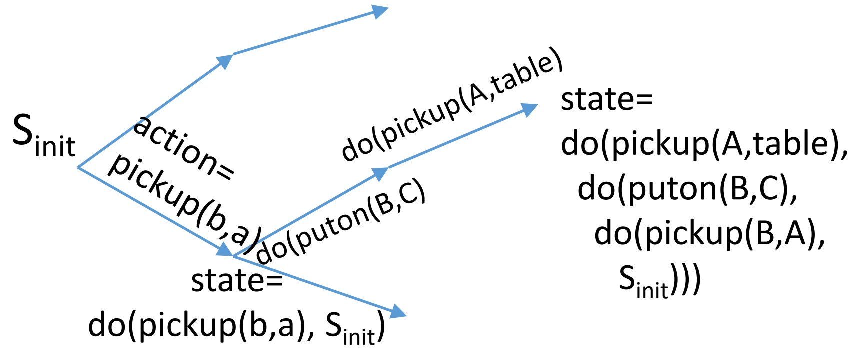
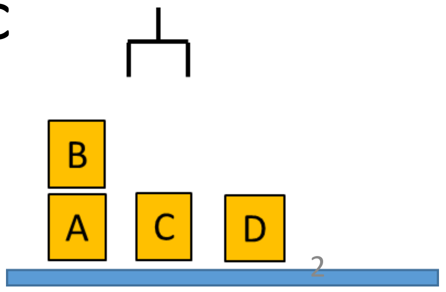
Planning (Ch. 11, skip 11.4-5; Sec 7.7)

- finding a *sequence of actions* to achieve goals
- requires reasoning about actions
- knowledge-level representation of the successor() function in search
- assumptions:
  - actions are discrete (state changes) and deterministic (no probability of failure)
  - goals are conjunctive (not disjunctive goals or maintenance goals, which require more complex algs)

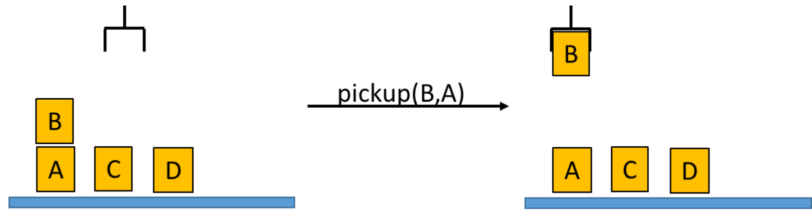
Situation Calculus



- for describing are reasoning about Actions in FOL
  - assume actions are <u>discrete state space</u>, fanning out from an initial state
  - add a 'situation' argument to each predicate (fluent)
  - could use S_{init} to refer to initial state
    - other states are denoted using the 'do' function, do(Act,State)
    - like anonymous names for all states based on action sequence
  - ∀s,x,y on(x,y,<u>s</u>)^clear(x,<u>s</u>)^ gripperEmpty(<u>s</u>) → holding(x,<u>do(pickup(x,y),s)</u>)^clear(y,<u>do(pickup(x,y),s)</u>)
  - axioms are universal rules over generic situations *s*
  - LHS=preconditions, RHS=effects



The Frame Problem



- The Frame Problem refers to the need to also specify all the things that *are not changed* by an action (p. 239, 249)
  - refers to animation frames or cells, background that remains constant
- for example, after we pickup(B,A), suppose we want to puton(B,C)
  - preconditions: must be holding B, C must be clear
  - holding(B) is a direct effect of pickup(B,A)
    - $\forall s,x,y\ on(x,y,\underline{s}) \wedge clear(x,\underline{s}) \wedge gripperEmpty(\underline{s}) \rightarrow holding(x,\underline{do(pickup(x,y),s)}) \wedge clear(y,\underline{do(pickup(x,y),s)})$
  - how do we know clear(C)??? not mentioned in rule for pickup(B,A), so how can we prove it is true in successor state?
- there are ways to do this (called writing 'Frame Axioms'):
  - $\forall s,x,y,z\ on(x,y,\underline{s}) \wedge clear(x,\underline{s}) \wedge gripperEmpty(\underline{s}) \wedge z \neq x \wedge z \neq y \wedge \rightarrow [clear(z,s) \leftrightarrow clear(z,\underline{do(pickup(x,y),s)})]$
  - i.e. if clear(z) was true before the action, it will still be true after, and vice versa, for any blocks other than x and y

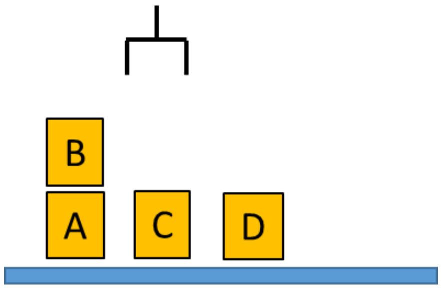
Frame Axioms

define Poss() for convenience; preconds say when it is Possible to do a given action

- Approach 1
  - for a specific action and *unaffected* predicate, if preconds hold, then if predicate was True before, it will be True after, and vice versa
  - picking up a block does not affect whether any other block is clear
  - ∀s,x,y,z on(x,y,s)^clear(x,s)^gripperEmpty(s)^z≠x^z≠y→
    [clear(z,s) ↔ clear(z,do(pickup(x,y),s))]
  - picking up a block does not affect whether the light is on in any room
  - ∀s,x,y,z on(x,y,s)^clear(x,s)^gripperEmpty(s)^room(z)→
    [lightOnIn(z,s) ↔ lightOnIn(z,do(pickup(x,y),s))]
  - but you would have to do this for almost all |Actions X Predicates| not scalable
- Approach 2 - the light would stay on for any action *except* turnOff
  - ∀s,x,y on(x,y,s)^clear(x,s)^gripperEmpty(s)→ Poss(pickup(x,y),s)
  - ∀s,a,z Poss(a,s)^a≠turnOffLight(z)^lightOnIn(z,s)→lightOnIn(z,do(a,s)))
- Approach 3: for each pred in succ state, list the ways it could be T
  - ∀s,x,y lightOnIn(z,do(a,s))↔[Poss(a,s) ^ (lightOnIn(z,s)^a≠turnOffLight(z)
    v a=turnOnLight(z))]

either it was on before and not affected by action, or we turned it on

Planning via Inference



- one could use Precond and Effects and Frame axioms to infer plans (sequences) of actions that entail the goal, like proving "∃s on(A,B,s)^on(B,C,s)" using resolution refutation or natural deduction
  - when proof succeeds, look at substitution for s in unifier: {s/do(puton(A,B),do(pickup(A,table),do(puton(B,C),do(pickup(B,A),Sinit))))}
- however, this is cumbersome and hard to control
  - inference might take many, many steps
- the goal is to develop Planning Algorithms that are more efficient at searching the space of sequences of actions

PDDL - Planning Domain Description Language

- for describing operators/actions
  - pre-conditions:
    - list of literals that must be satisfied to execute action
  - effects:
    - add-list: list of positive literals that will become true
    - delete-list: list of negative literals that will become false

$$
\begin{array}{l}
Init(At(C_1,\ SFO) \land At(C_2,\ JFK) \land At(P_1,\ SFO) \land At(P_2,\ JFK) \\
\quad \land\ Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
\quad \land\ Airport(JFK) \land Airport(SFO)) \\
Goal(At(C_1,\ JFK) \land At(C_2,\ SFO)) \\
Action(Load(c,\ p,\ a), \\
\quad \text{PRECOND: } At(c,\ a) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
\quad \text{EFFECT: } \neg At(c,\ a) \land In(c,\ p)) \\
Action(Unload(c,\ p,\ a), \\
\quad \text{PRECOND: } In(c,\ p) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
\quad \text{EFFECT: } At(c,\ a) \land \neg In(c,\ p)) \\
Action(Fly(p,\ from,\ to), \\
\quad \text{PRECOND: } At(p,\ from) \land Plane(p) \land Airport(from) \land Airport(to) \\
\quad \text{EFFECT: } \neg At(p,\ from) \land At(p,\ to))
\end{array}
$$

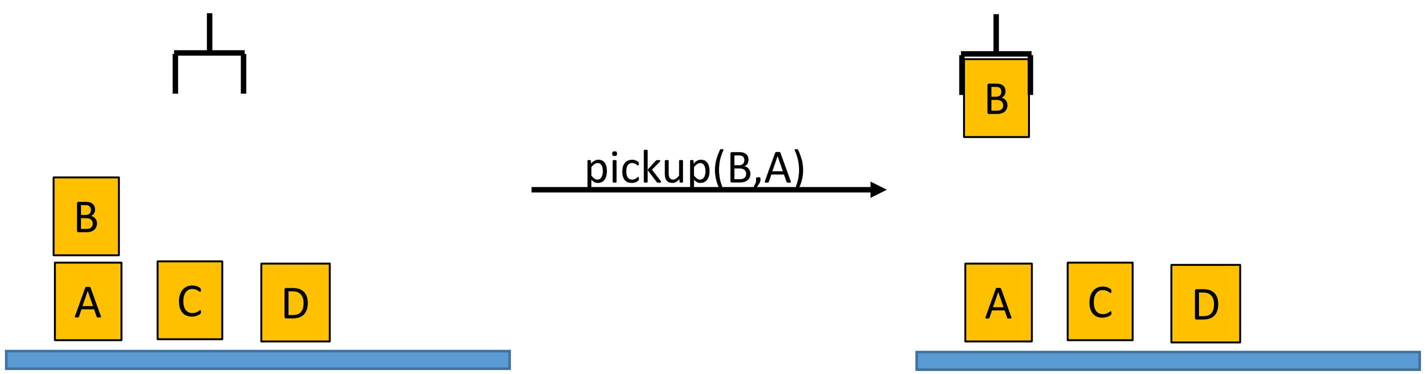
Figure 11.1 A PDDL description of an air cargo transportation planning problem.

$$
\begin{array}{l}
Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk)) \\
Goal(At(Spare, Axle)) \\
Action(Remove(obj, loc), \\
\quad \text{PRECOND: } At(obj, loc) \\
\quad \text{EFFECT: } \neg At(obj, loc) \land At(obj, Ground)) \\
Action(PutOn(t,\ Axle), \\
\quad \text{PRECOND: } Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \land \neg At(Spare, Axle) \\
\quad \text{EFFECT: } \neg At(t, Ground) \land At(t, Axle)) \\
Action(LeaveOvernight, \\
\quad \text{PRECOND: } \\
\quad \text{EFFECT: } \neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk) \\
\qquad\qquad \land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat,\ Trunk))
\end{array}
$$

Figure 11.2 The simple spare tire problem.

Example of PDDL operators from Blocksworld

- pickup(x,y):
  - pre-conds: on(x,y),clear(x),gripperEmpty()
  - effects: holding(x),clear(y),¬clear(x),¬on(x,y),¬ gripperEmpty()
- puton(x,y):
  - pre-conds: holding(x),clear(y)
  - effects: on(x,y),clear(x),gripperEmpty(),¬holding(x), ¬clear(y),



note: for simplicity, assume the table is always clear

pre-conds: on(B,A), clear(B), gripperEmpty()

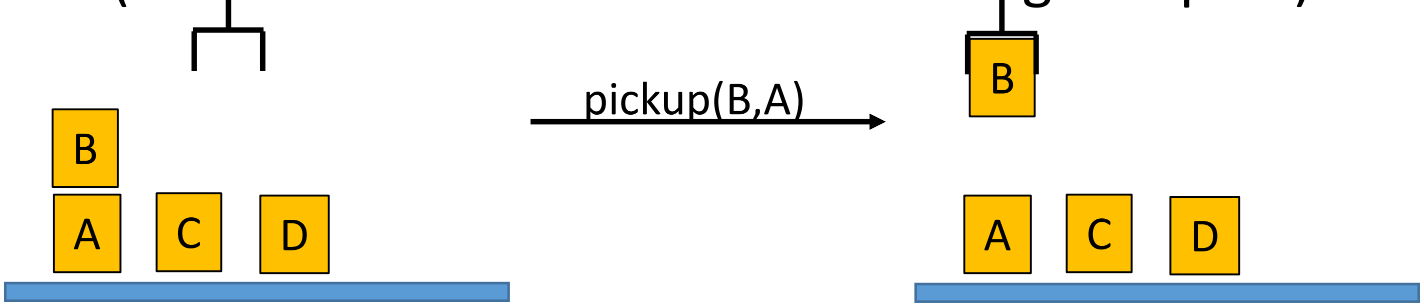
Effects: holding(B), clear(A), ¬on(B,A), ¬gripperEmpty()

- State Progression
  - given a set of literals describing a state, *compute* the description of the successor state for a given action using the state progression function:

"Progress()" qua verb not noun

$$\text{Progress(State,Op)} = \text{State} \setminus \text{Del(Op)} \cup \text{Add(Op)}$$

  - importantly, Progress(St,Op) solves the Frame Problem! (because all literals not mentioned get copied)



State s1:

on(B,A)	clear(B)
on(A,table)	clear(C)
on(C,table)	clear(D)
on(D,table)	GE()

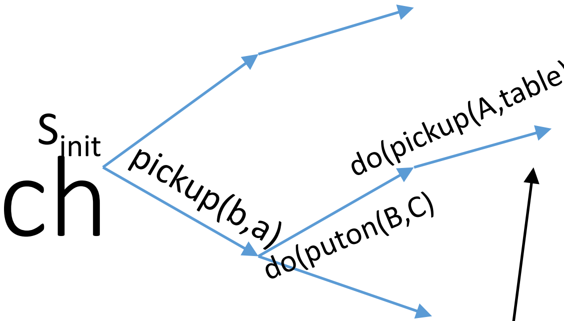
State s2=Progress(s1,pickup(B,A)):

	clear(A)
on(A,table)	clear(C)
on(C,table)	clear(D)
on(D,table)	holding(B)

red=delete-list

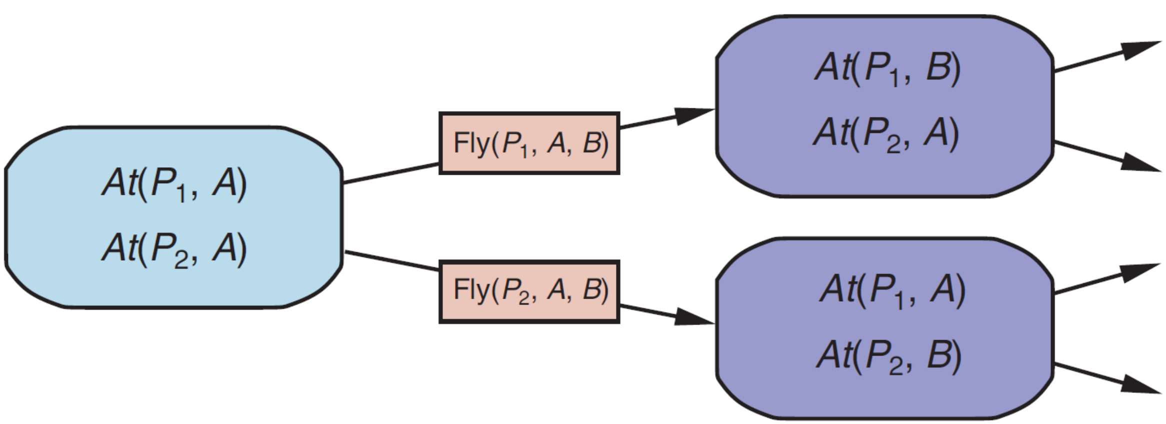
green=add-list

Forward State-Space Search



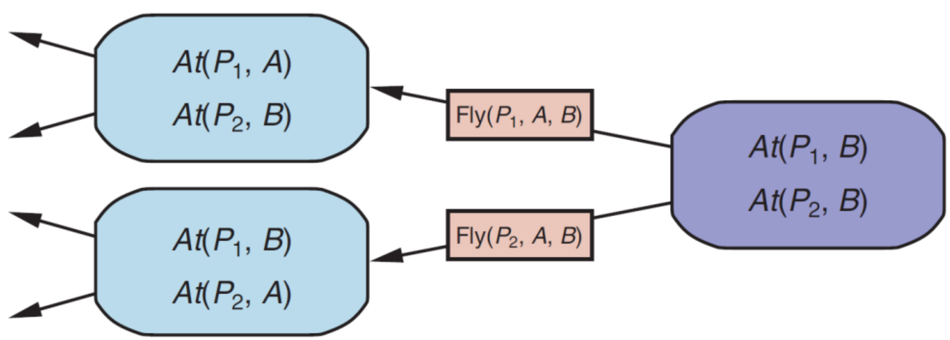
- The state progression function Progress() can be used to calculate what is true in every state descended from S_{init}
- could use this to do a *search* for a state in which the goal literals are true
  - use BFS? A*? what would a good heuristic be?

state contains on(A,B), on(B,C), based on Progress()



Goal Regression

- more efficient than forward State-Space Search
- Principle of Means-Ends Analysis (Newell&Simon)
  - identify a *difference* between the current and goal state, and find an operator that achieves that predicate as an effect
- more efficient than FSSS because it is goal-directed
  - form plan by working backwards from goal(s)
    - reduce goals to sub-goals
  - analogous to Back-chaining inference (recursive)



Goal Regression

from
Weld (1994)

Means-Ends Analysis: select action that is relevant

Algorithm: REGWS(init-state, cur-goals, Λ, path)

1. If init-state satisifes each conjunct in cur-goals,
2. Then return path,
3. Else do:
   (a) Let Act = **choose** from Λ an action whose effect matches at least one conjunct in cur-goals.
   (b) Let G = the result of regressing cur-goals through Act.
   (c) If no choice for Act was possible or G is undefined, or G $\supset$ cur-goals,
   (d) Then return failure,
   (e) Else return REGWS(init-state, G, Λ, Concatenate(Act, path)).

(consistency check, see next slide)

Regress(Goals,Op) = Goals \ Add(Op) ∪ Precond(Op)

- "weakest preimage": what is the minimal set of conditions which would allow op to be executed as last step and achieve Goals?

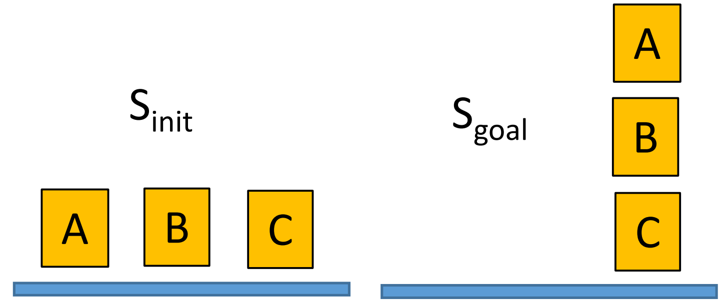
Goal Regression

consistency check:

If the result of regressing cur-goals through Act is to make G undefined then any plan that adds Act to this point in the path will fail. What might make G undefined? Recall that regression returns the weakest preconditions that must be true *before* Act is executed in order to make cur-goals true *after* execution. But what if one of Act's effects directly conflicts with cur-goals? That would make the weakest precondition undefined because *no matter* what was true before Act, execution would ruin things. A good example of this results when one tries to regress `((on A B) (on B C))` through `Move-A-from-B-to-Table`. Since this action negates `(on A B)`, the weakest preconditions are undefined.

(this can cause backtracking, as we wil see...)

- Example of Goal Regr
  - goal: on(a,b),on(b,c)
  - In each step, underline the selected subgoal to be achieved; becomes an effect of the action underneath that is selected to achieve it.
  - can be read-off plan backwards:
    1. *pickup(b,table)*
    2. *puton(b,c)*
    3. *pickup(a,table)*
    4. *puton(a,b)*



on(a,b),on(b,c) = S_{goal}

↑*puton(a,b)*

holding(a),clear(b),on(b,c)

↑ *pickup(a,table)*

on(a,table),clear(b),clear(a),GE,on(b,c)

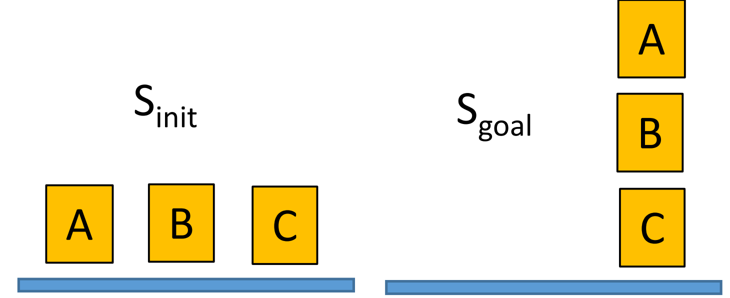
↑ *puton(b,c)*

on(a,table),holding(b),clear(c),clear(a)

↑ *pickup(b,table)*

on(a,table),on(b,table),clear(b),clear(c),clear(a)$\subseteq S_{init}$

- **Goal-Regression can involve Back-tracking**
  - choice-points depend on choices of which subgoal to achieve, and which operator to use
  - for example, if we chose on(b,c) first, the Goal-Regression would have failed, because there is not plan that ends in putting b on c



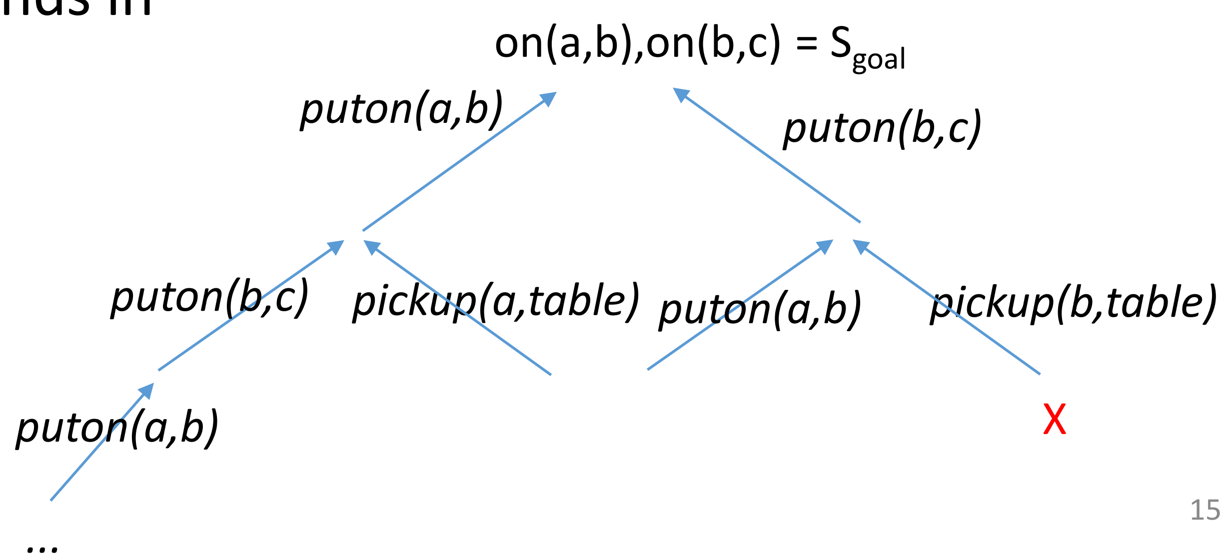
on(a,b),<u>on(b,c)</u> = S_{goal}

↑*puton(b,c)*

on(a,b),<u>holding(b)</u>,clear(c)

↑ *pickup(b,table)*

inconsistent preimage, so would have to back-track

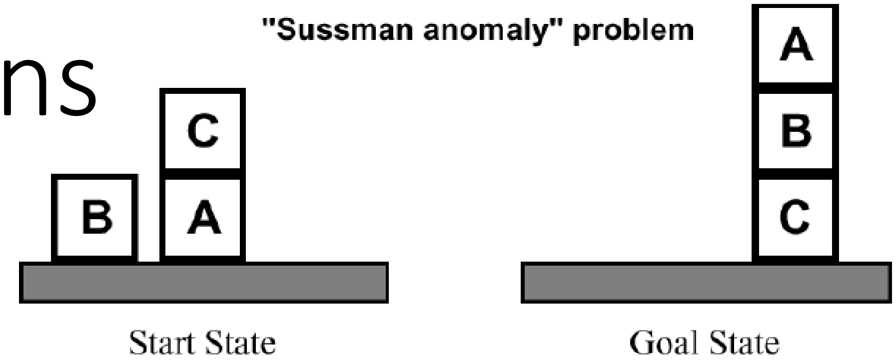


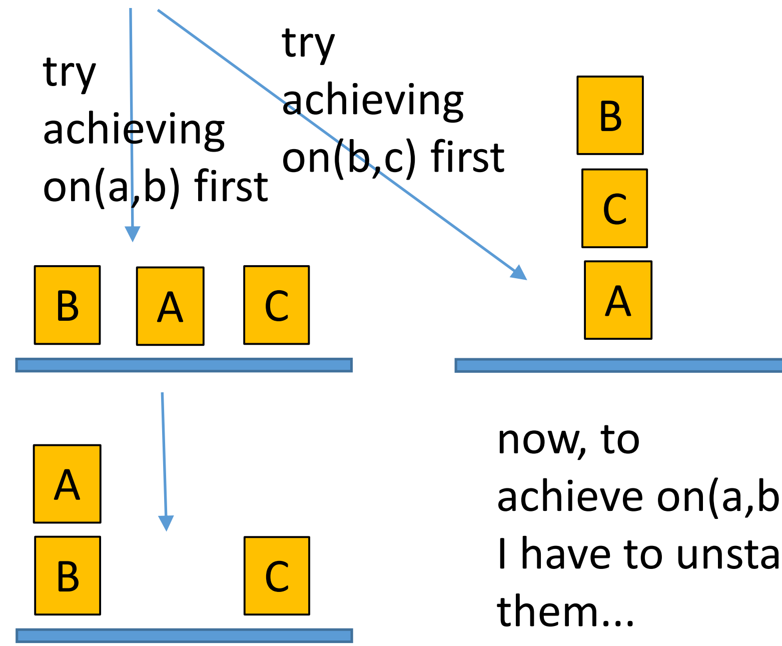
Subgoal Interactions

- when achieving one subgoal undoes the achievement of another
- Sussman Anomaly
  - goal: on(a,b), on(b,c)
- The lesson is that we need *non-linear planners* that *interleave actions*, rather than solving one subgoal at a time

solution: pickup(c,a)
puton(c,table)
pickup(b,table)
puton(b,c)
pickup(a,table)
puton(a,b)

blue is for actions for achieving on(a,b); red is for actions for achieving on(b,c)





now, to achieve on(a,b), I have to unstack them...

now, to achieve on(b,c), I have to unstack them...

Other Planners

- SatPlan - translate into a Boolean Satifiability Problem
- graph-based planners (POP, GraphPlan)
- abstraction planners/hierarchical planners (ABSTRIPS)
- Ordered Binary Decision Diagrams
- handling uncertainty in planners
- schedulers
- complexity of planning is NP-hard or worse (depending on expressiveness of the operator language)

SatPlan (read Sec 7.7 in textbook)

- translate precond/effect/frame axioms into propositional logic
  - make "ground versions" of sentences, one for each time step (for all combinations of objects and timesteps)
  - **propositionalization** (make ground predicates into prop syms, e.g. "clear(A,t1)" -> "clear_A_t1"
- add axioms for preconds and effects of each action <u>in each timestep</u>, like PickupAB1,PickupBA1, PickupAC1...PickupAB2...
  - PickupAB1→(ClearA1 ^ OnAB1 ^ HoldingA2 ^ ClearB2)
  - PickupAB2→(ClearA2 ^ OnAB2 ^ HoldingA3 ^ ClearB3)
- sentences: {action axioms} U {init_state at t0) U {goals at tN)
  - must anticipate the number of steps N
  - {action axioms} U {onAB0,clearA0,gripperEmpty0) U {onBA4)
- <u>solve as Boolean Satisfiability (e.g. using DPLL)</u>
- the "plan" is given by which action props are True in the model
  - e.g. pickupAB1, putonAtable2, pickupB3,putonBA4

SatPlan

$$At(P_1, JFK)^1 \;\Leftrightarrow\; (At(P_1, JFK)^0 \wedge \neg(Fly(P_1, JFK, SFO)^0 \wedge At(P_1, JFK)^0)) \\ \vee (Fly(P_1, SFO, JFK)^0 \wedge At(P_1, SFO)^0)\;. \qquad (11.1)$$

"if P1 is at JFK at t=1, then either

a) it was flown there, or

b) it was already there and not flown elsewhere"

alternatively:

Precond Axiom: $Fly(P1,JFK,SFO)^0 \rightarrow At(P1,JFK)^0$ // what must be true at time t to do action?

Effects Axioms: $Fly(P1,JFK,SFO)^0 \rightarrow At(P1,SFO)^1$ // what would be true at time t+1?

...and copies for all time steps, and every package, and every pair of cities...

$Fly(P1,JFK,SFO)^1 \rightarrow At(P1,SFO)^2$; $Fly(P1,JFK,SFO)^2 \rightarrow At(P1,SFO)^3$; $Fly(P1,JFK,SFO)^3 \rightarrow At(P1,SFO)^4$

or *t* if you think it will take *t* steps

planes swap places. Now, suppose the KB is

$$\textit{initial state} \wedge \textit{successor-state axioms} \wedge \textit{goal}^1\;, \qquad (11.2)$$

which asserts that the goal is true at time $T = 1$. You can check that the assignment in which

$$Fly(P_1, SFO, JFK)^0 \text{ and } Fly(P_2, JFK, SFO)^0$$

are true and all other action symbols are false is a model of the KB. So far, so good. Are

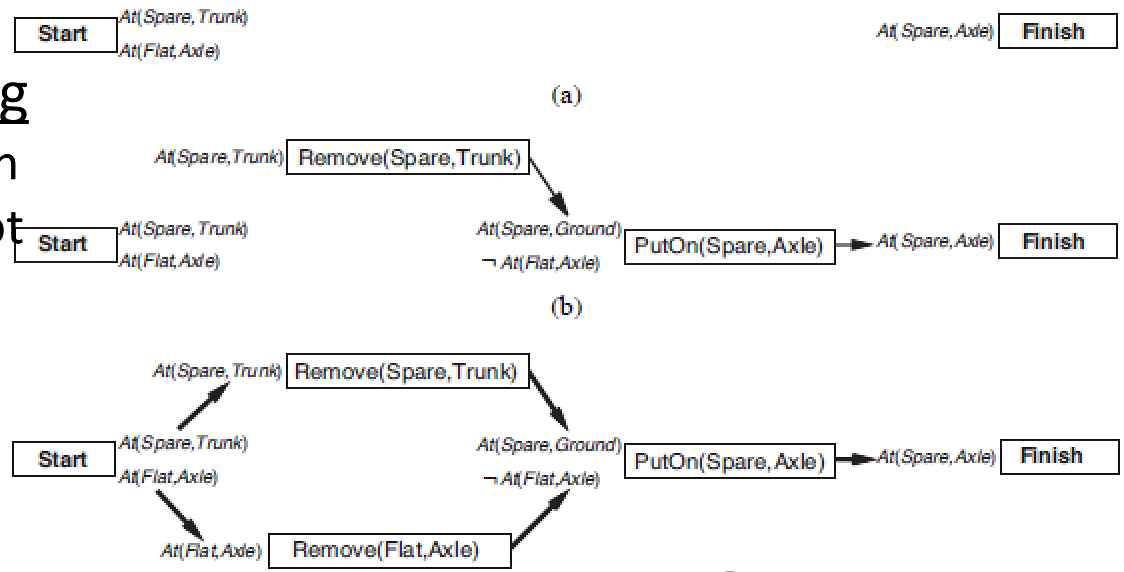
Mutual Exclusion axioms for actions

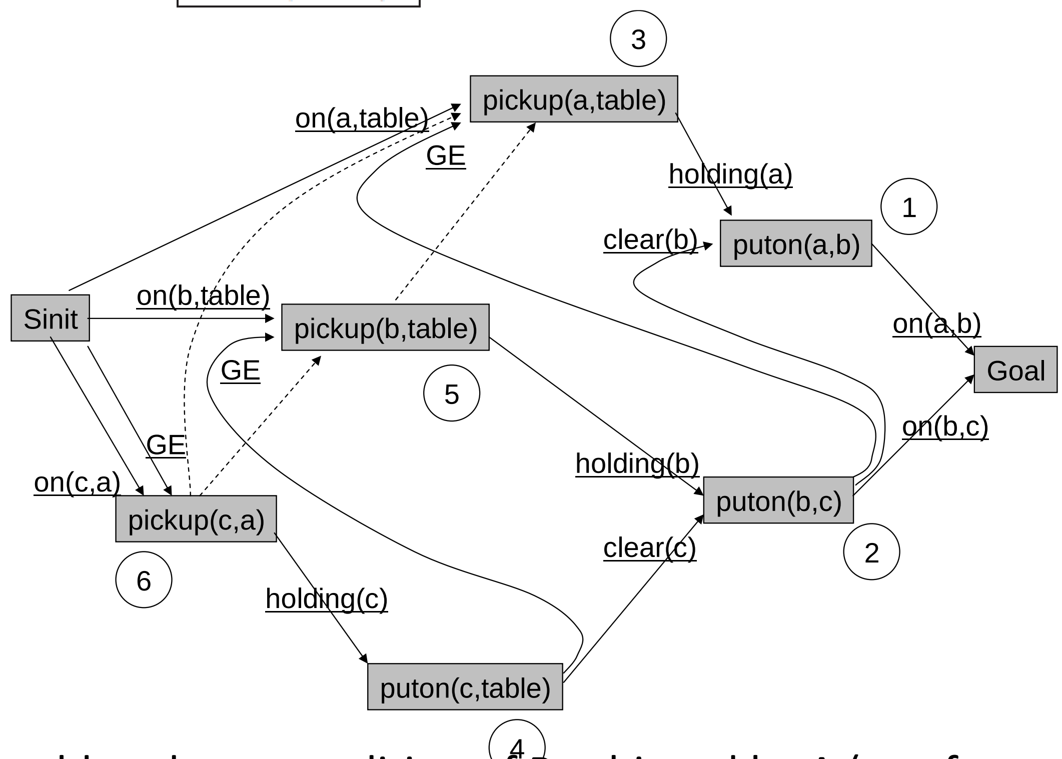
- at most on action proposition can be true in each timestep
  - pickupAB1->¬pickupAC1^¬pickupBA1^¬pickupBC1^...
  - pickupAC1->¬pickupAB1^¬pickupBA1^¬pickupBC1^...
  - pickupAB2->¬pickupAC2^¬pickupBA2^¬pickupBC2^...

pickupAB1	pickupAB2	pickupAB3
pickupAC1	pickupAC2	pickupAC3
pickupBA1	pickupBA2	pickupBA3
pickupBC1	pickupBC2	pickupBC3
pickupCA1	pickupCA2	pickupBtable3
pickupCB1	pickupCB2	pickupCA3
putonAB1	putonAB2	pickupCB3
putonAC1	putonAC2	putonAB3
putonBA1	putonBA2	putonAC3
...	putonBC2	putonBA3
	...	...

POP: Partial-Order Planning

- "non-linear planning"; search the space of *plan-graphs* (not just action sequences)
- principle of "least commitment" - don't force ordering of actions till necessary
- make a *graph* with actions as nodes
- add edges where effects of 1 action achieve preconditions of another action
- detect *conflicts**, and resolve by adding edges to force which action comes first
- in the end, extract the plan as a *linearization* (topological sort) of the graph

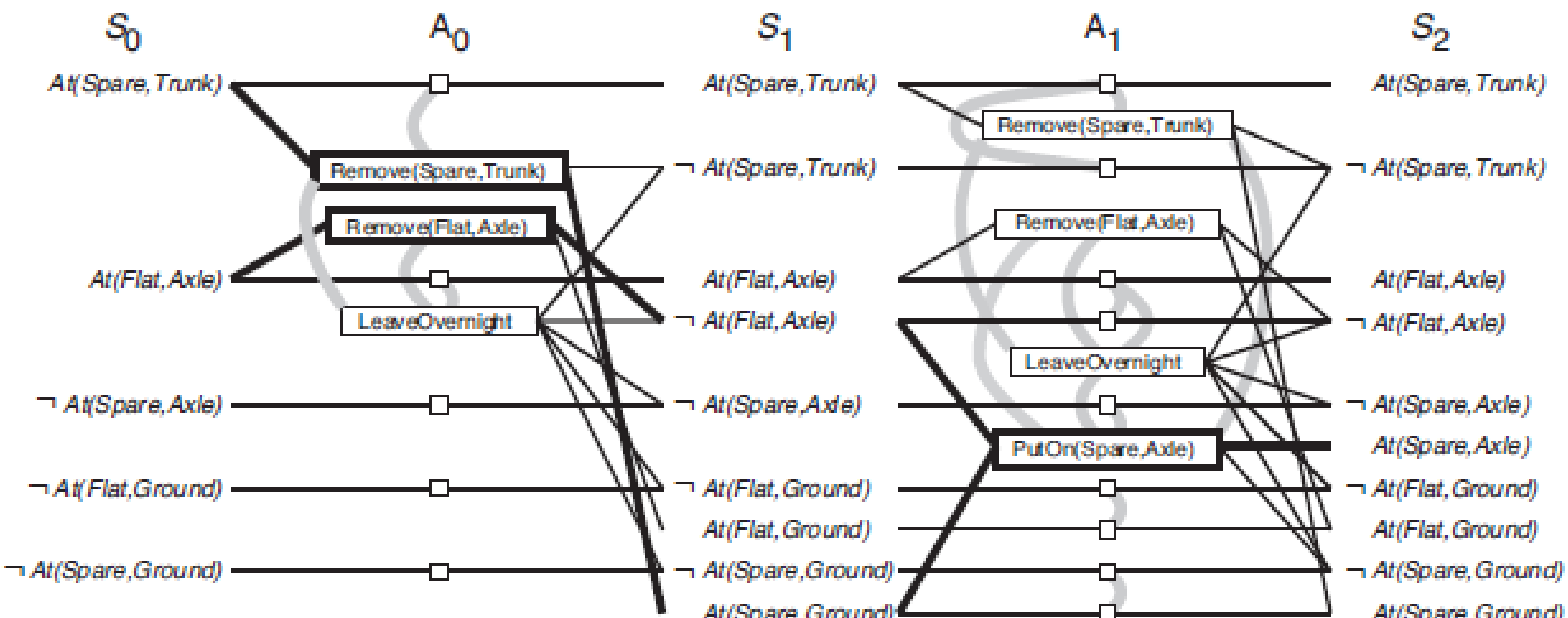




**conflicts* are where effect of action C could undo precondition of B achieved by A (e.g. for edgeA->B); add edge to force C to come before A or after B

GraphPlan (Blum and Furst)

- an even more complex graph-based planning algorithm that achieves combinations of subgoals in "layers"



Complexity of Planning

- complexity: planning is NP-hard
  - proved in (David Chapman, 1987, AI journal)
  - depends on expressiveness of pre- and post-conditions, e.g. disjunctive? conditional effects?...
  - reduction from...Sat (Boolean Satisfiability)
- fight complexity by simplifying operators by removing smaller details (pre-conditions that would be easy to fill-in and achieve later) ("abstraction planning")
- another approach: decompose the search space by doing "hierarchical planning"

Abstraction Planners

- focus on finding a correct sequence for the "big steps"
- try dropping/ignoring pre-conditions that are easily achieved (later)
  - similar to defining "relaxed operators" for search, like sliding tiles over each other in the tile puzzles
  - how do you automatically infer which preconditions are less relevant?
- ABSTRIPS (Craig Knoblock)
- also try state abstraction
  - drop variable or dimenstions of the state to reduce the size of the state space

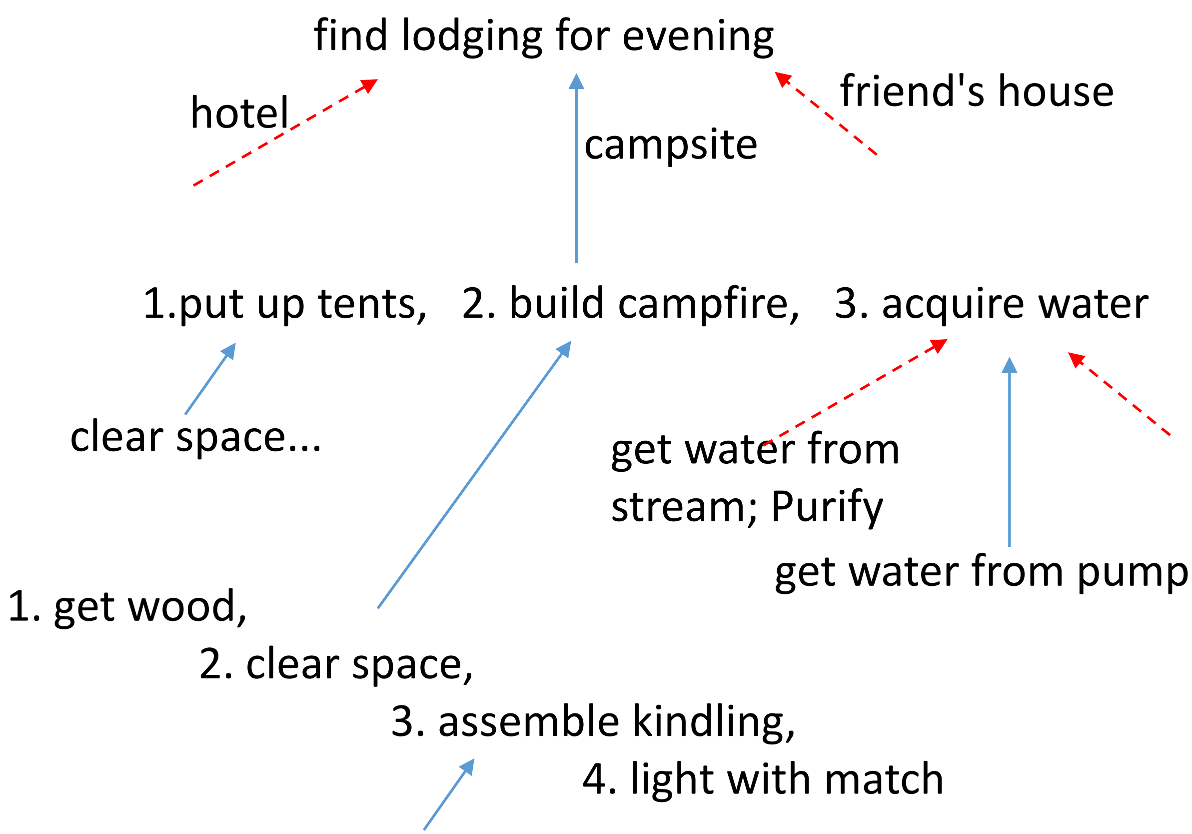
Hierarchical Planners

- Hierarchical Task Networks (HTNs)
  - reduces complexity of planning
- uses "plan libraries" consisting of *scripts* for different ways to achieve high-level activities *and* low-level activities
- HTNs work by elaboration: choose high-level actions, then fill in actions to achieve lower-level tasks
- challenges:
  - a) hard to accurately represent preconds and effects of high-level tasks (before knowing low-level actions)
  - b) does not allow for interactions between tasks (especially positive: sharing/overlap of steps)

<u>Plan Library:</u>

T0: find lodging for evening (hotel, campsite, friend's house...)
T1: setting up camp: put up tents, build campfire, acquire water...
T2: building a campfire: get wood, clear space, assemble kindling, light with match...
T3a: acquire water: get bucket, get water from stream
T3b: acquire water: get jug from backpack
T3c: acquire water: go to water pump
T4: treating blisters...
T5: cooking fish
T6: cooking canned chili...
...

Hierarchical Plan (HTN) for Camping



<u>Plan Library:</u>

T0: find lodging for evening (hotel, campsite, friend's house...)

T1: setting up camp: put up tents, build campfire, acquire water...

T2: building a campfire: get wood, clear space, assemble kindling, light with match...

T3a: acquire water: get water from stream;Purify

T3b: acquire water: get jug from backpack

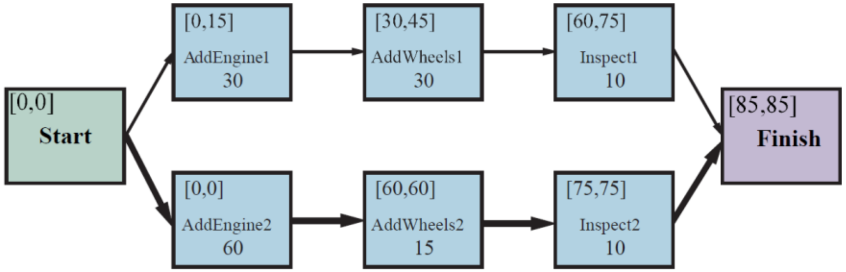
T3c: acquire water: get water from pump

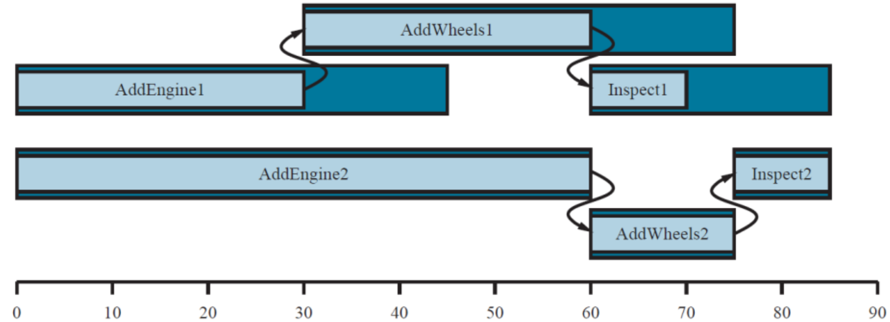
T4: treating blisters

...

Adaptive Planners

- plan monitoring and repair
- if something goes wrong (not as expected), do not want to re-plan from scratch (new initial state)
- can you "modify" the original plan, or "re-use" the search of the state space?
- online planning; contingent planning...





Scheduling

- what's the difference between planning and scheduling?
- both have actions with precedence constraints
- in planning we are usually satisfied with finding any sequence of actions that achieves the goal
- in scheduling
  - actions have duration
  - actions can overlap (parallel processes)
  - actions can have resource/mutual exclusion constraints
  - objective is usually to find a sequence of actions with *minimum makespan* (e.g. Critical Path Method, CPM)