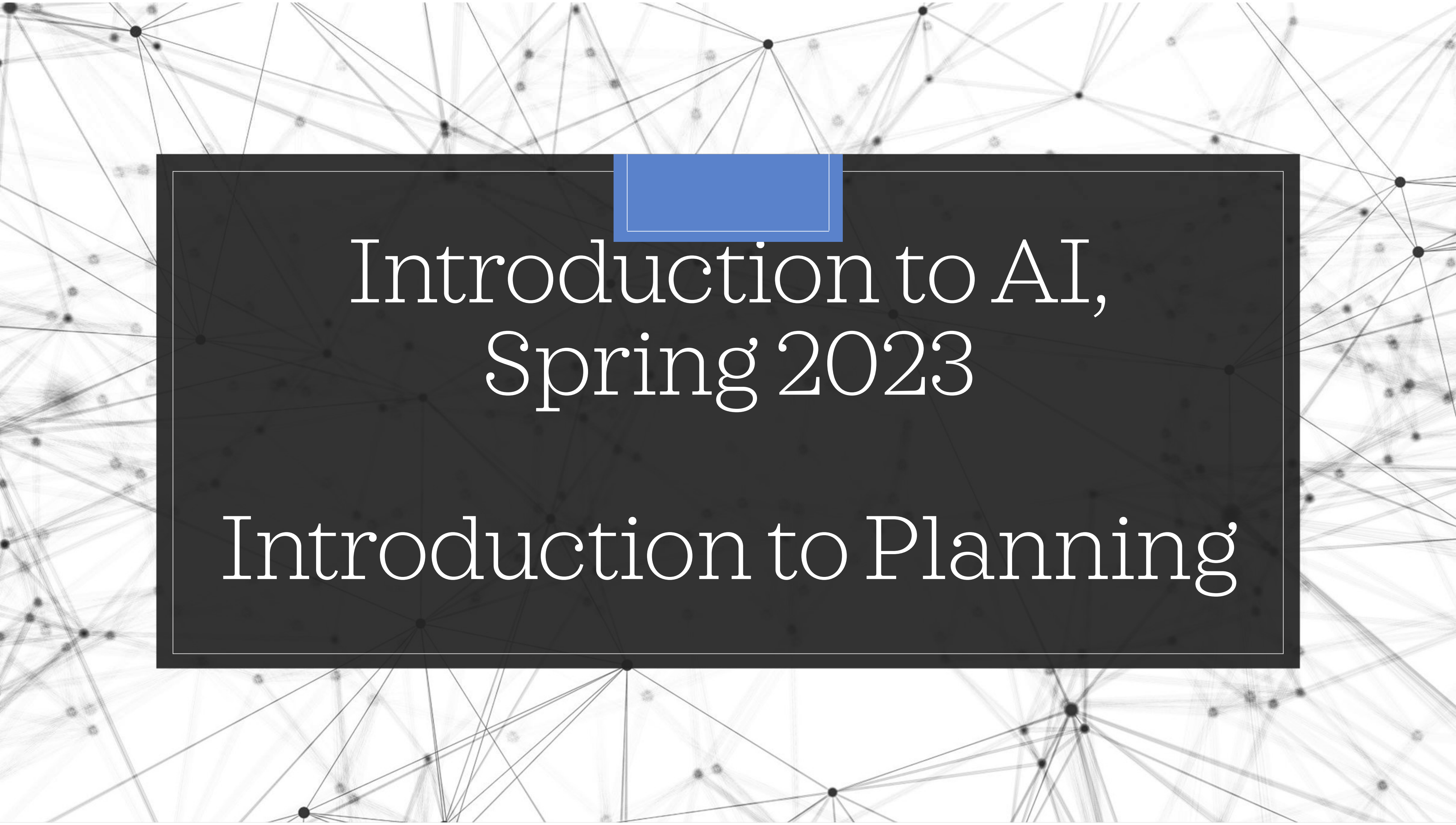


Introduction to AI, Spring 2023

Introduction to Planning

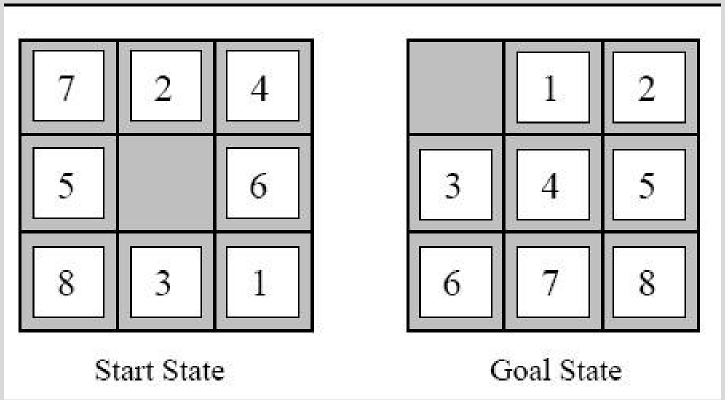
We’ve come so far! Look at all we’ve learned!

- One lesson:

Many different types of problems can be solved by search!

We’ve come so far! Look at all we’ve learned!

Lesson one: Search is Useful



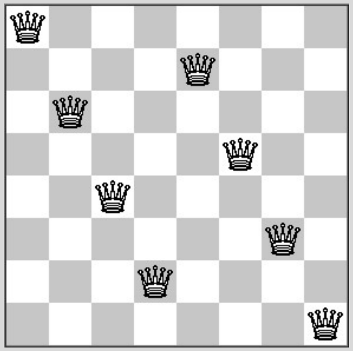
Start State

7	2	4
5		6
8	3	1

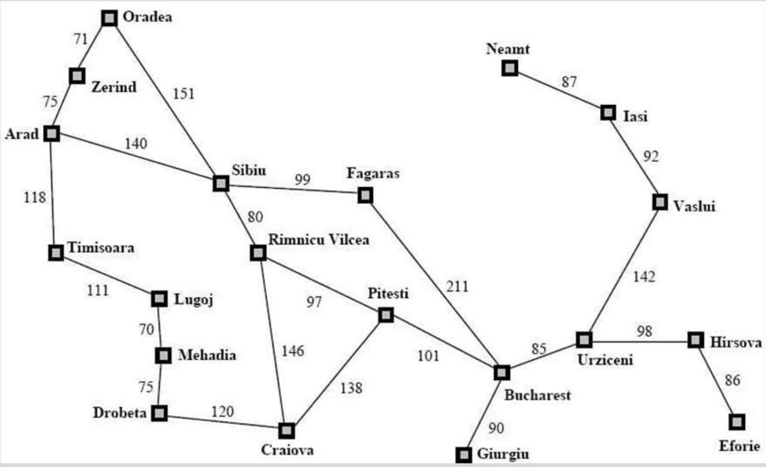
Goal State

	1	2
3	4	5
6	7	8

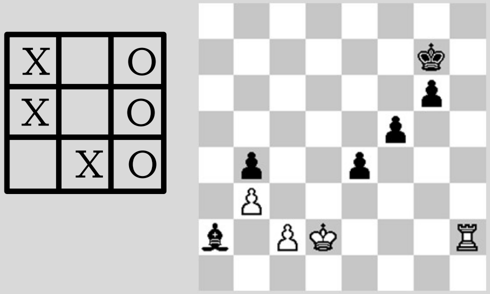
8-Puzzle



8-Queens

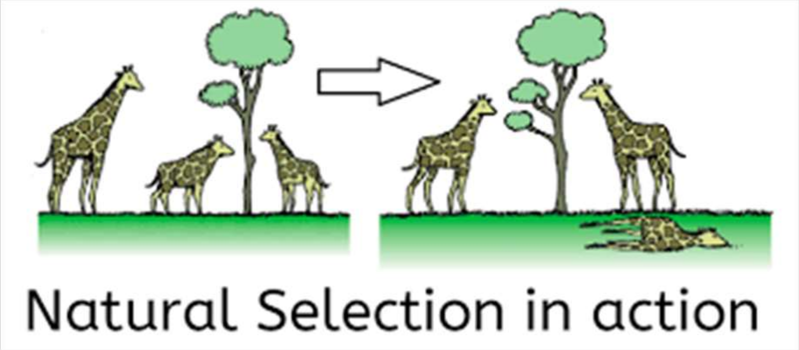


Navigation



X		O
X		O
	X	O

Game Theory Problems



Keeping Giraffes Alive

We’ve come so far! Look at all we’ve learned!

- Second Lesson:

Search needs things to work! It needs to know:

What a **state** is…

What **actions** can take us from one state to another…

What a **goal** state looks like

We’ve come so far! Look at all we’ve learned!

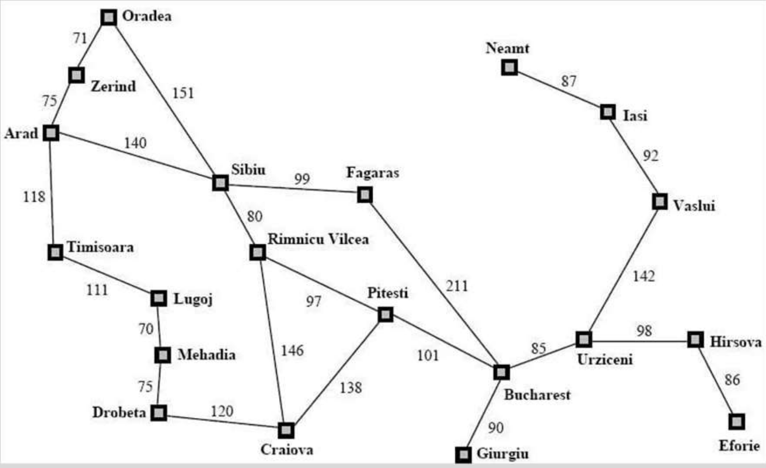
Lesson two: Search needs things

So, taking Navigation, for instance…

A **state** here is the current city you are in.

The **action** you can take is “travel to” (where you can only travel to cities that are adjacent to your current city).

The **goal** is the city you’d like to be in.



Navigation

We’ve come so far! Look at all we’ve learned!

- Third Lesson:

State spaces get big fast!

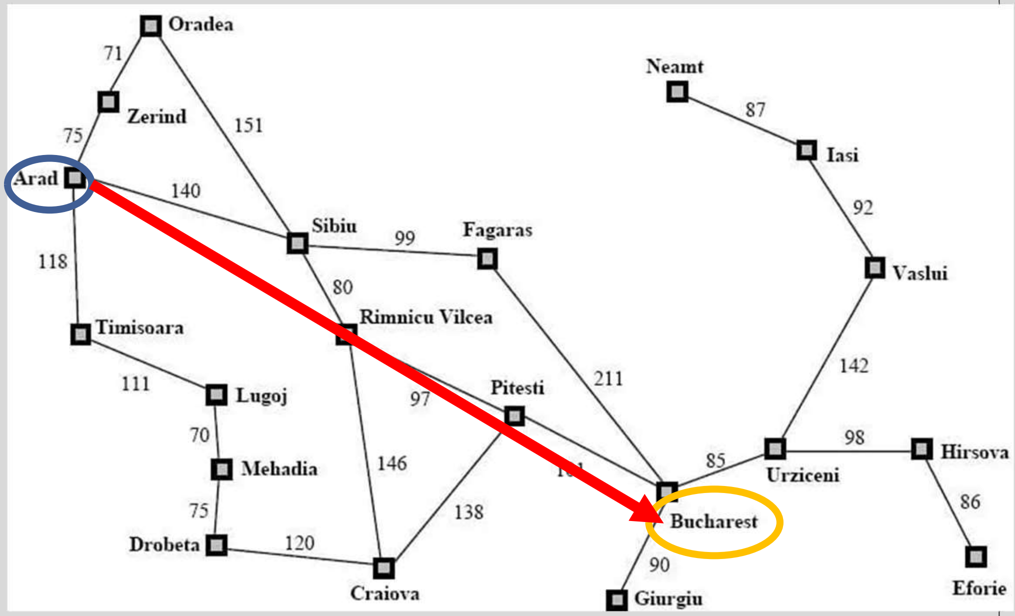
We need accurate heuristics and tricks to explore them efficiently.

We’ve come so far! Look at all we’ve learned!

Lesson three: Search relies on heuristics for efficiency.

Don’t just search randomly, that could take forever, and you’d have to visit a lot of “needless” states!

Have your goal in mind when making decisions!



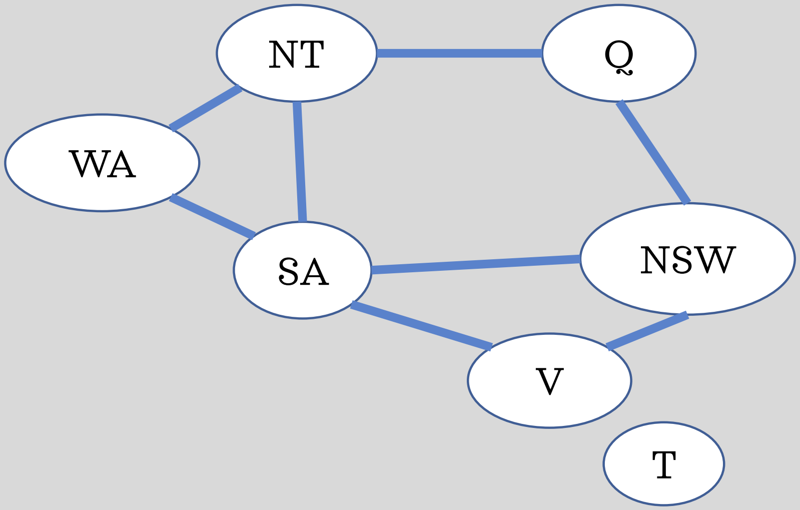
We’ve come so far! Look at all we’ve learned!

◦ Fourth Lesson:

Domain-Independent heuristics require **common representations** for all problems

We’ve come so far! Look at all we’ve learned!

Lesson four: We like common representations of problems



Like Constraint Satisfaction Problems!

Vars:	{WA, NT, SA, Q, NSW, V, T}

Domains:	WA = {red, green, blue},
	NT = {red, green, blue}, etc.

Constraints:	{WA != NT, WA != SA, etc.}

Knowledge Base:

$$BA1 \leftrightarrow PA2 \vee PB1$$
$$BB1 \leftrightarrow PB2 \vee PC1$$

Knowledge Base in CNF:

$$\neg BA1 \vee PA2 \vee PB1B$$
$$A1 \vee \neg PA2$$
$$BA1 \vee \neg PB1$$
$$\neg BB1 \vee PB2 \vee PC1$$
$$BB1 \vee \neg PB2$$
$$BB1 \vee \neg PC1$$

Or like representing logic problems in conjunctive normal form!

We’ve come so far! Look at all we’ve learned!

◦ All these lessons:

1. Many different types of problems can be solved by search!
2. Search needs to know what states are, what actions are, and what goals are.
3. State spaces get big fast -- We need accurate heuristics to explore them!
4. Domain-Independent heuristics require common representations for all problems

I know we’ve done this already, but once again we’re going to remind you just how crazy big search spaces can get…

Wumpus World Revisited.

- We all love Wumpus World.
- But, I confess, we've been living in a little bit of a lie.
- We've been glossing over the concept of **time**.

Wumpus World Revisited.

- Most aspects of Wumpus World are **atemporal**.
  - That is, not affected by time.

- E.g., either there is a pit in C2 or there isn’t.
  - For any given Wumpus World, either C2 *always* has a pit, or it *never* has a pit.

- But there are some aspects of Wumpus World that *do* change over time.

Wumpus World Revisited.

- Some of those time-sensitive variables include:
- The player location!
  - At time step 0: at A1
  - At time step 1: at A2
- If they still have their arrow!
  - At time step 0: Have Arrow
  - At time step 1: Have Arrow
  - At time step 2: Doesn’t have Arrow.
    - Implies that they shot their arrow as their “time step 1” action.

Wumpus World Revisited—Representing Time.

- A variable whose value changes over time is called a **fluent**.
- Up until now, we’ve managed to avoid dealing with fluents by essentially *only reasoning about the present.*
  - We can do this with some clever knowledge base manipulation.

Wumpus World Revisited—Representing Time.

- For example…

Add fact to knowledge base: Location A1

Then the player moves…

Remove Fact: Location A1
Add Fact: Location B1
Add Fact: Breeze B1

Wumpus World Revisited—Representing Time.

- When we treat the world like this, it let’s us deduce new facts about the present just fine!
- **But it prevents us from reasoning about the future.**
  - In Wumpus World, that’s *mostly ok* because with most things in it, if it is true in the present, it will remain true in the future.
    - Pits don’t move
    - Breezes don’t move
    - Chests don’t move
    - Stenches don’t move.
    - Wumpus doesn’t move.

Wumpus World Revisited—Representing Time.

- But in *real life*, the world is *overwhelmingly* full of fluents.

- States change over time constantly!

- As you, and other agents/people do things…
- And as, perhaps, your very environment changes itself…
- You want to be able to adapt!
- And not just to *respond* to these changes, but for your agent to *anticipate* likely outcomes based on all these moving parts!

Wumpus World Revisited—Representing Time.

- OK. So – representing time / including time in our reasoning seems like a nice thing to have. How to do it?
- One way: Discretize it into time-steps, and mark fluents (i.e., variables/facts that care about time) based on when they happened.

At A1 at time 0

Then the player moves…

At A1 at time 0
At B1 at time 1
Feel Breeze at time 1

Wumpus World Revisited—Representing Time.

◦ Drawing attention to a couple of small details here…

At A1 at time 0

Then the player moves…

We didn't remove this fact like before! We forever remember what life was like at time 0!

At A1 at time 0

At B1 at time 1

Feel Breeze at time 1

There being a breeze in B1 is *not* a fluent. Not affected by time.

But our agent's percepts *are* fluents. Depend on agent's location, which changes over time. *FEEL*ing the breeze is a fluent!

Wumpus World Revisited—Representing Time.

- So, cool! We did it! We introduced time! That wasn’t so bad…
- OR IS IT?

- Representing all of these discrete timesteps will get the job done, but it ends up adding a *lot* of complexity to the knowledge engineering process.
  - Now you need to write rules along the lines of “if you are in A1 at timestep 0, AND you are facing East at timestep 0 AND you take the “forward” action… NOW you are in B1 at timestep 1 AND you are facing East at timestep 1.
  - And you need to do this for *every* timestep! (easier with first order logic, but still!).

Wumpus World Revisited—Representing Time.

- It also explodes the number of **ground** facts!
  - i.e., facts that aren’t looking at variables, but actual constants/values.

- And what if time is not bound? i.e., you don’t limit the number of time steps that you have?

- Then the number of ground facts becomes infinite! You need to write rules like the one on the previous slide, for each timestep t from 0 to, well, infinity!
  - Impossible without quantifiers!

Wumpus World Revisited—Representing Time.

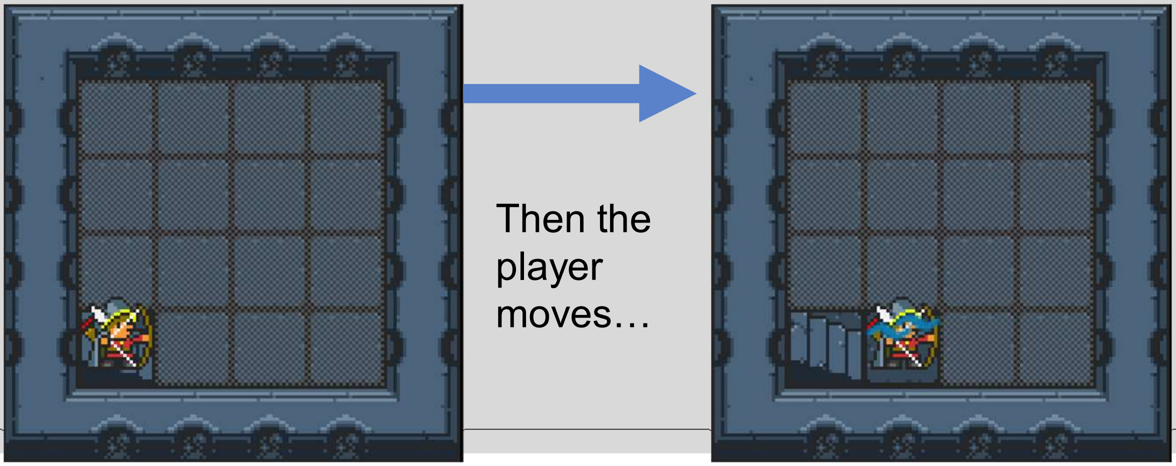
- And that sample rule that we just saw?
- “if you are in A1 at timestep 0, AND you are facing East at timestep 0 AND you take the “forward” action… NOW you are in B1 at timestep 1 AND you are facing East at timestep 1.”
- Pretty complicated huh?
- BUT IT WASN’T COMPLICATED ENOUGH!

The Frame Problem

- That rule did a good job of capturing everything that changed…
- But, sadly, it didn't do a great job of capturing what stayed the same.

"if you are in A1 at timestep 0, AND you are facing East at timestep 0 AND you take the "forward" action… NOW you are in B1 at timestep 1 AND you are facing East at timestep 1."

Time 0: at A1
Time 0: Have arrow
Time 0: Facing East



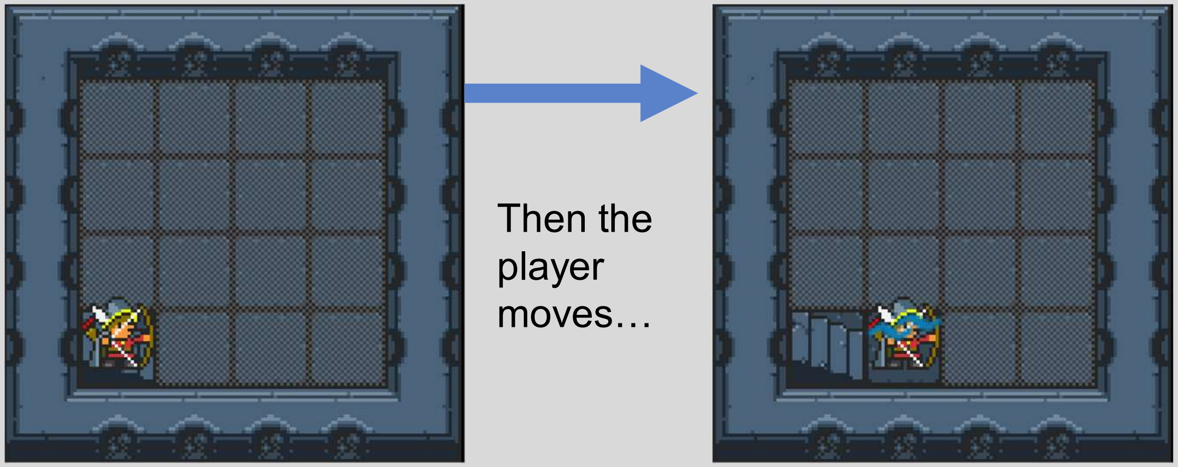
Then the player moves…

T0 : At A1
T0: Have Arrow
T0: Face East
T1: At B1
T1: Face East
T1: Feel Breeze

The Frame Problem

- Our rule didn't capture what "stayed the same", i.e., it didn't say anything about the arrow state!

Time 0: at A1
Time 0: Have arrow
Time 0: Facing East



Then the player moves…

T0 : At A1
T0: Have Arrow
T0: Face East
T1: At B1
T1: Face East
T1: Feel Breeze

So now the "have arrow" fact is entirely missing from Timestep 1.
Which basically means that we lost our arrow.

The Frame Problem

- This is known as the **frame problem**: when the results of actions fail to capture what remains *unchanged*.
- One approach: write a lot of rules known as **frame axioms** that *do* capture what doesn't change.
  - In propositional Logic: $\mathrm{Forward}^{t} \rightarrow (\mathrm{HaveArrow}^{t} \leftrightarrow \mathrm{HaveArrow}^{t+1})$
    - Writing a new line for each t!
  - In First Order Logic: $\forall t\ arrow(t_i)\wedge forward(t_i)\rightarrow arrow(t_{i+1})$
    - Still need to write lines like that for each possible action.
    - AND need to write a similar set of rules for each time sensitive predicate.

The Frame Problem

- So even this frame axiom approach is gnarly and complicates the amount of “set up” rules you need to write to capture the world.

- All of this is said to convince you: thinking about time is tough.

- But! As we already discussed! Thinking about time is important! Many real world problems depend on it!

Planning

- Enter Planning: the next little unit of our class!
- **Planning** is the science of reasoning about a sequence of actions that achieves some goal.
- Planning uses a simple representation of state, actions, and goals to deal with time and the frame problem.
- And Planning uses a logic-like representation of states and actions to allow for domain independent-heuristics.

- And note how nicely it dovetails with everything we just recapped!

The Structure of a Planning Problem

◦ You are given three things:

1. A description of the world and its **initial state**.
2. A set of **action templates**
3. A **goal**

The objective: find a sequence of ground actions that, when taken from the initial state, achieves the goal.

(remember, "ground" means all variables have been bound)

The Structure of a Planning Problem - Actions

- Actions are made up of two things:
- **Preconditions**: that which must be true before the action can happen.
- **Effects**: That which will become true after the action is taken.

What might preconditions and effects be for the action ”Eat Ice Cream”?

Sample preconditions: be in the same room as ice cream, have spoon/bowl.

Sample effects: Ice cream is gone, dishes dirty, you are happy, you have brain freeze.

The Structure of a Planning Problem

- So with that, we can be a little more precise with our formulation…
- You are given:

1. A description of the world and its **initial state**.
2. A set of **action templates**
3. A **goal**

- You want to find:

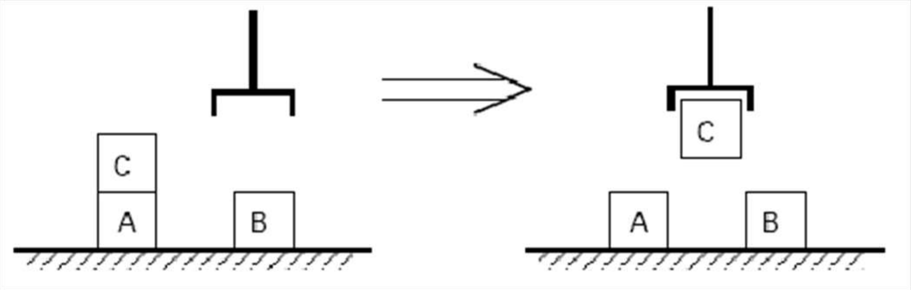
1. A sequence of ground actions…
2. Such that each action’s preconditions are true before you take the action…
3. And that, when all of the actions are taken, you have achieved your goal.

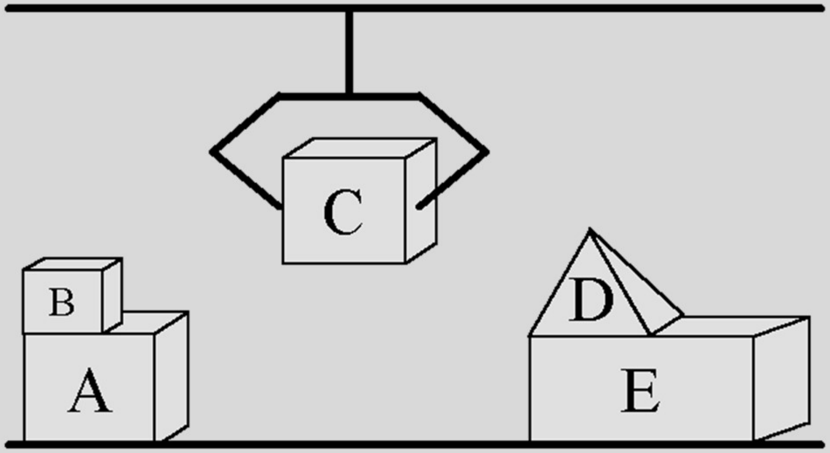
Logical Language

- And to further build on everything that we’ve learned so far:
  - We’ll represent most things…
    - initial state, Goal, Action preconditions, Action effects…
  - Using **conjunctions of function-free predicate logic** (*not* first order logic)
    - That means no quantifiers!
    - Also means no functions!
    - But “relationships” and “qualities” are A-OK.
      - E.g., Brothers(John, Richard) and King(John) are TOTALLY COOL.
      - But LeftLeg(John) is NOT (it is a function)
      - And $\forall t\ arrow(t_i) \wedge forward(t_i) \rightarrow arrow(t_{i+1})$ is NOT either (has a quantifier).

One Other Quick Recap Thing: Microworlds

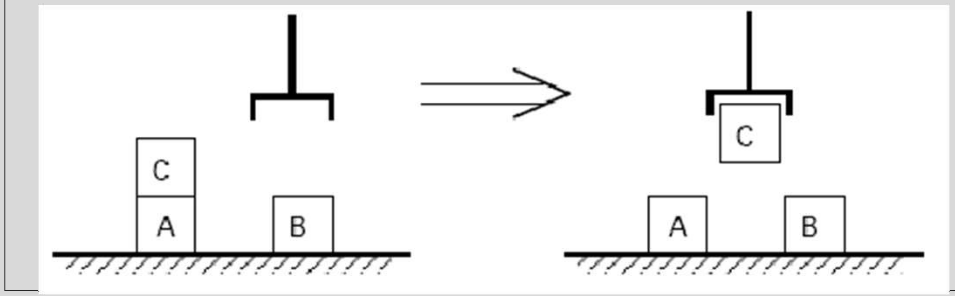
◦ When we first began the semester (on day 2), we introduced the concept of Microworlds with these images:

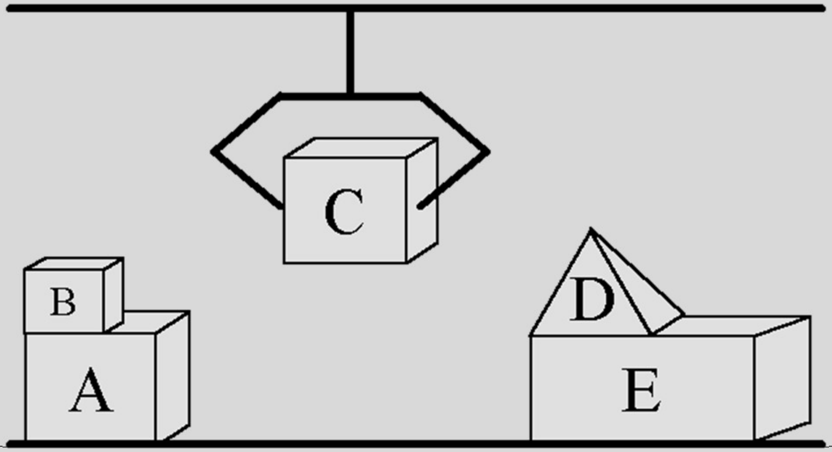




One Other Quick Recap Thing: Microworlds

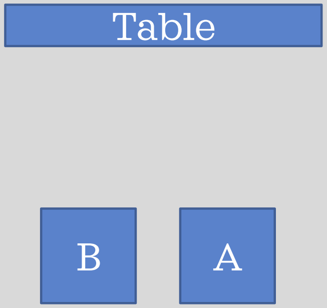
- When we first began the semester (on day 2), we introduced the concept of Microworlds with these images:
    - Vacuum World and Wumpus World are great and everything…
    - But lemme tell you all about **Blocks World**.





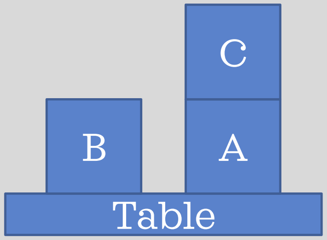
Blocks World

- Like all of our Micro Worlds, there are certain conventions here.
- The first convention is that you have a table.
- The second convention is that you have some blocks.



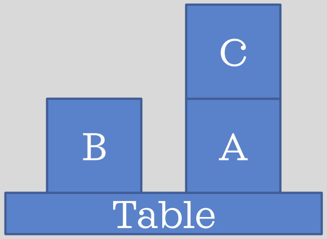
Blocks World

- Blocks can either be on the table…
- Or they can be on top of other blocks
  - Here, Block B and Block A on the Table
  - And Block C is on Block A.
- A block can only be moved when there's nothing on top of it.
  - So here, B and C are movable, but A is not (because C is in the way).



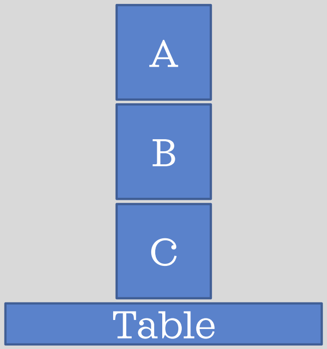
Blocks World

- As we’ve alluded to, the **initial state** is some configuration of these blocks:



Initial State

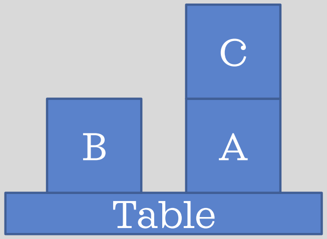
- And a **goal**, then, is some *other* arrangement of blocks that we want!



Goal State

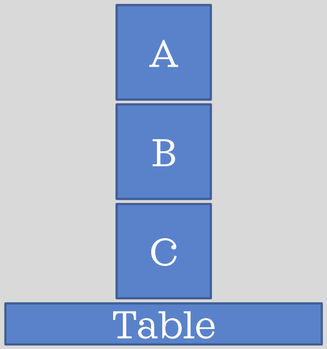
Blocks World: In Predicate Language

- The constant *Table* represents the table.
- And likewise, each block needs a constant as well.
    - Like *A, B*, and *C* for the example on the right.



Initial State

- The predicate *on(b, i)* means that "Block b is on Thing i"
    - Where "Thing" here can be either a table or a block.
    - i.e., a block can either be on the table, or on top of another block.
- We'll also have a predicate *block(b)*
    - Which means "b is a block"



Goal State

Blocks World: In Predicate Language

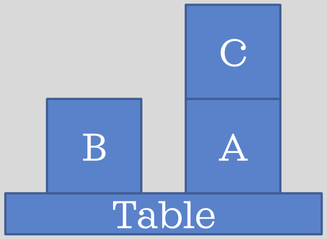
- So we remember: we represent things in planning as a conjunction of these predicates.

- With that in mind, the initial state is:

block(A) ∧ block(B) ∧ block(C) ∧ // A, B and C are blocks.

on(B,Table) ∧ on(A, Table) ∧ //B and A are on the table.

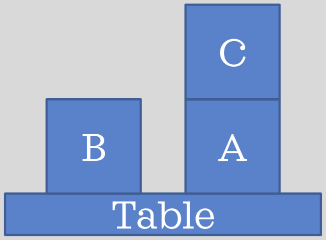
on(C,A) //C is on top of A.



Initial State

Blocks World: Closed World Assumption.

- One other thing: Planning uses the **Closed World Assumption**.
- That just means that facts not explicitly noted as true can safely be assumed to be false.
- E.g., ¬on(B,A) is considered true, even though we didn't explicitly include it in the initial state.



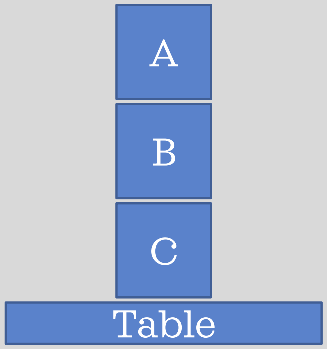
Initial State

Blocks World: Goal

- I'm going to tell you that our goal here is this:

on(A,B) ∧ on(B,C) //A is on B, and B is on C.

- This is a conjunction of things which must be true to be considered "done".
- It is *not* a complete description of the goal state!
  - For example doesn't specify what C is on top of.
  - That means that *it doesn't care* what C is on top of.
- This means that there might be potentially many goal states!
  - "As long as A is on B, and B is on C, I don't care what else is true"



Goal State

Blocks World: Action Template

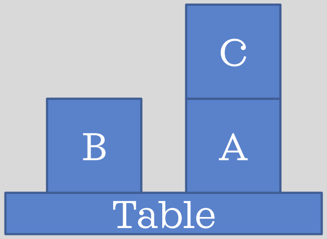
- The action template specifies the kinds of actions we can take.
- Just one action to start with: move

move(b, from, to)

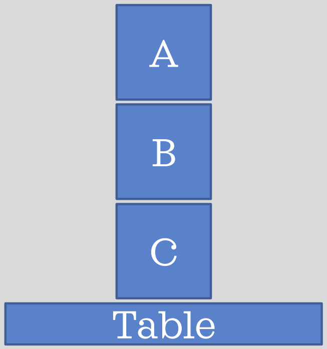
b == the block you want to move.

from == the current location

to == the destination.



Initial State



Goal State

Blocks World: Action Template

- An action is defined by **preconditions** and **effects**.

move(b, from, to)

preconditions: $block(b) \land on(b, from) \land \neg\exists x\ on(x, b) \land \neg\exists y\ on(y, to)$

"b is a block, b is on the origin block, there isn't anything on b, and there isn't anything on the destination block."

Effects: $on(b, to) \land \neg on(b, from)$

"b is now on the destination block, and b is no longer on the origin block"

Note that we had to specify the things that are no longer true, i.e. $\neg on(b, from)$

Blocks World: Action Template

move(b, from, to)

preconditions: $block(b) \land on(b, from) \land \neg\exists x\ on(x, b) \land \neg\exists y\ on(y, to)$

Effects: $on(b, to) \land \neg on(b, from)$

- But there's a problem here…
- We aren't allowed to use quantifiers! How can we compensate?
- We'll invent a new predicate: *clear(i)* means that thing i has nothing stacked on it.

Blocks World: Action Template

move(b, from, to)

~~**preconditions**: $block(b) \land on(b, from) \land \neg\exists x\ on(x, b) \land \neg\exists y\ on(y, to)$~~

preconditions: $block(b) \land on(b, from) \land clear(b) \land clear(to)$

Effects: $on(b, to) \land \neg on(b, from)$

- Great! Before we can move a block, we need to make sure that there’s nothing on the block we are moving [$clear(b)$] and nothing on the block we are moving it to. [$clear(to)$]
- That cleans up the “obvious” problem… but now there’s a little bit of extra book-keeping on the Effects end as well…

Blocks World: Action Template

move(b, from, to)

~~**preconditions**: $block(b) \wedge on(b, from) \wedge \neg\exists x\ on(x, b) \wedge \neg\exists y\ on(y, to)$~~

preconditions: $block(b) \wedge on(b, from) \wedge clear(b) \wedge clear(to)$

~~**Effects**: $on(b, to) \wedge \neg on(b, from)$~~

Effects:$on(b, to) \wedge \neg on(b, from) \wedge \neg clear(to) \wedge clear(from)$

- The destination block is no longer clear [$\neg clear(to)$] because we just put a block on it, but the origin block now *is* clear [$clear(from)$] because we just took whatever was on it off.

Blocks World: Action Template

move(b, from, to)

preconditions: $block(b) \wedge on(b, from) \wedge clear(b) \wedge clear(to)$

Effects:$on(b, to) \wedge \neg on(b, from) \wedge \neg clear(to) \wedge clear(from)$

- OK... This is looking pretty good... BUT… it is kind of assuming that you are moving a block onto another block… **What if you are moving a block to the table?**
- The solution: just create a second action for this!

Blocks World: Action Template

Action: moveToTable(b, from) // don’t need a “to”, always the same.

preconditions: $block(b) \wedge on(b, from) \wedge clear(b)$

// b is a block, it is on a thing represented by “from”, and nothing is on b.

Effects:$on(b, Table) \wedge \neg on(b, from) \wedge clear(from)$

// the block b is on the table, it is not where it used to be, and where it used to be is now clear.

Great!

Blocks World: Action Template

Action*: moveToTable(b, from)*

preconditions: $block(b) \land on(b, from) \land clear(b)$

Effects:$on(b, Table) \land \neg on(b, from) \land clear(from)$

Action*: move(b, from, to)*

preconditions: $block(b) \land on(b, from) \land clear(b) \land clear(to)$

Effects:$on(b, to) \land \neg on(b, from) \land \neg clear(to) \land clear(from)$

Cool, we have two action templates now! But a Plan is full of **ground actions**. i.e., each **step** in the plan is an action with all of its variables filled in.

Kind of the equivalent of calling a method by passing in arguments for its parameters.

Blocks World: Actions

***(Template) Action**: moveToTable(b, from)*

preconditions: $block(b) \land on(b, from) \land clear(b)$

Effects:$on(b, Table) \land \neg on(b, from) \land clear(from)$

Compare the above to…

(Ground) Action *moveToTable(C,A)*

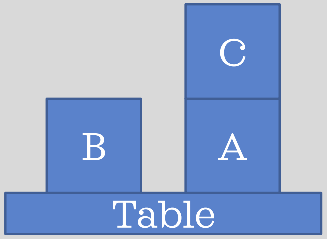
preconditions: $block(C) \land on(C, A) \land clear(C)$

Effects:$on(C, Table) \land \neg on(C, A) \land clear(A)$

We aren't speaking abstractly here now that it is ground!

Preconditions: C is a block, it is on A, and there is nothing on it.

Effects: C is now on the table, it is not on A, and A is clear.



Initial State

Blocks World: Action

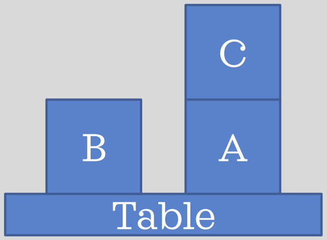
- So now… with all that under our belts…

- It would be nice if we could come up with a plan that achieves our goal!
- **And us, using our smart human brains, can maybe see the answer?**

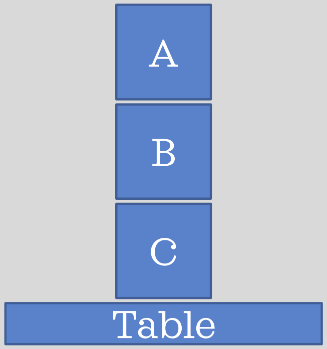
1.) moveToTable(C,A)

2.) move(B,Table,C)

3.) move(A,Table,B)



Initial State



Goal State

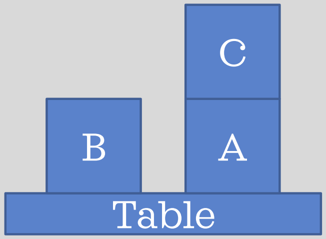
Blocks World: Action

- Right?

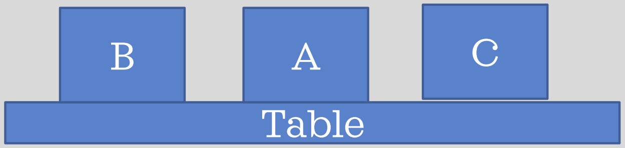
1.) moveToTable(C,A)

2.) move(B,Table,C)

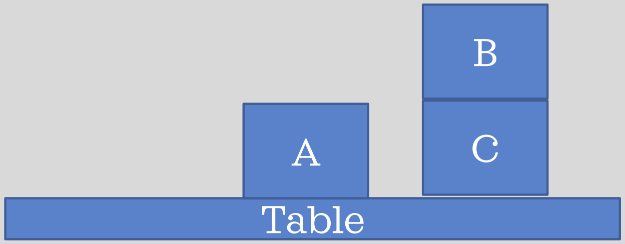
3.) move(A,Table,B)



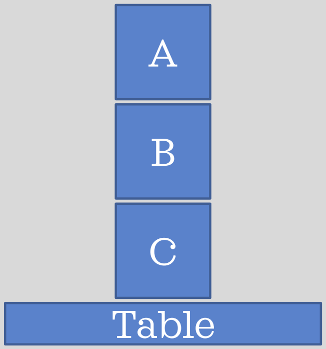
Initial State



After First Action



After Second Action



After Third Action (Goal State)

Blocks World: Action

- This is going to come up more in just a few slides, but just to give you a sneak peek…

- I'm going to start representing "ground preconditions, ground action, and ground effects" like this:

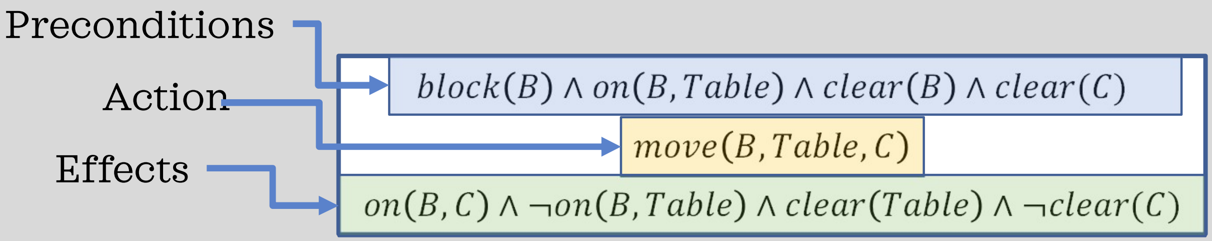
$$block(B) \land on(B, Table) \land clear(B) \land clear(C)$$

$$move(B, Table, C)$$

$$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$$

Blocks World: Action

- Where…
  - The first line are the preconditions
  - The second line is the action taken
  - The third line are the effects after taking that action



Preconditions

Action

Effects

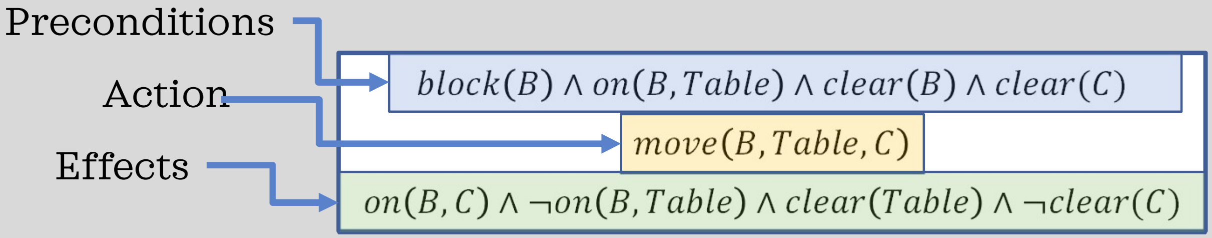
$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Blocks World: Action

- So in order to take the ground action “Move block B from the Table to on top of C” for example…
  - The following must be true before hand: B is a block, B is on the table, B is clear (i.e., nothing on top of it), and C is clear (nothing on top of it either).
  - And after taking the action the following becomes true: B is on C, B is not on the table, the table is clear, and C is not clear.



Preconditions

Action

Effects

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Complexity of Planning

- In Blocks World, we have:
  - 4 Objects (three blocks and the table)
  - 1 action template with 3 parameters *move(b, from, to)*
  - 1 action template with 2 parameters *moveToTable(b, from)*

- *Total number of different ground actions:*

$$4^3 + 4^2 = 64 + 16 = 80$$

Complexity of Planning

- In Blocks World, if instead we had:
  - **5 Objects** (four blocks and the table)
  - 1 action template with 3 parameters *move(b, from, to)*
  - 1 action template with 2 parameters *moveToTable(b, from)*

- *Total number of different ground actions:*

$$5^3 + 5^2 = 125 + 25 = 150$$

Complexity of Planning

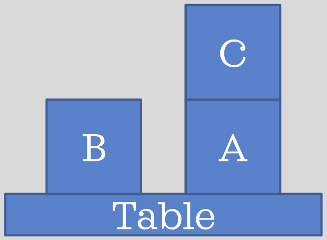
- In Blocks World, if instead we had:
  - **6 Objects** (five blocks and the table)
  - 1 action template with 3 parameters *move(b, from, to)*
  - 1 action template with 2 parameters *moveToTable(b, from)*

- *Total number of different ground actions:*

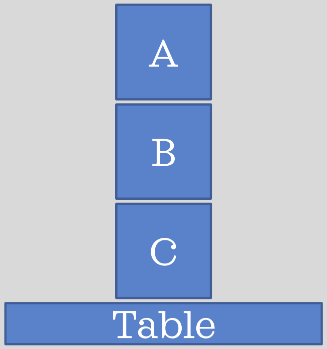
$$6^3 + 6^2 = 216 + 36 = 252$$

Complexity of Planning

- The question of, given a problem, is there a plan that exists that takes you from the initial state to the goal state, is called **PlanSAT**.
  - i.e., plan satisfiability! Is there a plan that solves the problem!
- PlanSAT is a complex problem…



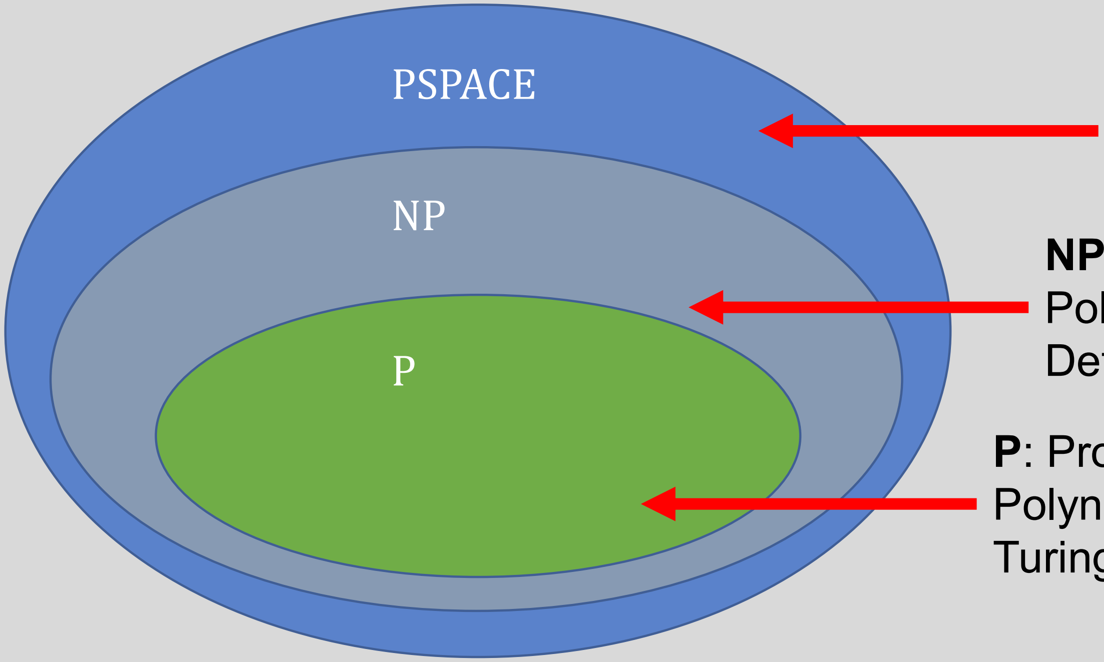
Initial State



Goal State

Complexity of Planning

- You will / have already learned more about this in your algorithms class…



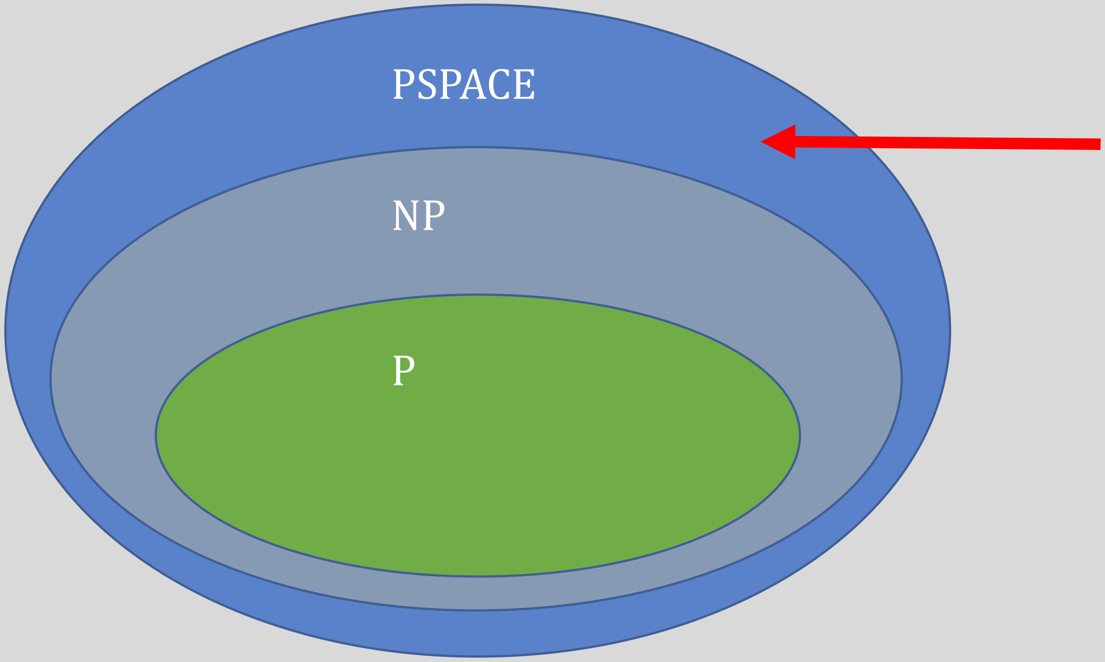
PSPACE: Problems that can solved with a deterministic Turing machine using a polynomial amount of space.

NP: Problems that can solved in Polynomial Time with a Non-Deterministic Turing machine.

P: Problems that can solved in Polynomial Time with a Deterministic Turing machine.

Complexity of Planning

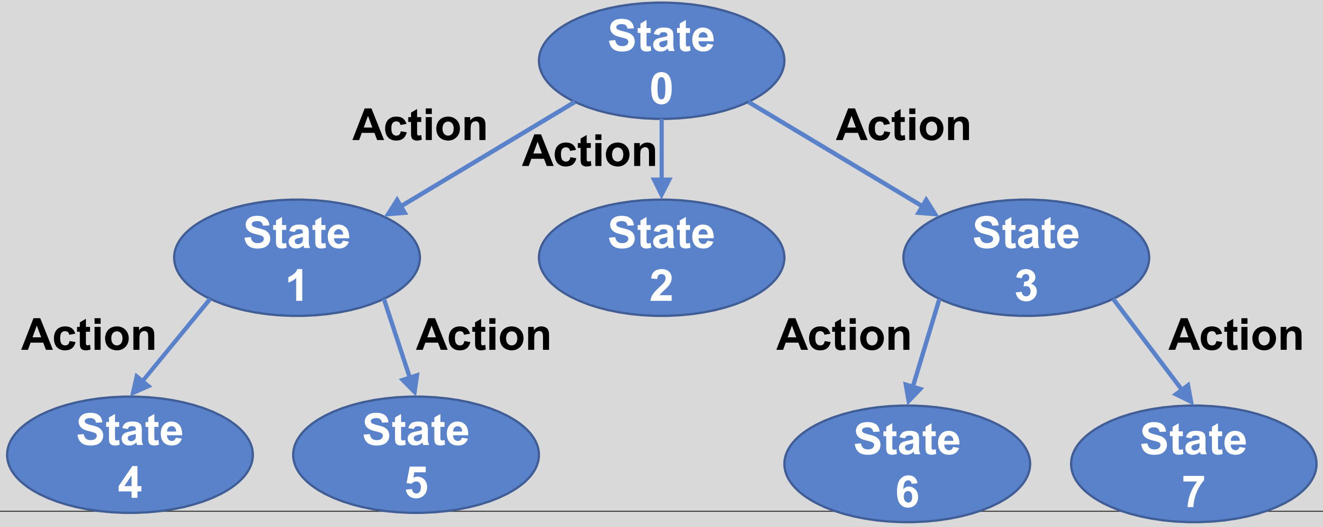
- You will / have already learned more about this in your algorithms class…



PlanSAT is here! The hard one!

Planning as Search

- This is in part because the state space blows up!
- If we think of each configuration as a state, and each action takes us from one state to another, even simple problems like Blocks World get big fast!



Planning as we’ll talk about it!

- Begin with a very naïve approach (coming up soon!)
- Then introduce least-commitment planning.
  - “CERTAIN things have to happen before OTHER CERTAIN things, but generally speaking we don’t are what order things happen in”
- Plan-Graph based Planning
  - “Don’t search over individual states… search over *entire plans*”
- State-Space Planning
  - “Didn’t we just say that made the space too big?”
  - Well, yes, but we will use clever heuristics to help us out!

GPS: General Problem Solver

- Developed in 1959 by Herbert A. Simon, J. C. Shaw, and Allen Newell
- One of the first programs to separate the description of the problem from the strategy for solving it
- Considered the first planner
- Quickly mired in combinatorial explosion
- One of its unfortunate shortcomings:

it fails to reason about interleaved goals

GPS: General Problem Solver

```
Let the initial state be the current state.
To satisfy some goal G:
    For each conjunct C of G:
        If C is true in the current state, do nothing.
        Else:
            Choose an action A which achieves C.
            For each precondition P of A, satisfy P.
            If all preconditions were satisfied:
                Add the action to the plan.
                Update the current state.
```

GPS: General Problem Solver

```
Let the initial state be the current state.
To satisfy some goal G:
    For each conjunct C of G:
        If C is true in the current state, do nothing.
        Else:
            Choose an action A which achieves C.
            For each precondition P of A, satisfy P.
            If all preconditions were satisfied:
                Add the action to the plan.
                Update the current state.
```

This is basically a recursive step. Find actions that convert the current state to a point where we are allowed to use this action.

Let’s see this play out!

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

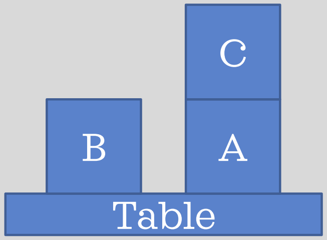
We begin with an initial state

And a goal state

$on(A, B) \wedge on(B, C)$

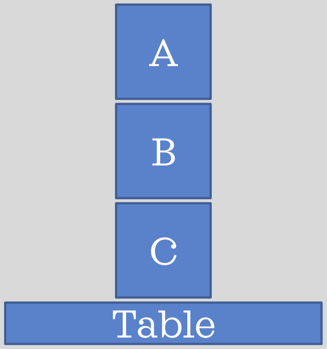
$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

We begin with an initial state



Initial State

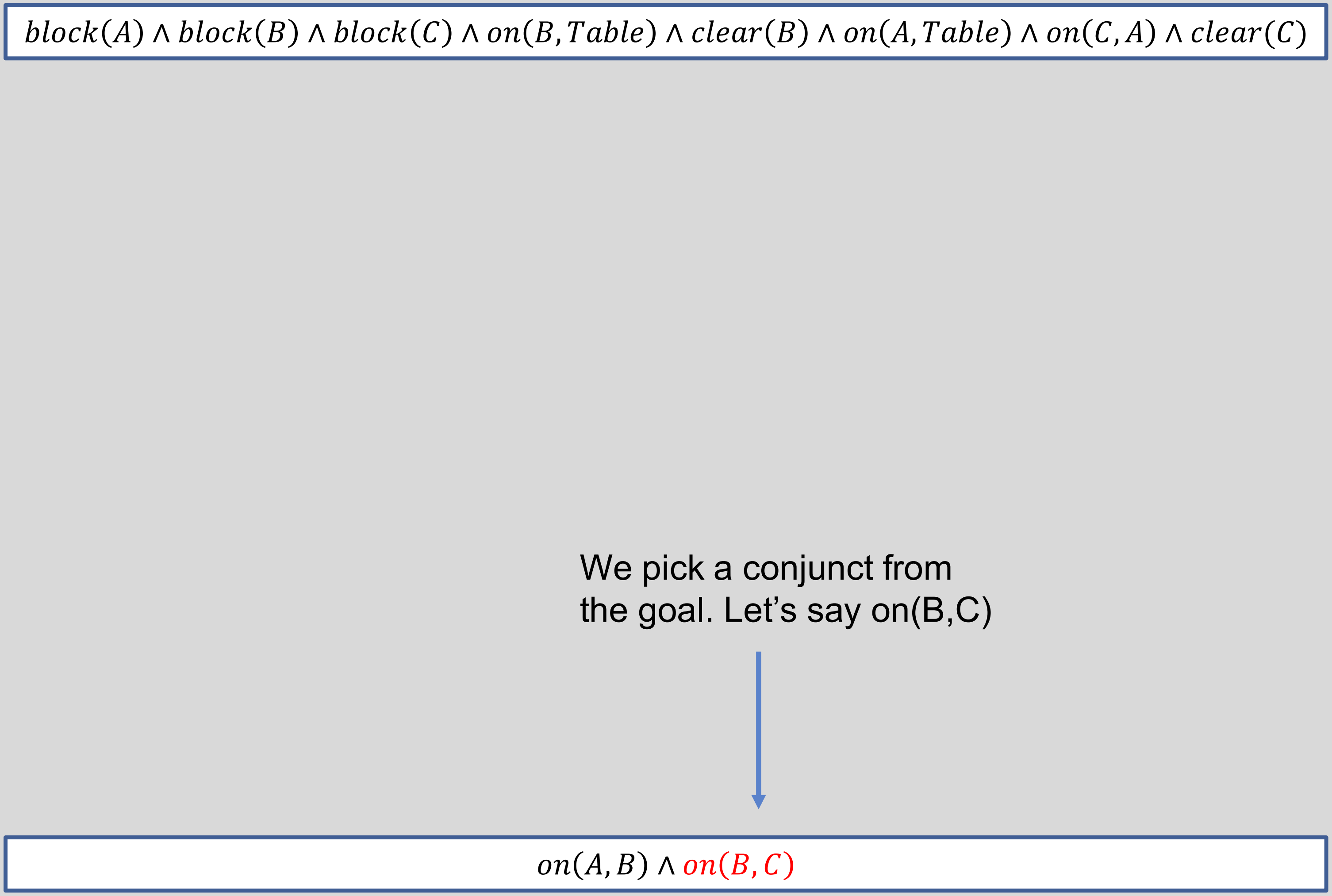
Look familiar?



Goal State

And a goal state

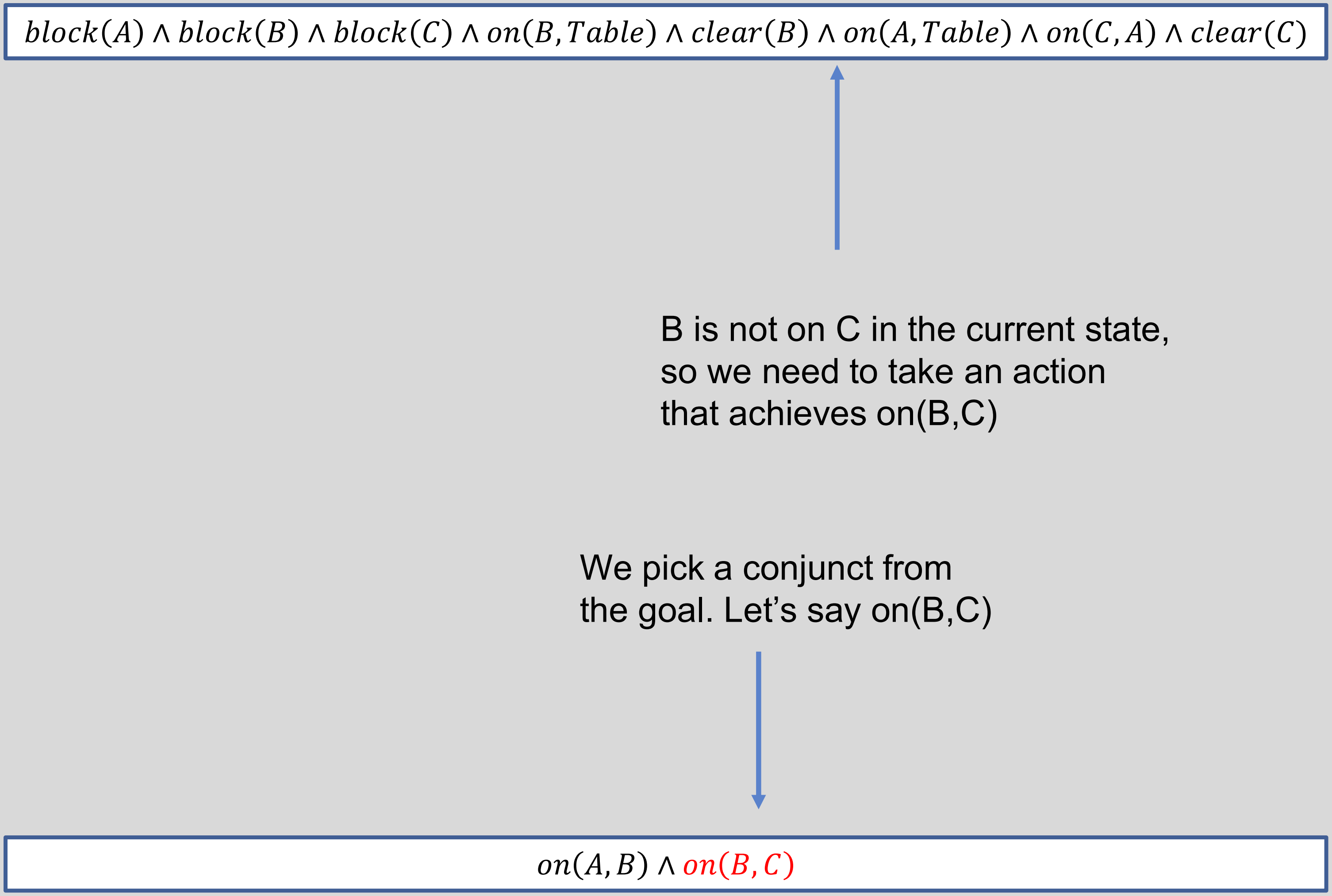
$on(A, B) \wedge on(B, C)$



$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

We pick a conjunct from the goal. Let’s say on(B,C)

$on(A, B) \land on(B, C)$

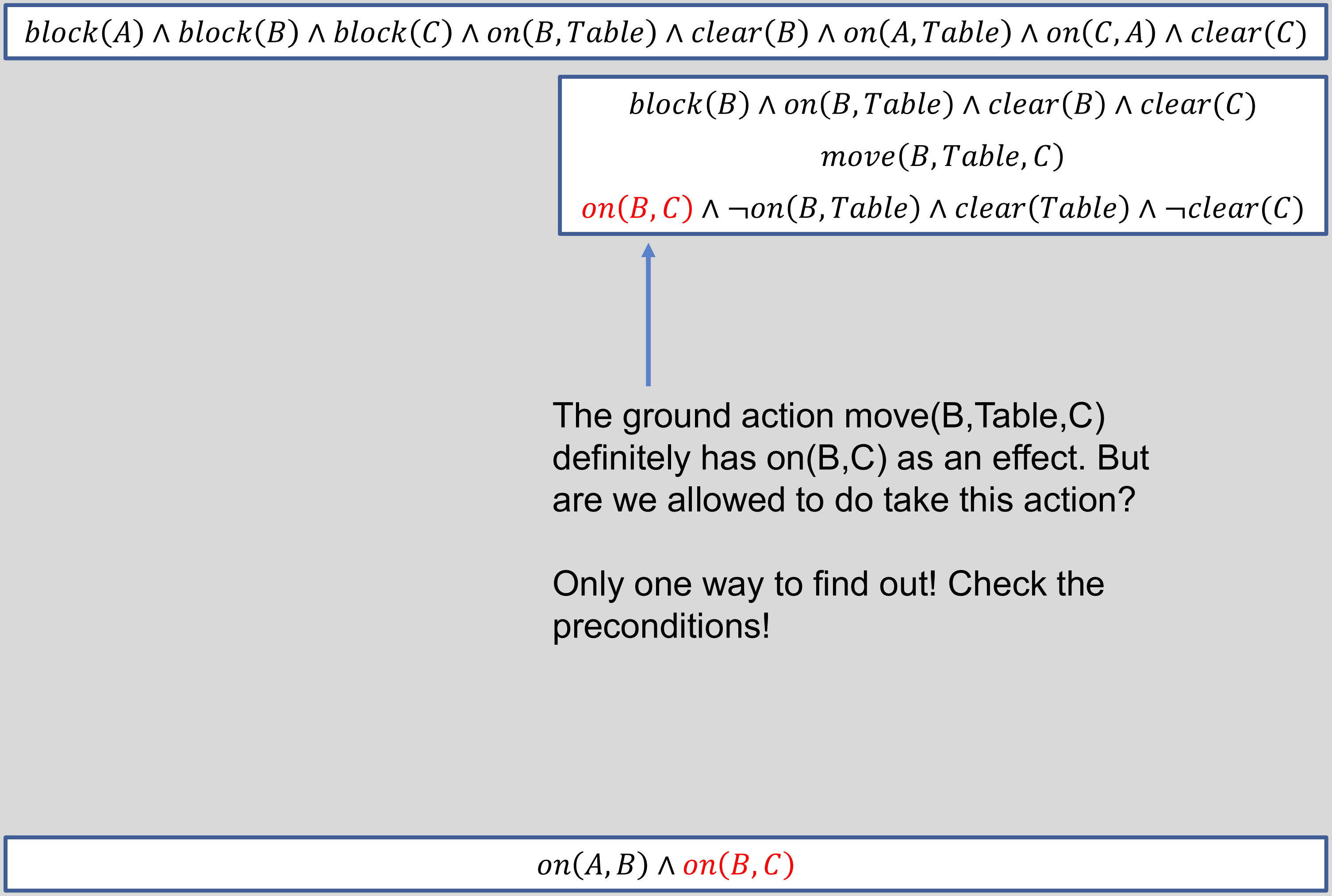


$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

B is not on C in the current state,
so we need to take an action
that achieves on(B,C)

We pick a conjunct from
the goal. Let's say on(B,C)

$on(A, B) \wedge on(B, C)$



$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

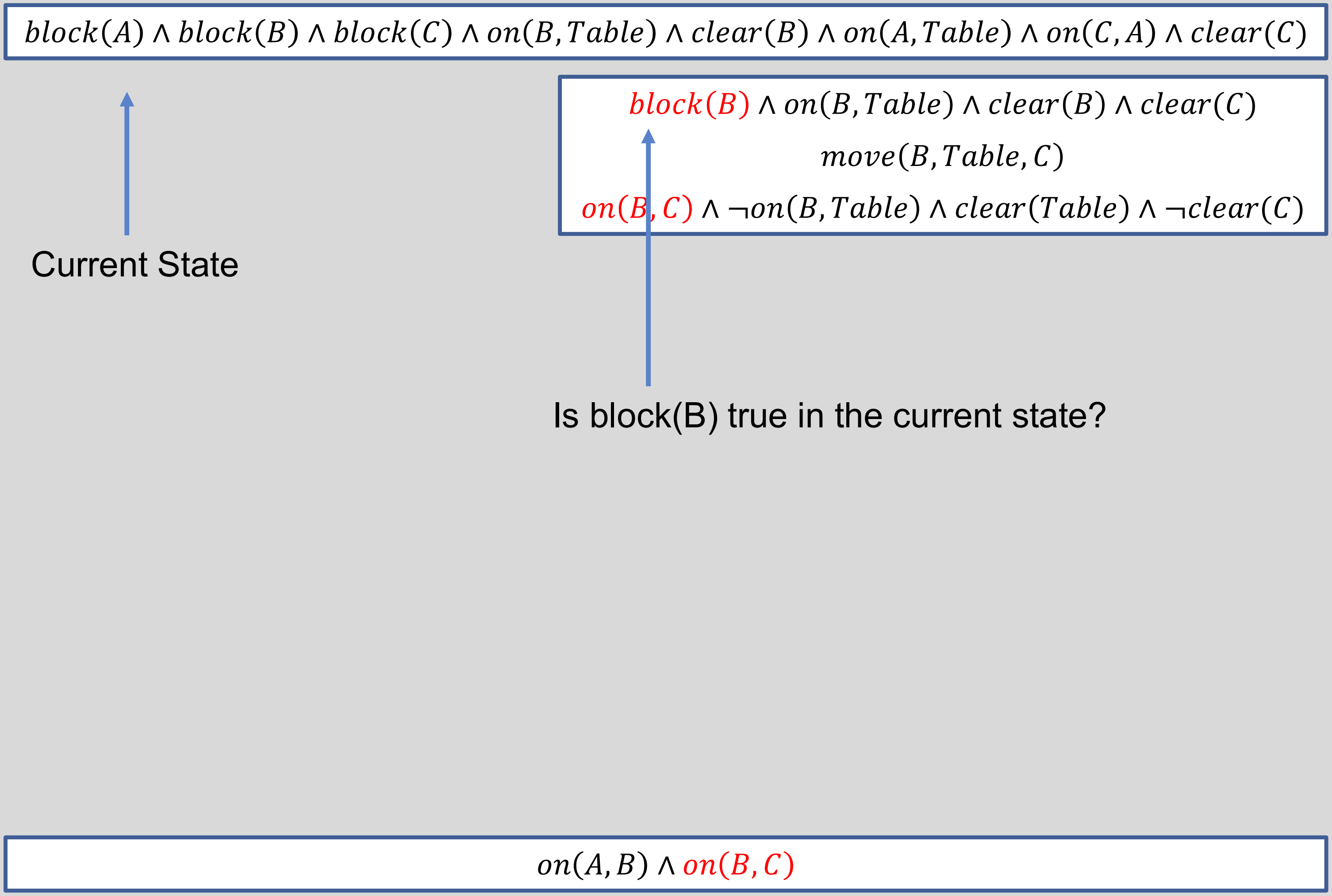
$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

The ground action move(B,Table,C) definitely has on(B,C) as an effect. But are we allowed to do take this action?

Only one way to find out! Check the preconditions!

$on(A, B) \land on(B, C)$



$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

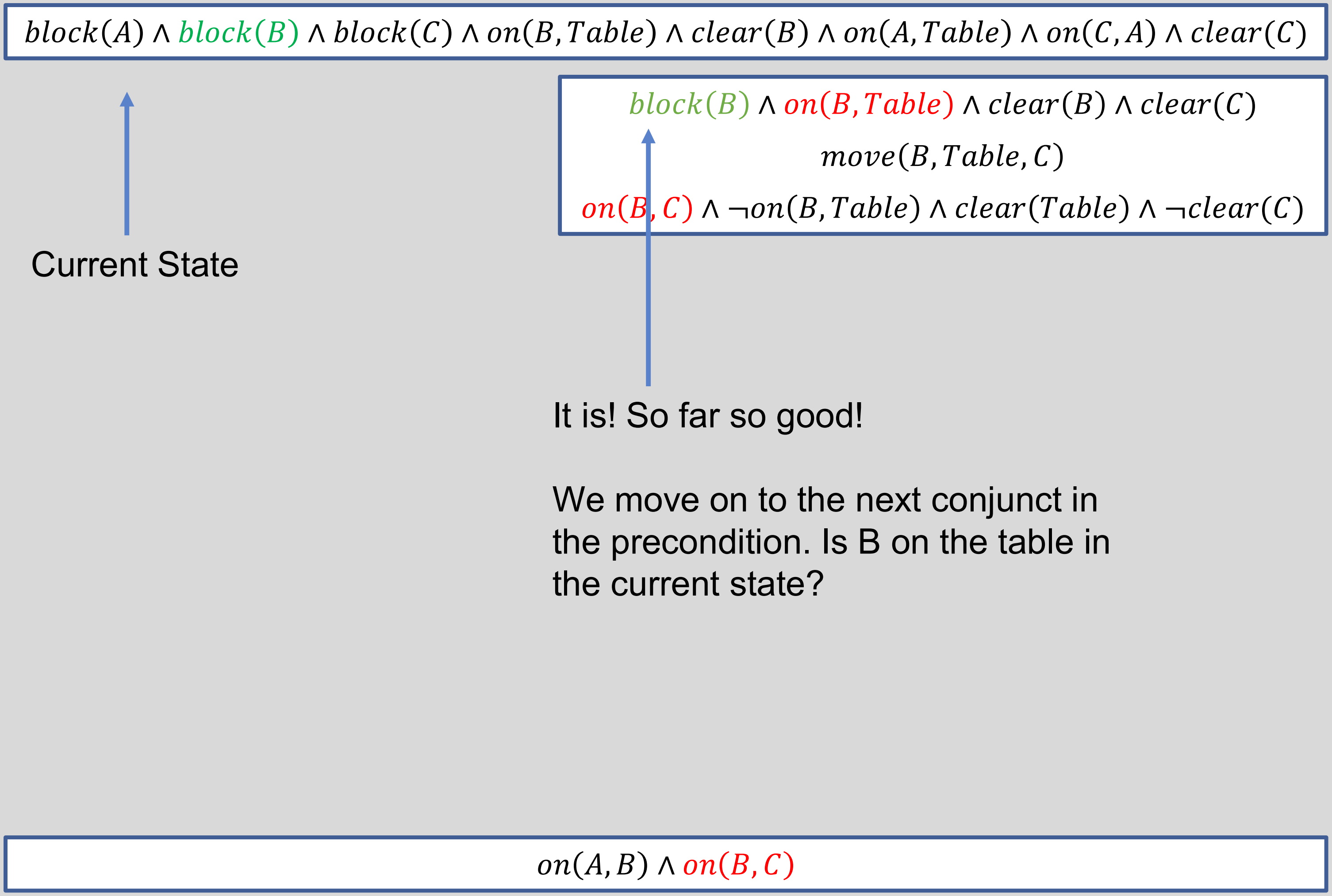
$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State

Is block(B) true in the current state?

$on(A, B) \wedge on(B, C)$



$block(A) \wedge block(B) \wedge block(C) \wedge on(B,Table) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A) \wedge clear(C)$

$block(B) \wedge on(B,Table) \wedge clear(B) \wedge clear(C)$

$move(B,Table,C)$

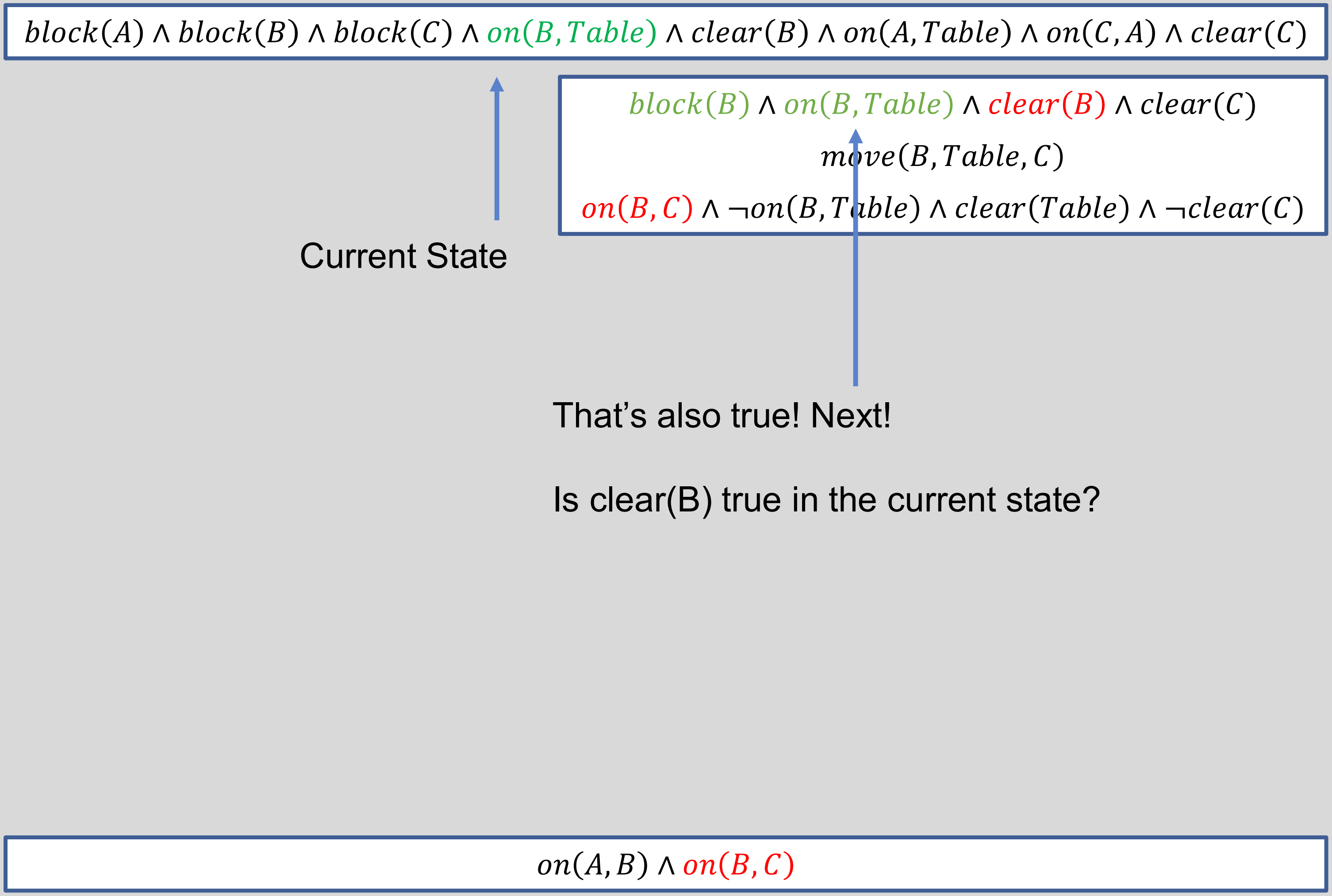
$on(B,C) \wedge \neg on(B,Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State

It is! So far so good!

We move on to the next conjunct in the precondition. Is B on the table in the current state?

$on(A,B) \wedge on(B,C)$



$block(A) \wedge block(B) \wedge block(C) \wedge on(B,Table) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A) \wedge clear(C)$

$block(B) \wedge on(B,Table) \wedge clear(B) \wedge clear(C)$

$move(B,Table,C)$

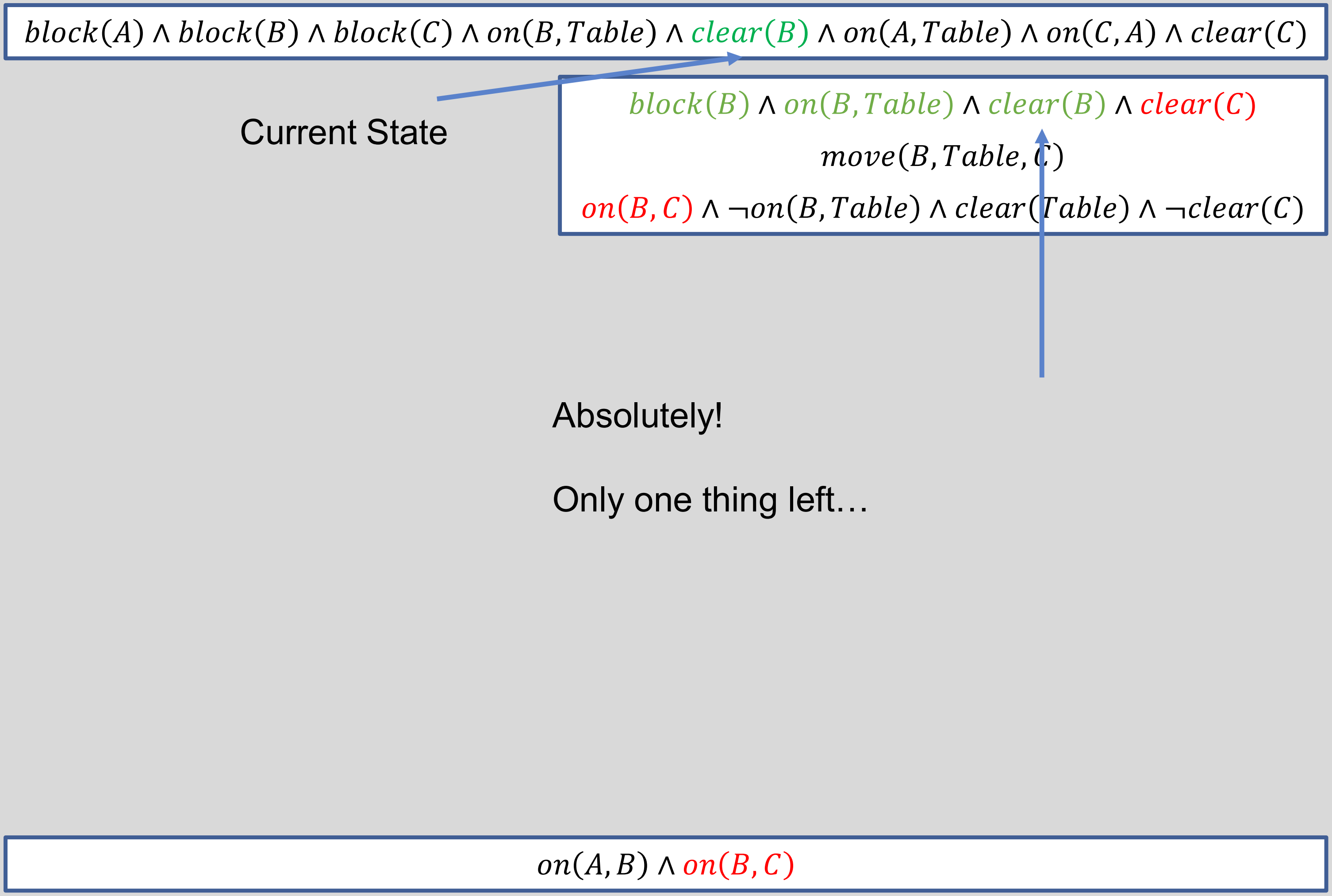
$on(B,C) \wedge \neg on(B,Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State

That’s also true! Next!

Is clear(B) true in the current state?

$on(A,B) \wedge on(B,C)$



$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

Current State

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

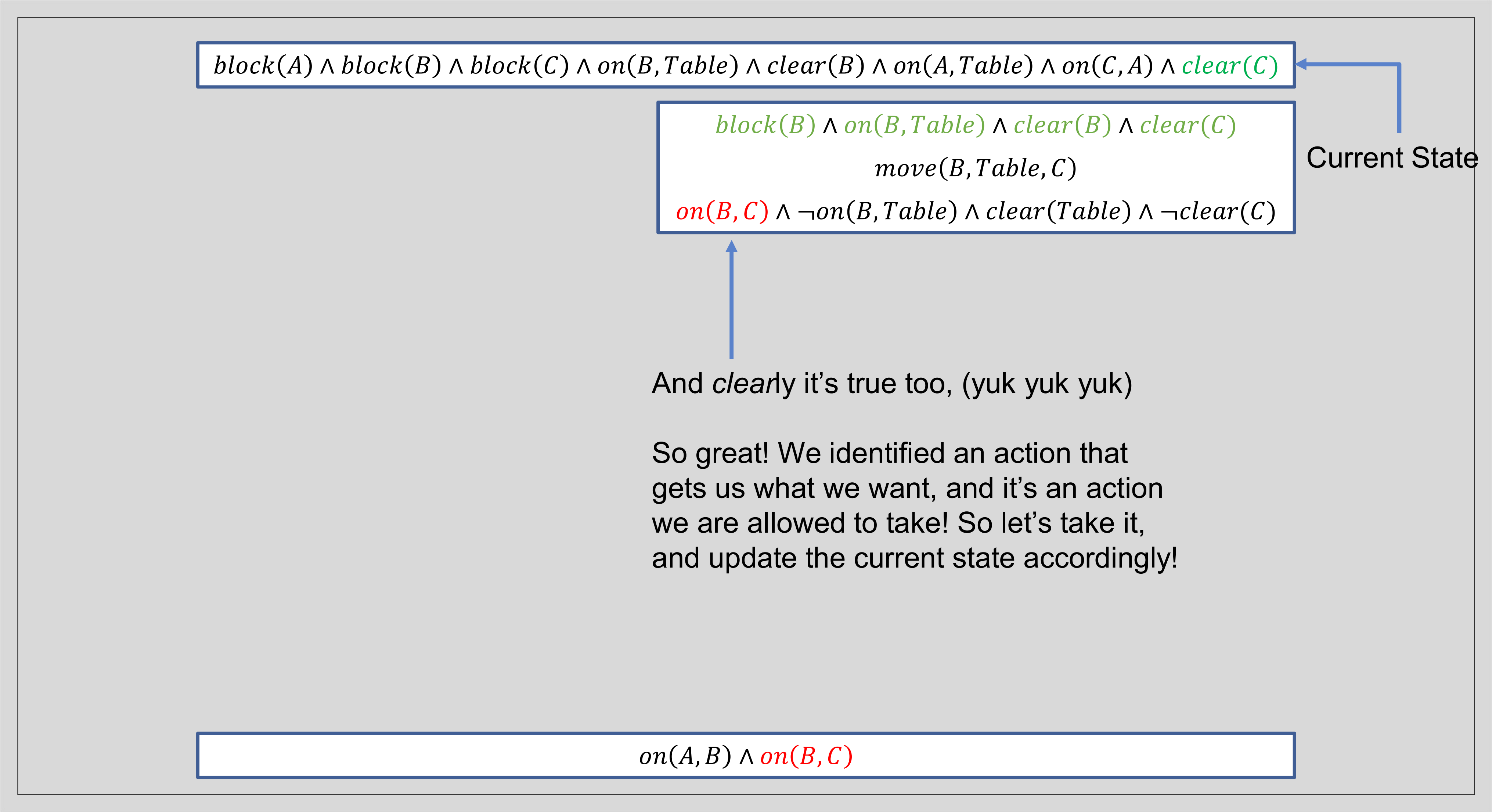
$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Absolutely!

Only one thing left…

$on(A, B) \land on(B, C)$



$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

Current State

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

And *clear*ly it’s true too, (yuk yuk yuk)

So great! We identified an action that gets us what we want, and it’s an action we are allowed to take! So let’s take it, and update the current state accordingly!

$on(A, B) \wedge on(B, C)$

$$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$$

$$block(B) \land on(B, Table) \land clear(B) \land clear(C)$$

$$move(B, Table, C)$$

$$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$$

Current State

$$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$$

$$on(A, B) \land on(B, C)$$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State $block(A) \wedge block(B) \wedge block(C) \wedge on(B, C) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A)$

So we apply all of the effects of the action on the original state.
Some things change, and some things stay the same:

*A, B, and C are still blocks.
*B is no longer on the table, it is instead on C.
*B is still clear, A is still on the table.
*C is still on A.
*C is no longer clear.

$on(A, B) \wedge on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

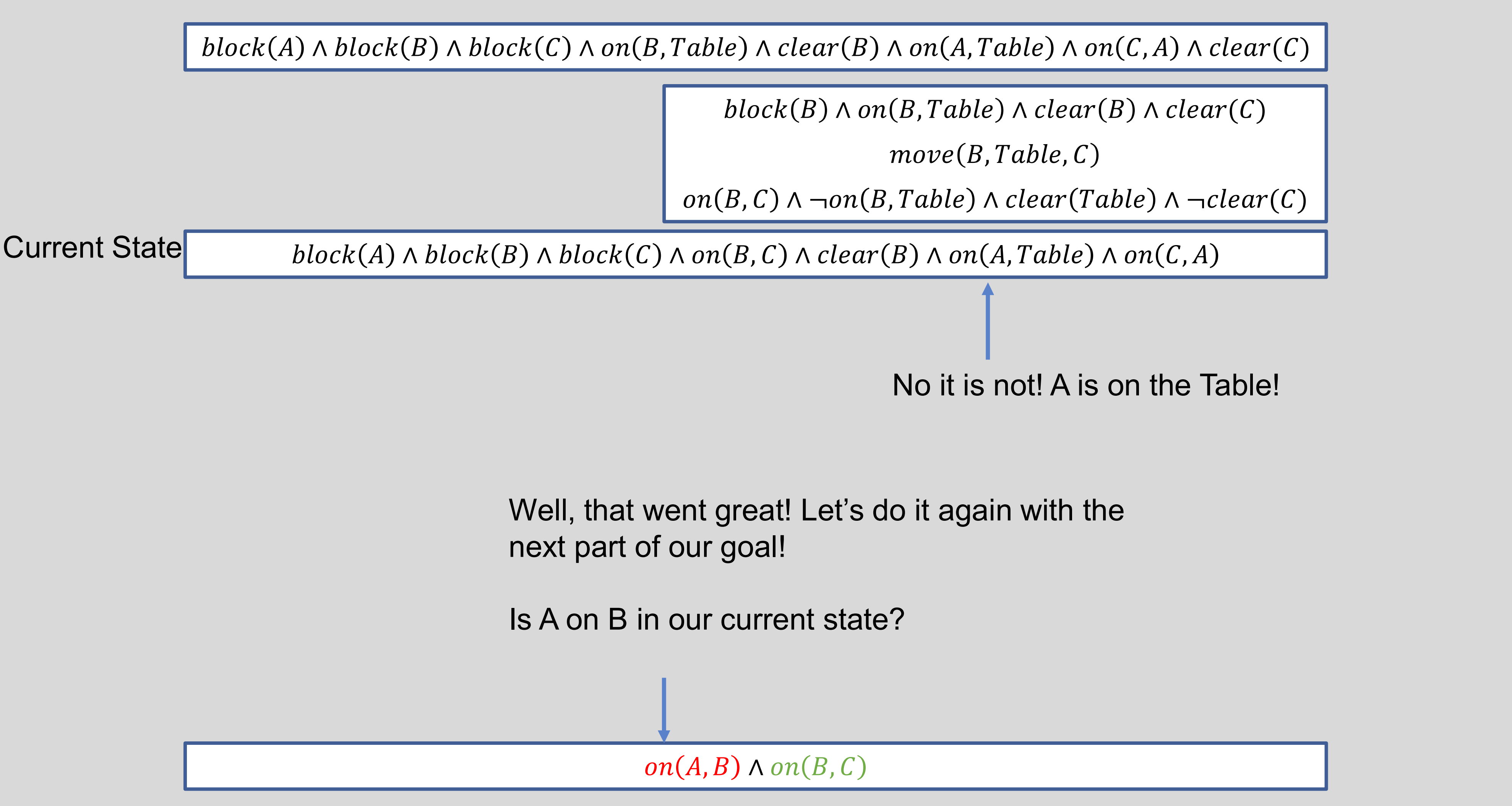
Current State $block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

Well, that went great! Let's do it again with the next part of our goal!

Is A on B in our current state?

$\downarrow$

$on(A, B) \land on(B, C)$



$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

No it is not! A is on the Table!

Well, that went great! Let’s do it again with the next part of our goal!

Is A on B in our current state?

$on(A, B) \land on(B, C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State $block(A) \wedge block(B) \wedge block(C) \wedge on(B, C) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A)$

We find an action that has on(A,B) as an effect: move(A, Table, B)

(printed down there because I know the future and know it will look prettier this way)

$block(A) \wedge on(A, Table) \wedge clear(A) \wedge clear(B)$

$move(A, Table, B)$

$on(A, B) \wedge \neg on(A, Table) \wedge clear(Table) \wedge \neg clear(B)$

$on(A, B) \wedge on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State $block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

We go through the preconditions, comparing them to the current state.

block(A), yes…

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

We go through the preconditions…

A on the Table, yes…

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$$

$$block(B) \land on(B, Table) \land clear(B) \land clear(C)$$

$$move(B, Table, C)$$

$$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$$

Current State

$$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$$

We go through the preconditions…

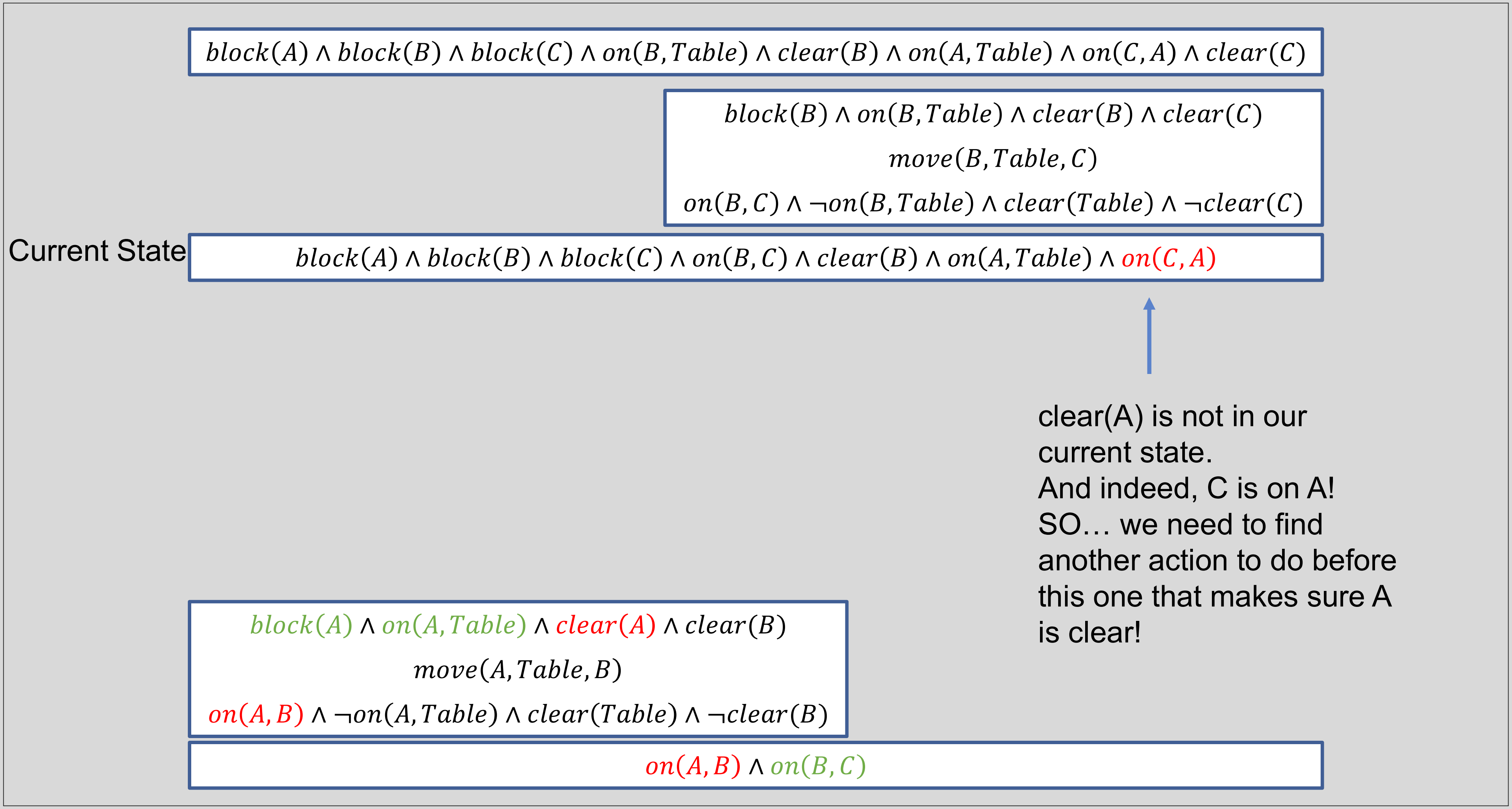
Is A clear? … uh oh…

$$block(A) \land on(A, Table) \land clear(A) \land clear(B)$$

$$move(A, Table, B)$$

$$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$$

$$on(A, B) \land on(B, C)$$



$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, C) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A)$

clear(A) is not in our current state.
And indeed, C is on A!
SO… we need to find another action to do before this one that makes sure A is clear!

$block(A) \wedge on(A, Table) \wedge clear(A) \wedge clear(B)$

$move(A, Table, B)$

$on(A, B) \wedge \neg on(A, Table) \wedge clear(Table) \wedge \neg clear(B)$

$on(A, B) \wedge on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State $block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

How about moveToTable(C,A)

That certainly achieves the effect we want. But are we allowed to do it?

Go through the preconditions…

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State $block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

C is a block… good…

Is C on A?

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(B) \land on(B,Table) \land clear(B) \land clear(C)$

$move(B,Table,C)$

$on(B,C) \land \neg on(B,Table) \land clear(Table) \land \neg clear(C)$

Current State $block(A) \land block(B) \land block(C) \land on(B,C) \land clear(B) \land on(A,Table) \land on(C,A)$

$block(C) \land on(C,A) \land clear(C)$

$moveToTable(C,A)$

$on(C,Table) \land \neg on(C,A) \land clear(A)$

$block(A) \land on(A,Table) \land clear(A) \land clear(B)$

$move(A,Table,B)$

$on(A,B) \land \neg on(A,Table) \land clear(Table) \land \neg clear(B)$

$on(A,B) \land on(B,C)$

Yes, C is on A, ok,

Is C clear?

No it is not. B is on C.

SIGH. OK. What’s another action that makes C clear?

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, C) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A)$

$block(B) \wedge on(B, C) \wedge clear(B)$

$moveToTable(B, C)$

$on(B, Table) \wedge \neg on(B, C) \wedge clear(C)$

$block(C) \wedge on(C, A) \wedge clear(C)$

$moveToTable(C, A)$

$on(C, Table) \wedge \neg on(C, A) \wedge clear(A)$

$block(A) \wedge on(A, Table) \wedge clear(A) \wedge clear(B)$

$move(A, Table, B)$

$on(A, B) \wedge \neg on(A, Table) \wedge clear(Table) \wedge \neg clear(B)$

$on(A, B) \wedge on(B, C)$

Well, moveToTable(B,C) will do the trick, assuming we are allowed to do it…

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(B) \land on(B, C) \land clear(B)$

$moveToTable(B, C)$

$on(B, Table) \land \neg on(B, C) \land clear(C)$

Yes B is a Block…

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(B) \land on(B, C) \land clear(B)$

$moveToTable(B, C)$

$on(B, Table) \land \neg on(B, C) \land clear(C)$

Yes, B is on C…

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

Current State

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(B) \land on(B, C) \land clear(B)$

$moveToTable(B, C)$

$on(B, Table) \land \neg on(B, C) \land clear(C)$

And yes, B is clear!

Hooray!

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

$on(A, B) \land on(B, C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B,Table) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A) \wedge clear(C)$

$block(B) \wedge on(B,Table) \wedge clear(B) \wedge clear(C)$

$move(B,Table,C)$

$on(B,C) \wedge \neg on(B,Table) \wedge clear(Table) \wedge \neg clear(C)$

Current State $block(A) \wedge block(B) \wedge block(C) \wedge on(B,C) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A)$

$block(B) \wedge on(B,C) \wedge clear(B)$

$moveToTable(B,C)$

$on(B,Table) \wedge \neg on(B,C) \wedge clear(C)$

$block(C) \wedge on(C,A) \wedge clear(C)$

$moveToTable(C,A)$

$on(C,Table) \wedge \neg on(C,A) \wedge clear(A)$

The preconditions are satisfied, so we take this action and update the current state!

$block(A) \wedge on(A,Table) \wedge clear(A) \wedge clear(B)$

$move(A,Table,B)$

$on(A,B) \wedge \neg on(A,Table) \wedge clear(Table) \wedge \neg clear(B)$

$on(A,B) \wedge on(B,C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B,Table) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A) \wedge clear(C)$

$block(B) \wedge on(B,Table) \wedge clear(B) \wedge clear(C)$

$move(B,Table,C)$

$on(B,C) \wedge \neg on(B,Table) \wedge clear(Table) \wedge \neg clear(C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B,C) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A)$

$block(B) \wedge on(B,C) \wedge clear(B)$

$moveToTable(B,C)$

$on(B,Table) \wedge \neg on(B,C) \wedge clear(C)$

Current State $block(A) \wedge block(B) \wedge block(C) \wedge on(B,Table) \wedge clear(B) \wedge on(A,Table) \wedge on(C,A) \wedge clear(C)$

$block(C) \wedge on(C,A) \wedge clear(C)$

$moveToTable(C,A)$

$on(C,Table) \wedge \neg on(C,A) \wedge clear(A)$

B has been moved on the table,
and C is clear!

$block(A) \wedge on(A,Table) \wedge clear(A) \wedge clear(B)$

$move(A,Table,B)$

$on(A,B) \wedge \neg on(A,Table) \wedge clear(Table) \wedge \neg clear(B)$

$on(A,B) \wedge on(B,C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(B) \wedge on(B, Table) \wedge clear(B) \wedge clear(C)$

$move(B, Table, C)$

$on(B, C) \wedge \neg on(B, Table) \wedge clear(Table) \wedge \neg clear(C)$

$block(A) \wedge block(B) \wedge block(C) \wedge on(B, C) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A)$

$block(B) \wedge on(B, C) \wedge clear(B)$

$moveToTable(B, C)$

$on(B, Table) \wedge \neg on(B, C) \wedge clear(C)$

Current State $block(A) \wedge block(B) \wedge block(C) \wedge on(B, Table) \wedge clear(B) \wedge on(A, Table) \wedge on(C, A) \wedge clear(C)$

$block(C) \wedge on(C, A) \wedge clear(C)$

$moveToTable(C, A)$

$on(C, Table) \wedge \neg on(C, A) \wedge clear(A)$

So now that precondition has been satisfied (and we didn”t undo any of the other preconditions!)

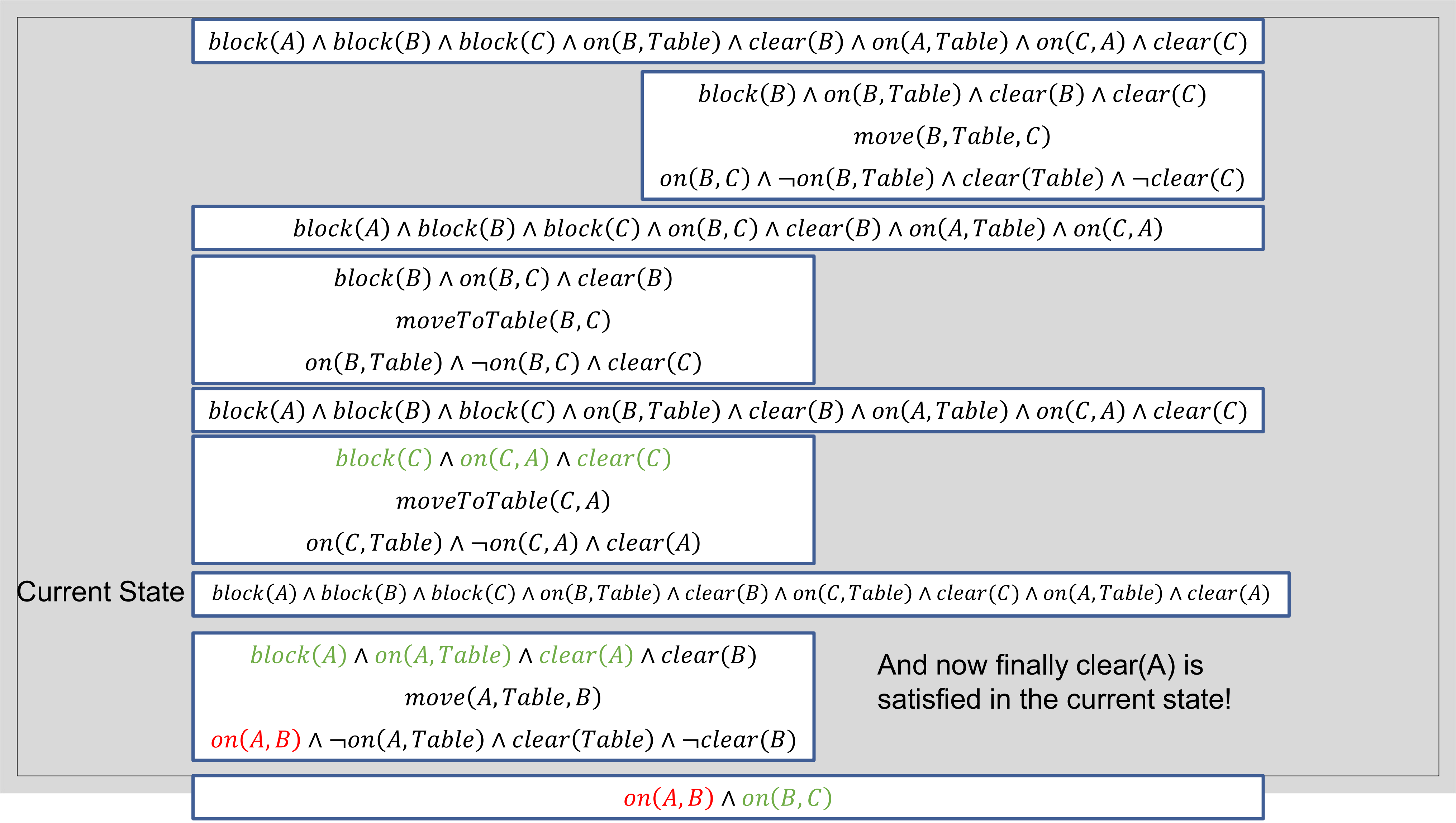
$block(A) \wedge on(A, Table) \wedge clear(A) \wedge clear(B)$

$move(A, Table, B)$

$on(A, B) \wedge \neg on(A, Table) \wedge clear(Table) \wedge \neg clear(B)$

So we can now successfully take the moveToTable(C,A) action to achieve clear(A)

$on(A, B) \wedge on(B, C)$



$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(B) \land on(B,Table) \land clear(B) \land clear(C)$

$move(B,Table,C)$

$on(B,C) \land \neg on(B,Table) \land clear(Table) \land \neg clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,C) \land clear(B) \land on(A,Table) \land on(C,A)$

$block(B) \land on(B,C) \land clear(B)$

$moveToTable(B,C)$

$on(B,Table) \land \neg on(B,C) \land clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(C) \land on(C,A) \land clear(C)$

$moveToTable(C,A)$

$on(C,Table) \land \neg on(C,A) \land clear(A)$

Current State $block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(C,Table) \land clear(C) \land on(A,Table) \land clear(A)$

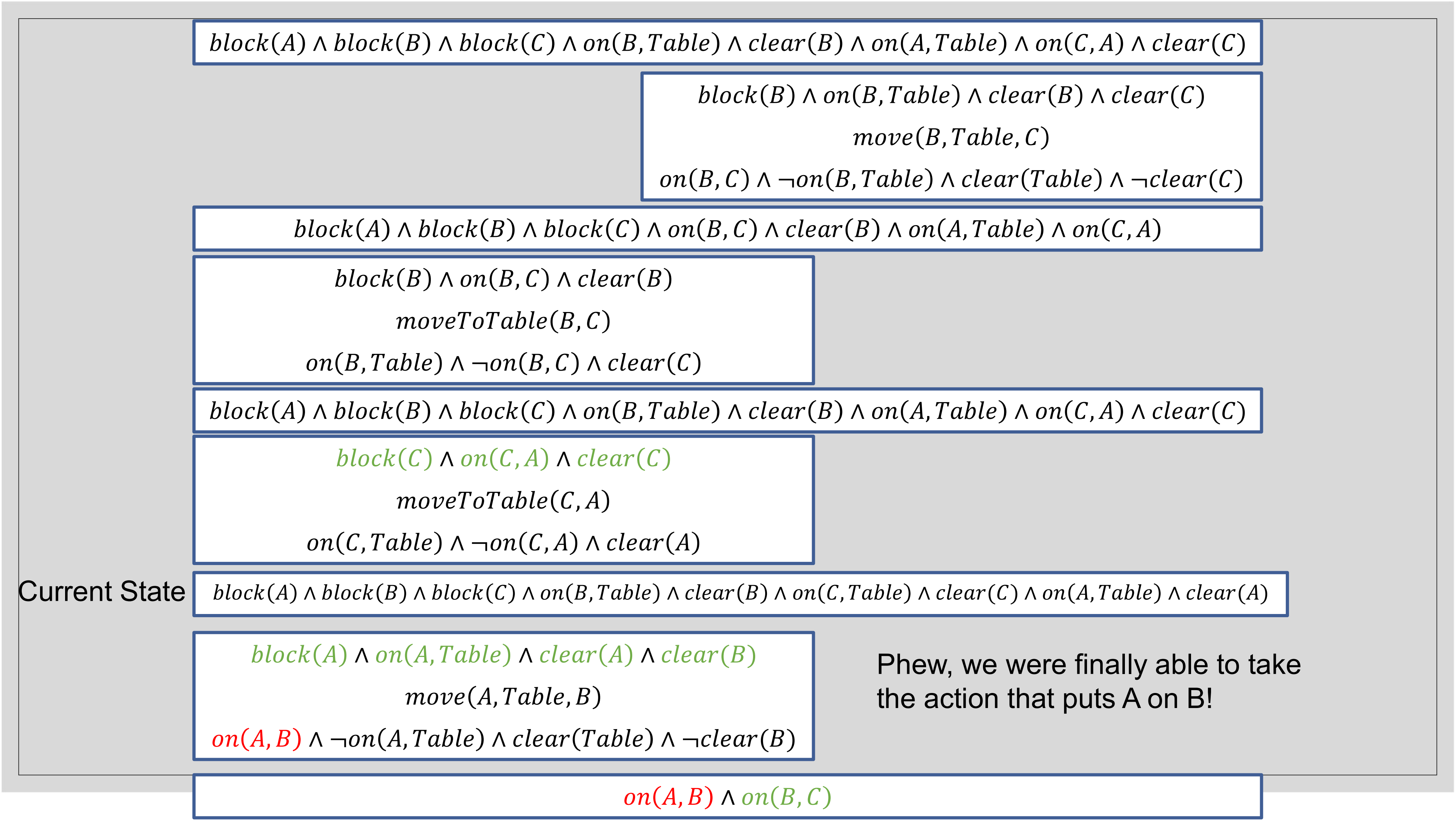
$block(A) \land on(A,Table) \land clear(A) \land clear(B)$

$move(A,Table,B)$

$on(A,B) \land \neg on(A,Table) \land clear(Table) \land \neg clear(B)$

And now finally clear(A) is satisfied in the current state!

$on(A,B) \land on(B,C)$



$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(B) \land on(B, Table) \land clear(B) \land clear(C)$

$move(B, Table, C)$

$on(B, C) \land \neg on(B, Table) \land clear(Table) \land \neg clear(C)$

$block(A) \land block(B) \land block(C) \land on(B, C) \land clear(B) \land on(A, Table) \land on(C, A)$

$block(B) \land on(B, C) \land clear(B)$

$moveToTable(B, C)$

$on(B, Table) \land \neg on(B, C) \land clear(C)$

$block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(A, Table) \land on(C, A) \land clear(C)$

$block(C) \land on(C, A) \land clear(C)$

$moveToTable(C, A)$

$on(C, Table) \land \neg on(C, A) \land clear(A)$

Current State $block(A) \land block(B) \land block(C) \land on(B, Table) \land clear(B) \land on(C, Table) \land clear(C) \land on(A, Table) \land clear(A)$

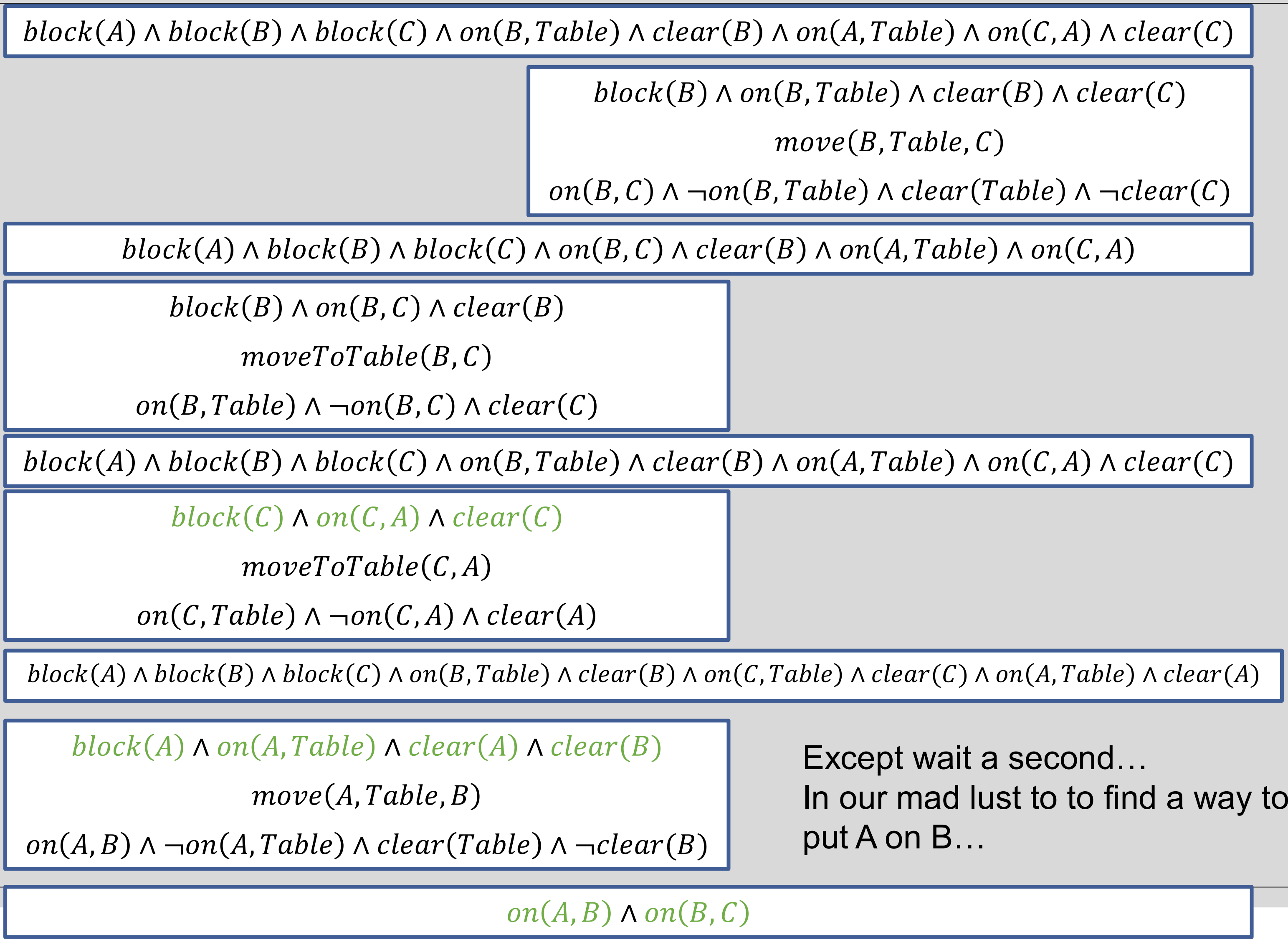
$block(A) \land on(A, Table) \land clear(A) \land clear(B)$

$move(A, Table, B)$

$on(A, B) \land \neg on(A, Table) \land clear(Table) \land \neg clear(B)$

Phew, we were finally able to take the action that puts A on B!

$on(A, B) \land on(B, C)$



$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(B) \land on(B,Table) \land clear(B) \land clear(C)$

$move(B,Table,C)$

$on(B,C) \land \neg on(B,Table) \land clear(Table) \land \neg clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,C) \land clear(B) \land on(A,Table) \land on(C,A)$

$block(B) \land on(B,C) \land clear(B)$

$moveToTable(B,C)$

$on(B,Table) \land \neg on(B,C) \land clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(C) \land on(C,A) \land clear(C)$

$moveToTable(C,A)$

$on(C,Table) \land \neg on(C,A) \land clear(A)$

Current State

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(C,Table) \land clear(C) \land on(A,Table) \land clear(A)$

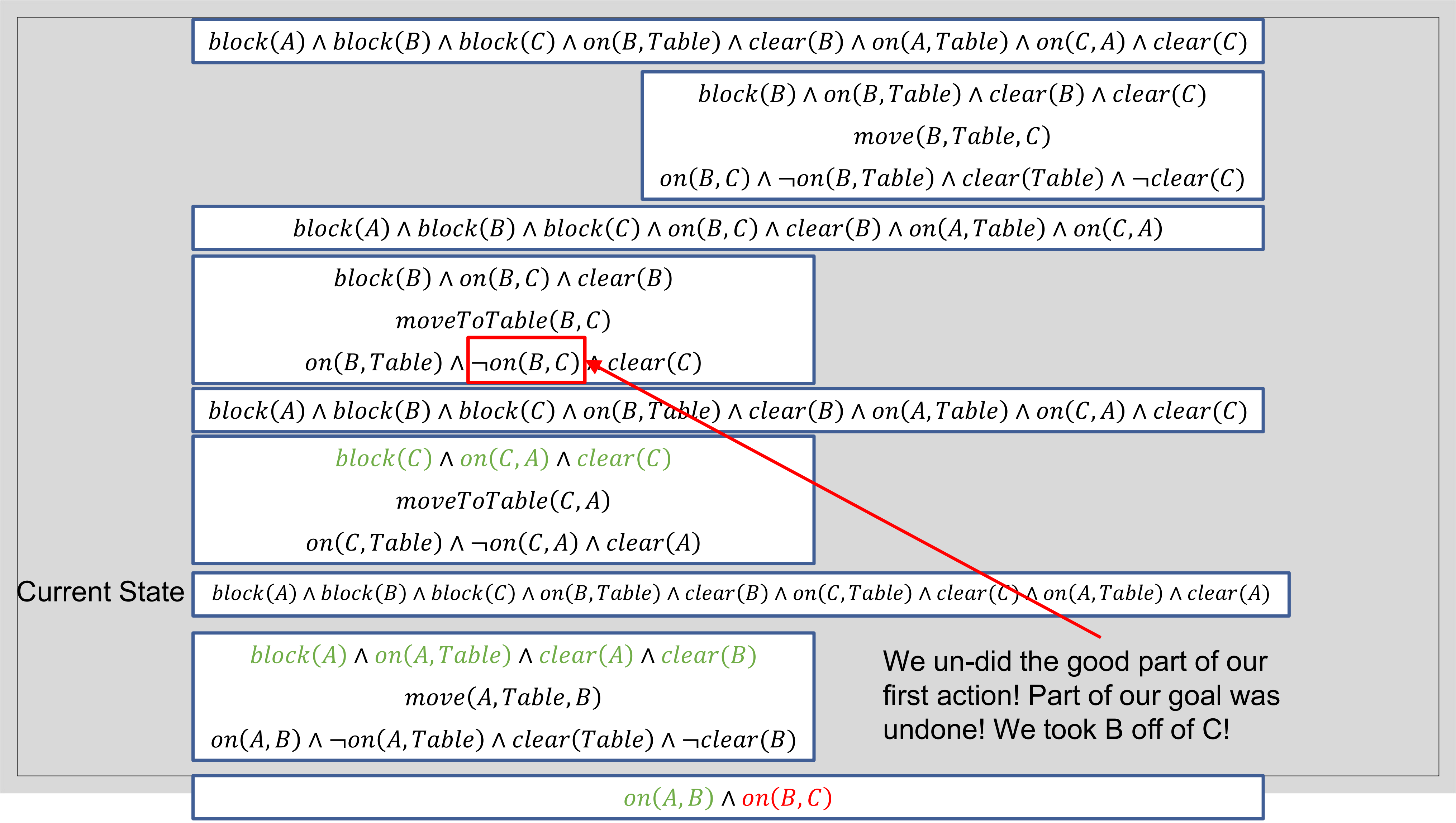
$block(A) \land on(A,Table) \land clear(A) \land clear(B)$

$move(A,Table,B)$

$on(A,B) \land \neg on(A,Table) \land clear(Table) \land \neg clear(B)$

$on(A,B) \land on(B,C)$

Except wait a second…
In our mad lust to to find a way to
put A on B…



$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(B) \land on(B,Table) \land clear(B) \land clear(C)$

$move(B,Table,C)$

$on(B,C) \land \neg on(B,Table) \land clear(Table) \land \neg clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,C) \land clear(B) \land on(A,Table) \land on(C,A)$

$block(B) \land on(B,C) \land clear(B)$

$moveToTable(B,C)$

$on(B,Table) \land \boxed{\neg on(B,C)} \land clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(C) \land on(C,A) \land clear(C)$

$moveToTable(C,A)$

$on(C,Table) \land \neg on(C,A) \land clear(A)$

Current State $block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(C,Table) \land clear(C) \land on(A,Table) \land clear(A)$

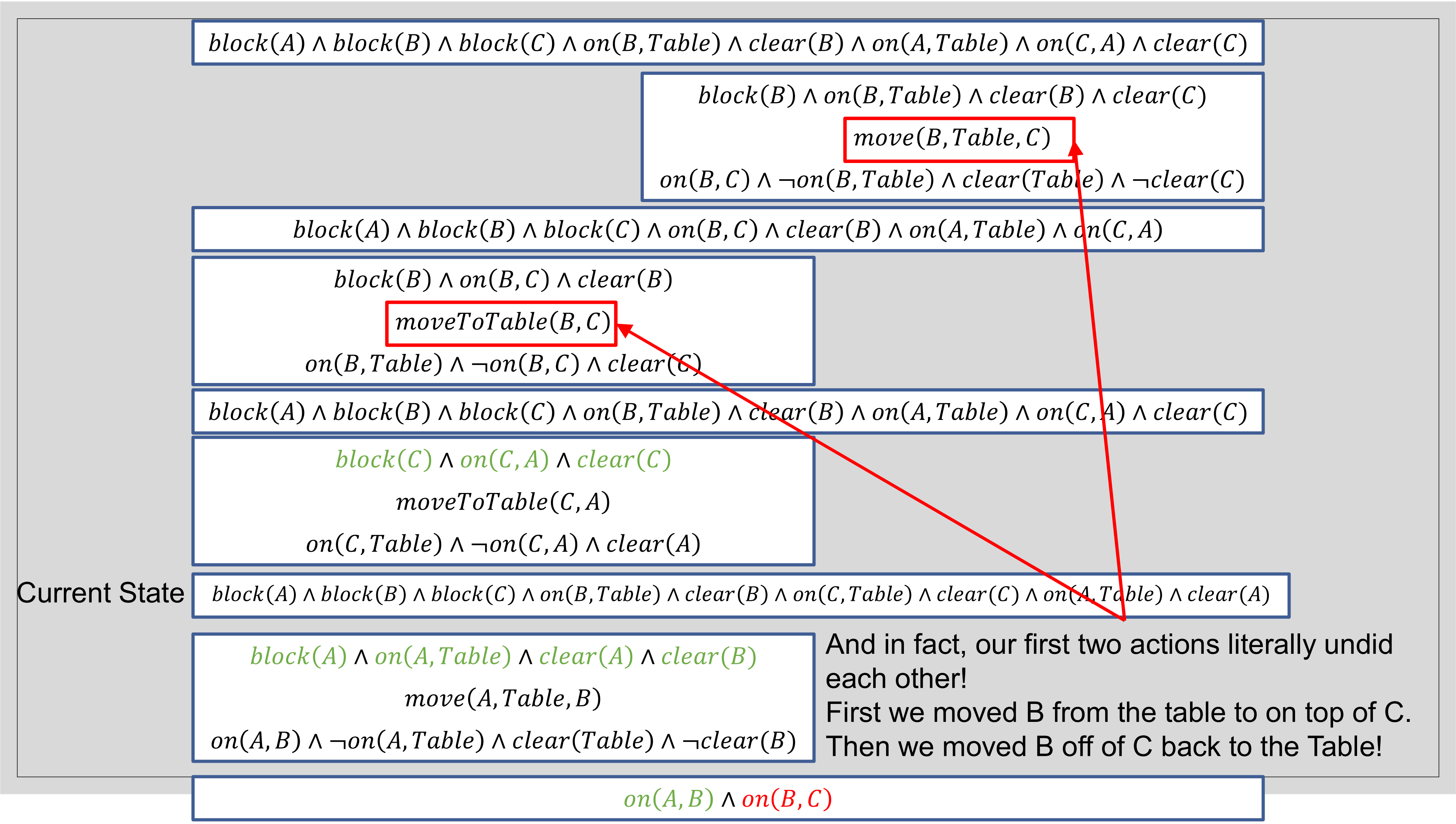
$block(A) \land on(A,Table) \land clear(A) \land clear(B)$

$move(A,Table,B)$

$on(A,B) \land \neg on(A,Table) \land clear(Table) \land \neg clear(B)$

We un-did the good part of our first action! Part of our goal was undone! We took B off of C!

$on(A,B) \land on(B,C)$



$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(B) \land on(B,Table) \land clear(B) \land clear(C)$

$\boxed{move(B,Table,C)}$

$on(B,C) \land \neg on(B,Table) \land clear(Table) \land \neg clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,C) \land clear(B) \land on(A,Table) \land on(C,A)$

$block(B) \land on(B,C) \land clear(B)$

$\boxed{moveToTable(B,C)}$

$on(B,Table) \land \neg on(B,C) \land clear(C)$

$block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(A,Table) \land on(C,A) \land clear(C)$

$block(C) \land on(C,A) \land clear(C)$

$moveToTable(C,A)$

$on(C,Table) \land \neg on(C,A) \land clear(A)$

Current State $block(A) \land block(B) \land block(C) \land on(B,Table) \land clear(B) \land on(C,Table) \land clear(C) \land on(A,Table) \land clear(A)$

$block(A) \land on(A,Table) \land clear(A) \land clear(B)$

$move(A,Table,B)$

$on(A,B) \land \neg on(A,Table) \land clear(Table) \land \neg clear(B)$

$on(A,B) \land on(B,C)$

And in fact, our first two actions literally undid each other!
First we moved B from the table to on top of C.
Then we moved B off of C back to the Table!

This Helps Illustrate Why Planning is Hard!

- We can break a planning problem down to individual goals…
  - “Have A on top of B”
  - Also, “Have B on top of C”

- But you can’t really solve those problems independently of each other!
  - Sometimes goals interfere with each other.
    - i.e., solving one undoes another, as we just saw.
  - Sometimes goals synergize with each other.
    - i.e., making progress on one helps you with another.
      - Still need to figure out how to recognize and leverage that!

So as we Discuss and Design Planners…

- We know a good planner has to…
  - Cope with goal interference.
  - Leverage goal synergy
  - Aggressively prune the search space through:
    - Abstraction (e.g., choosing what you represent)
    - Heuristics (e.g., being clever about what state to visit next).

So as we Discuss and Design Planners…

- And, as we’ve seen with so many things this semester…

- A planner is **sound** if it only produces valid plans (i.e., plans that are guaranteed to take you from the initial state to the goal, i.e. they work).

- A planner is **complete** if it is guaranteed to find a solution (i.e., a valid plan) if one exists.