Planning

Russell and Norvig: Chapter 11
CMSC421 – Fall 2005
Planning Agent

sensors

agent

actuators

environment
Planning Agent

sensors

agent

actuators

environment

A1  A2  A3
Planning problem

Find a **sequence of actions** that achieves a given **goal** when executed from a given **initial world state**. That is, given
- a set of operator descriptions (defining the possible primitive actions by the agent),
- an initial state description, and
- a goal state description or predicate,
compute a plan, which is
- a sequence of operator instances, such that executing them in the initial state will change the world to a state satisfying the goal-state description.

Goals are usually specified as a conjunction of subgoals to be achieved
Planning vs. problem solving

- Planning and problem solving methods can often solve the same sorts of problems.
- Planning is more powerful because of the representations and methods used.
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic).
- Search often proceeds through plan space rather than state space (though there are also state-space planners).
- Subgoals can be planned independently, reducing the complexity of the planning problem.
Goal of Planning
Goal of Planning

Choose actions to achieve a certain goal
Goal of Planning

- Choose actions to achieve a certain goal
- But isn’t it exactly the same goal as for problem solving?
Goal of Planning

- Choose actions to achieve a certain goal
- But isn’t it exactly the same goal as for problem solving?

Some difficulties with problem solving:

- The successor function is a **black box**: it must be “applied” to a state to know which actions are possible in that state and what are the effects of each one
Choose actions to achieve a certain goal

But isn't it exactly the same goal as for problem solving?

Some difficulties with problem solving:

- The successor function is a black box: it must be "applied" to a state to know which actions are possible in that state and what are the effects of each one.

Goal of Planning

- Suppose that the goal is HAVE(MILK).
Goal of Planning

- Suppose that the goal is HAVE(MILK).
- From some initial state where HAVE(MILK) is not satisfied, the successor function must be repeatedly applied to eventually generate a state where HAVE(MILK) is satisfied.
Goal of Planning

- Suppose that the goal is HAVE(MILK).
- From some initial state where HAVE(MILK) is not satisfied, the successor function must be repeatedly applied to eventually generate a state where HAVE(MILK) is satisfied.
- An explicit representation of the possible actions and their effects would help the problem solver select the relevant actions are possible in that state and what are the effects of each one.
Goal of Planning

• Suppose that the goal is HAVE(MILK).
• From some initial state where HAVE(MILK) is not satisfied, the successor function must be repeatedly applied to eventually generate a state where HAVE(MILK) is satisfied.
• An explicit representation of the possible actions and their effects would help the problem solver select the relevant actions otherwise, in the real world an agent would be overwhelmed by irrelevant actions.
Goal of Planning

Choose actions to achieve a certain goal

But isn’t it exactly the same goal as for problem solving?

Some difficulties with problem solving:

- The goal test is another black-box function, states are domain-specific data structures, and heuristics must be supplied for each new problem
Goal of Planning

Suppose that the goal is \( \text{HAVE} (\text{MILK}) \land \text{HAVE} (\text{BOOK}) \)

Without an explicit representation of the goal, the problem solver cannot know that a state where \( \text{HAVE} (\text{MILK}) \) is already achieved is more promising than a state where neither \( \text{HAVE} (\text{MILK}) \) nor \( \text{HAVE} (\text{BOOK}) \) is achieved.

States are domain-specific data structures, and heuristics must be supplied for each new problem.
Goal of Planning

- Choose actions to achieve a certain goal
- But isn’t it exactly the same goal as for problem solving?
- Some difficulties with problem solving:
  - The goal may consist of several nearly independent subgoals, but there is no way for the problem solver to know it
Goal of Planning

- Choose actions to achieve a certain goal

- Example:

  HAVE(MILK) and HAVE(BOOK) may be achieved by two nearly independent sequences of actions

Some difficulties with problem solving:

- The goal may consist of several nearly independent subgoals, but there is no way for the problem solver to know it.
Representations in Planning

Planning opens up the black-boxes by using logic to represent:

- Actions
- States
- Goals
Representations in Planning

Planning opens up the black-boxes by using logic to represent:

- Actions
- States
- Goals
Major approaches

- Situation calculus
- State space planning
- Partial order planning
- Planning graphs
- Hierarchical decomposition (HTN planning)
- Reactive planning
Major approaches

Planning rapidly changing subfield of AI

In biannual competition at AI Planning Systems Conference:
- four years ago, best planner did plan space search using SAT solver
- three years ago, the best planner did regression search
- last year, best planner did forward state space search with an inadmissable heuristic function

Hierarchical decomposition (HTN planning)

Reactive planning
Major approaches

Planning rapidly changing subfield of AI

In biannual competition at AI Planning Systems Conference:
- four years ago, best planner did plan space search using SAT solver
- three years ago, the best planner did regression search
- last year, best planner did forward state space search with an inadmissible heuristic function

Hierarchical decomposition (HTN planning)

cmsc722: Planning, taught by Prof. Nau
Situation Calculus Planning

- Formulate planning problem in FOL
- Use theorem prover to find proof (aka plan)
Representing change

- Representing change in the world in logic can be tricky.
- One way is just to change the KB
  - Add and delete sentences from the KB to reflect changes
  - How do we remember the past, or reason about changes?
- **Situation calculus** is another way
- A **situation** is a snapshot of the world at some instant in time
- When the agent performs an action A in situation S1, the result is a new situation S2.
Situations
Situation calculus

- A **situation** is a snapshot of the world at an interval of time during which nothing changes.
- Every true or false statement is made with respect to a particular situation.
  - Add **situation variables** to every predicate.
  - at(hunter,1,1) becomes at(hunter,1,1,s0): at(hunter,1,1) is true in situation (i.e., state) s0.
- Add a new function, **result(a,s)**, that maps a situation s into a new situation as a result of performing action a. For example, result(forward, s) is a function that returns the successor state (situation) to s.
- Example: The action agent-walks-to-location-y could be represented by
  - $(\forall x)(\forall y)(\forall s) \ (at(Agent,x,s) \land \neg onbox(s)) \rightarrow at(Agent,y,result(walk(y),s)))$
Situation calculus planning
Situation calculus planning

**Initial state:** a logical sentence about (situation) $S_0$

$$\text{At(Home, } S_0) \land \sim\text{Have(Milk, } S_0) \land \sim\text{Have(Bananas, } S_0) \land \sim\text{Have(Drill, } S_0)$$
Situation calculus planning

- **Initial state:** a logical sentence about (situation) $S_0$
  \[ \text{At(Home, } S_0 \text{)} ^ \wedge \sim \text{Have(Milk, } S_0 \text{)} ^ \wedge \sim \text{Have(Bananas, } S_0 \text{)} ^ \wedge \sim \text{Have(Drill, } S_0 \text{)} \]

- **Goal state:**
  \[ (\exists s) \text{At(Home,} s\text{)} ^ \wedge \text{Have(Milk,} s\text{)} ^ \wedge \text{Have(Bananas,} s\text{)} ^ \wedge \text{Have(Drill,} s\text{)} \]
Situation calculus planning

**Initial state:** a logical sentence about (situation) $S_0$

\[
\text{At(Home, } S_0) \land \neg\text{Have(Milk, } S_0) \land \neg\text{Have(Bananas, } S_0) \land \neg\text{Have(Drill, } S_0)\\
\]

**Goal state:**

\[
(\exists s) \text{At(Home,} s) \land \text{Have(Milk,} s) \land \text{Have(Bananas,} s) \land \text{Have(Drill,} s)\\
\]

**Operators** are descriptions of actions:

\[
\forall (a,s) \text{Have(Milk,Result(a,s)) } \leftrightarrow ((a=\text{Buy(Milk)} \land \text{At(Grocery,} s)) \lor (\text{Have(Milk,} s) \land a\sim=\text{Drop(Milk))})\\
\]
Situation calculus planning

**Initial state:** a logical sentence about (situation) $S_0$

$\text{At(Home, } S_0 \text{)} \land \neg \text{Have(Milk, } S_0 \text{)} \land \neg \text{Have(Bananas, } S_0 \text{)} \land \neg \text{Have(Drill, } S_0 \text{)}$

**Goal state:**

$(\exists s) \text{At(Home, } s \text{)} \land \text{Have(Milk, } s \text{)} \land \text{Have(Bananas, } s \text{)} \land \text{Have(Drill, } s \text{)}$

**Operators** are descriptions of actions:

$\forall (a, s) \text{ Have(Milk, Result(a, s)) } \iff ((a=\text{Buy(Milk)} \land \text{At(Grocery, } s \text{)}) \lor (\text{Have(Milk, } s \land a\sim=\text{Drop(Milk)})))$

**Result(a, s)** names the situation resulting from executing action $a$ in situation $s$. 
Situation calculus planning

**Initial state:** a logical sentence about (situation) \( S_0 \)

\[
\text{At(Home, } S_0 \text{)} ^ \land \neg \text{Have(Milk, } S_0 \text{)} ^ \land \neg \text{Have(Bananas, } S_0 \text{)} ^ \land \neg \text{Have(Drill, } S_0 \text{)}
\]

**Goal state:**

\[
(\exists s) \text{At(Home, } s \text{)} ^ \land \text{Have(Milk, } s \text{)} ^ \land \text{Have(Bananas, } s \text{)} ^ \land \text{Have(Drill, } s \text{)}
\]

**Operators** are descriptions of actions:

\[
\forall (a, s) \text{ Have(Milk, Result(a, s)) } \iff ((a=\text{Buy(Milk)} ^ \land \text{At(Grocery, } s \text{)}) v (\text{Have(Milk, } s \text{)} ^ \land a\sim=\text{Drop(Milk)}))
\]

**Result(a,s)** names the situation resulting from executing action \( a \) in situation \( s \).

**Action sequences are also useful:** \( \text{Result'}(l,s) \) is the result of executing the list of actions \( l \) starting in \( s \):

\[
(\forall s) \text{ Result'}([], s) = s
\]

\[
(\forall a, p, s) \text{ Result'}([a|p]s) = \text{Result'}(p, \text{Result'}(a, s))
\]
Situation calculus planning II
Situation calculus planning II

A solution is thus a plan that when applied to the initial state yields a situation satisfying the goal query:

\[\text{At(Home,Result'(p,S_0))} \land \text{Have(Milk,Result'(p,S_0))} \land \text{Have(Bananas,Result'(p,S_0))} \land \text{Have(Drill,Result'(p,S_0))}\]
A solution is thus a plan that when applied to the initial state yields a situation satisfying the goal query:

\[
\text{At(Home,Result}'(p,S_0)) \\
\wedge \text{Have(Milk,Result}'(p,S_0)) \\
\wedge \text{Have(Bananas,Result}'(p,S_0)) \\
\wedge \text{Have(Drill,Result}'(p,S_0))
\]

Thus we would expect a plan (i.e., variable assignment through unification) such as:

\[p = [\text{Go(Grocery)}, \text{Buy(Milk)}, \text{Buy(Bananas)}, \text{Go(HardwareStore)}, \text{Buy(Drill)}, \text{Go(Home)}]\]
SC planning: analysis

- This is fine in theory, but remember that problem solving (search) is exponential in the worst case.
- Also, resolution theorem proving only finds a proof (plan), not necessarily a good plan.
- Another important issue: the Frame Problem.
The Frame Problem
The Frame Problem

In SC, need not only axioms to describe what changes in each situation, but also need axioms to describe what stays the same (can do this using successor-state axioms)
The Frame Problem

- In SC, need not only axioms to describe what changes in each situation, but also need axioms to describe what stays the same (can do this using successor-state axioms)

- Qualification problem: difficulty in specifying all the conditions that must hold in order for an action to work
The Frame Problem

In SC, need not only axioms to describe what changes in each situation, but also need axioms to describe what stays the same (can do this using successor-state axioms)

Qualification problem: difficulty in specifying all the conditions that must hold in order for an action to work

Ramification problem: difficulty in specifying all of the effects that will hold after an action is taken
So...

we restrict the language and use a special-purpose algorithm (a planner) rather than general theorem prover
Basic representations for planning
Basic representations for planning

Classic approach first used in STRIPS planner circa 1970
Basic representations for planning

- Classic approach first used in STRIPS planner circa 1970
- States represented as a conjunction of ground literals
  - at(Home)
Basic representations for planning

- Classic approach first used in STRIPS planner circa 1970

- States represented as a conjunction of ground literals
  - \( \text{at(Home)} \)

- Goals are conjunctions of literals, but may have existentially quantified variables
  - \( \text{at(?x)} \land \text{have(Milk)} \land \text{have(bananas)} \) ...
Basic representations for planning

- Classic approach first used in STRIPS planner circa 1970

- States represented as a conjunction of ground literals
  - at(Home)

- Goals are conjunctions of literals, but may have existentially quantified variables
  - at(?x) ^ have(Milk) ^ have(bananas) ...

- Do not need to fully specify state
  - Non-specified either don’t-care or assumed false
  - Represent many cases in small storage
  - Often only represent changes in state rather than entire situation
Basic representations for planning

- Classic approach first used in **STRIPS** planner circa 1970
- States represented as a conjunction of ground literals
  - at(Home)
- Goals are conjunctions of literals, but may have existentially quantified variables
  - at(?x) ^ have(Milk) ^ have(bananas) ...
- Do not need to fully specify state
  - Non-specified either don’t-care or assumed false
  - Represent many cases in small storage
  - Often only represent changes in state rather than entire situation
- Unlike theorem prover, not seeking whether the goal is true, but is there a sequence of actions to attain it
Operator/action representation

Operators contain three components:

- **Action description**
- **Precondition** - conjunction of positive literals
- **Effect** - conjunction of positive or negative literals which describe how situation changes when operator is applied

Example:

\[
\text{Op[Action: Go(there), Precond: At(here) \land Path(here,there), Effect: At(there) \land \neg\text{At(here)}]}
\]

All variables are universally quantified

Situation variables are implicit

- preconditions must be true in the state immediately before operator is applied; effects are true immediately after
Blocks world

The **blocks world** is a micro-world that consists of a table, a set of blocks and a robot hand.

Some domain constraints:
- Only one block can be on another block
- Any number of blocks can be on the table
- The hand can only hold one block

Typical representation:

```
ontable(a)
ontable(c)
on(b,a)
handempty
clear(b)
clear(c)
```
State Representation
State Representation

Conjunction of propositions:
BLOCK(A), BLOCK(B), BLOCK(C),
ON(A,TABLE), ON(B,TABLE), ON(C,A),
CLEAR(B), CLEAR(C), HANDEMPY
Goal Representation

Conjunction of propositions:
ON(A,TABLE), ON(B,A), ON(C,B)
Goal Representation

Conjunction of propositions:
ON(A, TABLE), ON(B, A), ON(C, B)

The goal G is achieved in a state S if all the propositions in G are also in S
Action Representation
Action Representation

Unstack(x,y)

Precondition: conjunction of propositions
Action Representation

Unstack(x,y)

- \( P = \) HANDEMP, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)

Precondition: conjunction of propositions
Action Representation

Unstack(x,y)

P = HANDEMPMY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)

E = ¬HANDEMPMY, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

Effect: list of literals
Precondition: conjunction of propositions
**Action Representation**

Unstack(x,y)

- **P =** HANDEMPNY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- **E =** ¬HANDEMPNY, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

**Effect:** list of literals

**Precondition:** conjunction of propositions

**Means:** Remove HANDEMPNY from state
**Action Representation**

Unstack(x,y)

- **P =** HANDEMPNT, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- **E =** ¬HANDEMPNT, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

**Precondition:** conjunction of propositions

**Effect:** list of literals

Means: Remove HANDEMPNT from state

Means: Add HOLDING(x) to state

Sunday, February 28, 2010
**Example**

**Unstack(C,A)**

- **P** = HANDEMPNTY, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- **E** = \(\neg\)HANDEMPNTY, \(\neg\)CLEAR(C), HOLDING(C), \(\neg\)ON(C,A), CLEAR(A)
Example

Unstack(C,A)
- P = HANDEmpty, BLOCK(C), BLOCK(A), CLEAR(C), ON(C,A)
- E = ¬HANDEmpty, ¬CLEAR(C), HOLDING(C), ¬ON(C,A), CLEAR(A)
Example

Unstack(C,A)
- $P = \text{HANDEMTY}, \text{BLOCK}(C), \text{BLOCK}(A), \text{CLEAR}(C), \text{ON}(C,A)$
- $E = \neg\text{HANDEMTY}, \neg\text{CLEAR}(C), \text{HOLDING}(C), \neg\text{ON}(C,A), \text{CLEAR}(A)$
Action Representation
Action Representation

Unstack(x,y)

- $P = \text{HANDEMTY}, \text{BLOCK}(x), \text{BLOCK}(y), \text{CLEAR}(x), \text{ON}(x,y)$
- $E = \neg \text{HANDEMTY}, \neg \text{CLEAR}(x), \text{HOLDING}(x), \neg \text{ON}(x,y), \text{CLEAR}(y)$
**Action Representation**

Unstack\((x,y)\)
- \(P = \text{HANDEMPY, BLOCK}(x), \text{BLOCK}(y), \text{CLEAR}(x), \text{ON}(x,y)\)
- \(E = \neg\text{HANDEMPY}, \neg\text{CLEAR}(x), \text{HOLDING}(x), \neg\text{ON}(x,y), \text{CLEAR}(y)\)

Stack\((x,y)\)
- \(P = \text{HOLDING}(x), \text{BLOCK}(x), \text{BLOCK}(y), \text{CLEAR}(y)\)
- \(E = \text{ON}(x,y), \neg\text{CLEAR}(y), \neg\text{HOLDING}(x), \text{CLEAR}(x), \text{HANDEMPY}\)
Action Representation

Unstack(x,y)
- P = HANDEMPTEY, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
- E = ¬HANDEMPTEY, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

Stack(x,y)
- P = HOLDING(x), BLOCK(x), BLOCK(y), CLEAR(y)
- E = ON(x,y), ¬CLEAR(y), ¬HOLDING(x), CLEAR(x), HANDEMPTEY

Pickup(x)
- P = HANDEMPTEY, BLOCK(x), CLEAR(x), ON(x, TABLE)
- E = ¬HANDEMPTEY, ¬CLEAR(x), HOLDING(x), ¬ON(x, TABLE)
Action Representation

Unstack(x,y)
• P = HANDEMPYT, BLOCK(x), BLOCK(y), CLEAR(x), ON(x,y)
• E = ¬HANDEMPYT, ¬CLEAR(x), HOLDING(x), ¬ON(x,y), CLEAR(y)

Stack(x,y)
• P = HOLDING(x), BLOCK(x), BLOCK(y), CLEAR(y)
• E = ON(x,y), ¬CLEAR(y), ¬HOLDING(x), CLEAR(x), HANDEMPYT

Pickup(x)
• P = HANDEMPYT, BLOCK(x), CLEAR(x), ON(x,TABLE)
• E = ¬HANDEMPYT, ¬CLEAR(x), HOLDING(x), ¬ON(x,TABLE)

PutDown(x)
• P = HOLDING(x)
• E = ON(x,TABLE), ¬HOLDING(x), CLEAR(x), HANDEMPYT
Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(b,c)
- on(a,b)

A plan:
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)
Another BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(a,b)
- on(b,c)

A plan:
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)
Simple planning algorithms assume that the goals to be achieved are independent:

- Each can be solved separately and then the solutions concatenated.
Goal interaction

Simple planning algorithms assume that the goals to be achieved are independent:
- Each can be solved separately and then the solutions concatenated.

This planning problem, called the “Sussman Anomaly,” is the classic example of the goal interaction problem:
- Solving on(A,B) first (by doing unstack(C,A), stack(A,B) will be undone when solving the second goal on(B,C) (by doing unstack(A,B), stack(B,C)).
- Solving on(B,C) first will be undone when solving on(A,B).

![Initial state](image1)

![Goal state](image2)
Goal interaction

Simple planning algorithms assume that the goals to be achieved are independent:
- Each can be solved separately and then the solutions concatenated.

This planning problem, called the "Sussman Anomaly," is the classic example of the goal interaction problem:
- Solving on(A,B) first (by doing unstack(C,A), stack(A,B) will be undone when solving the second goal on(B,C) (by doing unstack(A,B), stack(B,C)).
- Solving on(B,C) first will be undone when solving on(A,B).

Classic STRIPS could not handle this, although minor modifications can get it to do simple cases.