Planning

Representation and reasoning about actions in the situation calculus

Planning in STRIPS

Why planning

So far we studied problem solving by search, and knowledge representation and reasoning

However, intelligent agents are not simply passive problem solvers or reasoners

Intelligent agents must act on the world.

We want them to act in intelligent ways

- taking purposeful actions,
- predicting the expected effect of such actions,
- composing actions together to achieve complex goals

Activity Planning for the Mars Exploration Rovers (MER)

Operating MER is a challenging, time-pressured task.

Each day, the operations team must generate a new plan describing the rover activities for the next day.

These plans must abide by resource limitations, safety rules, and temporal constraints.

Automated planning is used in generating the plans.

Planning in Manufacturing Automation

Sheet-metal bending machines

Plan the sequence of bends

Planning in Bridge Baron

Bridge Baron is a computer program that plays bridge.

It won the 1997 world championship of computer bridge.

It uses AI planning techniques to plan its declarer play.

Other applications

Scheduling

- Supply chain management
- Hubble Space Telescope scheduler
- Workflow management

Air Traffic Control

- Route aircraft between runways and terminals.
- Crafts must be kept safely separated.
- Safe distance depends on craft and mode of transport.
- Minimize taxi and wait time.

Other applications

Character Animation

- Generate step-by-step character behaviour from high-level spec

Plan-based intelligent user interfaces

Web Service Composition

Genome Rearrangement

Planning

To do planning, we need to reason about what the world will be like after doing a number of actions.

Only now we want to reason about dynamic environments.

- in(robby,room1), lightOn(room1) are true: will they be true after robby performs the action turnOffLights?
- in(robby,room1) is true: what does robby need to do to make in(robby,room3) true?

Reasoning about the effects of actions, and computing what actions can achieve certain effects is at the heart of decision making.

Planning under Uncertainty

One of the major complexities in planning that we will deal with later is planning under uncertainty.

> Our knowledge of the world will almost certainly be incomplete. We may wish to model this probabilistically.
>
> Sensing is subject to noise (especially in robots).
>
> Actions and effectors are also subject to error (uncertainty in their effects).

Classical Planning

For now we restrict our attention to the deterministic case.

- Complete initial state specifications
- Deterministic effects of actions

What we will cover

- Representing and reasoning about actions in the situation calculus
- Planning in STRIPS
- Planning as search

Situation Calculus

First we look at how to represent and reason about dynamic worlds within first-order logic.

The situation calculus is an important formalism developed for this purpose.

Building blocks: actions, situations, fluents

Actions

Action functions such as $pickup(x)$, $stack(x, y)$

A set of primitive action objects in the domain of individuals.

Action functions mapping objects to primitive action objects.

$pickup(x)$ is a function mapping block x to the primitive action object corresponding to picking up block x

$stack(x, y)$ is a function mapping blocks x and y to the primitive action object corresponding to stacking x on top of y

Situations

Situations denote possible action histories

A set of situation objects in the domain of individuals.

A special constant S_0 denoting the initial situation, where no actions have been performed

A special function $do(a, s)$ denoting the situation resulting from doing action a in situation s

e.g., $do(put(a, b), do(put(b, c), S_0))$

Note that situations are different from states

e.g., when you flip a switch once and three times, the action histories are different, but the states are the same

Fluents

- Fluents are predicates or functions whose values may vary from situation to situation
- These are written using predicate or function symbols whose last argument is a situation
- *e.g.*, $Holding(r, x, s)$: robot r is holding object x in situation s
- Can have:
  $\neg Holding(r, x, s) \wedge Holding(r, x, do(pickup(r, x), s))$

Preconditions and effects

Actions typically have preconditions: what needs to be true for the action to be performed

- to do $pickup(r, x)$ in situation s, we need to have $\forall z.\neg Holding(r, z, s) \wedge \neg Heavy(x) \wedge NextTo(r, x, s)$
- to do $repair(r, x)$ in situation s, we need to have $HasGlue(r, s) \wedge Broken(x, s)$

Actions typically have effects: the fluents that change as the result of performing the action

- $Fragile(x) \supset Broken(x, do(drop(r, x), s))$
- $\neg Broken(x, do(repair(r, x), s))$

An example of preconditions and effects

Action effects are conditional on their preconditions being true.

To specify action $pickup(x)$, we have

$$\begin{aligned}\forall s, x.\ & ontable(x, s) \wedge clear(x, s) \wedge handempty(s) \\ & \rightarrow holding(x, do(pickup(x), s)) \\ & \wedge \neg handempty(do(pickup(x), s)) \\ & \wedge \neg ontable(x, do(pickup(x), s)) \\ & \wedge \neg clear(x, do(pickup(x), s))\end{aligned}$$

Plan generation

There are many ways to generate plans.

Here we show how to do it by representing actions in the SitCalc and generating a plan via deductive plan synthesis.

This is not the approach taken by state-of-the-art planners, as we will see later,

but it is where the field started,

and is still used for specifying and studying more complex tasks in reasoning about actions and change.

Plan generation

Let's use Resolution to find a plan.

The KB contains a description of

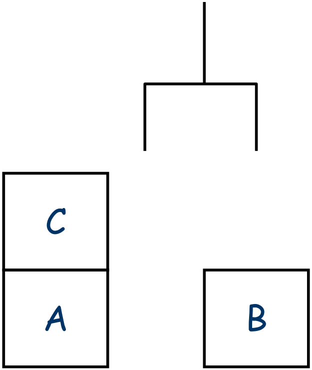
- the initial state of the world
- the precondition axioms for actions
- the effect axioms for actions

The query is the goal condition that we wish to achieve

Plan generation

1. $clear(c, S_0)$
2. $on(c, a, S_0)$
3. $clear(b, S_0)$
4. $ontable(a, S_0)$
5. $ontable(b, S_0)$
6. $handempty(S_0)$

Query: $\exists z.holding(b, z)$
7. $(\neg holding(b, z), ans(z))$



Does there exists a situation in which the robot is holding b?
If so, what is the name of that situation?

Plan generation

Convert "pickup" action axiom into clause form:

$$\begin{aligned} \forall s, x.\ ontable(x, s) \wedge clear(x, s) \wedge handempty(s) \\ \rightarrow holding(x, do(pickup(x), s)) \\ \wedge \neg handempty(do(pickup(x), s)) \\ \wedge \neg ontable(x, do(pickup(x), s)) \\ \wedge \neg clear(x, do(pickup(x), s)) \end{aligned}$$

8. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
   $holding(x, do(pickup(x), s)))$
9. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s), ,$
   $\neg handempty(do(pickup(x), s)))$
10. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
    $\neg ontable(x, do(pickup(x), s)))$
11. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s), ,$
    $\neg clear(x, do(pickup(x), s)))$

All clauses

1. $clear(c, S_0)$
2. $on(c, a, S_0)$
3. $clear(b, S_0)$
4. $ontable(a, S_0)$
5. $ontable(b, S_0)$
6. $handempty(S_0)$
7. $(\neg holding(b, z), ans(z))$
8. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
   $holding(x, do(pickup(x), s)))$
9. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
   $\neg handempty(do(pickup(x), s)))$
10. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
    $\neg ontable(x, do(pickup(x), s)))$
11. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
    $\neg clear(x, do(pickup(x), s)))$

Resolution

12. R[7a,8d]x=b,z=~~holding(b,~~ do(pickup(~~x~~),s)~~)~~
    $(\neg ontable(b,s), \neg clear(b,s), \neg handempty(s), ans(do(pickup(b),s)))$
13. R[5,12a]s=S_0
    $(\neg clear(b,S_0), \neg handempty(S_0), ans(do(pickup(b),S_0)))$
14. R[3,13a] $(\neg handempty(S_0), ans(do(pickup(b),S_0)))$
15. R[6,14a] $ans(do(pickup(b),S_0))$

Thus the plan to achieve "holding(b)" from situation S_0 is simply one action: pickup(b).

Two common types of reasoning

Temporal projection:
Predicting the effects of a given sequence of actions
e.g., $on(b, c, do(stack(b, c), do(pickup(b), S_0)))$

Plan synthesis:
Computing a sequence of actions that achieve a goal condition
e.g., $\exists s.on(b, c, s) \wedge on(c, a, s)$

The frame problem

Unfortunately, although $on(c, a, do(pickup(b), S_0))$ should hold, it can't be proved via resolution.

1. $clear(c, S_0)$
2. $on(c, a, S_0)$
3. $clear(b, S_0)$
4. $ontable(a, S_0)$
5. $ontable(b, S_0)$
6. $handempty(S_0)$
7. $(\neg holding(b, z), ans(z))$
8. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
   $holding(x, do(pickup(x), s)))$
9. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
   $\neg handempty(do(pickup(x), s)))$
10. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
    $\neg ontable(x, do(pickup(x), s)))$
11. $(\neg ontable(x, s), \neg clear(x, s), \neg handempty(s),$
    $\neg clear(x, do(pickup(x), s)))$
12. $\neg on(c, a, do(pickup(b), S_0))$

Nothing can resolve with 12!

What's wrong?

We stated the effects of pickup(b), but did not state that it doesn't affect on(c,a).

Problem: need to know a vast number of such non-effects.

Few actions affect the value of a given fluent; most leave it invariant.

e.g., an object's colour is unaffected by picking things up, opening a door, using the phone, turning on a light, electing a new Prime Minister of Canada, etc.

The frame problem

Finding an effective way of specifying the non-effects of actions, without having to explicitly write them all down is the frame problem.

Good solutions have been proposed, *e.g.*, successor state axioms.

Computation problems

SitCalc is a very powerful representation

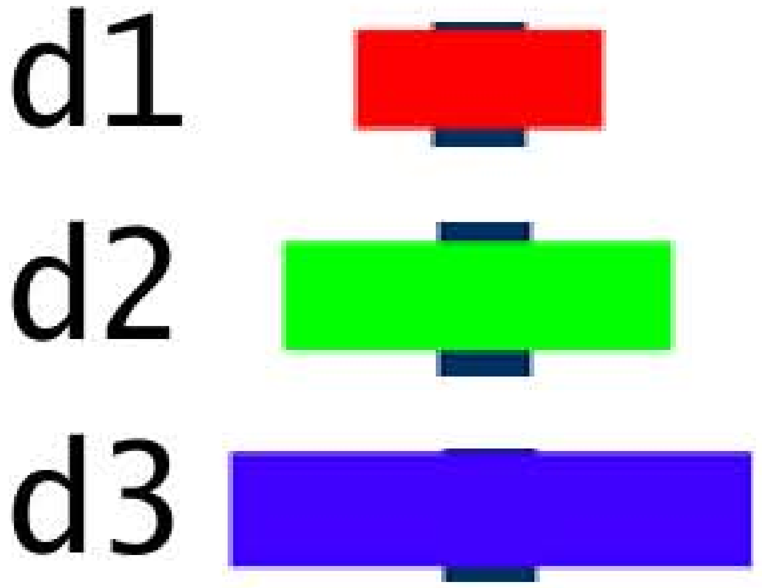
Very good for studying the formal properties of planning and for modelling more complex issues in planning.

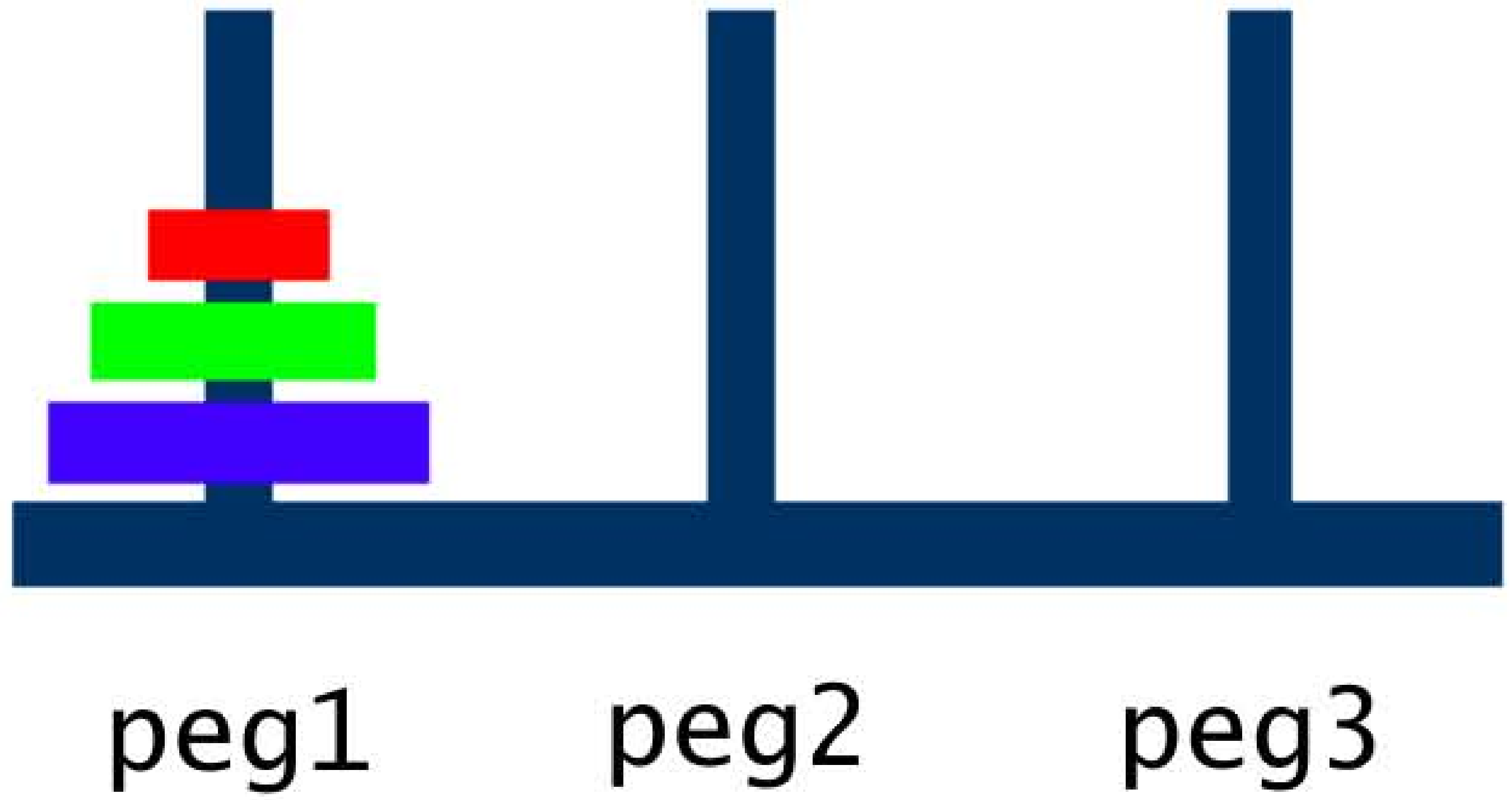
But it is not efficient enough for automated plan generation.

Next we will study some less expressive representations that support more efficient planning.

Exercise

将汉诺塔的移动问题形式化为一个规划问题。假设有3个盘子





我们使用6个常量：$peg1, peg2, peg3, d1, d2, d3$，3个谓词：

1. $clear(x)$：柱x上没有盘子，或者盘x上没有其他盘子；
2. $on(x, y)$：盘x在盘y上，或者盘x在柱y上并且盘x不在其他盘子上；
3. $smaller(x, y)$：盘x比盘y小，或者x是盘并且y是柱子。

假设只有一个动作：$move(disc, from, to)$：将盘$disc$从盘或柱$from$上移到盘或柱to上。

试写出$move$动作的公理，初始知识库，以及目标。

Situation calculus

Plan generation: Computing a sequence of actions that achieve a goal condition

Temporal projection: Predicting the effects of a given sequence of actions

SitCalc: actions, situations, fluents

The preconditions and effects of actions

Plan generation in SitCalc via resolution

The frame problem

Simplifying the Planning Problem

Perform Classical Planning. No incomplete or uncertain knowledge.

Assume complete information about the initial state through the closed-world assumption (CWA).

Assume action preconditions are restricted to conjunctions of ground atoms.

Assume action effects are restricted to making ground atoms true or false. No conditional effects, no disjunctive effects, etc.

Closed World Assumption (CWA)

The knowledge base used to represent a state of the world is a list of positive ground atomic facts. (Like a database.)

Closed World Assumption (CWA) is the assumption that

- the constants mentioned in KB are all the domain objects.
- if a ground atomic fact is not in our list of "known" facts, its negation must be true.

This gives complete information about the state of the system.

Querying a Closed World KB

CWA treats a knowledge base like a database:

- *e.g.*, if $employed(John, CIBC)$ is not in the database, we conclude that $\neg employed(John, CIBC)$ is true.

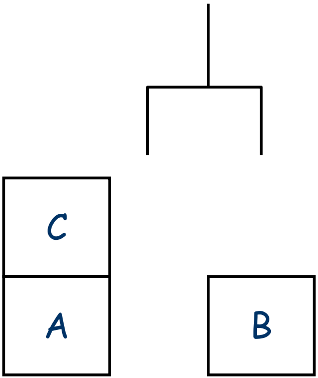
Such a KB is called a CW-KB (Closed World Knowledge Base)

With a CW-KB, we can evaluate the truth or falsity of arbitrarily complex first-order formulas.

- The CW-KB acts as a model

This process is very similar to query evaluation in databases.

CWA example



KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

Arbitrary queries:

1. $clear(c) \wedge clear(b)$?
2. $\neg on(b,c)$?
3. $on(a,c) \vee on(b,c)$?
4. $\exists X.on(X,c)$? (D = {a,b,c})
5. $\forall X.ontable(X) \rightarrow X = a \vee X = b$?

STRIPS representation

STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Fikes and Nilsson in 1971.

The same name was later used to refer to the formal language of the inputs to this planner.

Actions are modeled as ways of modifying the world.

Since the world is represented as a CW-KB, a STRIPS action represents a way of updating the CW-KB.

An action yields a new KB, describing the new world – the world resulting from executing the action

SitCalc vs. STRIPS

In the SitCalc, we could

Have incomplete information (specified by a FOL formula)

Refer to different situations at the same time in one formula,

- *e.g.*, $on(a, b, s_0) \wedge \neg on(a, b, s_1)$

In STRIPS, we have

Complete information (specified by a CW-KB)

Information about the state of the world at different times would have to be captured by two separate databases.

STRIPS Actions

STRIPS represents an action using 3 lists.

- A list of action preconditions.
- A list of action add effects.
- A list of action delete effects.

These lists contain variables, so that we can represent a whole class of actions with one specification.

Each ground instantiation of the variables yields a specific action.

STRIPS Actions: Example

pickup(X):

- Pre: {handempty, clear(X), ontable(X)}
- Adds: {holding(X)}
- Dels: {handempty, clear(X), ontable(X)}

"pickup(X)" is called a STRIPS operator.

a particular instance (e.g. "pickup(a)") is called an action.

Operation of a STRIPS action

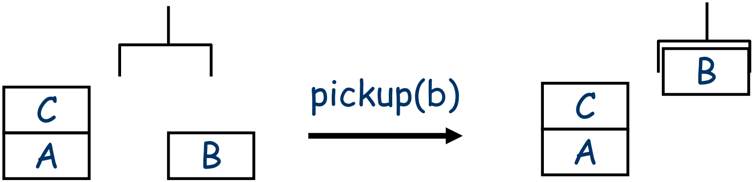
For a particular STRIPS action (ground instance) to be applicable to a state (a CW-KB)

- Every fact in its precondition list must be true in KB.
- This amounts to testing membership since we have only atomic facts in the precondition list.

If the action is applicable, the new state is generated by

- Removing all facts in Dels from KB, then
- Adding all facts in Adds to KB.

Operation of a Strips Action: Example



pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}

STRIPS Blocks World Operators (1)

pickup(X) (from the table)
Pre: {clear(X), ontable(X),handempty}
Add: {holding(X)}
Dels: {clear(X), ontable(X),handempty}

putdown(X) (on the table)
Pre: {holding(X)}
Add: {clear(X), ontable(X), handempty}
Del: {holding(X)}

STRIPS Blocks World Operators (2)

unstack(X,Y) (pickup from a stack of blocks)
Pre: {clear(X), on(X,Y), handempty}
Add: {holding(X), clear(Y)}
Del: {clear(X), on(X,Y), handempty}

stack(X,Y) (putdown on a block)
Pre: {holding(X),clear(Y)}
Add: {on(X,Y), handempty, clear(X)}
Del: {holding(X),clear(Y)}

8 Puzzle as a planning problem

The constants

A constant representing each position: P1,···,P9

P1	P2	P3
P4	P5	P6
P7	P8	P9

A constant for each tile (and blank): B,T1, ···, T8.

8 Puzzle

The predicates

adjacent(X,Y): position X is next to position Y

- *e.g.*, adjacent(P5,P2), adjacent(P5,P4), adjacent(P5,P8), ...

at(X,Y): tile X is at position Y

- *e.g.*, at(T1,P1), at(T2,P2), at(T5,P3), ...

P1	P2	P3
P4	P5	P6
P7	P8	P9

1	2	5
7	8	
6	4	3

8 Puzzle

A single operator (creating lots of ground actions)

slide(T,X,Y)

Pre: {at(T,X), at(B,Y), adjacent(X,Y)}

Add: {at(B,X), at(T,Y)}

Del: {at(T,X), at(B,Y)}

at(T1,P1), at(T5,P3), at(T8,P5), at(B,P6), ...,

1	2	5
7	8	
6	4	3

slide(T8,P5,P6) →

at(T1,P1), at(T5,P3), at(B,P5), at(T8,P6), ...,

1	2	5
7		8
6	4	3

STRIPS has no Conditional Effects

Unlike ADL (Action Description Language) which we'll see later, STRIPS has no conditional effects

e.g., if we allow conditional effects, we can treat putdown(X) as stack(X,Y) where Y=table,

Since STRIPS has no conditional effects, we must sometimes utilize extra actions: one for each type of condition.

We embed the condition in the precondition, and then alter the effects accordingly.

Beyond STIPS

STRIPS operators are not very expressive

ADL (Action Description Language) extends the expressivity of STRIPS

ADL operators add a number of features to STRIPS

- Their preconditions can be arbitrary formulas
- They can have conditional and universal effects

ADL Operators Example

Blocks World Assumptions:
The table has infinite space, so it is always clear.

If we stack something on the table (Y=Table), we cannot delete clear(Table),

But if Y is an ordinary block we must delete clear(Y).

move(X,Y,Z)
Pre: $on(X, Y) \wedge clear(Z)$
Effs: ADD[on(X,Z)]
DEL[on(X,Y)]
Z $\neq$ table $\rightarrow$ DEL[clear(Z)]
Y $\neq$ table $\rightarrow$ ADD[clear(Y)]

ADL Operators Example

move(c,a,b)
Pre: on(c,a) $\wedge$ clear(b)
Effs: ADD[on(c,b)]
DEL[on(c,a)]
b $\neq$ table $\rightarrow$ DEL[clear(b)]
a $\neq$ table $\rightarrow$ ADD[clear(a)]

KB = { clear(c), clear(b),
on(c,a),
on(a,table),
on(b,table)}

KB = { on(c,b)
clear(c), clear(a)
on(a,table),
on(b,table)}

ADL Operators with universal effects

clearTable()
Pre:
Effs: $\forall X.on(X, table) \rightarrow DEL[on(X, table)]$

KB=$\{on(a, table), on(b, table), on(c, table)\}$ $\Rightarrow$ KB=$\{\}$

Planning as a Search Problem

Given

- a CW-KB representing the initial state,
- A set of STRIPS or ADL operators mapping a state to a new state
- a goal condition (specified as a formula)

The planning problem is to determine a sequence of actions that when applied to the initial CW-KB yield an updated CW-KB which satisfies the goal.

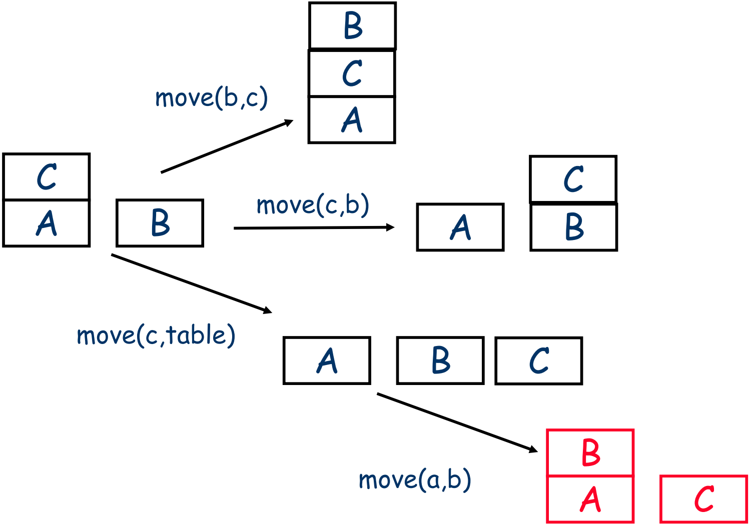
This is known as the classical planning task.

Planning as Search

This is a search problem in which our state representation is a CW-KB.

- The initial CW-KB is the initial state.
- The actions are operators mapping a state (a CW-KB) to a new state (an updated CW-KB).
- It can be checked if the goal is satisfied by any state (CW-KB).
- Typically the goal is a conjunction of ground facts, so we just need to check if all of them are contained in the CW-KB.

An example



Problem and solution

Problem: Search tree is generally quite large

- randomly reconfiguring 9 blocks takes thousands of CPU seconds

But the representation suggests some structure.

- Each action only affects a small set of facts

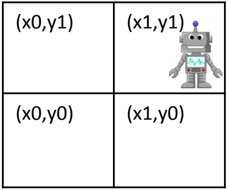
Planning algorithms are designed to take advantage of the fact that the representation makes the "locality" of action changes explicit.

We will look at one technique: Relaxed Plan heuristics used with heuristic search.

The heuristics are domain independent. So the technique belongs to domain-independent heuristic search for planning

Exercise

如图所示，有4个房间。
开始时，机器人在房间$(x1, y1)$。
机器人的目标是访问所有的房间。



我们使用4个常量：$locx0y0, locx0y1, locx1y0, locx1y1$，
3个谓词：

$at(loc)$：机器人在房间loc；

$visited(loc)$：机器人访问过房间loc；

$connected(loc1, loc2)$：房间$loc1$与房间$loc2$相邻。

假设只有一个动作：$move(from, to)$：机器人从房间$from$移动到房间to，前提是这两个房间相邻。

试写出$move$动作的STRIPS表示，初始知识库，目标，及一个解

Planning in STRIPS

The world is represented as a CW-KB, *i.e.*, a list of ground atoms with CWA

- the constants mentioned in KB are all the domain objects
- if a ground atom is not in the list, it is false

An action is represented using 3 lists: Pre, Add and Del

ADL extends STRIPS with arbitrary preconditions, conditional and universal effects

Classical planning

Given

- a CW-KB representing the initial state,
- A set of STRIPS operators mapping a state to a new state
- a goal condition

Determine a sequence of actions that transforms the initial CW-KB to a CW-KB satisfying the goal

Relaxed problem

We make the assumption that

- the precondition of each action is a set of positive facts, and
- the goal is a set of positive facts.

Recall that we can obtain heuristics by solving a relaxed version of 8-puzzle in which we relax one of the restrictions:

- move to adjacent field only
- move to blank field only

The idea here is similar: consider what happens if we ignore the delete lists of actions.

This yields a "relaxed problem" that can produce a useful heuristic estimate.

STRIPS Blocks World Operators - Relaxed

- pickup(X)
  Pre: {handempty, ontable(X), clear(X)}
  Add: {holding(X)}
  ~~Del: {handempty, ontable(X), clear(X)}~~
- putdown(X)
  Pre: {holding(X)}
  Add: {handempty, ontable(X), clear(X)}
  ~~Del: {holding(X)}~~
- unstack(X,Y)
  Pre: {handempty, clear(X), on(X,Y)}
  Add: {holding(X), clear(Y)}
  ~~Del: {handempty, clear(X), on(X,Y)}~~
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {handempty, clear(X), on(X,Y)}
  ~~Del: {holding(X),clear(Y)}~~

Relaxed problem

Theorem. The length of an optimal plan for the relaxed problem is bounded by the length of an optimal plan for the original problem.

Proof:

Let P be the original problem, and P' the relaxed problem.

We show that $Sols(P) \subseteq Sols(P')$.

Then it follows that $Minlen(Sols(P')) \leq Minlen(Sols(P))$.

Proof

Let $a_0, \ldots, a_{n-1}$ be a solution to P.

We show that it is also a solution to P'.

Let s_0 denote the initial state.

For $i < n$, let $s_{i+1} = s_i \cup add(a_i) - del(a_i)$.

Then $Goal \subseteq s_n$, and for $i < n$, $pre(a_i) \subseteq s_i$.

Let $s'_0 = s_0$, and for $i < n$, let $s'_{i+1} = s'_i \cup add(a_i)$.

We show by induction on i that for $i \leq n$, $s_i \subseteq s'_i$.

Thus $Goal \subseteq s_n \subseteq s'_n$, and for $i < n$, $pre(a_i) \subseteq s_i \subseteq s'_i$.

So $a_0, \ldots, a_{n-1}$ is also a solution to P'.

Proof

We show by induction on i that for $i \leq n$, $s_i \subseteq s'_i$.

Basis: $i = 0$, obviously.

Induction: Let $i < n$. Assume $s_i \subseteq s'_i$. We prove $s_{i+1} \subseteq s'_{i+1}$.

Since a_i is applicable in s_i, $pre(a_i) \subseteq s_i$.

Hence $pre(a_i) \subseteq s_i \subseteq s'_i$. So a_i is applicable in s'_i.

So $s_{i+1} = s_i \cup add(a_i) - del(a_i) \subseteq s'_i \cup add(a_i) = s'_{i+1}$.

Computing the heuristic

So the length of an optimal relaxed plan could serve as an admissible heuristic for A^*.

However, it is NP-Hard to compute an optimal relaxed plan

So how do we compute the heuristic?

Build a layered structure from a state S that reaches the goal.

Count how many actions are required in a relaxed plan.

Use it as our heuristic estimate of the distance of S to goal.

Reachability analysis

Start with the initial state S_0.

Alternate between state and action layers.

Find all actions whose preconditions are contained in S_0.

These actions comprise the first action layer A_0.

The next state layer $S_1 = S_0$ U all facts added by actions in A_0.

Continue this process

Reachability analysis

In general,

- A_i: set of actions not in A_{i-1} whose preconditions are in S_i
- $S_i = S_{i-1}$ U the add lists of all actions in A_i

Intuitively

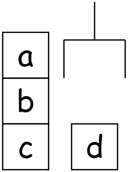
- the actions at level A_i are the actions that could be executed at the i-th step of some plan, and
- the facts in level S_i are the facts that could be made true within a plan of length i

Some of the actions/facts have this property. But not all!

We continue until

- the goal G is contained in the state layer, or
- the state layer no longer changes (reaches fix point).

Example

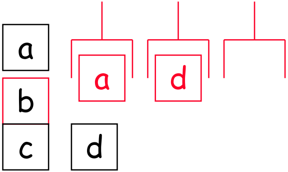


on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

A_0

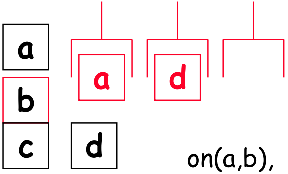


on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state as some of these facts cannot be true at the same time!

Example



on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)

S_1

$\left.\begin{array}{l}\text{unstack(a,b)}\\ \text{pickup(d)}\end{array}\right\}$ from $\boxed{A_0}$

putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,a),
stack(d,b),
stack(d,d),
unstack(b,c)
...

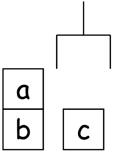
Impossible, but we don't know because we ignore dels.

$\boxed{A_1}$

Example

Reachability

to reach on(c,b) requires 4 actions



on(a,b),
ontable(c),
ontable(b),
clear(a),
clear(c),
handempty

S_0

unstack(a,b)
pickup(c)

A_0

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

S_1

stack(c,b)
...

A_1

on(c,b),
...

but stack(c,b) cannot be executed after one step

Properties of state and action layers

Proposition. Let $a_0, \ldots, a_{n-1}$ be a sequence of actions applicable in S_0. Let $s_0 = S_0$, and for $i < n$, $s_{i+1} = s_i \cup add(a_i) - del(a_i)$. Then for $i < n$, there exist $j, k \leq i$ s.t. $a_i \in A_k$ and $s_i \subseteq S_j$.

We prove by induction on i:

Basis: $i = 0$, obviously.

Induction: Let $i < n$. Assume there exists $j \leq i$ s.t. $s_i \subseteq S_j$.

Since $pre(a_i) \subseteq s_i$, $pre(a_i) \subseteq S_j$.

Let k be the least $u \leq j$ s.t. $pre(a_i) \subseteq S_u$.

Then $a_i \in A_k$.

So $add(a_i) \subseteq S_{k+1} \subseteq S_{j+1}$.

Thus $s_{i+1} \subseteq s_i \cup add(a_i) \subseteq S_j \cup S_{j+1} = S_{j+1}$.

Unsolvability

Theorem. Suppose the state layer stops changing and the goal is not satisfied. Then the original planning problem is not solvable.

Proof:

Assume that $a_0, \ldots, a_{n-1}$ is a solution to the original problem.

Then $Goal \subseteq s_n$.

By Proposition, there exists $m \leq n$ s.t. $Goal \subseteq s_n \subseteq S_m$.

This contradicts the assumption.

CountActions

Suppose the goal G is contained in the state layer

We want to compute a good relaxed plan

The idea is to choose a minimum subset of A_i for each i

CountActions

CountActions(G,S_K):
/* Here G is contained in S_K, and we compute the number of actions contained in a relaxed plan achieving G. */

If $K = 0$ return 0

Split G into $G_P = G \cap S_{K-1}$ and $G_N = G - G_P$

- G_P contains the previously achieved (in S_{K-1}), and
- G_N contains the just achieved parts of G (only in S_K).

Find a minimal set of actions A whose add effects cover G_N.

- cannot contain redundant actions,
- but may not be the minimum sized set
  (computing the minimum sized set of actions is the set cover problem and is NP-Hard)

NewG := G_P U preconditions of A.

return CountAction(NewG,S_{K-1}) + size(A)

The set cover problem

Given a set of elements $\{1, 2, ..., n\}$ (called the universe) and a collection S of m sets whose union equals the universe

Identify the smallest sub-collection of S whose union equals the universe.

e.g., $U = \{1, 2, 3, 4, 5\}$, $S = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$.
A smallest cover is $\{\{1, 2, 3\}, \{4, 5\}\}$.

An example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$
$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$
$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$
$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$
$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

Goal: f_6, f_5, f_1

Actions:

$[f_1]a_1[f_4]$
$[f_2]a_2[f_5]$
$[f_2, f_4, f_5]a_3[f_6$

An example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \; [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$\mathbf{G = \{f_6,f_5,f_1\}}$

$\mathbf{G_N = \{f_6\}}$ **(newly achieved)**

$\mathbf{G_p = \{f_5, f_1\}}$ **(achieved before)**

An example

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$\mathbf{G = \{f_6,f_5, f_1\}}$

We split G into G_P and G_N:

CountActs(G,S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup$ **Pre(A)**

$\mathbf{G_1 = \{f_5,f_1,f_2,f_4\}}$

Return

1 + CountActs(G_1,S_1)

An example

Now, we are at level S1

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], \ [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1,f_2,f_3,f_4,f_5\}$

$A_1 = \{[f_2,f_4,f_5]a_3[f_6]\}$

$S_2 = \{f_1,f_2,f_3,f_4,f_5,f_6\}$

$G_1 = \{f_5,f_1,f_2,f_4\}$

We split G_1 into G_P and G_N:

$\mathbf{G_N} = \{f_5,f_4\}$

$\mathbf{G_P} = \{f_1,f_2\}$

CountActs(G_1,S_1)

$G_P = \{f_1,f_2\}$ **//already in S0**

$G_N = \{f_4,f_5\}$ **//New in S1**

$A = \{a_1,a_2\}$ **//adds all in G_N**

//the new goal: $G_P \cup$ Pre(A)

$\mathbf{G_2 = \{f_1,f_2\}}$

Return

2 + CountActs(G_2,S_0)

Next, we are at level S0, simply return 0

So, in total CountActs(G,S2)=1+2+0=3

CountActions

Theorem. Suppose that $Goal \subseteq S_k$. For $i < k$, let A'_{i-1} denote the A obtained by calling CountActions(G_i,S_i). Then $A'_0, \ldots, A'_{k-1}$ is a relaxed plan.

Proof: We prove by induction on $i \leq k$ that $A'_0, \ldots, A'_{i-1}$ is a relaxed plan for achieving G_i.

> Basis: $i = 0$. We have $G_0 = G_1 \cap S_0 \cup pre(A'_0)$. Since $Pre(A_0) \subseteq S_0$, $G_0 \subseteq S_0$. So the empty plan achieves G_0.
>
> Induction: Let $i < k$. Assume $A'_0, \ldots, A'_{i-1}$ achieves G_i.
>
> We have $G_P = G_{i+1} \cap S_i$, $G_i = G_P \cup pre(A'_i)$, and $G_N = G_{i+1} - G_P \subseteq add(A'_i)$.
>
> So $pre(A'_i) \subseteq G_i$, $G_{i+1} - G_i \subseteq G_{i+1} - G_P \subseteq add(A'_i)$.
>
> Thus $A'_0, \ldots, A'_{i-1}, A'_i$ is a relaxed plan for achieving G_{i+1}.

CountActions

However, CountActions does NOT compute the length of an optimal relaxed plan, because.

> The choice of which action set to use to achieve G_N ("just achieved part of G") is not necessarily optimal
>
> Even if we picked a true minimum set A at each stage, we might not obtain a minimum set of actions for the entire plan, since the set A picked at each stage influences what set can be used at the next stage!

CountActions

It is NP-Hard to compute an optimal relaxed plan.

> So CountActions cannot be made into an admissible heuristic without making it much harder to compute.
>
> Empirically, refinements of CountActions performs very well on a number of sample planning domains.

An exercise: blocks world planning

There are a collection of blocks: a block can be on the table, or on top of another block. There are three predicates:

$clear(x)$: there is no block on top of block x;

$on(x, y)$: block x is on top of block y; and

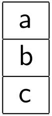
$onTable(x)$: block x is on the table.

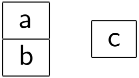
There are three actions:

$move(x, y, z)$: move block x from block y onto block z, provided x is on y, both x and z are clear;

$moveFromTable(x, y)$: move block x from the table onto block y, provided x is on the table, both x and y are clear; and

$moveToTable(x, y)$: move block x from block y onto the table, provided x is on y, and x is clear.

The initial state is: 

The goal state is: 

An exercise: blocks world planning

Write the STRIPS representation of the actions, the initial KB, and the goal.

Use reachability analysis to compute the heuristic value for the initial state. Draw the state and action layers. For each call of CountActions, indicate the values of G, G_P, G_N, and A.