CMSC 471
Fall 2012

Class #19

Thursday, November 1, 2012
Planning

Kevin Winner, winnerk1@umbc.edu

Planning

Chapter 10.1-10.2, 10.4.2-10.4.4

Some material adopted from notes
by Andreas Geyer-Schulz
and Chuck Dyer

Today's Class

- What is planning?
- Approaches to planning
  - GPS / STRIPS
  - Situation calculus formalism [revisited]
  - Partial-order planning

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 goals 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

Typical Assumptions

- **Atomic time**: Each action is indivisible
- **No concurrent actions** are allowed (though actions do not need to be ordered with respect to each other in the plan)
- **Deterministic actions**: The result of actions are completely determined—there is no uncertainty in their effects
- Agent is the **sole cause of change** in the world
- Agent is **omniscient**: Has complete knowledge of the state of the world
- **Closed world assumption**: everything known to be true in the world is included in the state description. Anything not listed is false.

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)

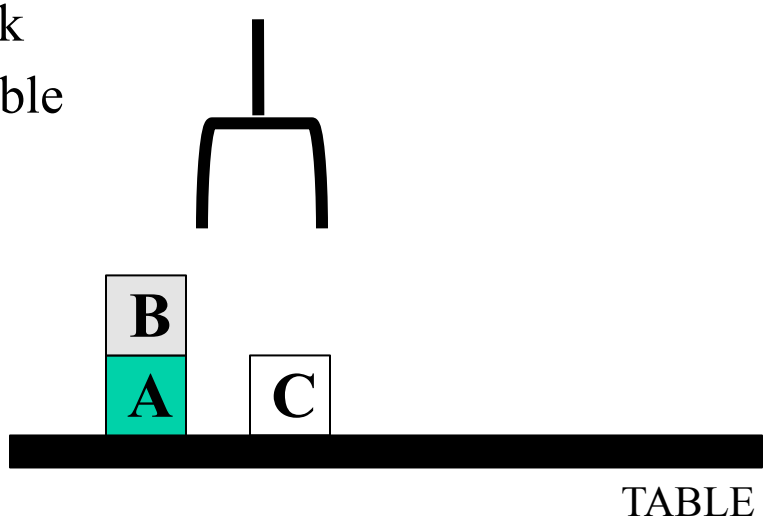
ontable(c)

on(b,a)

handempty

clear(b)

clear(c)



Major Approaches

- GPS / STRIPS
- **Situation calculus**
- **Classical planning**
- **Partial order planning**

General Problem Solver

- The General Problem Solver (GPS) system was an early planner (Newell, Shaw, and Simon)
- GPS generated actions that reduced the difference between some state and a goal state
- GPS used Means-Ends Analysis
  - Compare what is given or known with what is desired and select a reasonable thing to do next
  - Use a table of differences to identify procedures to reduce types of differences
- GPS was a state space planner: it operated in the domain of state space problems specified by an initial state, some goal states, and a set of operations

Situation Calculus Planning

- Intuition: Represent the planning problem using first-order logic
  - Situation calculus lets us reason about changes in the world
  - Use theorem proving to "prove" that a particular sequence of actions, when applied to the situation characterizing the world state, will lead to a desired result

Situation Calculus Planning, cont.

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

  $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg Have(Bananas, S_0) \wedge \neg Have(Drill, S_0)$

- **Goal state**:

  $(\exists s)\ At(Home,s) \wedge Have(Milk,s) \wedge Have(Bananas,s) \wedge Have(Drill,s)$

- **Operators** are descriptions of how the world changes as a result of the agent's actions:

  $\forall(a,s)\ Have(Milk,Result(a,s)) \Leftrightarrow$
  $((a=Buy(Milk) \wedge At(Grocery,s)) \vee (Have(Milk, s) \wedge a \neq Drop(Milk)))$

- Result(a,s) names the situation resulting from executing action a in situation s.
- Action sequences are also useful: Result'(l,s) is the result of executing the list of actions (l) starting in s:

  $(\forall s)\ Result'([],s) = s$

  $(\forall a,p,s)\ Result'([a|p]s) = Result'(p,Result(a,s))$

Situation Calculus, cont.

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

  $At(Home, Result'(p,S_0))$

  $\wedge$ $Have(Milk, Result'(p,S_0))$

  $\wedge$ $Have(Bananas, Result'(p,S_0))$

  $\wedge$ $Have(Drill, Result'(p,S_0))$

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

  p = [Go(Grocery), Buy(Milk), Buy(Bananas), Go(HardwareStore), Buy(Drill), Go(Home)]

Situation Calculus: Blocks World

- Here's an example of a situation calculus rule for the blocks world:
  - Clear (X, Result(A,S)) ↔
    [Clear (X, S) ∧
    (¬(A=Stack(Y,X) ∨ A=Pickup(X))
    ∨ (A=Stack(Y,X) ∧ ¬(holding(Y,S))
    ∨ (A=Pickup(X) ∧ ¬(handempty(S) ∧ ontable(X,S) ∧ clear(X,S))))]
    ∨ [A=Stack(X,Y) ∧ holding(X,S) ∧ clear(Y,S)]
    ∨ [A=Unstack(Y,X) ∧ on(Y,X,S) ∧ clear(Y,S) ∧ handempty(S)]
    ∨ [A=Putdown(X) ∧ holding(X,S)]
- English translation: A block is clear if (a) in the previous state it was clear and we didn't pick it up or stack something on it successfully, or (b) we stacked it on something else successfully, or (c) something was on it that we unstacked successfully, or (d) we were holding it and we put it down.
- Whew!!! There's gotta be a better way!

Situation Calculus 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
- So we restrict the language and use a special-purpose algorithm (a planner) rather than general theorem prover

Basic Representations for Planning

- Classic approach first used in the STRIPS planner circa 1970
- States represented as a conjunction of ground literals
  - at(Home) ∧ ¬have(Milk) ∧ ¬have(bananas) ...
- Goals are conjunctions of literals, but may have variables which are assumed to be existentially quantified
  - 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

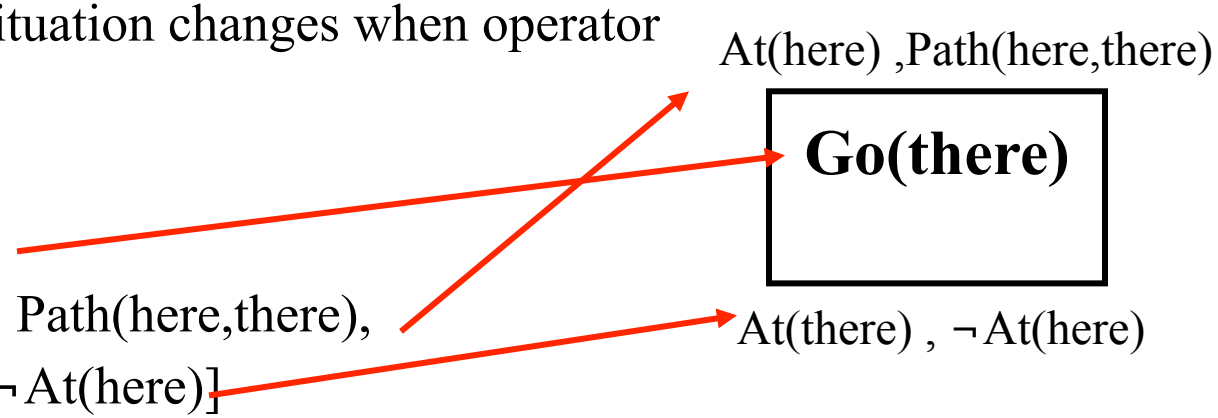
Operator/Action Representation

- 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:

  Op[Action: Go(there),

  Precond: At(here) ∧ Path(here,there),

  Effect: At(there) ∧ ¬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 Operators

- Here are the classic basic operations for the blocks world:
  - stack(X,Y): put block X on block Y
  - unstack(X,Y): remove block X from block Y
  - pickup(X): pickup block X
  - putdown(X): put block X on the table
- Each will be represented by
  - a list of preconditions
  - a list of new facts to be added (add-effects)
  - a list of facts to be removed (delete-effects)
  - optionally, a set of (simple) variable constraints
- For example:

  preconditions(stack(X,Y), [holding(X), clear(Y)])
  deletes(stack(X,Y), [holding(X), clear(Y)]).
  adds(stack(X,Y), [handempty, on(X,Y), clear(X)])
  constraints(stack(X,Y), [X≠Y, Y≠table, X≠table])

Blocks World Operators II

operator(stack(X,Y),
Precond [holding(X), clear(Y)],
Add [handempty, on(X,Y), clear(X)],
Delete [holding(X), clear(Y)],
Constr [X≠Y, Y≠table, X≠table]).

operator(unstack(X,Y),
[on(X,Y), clear(X), handempty],
[holding(X), clear(Y)],
[handempty, clear(X), on(X,Y)],
[X≠Y, Y≠table, X≠table]).

operator(pickup(X),
[ontable(X), clear(X), handempty],
[holding(X)],
[ontable(X), clear(X), handempty],
[X≠table]).

operator(putdown(X),
[holding(X)],
[ontable(X), handempty, clear(X)],
[holding(X)],
[X≠table]).

STRIPS Planning

- STRIPS maintains two additional data structures:
  - **State List** - all currently true predicates.
  - **Goal Stack** - a push down stack of goals to be solved, with current goal on top of stack.
- If current goal is not satisfied by present state, examine add lists of operators, and push operator and preconditions list on stack. (Subgoals)
- When a current goal is satisfied, POP it from stack.
- When an operator is on top stack, record the application of that operator on the plan sequence and use the operator's add and delete lists to update the current state.

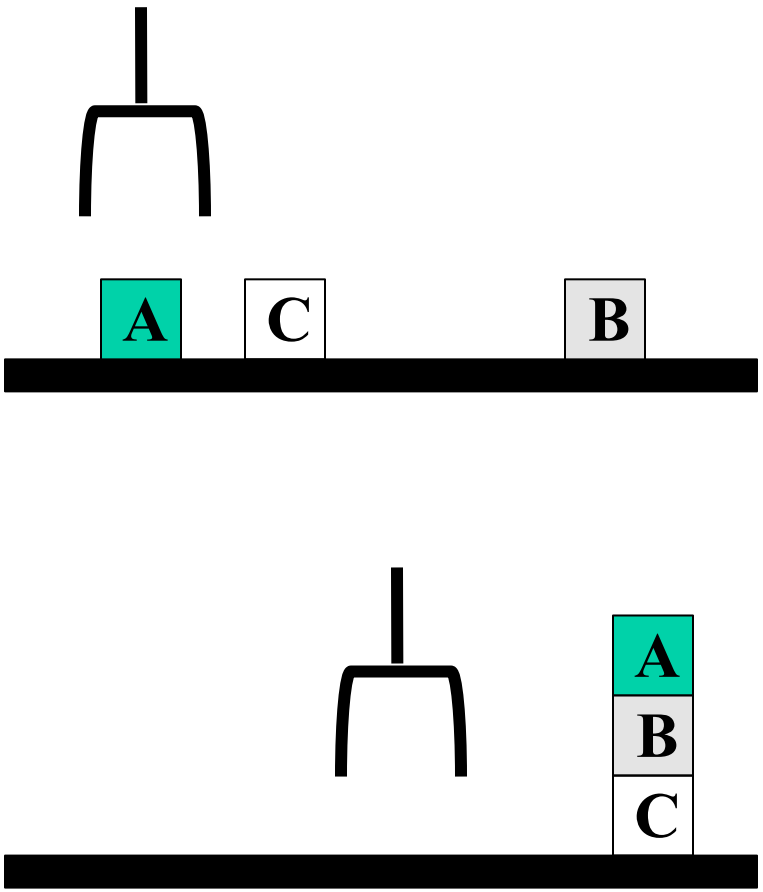
Typical Blocks World Planning Problem

Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(b,c)
- on(a,b)
- ontable(c)



A plan:

- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

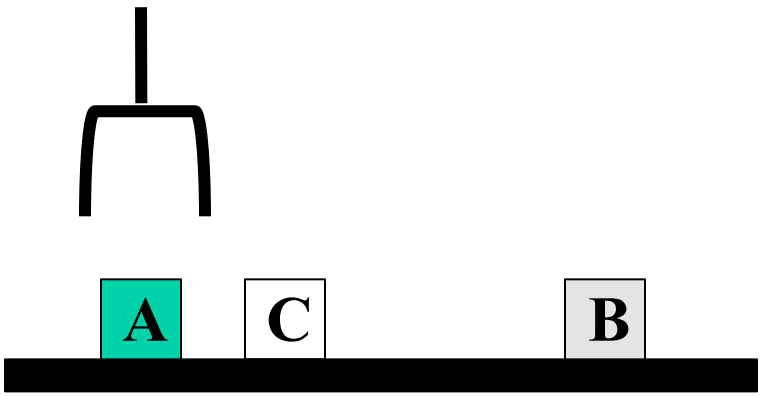
Another Blocks World Planning Problem

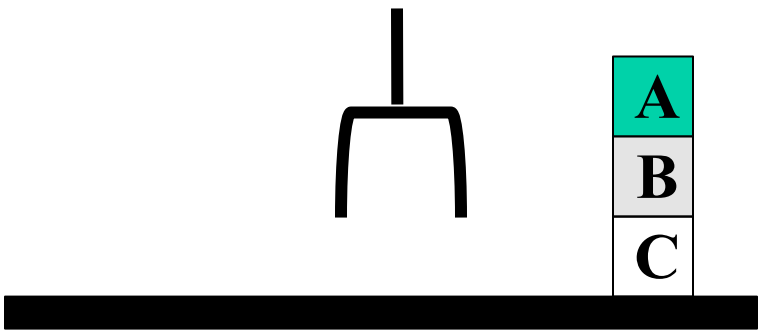
Initial state:

- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:

- on(a,b)
- on(b,c)
- ontable(c)



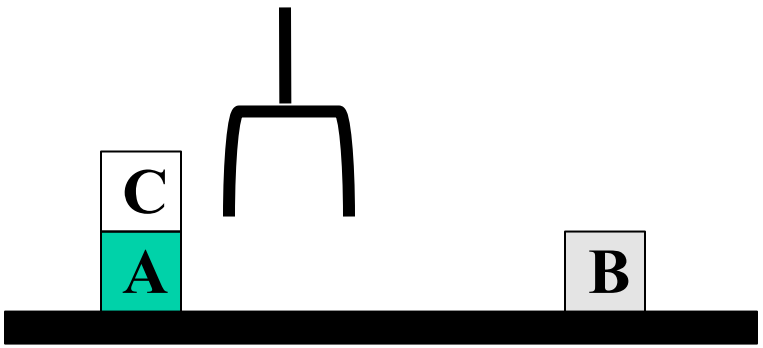


A plan:

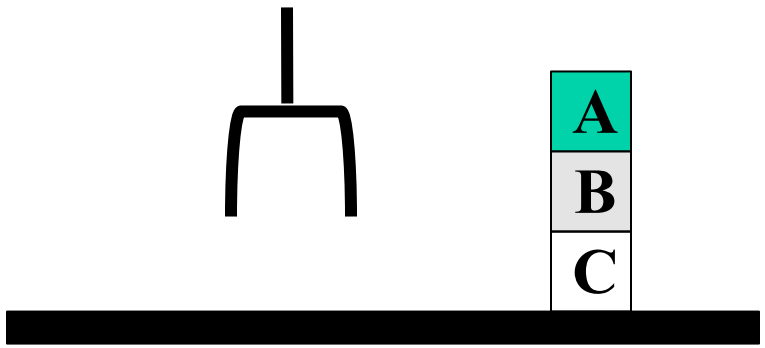
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

Goal Interactions

- 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



Initial state



Goal state