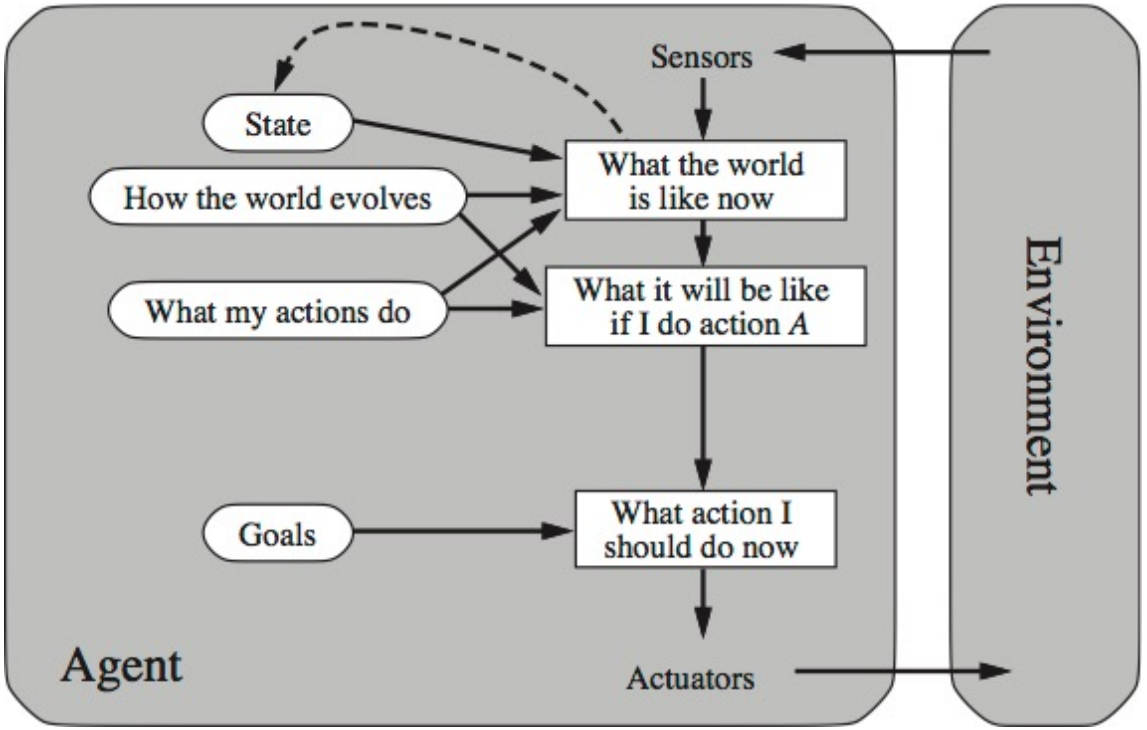
Planning

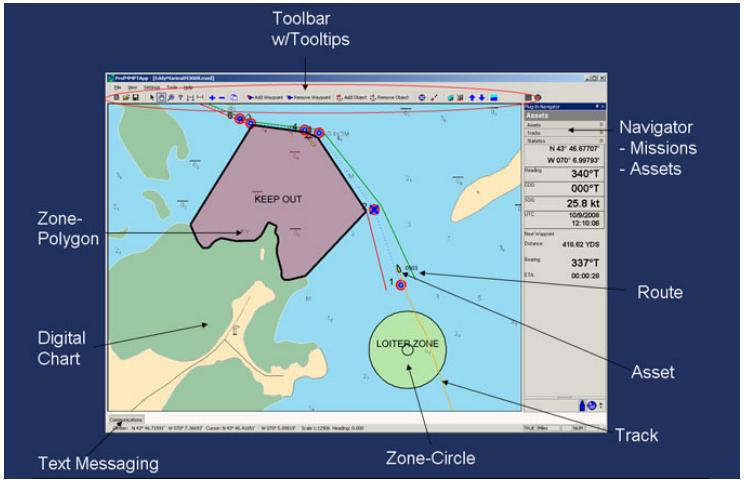
Larry Holder
School of EECS
Washington State University

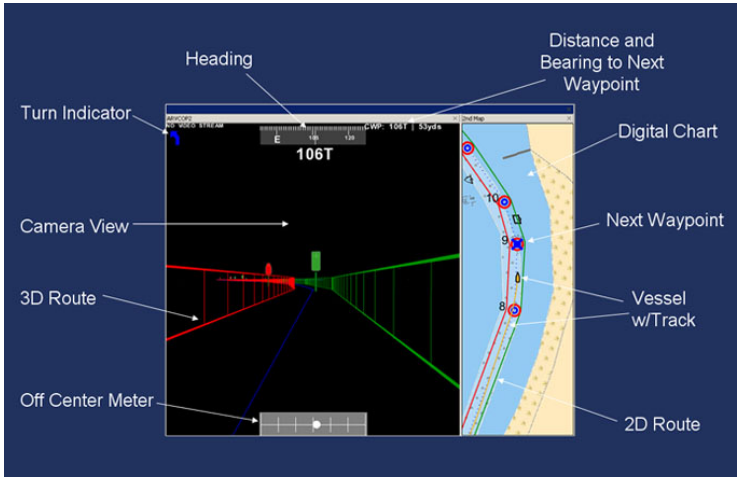
Planning

- Goal-based agent
- Determine a sequence of actions to achieve a goal



Planning Applications





Planning Approaches

- Search-based approach
  - Does not reason about actions (black boxes)
  - Inefficient when many actions (branching factor)
- Logic-based approach
  - Reasoning about change over time cumbersome (e.g., frame axioms)
  - Inefficient due to many applicable rules
- Can we combine the best of both?

Example: Blocks World

Start State

Goal State

Example: Blocks World

Init(On(A, Table) $\wedge$ On(B, Table) $\wedge$ On(C, A) $\wedge$ Block(A) $\wedge$ Block(B) $\wedge$
Block(C) $\wedge$ Clear(B) $\wedge$ Clear(C))

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

Action(Move(b, x, y),
PRECOND: On(b, x) $\wedge$ Clear(b) $\wedge$ Clear(y) $\wedge$ Block(b) $\wedge$ Block(y) $\wedge$
$(b \neq x) \wedge (b \neq y) \wedge (x \neq y)$,
EFFECT: On(b, y) $\wedge$ Clear(x) $\wedge \neg$On(b, x) $\wedge \neg$Clear(y))

Action(MoveToTable(b, x),
PRECOND: On(b, x) $\wedge$ Clear(b) $\wedge$ Block(b) $\wedge$ $(b \neq x)$
EFFECT: On(b, Table) $\wedge$ Clear(x) $\wedge \neg$On(b, x))

Definitions

- Planning Domain Definition Language (PDDL)
- Initial state
- Actions
- Results
- Goal test

PDDL: State

- Conjunction of ground functionless atoms (i.e., positive ground literals)
  - At(Robot1, Room1) ∧ At(Robot2, Room3)
  - At(Home) ∧ Have(Milk) ∧ Have(Bananas) ∧ …
  - At(Home) ∧ IsAt(Umbrella, Home) ∧ CanBeCarried(Umbrella) ∧ IsUmbrella(Umbrella) ∧ HandEmpty ∧ Dry
- The following are not okay as part of a state
  - ¬At(Home) (a negative literal)
  - IsAt(x, y) (not ground)
  - IsAt(Father(Fred), Home) (uses a function symbol)
- Closed-world assumption
  - If don't mention At(Home), then assume ¬At(Home)

PDDL: Goal Test

- Goal
  - Conjunction of literals (positive or negative, possibly with variables)
  - Variables are existentially quantified
  - A partially specified state
- Examples
  - At(Home) $\wedge$ Have(Milk) $\wedge$ Have(Bananas) $\wedge$ Rich $\wedge$ Famous
  - At(x) $\wedge$ Sells(x, Milk) (be at a store that sells milk)
- A state s satisfies goal g if s contains (unifies with) all the literals of g
  - At(Home) $\wedge$ Have(Milk) $\wedge$ Have(Bananas) $\wedge$ Rich $\wedge$ Famous satisfies At(x) $\wedge$ Rich $\wedge$ Famous

PDDL: Actions

- Actions are modeled as state transformations
- Actions are described by a set of action schemas that implicitly define ACTIONS(s) and RESULT(s,a)
- Description of an action should only mention what changes (address frame problem)
- Action schema:

Action(Fly(p, from, to)
PRECOND: At(p, from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
EFFECT: $\neg$At(p, from) $\wedge$ At(p, to))

PDDL: Actions

- Precondition: What must be true for the action to be applicable
- Effect: Changes to the state as a result of taking the action
- Conjunction of literals (positive or negative)

Action(Fly(p, from, to)
PRECOND: At(p, from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
EFFECT: $\neg$At(p, from) $\wedge$ At(p, to))

Action(Fly(P1, SEA, LAX) (ground action)
PRECOND: At(P1, SEA) $\wedge$ Plane(P1) $\wedge$ Airport(SEA) $\wedge$ Airport(LAX)
EFFECT: $\neg$At(P1, SEA) $\wedge$ At(P1, LAX))

PDDL: Actions

- Action a can be executed in state s if s entails the precondition of a
  - $(a \in \text{ACTIONS}(s)) \Leftrightarrow s \models \text{PRECOND}(a)$
  - where any variables in a are universally quantified
- For example:

$$\forall p, from, to\ (\text{Fly}(p, from, to) \in \text{ACTIONS}(s)) \Leftrightarrow$$
$$s \models (\text{At}(p, from) \wedge \text{Plane}(p) \wedge \text{Airport}(from) \wedge \text{Airport}(to))$$

PDDL: Actions

- The result of executing action a in state s is defined as state s'
- State s' contains the literals of s minus negative literals in EFFECT plus positive literals in EFFECT
- Negated literals in EFFECT called delete list or DEL(a)
- Positive literals in EFFECT called add list or ADD(a)
- RESULT(s, a) = (s – DEL(a)) ∪ ADD(a)

PDDL: Actions

- For action schemas any variable in EFFECT must also appear in PRECOND
  - RESULT(s, a) will therefore have only ground atoms
- Time is implicit in action schemas
  - Precondition refers to time t
  - Effect refers to time t+1
- Action schema can represent a number of different actions
  - Fly(Plane1, LAX, JFK)
  - Fly(Plane3, SEA, LAX)

Example: Air Cargo Transport

Init(At(C1, SFO) $\wedge$ At(C2, JFK) $\wedge$ At(P1, SFO) $\wedge$ At(P2, JFK) $\wedge$ Cargo(C1) $\wedge$
Cargo(C2) $\wedge$ Plane(P1) $\wedge$ Plane(P2) $\wedge$ Airport(JFK) $\wedge$ Airport(SFO))

Goal(At(C1, JFK) $\wedge$ At(C2, SFO))

Action(Load(c, p, a),
PRECOND: At(c, a) $\wedge$ At(p, a) $\wedge$ Cargo(c) $\wedge$ Plane(p) $\wedge$ Airport(a)
EFFECT: $\neg$At(c, a) $\wedge$ In(c, p))

Action(Unload(c, p, a),
PRECOND: In(c, p) $\wedge$ At(p, a) $\wedge$ Cargo(c) $\wedge$ Plane(p) $\wedge$ Airport(a)
EFFECT: At(c, a) $\wedge$ $\neg$In(c, p))

Action(Fly(p, from, to),
PRECOND: At(p, from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
EFFECT: $\neg$At(p, from) $\wedge$ At(p, to))

[Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK), Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO)]

PDDL for Cargo Domain

```
(define (domain CARGO)

  (:predicates (At ?x ?y) (In ?x ?y) (Cargo ?x) (Plane ?x) (Airport ?x))

  (:action Load
    :parameters (?c ?p ?a)
    :precondition (and (At ?c ?a) (At ?p ?a) (Cargo ?c) (Plane ?p) (Airport ?a))
    :effect (and (not (At ?c ?a)) (In ?c ?p))
  )

  (:action Unload
    :parameters (?c ?p ?a)
    :precondition (and (In ?c ?p) (At ?p ?a) (Cargo ?c) (Plane ?p) (Airport ?a))
    :effect (and (not (In ?c ?p)) (At ?c ?a))
  )

  (:action Fly
    :parameters (?p ?from ?to)
    :precondition (and (At ?p ?from) (Plane ?p) (Airport ?from) (Airport ?to))
    :effect (and (not (At ?p ?from)) (At ?p ?to))
  )
)
```

PDDL for Cargo Problems

```
(define (problem prob2)
  (:domain CARGO)
  (:objects C1 C2 P1 P2 JFK SFO)
  (:init (At C1 SFO) (At C2 JFK) (At P1 SFO) (At P2 JFK)
         (Cargo C1) (Cargo C2) (Plane P1) (Plane P2)
         (Airport JFK) (Airport SFO))
  (:goal (and (At C1 JFK) (At C2 SFO)))
)
```

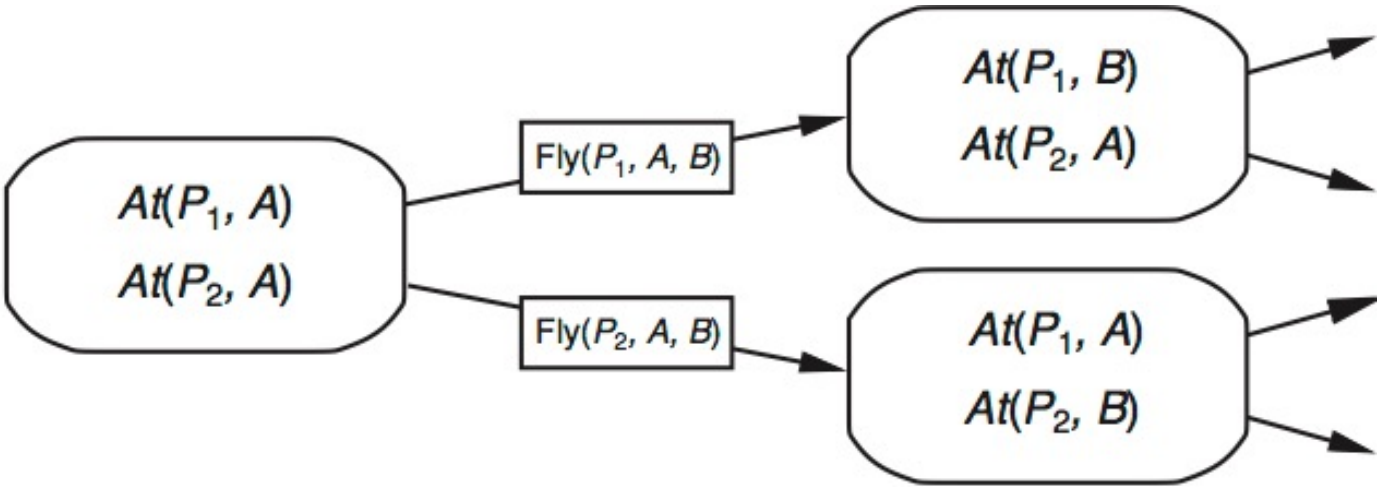
```
(define (problem prob4)
  (:domain CARGO)
  (:objects C1 C2 C3 C4 P1 P2 JFK SFO)
  (:init (At C1 SFO) (At C2 JFK) (At C3 SFO) (At C4 JFK)
         (At P1 SFO) (At P2 JFK)
         (Cargo C1) (Cargo C2) (Cargo C3) (Cargo C4)
         (Plane P1) (Plane P2)
         (Airport JFK) (Airport SFO))
  (:goal (and (At C1 JFK) (At C2 SFO) (At C3 JFK) (At C4 SFO)))
)
```

Solving Planning Problems

- State-space search approach
- Choose actions based on:
  - PRECOND satisfied by current state (forward)
  - EFFECT satisfies current goals (backward)
- Apply any of the previous search algorithms
- Search tree usually large
  - Many instantiations of applicable actions

Forward State-Space Search

- Start at initial state
- Apply actions until current state satisfies goal

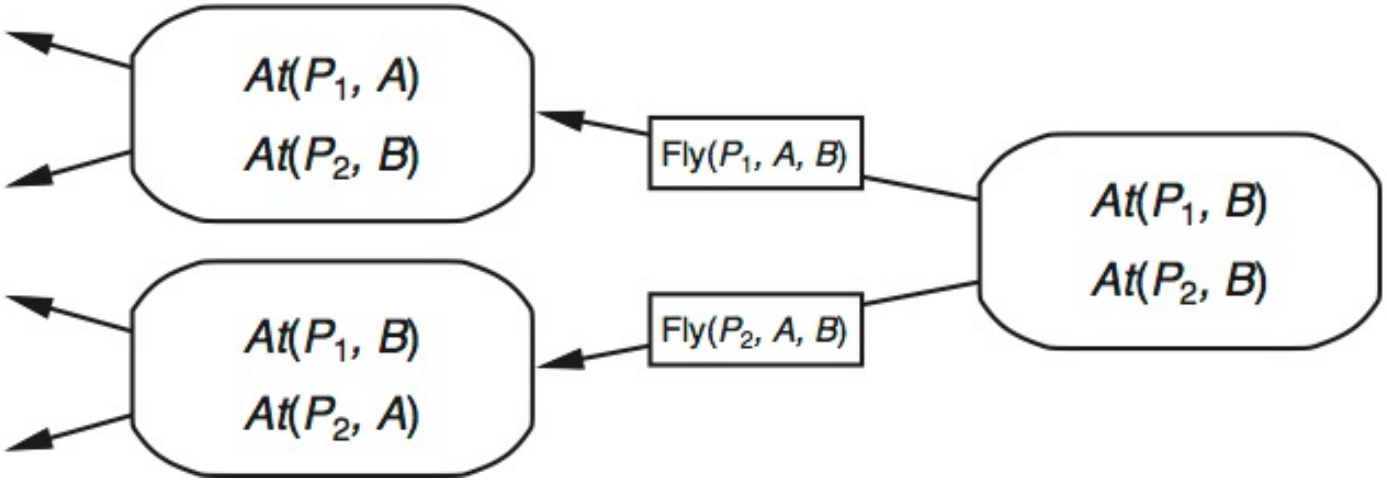


Forward State-Space Search

- Problems
  - Prone to exploring irrelevant actions
    - Example: Goal is Own(book), have action Buy(book), 130 million books
  - Planning problems often have large state spaces
    - Example: Cargo at 10 airports, each with 5 airplanes and 20 pieces of cargo
      - Goal: Move 20 pieces from airport A to airport B (41 steps)
      - Average actions applicable to a state is 2000
      - Search graph has 2000^{41} nodes
- Need accurate heuristics
  - Many real-world applications have strong heuristics

Backward State-Space Search

- Start at goal
- Apply actions backward until we find a sequence of steps that reaches initial state
- Only considers actions relevant to the goal



Backward State-Space Search

- Works only when we know how to regress from a state description to the predecessor state description
- Given ground goal description g and ground action a, the regression from g over a gives a state g' defined by
  - $g' = (g - ADD(a)) \cup PRECOND(a)$

What about DEL(a)?

Backward State-Space Search

- Also need to handle partially uninstantiated actions and states, not just ground ones
  - For example: Goal is At(C2, SFO)
  - Suggested action: Unload(C2, p', SFO)

Action (Unload(C2, p', SFO),
PRECOND: In(C2, p') ∧ At(p', SFO) ∧ Cargo(C2) ∧ Plane(p') ∧ Airport(SFO),
EFFECT: At(C2, SFO) ∧ ¬In(C2, p')

  - Regressed state description:
    - g' = In(C2, p') ∧ At(p', SFO) ∧ Cargo(C2) ∧ Plane(p') ∧ Airport(SFO)

Backward State-Space Search

- Want actions that could be the last step in a plan leading up to the current goal
- At least one of the action's effects (either positive or negative) must unify with an element of the goal
- The action must not have any effect (positive or negative) that negates an element of the goal
  - For example, goal is $A \wedge B \wedge C$ and an action has the effect $A \wedge B \wedge \neg C$.

Backward State-Space Search

- Problems revisited
  - Goal: Own(0136042597)
  - Initial state: 130 million books
  - Action: A = Action(Buy(b), PRECOND: Book(b), EFFECT: Own(b))
  - Unify goal Own(0136042597) with effect Own(b), producing θ = {b/0136042597}
  - Regress over action SUBST(θ, A) to produce the predecessor state description Book(0136042597) (which is satisfied by initial state)

Heuristics for Planning

- Efficient planning (forward or backward) requires good heuristics
- Estimate solution length
  - Ignore some or all preconditions

Action(Fly(p, from, to),
PRECOND: At(p, from) $\wedge$ ~~Plane(p)~~ $\wedge$ Airport(from) $\wedge$ Airport(to)
EFFECT: $\neg$At(p, from) $\wedge$ At(p, to))

  - Ignore delete list

Action(Fly(p, from, to),
PRECOND: At(p, from) $\wedge$ Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)
EFFECT: ~~$\neg$At(p, from)~~ $\wedge$ At(p, to))

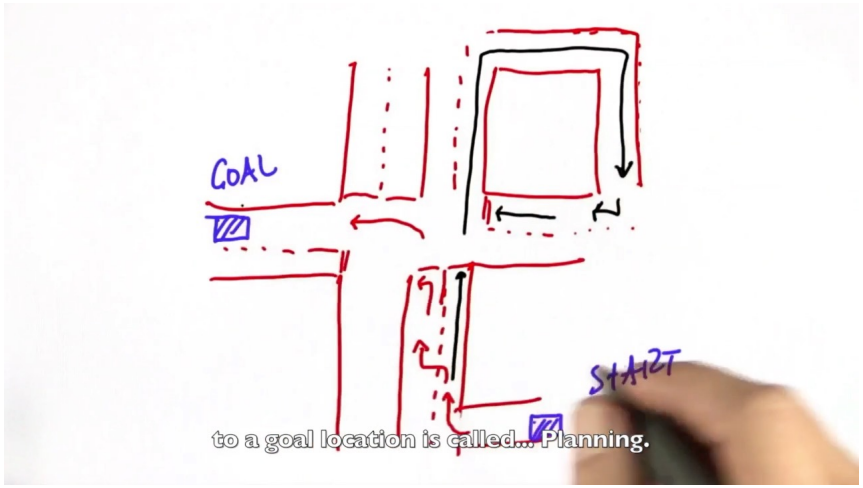
Heuristics for Planning (cont.)

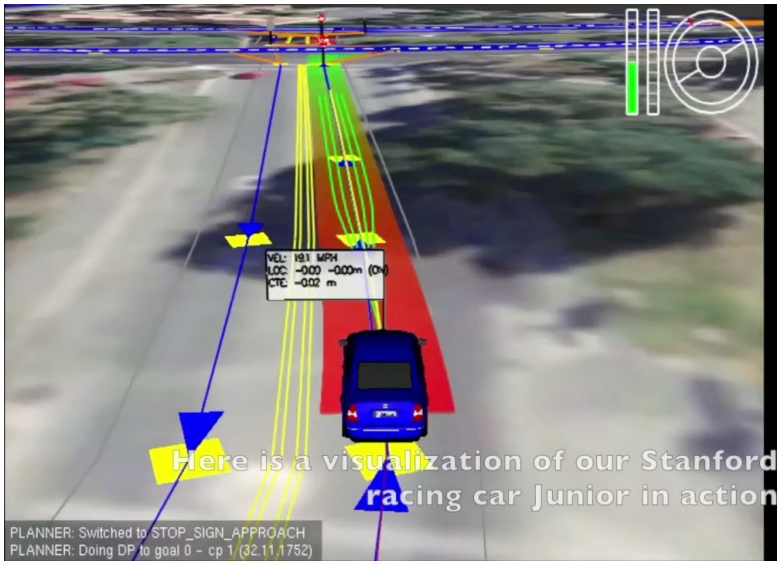
- Estimate solution length
  - Use state abstraction

Action(Fly(p, from, to),
PRECOND: At(p, from) $\wedge$ ~~Plane(p) $\wedge$ Airport(from) $\wedge$ Airport(to)~~
EFFECT: $\neg$At(p, from) $\wedge$ At(p, to))

  - Assume subgoals independent
    - On(A,B) $\wedge$ On(B,C)

Example: Car Navigation





State of the Art Planning

- International Planning Competition (IPC)
  - http://ipc.icaps-conference.org
- Fast-Downward
  - Forward planner
  - Focus on good heuristics
  - Supports full PDDL
  - http://www.fast-downward.org

Summary: Planning

- Planning combines search and logic
- State-space search can operate in the forward or backward direction
- State of the art approaches use combination of techniques
- Many real-world applications: mission planning, scheduling, navigation