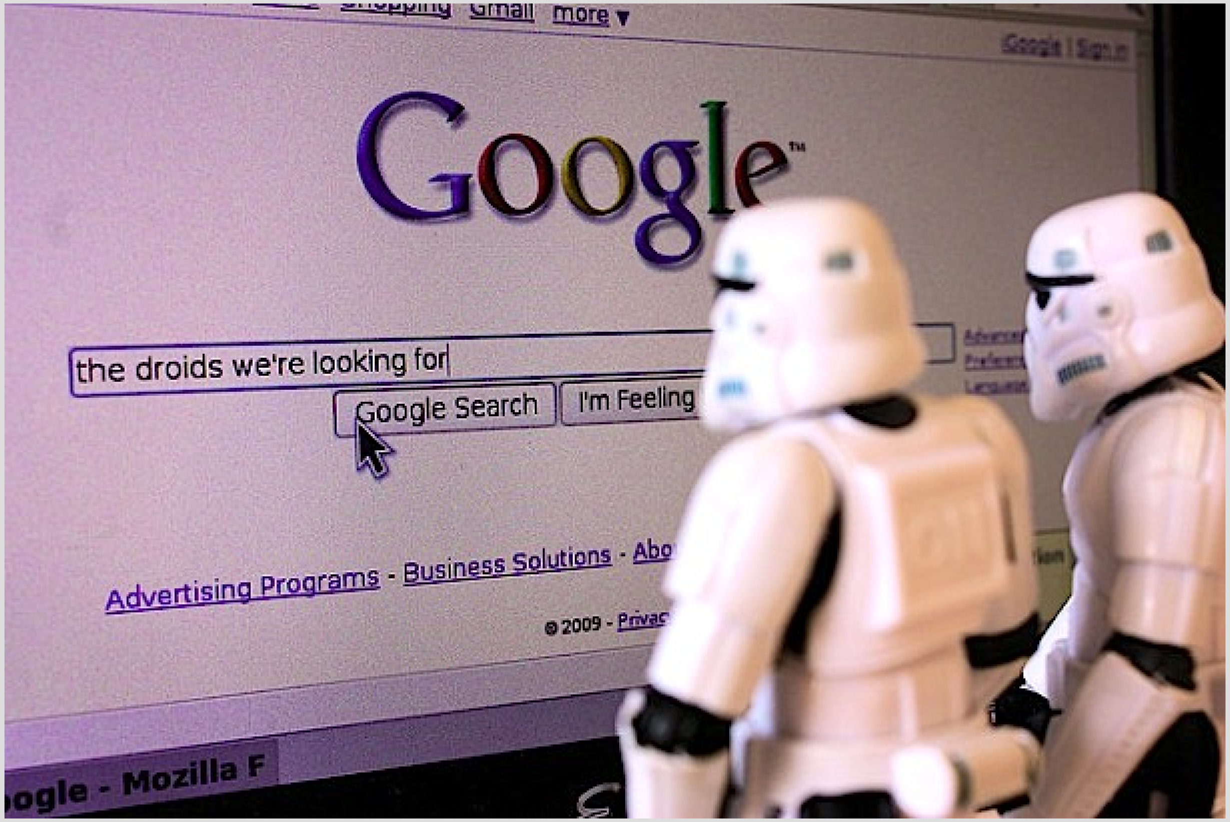
Planning (Ch. 10)


Google
the droids we're looking for
Google Search
I'm Feeling
Advertising Programs - Business Solutions - Abo
©2009 - Privac
oogle - Mozilla F

Planning

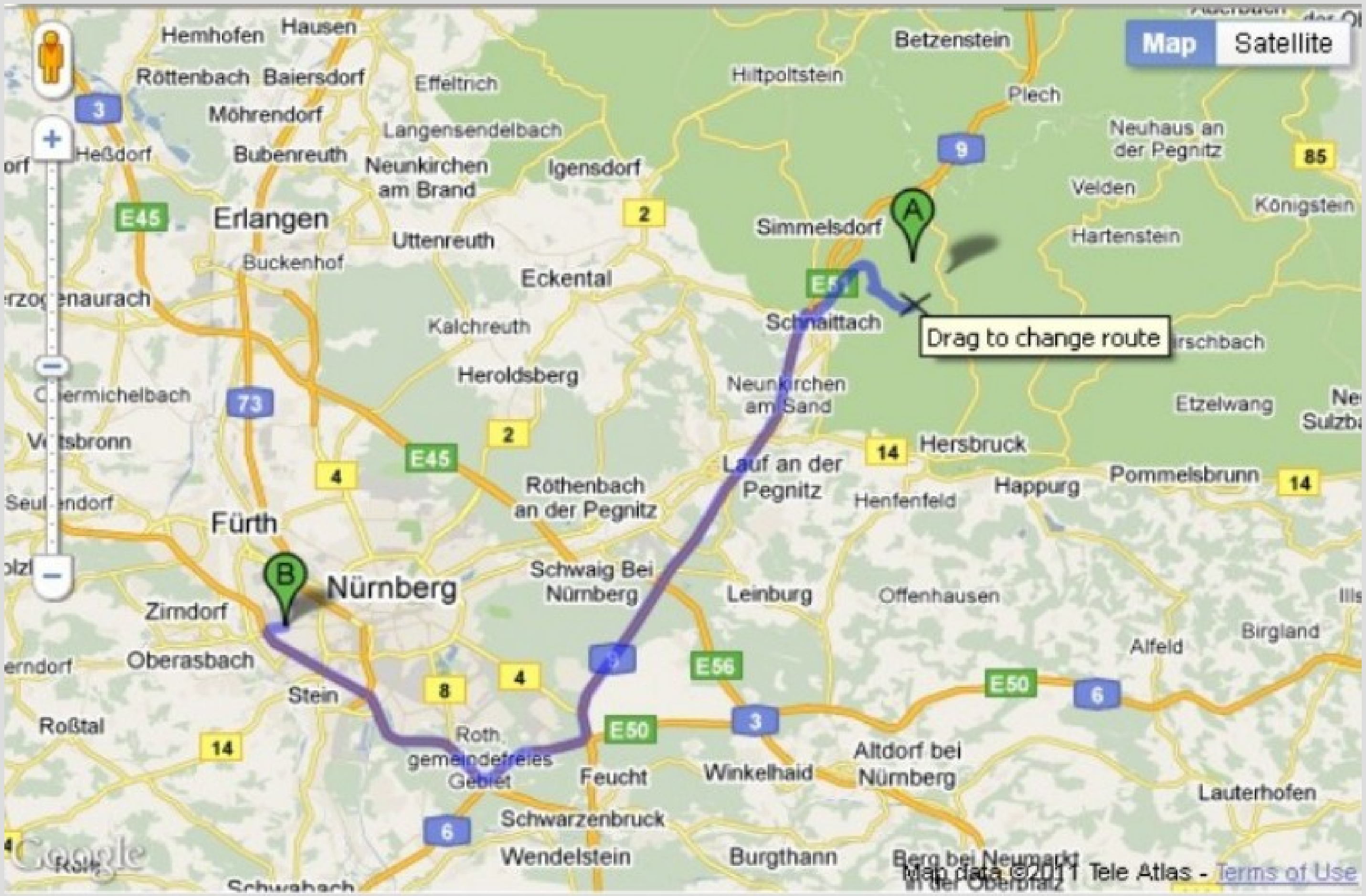
Planning is doing a sequence of actions to achieve one or more goals

This differs from search in that there are often multiple objectives that must be done

You can always reduce a planning problem to a search problem, but this is quite often very expensive

Search

Search: How to get from point A to point B quickly? (Only considering traveling)



Planning

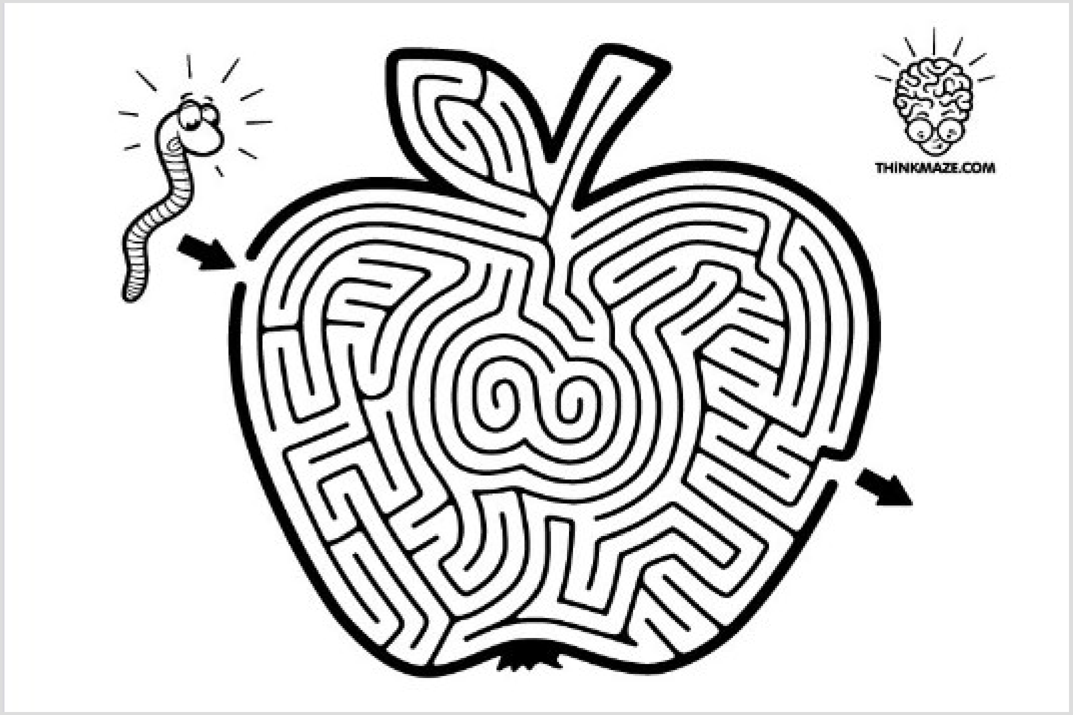
Planning: multiple tasks/subtasks need to be done and in what order? (pack, travel, unpack)

Search vs planning

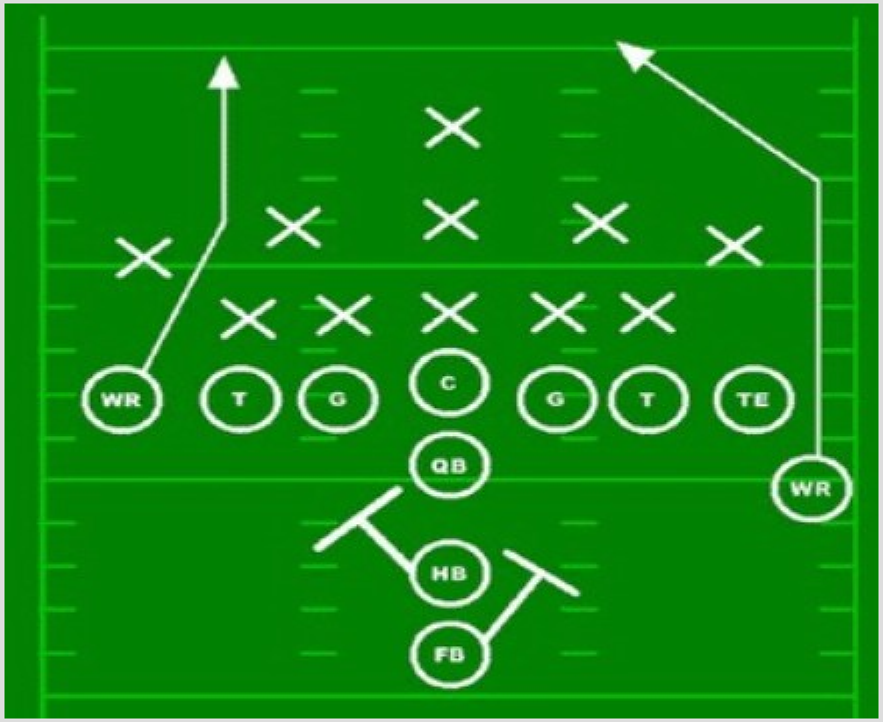
Searching: finding a single goal

Planning: must complete multiple tasks on the way to an ultimate goal

Search:



Plan:



Planning: definitions

The book uses Planning Domain Definition Language (PDDL) to represent states/actions

PDDL is very similar to first order logic in terms of notation (states are now similar to what our knowledge base was)

The large difference is that we need to define actions to move between states

Planning: assumptions

We make the same 3 assumptions as FO logic:

1. Objects are unique (i.e. $\neg(Bob = Jack)$)

2. All un-said sentences are false

   Thus if I only say: $Brother(James, Bob)$

   I also imply: $\neg Brother(James, Jack)$

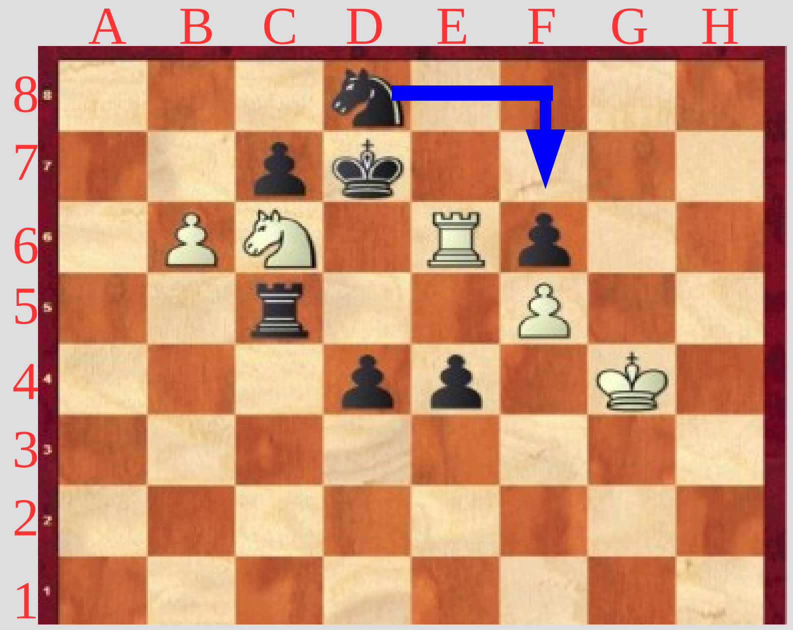
3. Only objects I have specified exists (i.e. There is no $Davis$ object unless I explicitly use it at some point)

Planning: actions

$$\text{State} = \boxed{BKnight(D,8)} \wedge BPawn(C,7) \wedge BKing(D,7) \wedge WPawn(B,6) \wedge WKnight(C,6) \wedge WRook(E,6) \wedge \ldots \wedge \boxed{Turn(Black)}$$

Apply: $MoveBKnight(D,8)$

$$\text{State} = \boxed{BKnight(F,7)} \wedge BPawn(C,7) \wedge BKing(D,7) \wedge WPawn(B,6) \wedge WKnight(C,6) \wedge WRook(E,6) \wedge \ldots \wedge \boxed{Turn(White)}$$

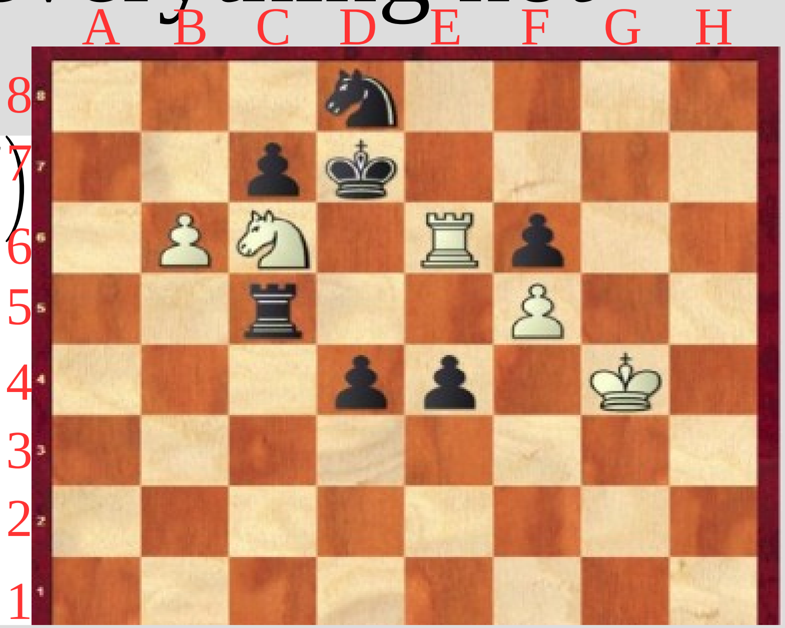


Planning: state

A state is all of the facts ANDed together in FO logic, but are not allowed to have:

1. Variables(otherwise it would not be specific)
2. Functions (just replace them with objects)
3. Negations (as we assume everything not mentioned is false)

$$State = BKnight(D,8) \wedge BPawn(C,7)$$
$$\wedge BKing(D,7) \wedge WPawn(B,6)$$
$$\wedge WKnight(C,6) \wedge WRook(E,6) \wedge \ldots$$
$$\wedge Turn(Black)$$



Planning: actions

Actions have three parts:

1. Name (similar to a function call)
2. Precondition (requirements to use action)
3. Effect (unmentioned states do not change)

For example:

Action($MoveBKnight1(x, y)$,

Precondition: $BKnight(x, y) \wedge Turn(Black)$,

Effect: $\neg BKnight(x, y) \wedge BKnight(x + 2, y - 1)$

$\wedge \neg Turn(Black) \wedge Turn(White)$)

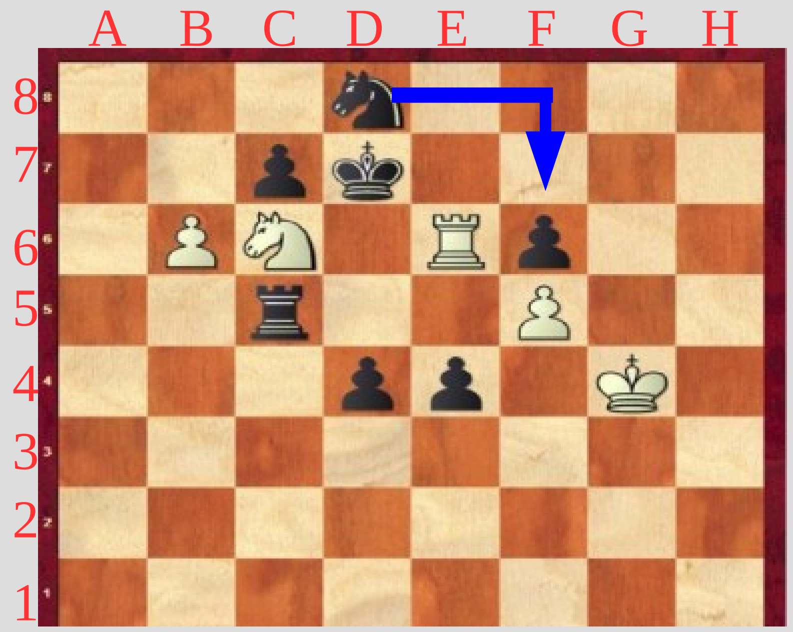
remove black's turn

Planning: actions

$$\text{State} = \boxed{BKnight(D,8)} \wedge BPawn(C,7)$$
$$\wedge BKing(D,7) \wedge WPawn(B,6)$$
$$\wedge WKnight(C,6) \wedge WRook(E,6) \wedge \ldots \wedge \boxed{Turn(Black)}$$

Apply: $MoveBKnight(D,8)$

$$\text{State} = \boxed{BKnight(F,7)} \wedge BPawn(C,7)$$
$$\wedge BKing(D,7) \wedge WPawn(B,6)$$
$$\wedge WKnight(C,6) \wedge WRook(E,6) \wedge \ldots$$
$$\wedge \boxed{Turn(White)}$$



Planning: example

Let's look at a grocery store example:
Objects = store locations and food items

$$\text{Goal} = At(Checkout) \wedge Cart(Milk) \wedge Cart(Apples)$$
$$\wedge Cart(Eggs) \wedge Cart(ToiletPaper) \wedge Cart(Bananas)$$
$$\wedge Cart(Bread) \wedge \neg Cart(Candy)$$

Aisle 1 = Milk, Eggs
Aisle 2 = Apples, Bananas
Aisle 3 = Bread, Candy, ToiletPaper

Planning: example

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

Action($AddApples()$,
Precondition: $At(Aisle2)$,
Effect: $Cart(Apples)$)

Action($AddMilk()$,
Precondition: $At(Aisle1)$,
Effect: $Cart(Milk)$)

Action($AddBananas()$,
Precondition: $At(Aisle2)$,
Effect: $Cart(Bananas)$)

Action($AddEggs()$,
Precondition: $At(Aisle1)$,
Effect: $Cart(Eggs)$)

Action($AddBread()$,
Precondition: $At(Aisle3)$,
Effect: $Cart(Bread)$)

Action($AddCandy()$,
Precondition: $At(Aisle3)$,
Effect: $Cart(Candy)$)

Action($AddToiletPaper()$,
Precondition: $At(Aisle3)$,
Effect: $Cart(ToiletPaper)$)

Planning: example

Initial state = At(Door)

A possible solution:

1. GoTo(Aisle1)
2. Add(Milk)
3. Add(Eggs)
4. GoTo(Aisle2)
5. Add(Apples)
5. GoTo(Aisle3)
6. Add(Bread)
7. Add(ToiletPaper)
8. GoTo(Aisle2)
8. Add(Bananas)
9. GoTo(Checkout)

Not most efficient, but goal reached

Planning: decidability

Since our planning is similar to FO logic, it is unsurprisingly semi-decidable as well

Thus, in general you will be able to find a solution if it exists, but possibly be unable to tell if a solution does not exist

If there are no functions or we know the goal can be found in a finite number of steps, then it is decidable

Planning: actions

If we treat the current state like a knowledge base and actions with $\forall$s for every variable...

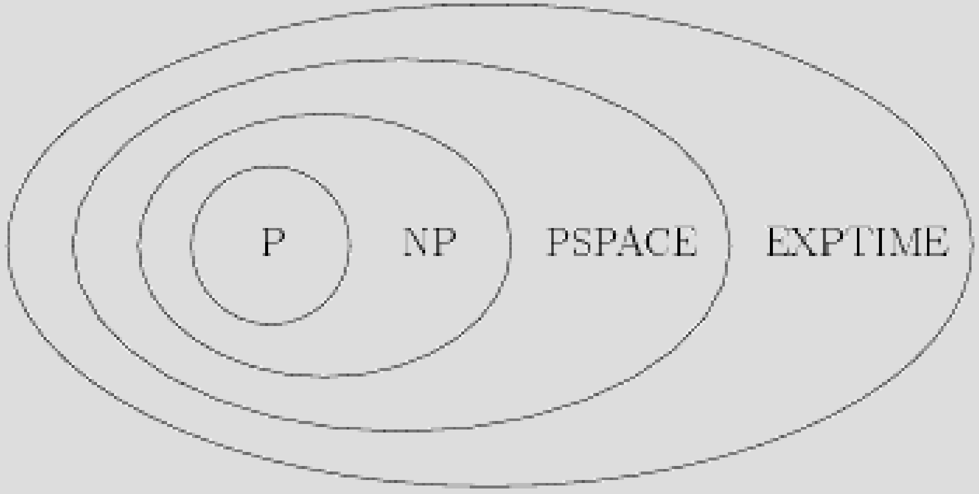
“state entails Precondition(A)” means action A's preconditions are met for the state

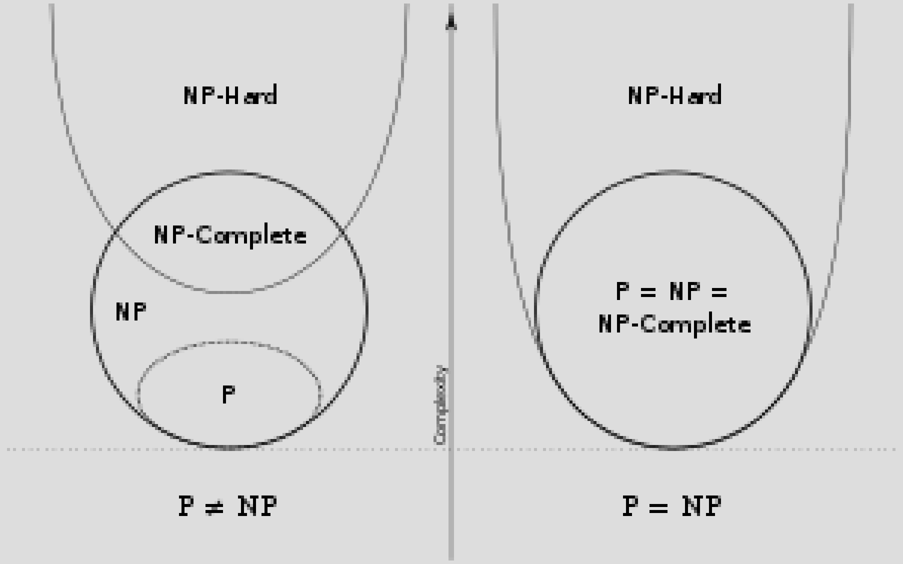
Thus if each action uses v variables, each with k possible values, there are $O(k^v)$ actions (we can ignore actions that do not change the current state in some cases)

Planning: difficulty

PlanSAT tells whether a solution exists or not, but takes PSPACE to tell

If negative preconditions are not allowed, we find a solution in P, and optimal in NP-hard





Planning: algorithms

Again similar to FO logic, there are two basic algorithms you can use to try and plan:

1. Forward search - similar to BFS and check all states you can find in 1 action, then 2 actions, then 3... until you find the goal state

2. Backward search - start at goal and try to work backwards to initial state

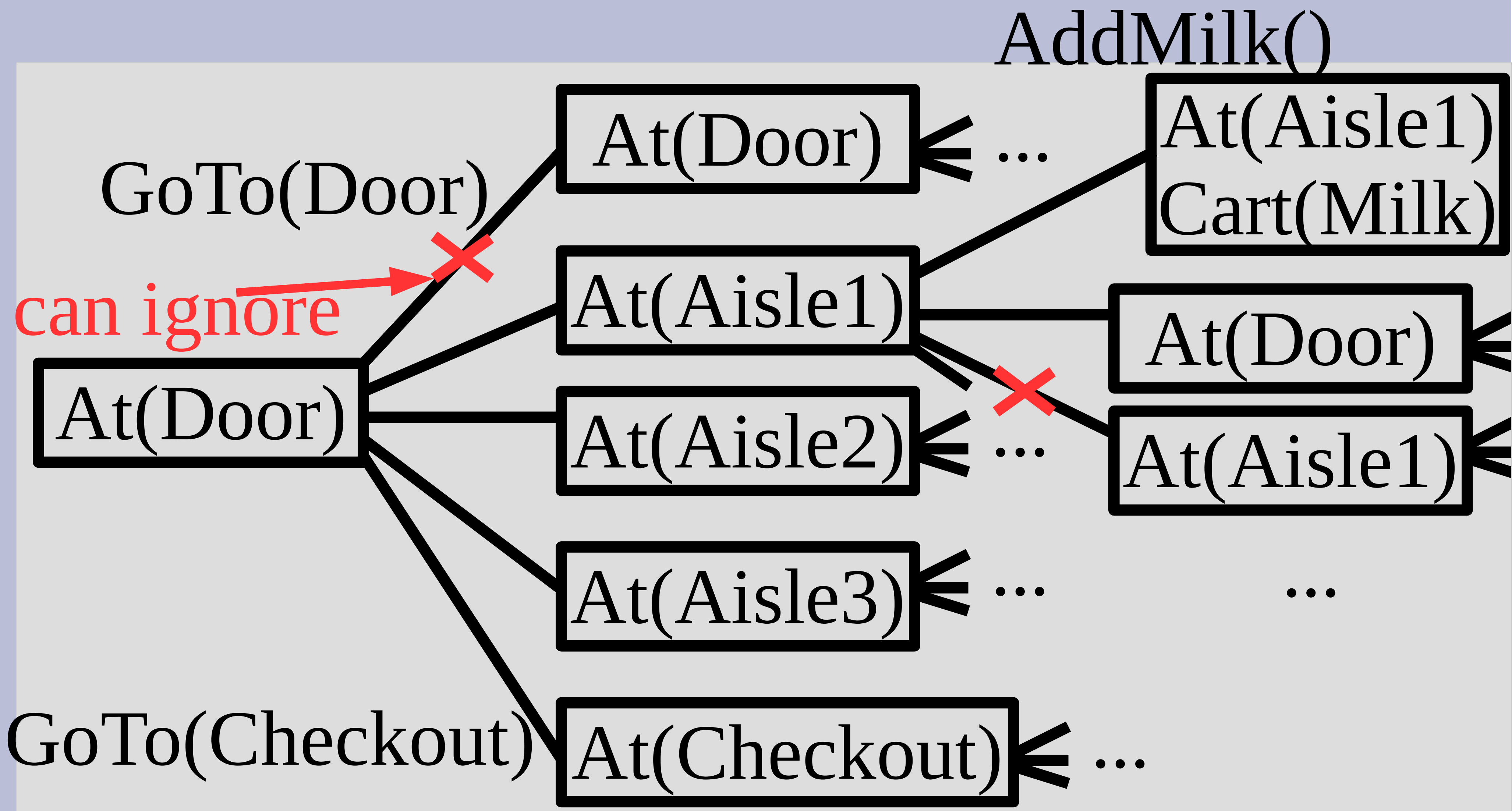
Forward search

Forward search is a brute force search that finds all possible states you can end up in

Each action is tested on each state currently known and is repeated until the goal is found

This can be quite costly, as actions that do not lead to the goal could be repeatedly explored (we will see a way to improve this)

Forward search



Forward search

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

You try it!

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

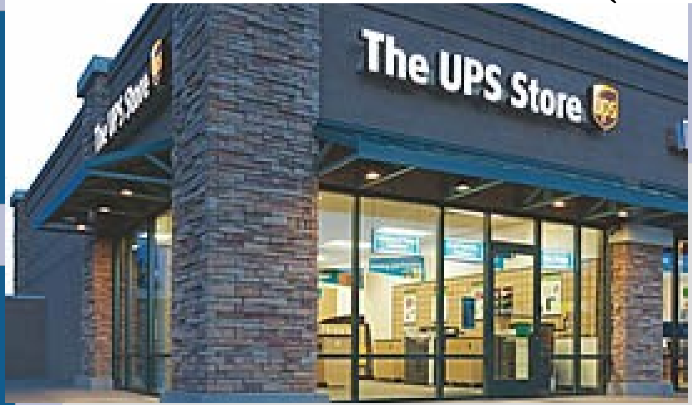
^ Package(UPSD, P2) ^ Mobile(Truck)

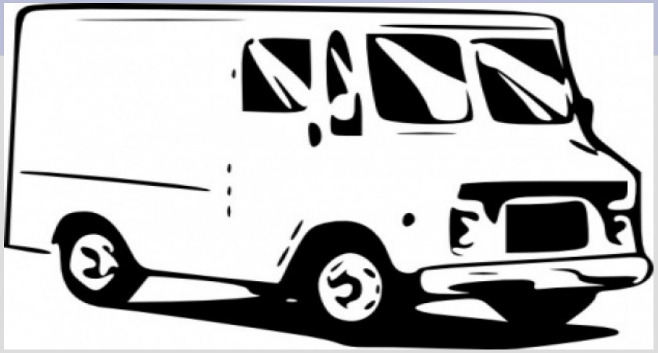
Goal: Package(H1, P1) ^ Package(H2, P2)

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

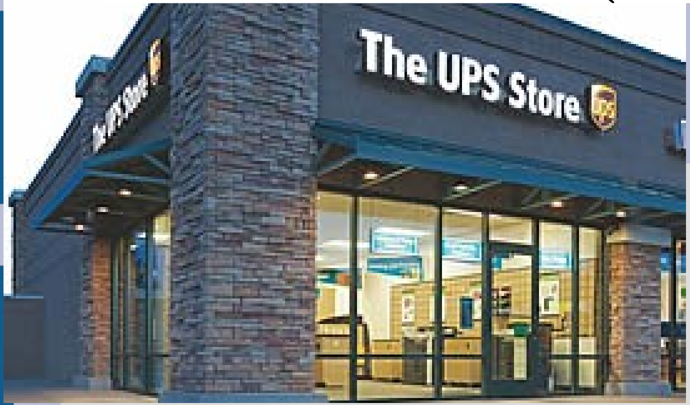
$$\text{Action}(\ Deliver(m, x, y),$$
$$\text{Precondition: } At(m, y) \wedge Package(m, x),$$
$$\text{Effect: } \neg Package(m, x) \wedge Package(y, x) \wedge At(m, y))$$

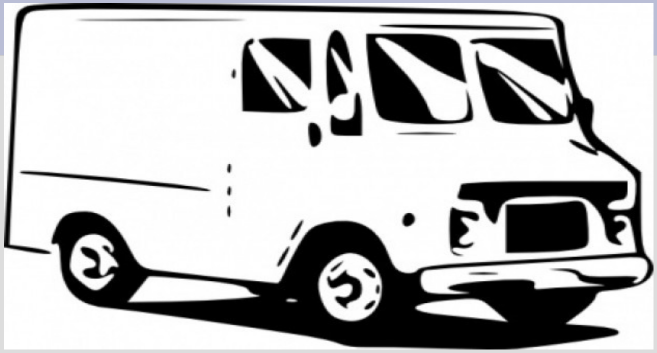
$$At(Truck, UPSD) \land Package(UPSD, P1) \land Package(UPSD, P2) \land Mobile(Truck)$$





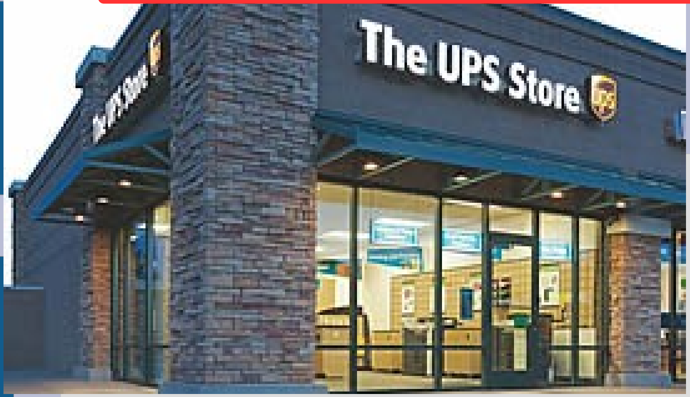
$$At(Truck, UPSD) \wedge Package(UPSD, P1)$$
$$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$$

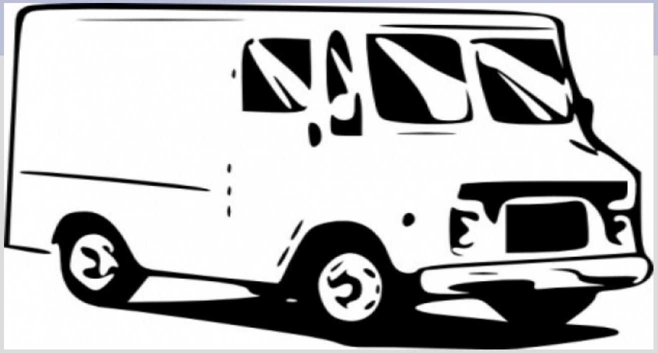




Action($Load(m, x, y)$,
Precondition: $At(m, y) \wedge Package(y, x)$,
Effect: $\neg Package(y, x) \wedge Package(m, x)$)

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





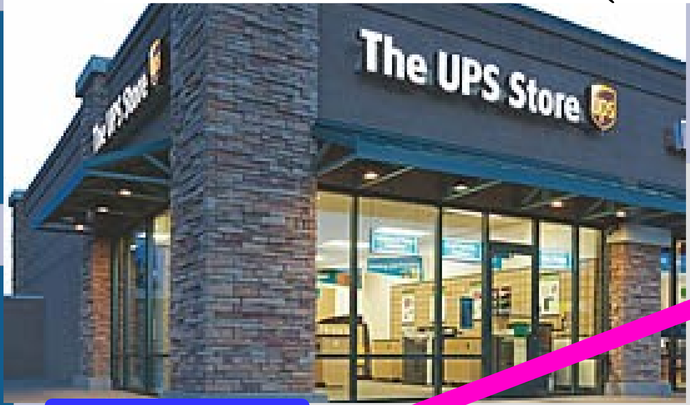
Find match
m/Truck
x/P1, y/UPSD

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

$At(Truck, UPSD) \wedge \cancel{Package(UPSD, P1)}$

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

$\boxed{\wedge Package(Truck, P1}$



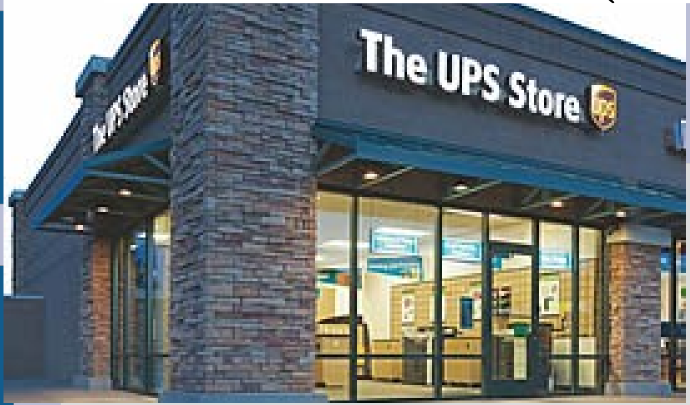
Apply effects

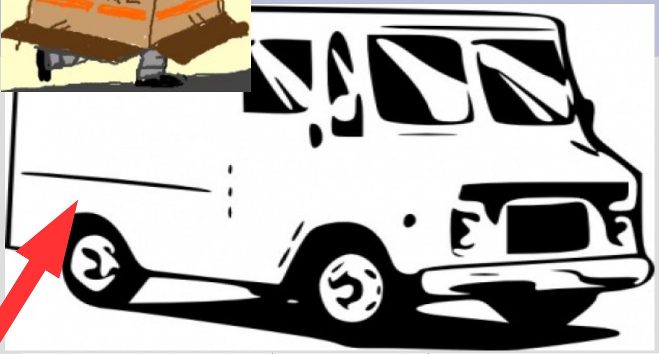
Action($Load(Truck, P1, UPSD)$,

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

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

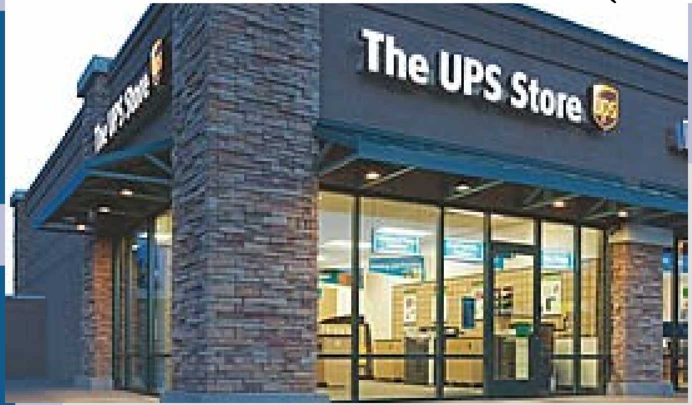
$$At(Truck, UPSD) \wedge Package(Truck, P1)$$
$$\wedge Package(UPSD, P2) \wedge Mobile(Truck)$$





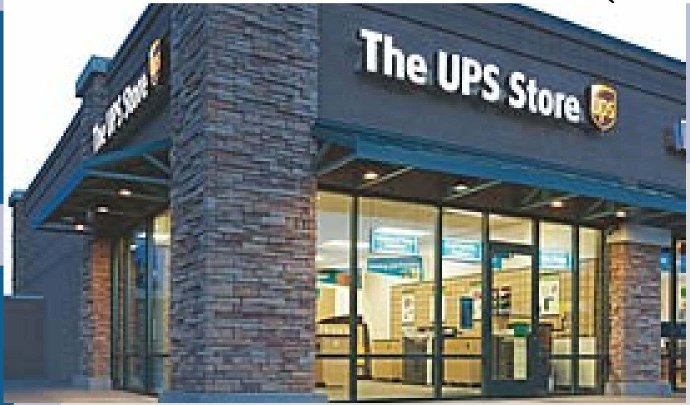
Action($Load(Truck, P2, UPSD)$,
Precondition: $At(Truck, UPSD) \wedge Package(UPSD, P2)$,
Effect: $\neg Package(UPSD, P2) \wedge Package(Truck, P1)$)

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



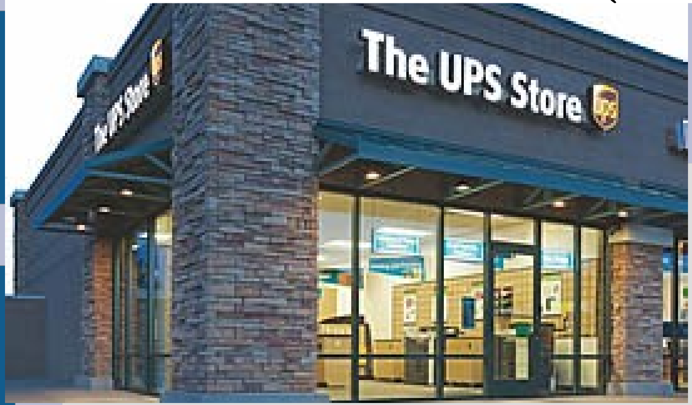
Action($GoTo(Truck, USPD, H1)$,
Precondition: $At(Truck, USPD) \wedge Mobile(Truck)$,
Effect: $\neg At(Truck, USPD) \wedge At(Truck, H1)$)

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

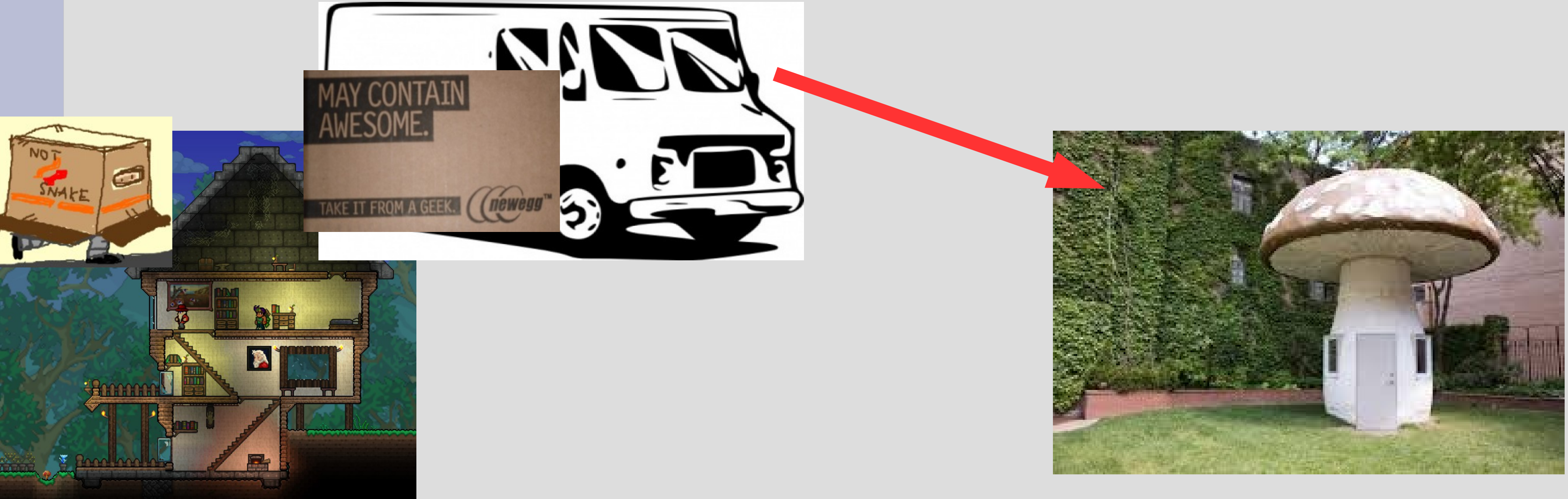


Action($Deliver(Truck, P1, H1)$,
Precondition: $At(Truck, H1) \wedge Package(Truck, P1)$,
Effect: $\neg Package(Truck, P1) \wedge Package(H1, P1)$)

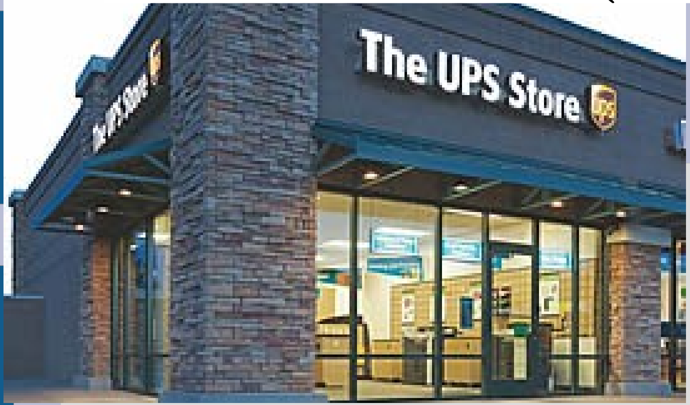
$$At(Truck, H1) \wedge Package(H1, P1)$$
$$\wedge Package(Truck, P2) \wedge Mobile(Truck)$$



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

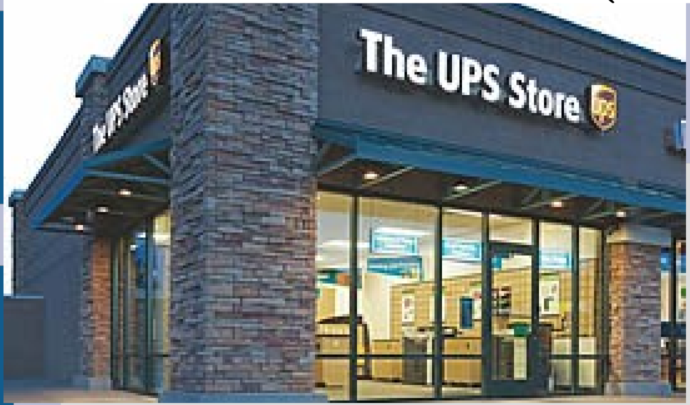


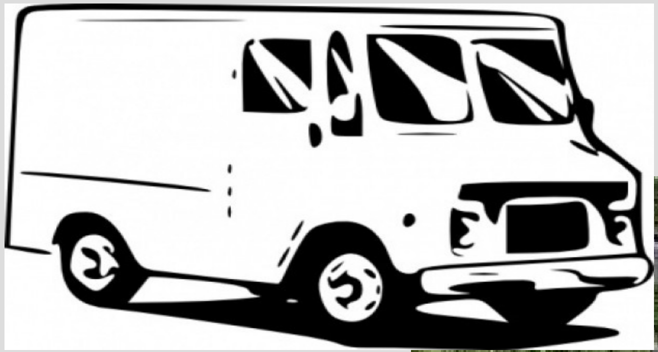
$At(Truck, H2) \wedge Package(H1, P1)$
$\wedge Package(Truck, P2) \wedge Mobile(Truck)$



Action($Deliver(Truck, P2, H2)$,
Precondition: $At(Truck, H2) \wedge Package(Truck, P2)$,
Effect: $\neg Package(Truck, P2) \wedge Package(H2, P2)$)

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





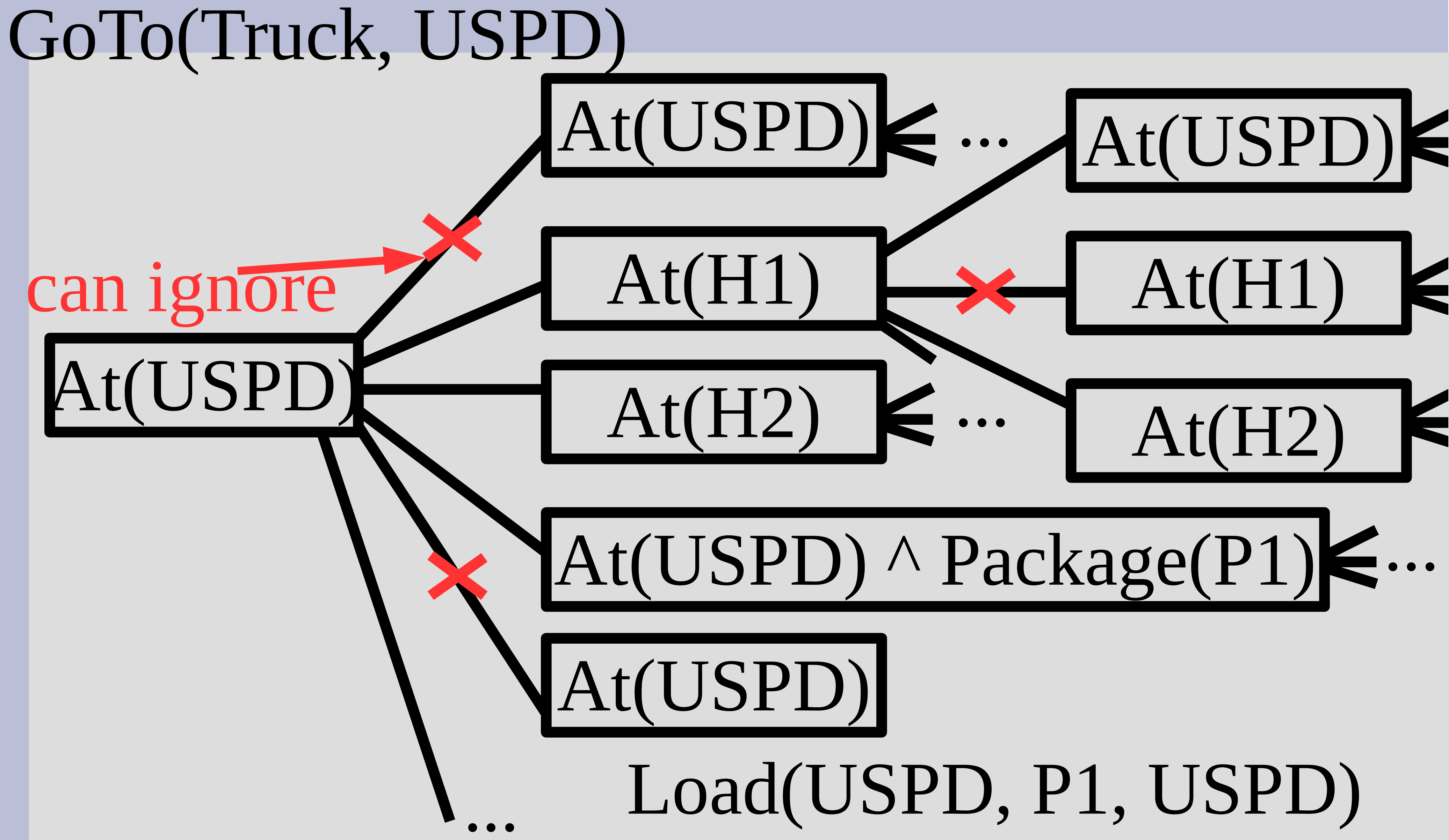
Forward search

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 (Move, 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

You can convert by:

1. Removing action effects (in reverse)
   All positive relations are removed
   If we are using negative relations, all negative relations are removed from state
2. Adding in precondition effects

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

Backward search

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

So if we started with: Package(H2,P2)
Substitute: y/H2, x/P2 (m can stay just “m”)
Remove positives: Package(H2,P2)
Add negatives: (nothing to do)
Add precondition: At(m,H2) ^ Package(m,P2)

Final result: At(m,H2) ^ Package(m, P2)