Warm Up: Represent this Blocks World

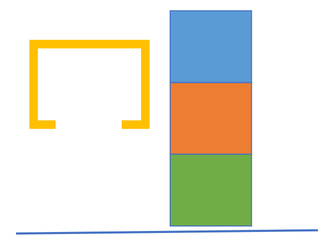
A robot arm (yellow) can pick up and put down blocks to form stacks.

It cannot pick up a block that has another block on top of it.

It cannot pick up more than one block at a time.

Any number of blocks can sit on the table.

Only one block fits on top of another.



How would you represent this block world to be able to run BFS?

How would you represent this block world to use logic to find a plan?

Hint: How would you represent the states, actions, goals, transitions?

Announcements

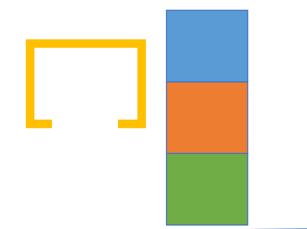
Midterm 1 Exam Graded

Assignments:

- HW5
  - Due Tue 2/25, 10pm
- P3: Logic and Classical Planning
  - Out 2/22, Due Thu 3/5 10 pm before spring break!
- HW6
  - Out Tue 2/25, 10pm
  - Due Tue 3/3, 10pm

AI: Representation and Problem Solving

Classical Planning
or Symbolic Planning



Instructors: Pat Virtue & Stephanie Rosenthal

Slide credits: CMU AI, http://ai.berkeley.edu

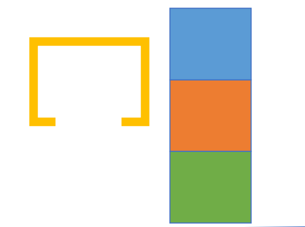
Represent this Blocks World

A robot arm (yellow) can pick up and put down blocks to form stacks.

It cannot pick up a block that has another block on top of it.

It cannot pick up more than one block at a time.

Any number of blocks can sit on the table.



How would you represent this block world to be able to run BFS?

States {on: [(blue, ora), (ora., green)], hand: None
orange,
table: [green] }

Actions

Goals

Transitions

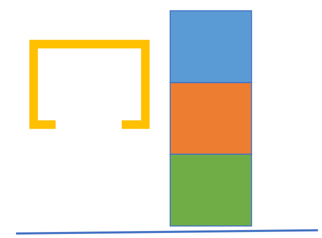
Represent this Blocks World

A robot arm (yellow) can **pick up** and **put down** blocks to form stacks.

It cannot pick up a block that has another block on top of it.

It cannot pick up more than one block at a time.

Any number of blocks can sit on the table.



How would you represent this block world to be able to run logical planning?

States BlueOnOrange ∧ OrangeOnGreen ∧ GreenOnTable ∧

Actions HandEmpty ∧ ¬OrangeOnBlue ∧ ¬GreenOn ..

Goals

Transitions hand : blue[1] ⟺ (hand : blue[0] ∧ Donothing) ∨

(hand : empty[0] ∧ pickup blue)

Pros and Cons of the Approaches

BFS

+ concise representation of objects

- more challenging to write a definition of the actions for each possible state

Logical Planning

+ successor-state axioms are relatively concise to write but there are many

- states and fluents are cumbersome to write and debug

Classical Planning:

+ concise object representation and clearer action definitions

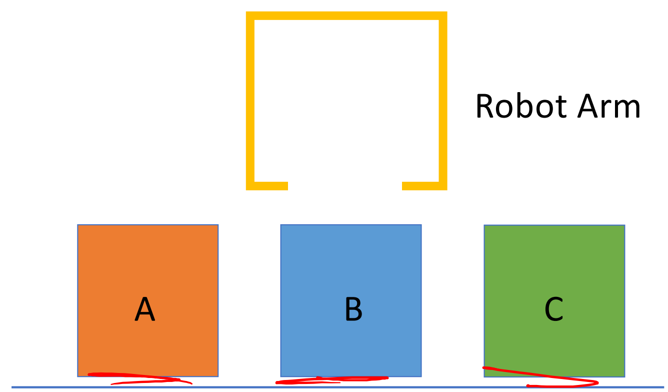
- still only works for deterministic fully observable worlds

Symbolic Representation

In Logic, we represent each object in location as a Boolean proposition

A-on-Table
B-on-Table
C-on-Table
Hand-Empty
A-on-B
A-on-C
B-on-C
Clear-A
Clear-C

A-In-Hand
B-In-Hand
C-In-Hand
B-on-A
C-on-A
C-on-B
Clear-B

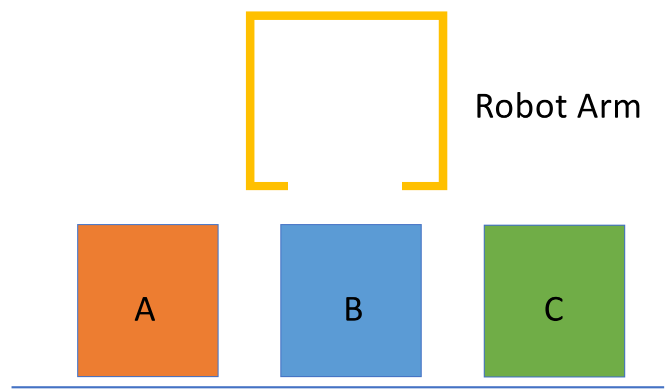


Symbolic Representation

In Logic, we represent each object in location as a Boolean proposition

The issue is that every time you add an object into this world, your number of propositions expands exponentially as do your successor-state axioms

A-on-Table	A-In-Hand
B-on-Table	B-In-Hand
C-on-Table	C-In-Hand
D-on-Table	D-In-Hand
Hand-Empty	
A-on-B	B-on-A
A-on-C	C-on-A
B-on-C	C-on-B
D-on-A	D-on-B
D-on-C	



D-on-A[t] <=> ?
D-on-B[t] <=> ?
D-on-C[t] <=> ?
D-In-Hand <=> ?
D-on-Table <=> ?

Idea of Classical Planning

Notes: We can give them any names we want!
And we can change representation too!

Represent objects/values separately from the state (instances)

→ A, B, C, ~~Table~~

Define predicates as true/false functions over the objects

On(A, Table) = True

Ontable(A)

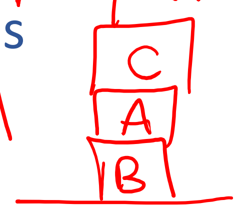
On(A, B) = False

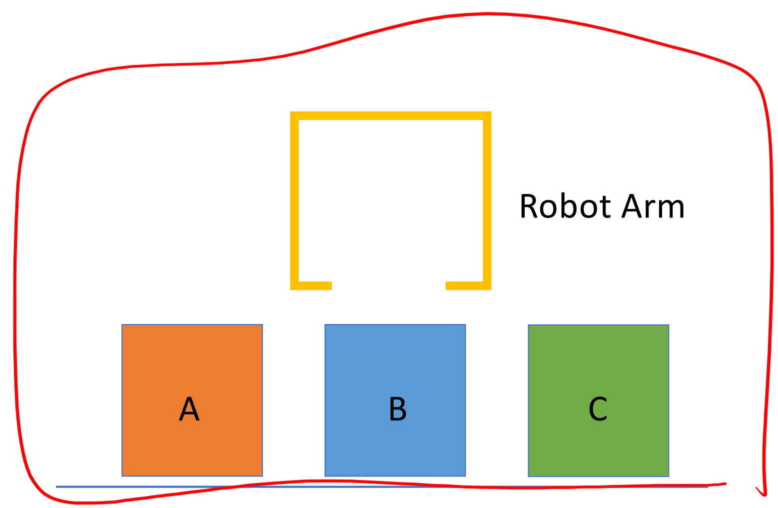
States are conjunctions of predicates

OnTable(A) ∧ OnTable(B) ∧ Ontable(C) ∧ HandEmpty()

Goals are conjunctions of predicates

On(A, B) ∧ On(C, A) ∧ OnTable(B)





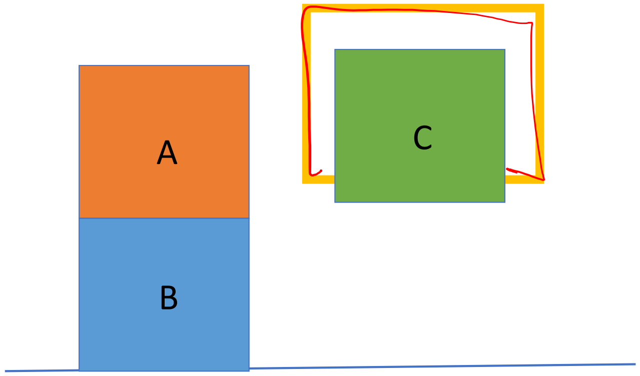
Piazza Poll 1

Which predicates apply to this state? (Select all that apply)

Instances: A, B, C

Predicates:

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) top bottom
8) On-Block(A,B)
9) HandEmpty()



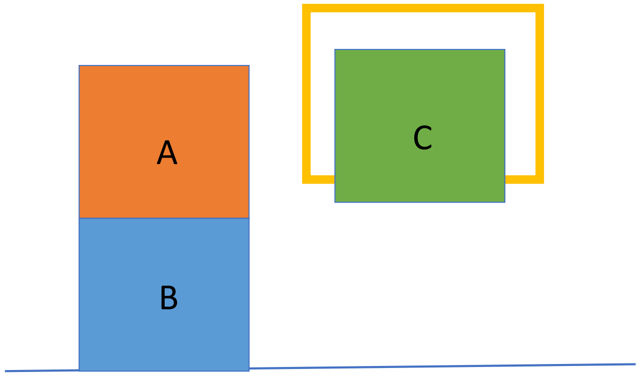
Piazza Poll 1

Which predicates apply to this state? (Select all that apply)

Instances: A, B, C

Predicates:

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()



Full State Description

Instances: A, B, C

Predicates:

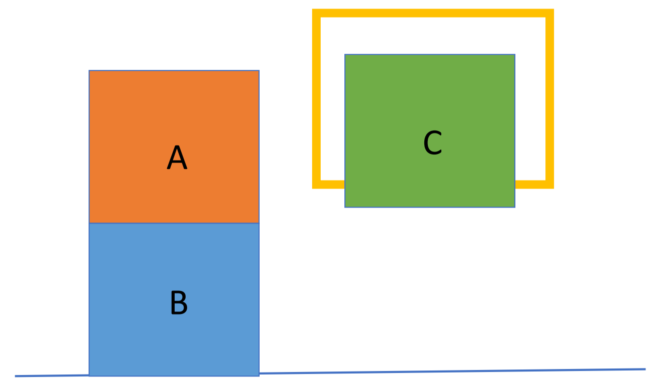
In-Hand(C)

On-Table(B)

On-Block(A,B)

Clear(A)

Clear(C)



Optional: ~HandEmpty(), ~On-Table(C), ~On-Table(A), ~On-Block(B,A), ~On-Block(C,A), ~On-Block(B,C), ~On-Block(C,B), ~On-Block(A,C), ~Clear(B), ~In-Hand(A), ~In-Hand(B)

Operators

Operators change the state by adding/deleting predicates

Preconditions:

Actions can be applied only if all precondition predicates are true in the current state

Effects:

New state is a copy of the current predicates with the addition or deletion of specified predicates

Unlike the successor-state axioms, we do not explicitly represent time and we can use our objects and predicates to more easily scale to new more complex problems (e.g., new objects, predicates, and operators).

Rules of Block World

Blocks are picked up and put down by the hand

Blocks can be picked up only if they are clear

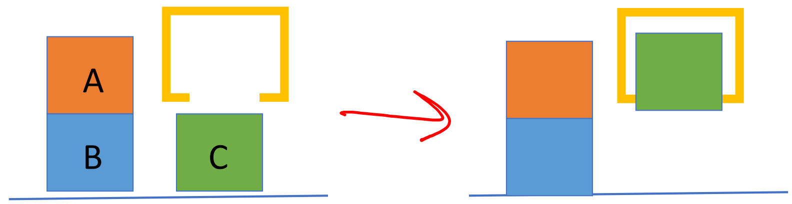
Hand can pick up a block only if the hand is empty

Hand can pick up and put down blocks on blocks or on the table

Pickup Block C from Table (State Transition)

Start

Next State



Instances:

Blocks A, B, C

Possible Predicates:

HandEmpty()

On-Table(block)

On-Block(b1,b2)

Clear(block)

In-Hand(block)

precondition

Start State:

HandEmpty()

On-Table(B)

On-Table(C)

On-Block(A,B)

Clear(A)

Clear(C)

Next State:

In-Hand(C) ← added

On-Table(B)

On-Block(A,B)

Clear(A)

Clear(C)

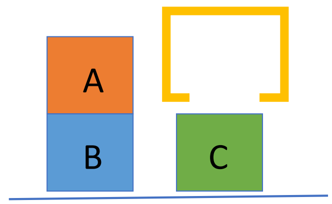
ontable (c) ← deleted

handempty () ←

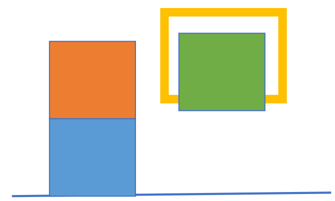
Pickup Block C from Table (Preconditions, Effects)

Instances:
Blocks A, B, C

Possible Predicates:
HandEmpty()
On-Table(block)
On-Block(b1,b2)
Clear(block)
In-Hand(block)



<u>State:</u>
HandEmpty()
On-Table(B)
On-Table(C)
On-Block(A,B)
Clear(A)
Clear(C)



<u>State:</u>
In-Hand(C)
On-Table(B)
On-Block(A,B)
Clear(A)
Clear(C)
Delete HandEmpty()
Delete On-Table(C)

Operator: Pickup-Block-C from Table

Preconditions	Effects
HandEmpty()	Add In-Hand(C)
Clear(C)	Delete HandEmpty()
On-Table(C)	On-Table(C)

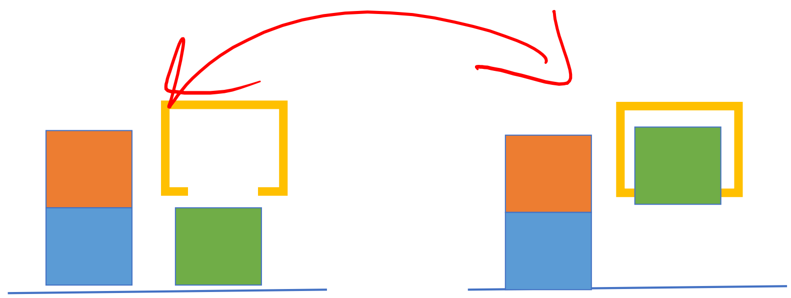
Operator: Pickup-Block from Table

variable

Preconditions	Effects
HandEmpty()	Add In-Hand(block)
Clear(block)	Delete HandEmpty()
On-Table(block)	On-Table(block)

Create a variable that takes on the value of a particular instance for all times it appears in an operator.

Operator: PutDown-Block on Table



Preconditions

In-Hand(block)

Effects

Add $\boxed{\text{HandEmpty()}}$

On-Table(block)

Delete In-Hand(block)

~In-Hand(block)

Why do we not need to check if ~HandEmpty() is true?

Full State Description

Instances: A, B, C

Predicates:

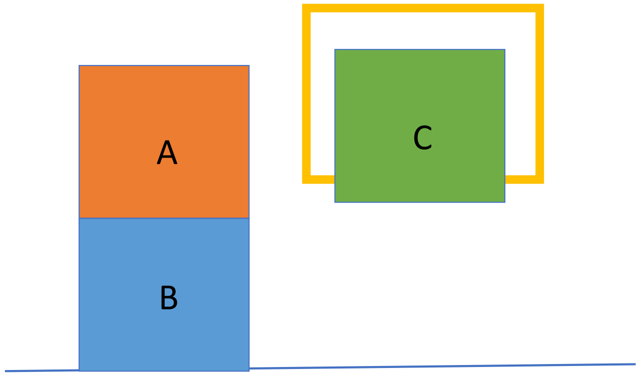
In-Hand(C)

On-Table(B)

On-Block(A,B)

Clear(A)

Clear(C)



Optional: ~HandEmpty(), ~On-Table(C), ~On-Table(A), ~On-Block(B,A), ~On-Block(C,A), ~On-Block(B,C), ~On-Block(C,B), ~On-Block(A,C), ~Clear(B), ~In-Hand(A), ~In-Hand(B)

RULE OF THUMB: If you must match that Predicate is explicitly not true, you must include ~Predicate in the state description.

Operators for Block Stacking

Pickup_Table(b):

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

Add: In-Hand(b)

Delete: HandEmpty(), On-Table(b)

Pickup_Block(b,c):

Pre: HandEmpty(), On-Block(b,c), b!=c

Add: In-Hand(b), Clear(c)

Delete: HandEmpty(), On-Block(b,c)

Putdown_Table(b):

Pre: In-Hand(b)

Add: HandEmpty(), On-Table(b)

Delete: In-Hand(b)

Putdown_Block(b,c):

Pre: In-Hand(b), Clear(c)

Add: HandEmpty(), On-Block(b,c)

Delete: Clear(c), In-Hand(b)

Why do we need separate operators for table vs on a block?

Example Matching Operators

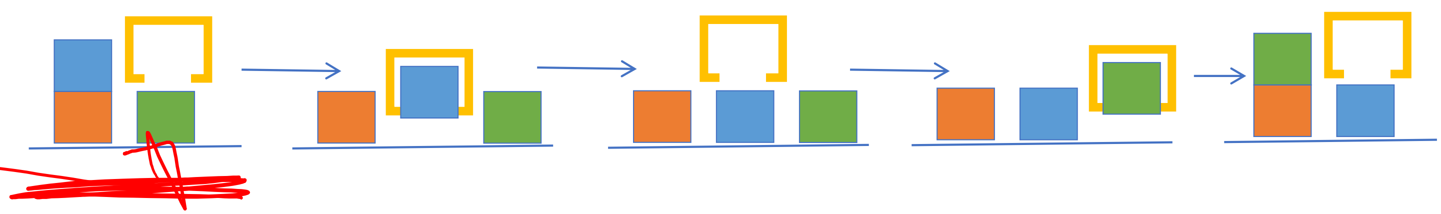
HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Example Matching Operators

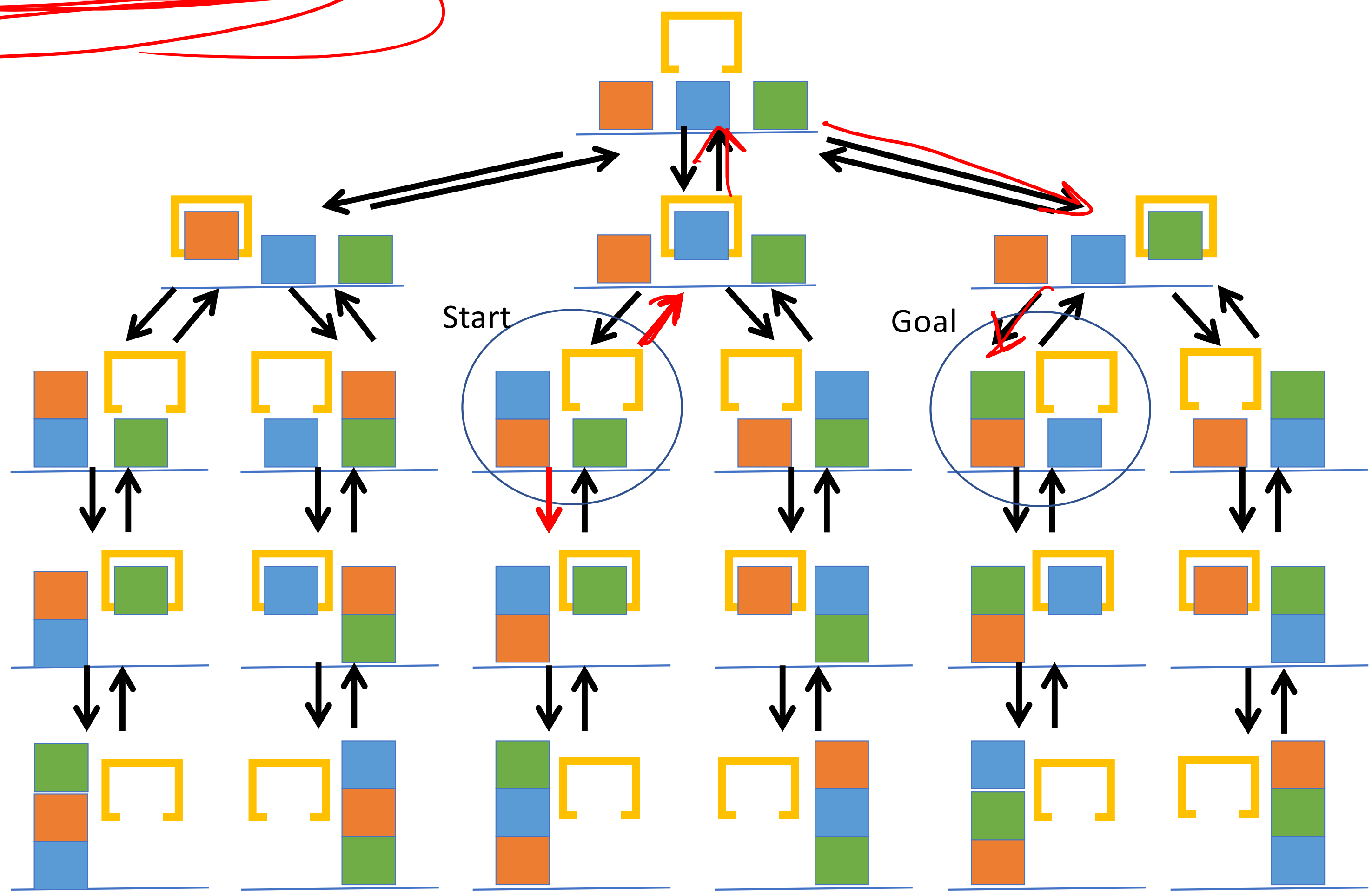
HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Pickup_Block(b,c):
Pre: HandEmpty(), On-Block(b,c), b!=c
Add: In-Hand(b), Clear(c)
Delete: HandEmpty(), On(b,c)

Pickup_Table(b):
Pre: HandEmpty, Clear(b), On-Table(b)
Add: In-Hand(b)
Delete: HandEmpty(), On-Table(b)



Planning Graph (sometimes called Reachability Graph)



Example Matching Operators

HandEmpty() & On-Table(O) & On-Block(B,O) & Clear(B) & On-Table(G) & Clear(G)

Pickup_Block(B,O)

On-Table(O) & Clear(B) & On-Table(G) & Clear(G) & In-Hand(B) & Clear(O)

Putdown_Table(B)

On-Table(O) & Clear(O) & On-Table(G) & Clear(G) & Clear(B) & On-Table(B) & HandEmpty()

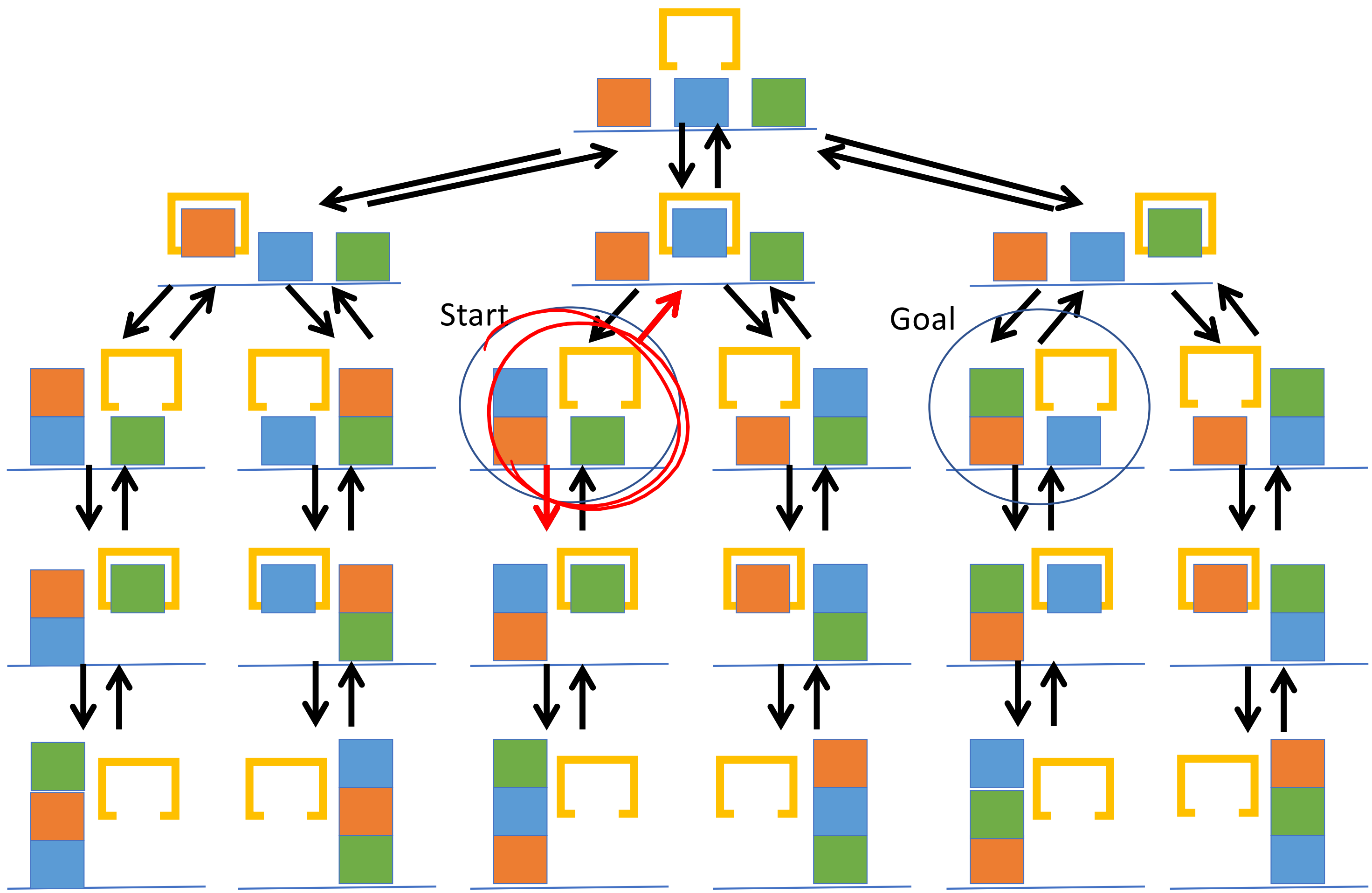
Pickup_Table(G)

On-Table(O) & Clear(B) & Clear(G) & Clear(O) & On-Table(B) & In-Hand(G)

Putdown_Block(G,O)

On-Table(O) & Clear(B) & Clear(G) & On-Table(B) & On-Block(G,O) & HandEmpty()

What kind of search can we do with a Planning Graph?



Finding Plans with Symbolic Representations

Breadth-First Search

Sound? Yes

Complete? Yes

Optimal? Yes

Soundness - all solutions found are legal plans

Completeness - a solution can be found whenever one actually exists

Optimality - the order in which solutions are found is consistent with some measure of plan quality

Linear Planning

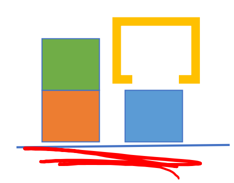
Since we have a conjunction of goal predicates, let’s try to solve one at a time

- Maintain a stack of achievable goals
- Use BFS (or anything else) to find a plan to achieve that single goal
- Add a goal back on the stack if a later change makes it violated

Linear Planning Example 1

Goal Stack:
On-Table(B)
On-Table(O)
On-Block(G,O)
Clear(G)
Clear(B)

Action Plan:
On-Table(B)
Pickup-Block(B,O)
Put-Table(B)



Linear Planning Example 1

Goal Stack:
On-Table(O)
On-Block(G,O)
Clear(G)
Clear(B)

Action Plan:
- On-Table(B)
  - Pickup-Block(B,O)
  - Put-Table(B)
- On-Table(O)

Linear Planning Example 1

Goal Stack:
On-Block(G,O)
Clear(G)
Clear(B)

Action Plan:
- On-Table(B)
  - Pickup-Block(B,O)
  - Put-Table(B)
- On-Table(O)
- On-Block(G,O)
  - Pickup-Table(G)
  - Put-Block(G,O)

Linear Planning Example 1

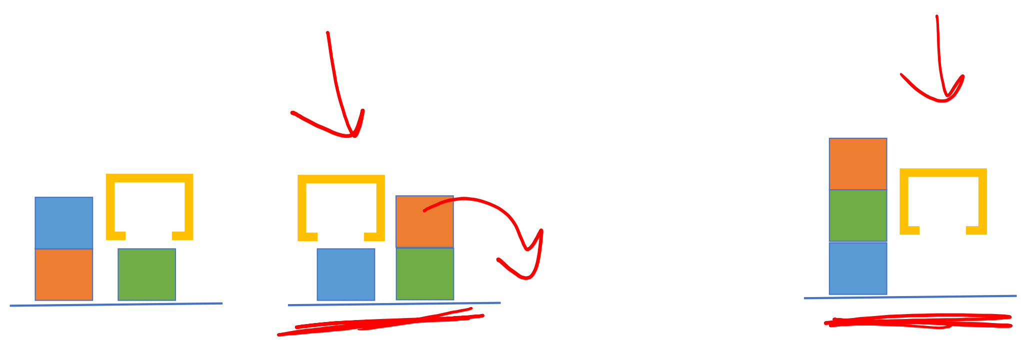
Goal Stack:
Clear(G)
Clear(B)

Action Plan:
- On-Table(B)
  - Pickup-Block(B,O)
  - Put-Table(B)
- On-Table(O)
- On-Block(G,O)
  - Pickup-Table(G)
  - Put-Block(G,O)
- Clear(G)
- Clear(B)

Linear Planning Example 2

Goal Stack:
On-Block(O,G)
On-Table(B)
On-Block(G,B)
Clear(O)

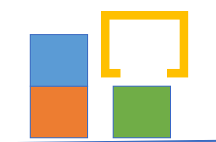
Action Plan:
- On-Block(O,G)
  - Pickup-Block(B,O)
  - ~~Put-Table(B)~~ orange on table
  - Pickup-Table(O)
  - Put-Block(O,G)

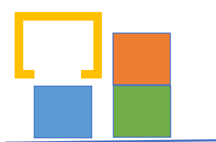


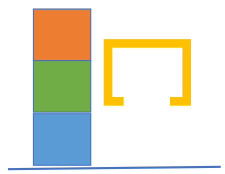
Linear Planning Example 2

Goal Stack:
On-Table(B)
On-Block(G,B)
Clear(O)

Action Plan:
- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
  - Pickup-Table(O)
  - Put-Block(O,G)
- On-Table(B)



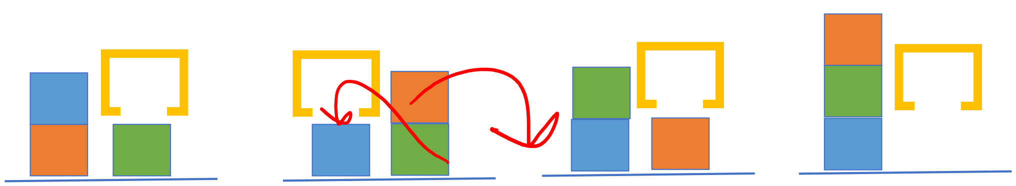




Linear Planning Example 2

Goal Stack:
On-Block(G,B)
Clear(O)

Action Plan:
- On-Block(O,G)
    - Pickup-Block(B,O)
    - Put-Table(B)
    - Pickup-Table(O)
    - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
    - Pickup-Block(O,G)
    - Put-Table(O)
    - Pickup-Table(G)
    - Put-Block(G,B)



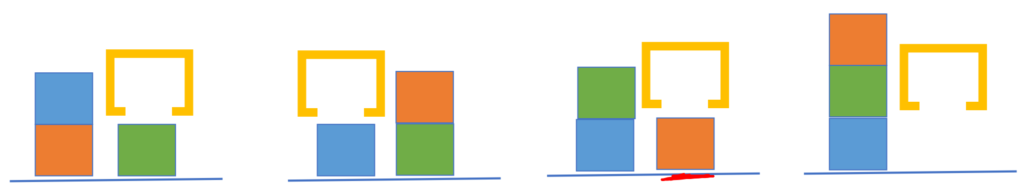
Linear Planning Example 2

Goal Stack:
On-Block(O,G)
Clear(O)

Action Plan:

- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
  - Pickup-Table(O)
  - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
  - Pickup-Block(O,G)
  - Put-Table(O)
  - Pickup-Table(G)
  - Put-Block(G,B)

- On-Block(O,G)
  - Pickup-Table(O)
  - Put-Block(O,G)

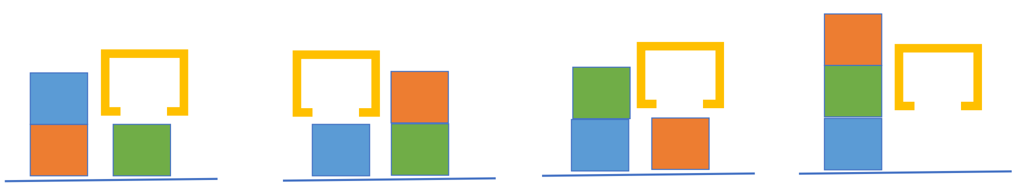


Linear Planning Example 2

Goal Stack:
Clear(O)

Action Plan:

- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
  - Pickup-Table(O)
  - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
  - Pickup-Block(O,G)
  - Put-Table(O)
  - Pickup-Table(G)
  - Put-Block(G,B)
- On-Block(O,G)
  - Pickup-Table(O)
  - Put-Block(O,G)
- Clear(O)



Linear Planning Example 2

Goal Stack:

Action Plan:

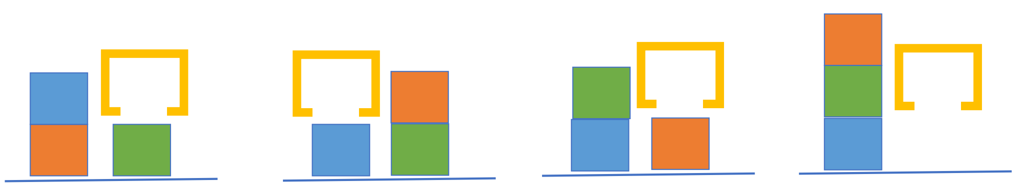
- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
  - Pickup-Table(O)
  - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
  - Pickup-Block(O,G)
  - Put-Table(O)
  - Pickup-Table(G)
  - Put-Block(G,B)
- On-Block(O,G)
  - Pickup-Table(O)
  - Put-Block(O,G)
- Clear(O)

What happened?

Is linear planning sound?

Is linear planning complete?

Is linear planning optimal?



Sussman's Anomaly

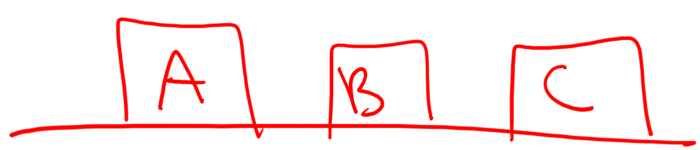
A weakness of linear planning is that sometimes you get long plans

One goal can be achieved

The second goal immediately undoes it

In fact, there are some problems for which solving goals one at a time will never result in a feasible plan.

Note: This isn't just a choice of goals. The anomaly happens no matter which goal is first



Linear Planning Soundness, Completeness, Optimality

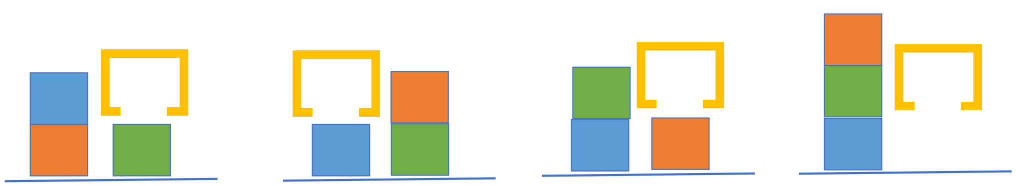
Goal Stack:

Action Plan:

- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
  - Pickup-Table(O)
  - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
  - Pickup-Block(O,G)
  - Put-Table(O)
  - Pickup-Table(G)
  - Put-Block(G,B)
- On-Block(O,G)
  - Pickup-Table(O)
  - Put-Block(O,G)
- Clear(O)

What happened?

Is linear planning sound?
Is linear planning complete?
Is linear planning optimal?



Piazza Poll 2

Goal Stack:

Action Plan:

- On-Block(O,G)
    - Pickup-Block(B,O)
    - Put-Table(B)
    - Pickup-Table(O)
    - Put-Block(O,G)
- On-Table(B)
- On-Block(G,B)
    - Pickup-Block(O,G)
    - Put-Table(O)
    - Pickup-Table(G)
    - Put-Block(G,B)
- On-Block(O,G)
    - Pickup-Table(O)
    - Put-Block(O,G)
- Clear(O)

a plan returned is legal

What happened?

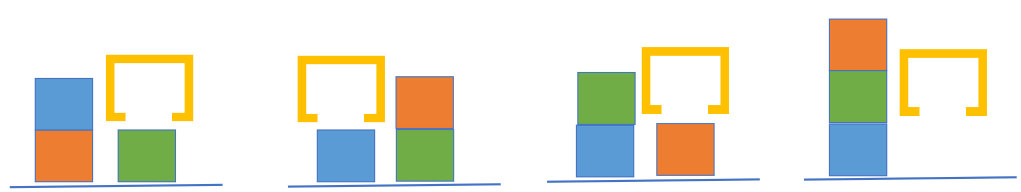
Yes Is linear planning sound?

No Is linear planning complete?

No Is linear planning optimal?

shortest

it finds a plan if it exist



Non-Linear Planning

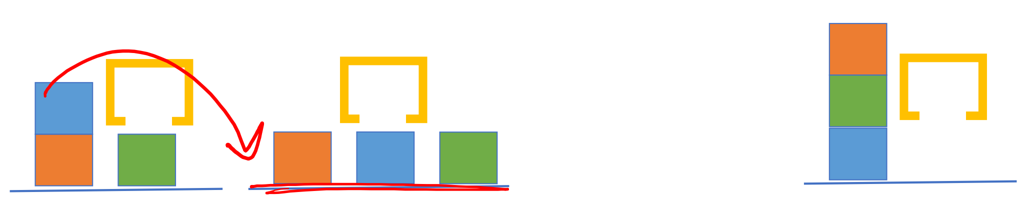
Interleave goals to achieve plans

- Maintain a set of unachieved goals
- Search all interleavings of goals
- Add a goal back to the set if a later change makes it violated

Non-Linear Planning Example (same as Lin Plan 2)

Goal Set:
On-Block(O,G)
On-Table(B)
On-Block(G,B)
Clear(O)

Action Plan:
- On-Block(O,G)
    - Pickup-Block(B,O)
    - Put-Table(B)
- Stop and Switch Goals!

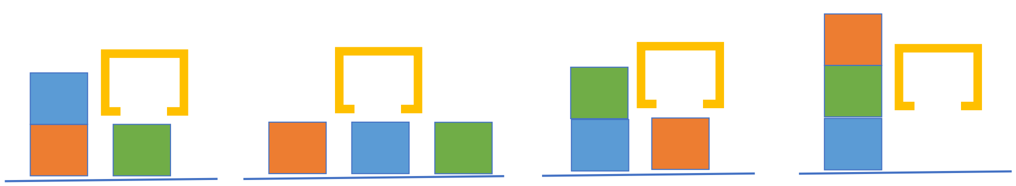


Linear Planning Example 2

Goal Set:
On-Block(O,G)
On-Table(B)
On-Block(G,B)
Clear(O)

Action Plan:

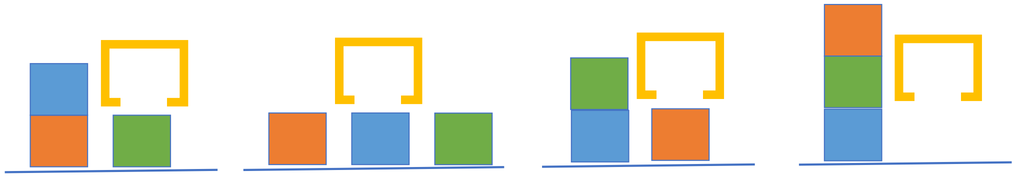
- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
- Stop and Switch Goals!
- On-Block(G,B)
  - Pickup-Table(G)
  - Put-Block(G,B)



Non-Linear Planning Example

Goal Set:
On-Block(O,G)
On-Table(B)
On-Block(G,B)
Clear(O)

Action Plan:
- On-Block(O,G)
  - Pickup-Block(B,O)
  - Put-Table(B)
- Stop and Switch Goals!
- On-Block(G,B)
  - Pickup-Table(G)
  - Put-Block(G,B)
- On-Block(O,G)
  - Pickup-Table(O)
  - Put-Block(O,G)



Non-Linear Planning

Interleave goals to achieve plans

- Maintain a set of unachieved goals
- Search all interleavings of goals
- Add a goal back to the set if a later change makes it violated

It is complete, but takes longer to search

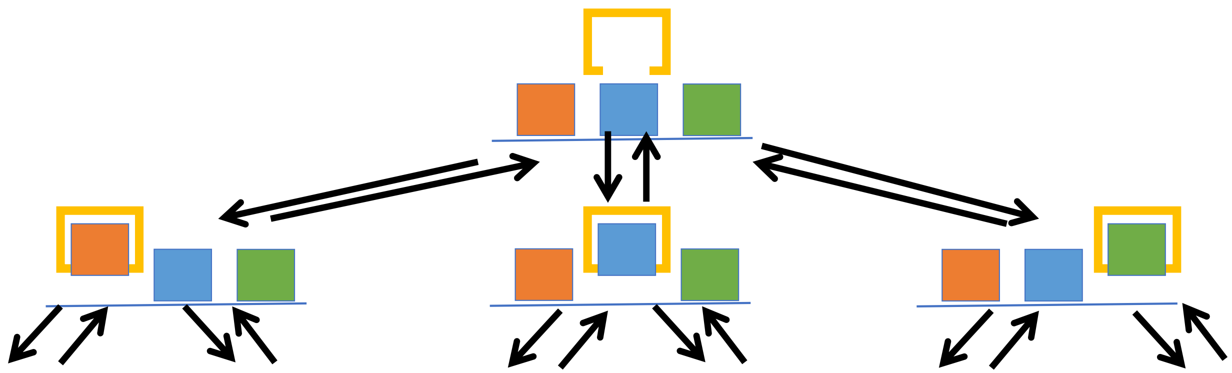
It can produce shorter plans

- Optimal plans if all interleavings are searched

Size of the Search Tree

A planning tree's size is exponential in the number of predicates

Even if we use linear or non-linear planning, they use this graph



Can we reduce the size of the planning graph?

GraphPlan and GraphPlan Graph Representation

Construct an approximation of the planning graph in polynomial space

- The GraphPlan graph computes the **possibly reachable** states although they aren't necessarily **feasible** multiple actions

first time I have a plan w/o mutex actions

Planning graphs contain two types of layers

- Proposition layers – all reachable predicates
- Action layers – actions that could be taken
- Both layers represent one time step

Effects

Actions

GraphPlan algorithm includes two subtasks

- **Extend**: One time step (two layers) in the planning graph
- **Search**: Find a valid plan in the planning graph

2N

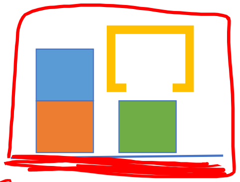
GraphPlan finds a plan or proves that no plan has fewer time steps

- Each ~~time step~~ can contain multiple actions

Levels

Level

Building a GraphPlan Graph



Start the planning graph with all starting predicates

HandEmpty

On-Table(O)

On-Table(G)

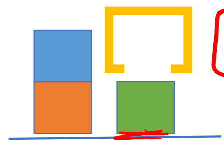
On(B,O)

Clear(G)

Clear(B)

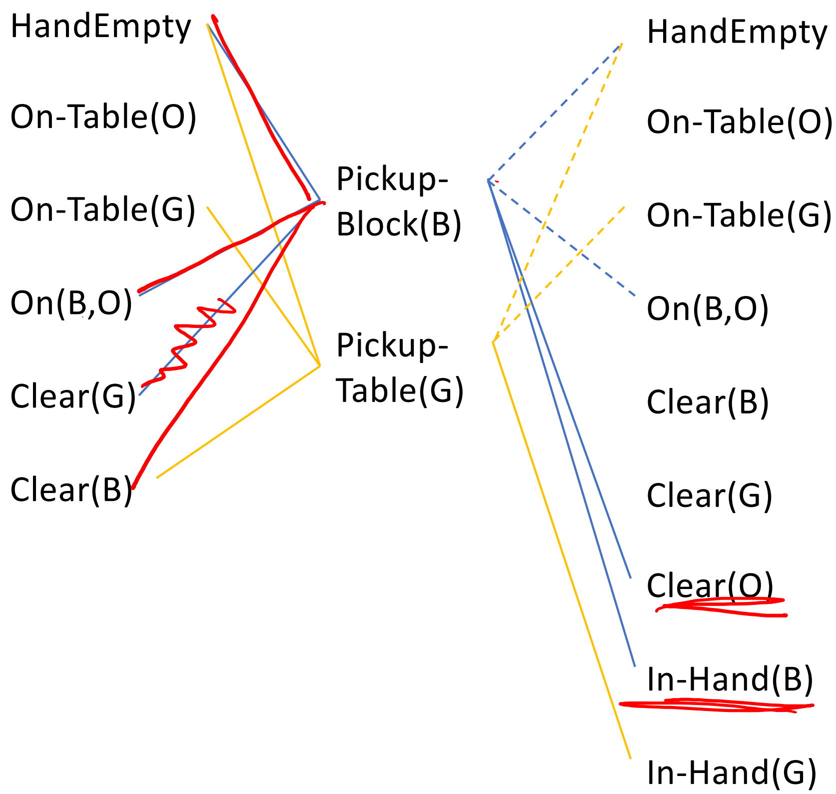
level 0

Building a GraphPlan Graph



Extend the graph with all applicable actions

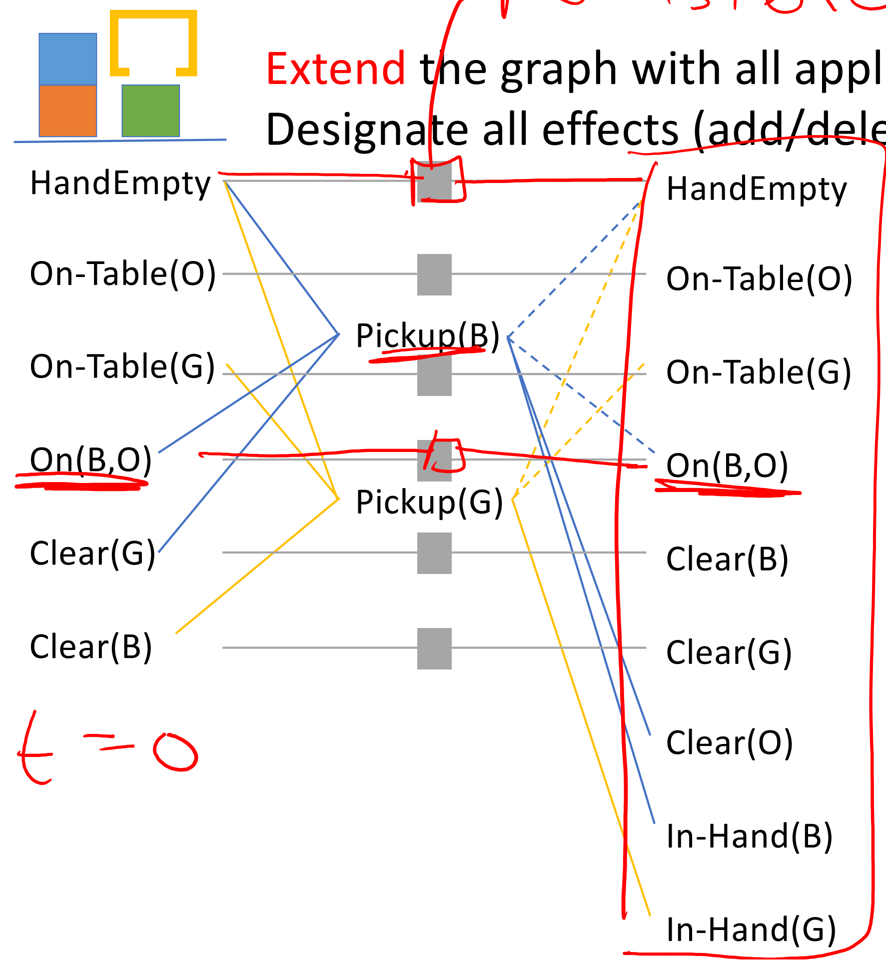
Designate all effects (add/delete)



Building a GraphPlan Graph

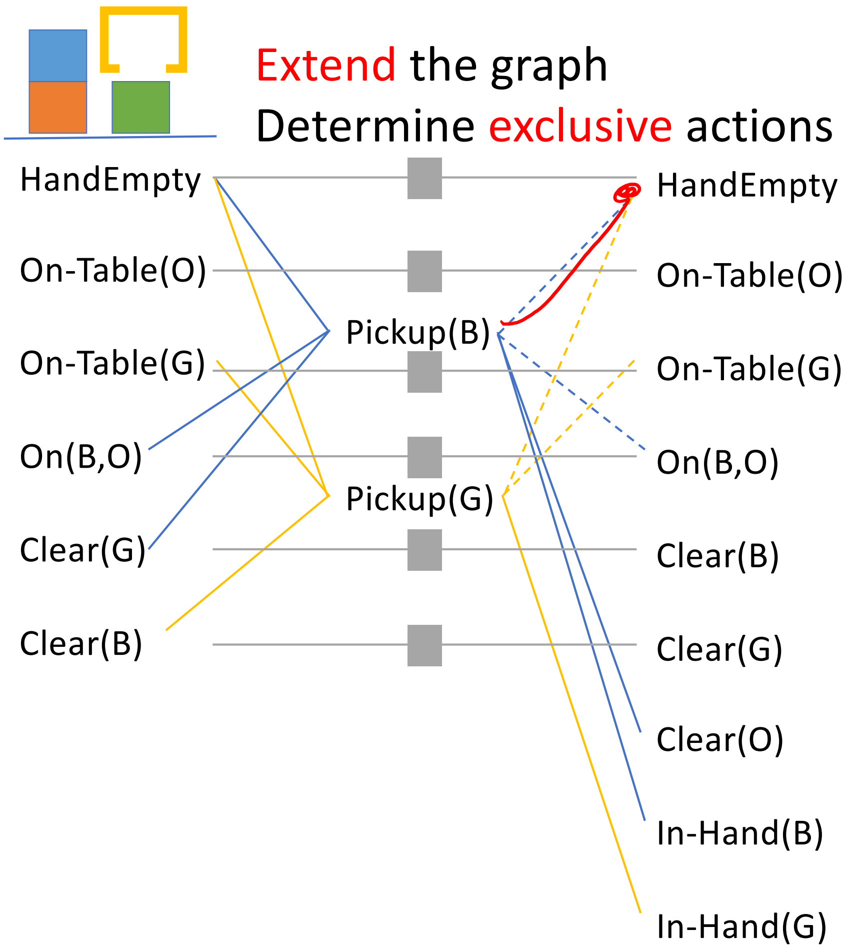
persistence operator, no-op

Extend the graph with all applicable actions, noops
Designate all effects (add/delete)



Building a GraphPlan Graph

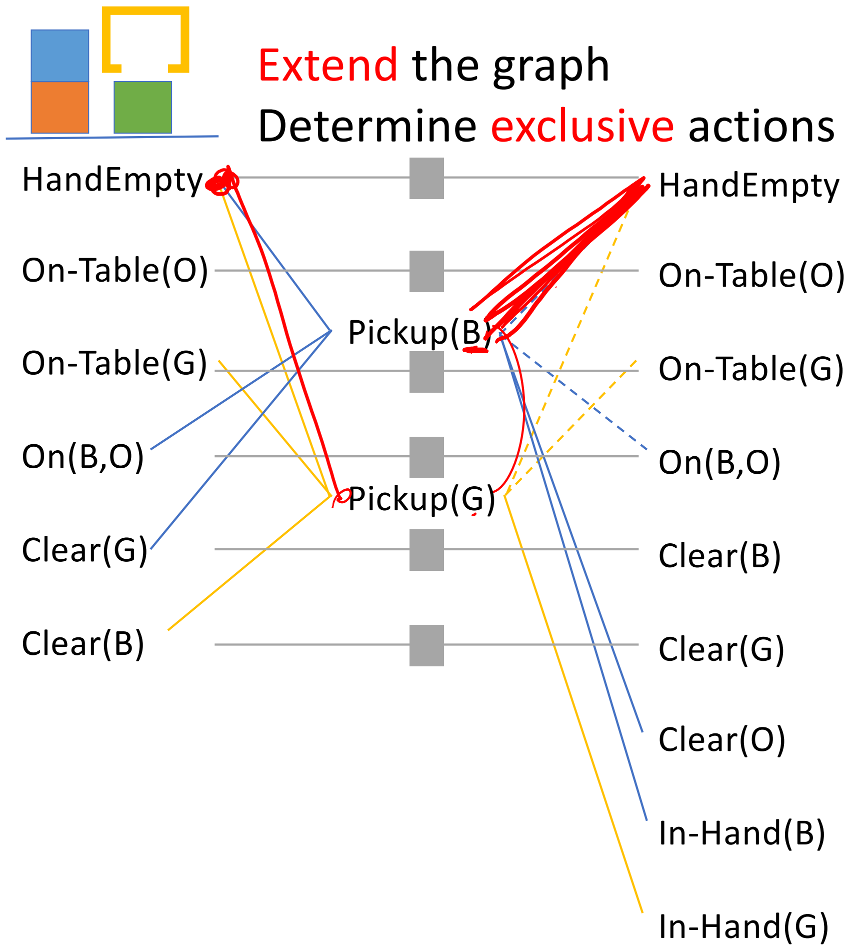
Extend the graph
Determine exclusive actions



Actions A and B are ***exclusive (mutex)*** at action-level *i*, if:

- **Interference**: one action effect deletes or negates a **precondition** of the other
- **Inconsistency**: one action effect deletes or negates the **effect** of the other
- **Competing Needs**: the actions have preconditions that are mutex in proposition-level $i - 1$

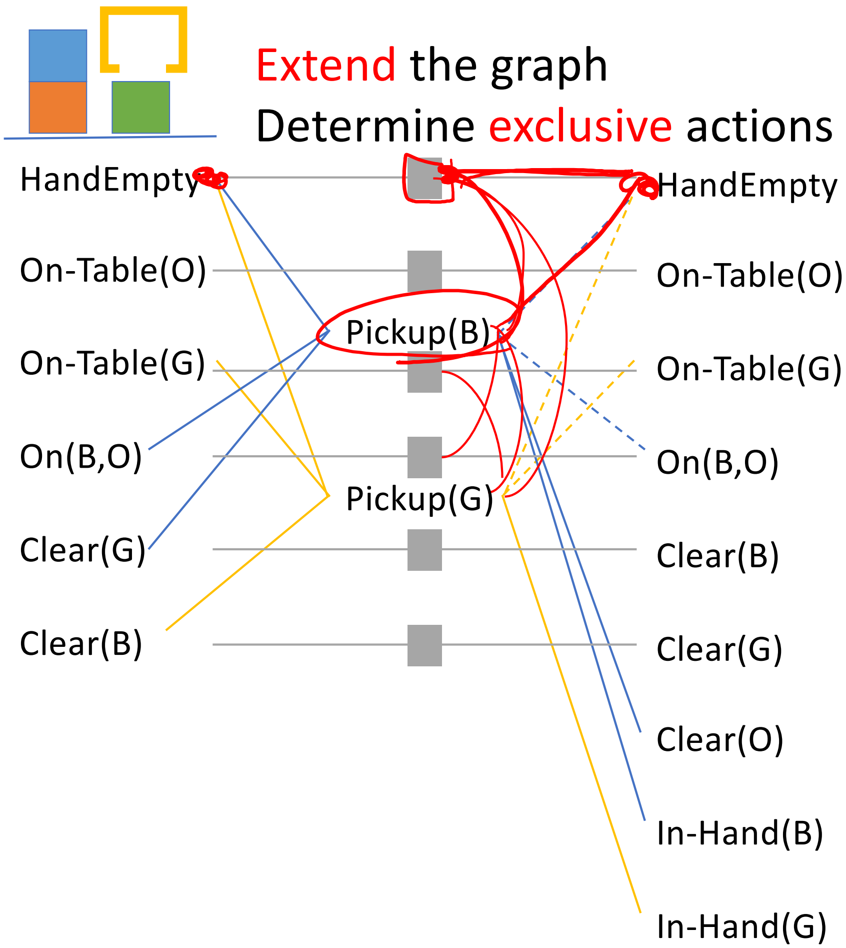
Building a GraphPlan Graph



Actions A and B are ***exclusive (mutex)*** at action-level i, if:

- **Interference**: one action effect deletes or negates a **precondition** of the other
- **Inconsistency**: one action effect deletes or negates the **effect** of the other
- **Competing Needs**: the actions have preconditions that are mutex in proposition-level $i - 1$

Building a GraphPlan Graph



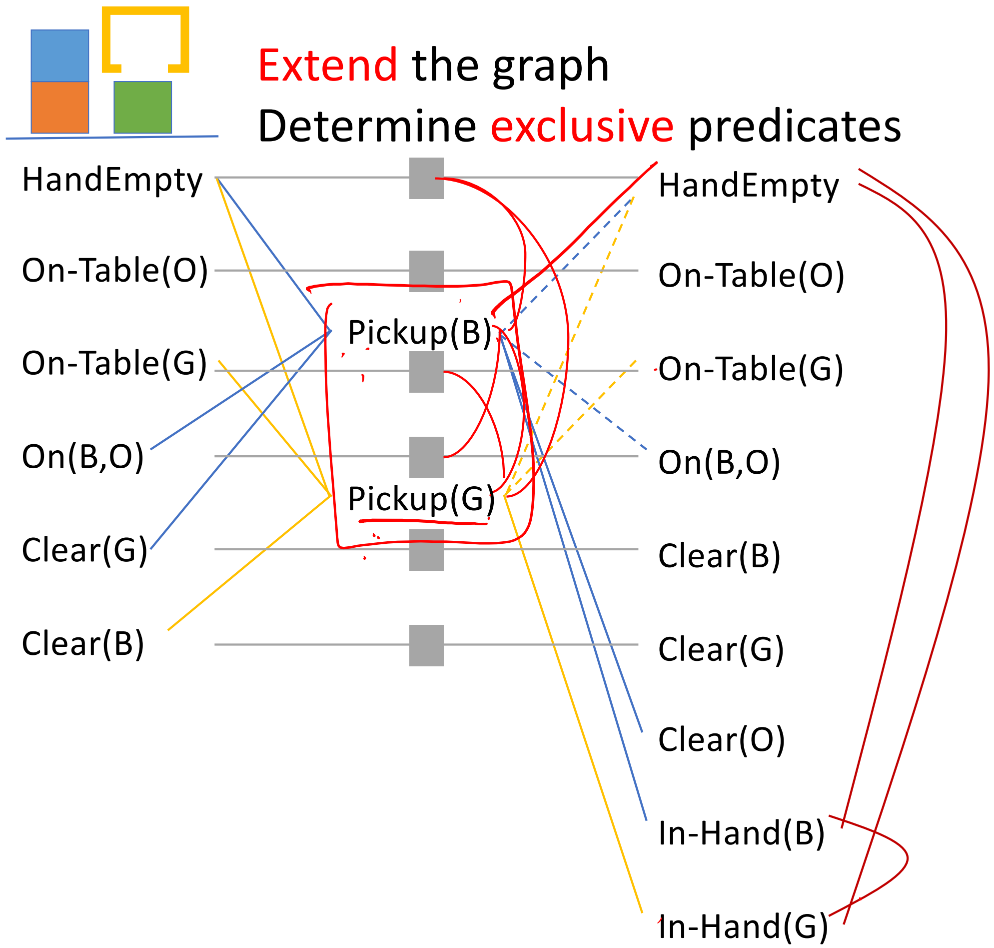
Actions A and B are ***exclusive (mutex)*** at action-level *i*, if:

Interference: one action effect deletes or negates a **precondition** of the other

Inconsistency: one action effect deletes or negates the **effect** of the other

Competing Needs: the actions have preconditions that are mutex in proposition-level *i* – 1

Building a GraphPlan Graph



Actions A and B are ***exclusive (mutex)*** at action-level *i*, if:

- **Interference**: one action effect deletes or negates a **precondition** of the other
- **Inconsistency**: one action effect deletes or negates the **effect** of the other
- **Competing Needs**: the actions have preconditions that are mutex in proposition-level *i* – 1

Propositions P and Q are ***exclusive (mutex)*** at action-level *i*, if:

- **Negation:** They are the negation of each other
- **Inconsistent Support**: **all** actions that produce those propositions at i-1 are mutex

Searching the GraphPlan Graph

Extend until first time all goals are present at a proposition level

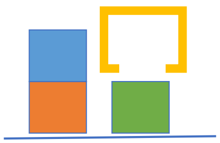
- May not be a solution, at that level

For each goal (predicate) present (in some arbitrary order)

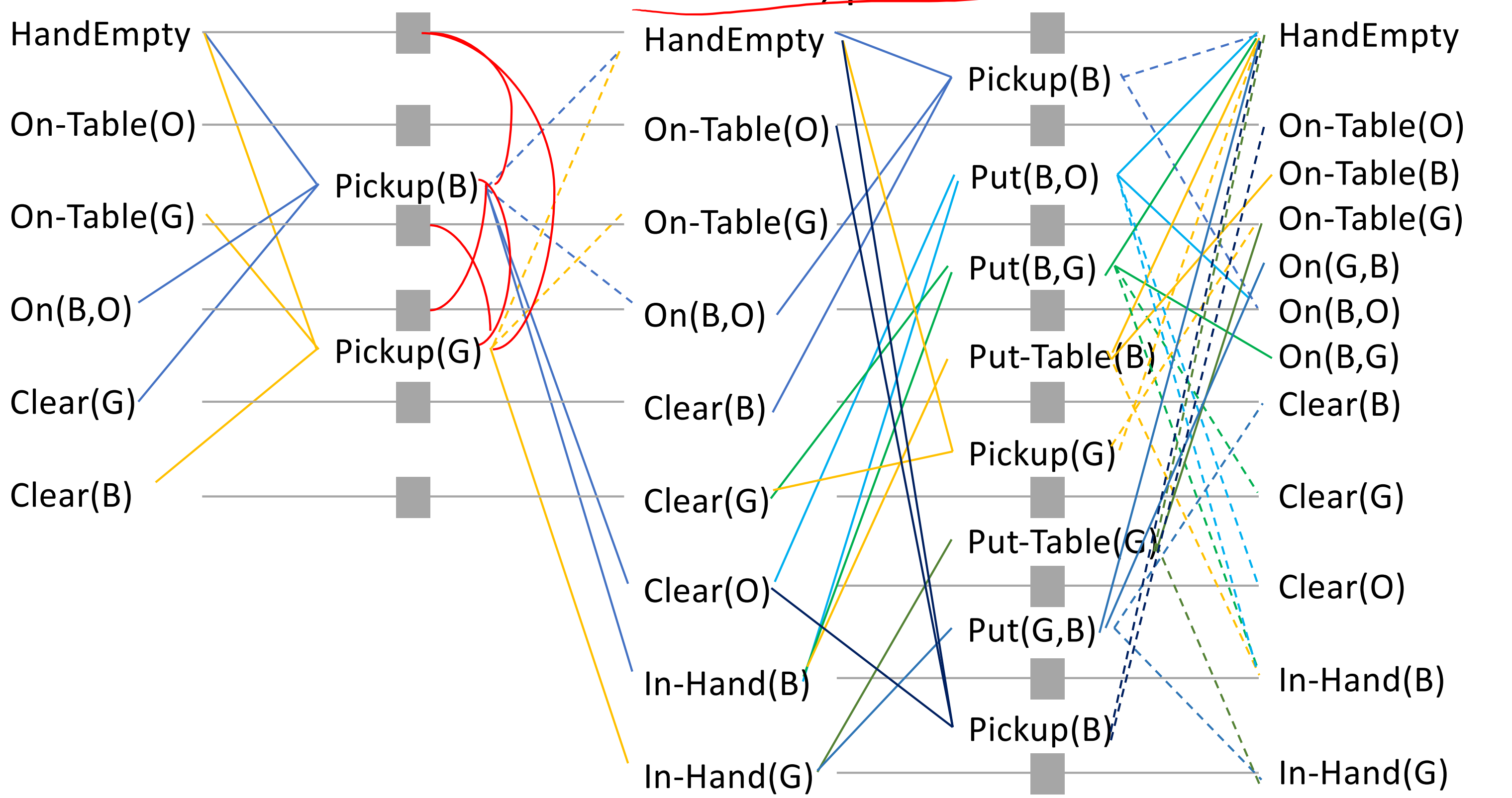
- Select an action at level *i-1* that achieves that goal and is not exclusive with any other action already selected at that level
- Add all its preconditions to the set of goals at level *i-2*
- Do this for all the goals at level *i*
  - Use already selected actions, when possible
- Backtrack if no non-exclusive action exists

If search is exhausted, extend planning graph one more proposition level

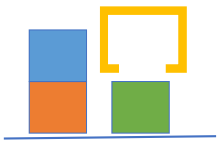
Building a GraphPlan Graph



Extend the graph
Determine exclusive actions, predicates

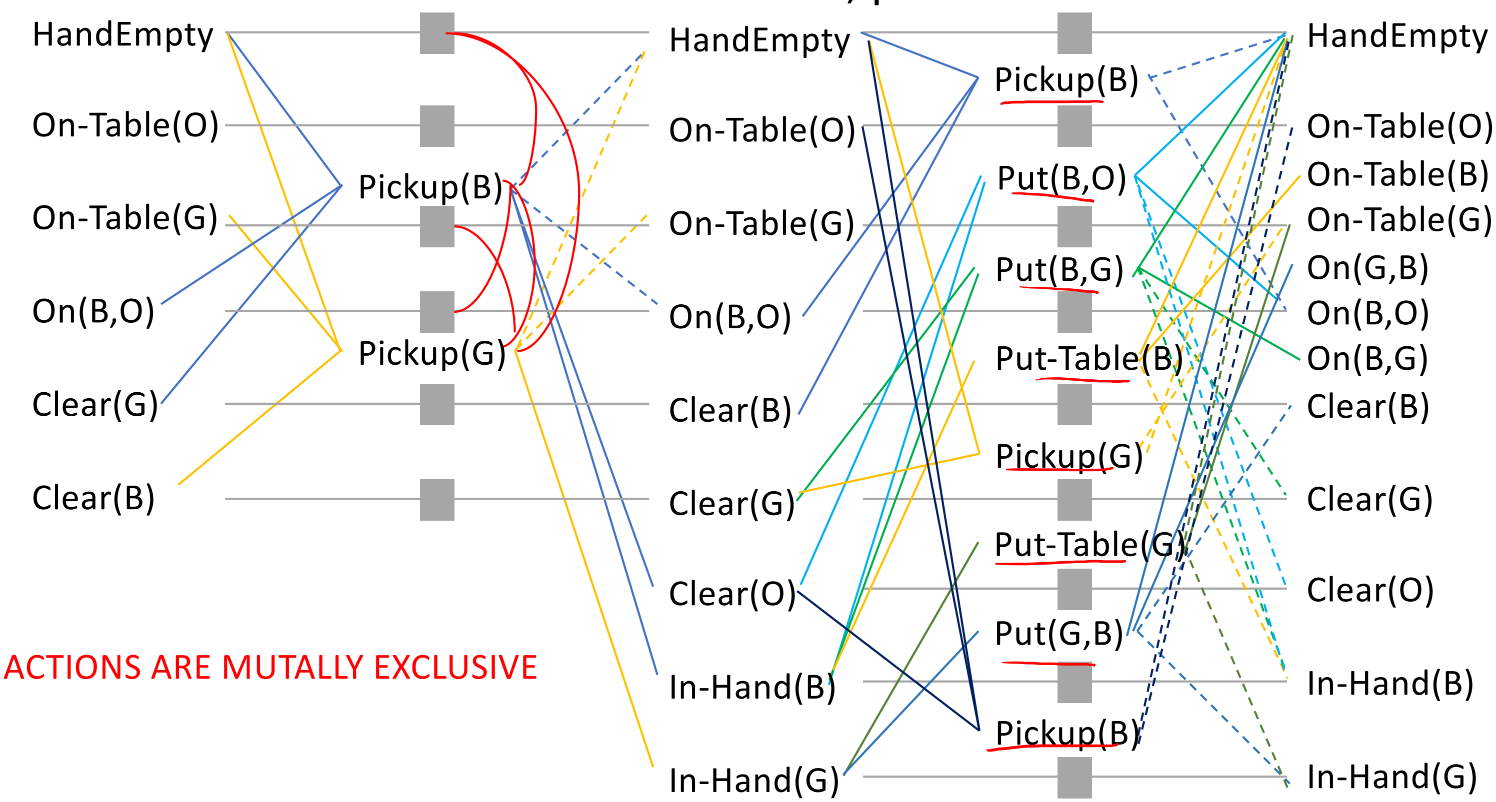


Building a GraphPlan Graph



Extend the graph

Determine exclusive actions, predicates



ALL 2nd LEVEL ACTIONS ARE MUTALLY EXCLUSIVE

Piazza Poll 3

What kind of mutex are actions to each other? (select all that apply)

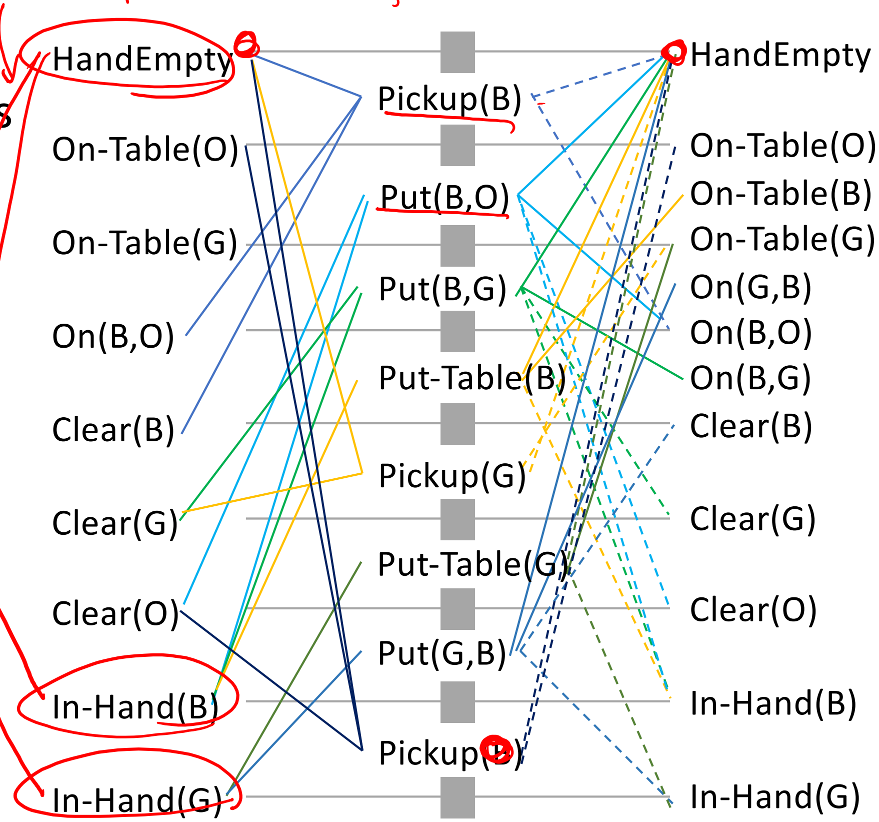
1) Pickup-pickup are interference
2) Pickup-pickup are inconsistency ?
3) Pickup-pickup are competing needs
4) Pickup-put are interference
5) Pickup-put are inconsistency
6) Pickup-put are competing needs

Actions A and B are ***exclusive (mutex)*** at action-level i, if:

- **Interference**: one action effect deletes or negates a **precondition** of the other
- **Inconsistency**: one action effect deletes or negates the **effect** of the other
- **Competing Needs**: the actions have preconditions that are mutex in proposition-level $i - 1$

mutex preconditions

dashes are deletes



GraphPlan Takeaways

GraphPlan is a relaxation of other classical planning search techniques like BFS

It creates a different kind of graph that allows you to decide that **no plan is reachable at a given depth**.

If it finds a **reachable** solution, it may not be a feasible solution because it **allows you to perform multiple actions at the same time**.

The search graph is linear space in the number of predicates

Know the differences between the mutex conditions!!

Implementing Symbolic Representations

Literals: Each thing/object in our model

i = Instance("name",TYPE)

Inky Pink Clyde

Variables: Can take on any TYPE thing

v = Variable("v_name",TYPE)

GHOST

Block World Example:

Pickup_from_Table(b):

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

Add: In-Hand(b)

Delete: HandEmpty(), On-Table(b)

Instances: "A", "B", "C" of type BLOCK
Variable: "b" of type BLOCK
In this operator, b can take on the value of any block instance

Implementing Symbolic Representations

Literals: Each thing/object in our model

i_a = Instance("A",BLOCK), i_b = Instance("B",BLOCK")

Variables: Can take on any TYPE thing

b = Variable("b",BLOCK)

ALERT: no two literals nor variables can have the same string name!!

Block World Example:

Pickup_from_Table(b):

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

Add: In-Hand(b)

Delete: HandEmpty(), On-Table(b)

Implementing Symbolic Representations

Literals: Each thing/object in our model

i_a = Instance("A",BLOCK), i_b = Instance("B",BLOCK")

Variables: Can take on any TYPE thing

integer 1

v_block = Variable("b",BLOCK)

i_1 = I("1",INT)

Propositions: Predicate Relationships

Proposition("relation", v_a, i, ...)

ALERT: variables and instances do not have to start with i_ and v_

Block World Example:

HandEmpty(), Clear(b), On-Table(b), On-Block(b1,b2)

Proposition("handempty"), Proposition("clear",v_block), Proposition("on-table",v_block), Proposition("on-block",v_block, i_a)

Initial State and Goal State

Create lists of Propositions as the initial state and goal state

initial = [Proposition(“handempty”), Proposition(“on-table”, i_c), Proposition(“on-table”, i_b), Proposition(“on-block”, i_a, i_b), Proposition(“clear”, i_a), Proposition(“clear”, i_c)]

Goal = [Proposition(“on-table”,i_b), Proposition(“on-table”,i_c), Proposition(“on-block”,i_a, i_c), Proposition(“clear”,i_a), Proposition(“clear”,”c”)]

Implementing Symbolic Representations

Operators: the actions we take change state

```
pickup_table = Operator("pick_table", #name
                        [Proposition("handempty",), #preconditions
                         Proposition("clear", v_block),
                         Proposition("on-table", v_block)],
                        [Proposition("in-hand", v_block)], #add effects
                        [Proposition("handempty"), #delete effects
                         Proposition("on-table", v_block]
                        )
```

Lists are conjunctions!

All propositions with a variable must take on the same instance!

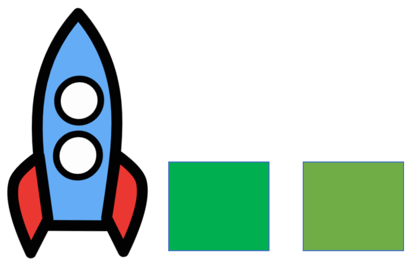
Variables that don't match name don't have to be the same but can be unless otherwise specified!

We provide the GraphPlan implementation

You will create the representation, which will be passed into our GraphPlan implementation

Another Example - Rocket Ship

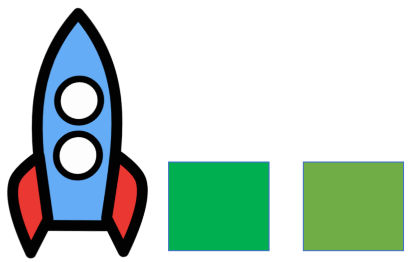
Suppose we have a rocket ship that can only be used once.
It has to carry two payloads.



Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once.
It has to carry two payloads.

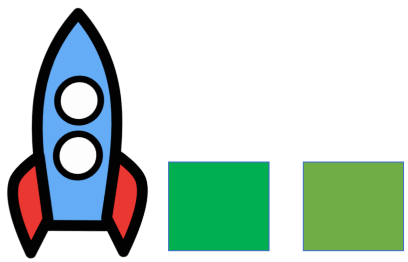
Literals?



Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once.
It has to carry two payloads.

Literals: Rocket, G, O, LocA, LocB

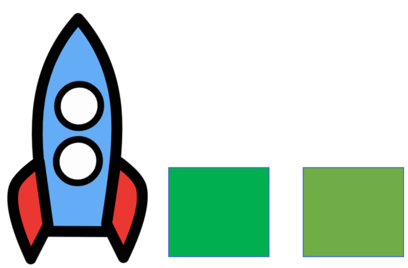


Another Example - Rocket Ship

Suppose we have a rocket ship that can only be used once.
It has to carry two payloads.

I create literals and variables as I go through the problem. In order to create the start state and the goal state, I need the literals defined.

Literals: Rocket, G, O, LocA, LocB
Start state:
At(Rocket, LocA), Has-Fuel(),
Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)



Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB

Start state:

At(Rocket, LocA), Has-Fuel(),
Unloaded(G,LocA), Unloaded(O,LocA)

Goal state:

At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

As I create my operators, I will add variables.

Move: Load: Unload:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

Move:
P:
A:
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
At(Rocket, LocA), Has-Fuel(),
Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L

The rocket starts at a location, and it could be either location. I need to add a location variable

Move:
P: At(Rocket,L)
A:
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
  At(Rocket, LocA), Has-Fuel(),
  Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
  At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L

Move:
P: At(Rocket,L), Has-Fuel()
A:
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

The rocket needs to go to a destination, which needs to be different from the start location. We need to define a dest variable.

Move:
P: At(Rocket,L), Has-Fuel(), L!=Dest
A: At(Rocket,Dest)
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Move:
P: At(Rocket,L), Has-Fuel(), L!=Dest
A: At(Rocket,Dest)
D: Has-Fuel(),At(Rocket,L)

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest

Load:
P:
A:
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
At(Rocket, LocA), Has-Fuel(),
Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest, Pkg

The rocket needs to load a specific package G or O. The load action doesn't care which package it is. We need a variable pkg to use.

Load:
P: At(Rocket,L), Unloaded(Pkg,L)
A:
D:

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB
Start state:
    At(Rocket, LocA), Has-Fuel(),
    Unloaded(G,LocA), Unloaded(O,LocA)
Goal state:
    At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)
Variables: L, Dest, Pkg

Load:
P: At(Rocket,L), Unloaded(Pkg,L)
A: Loaded(Pkg,Rocket)
D: Unloaded(Pkg,L)

Another Example - Rocket Ship

Literals: Rocket, G, O, LocA, LocB

No new variables needed for unload.

Start state:

At(Rocket, LocA), Has-Fuel(),
Unloaded(G,LocA), Unloaded(O,LocA)

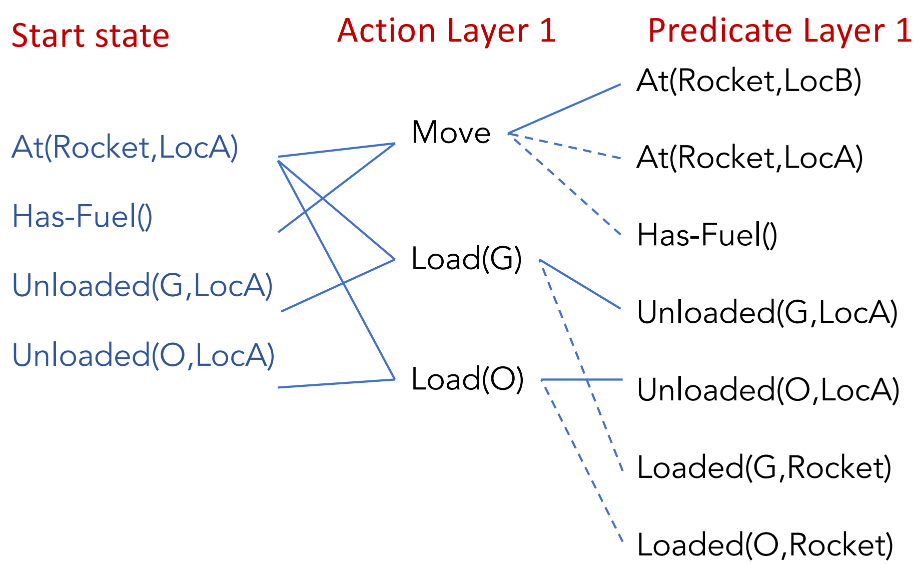
Goal state:

At(Rocket, LocB), Unloaded(G,LocB), Unloaded(O,LocB)

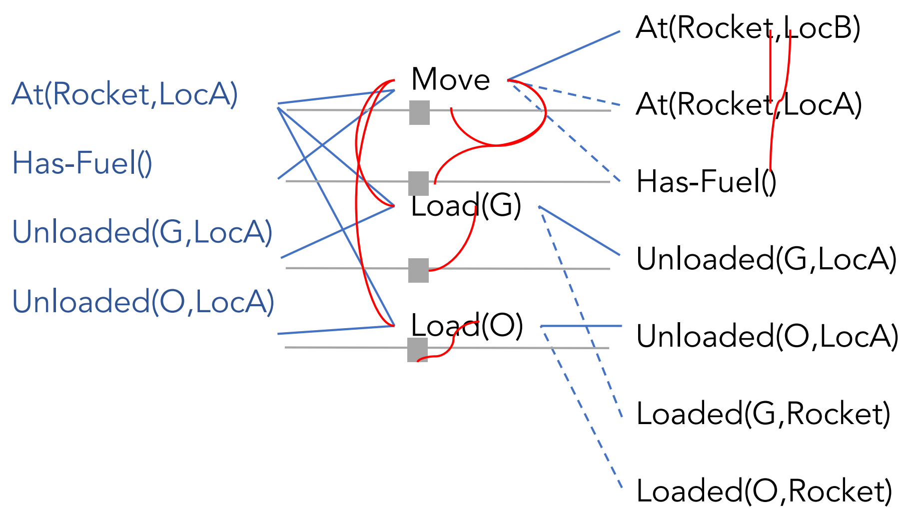
Variables: L, Dest, Pkg

Unload:
P: At(Rocket,Dest), Loaded(Pkg,Rocket)
A: Unloaded(Pkg,Dest)
D: Loaded(Pkg,Rocket)

Rocket Ship GraphPlan Graph



Rocket Ship GraphPlan Graph



Mutex Actions

Interference:

Move deletes At which is a precondition of Load

Inconsistent:

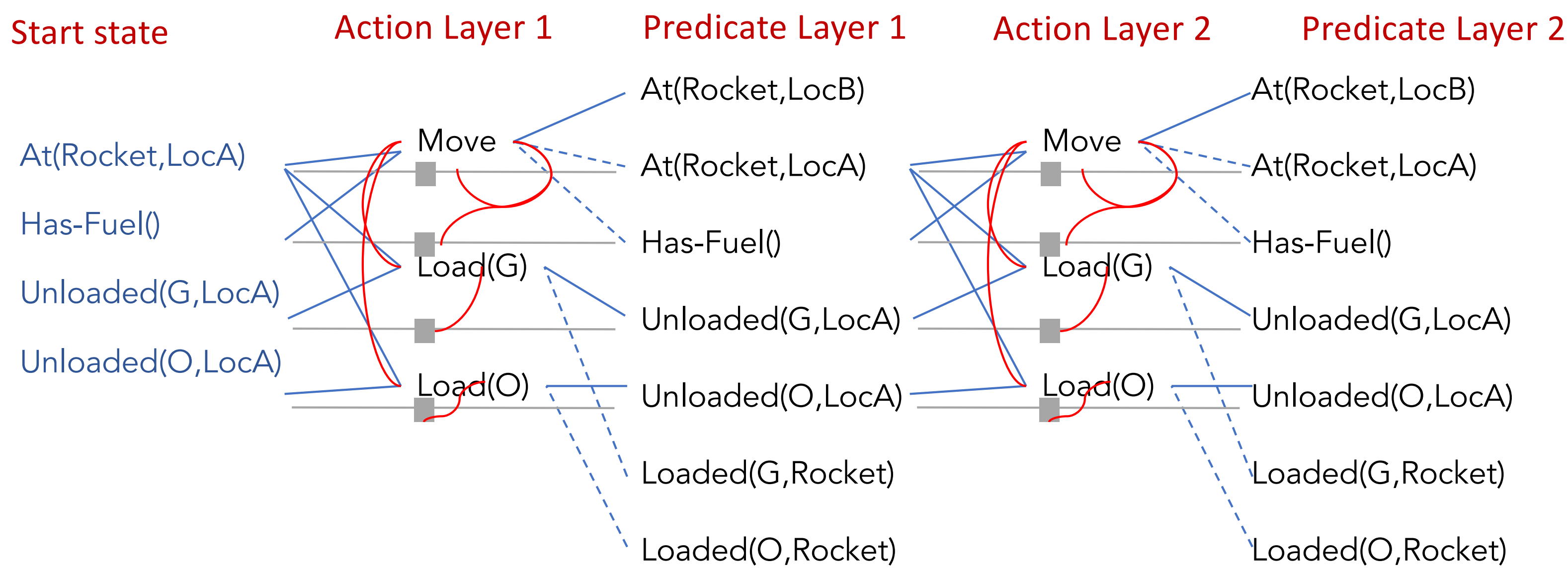
Move deletes At but noop adds it

Move deletes Has-Fuel but noop adds it

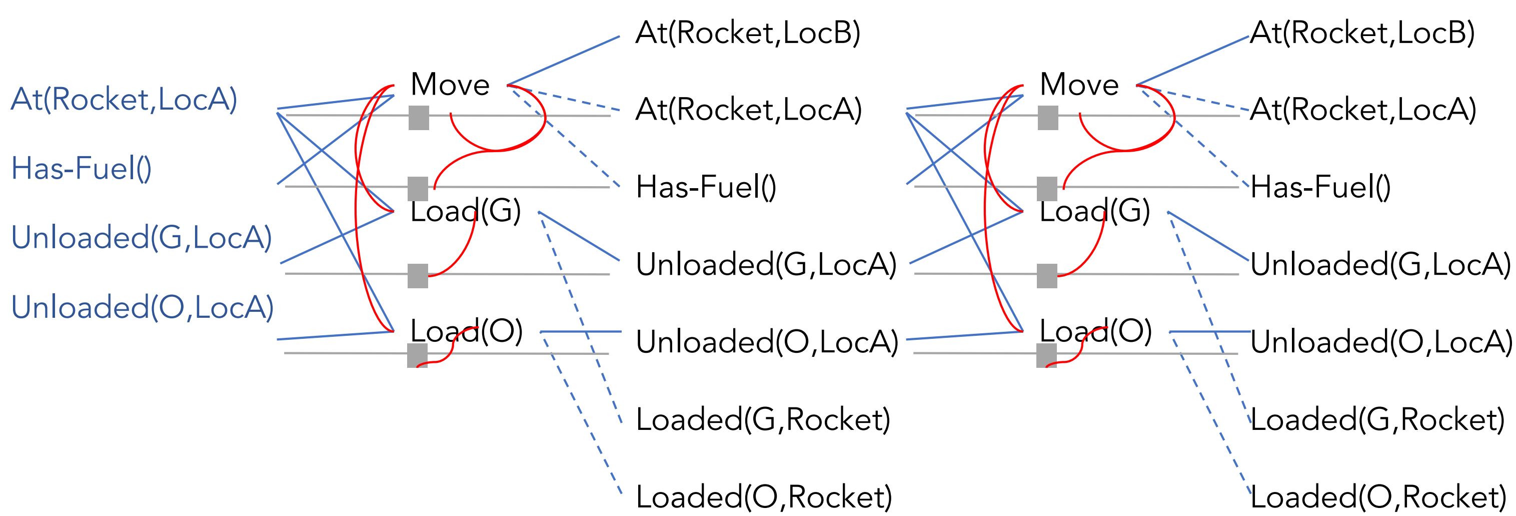
Mutex Propositions:

- At(Rocket,LocB) and At(Rocket,LocA) because Move and noop are mutex actions
- What else?

Rocket Ship GraphPlan Graph



Rocket Ship GraphPlan Graph



At time 1: Move can be performed OR both Load actions
At time 2: Possible plans include:

Load(G), Load(O), Move(LocB) ← reachable goal in two steps but feasible in three
Load(G), Move(LocB)
Load(O), Move(LocB)