Scheduling Project Presentations for next week (three presentations per class)

- March 22: Guest Lecture on Artificial Consciousness
- **March 24**: Aaron Moss; Dan Arnold; Geoff Davidson
- **March 26**: Adam Beck; Mat Roscoe; Paddy O'Brien

Final Project Presentations – 'Non-BRAHMS'

- **April 7**: Adam Beck; Mat Roscoe; Paddy O'Brien

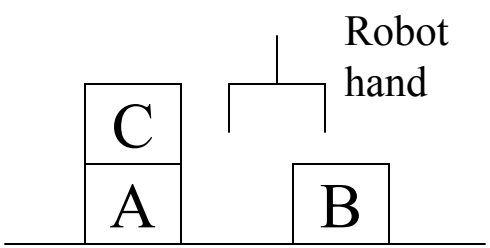
- **April 9**: Aaron Moss; Dan Arnold; Geoff Davidson

Tomorrow (March 18) from 9-10AM in Head Hall 135 discussion of projects

Everyone – also the BRAHMS Projects!

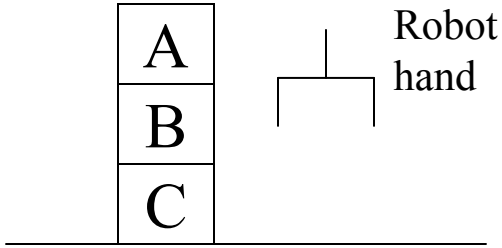
Stanford research Institute Problem Solver

•STRIPS : A planning system – Has rules with precondition deletion list and addition list



START

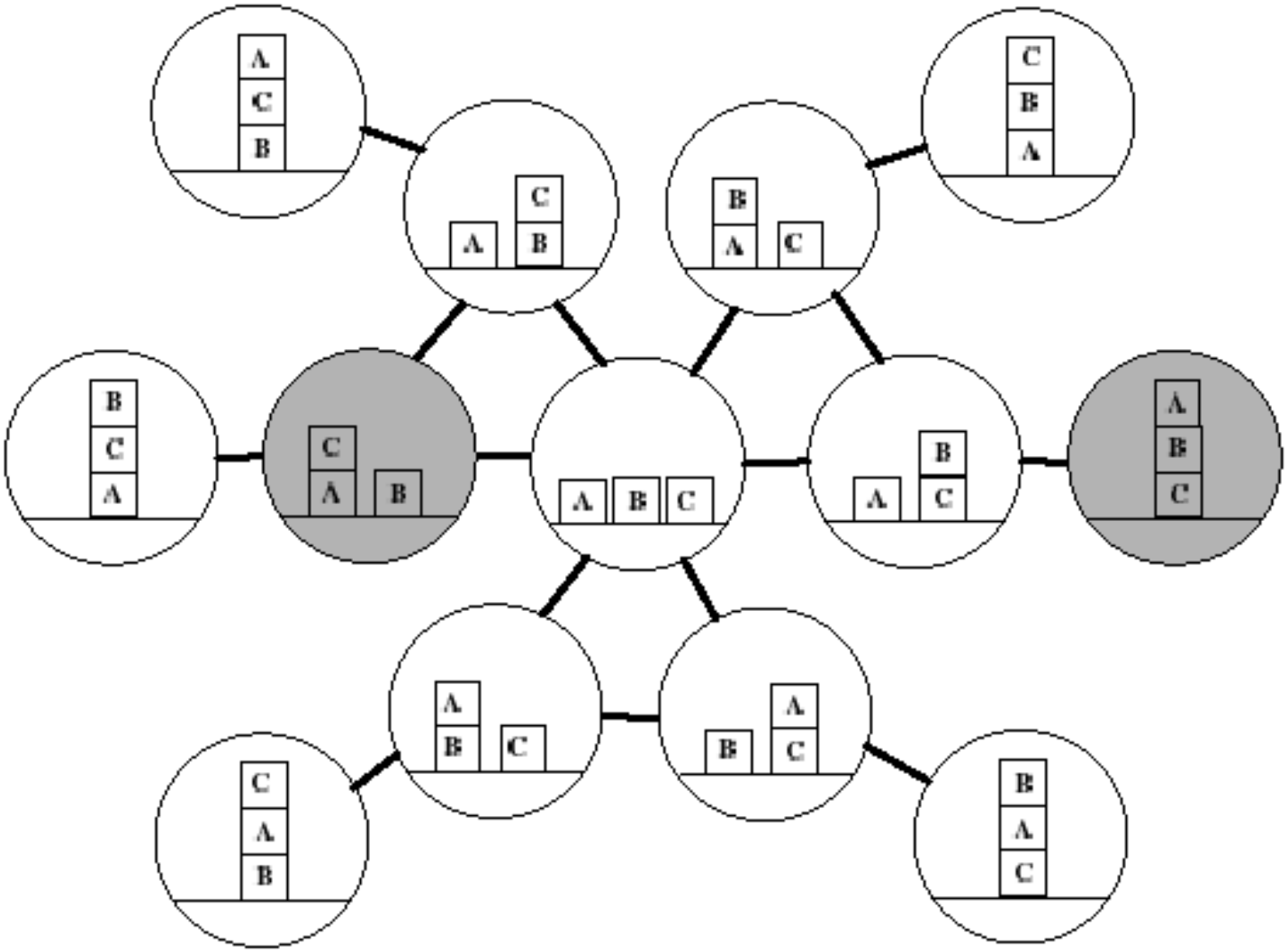
on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)



GOAL

on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

State Space: Blocks World



Rules

- *R1 :* ***pickup(x)***

  Precondition & Deletion List : handempty, on(x,table), clear(x)

  Add List (Postcondition) : holding(x)

- *R2 :* ***putdown(x)***

  Precondition & Deletion List : holding(x)

  Add List (Postconditions) : handempty, on(x,table), clear(x)

Rules

- *R3 :* ***stack(x,y)***

  Precondition & Deletion List :holding(x), clear(y)

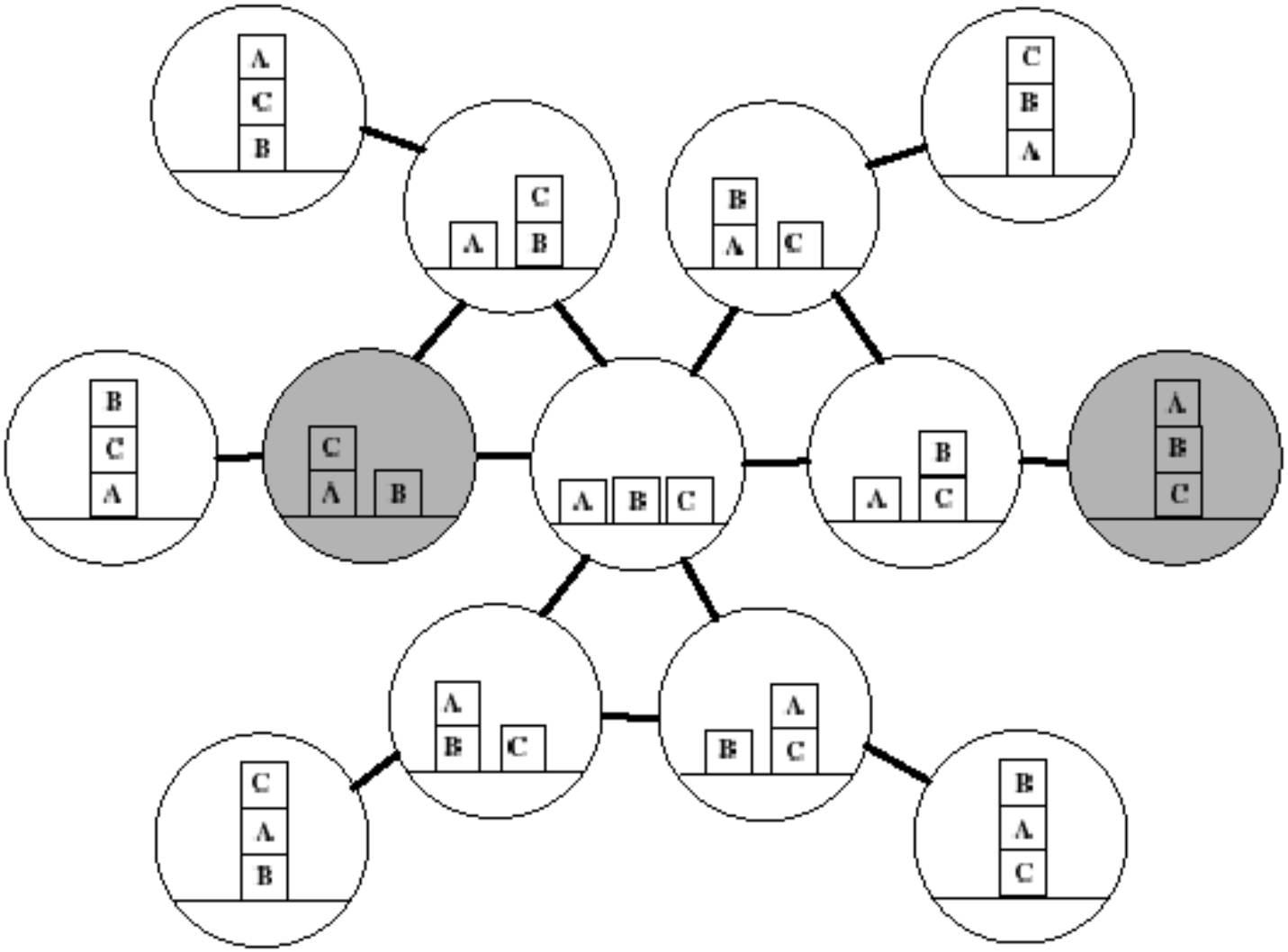
  Add List (Postconditions) : on(x,y), clear(x), handempty

- *R4 :* ***unstack(x,y)***

  Precondition & Deletion List : on(x,y), clear(x),handempty

  Add List (Postconditions) : holding(x), clear(y)

State Space: Blocks World



Triangular Table

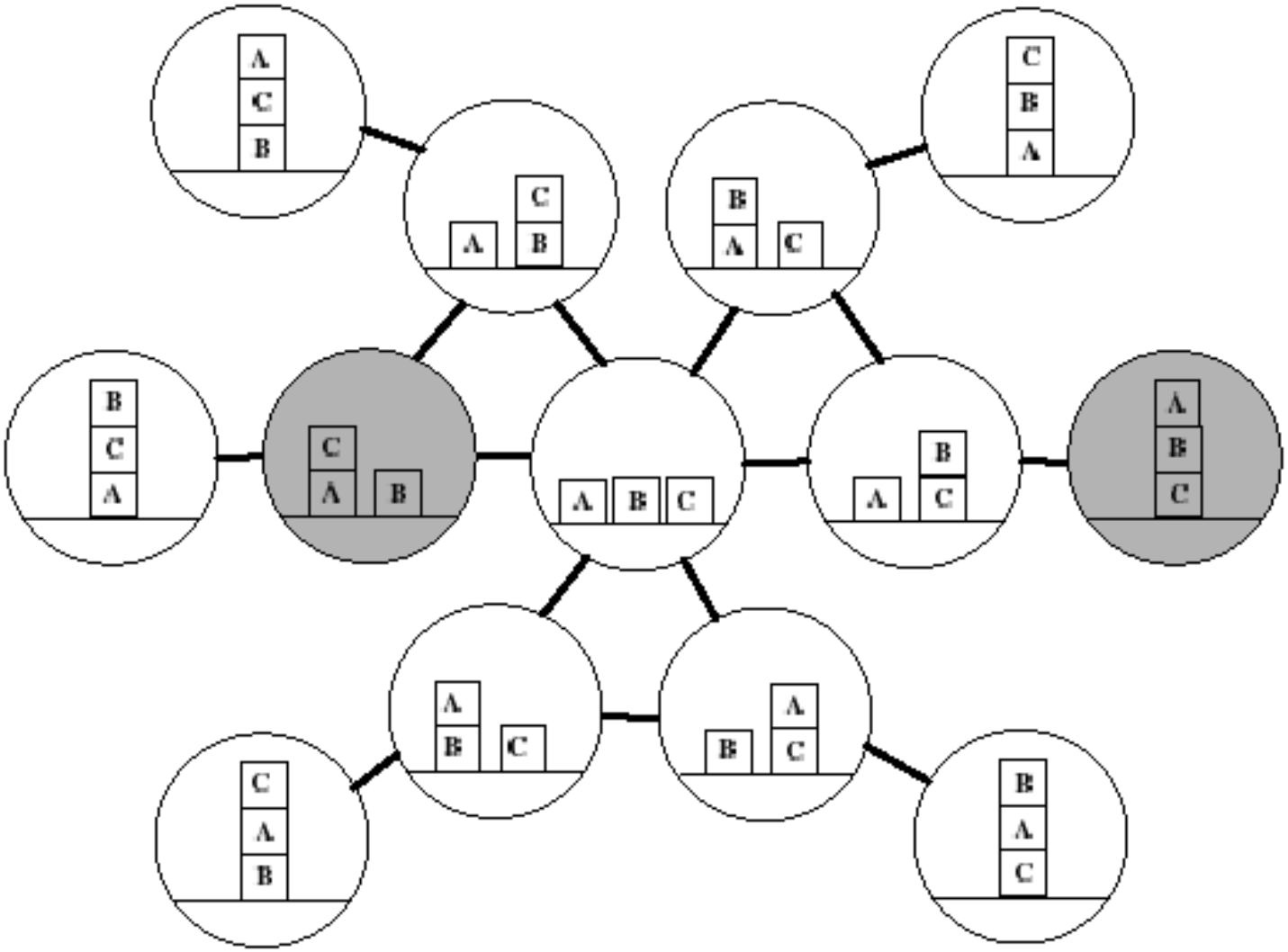
1	on(C,A) clear(C) hand empty	unstack(C,A)				
2		holding(C)	putdown(C)			
3	on(B,table)		hand empty	pickup(B)		
4			clear(C)	holding(B)	stack(B,C)	
5	on(A,table)	clear(A)			hand empty	pickup(A)
6					clear(B)	holding(A)	stack(A,B)
7			on(C,table)		on(B,C)		on(A,B) clear(A)
	0	1	2	3	4	5	6

Triangular Table

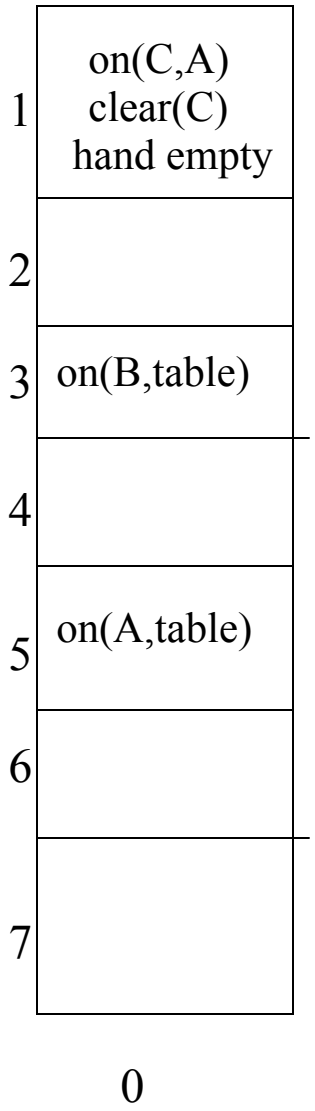
- For n operations in the plan, there are :
  - (n+1) rows : 1 → n+1
  - (n+1) columns : 0 → n
- At the end of the i^{th} row, place the i^{th} component of the plan.
- The row entries for the i^{th} step contain the pre-conditions for the i^{th} operation.
- The column entries for the j^{th} column contain the add list for the rule on the top.
- The $<i,j>^{th}$ cell (where $1 \leq i \leq n+1$ and $0 \leq j \leq n$) contain the pre-conditions for the i^{th} operation that are added by the j^{th} operation.
- The first column indicates the starting state and the last row indicates the goal state.

Connection between triangular matrix and state space search

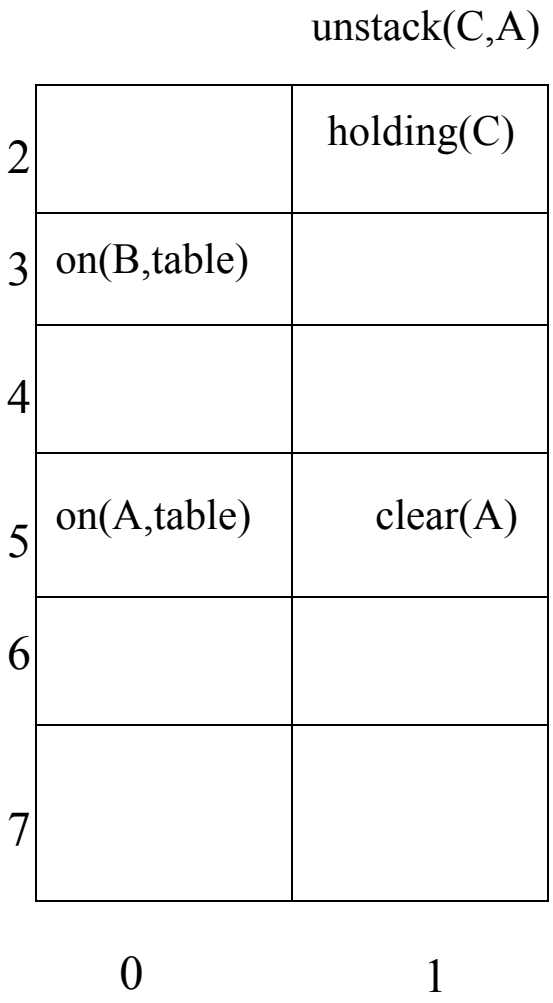
State Space: Blocks World



Kernel 0 = S_0 (starting state)



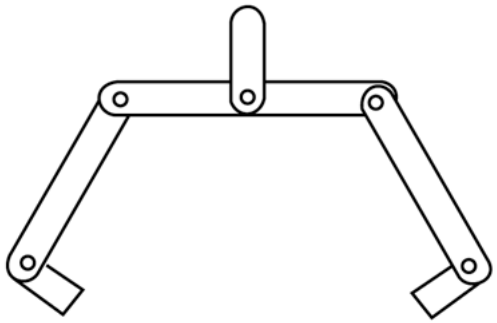
Kernel 1= State S_1

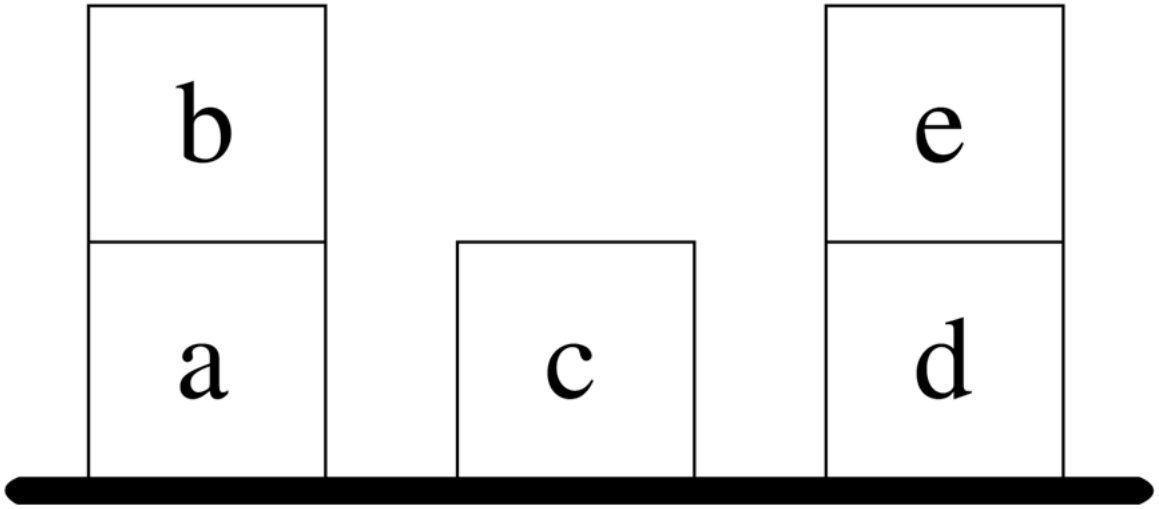


Importane of Kernel

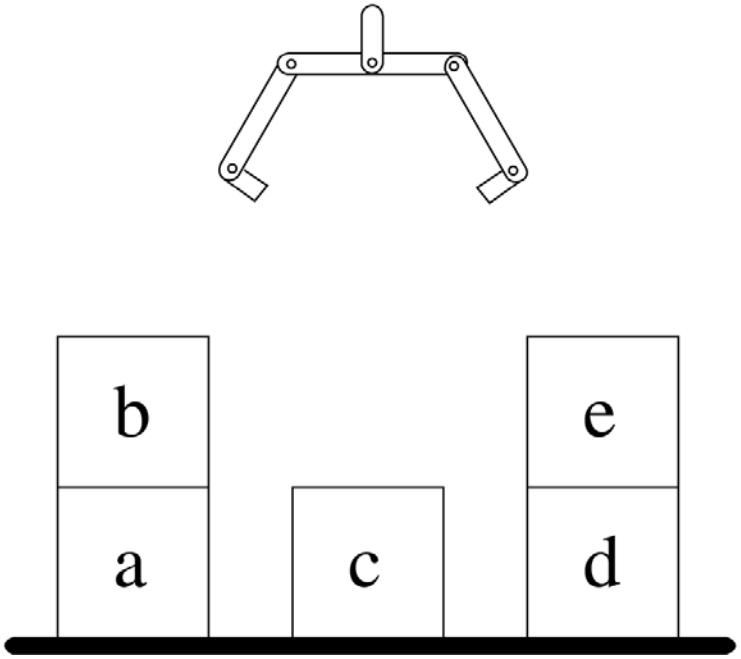
- The kernel in the table controls the execution of the plan
- At any step of the execution the current state as given by the sensors is matched with the largest kernel in the perceptual world of the robot, described by the table

The blocks world





Represent this world using predicates

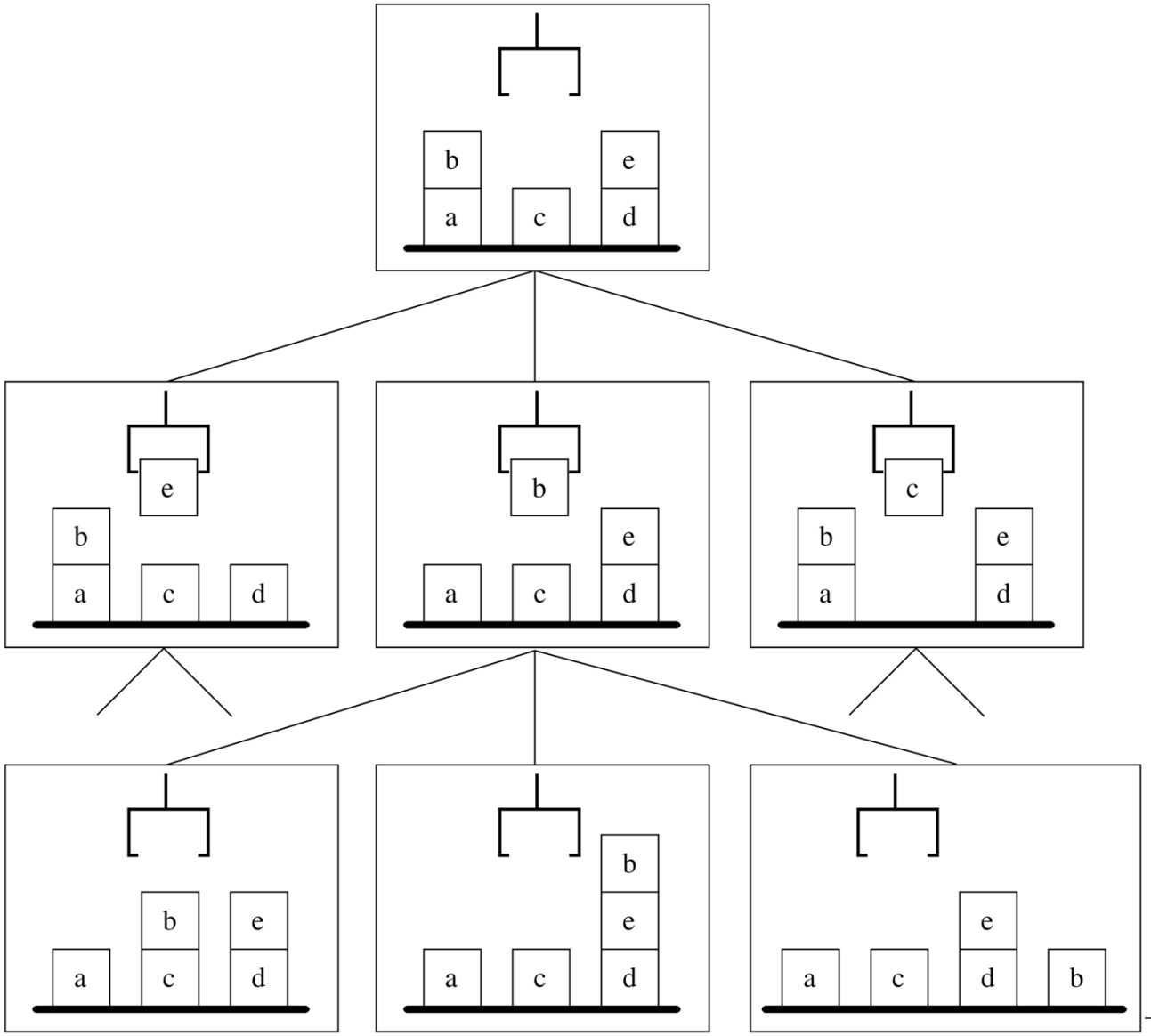


- ontable(a)
  ontable(c)
  ontable(d)
  on(b,a)
  on(e,d)
  clear(b)
  clear(c)
  clear(e)
  gripping()

The robot arm can perform these tasks

- pickup (W): pick up block W from its current location on the table and hold it
- putdown (W): place block W on the table
- stack (U, V): place block U on top of block V
- unstack (U, V): remove block U from the top of block V and hold it
- All assume that the robot arm can precisely reach the block.

Portion of the search space of the blocks world example



The STRIPS representation

- Special purpose representation.
- An operator is defined in terms of its:
- name,
  parameters,
  preconditions, and
  results.
- A planner is a special purpose algorithm, i.e., it's not a general purpose logic theorem prover.

Four operators for the blocks world

- pickup(X)
  - P: gripping() ∧ clear(X) ∧ ontable(X)
  - A: gripping(X)
  - D: ontable(X) ∧ gripping()
- putdown(X)
  - P: gripping(X)
  - A: ontable(X) ∧ gripping() ∧ clear(X)
  - D: gripping(X)
- stack(X,Y)
  - P: gripping(X) ∧ clear(Y)
  - A: on(X,Y) ∧ gripping() ∧ clear(X)
  - D: gripping(X) ∧ clear(Y)
- unstack(X,Y)
  - P: gripping() ∧ clear(X) ∧ on(X,Y)
  - A: gripping(X) ∧ clear(Y)
  - D: on(X,Y) ∧ gripping()

Notice the simplification

- *Preconditions*, *add lists*, and *delete lists* are all conjunctions.

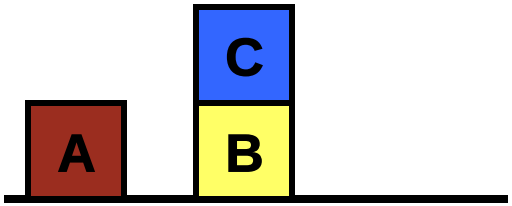
We don't have the full power of predicate logic.

- The same applies to *goals*. Goals are conjunctions of predicates.
- A detail:

Why do we have two operators for picking up (pickup and unstack), and two for putting down (putdown and stack)?

Situation calculus - example

- blocks world
  - tabletop and three blocks
  - actions and situations



} objects/terms in FOL

eternal

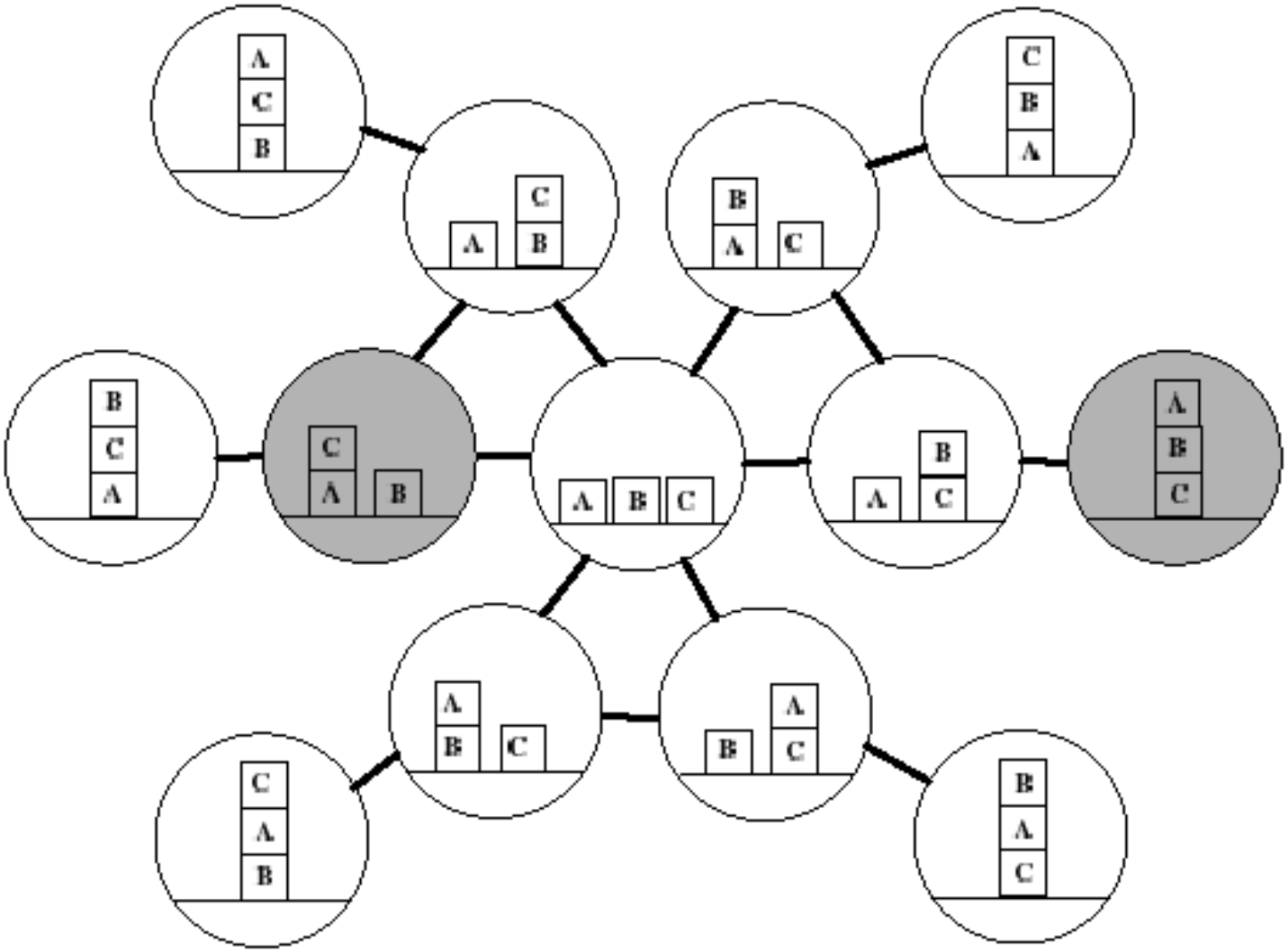
- Table(x)
- Block(x)

fluent

- On(x,y,s)
- ClearTop(x,s)

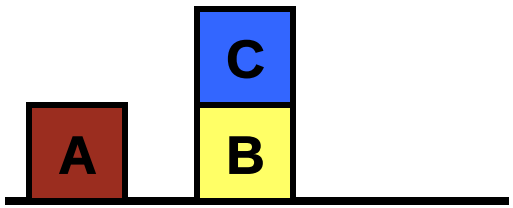
s is situation variable

State Space: Blocks World



Situation calculus - example

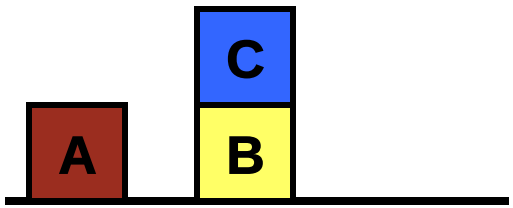
- actions are functions (objects)
- situations are objects



- PutOn(x,y)
  - preconditions:
    - ClearTop(x,s)
    - ClearTop(y,s) V
      Table(y)
  - effect:
    - On(x,y,Result(PutOn(x,y),s))

Situation calculus - example

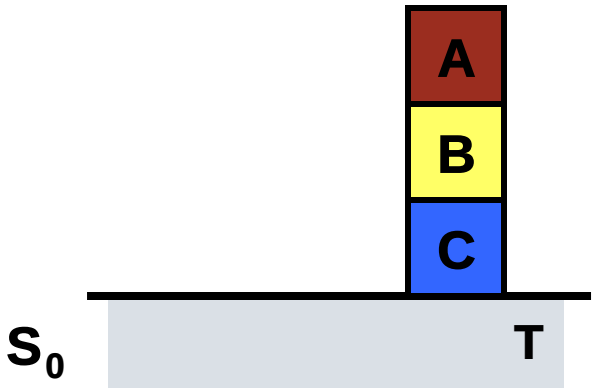
- each situation is a function of the previous one – Result function



- Result(a,s)
  - preconditions:
    - action a can be applied at s
  - effect:
    - Result is next situation after a is applied at s

Situation calculus - example

- e.g. KB:
  - function PutOn(x,y)
  - $\forall x\ (\sim \exists y\ On(y,x,s)) \Rightarrow ClearTop(x,s)$
  - $\forall x,y,s\ ClearTop(x,s) \wedge$
    $(ClearTop(y,s) \vee Table(y))$
    $\Rightarrow On(x,y,Result(PutOn(x,y),s))$
  - constants: A, B, C, T, S_0, S_1, S_2,…
  - Table(T), Block(A), Block(B), Block(C)
  - $On(A,B,S_0)$, $On(B,C,S_0)$, $On(C,T,S_0)$



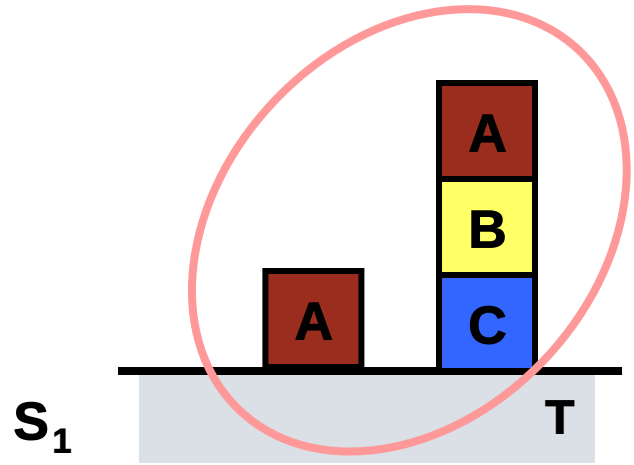
Situation calculus - example

- action: PutOn(A,T)
  - preconditions: ClearTop(A,S_0),Table(T)
  - effect: On(A,T,Result(PutOn(A,T),S_0))

BUT…

what happens to other fluents?

- some propagated, some not



Situation calculus - example

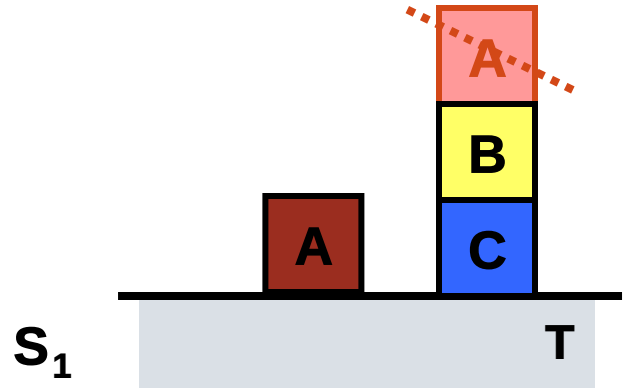
- 'On' axiom:

$\forall x,y,z,a,s\ On(x,y,Result(a,s)) \Leftrightarrow$

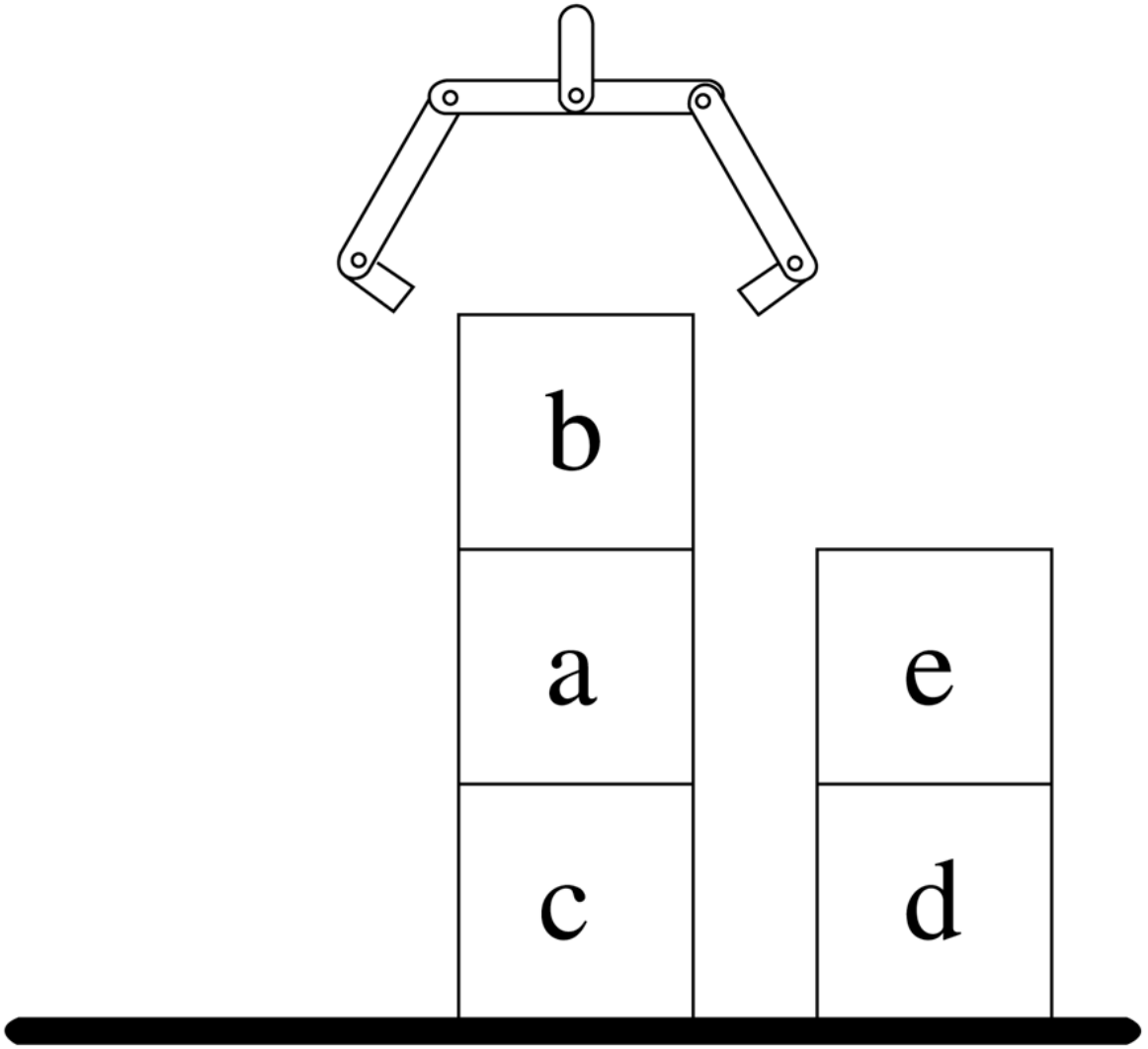
$[ClearTop(x,s) \wedge (ClearTop(y,s) \vee Table(y)) \wedge$

$a = PutOn(x,y)$

$\vee\ [\ On(x,y,s) \wedge \sim(a = PutOn(x,z))]$



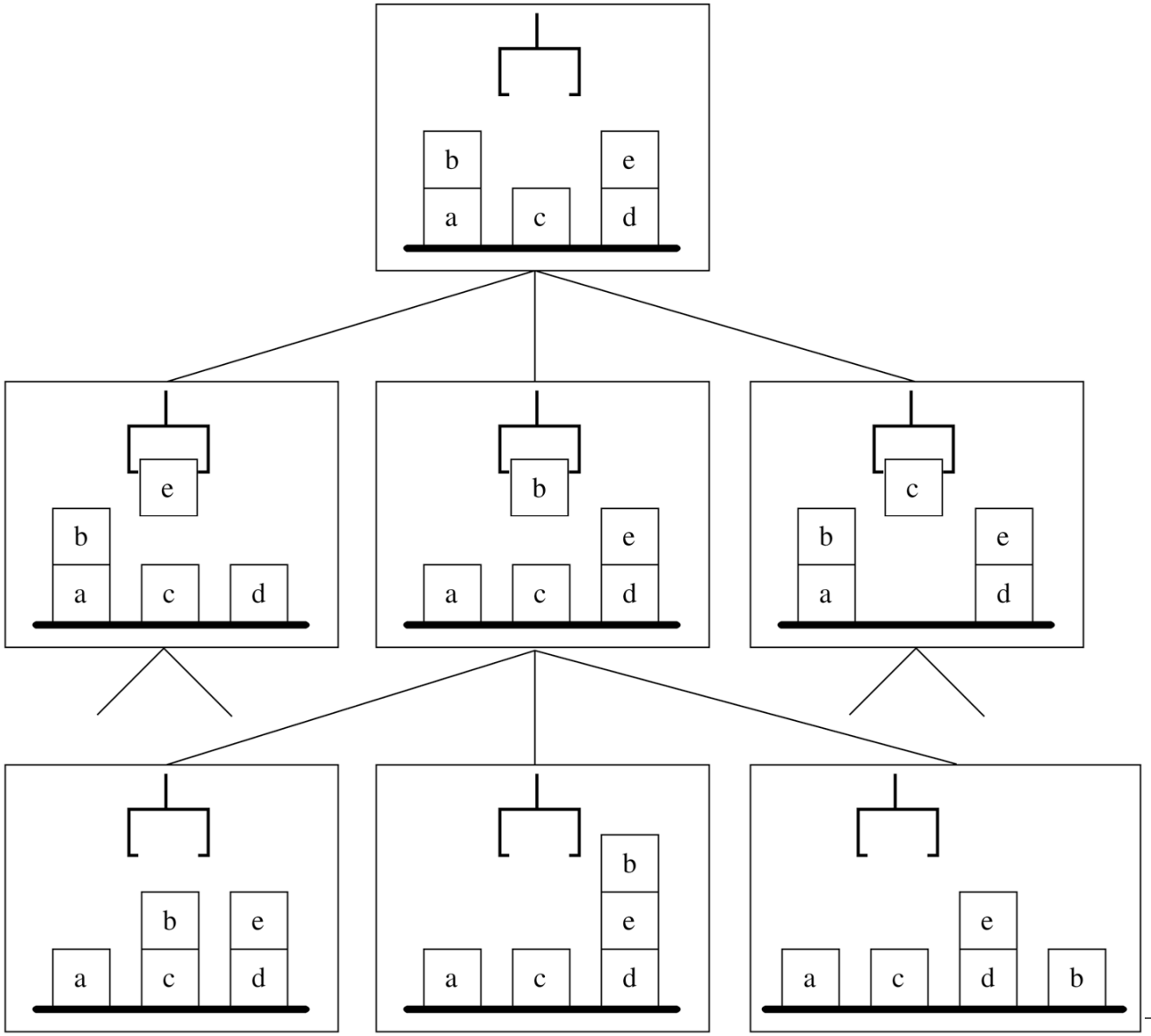
A goal state for the blocks world



A state space algorithm for STRIPS operators

- Search the space of situations (or states). This means each node in the search tree is a state.
- The root of the tree is the start state.
- Operators are the means of transition from each node to its children.
- The goal test involves seeing if the set of goals is a subset of the current situation.

Now, the following graph makes much more sense



Problems in representation

- *Frame problem*: List everything that does not change. It is no more a significant problem because what is not listed as changing (via the add and delete lists) is assumed to be not changing.
- *Qualification problem*: Can we list every precondition for an action? For instance, in order for PICKUP to work, the block should not be glued to the table, it should not be nailed to the table, …
- It still is a problem. A partial solution is to prioritize preconditions, i.e., separate out the preconditions that are worth achieving.

Problems in representation (cont'd)

- *Ramification problem*: Can we list every result of an action? For instance, if a block is picked up its shadow changes location, the weight on the table decreases, …
- It still is a problem. A partial solution is to code rules so that inferences can be made. For instance, allow rules to calculate where the shadow would be, given the positions of the light source and the object. When the position of the object changes, its shadow changes too.

A sampler of planning algorithms

- Forward chaining
  - Work in a state space
  - Start with the initial state, try to reach the goal state using forward progression
- Backward chaining
  - Work in a state space
  - Start with the goal state, try to reach the initial state using backward regression
- Partial order planning
  - Work in a plan space
  - Start with an empty plan, work from the goal to reach a complete plan

State Space Searching

Progression Planners

- Search **top-down** from initial state to the goal state.
- This algorithm will build a path from the **initial state** to the **goal**.
- The algorithm also **keep a record** of the plan it has built at any stage to the current state.

State Space Searching

Progression Planners (cont)

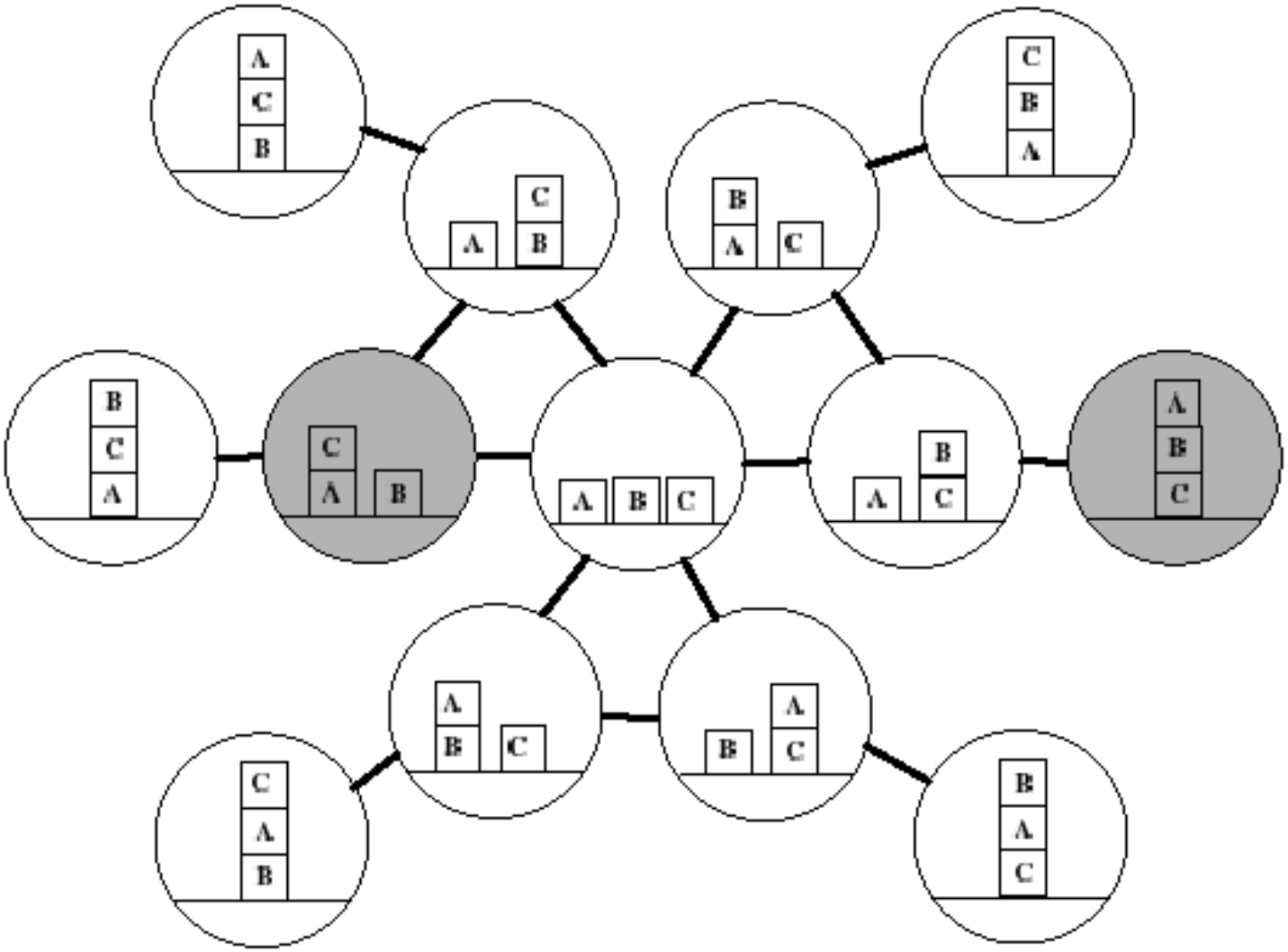
- Progression planners can use **any of the search methods**, both blind and heuristic incline.

- A depth-first search algorithm is summarised below:

1. If the current state *S* satisfies the goal then return the path.

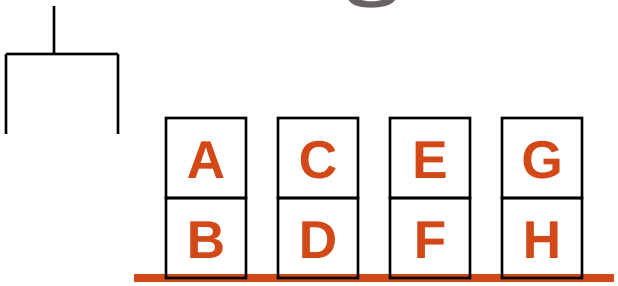
2. Else,
   (a) try and pick an appropriate action *A* whose precondition is satisfied by the current world state.
   (b) if there is no such action, then backtrack.
   (c) else, move to the state in the search space *S'* that would result from performing that action in the current state *S*,
   and then find a path (plan) *P'* that goes from that new state to the goal.
   Returning the complete path *P* from *S* to the goal (where *P* = [*A*|*P'*], the list of actions consisting of *A* followed by *P*).

State Space: Blocks World

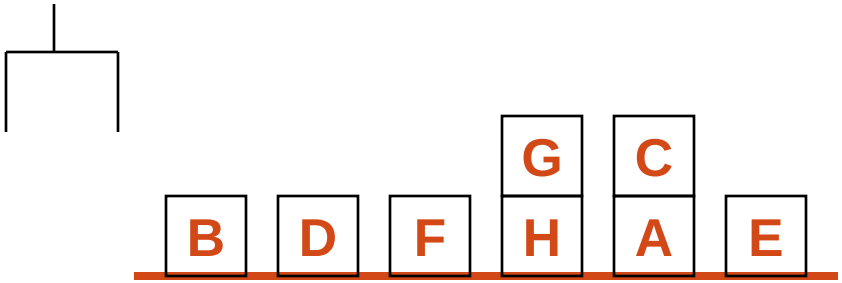


Forward chaining

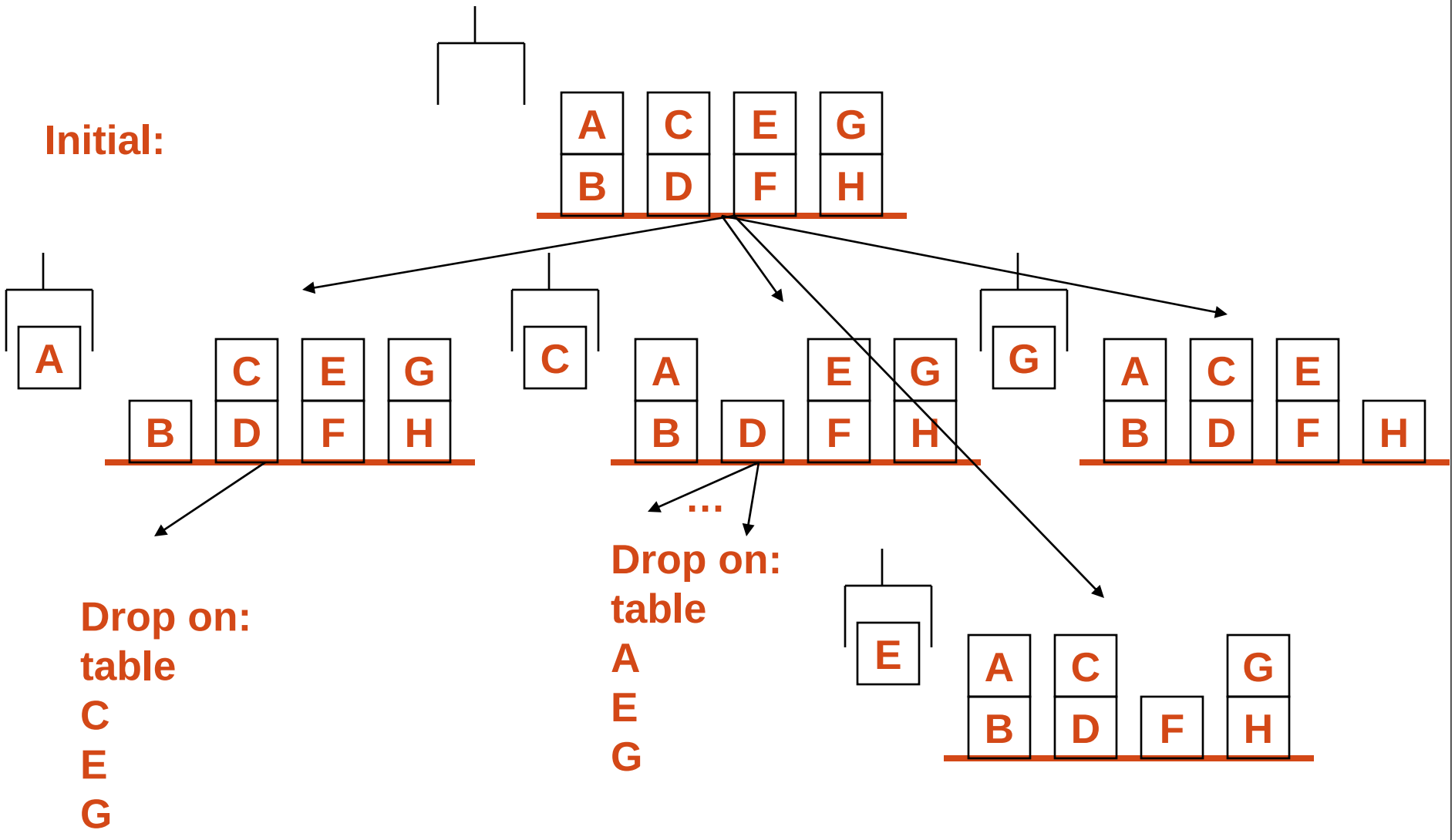
Initial:



Goal :



1st and 2nd levels of search



Results

- A plan is:
  - unstack (A, B)
  - putdown (A)
  - unstack (C, D)
  - stack (C, A)
  - unstack (E, F)
  - putdown (F)
- Notice that the final locations of D, F, G, and H need not be specified
- Also notice that D, F, G, and H will never need to be moved. But there are states in the search space which are a result of moving these. Working backwards from the goal might help.

State Space Searching

Regression Planners

- Search **bottom-up** from the goal state to the initial state.
- This algorithm builds a path from the **goal** to the **initial state**.
- The algorithm also **keeps a record** of the plan it has built at any stage to the current state.

State Space Searching

Regression Planners (cont)

- Similarly regression planners can use **any of the search methods**.

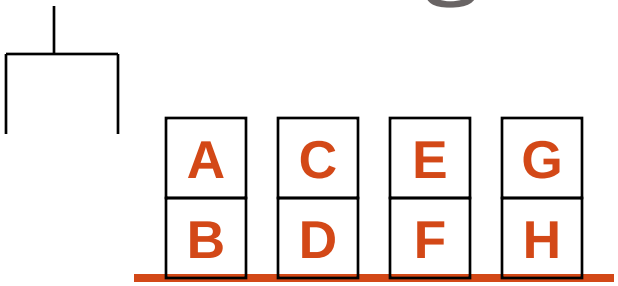
- A depth-first search algorithm is summarised below:

1. If the current state *S* satisfies the goal then return the path.

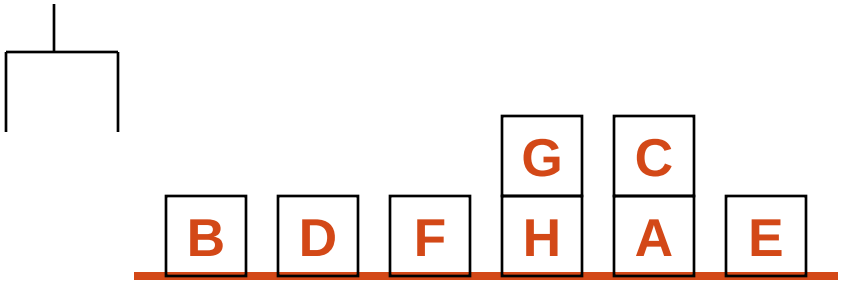
2. Else,
   (a) try and pick an appropriate action *A* whose effect is satisfied by the current world state.
   (b) if there is no such action, then backtrack.
   (c) else, move to an appropriate state in the search space *S'* that would satisfy the preconditions of the action *A*, and that would result in the state *S*, if *A* was performed, and then recursively find a path (plan) *P'* that goes from that new state to the initial state.
   Returning the complete path *P* from *S* to the initial state (where *P* = [*A*|*P'*], the list of actions consisting of *A* followed by *P*).

Backward chaining

Initial:



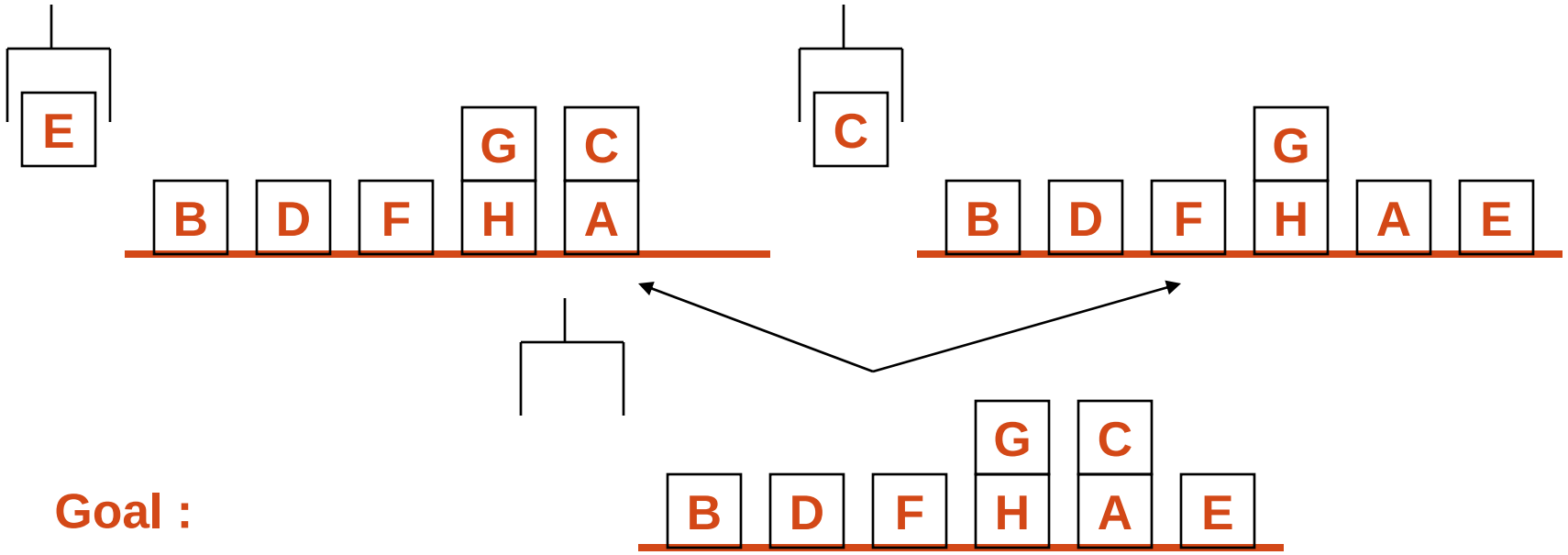
Goal :



1st level of search

For E to be on the table, the last action must be putdown(E)

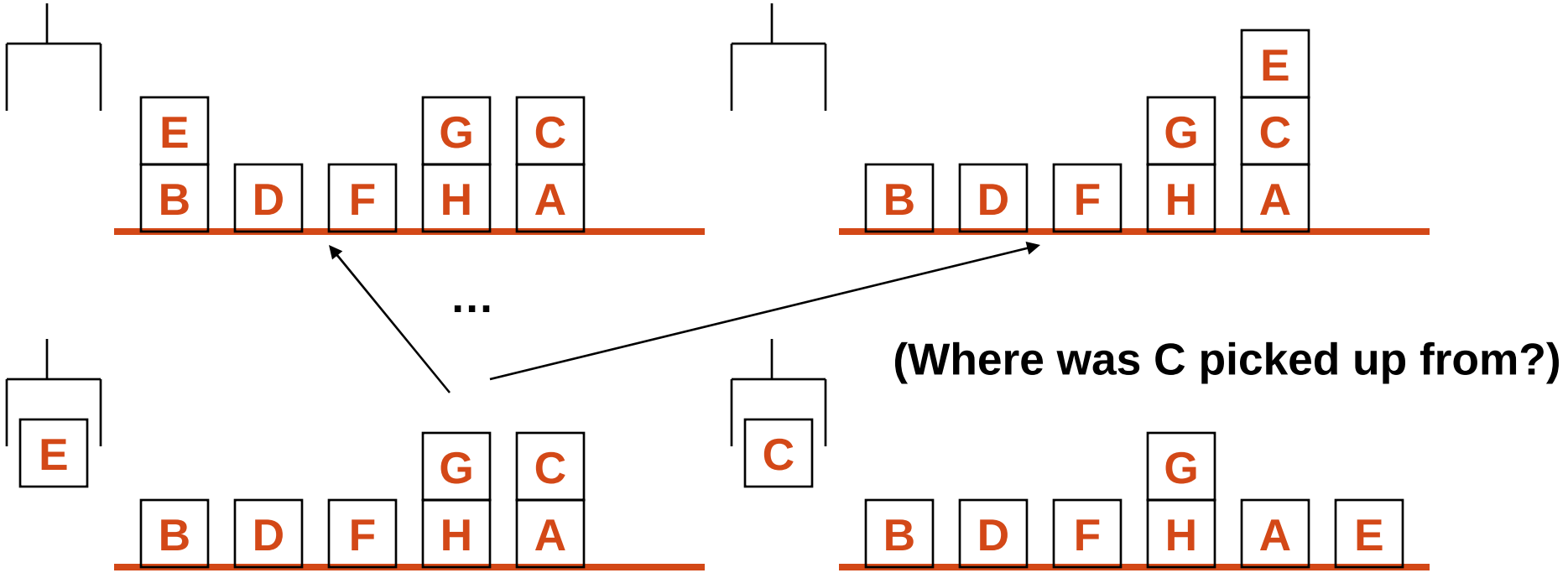
For C to be on A, the last action must be stack(C,A)



Goal :

2nd level of search

Where was E picked up from?



Results

- The same plan can be found
  - unstack (A, B)
  - putdown (A)
  - unstack (C, D)
  - stack (C, A)
  - unstack (E, F)
  - putdown (F)
- Now, the final locations of D, F, G, and H need to be specified
- Notice that D, F, G, and H will never need to be moved. But observe that from the second level on the branching factor is still high

Plan Space Searching

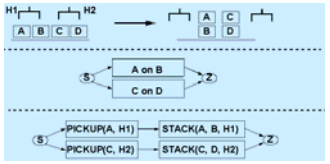
- An alternative way of viewing the planning problem is to see it as a **search through possible plans**.
- The main motivation for plan space searching is to **avoid back-tracking** by looking at the goals in an order different from execution order.
- Search space consisting of **states** of the world are **linked by actions**.

Plan Space Searching (cont)

- Plan space is a **collection** of **partially specified plans** linked by operators that refine a plan into a more detailed one.
- **The initial plan**, consists of some **unspecified actions** that takes the initial state into the goal state.
- The goal will be **fully specified plan** (or plans) that performs the desired function.

Partial Ordered Planning

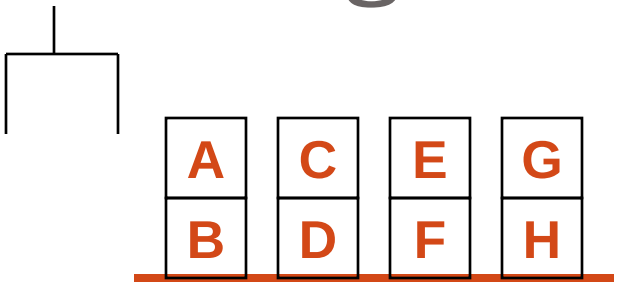
Introduction



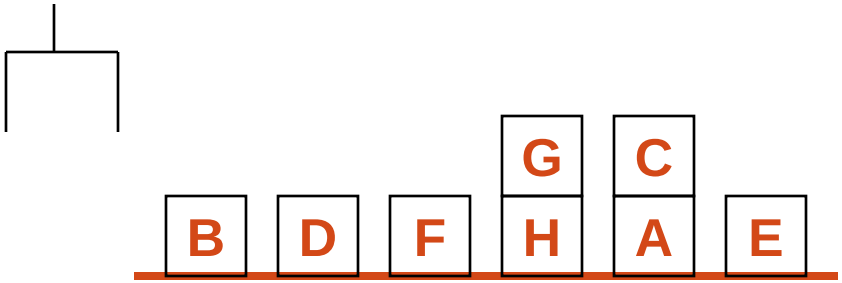
- A partially ordered plan is a general representation of plans.
- Idea:
  - Working parallel on several sub-goals.
  - Ordering of goals based on interactions.
- Underlying assumption:
  - Not many interactions.
- Partially ordered plan = directed graph (AND).

Forward chaining

Initial:



Goal :



Partial-order planning (POP)

- Notice that the resulting plan has two parallelizable threads:

unstack (A,B)
putdown (A)
unstack (C,D)
stack (C,A)

&

unstack (E, F)
putdown (F)

- These steps can be interleaved in 3 different ways:

unstack (E, F)
putdown (F)
unstack (A,B)
putdown (A)

unstack (A,B)
putdown (A)
unstack (E, F)
putdown (F)

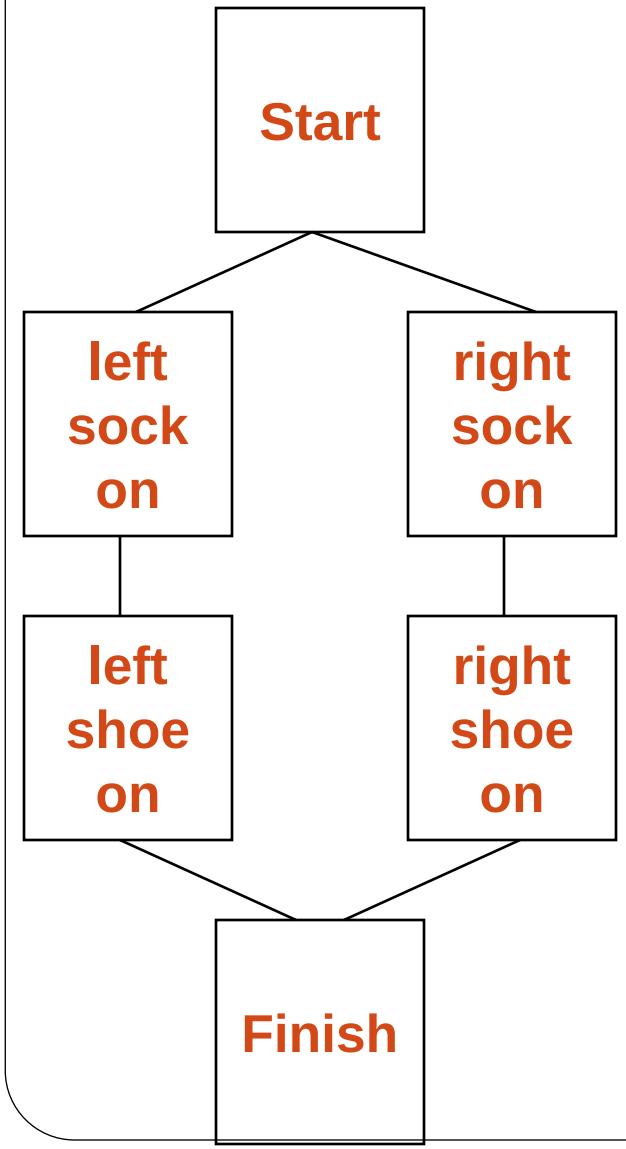
unstack (A,B)
putdown (A)
unstack (C,D)
stack (C,A)

unstack (C,D)
stack (C,A)
unstack (E, F)
putdown (F)

Partial-order planning (cont'd)

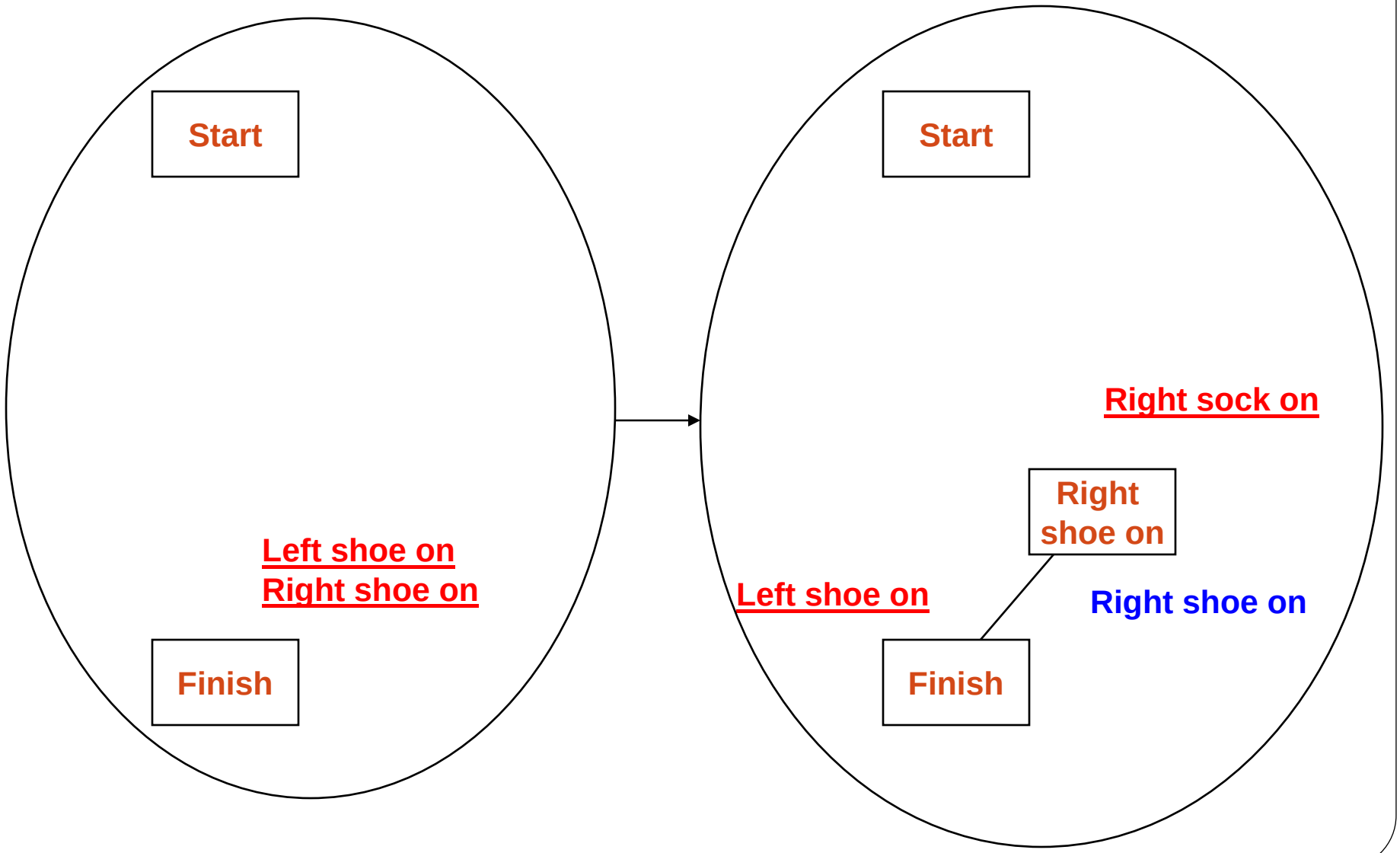
- Idea: Do not order steps unless it is necessary
- Then a partially ordered plan represents several totally ordered plans
- That decreases the search space
- But still the planning problem is not solved, good heuristics are crucial

Partial-order planning (cont'd)

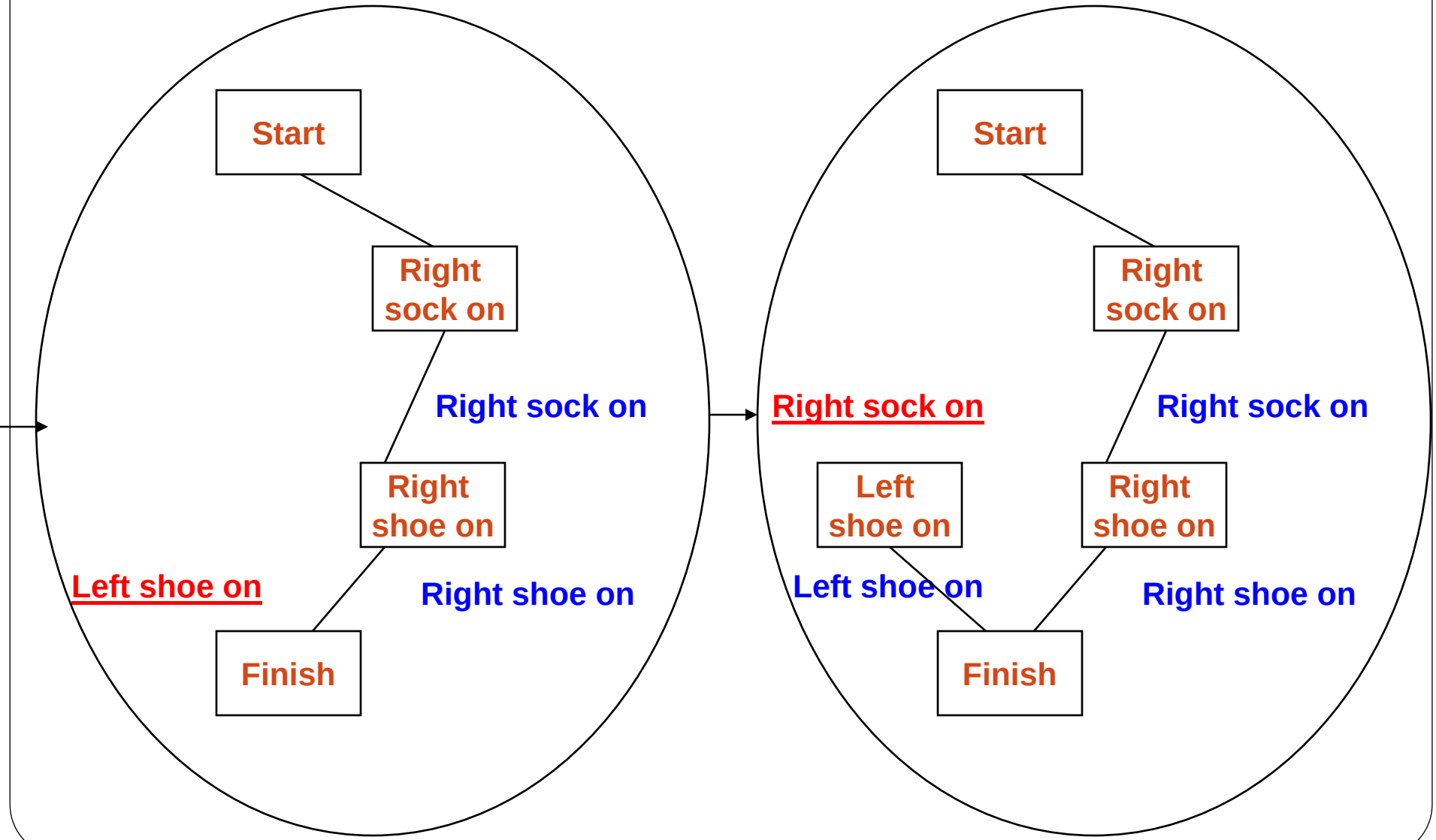


Start	Start	Start	Start	Start	Start
Left sock on	Right sock on	Left sock on	Right sock on	Left sock on	Right sock on
Left shoe on	Right shoe on	Right sock on	Left sock on	Right sock on	Left sock on
Right sock on	Left sock on	Left shoe on	Right shoe on	Right shoe on	Left shoe on
Right shoe on	Left shoe on	Right shoe on	Left shoe on	Left shoe on	Right shoe on
Finish	Finish	Finish	Finish	Finish	Finish

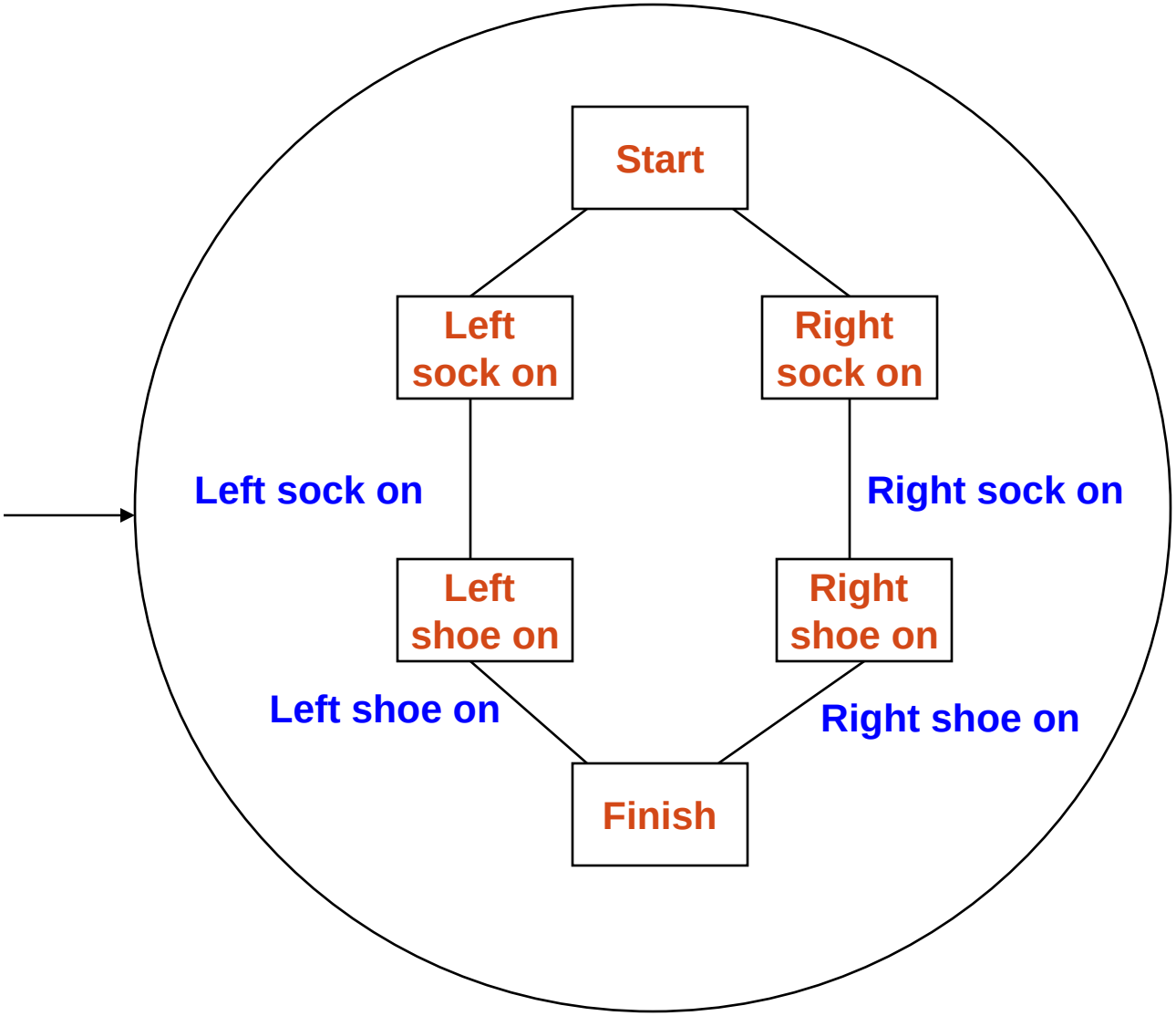
POP plan generation



POP plan generation (cont'd)



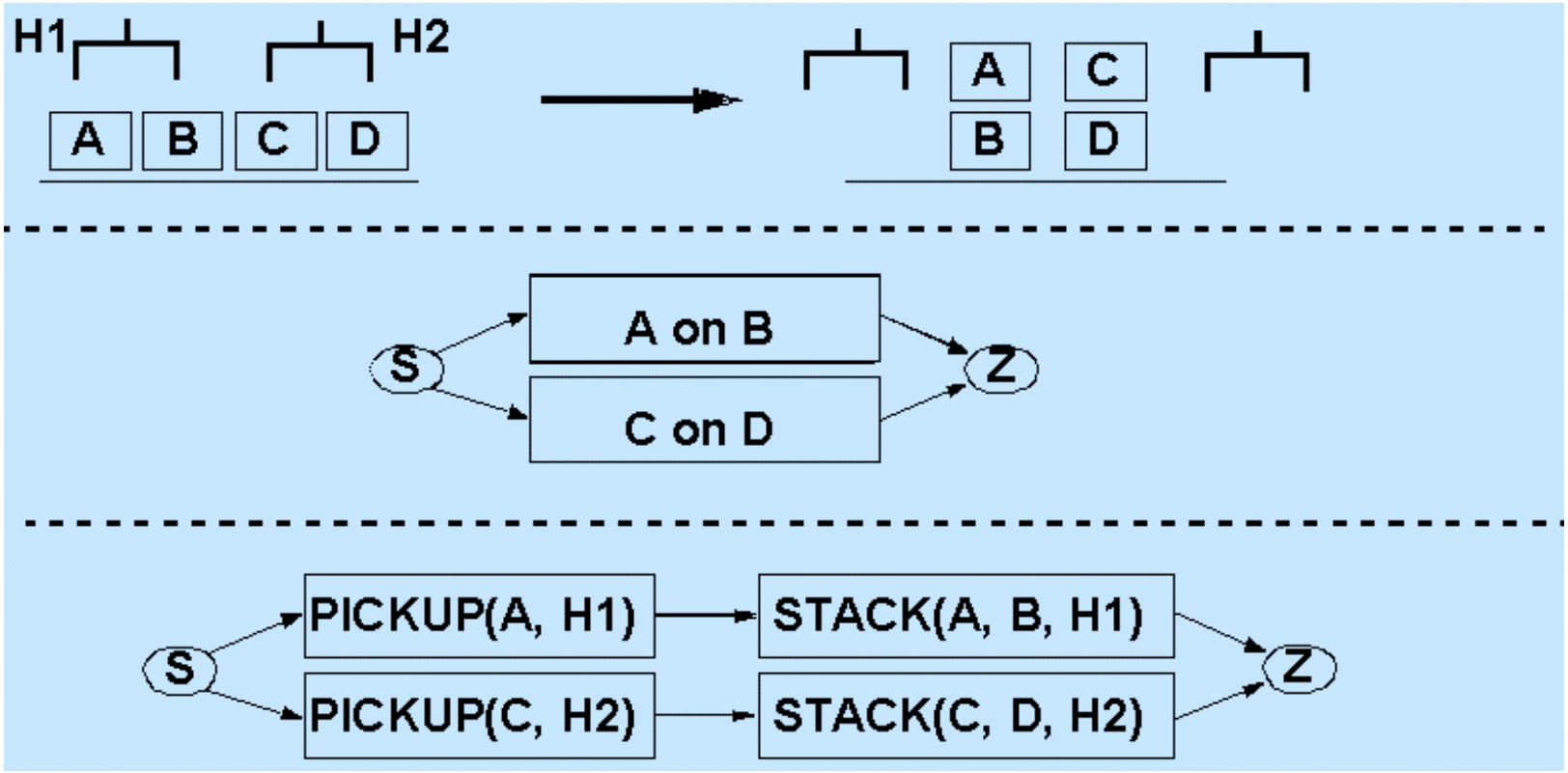
POP plan generation (cont'd)



DONE!

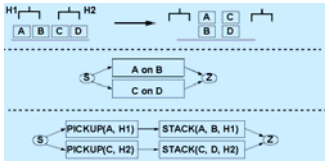
Partial Ordered Planning

An Example



Partial Ordered Planning

Interpretation

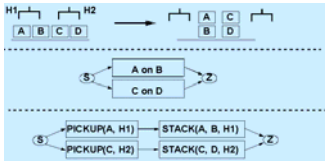


- An ordered pair **P = (O,<)** is a **plan** :⇔ **O** is a set of **nodes**, **<** is a strict **partial order** on O with definite smallest element *Start(P)* and definite largest element *Goal(P)*

- A partially ordered plan can be executed in **any total order** that is **compatible** with the **partial order**.
  - PICKUP(A, H1), PICKUP(C, H2), Stack(A,B,H1), Stack(C,D,H2)
  - PICKUP(A, H1), Stack(A,B,H1), PICKUP(C, H2), Stack(C,D,H2)
  - not ok: PICKUP(A, H1), Stack(A,B,H1), Stack(C,D,H2), PICKUP(C, H2)

Partial Ordered Planning

Interpretation (cont)



- **Stronger interpretation** (less execution possibilities): Parallel branches can be ordered only in total.

- **Weaker interpretation** (more execution possibilities): Parallel execution allowed.

Partially ordered *plans* vs. Non-linear *planning*

- Representation of the plan: partially ordered plan

- Inserting parts of plans at an arbitrary location: non linear planning

Shortcomings of AI Planning in General

- Not every actions can be described with STRIPS-like operators:
  - money transfer: new balance is a function of the old
  - alternative post-conditions
    - PAINTBLACK(x)
      precondition: x is white
      (Why not blue? to know what to delete!)

- No complete knowledge about the world.

Shortcomings of AI Planning in General (cont)

- The world is not stable
  - Re-planning must be supported
- Goals are not clearly defined
- Nobody plans the solution of everyday tasks
- Humans learn