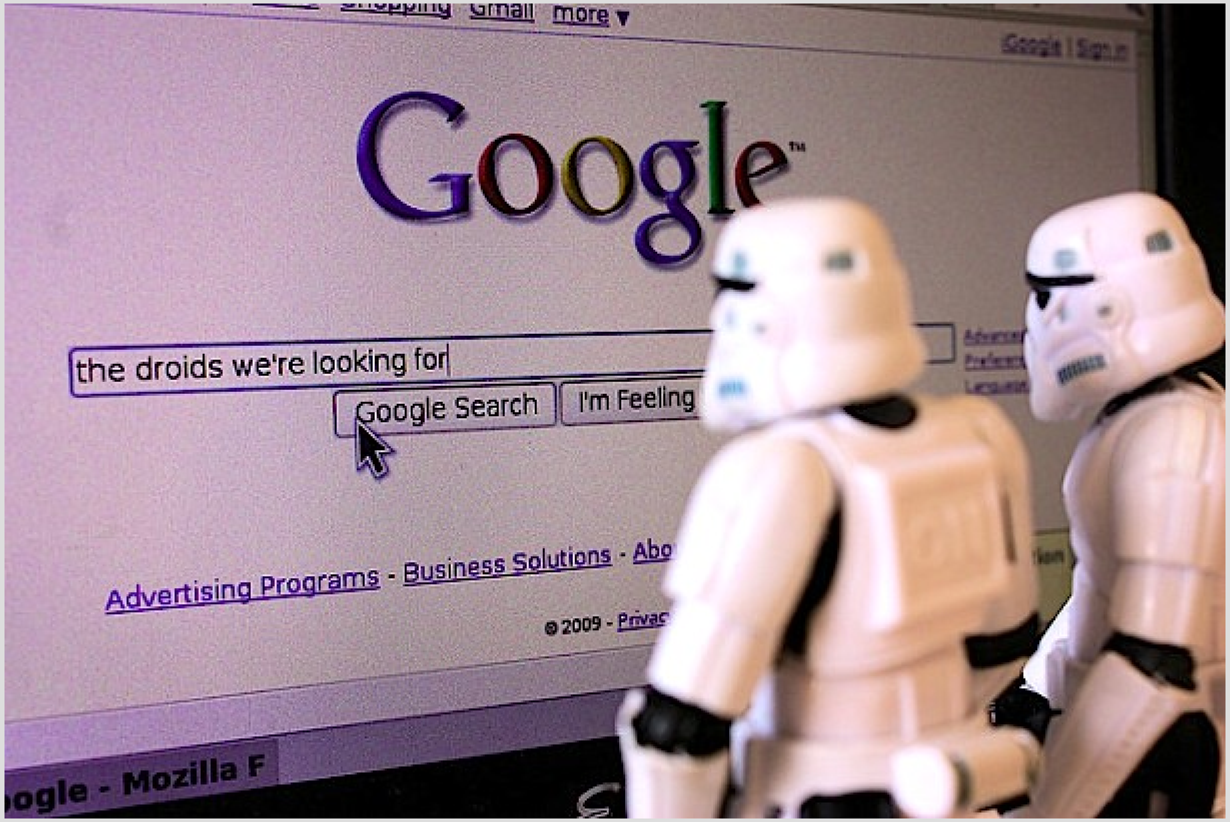
Planning (Ch. 10)



Planning

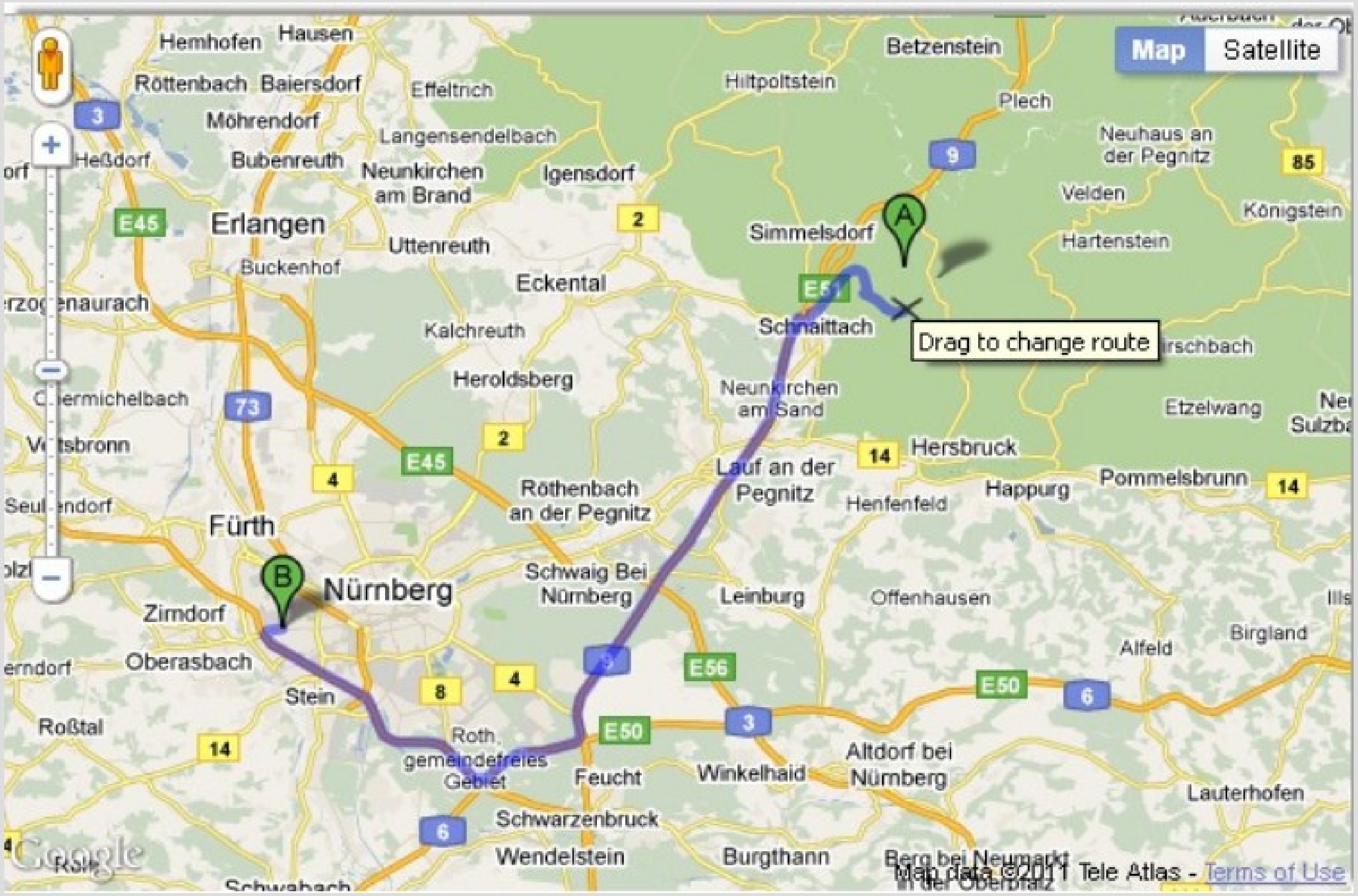
Planning is doing a sequence of actions to achieve one or more goals

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

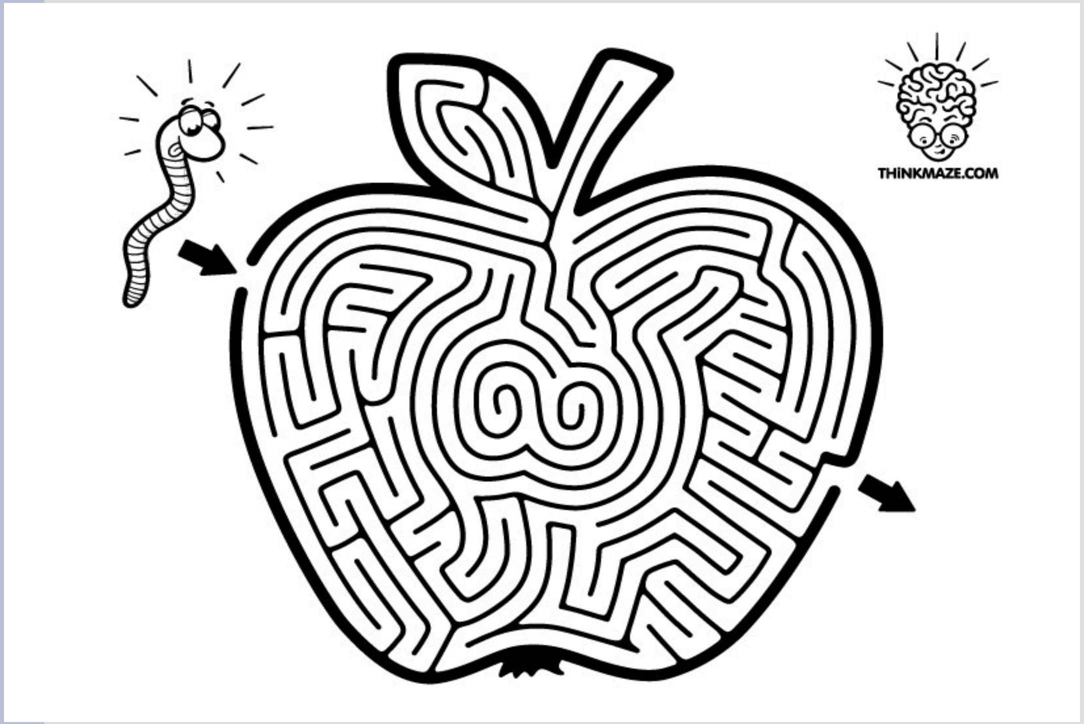
Planning: multiple tasks/subtasks need to be done and in what order? (pack, travel, unpack)

Search vs planning

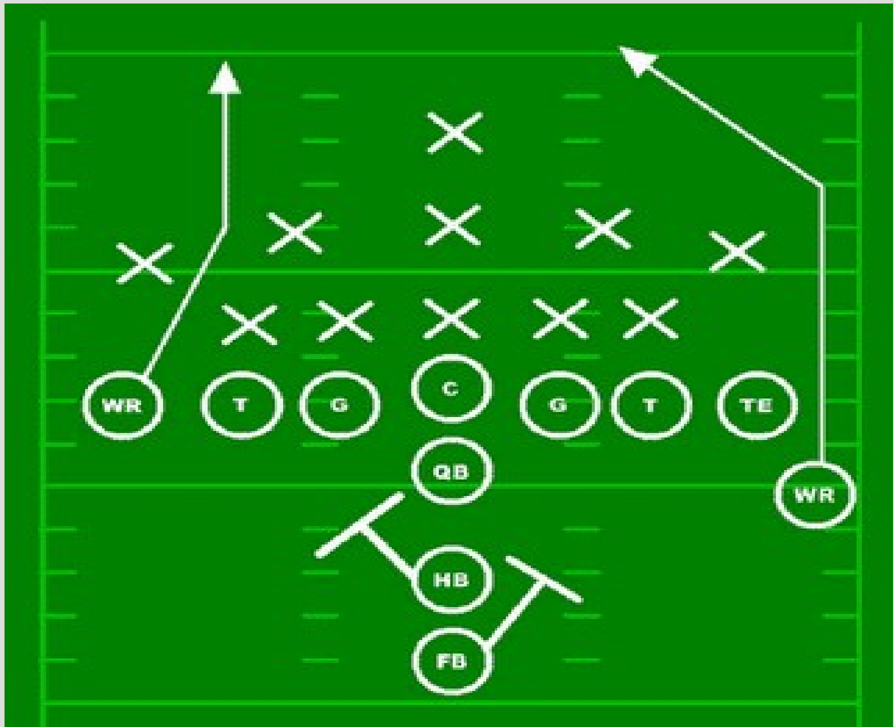
Searching: finding a single goal

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: $\neg Brother(James, Jack)$

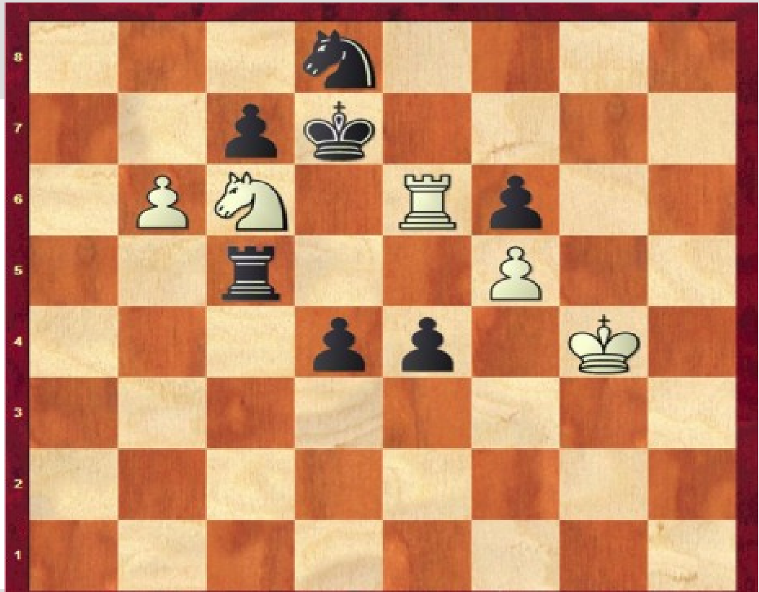
3. Only objects I have specified exists (i.e. There is no $Davis$ object unless I explicitly use it at some point)

Planning: 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)

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



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

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

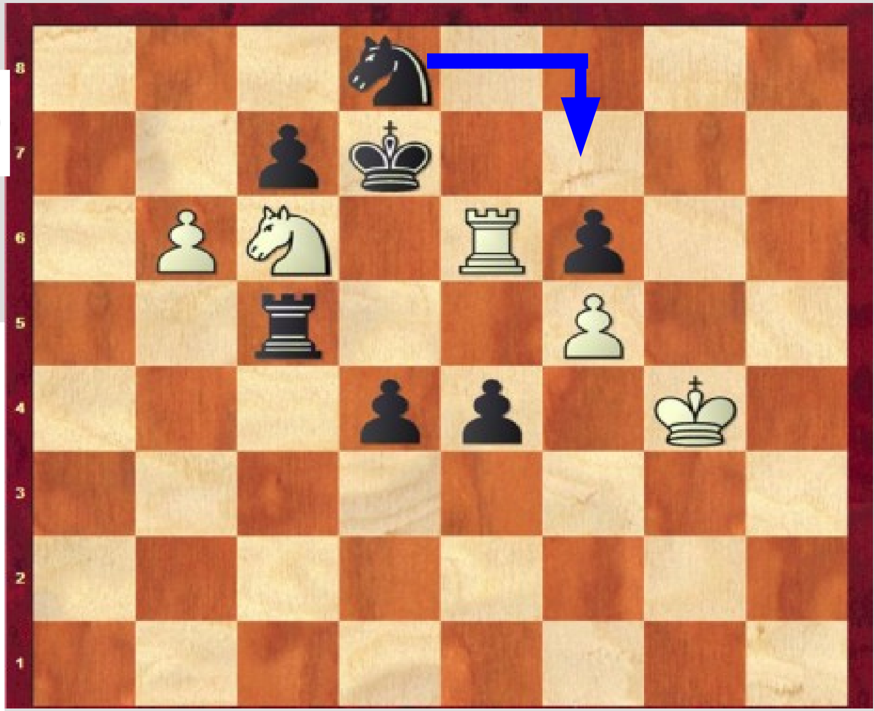
remove black's turn

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

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

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 ∀s for every variable...

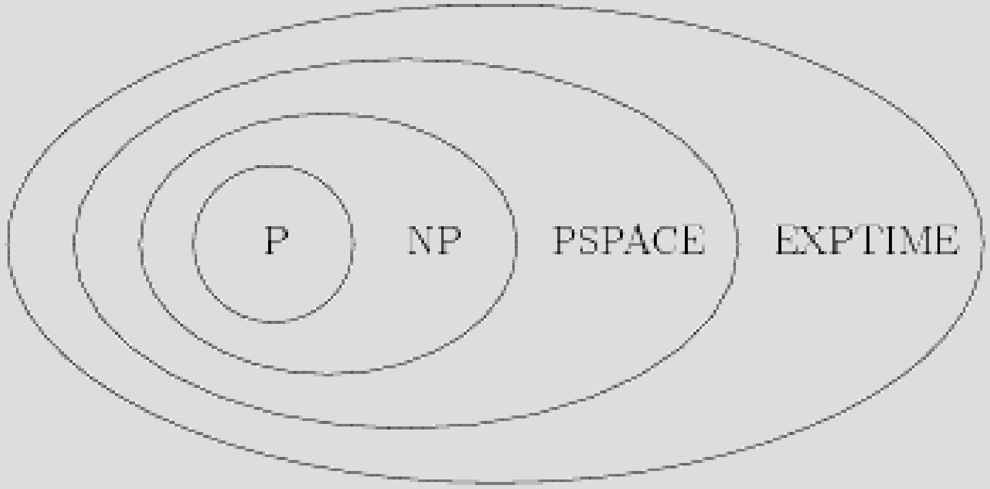
“state entails Precondition(A)” means action A's preconditions are met for the state

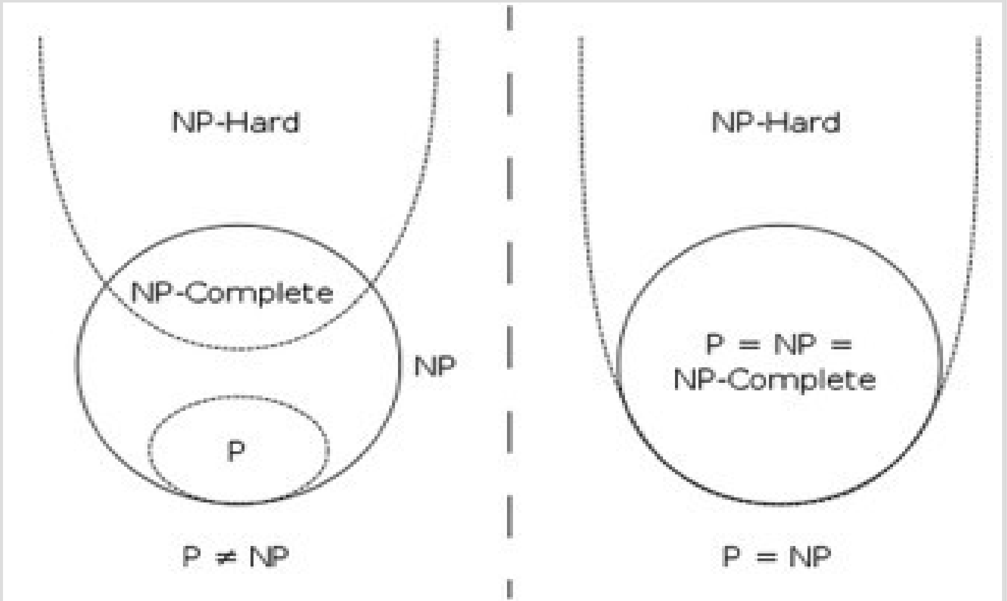
Thus if each action uses v variables, each with k possible values, there are $O(k^v)$ actions (we can ignore actions that do not change the current state in some cases)

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

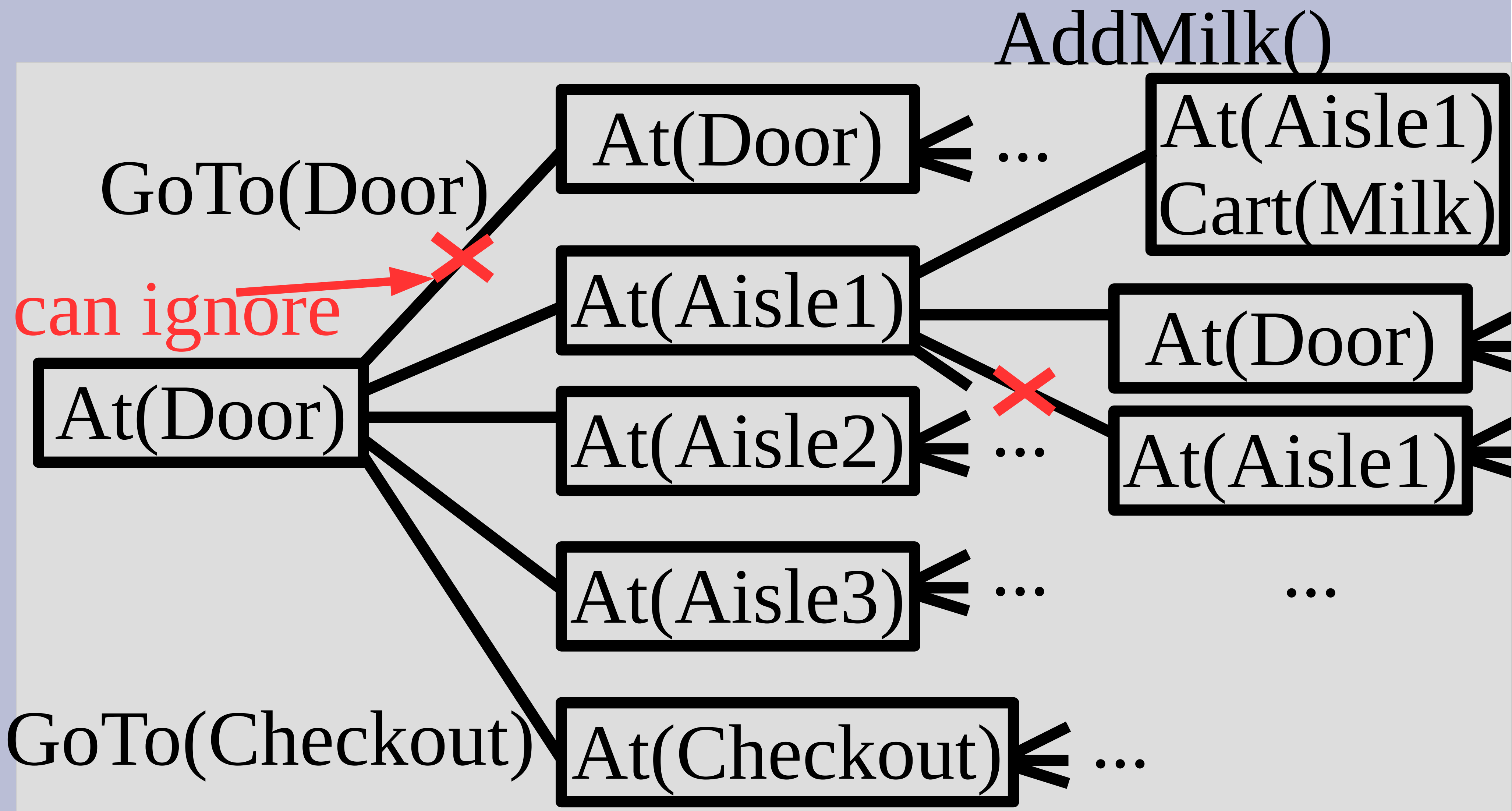
Forward search

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

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

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

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

Forward search

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

Can I simplify to this?

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

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

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

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

Forward search

No...

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

We can do:

Load(H1, P1, UPSD) (m=H1, x=P1, y=UPSD)

As our current state is:

At(UPSD) ^ Package(UPSD, P1)^Package(UPSD, P2)

Somehow we just transported the package from the UPSD to H1?

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:

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

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

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

Backward search

We also need some easy definition of what is “away from the goal”

Some problems, such as n-queens do not have a great “away from goal” definition

While stepping backwards, you need to find any “swapped” preconditions that are applicable and find a valid substitution

Heuristics for planning

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

Heuristics in planning are also the same, we want an admissible one found from relaxing the problem and solving that optimally

There are two ways to always do this:

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

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)