EECS 391: Introduction to AI

Soumya Ray

Website: http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tue 2:30-4:00 or by appointment

Announcements

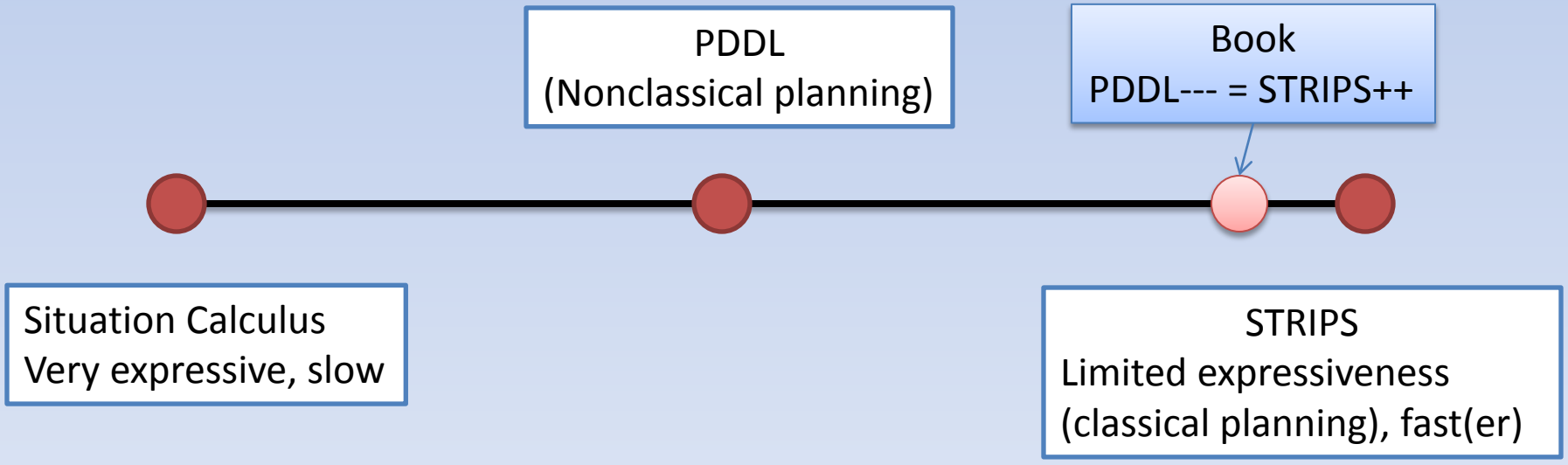
- PA3 online today
- Feng's office hours this week F 2:30-4pm

Today

- Automated Planning (Chapter 10.1-2, 10.4.2, 10.4.4)

Representing a Planning Problem

- For classical planning, one fragment of FOL that is used is called STRIPS ("Stanford Research Institute Problem Solver")



Representing States in STRIPS

- States in STRIPS are conjunctions of unnegated, ground, function-free literals
  - All conditions that hold in that state
  - *Block(A)*, *Block(B)*, *On(A,B)*, *On(B, Table)*, *GripperEmpty*
  - The “Closed World Assumption” is used

Representing Goals in STRIPS

- Goals are conjunctions of unnegated function-free ground literals
- Goals may not fully determine a state of the world
  - In this case, the goal is any state where these literals hold
- Example: $On(A,E) \wedge On(B,D)$

Representing Actions in STRIPS

- Want to represent an action of picking up a block from the table
- *Pickup_from_Table(x)*
- Preconditions: *Block(x), GripperEmpty, Clear(x), On(x,Table)*
- Add List: *Holding(x)*
- Delete List: ¬*GripperEmpty*, ¬*On(x,Table)*

Applicability

- An action is applicable in all states where its preconditions are satisfied
  - $s \models \text{precond}(A)$?
  - If an action is not applicable, it is not used
  - For first-order action schemas, this test involves unification

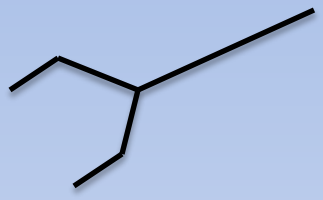
The STRIPS assumption

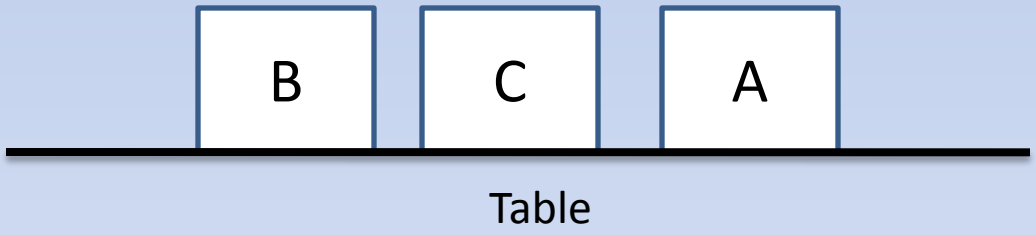
- Every possible effect of actions are listed
  - i.e., if a literal does not appear in the effects list, it is unchanged in the resulting state
  - Solves the "frame problem" in situation calculus

Restrictions in STRIPS

- States are described by unnegated ground function-free literals
- CWA
- Ground conjunctive goals
- Conjunctive effects
- No equality

Example: Blocks World





Example

- Init($On(A,Table) \wedge On(B,Table) \wedge On(C,Table) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(A) \wedge Clear(B) \wedge Clear(C)$)
- Goal($On(A,B)$)
- Action($MoveToTable(b,x)$,
  - Preconditions($On(b,x) \wedge Clear(b) \wedge Block(b) \wedge Block(x)$)
  - Effects($On(b,Table) \wedge Clear(x) \wedge \neg On(b,x)$)

Planning Algorithms

- Given a STRIPS representation of a classical planning problem, how do we solve it?
  - Since the world is static, deterministic, fully observable, we could use search
  - Remember that in this case, the search algorithm is actually performing logical inference

Kinds of Search

- Search algorithms for classical planning fall into two categories
  - "State space planners": States of the search problem are states of the world; search operators are actions of the world
  - "Plan space planners": States of the search problem are partial plans; search operators are modifications to the current partial plan

Forward State-Space Search

- "Progression" planning

- Setup:
  - States=world states
  - Initial state=given
  - Operators=*applicable* actions
  - Goal test=given
  - Operator costs=given

Forward State Space Search

- We could apply any search algorithm, e.g. A*

- The key differences are:
  - Only applicable actions need to be explored
  - Getting the next state is done through the STRIPS specification of states and actions
  - Heuristics are based on planning ideas

Backward State-Space Search

- Forward search has trouble with irrelevant actions
  - E.g., suppose there are 50 blocks on the table and I just want A on top of B
- This can be solved by backward search ("regression planning")
  - But this requires "inverting" the search operators (i.e. the actions)
  - Fortunately this is easy to do for STRIPS

Search Heuristics

- Typically, heuristics are used with state space search to make planners more efficient
- From any state, want to estimate the number of actions to search termination admissibly
- Two possibilities:
  - Relax the planning problem
  - Consider subproblems

Relaxed Plans

- There are different ways to arrive at a less constrained planning problem

- One way is to remove all DELETE effects from STRIPS actions
  - This is admissible (why?)
  - To estimate this cost, need to run an internal planning loop; but this is usually very fast

Subproblems

- The goal is a conjunction of literals
- We can generate subproblems by just considering a single literal at a time
  - "Subgoal Independence" (admissible)
- Combine with max, as usual

Total Order Plans

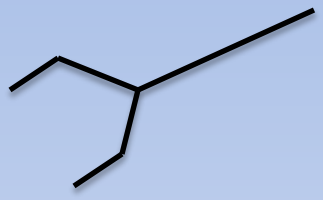
- In a Total Order plan, every pair of actions A_1 and A_2 has a *temporal ordering constraint*
  - Either A_1 is done first, or A_2
  - Forward and Backward state space planners are like this

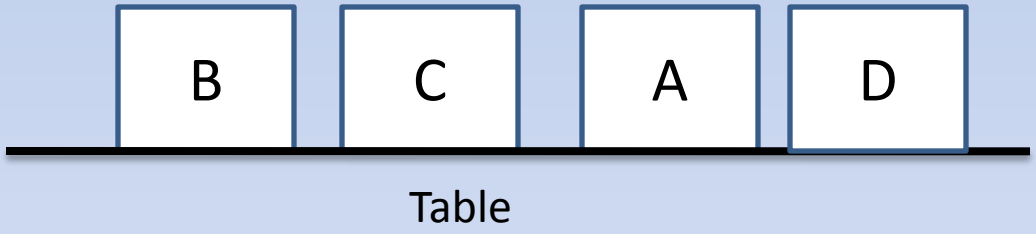
Partial Order Plans

- In many situations, actions do not have to be done in order
  - Might be trying to achieve unrelated things

- This is a *partial order* plan: a plan with some actions that have no temporal ordering constraints between them
  - i.e. there is some A_1, A_2 so that A_1 does not have to be completed before A_2 and A_2 does not have to be completed before A_1 for the plan to succeed

Blocks World





Goal: $On(A,B) \wedge On(C,D)$

A dummy action with no preconditions and effect==initial state

Start

Move(A,Table,B)

Move(C,Table,D)

End

A dummy action with no effects and preconditions==goal

Partial Order Plans

- Represented as a set of actions and ordering constraints ($A < B$)
- A total order plan derived from a POP is called a linearization of the POP
- Partial Order plans have two advantages over total order plans
  - Flexibility when executing the plan
  - Action Parallelism

Executing a POP Sequentially

- If we have a solution POP, any linearization of it is a solution too

- To execute a POP sequentially , we execute some linearization of it
  - At each “choice point”, choose any available action

Finding a POP

- To find POPs, we will perform a search
- The states of the search space will be partial solutions, augmented with some bookkeeping information
- Starting with an empty POP (only “Start” and “End”), we will add actions and ordering constraints until we have a valid POP

States of the Search Space

- A state will have
  - The partial POP (list of actions and ordering constraints)
    - Ordering constraints can't introduce cycles
  - A list of open conditions (initially the goal)
    - Open condition: A literal that is not the effect of some action in the plan
    - The termination states of the search are those where this list is empty
  - A list of "causal links"

Causal Links

- A causal link $A \xrightarrow{p} B$ indicates that action A achieves literal p for action B
  - B requires p as a precondition
  - A has p as an effect

- An action C *conflicts with, or threatens, a causal link* $A \xrightarrow{p} B$ if C has $\neg p$ as an effect and could come after A and before B

Consistent Plans

- A *consistent plan* is a POP where the ordering constraints have no cycles and there are no conflicts with causal links

- So a solution is a consistent plan with no open conditions

POP Algorithm

- Initial State: Plan=$(Start, End, Start < End)$, open conditions=preconditions of End (goal), no causal links

- Search operators: Pick an open condition, and an action to satisfy it; generate next state (POP)

- Goal test: empty list of open conditions

Generating the Next State

- Suppose we pick condition p on action B to satisfy, using action A
  - A might be an action already in the plan
- To generate the next state:
  - Remove p from list of conditions, add A to list of actions, add A's preconditions to list of conditions
  - Add a causal link $A \xrightarrow{p} B$ and ordering constraint $A < B$
  - Add $Start < A$ and $A < End$
  - Resolve conflicts/threats if any

Conflict Resolution

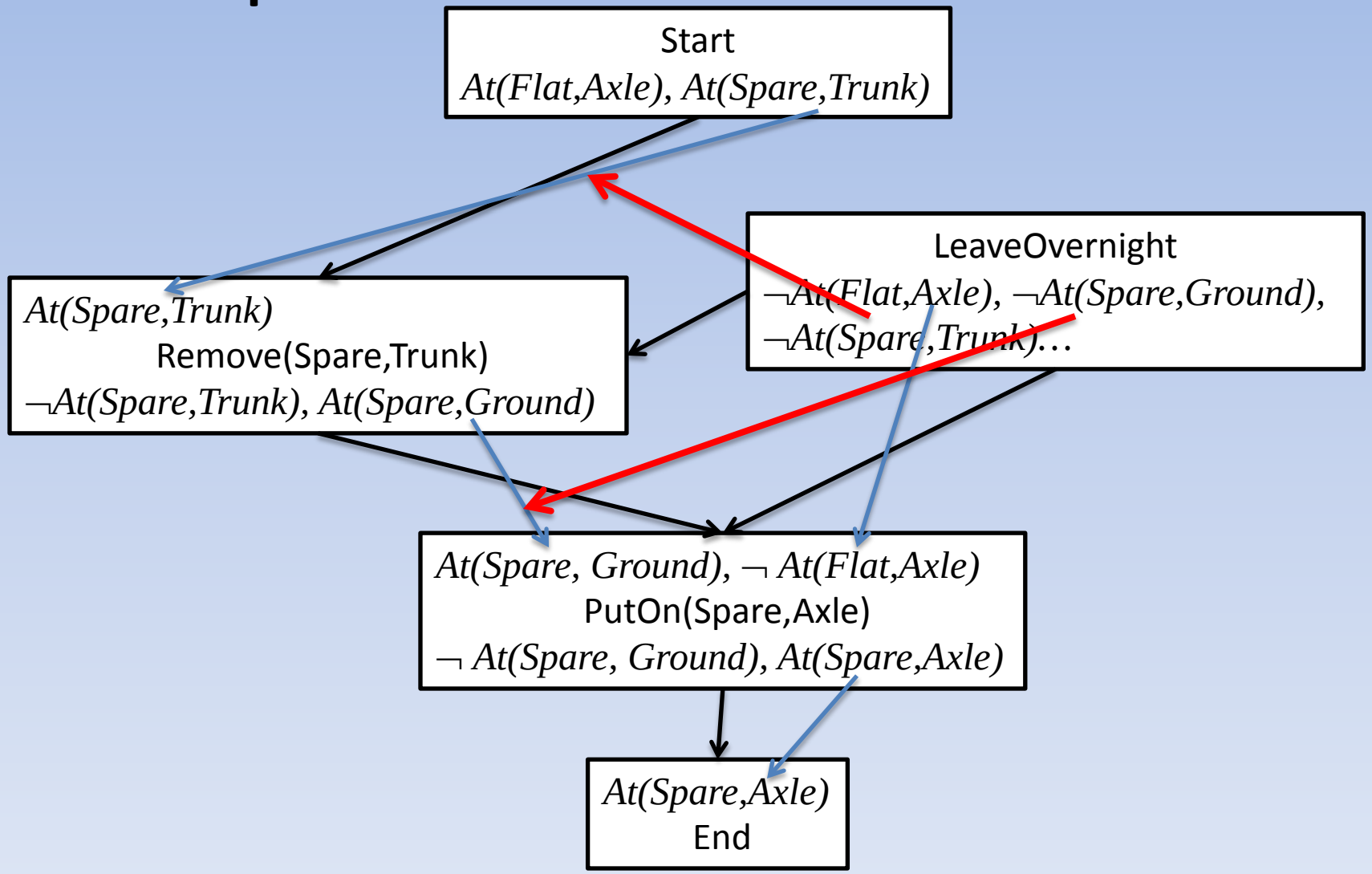
- Conflicts can arise between
  - The new causal link and existing actions
  - The new action and existing causal links

- To resolve a conflict, add ordering constraints
  - Suppose C conflicts with $A \xrightarrow{p} B$
  - Then add either $C < A$ or $B < C$, and generate successor states for each (assuming no cycles)

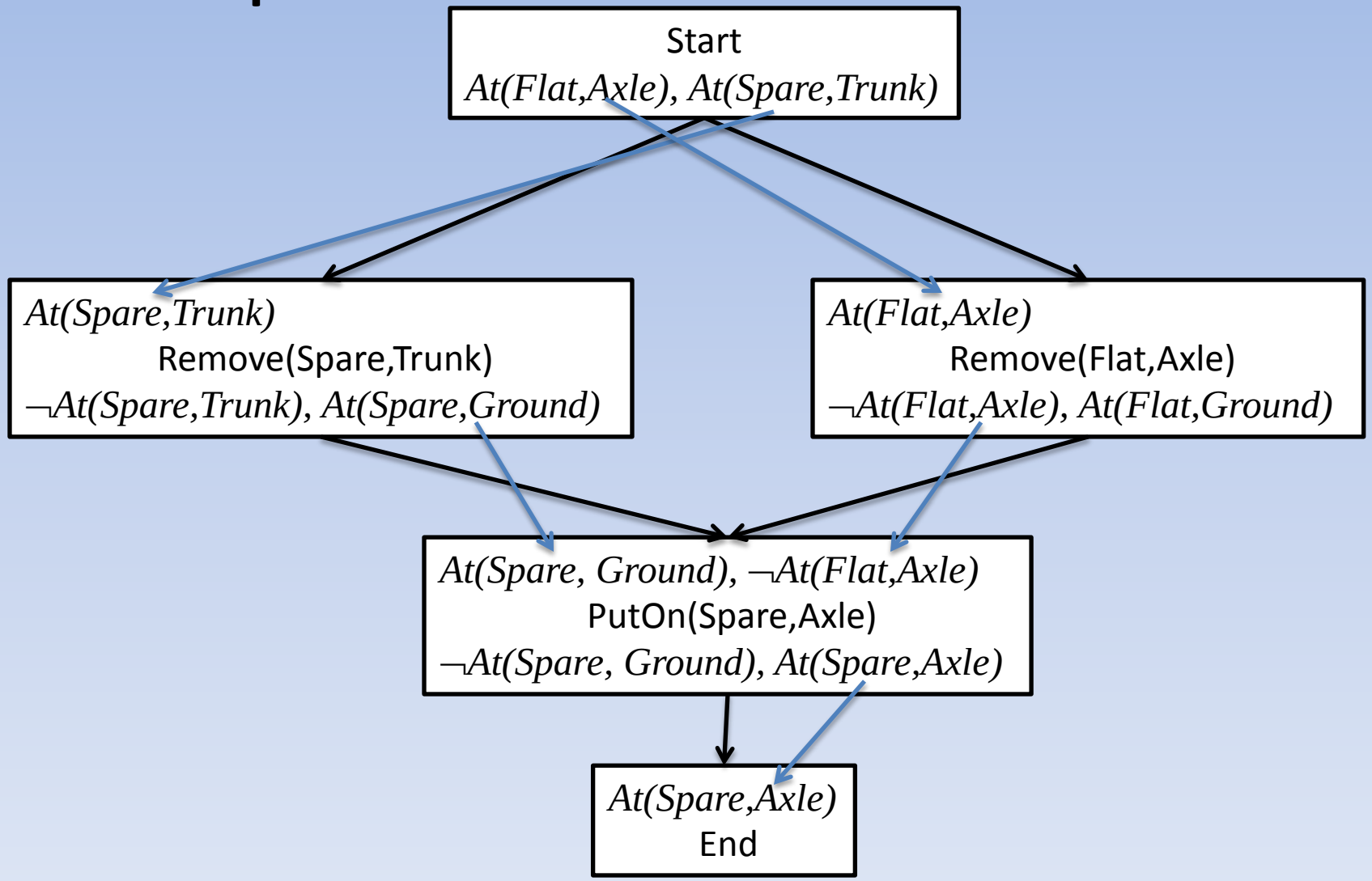
Example

- Init: *At(Flat,Axle),At(Spare,Trunk)*
- Goal: *At(Spare,Axle)*
- *Remove(Spare,Trunk)*
  - PRE: *At(Spare,Trunk)*
  - EFF: *¬At(Spare,Trunk), At(Spare,Ground)*
- *Remove(Flat,Axle)*
  - PRE: *At(Flat,Axle)*
  - EFF: *¬At(Flat,Axle), At(Flat,Ground)*
- *PutOn(Spare,Axle)*
  - PRE: *At(Spare, Ground), ¬At(Flat,Axle)*
  - EFF: *¬At(Spare, Ground), At(Spare,Axle)*
- *LeaveOvernight*
  - PRE:
  - EFF: *¬At(Spare,Trunk), ¬At(Spare, Ground), ¬At(Spare,Axle), ¬At(Flat,Axle), ¬At(Flat,Ground)*

Example


Start
At(Flat,Axle), At(Spare,Trunk)
LeaveOvernight
¬At(Flat,Axle), ¬At(Spare,Ground),
¬At(Spare,Trunk)…
At(Spare,Trunk)
Remove(Spare,Trunk)
¬At(Spare,Trunk), At(Spare,Ground)
At(Spare, Ground), ¬ At(Flat,Axle)
PutOn(Spare,Axle)
¬ At(Spare, Ground), At(Spare,Axle)
At(Spare,Axle)
End

Example

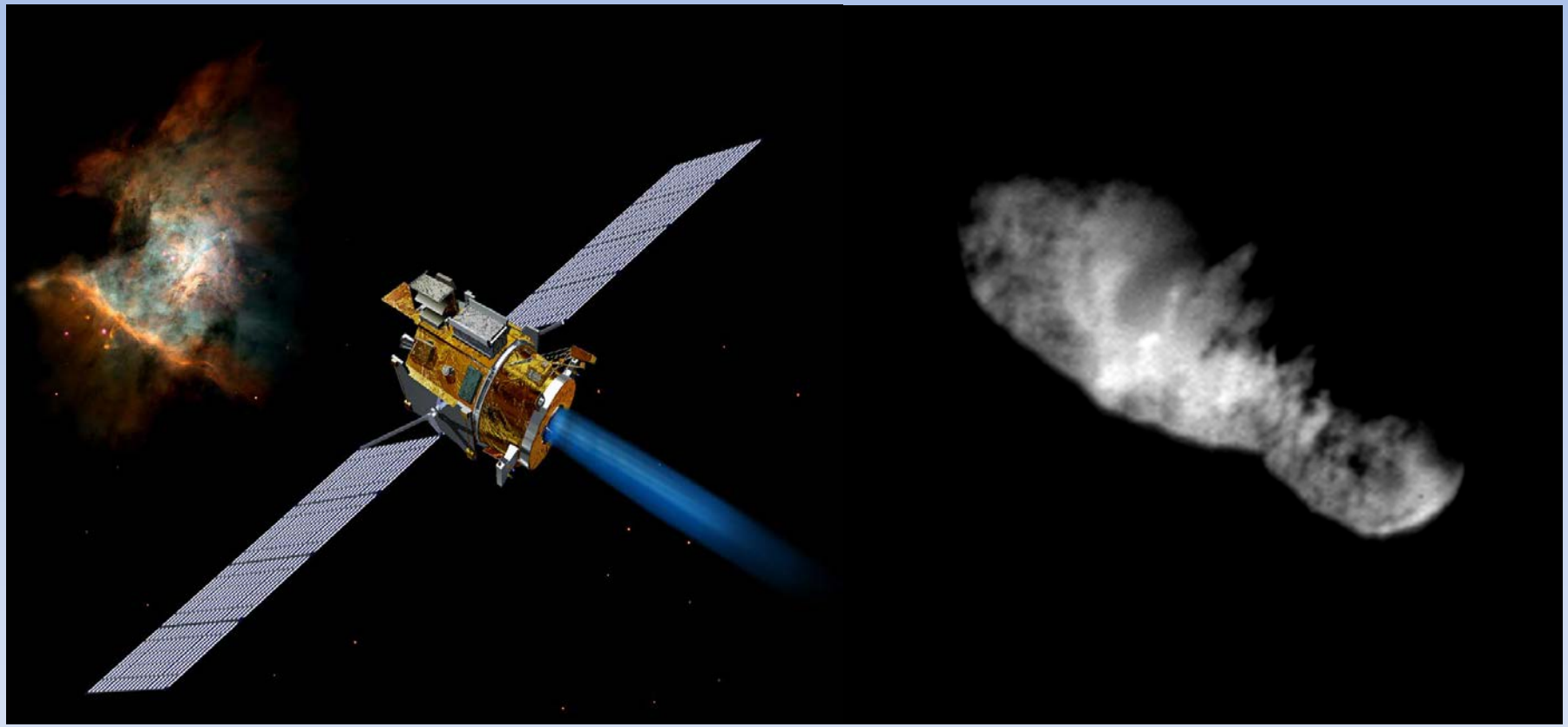

Start
At(Flat,Axle), At(Spare,Trunk)
At(Spare,Trunk)
Remove(Spare,Trunk)
¬At(Spare,Trunk), At(Spare,Ground)
At(Flat,Axle)
Remove(Flat,Axle)
¬At(Flat,Axle), At(Flat,Ground)
At(Spare, Ground), ¬At(Flat,Axle)
PutOn(Spare,Axle)
¬At(Spare, Ground), At(Spare,Axle)
At(Spare,Axle)
End

Other algorithms

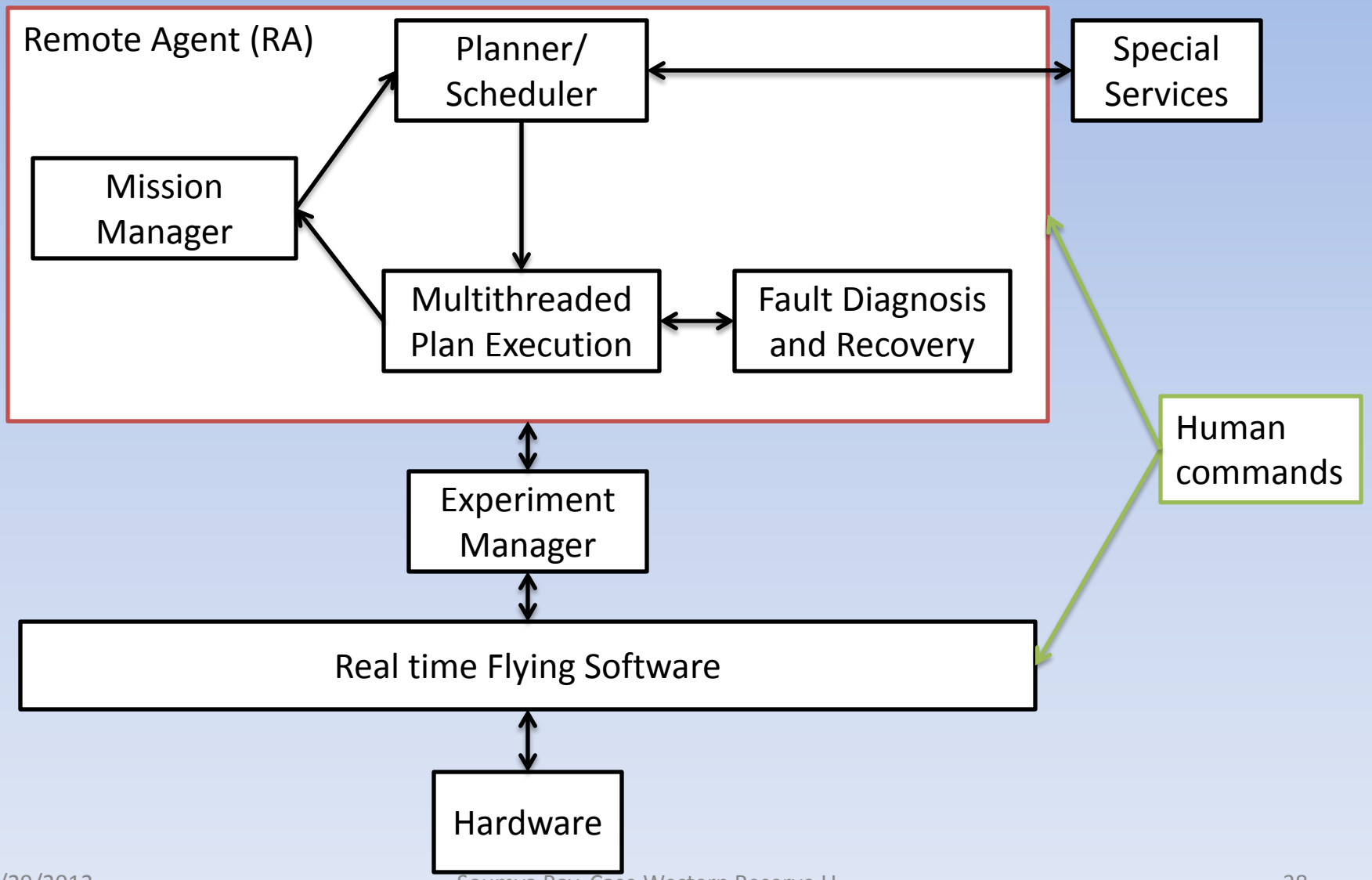
- For classical planning: GraphPlan, SATplan etc
- Algorithms for nonclassical planning
- (EECS 491)

Application: Deep Space 1

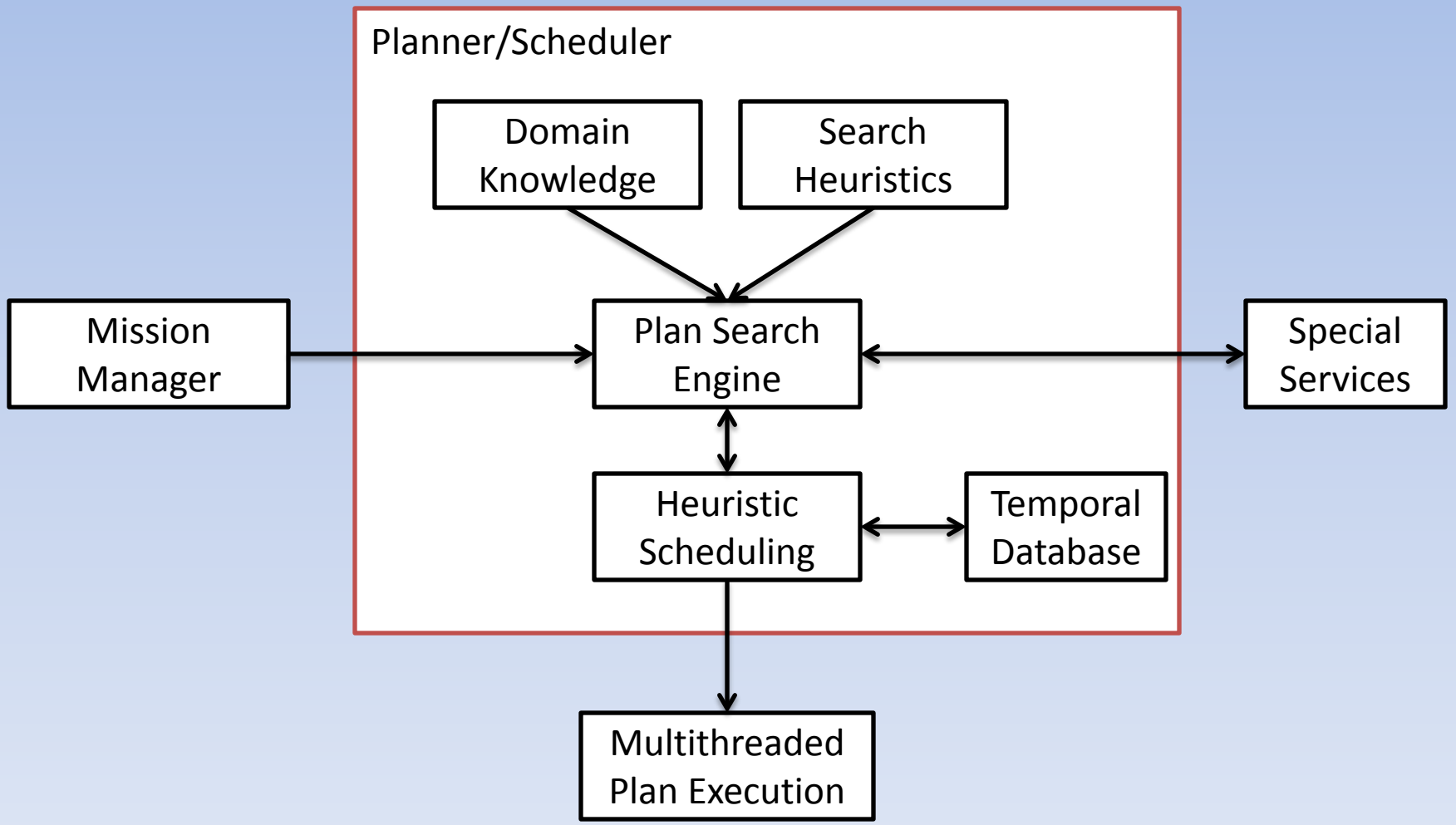
(Active mission Oct 24 1998-Dec 18 2001)



Controller Architecture



Planner/Scheduler



Summary

- We learned about:
  - Forward State Space search
  - Backward Search
  - Partial Order Planning