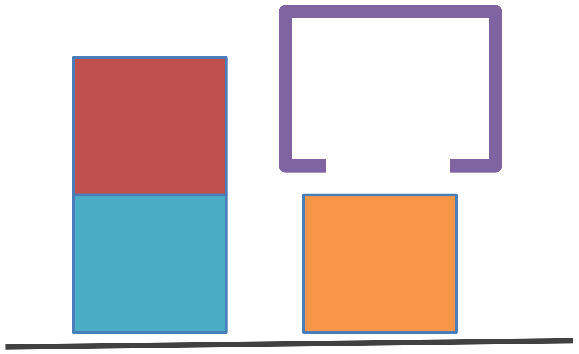
Blocks are picked up and put down by the hand

Blocks can be picked up only if they are clear

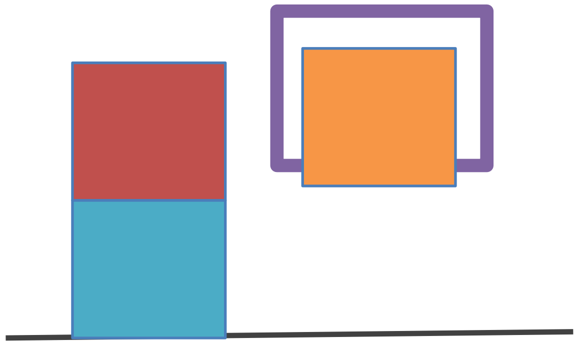
Hand can pick up a block only if the hand is empty

Hand can put down blocks on blocks or on the table

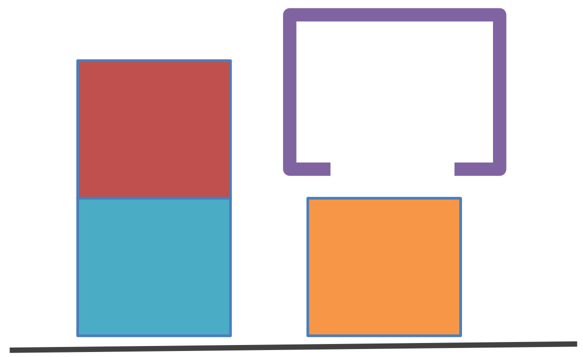
Pick Up Block from Table Example

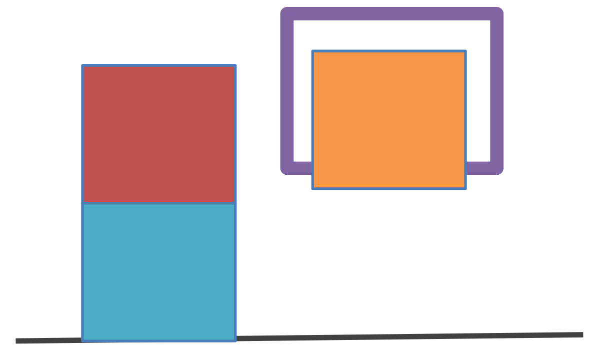


Preconditions?



Pick Up Block from Table Example





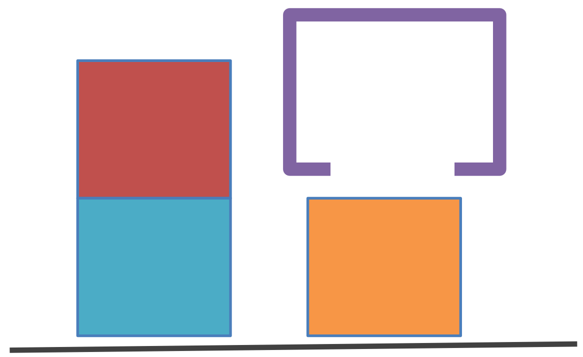
Preconditions

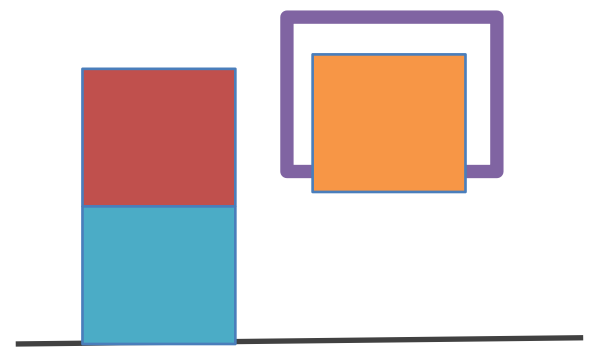
HandEmpty

On-Table(b)

Clear(b)

Pick Up Block from Table Example



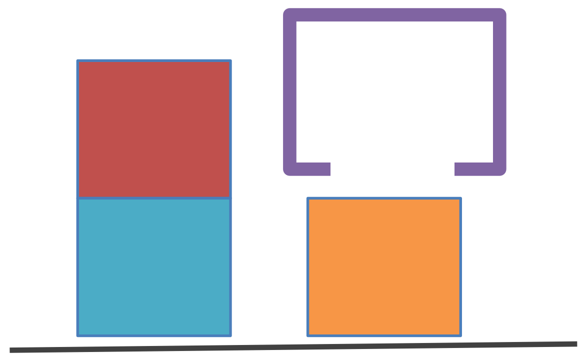


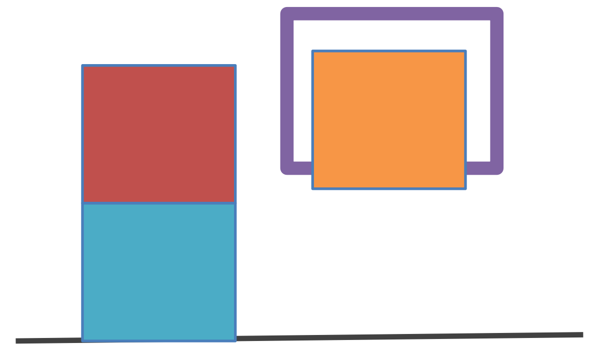
Preconditions

- HandEmpty
- On-Table(b)
- Clear(b)

Effects?

Pick Up Block from Table Example





Preconditions

HandEmpty

On-Table(b)

Clear(b)

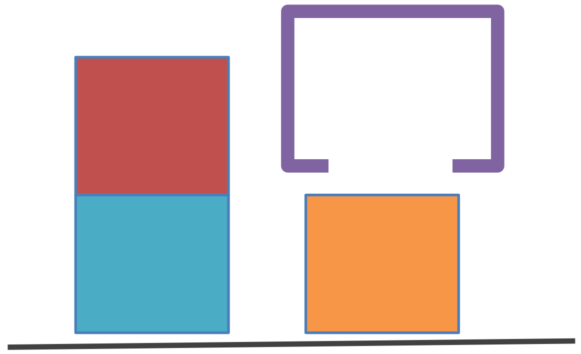
Effects

Add: Holding(b)

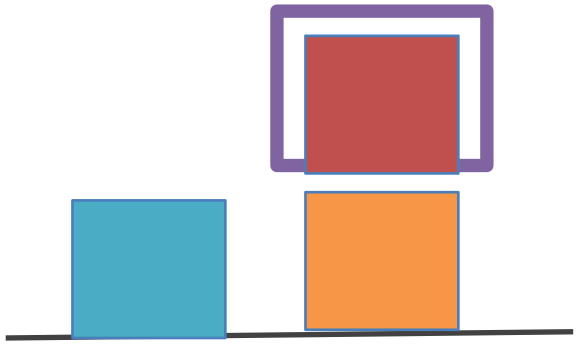
Delete: On-Table(b)

HandEmpty

Pick Block from Block Example



Preconditions



Effects

Operators for Block Stacking

Pickup_from_Table(b):

- Pre: HandEmpty, Clear(b), On-Table(b)
- Add: Holding(b)
- Delete: HandEmpty, On-Table(b)

Pickup_from_Block(b,c):

- Pre: HandEmpty, On(b,c)
- Add: Holding(b), Clear(c)
- Delete: HandEmpty, On(b,c)

Operators for Block Stacking

Pickup_from_Table(b):

Pre: HandEmpty, Clear(b), On-Table(b)

Add: Holding(b)

Delete: HandEmpty, On-Table(b)

Pickup_from_Block(b,c):

Pre: HandEmpty, On(b,c)

Add: Holding(b), Clear(c)

Delete: HandEmpty, On(b,c)

Putdown_on_Table(b):

Pre: Holding(b)

Add: HandEmpty, On-Table(b)

Delete: Holding(b)

Putdown_on_Block(b,c):

Pre: Holding(b), Clear(c)

Add: HandEmpty, On(b,c)

Delete: Clear(c), Holding(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Pickup_from_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: Holding(b)
Delete: HandEmpty, On-Table(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Pickup_from_Block(b,c):
Pre: HandEmpty, On(b,c), Clear(c), b!=c
Add: Holding(b), Clear(c)
Delete: HandEmpty, On(b,c)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R) ∧ HandEmpty ∧ On-Table(T)

Putdown_on_Table(b):
Pre: Holding(b)
Add: HandEmpty, On-Table(b)
Delete: Holding(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R) ∧ HandEmpty ∧ On-Table(T)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R) ∧ HandEmpty ∧ On-Table(T)

Pickup_from_Table(O)

Pickup_from_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: Holding(b)
Delete: HandEmpty, On-Table(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R) ∧ HandEmpty ∧ On-Table(T)

Pickup_from_Table(O)

On-Table(R) ∧ Clear(T) ∧ Clear(O) ∧ Clear(R) ∧ On-Table(T) ∧ Holding(O)

Pickup_from_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: Holding(b)
Delete: HandEmpty, On-Table(b)

Example Plan of Actions

HandEmpty ∧ On-Table(R) ∧ On(T,R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O)

Pickup_from_Block(T,R)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Holding(T) ∧ Clear(R)

Putdown_on_Table(T)

On-Table(R) ∧ Clear(T) ∧ On-Table(O) ∧ Clear(O) ∧ Clear(R) ∧ HandEmpty ∧ On-Table(T)

Pickup_from_Table(O)

On-Table(R) ∧ Clear(T) ∧ Clear(O) ∧ Clear(R) ∧ On-Table(T) ∧ Holding(O)

Putdown_on_Block(O,R)

On-Table(R) ∧ Clear(T) ∧ Clear(O) ∧ On-Table(T) ∧ On(O,R) ∧ HandEmpty

Planning is a Search Problem

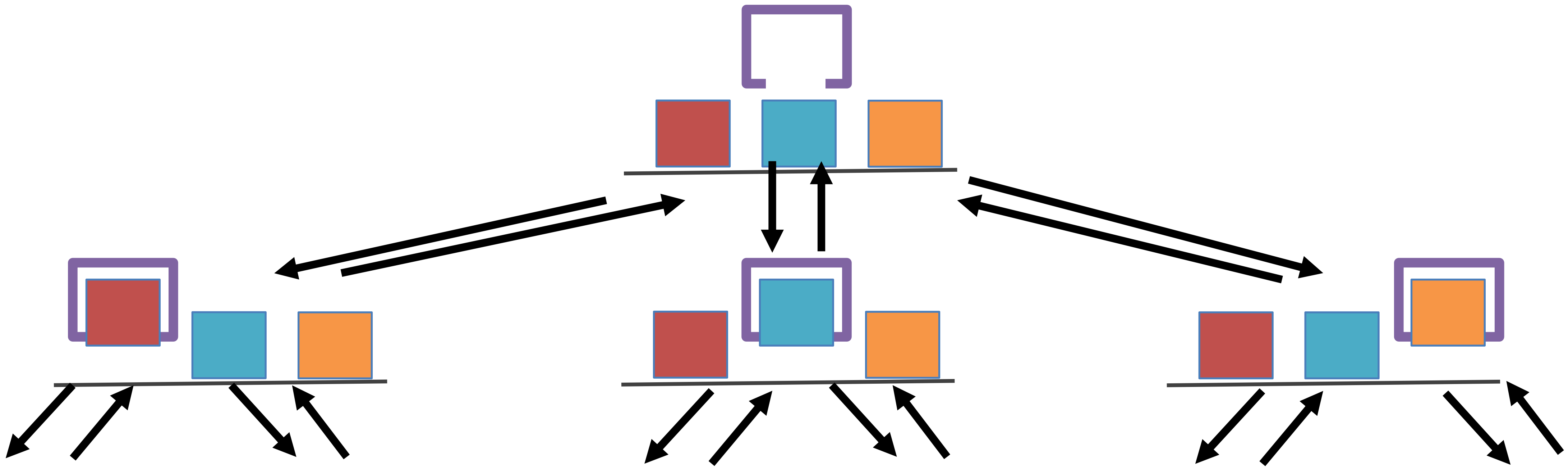
Planning problem can be represented as a graph:

States: Conjunctions of Predicates

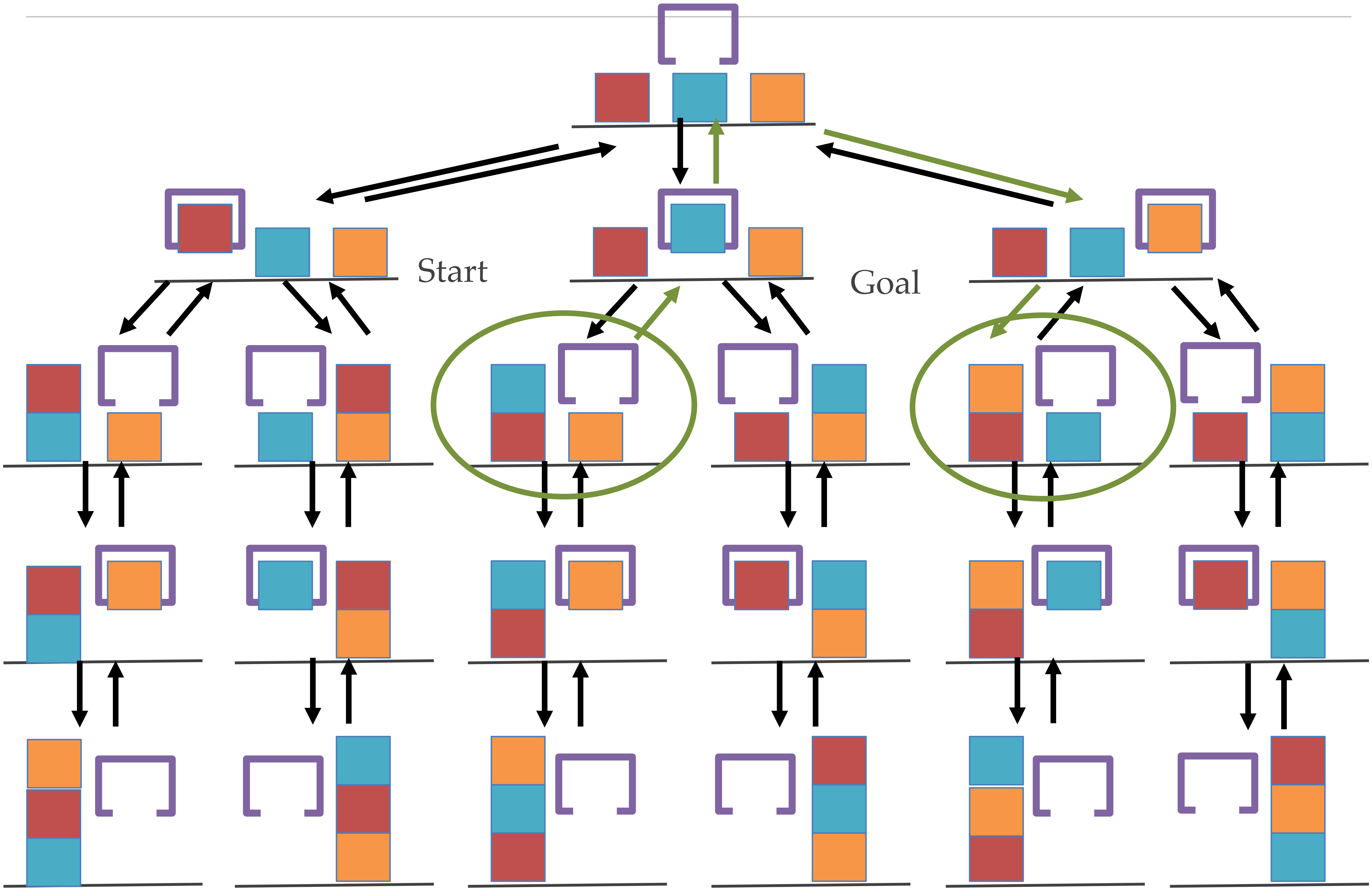
Arrows = Actions

Space complexity in terms of # predicates p

Search algorithm can be applied!



Example



Planners

- Any search algorithm (BFS, DFS, IDS…) could be used
- Here, search can also be performed backward!
  - From a given state $X_1 \land \cdots \land X_k$, consider relevant and consistent actions
  - An action is relevant if it achieves one of the X_i's
  - An action is consistent if it doesn't undo one of the X_i's
  - An operator can easily be reversed by modifying the current state:
    - Remove its positive effects
    - Add its preconditions (unless it already appears in the state)
  - Stop backward search when reaching state that is satisfied by initial state
- However, as STRIPS allow to describe compactly very large problems, A* is needed with very good heuristics

Efficient Algorithms

- STRIPS planning is PSPACE-complete
- Forward or backward A*
  - Domain-independent heuristics
    - # of unsatisfied literals in goal
  - Relaxed problem
    - sum of costs for achieving each literal in goal
    - may be inadmissible
- Specialized algorithms
  - e.g., Graphplan (See 10.3 in AIMA)

Properties of Planners

Soundness

- A planning algorithm is *sound* if all solutions found are legal plans

Completeness

- A planning algorithm is *complete* if a solution can be found whenever one actually exists

Optimality

- A planning algorithm is *optimal* if the solution optimizes some measure of plan quality