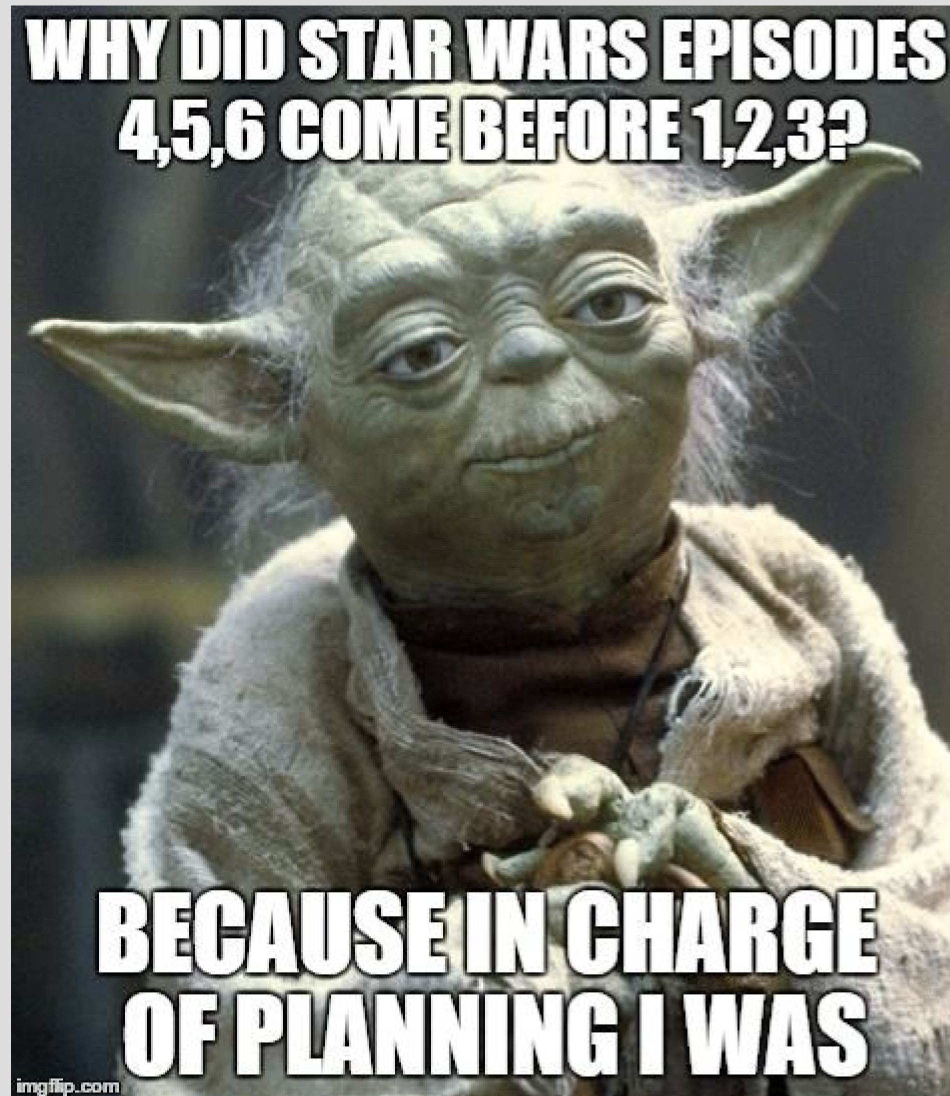
# Planning (Ch. 10)



# Forward search

$$\text{Action}(\ GoTo(x, y, z),$$
$$\text{Precondition: } At(x, y) \wedge Mobile(x),$$
$$\text{Effect: } \neg At(x, y) \wedge At(x, z))$$

Last time...

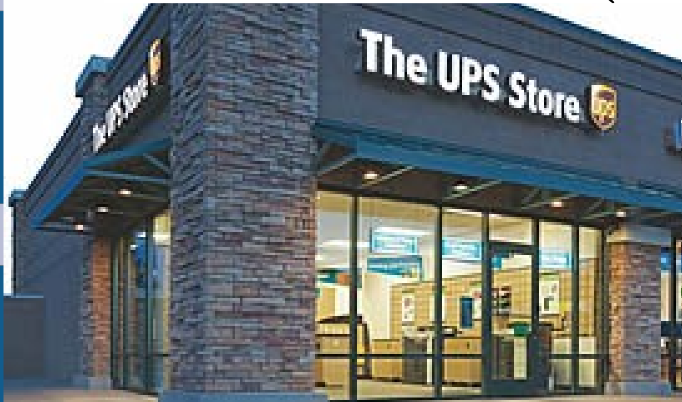
Initial: At(Truck, UPSD) ^ Package(UPSD, P1) ^ Package(UPSD, P2) ^ Mobile(Truck)

Goal: Package(H1, P1) ^ Package(H2, P2)

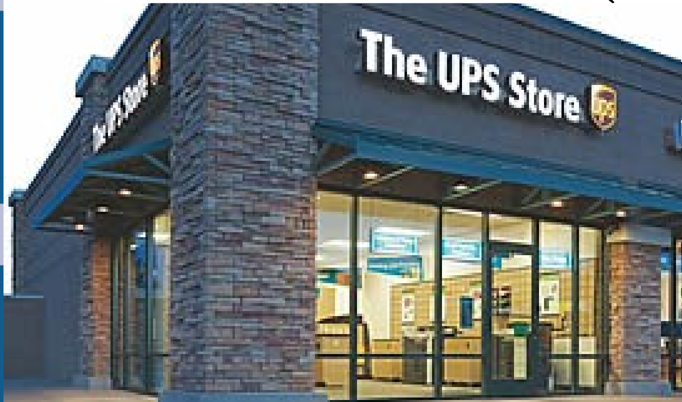
$$\text{Action}(\ Load(m, x, y),$$
$$\text{Precondition: } At(m, y) \wedge Package(y, x),$$
$$\text{Effect: } \neg Package(y, x) \wedge Package(m, x) \wedge At(m, y))$$

$$\text{Action}(\ Deliver(m, x, y),$$
$$\text{Precondition: } At(m, y) \wedge Package(m, x),$$
$$\text{Effect: } \neg Package(m, x) \wedge Package(y, x) \wedge At(m, y))$$

$$At(Truck, UPSD) \wedge Package(UPSD, P1)$$
$$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$$

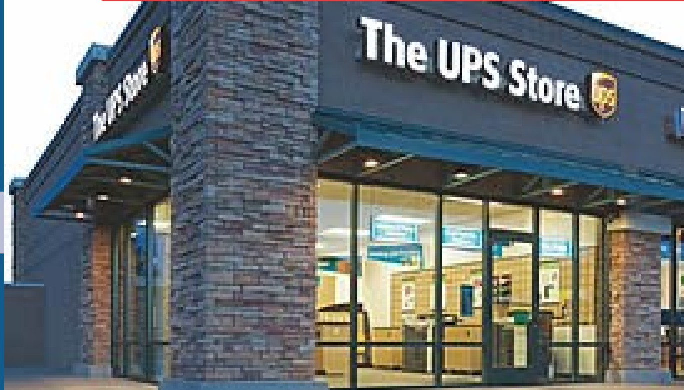


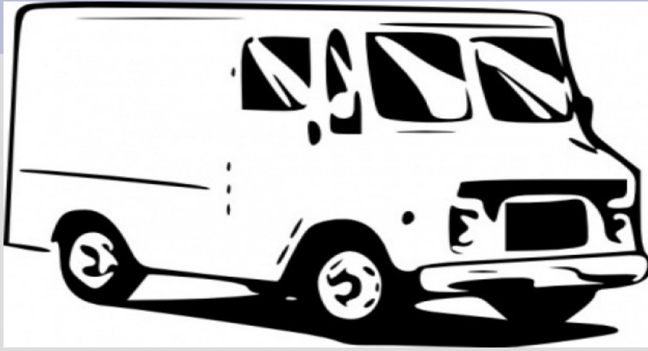
$$At(Truck, UPSD) \wedge Package(UPSD, P1)$$
$$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$$



Action( $Load(m, x, y)$,
Precondition: $At(m, y) \wedge Package(y, x)$,
Effect: $\neg Package(y, x) \wedge Package(m, x)$)

$At(Truck, UPSD) \wedge Package(UPSD, P1)$
$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$



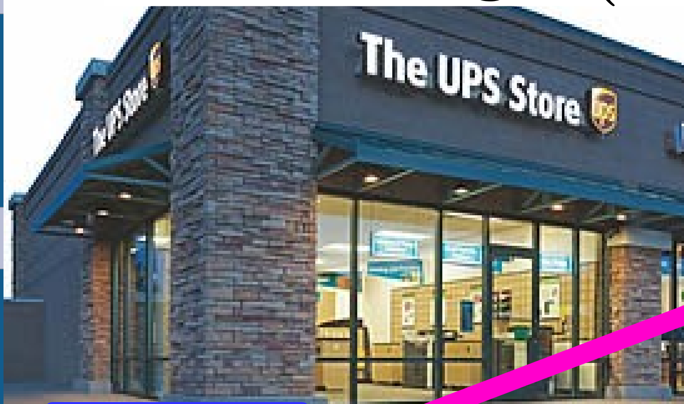
Find match
m/Truck
x/P1, y/UPSD

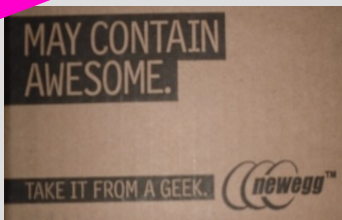
Action( $Load(m, x, y)$,
Precondition: $At(m, y) \wedge Package(y, x)$,
Effect: $\neg Package(y, x) \wedge Package(m, x)$)

$At(Truck, UPSD) \wedge$ ~~$Package(UPSD, P1)$~~

$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$

$\wedge Package(Truck, P1$





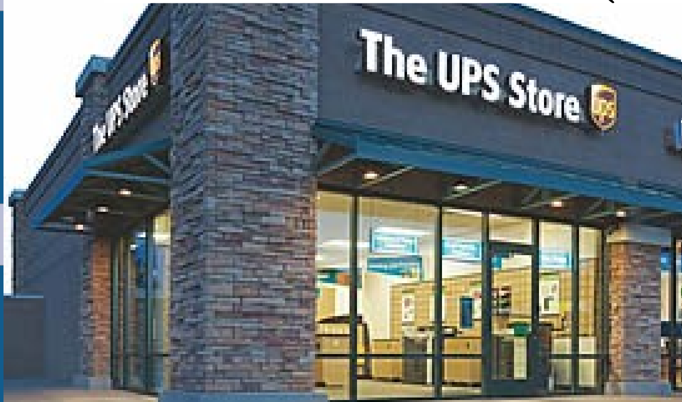

Apply effects

Action( $Load(Truck, P1, UPSD)$,

Precondition: $At(Truck, UPSD) \wedge Package(UPSD, P1)$,

Effect: $\neg Package(UPSD, P1) \wedge Package(Truck, P1)$)

$$At(Truck, UPSD) \wedge Package(Truck, P1)$$
$$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$$

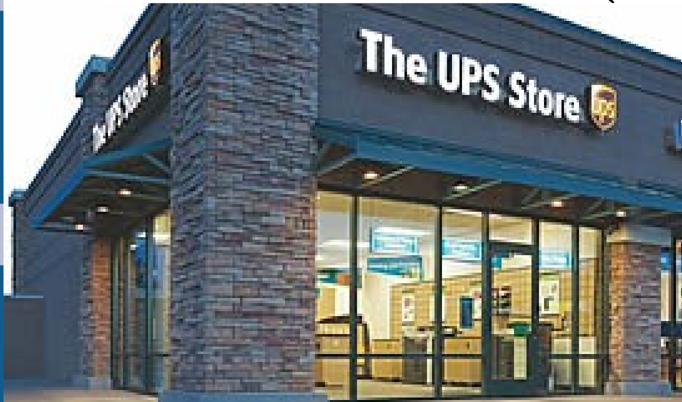


Action( $Load(Truck, P2, UPSD)$,

Precondition: $At(Truck, UPSD) \wedge Package(UPSD, P2)$,

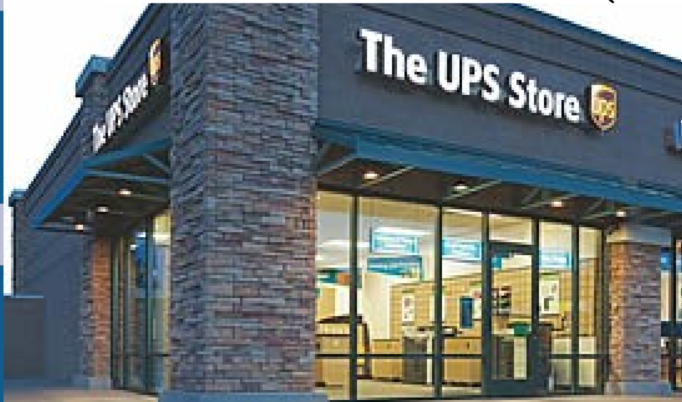
Effect: $\neg Package(UPSD, P2) \wedge Package(Truck, P1)$)

$At(Truck, UPSD) \wedge Package(Truck, P1)$
$\wedge Package(Truck, P2) \wedge Mobile(Truck)$



Action( $GoTo(Truck, USPD, H1)$,
Precondition: $At(Truck, USPD) \wedge Mobile(Truck)$,
Effect: $\neg At(Truck, USPD) \wedge At(Truck, H1)$)

$$At(Truck, H1) \wedge Package(Truck, P1) \wedge Package(Truck, P2) \wedge Mobile(Truck)$$

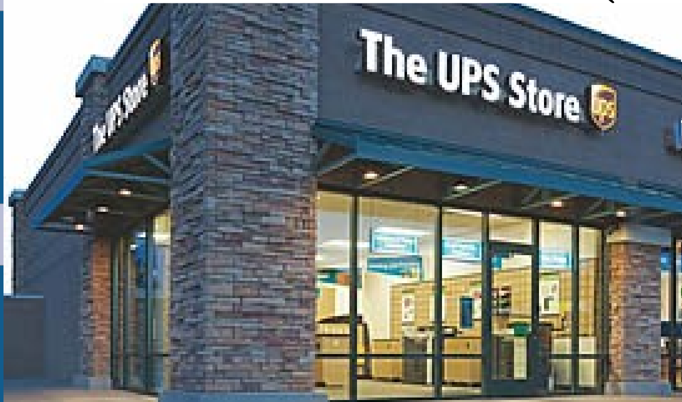



Action( $Deliver(Truck, P1, H1)$,

Precondition: $At(Truck, H1) \wedge Package(Truck, P1)$,

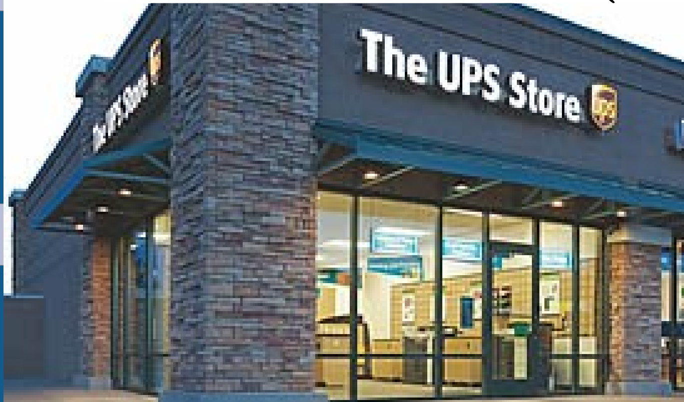
Effect: $\neg Package(Truck, P1) \wedge Package(H1, P1)$)

$$At(Truck, H1) \wedge Package(H1, P1)$$
$$\wedge Package(Truck, P2) \wedge Mobile(Truck)$$




Action( $GoTo(Truck, H1, H2)$,
Precondition: $At(Truck, H1) \wedge Mobile(Truck)$,
Effect: $\neg At(Truck, H1) \wedge At(Truck, H2)$)

$$At(Truck, H2) \wedge Package(H1, P1)$$
$$\wedge Package(Truck, P2) \wedge Mobile(Truck)$$

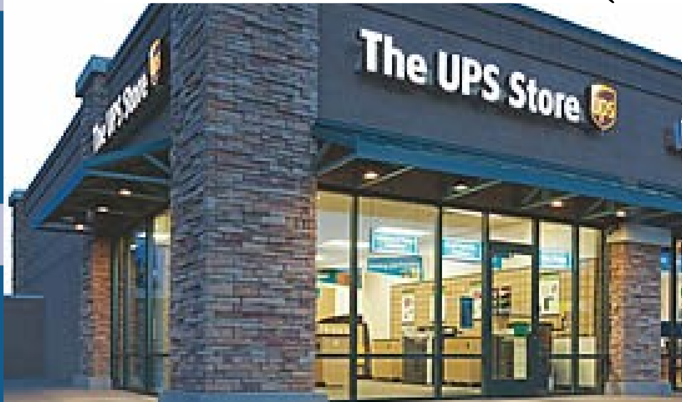



Action( $Deliver(Truck, P2, H2)$,

Precondition: $At(Truck, H2) \wedge Package(Truck, P2)$,

Effect: $\neg Package(Truck, P2) \wedge Package(H2, P2)$)

$At(Truck, H2) \wedge Package(H1, P1)$
$\wedge Package(H2, P2) \wedge Mobile(Truck)$



# Forward search

While the solution might seem obvious to us, the search space is (surprisingly) quite large

The brute force way (forward search) simply looks at all valid actions from the current state

We can then search it in using BFS (or iterative deepening) to find fewest action cost goal

# Forward search

# Forward search

Actions: 3 (2 unique ones, as Deliver = Load)
Objects: 6 (Truck, USPD, H1, H2, P1, P2)
Min moves to goal: 6 (L, L, G, D, G, D)

Despite this problem being simplistic,
the branching factor is about 4 to 5
(even with removing redundant actions)

This means we could search around 10,000
states before we found the goal

# Forward search

This search is actually much more than the number of states due to redundant paths

Package() can be: UPSD, Truck, H1, H2
At() can be: USPD, Truck, H1, H2, P1, P2

There are 2 packages for Package()
There is 1 truck for Truck()

So total states = 4^2 * 6 = 96

# Backward search

Backward search is also similar to FO's backward search

Start at goal and do actions in reverse (swap effect and precondition), except substitute:

$$\text{Action}(\ GoTo(x,y),\ \text{Precondition: } At(x),\ \text{Effect: } \neg At(x) \wedge At(y))$$

$$\text{Action}^{-1}(\ GoTo(x,y),\ \text{Precondition: } \neg At(x) \wedge At(y),\ \text{Substitute: } At(x))$$

Example:
Goal = At(Home)
Initial = At(Class)
Unify: {x/Home,
y/Class} ... done

# Backward search

If applying the substitution is more difficult, you can convert by:

1. Apply effect to precondition
2. Negate effects, add original precondition

$$\text{Action}(\ Deliver(m,x,y),$$
$$\text{Precondition: } At(m,y) \land Package(m,x),$$
$$\text{Effect: } \neg Package(m,x) \land Package(y,x))$$

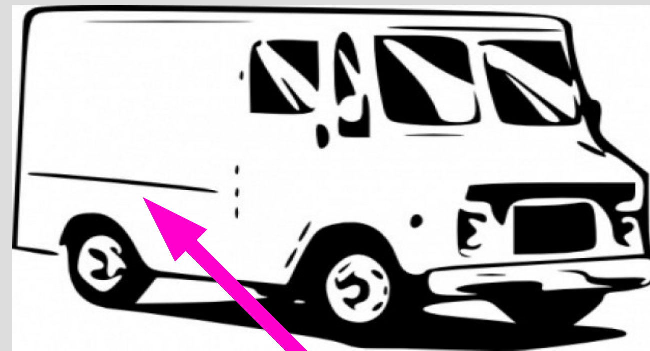
$$\text{Action}(\ Deliver(m,x,y)^{-1},$$
$$\text{Precondition: } At(m,y) \land Package(m,x) \land \neg Package(m,x) \land Package(y,x))$$
$$\text{Effect: } At(m,y) \land Package(m,x) \land Package(m,x) \land \neg Package(y,x))$$

Remove

Redundant

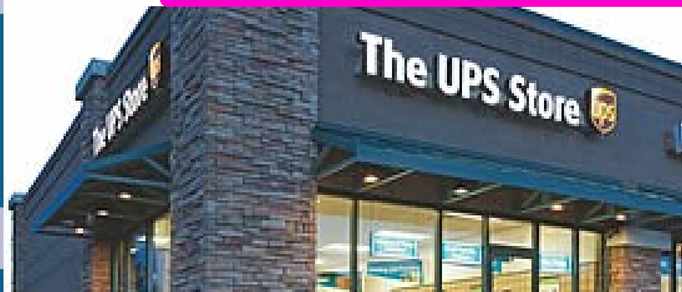
$At(Truck, H2) \wedge Package(H1, P1)$
$\wedge Package(H2, P2) \wedge Mobile(Truck)$

Action( $Deliver(m, x, y)^{-1}$,
Precondition: $At(m, y) \wedge Package(y, x)$)
Effect: $At(m, y) \wedge Package(m, x) \wedge \neg Package(y, x)$)



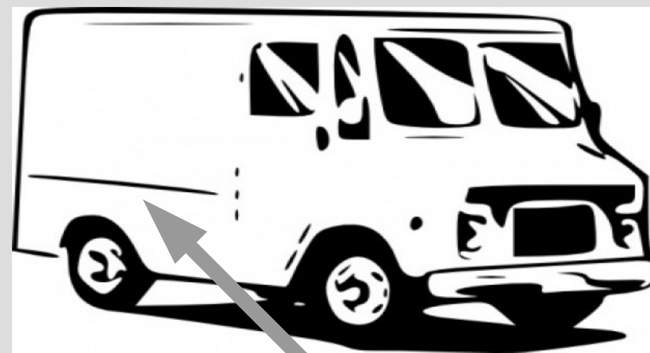
Unify:
m/Truck,
x/P2, y/H2

$$At(Truck, H2) \wedge Package(H1, P1)$$
$$\wedge \cancel{Package(H2, P2)} \wedge Mobile(Truck)$$
$$\wedge Package(Truck, P2)$$

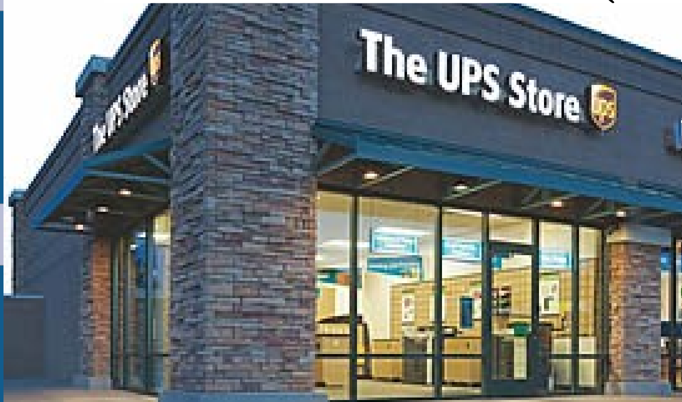



$$\text{Action}(\ Deliver(m, x, y)^{-1},$$
$$\text{Precondition: } At(m, y) \wedge Package(y, x))$$
$$\text{Effect: } At(m, y) \wedge Package(m, x) \wedge \neg Package(y, x))$$

No change



$At(Truck, H2) \wedge Package(H1, P1)$
$\wedge Package(Truck, P2) \wedge Mobile(Truck)$




Try to continue from here!

Action( $Load(m, x, y)$,
Precondition: $At(m, y) \wedge Package(y, x)$,
Effect: $\neg Package(y, x) \wedge Package(m, x) \wedge At(m, y)$)

Action( $Deliver(m, x, y)$,
Precondition: $At(m, y) \wedge Package(m, x)$,
Effect: $\neg Package(m, x) \wedge Package(y, x) \wedge At(m, y)$)

Action( $GoTo(x, y, z)$,
Precondition: $At(x, y) \wedge Mobile(x)$,
Effect: $\neg At(x, y) \wedge At(x, z)$)

# Heuristics for planning

Backwards search has a smaller branching factor in general, but it is hard to use heuristics

This is due to it looking at sets of states, and not a single state for the next action

For this reason, it is often better to apply a good heuristic to the dumb forward search

# Heuristics for planning

Reformulate our grilling problem as actions:

$$\text{Action}(\ MakeSandwitch(x),$$
$$\text{Precondition: } Meat(x) \land Make(Bread, x, Bread))$$
$$\text{Effect: } \neg Meat(x) \land Sandwitch(Bread))$$

If our goal is just “Sandwitch(Bread)”, backtracking search would try to solve:

$$Meat(x) \land Make(Bread, x, Bread))$$

... but since “x” is still a variable, this represents a set of states rather than one

# Heuristics for planning

In “search” we had no generalize-able heuristics as each problem could be different

Heuristics in planning are found the same way, we (1) relax the problem (2) solve it optimally

Two generic ways to always do this are:

1. Add more actions
2. Reduce number of states

# Heuristics: add actions

Multiple ways to add actions (to goal faster):

1. Ignore preconditions completely - also ignore any effects not related to goals

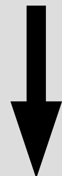
   This becomes set-covering problem, which is NP-hard but has P approximations

2. Ignore any deletions in effects (i.e. anything with $\neg$), also NP-hard but P approximation

# Ignore preconditions

By simply removing preconditions, we allow every action to happen at every state

$$\text{Action}(\ GoTo(x, y, z),$$
$$\text{Precondition: } At(x, y) \wedge Mobile(x),$$
$$\text{Effect: } \neg At(x, y) \wedge At(x, z))$$



$$\text{Action}(\ GoTo(x, y, z),$$
$$\text{Precondition: }$$
$$\text{Effect: } \neg At(x, y) \wedge At(x, z))$$

# Ignore preconditions

More importantly for the solution is how the Delivery action changes

The USPD can now just directly deliver to houses, so goal is: Deliver(USPD, P1, H1) and then Deliver(USPD, P2, H2)

$$\text{Action}(\ Deliver(m, x, y),$$
$$\text{Precondition:}$$
$$\text{Effect: } \neg Package(m, x) \land Package(y, x))$$

# Ignore negative effects

To use this heuristic, the goal cannot have negative functions/literals (i.e. $\neg At(Truck, H2)$)

This can always be rewritten to something else (for above $At(Truck, USPD) \vee At(Truck, H1) \vee At(Truck, Truck)$)

$$\text{Action}(\ GoTo(x, y, z),$$
$$\text{Precondition: } At(x, y) \wedge Mobile(x),$$
$$\text{Effect: } \neg At(x, y) \wedge At(x, z))$$

$$\text{Action}(\ GoTo(x, y, z),$$
$$\text{Precondition: } At(x, y) \wedge Mobile(x),$$
$$\text{Effect: } At(x, z))$$

# Ignore negative effects

For the UPS delivery example, it does not help us find a solution faster (min is 6 still)

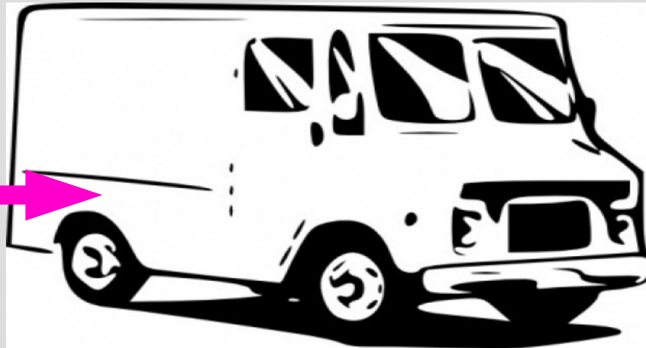
However, there are many more solutions as every action “copies” instead of “moves”

For example, a solution could be:
Move, Move, Load, Load, Deliver, Deliver

This is possible as truck exists at all 3 spots!

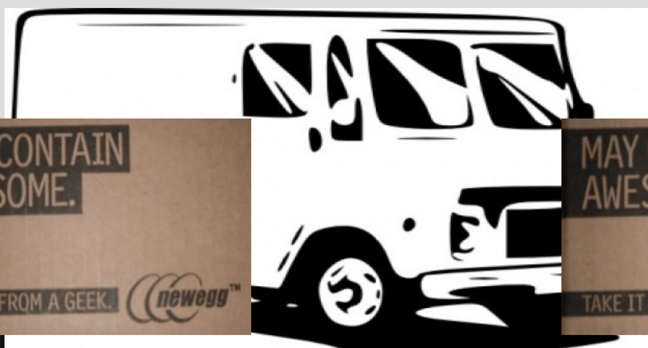
$$At(Truck, UPSD) \wedge At(Truck, H1) \wedge At(Truck, H2)$$
$$\wedge Package(UPSD, P1) \wedge Package(UPSD, P2) \wedge Mobile(Truck)$$

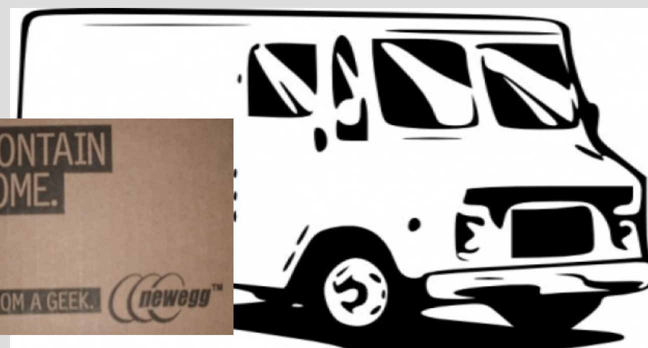
## After 2 moves... then load...



$At(Truck, UPSD) \wedge At(Truck, H1) \wedge At(Truck, H2) \wedge Package(Truck, P2)$
$\wedge Package(UPSD, P1) \wedge Package(UPSD, P2) \wedge Mobile(Truck)$

# After 2 moves... then load...





# Heuristics: group states

Group similar states together into “super states” and solve the problem within “super states” separately (divide & conquer)

A admissible but bad heuristic would be the maximum of all “super states” individual solutions (but this is often poor)

A possibly non-admissible would be the sum of all “super states” (need independence)

# Heuristics: group states

These “super states” can created in many ways

1. Delete relations/fluents (e.g. no more “At”)
2. Merge objects/literals (e.g. merge UPSD and Truck)

You then need to solve two problems:

1. Between the abstract “super states”
2. Within each “super state”

# Heuristics: group states

Consider if there were 3 houses, but only two needed packages

We could remove all “At”s for this third house, as we can easily abstract it away

In this case the “super state” solution is the actual solution as there is no need to add back in a third house

# Heuristics: group states

For example, if we were instead delivering 3 packages, 1 to H1 and 2 to H2...

We combine the two packages for H2 into a single “super package” with only one load and deliver (overall “super state” solution)

We then can simply see that each load/deliver corresponds to two individual loads/delivers (within super state solution)