Artificial Intelligence: Foundations & Applications

Planning

Prof. Partha P. Chakrabarti & Arijit Mondal
Indian Institute of Technology Kharagpur

Techniques seen till now

- Search
  - Most fundamental approach
  - Need to define states, moves, state-transiton rules, etc.
- CSP
  - Search through constraint propagation
- Propositional logic
  - Deduction in a single state, no state change

- Probabilistic reasoning
  - Logic augmented with probabilities
- Temporal logic
  - Logic involving time
- Planning
  - Search involving logic
  - Change of states

Real world planning problems

- Autonomous vehicle navigation
- Robotics movement
- Travel planning
- Process control
- Assembly line
- Military operations
- Information gathering
- many more . . .

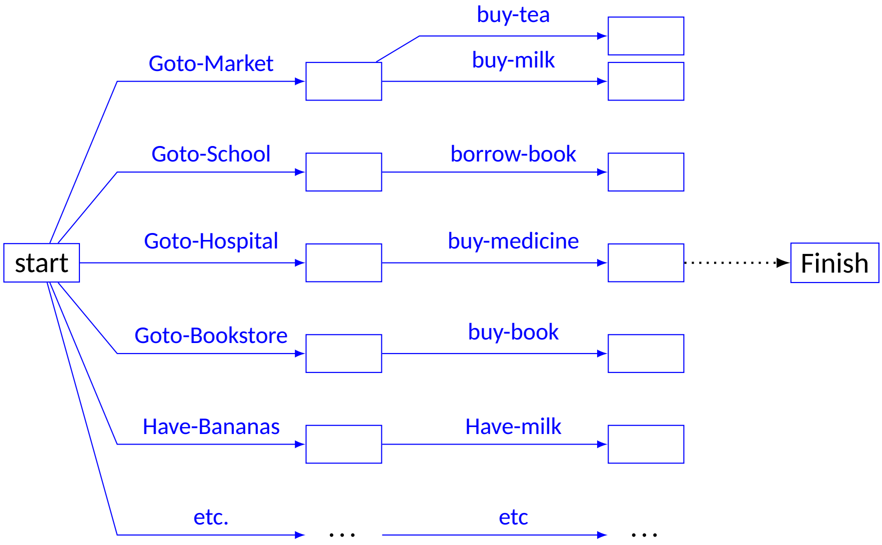
A simple planning problem

- Get me milk, bananas and a book
- Given
  - Initial state - agent is at *home* without milk, bananas and book
  - Goal state - agent is at *home* with milk, bananas and book
  - Actions / Moves - agent can perform on a given state
    - Buy(X) - buy item X where $X \in \{milk, bananas, book\}$
    - Steal(X) - steal item X where $X \in \{milk, bananas, book\}$
    - Goto(X) - move to X where $X \in \{market, home\}$
    - ...

The planning problem

- Generate one possible way to achieve a certain goal given an initial situation and a set of actions
- Similar to search problems
  - Start state
  - List of moves
  - Result of moves
  - Goal state

Search



buy-tea
Goto-Market
buy-milk
Goto-School
borrow-book
Goto-Hospital
buy-medicine
start
Finish
Goto-Bookstore
buy-book
Have-Bananas
Have-milk
etc.
. . .
etc
. . .

Planning vs Search

- Actions have requirements and consequences that should constrain applicability in a given state
  - Stronger interaction between actions and states needed
- Most parts of the world are independent of most other parts
  - Solve subgoals independently
- Human beings plan goal-directed, they construct important intermediate solutions first
  - Flexible sequence for construction of solution
- Planning systems do the following
  - Unify action and goal representation to allow selection (use logical language for both)
  - Divide-and-conquer by subgoaling
  - Relax requirement for sequential construction of solutions

STRIPS

- **STanford Research Institute Problem Solver**
- **Many planners today use specification languages that are variants of the one used in STRIPS**

Representation

- States - conjunction of propositions
  - Example: AT(Home)$\wedge\neg$ Have(tea)$\wedge\neg$Have(bananas)$\wedge\neg$Have(book)
- Close world assumption - atoms that are not present are treated as false
- Actions - Serves as names
  - Precondition - conjunction of literals
  - Effect - conjunction of literals
  - Example:
    - Action: Goto(Market)
    - Precondition: AT(home)
    - Effect: AT(Market)
- Plan - Solution for the problem
  - A set of plan steps. Each step is one of the operators for the problem.
  - A set of step ordering constraints. Each ordering constraint is of the form $S_i \prec S_j$, indicating S_i must occur sometime before S_j.

Example - Flight operation

- Flying a plane from one location to another
- Actions - FLY(plane-id, from, to)
  - Preconditions - AT(plane-id,from)$\wedge$Airport(from)$\wedge$Airport(to)
  - Effects - $\neg$AT(plane-id,from)$\wedge$AT(plane-id, to)

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \land AT(C_2, DEL) \land AT(P_1, CCU) \land AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \land AT(C_2, CCU)$

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \land AT(C_2, DEL) \land AT(P_1, CCU) \land AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \land AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**

Example - Air Cargo

- Cargo transport involving loading and unloading and flying it from one place to another
- Initial state - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- Goal state - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- Action - Load(c, p, a)
  - Precondition - AT(c, a) ∧ AT(p, a)
  - Effect - ¬AT(c, a) ∧ In(c, p)
- Action - Unload(c, p, a)
  - Precondition - In(c, p) ∧ AT(P, a)
  - Effect - AT(c, a) ∧ ¬In(c, p)

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**
  - **Precondition - In(c, p) ∧ AT(P, a)**
  - **Effect - AT(c, a) ∧ ¬In(c, p)**
- **Action - Fly(p, from, to)**

Example - Air Cargo

- Cargo transport involving loading and unloading and flying it from one place to another
- Initial state - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- Goal state - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- Action - Load(c, p, a)
  - Precondition - AT(c, a) ∧ AT(p, a)
  - Effect - ¬AT(c, a) ∧ In(c, p)
- Action - Unload(c, p, a)
  - Precondition - In(c, p) ∧ AT(P, a)
  - Effect - AT(c, a) ∧ ¬In(c, p)
- Action - Fly(p, from, to)
  - Precondition - AT(p, from)
  - Effect - ¬AT(p, from) ∧ AT(p, to)

Example - Air Cargo

- Cargo transport involving loading and unloading and flying it from one place to another
- Initial state - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- Goal state - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- Action - Load(c, p, a)
  - Precondition - AT(c, a) ∧ AT(p, a)
  - Effect - ¬AT(c, a) ∧ In(c, p)
- Action - Unload(c, p, a)
  - Precondition - In(c, p) ∧ AT(P, a)
  - Effect - AT(c, a) ∧ ¬In(c, p)
- Action - Fly(p, from, to)
  - Precondition - AT(p, from)
  - Effect - ¬AT(p, from) ∧ AT(p, to)

- Plan

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**
  - **Precondition - In(c, p) ∧ AT(P, a)**
  - **Effect - AT(c, a) ∧ ¬In(c, p)**
- **Action - Fly(p, from, to)**
  - **Precondition - AT(p, from)**
  - **Effect - ¬AT(p, from) ∧ AT(p, to)**

- **Plan**
  - **Load(**C_1**,** P_1**, CCU)**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**
  - **Precondition - In(c, p) ∧ AT(P, a)**
  - **Effect - AT(c, a) ∧ ¬In(c, p)**
- **Action - Fly(p, from, to)**
  - **Precondition - AT(p, from)**
  - **Effect - ¬AT(p, from) ∧ AT(p, to)**

- **Plan**
  - **Load(C_1, P_1, CCU)**
  - **Fly(P_1, CCU, DEL)**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \wedge AT(C_2, DEL) \wedge AT(P_1, CCU) \wedge AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \wedge AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**
  - **Precondition - In(c, p) ∧ AT(P, a)**
  - **Effect - AT(c, a) ∧ ¬In(c, p)**
- **Action - Fly(p, from, to)**
  - **Precondition - AT(p, from)**
  - **Effect - ¬AT(p, from) ∧ AT(p, to)**

- **Plan**
  - **Load(C_1, P_1, CCU)**
  - **Fly(P_1, CCU, DEL)**
  - **Unload(C_1, P_1, DEL)**

Example - Air Cargo

- **Cargo transport involving loading and unloading and flying it from one place to another**
- **Initial state** - $AT(C_1, CCU) \land AT(C_2, DEL) \land AT(P_1, CCU) \land AT(P_2, DEL)$
- **Goal state** - $AT(C_1, DEL) \land AT(C_2, CCU)$

- **Action - Load(c, p, a)**
  - **Precondition - AT(c, a) ∧ AT(p, a)**
  - **Effect - ¬AT(c, a) ∧ In(c, p)**
- **Action - Unload(c, p, a)**
  - **Precondition - In(c, p) ∧ AT(P, a)**
  - **Effect - AT(c, a) ∧ ¬In(c, p)**
- **Action - Fly(p, from, to)**
  - **Precondition - AT(p, from)**
  - **Effect - ¬AT(p, from) ∧ AT(p, to)**

- **Plan**
  - **Load(C_1, P_1, CCU)**
  - **Fly(P_1, CCU, DEL)**
  - **Unload(C_1, P_1, DEL)**
  - **Load(C_2, P_2, DEL)**
  - **Fly(P_2, DEL, CCU)**
  - **Unload(C_2, P_2, CCU)**

Example - Flat tire

- **Change a flat tire with a spare one**

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) $\wedge$ Tire(Spare) $\wedge$ AT(Flat, Axle) $\wedge$ At(Spare, Trunk)

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) $\wedge$ Tire(Spare) $\wedge$ AT(Flat, Axle) $\wedge$ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) $\wedge$ Tire(Spare) $\wedge$ AT(Flat, Axle) $\wedge$ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) $\wedge$ Tire(Spare) $\wedge$ AT(Flat, Axle) $\wedge$ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)
  - Preconditions - AT(obj, loc)
  - Effects - $\neg$AT(obj,loc) $\wedge$ AT(obj, Ground)

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) $\wedge$ Tire(Spare) $\wedge$ AT(Flat, Axle) $\wedge$ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)
  - Preconditions - AT(obj, loc)
  - Effects - $\neg$AT(obj,loc) $\wedge$ AT(obj, Ground)
- Action - PutOn(t, axle)

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) ∧ Tire(Spare) ∧ AT(Flat, Axle) ∧ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)
  - Preconditions - AT(obj, loc)
  - Effects - ¬AT(obj,loc) ∧ AT(obj, Ground)
- Action - PutOn(t, axle)
  - Preconditions - Tire(t) ∧ AT(t, Ground) ∧ ¬AT(Flat, axle)
  - Effects - ¬AT(t, Ground) ∧ AT(t, axle)

Example - Flat tire

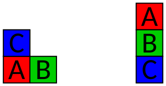
- Change a flat tire with a spare one
- Initial state - Tire(flat) ∧ Tire(Spare) ∧ AT(Flat, Axle) ∧ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)
  - Preconditions - AT(obj, loc)
  - Effects - ¬AT(obj,loc) ∧ AT(obj, Ground)
- Action - PutOn(t, axle)
  - Preconditions - Tire(t) ∧ AT(t, Ground) ∧ ¬AT(Flat, axle)
  - Effects - ¬AT(t, Ground) ∧ AT(t, axle)
- Plan

Example - Flat tire

- Change a flat tire with a spare one
- Initial state - Tire(flat) ∧ Tire(Spare) ∧ AT(Flat, Axle) ∧ At(Spare, Trunk)
- Goal state - AT(Spare, Axle)
- Action - Remove(obj, loc)
  - Preconditions - AT(obj, loc)
  - Effects - ¬AT(obj,loc) ∧ AT(obj, Ground)
- Action - PutOn(t, axle)
  - Preconditions - Tire(t) ∧ AT(t, Ground) ∧ ¬AT(Flat, axle)
  - Effects - ¬AT(t, Ground) ∧ AT(t, axle)
- Plan
  - Remove(Flat, Axle)
  - Remove(Spare, Trunk)
  - PutOn(Spare, Axle)

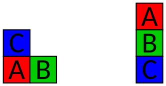
Example - Blocks world

- **Build a 3-block tower**



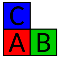
Example - Blocks world

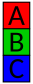
- **Build a 3-block tower**
- **Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)**



Example - Blocks world

- **Build a 3-block tower**
- **Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)**
- **Goal state - ON(A,B) ∧ ON(B,C)**

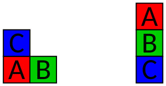




Example - Blocks world

- Build a 3-block tower
- Initial state - ON(A,Table) $\wedge$ ON(B,Table) $\wedge$ ON(C,A) $\wedge$ Clear(B) $\wedge$ Clear(C)
- Goal state - ON(A,B) $\wedge$ ON(B,C)

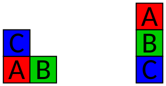
- Action - move(x, y)



Example - Blocks world

- Build a 3-block tower
- Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)
- Goal state - ON(A,B) ∧ ON(B,C)

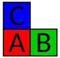
- Action - move(x, y)
  - Precondition - Clear(x) ∧ Clear(y)
  - Effect - ON(x, y)

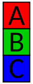


Example - Blocks world

- Build a 3-block tower
- Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)
- Goal state - ON(A,B) ∧ ON(B,C)

- Action - move(x, y)
  - Precondition - Clear(x) ∧ Clear(y)
  - Effect - ON(x, y)
- Action - moveToTable(x, Table)

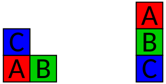




Example - Blocks world

- Build a 3-block tower
- Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)
- Goal state - ON(A,B) ∧ ON(B,C)

- Action - move(x, y)
  - Precondition - Clear(x) ∧ Clear(y)
  - Effect - ON(x, y)
- Action - moveToTable(x, Table)
  - Precondition - Clear(x)
  - Effect - ON(x, Table)

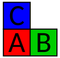


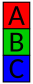
Example - Blocks world

- Build a 3-block tower
- Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)
- Goal state - ON(A,B) ∧ ON(B,C)

- Action - move(x, y)
  - Precondition - Clear(x) ∧ Clear(y)
  - Effect - ON(x, y)
- Action - moveToTable(x, Table)
  - Precondition - Clear(x)
  - Effect - ON(x, Table)

- Plan



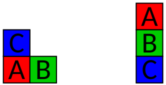


Example - Blocks world

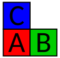
- Build a 3-block tower
- Initial state - ON(A,Table) ∧ ON(B,Table) ∧ ON(C,A) ∧ Clear(B) ∧ Clear(C)
- Goal state - ON(A,B) ∧ ON(B,C)

- Action - move(x, y)
  - Precondition - Clear(x) ∧ Clear(y)
  - Effect - ON(x, y)
- Action - moveToTable(x, Table)
  - Precondition - Clear(x)
  - Effect - ON(x, Table)

- Plan
  - moveToTable(C, Table)
  - move(B, C)
  - move(A, B)

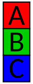


Blocks world - I



On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

On(A,B) ∧ On(B,C)

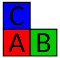


Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

- Move C to the table
  - It achieves none of the goal predicates
- Move C to top of B
  - It achieves none of the goal predicates
- Move B to top of C
  - It achieves On(B,C)

Blocks world - I

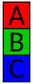


On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

Clear(C) ∧ Clear(B)

$\boxed{\text{Move(B,C)}}$

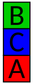
On(A,B) ∧ On(B,C)



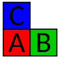
Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

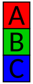
Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

We obtain the following

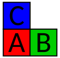


Blocks world - II



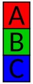


Blocks world - II

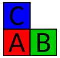


On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

On(A,B) ∧ On(B,C)



Blocks world - II

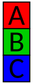


On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

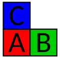
Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

On(A,B) ∧ On(B,C)



Blocks world - II



On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

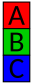
Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

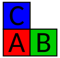
MoveTT(C)

Move(A,B)

On(A,B) ∧ On(B,C)



Blocks world - II



On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

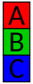
Clear(C)

MoveTT(C)

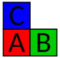
Clear(A) ∧ On(C,Table)

Move(A,B)

On(A,B) ∧ On(B,C)



Blocks world - II



On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

Clear(C)

MoveTT(C)

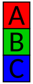
Clear(A) ∧ On(C,Table)

Clear(A) ∧ Clear(B)

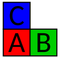
Move(A,B)

¬Clear(B)

On(A,B) ∧ On(B,C)



Blocks world - II



On(C,A) ∧ On(A,Table) ∧ On(B,Table) ∧ Clear(C) ∧ Clear(B)

Clear(C)

MoveTT(C)

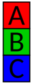
Clear(A) ∧ On(C,Table)

Clear(A) ∧ Clear(B)

Move(A,B)

¬Clear(B)

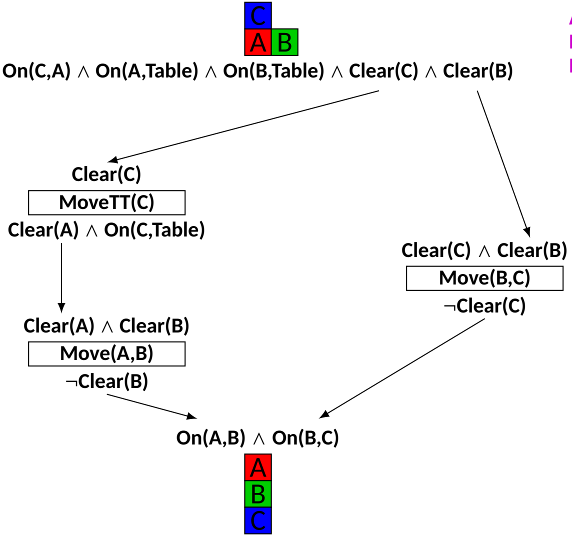
On(A,B) ∧ On(B,C)



Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

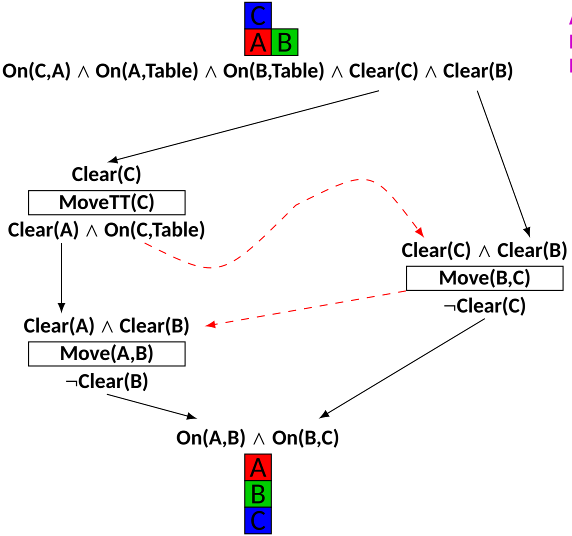
Blocks world - II



Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

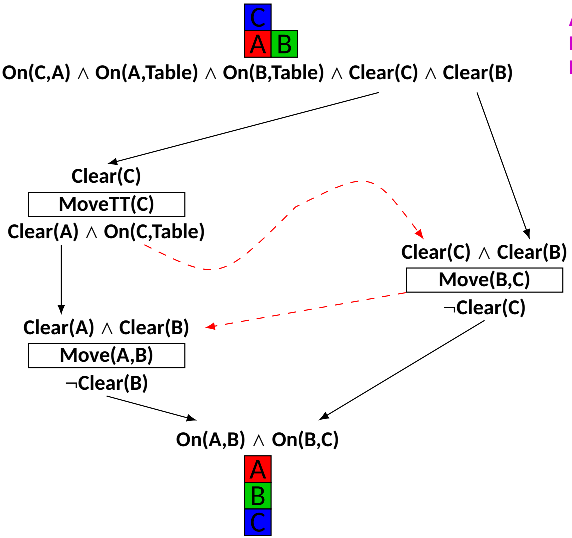
Blocks world - II



Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

Blocks world - II



Action: Move(X,Y)
Precondition: Clear(X) ∧ Clear(Y)
Effect: On(X,Y)

Action: MoveTT(X)
Precondition: Clear(X)
Effect: On(X,Table)

- Total ordering is
  - MoveTT(C)
  - Move(B,C)
  - Move(A,B)

Shocks

- Initial state : ∅
- Goal state: LeftShoeOn ∧ RightShoeOn
- Action - LeftSock
  - Precondition: ∅
  - Effect: LeftSockOn
- Action - RightSock
  - Precondition: ∅
  - Effect: RightSockOn
- Action - LeftShoe
  - Precondition: LeftSockOn
  - Effect: LeftShoeOn
- Action - RightSock
  - Precondition: RightSockOn
  - Effect: RightShoeOn

Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**

Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**

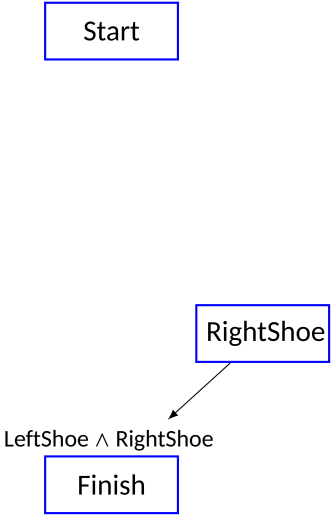
Start

LeftShoe ∧ RightShoe

Finish

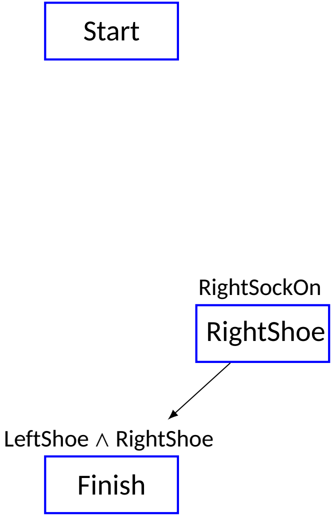
Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**



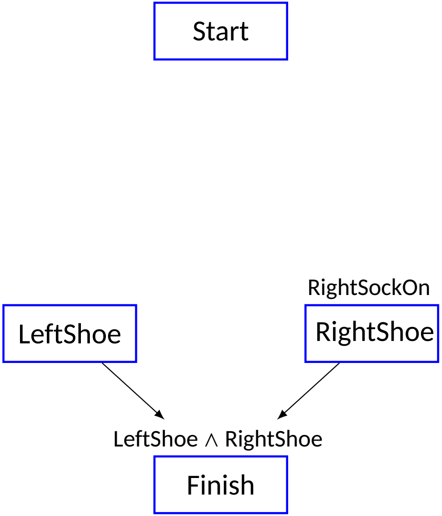
Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**



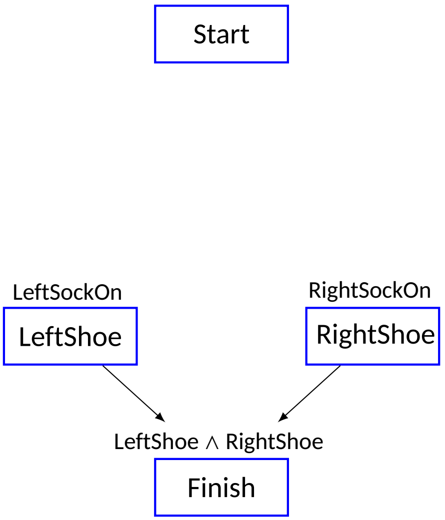
Shocks

- Initial state : ∅
- Goal state: LeftShoeOn ∧ RightShoeOn
- Action - LeftSock
  - Precondition: ∅
  - Effect: LeftSockOn
- Action - RightSock
  - Precondition: ∅
  - Effect: RightSockOn
- Action - LeftShoe
  - Precondition: LeftSockOn
  - Effect: LeftShoeOn
- Action - RightSock
  - Precondition: RightSockOn
  - Effect: RightShoeOn



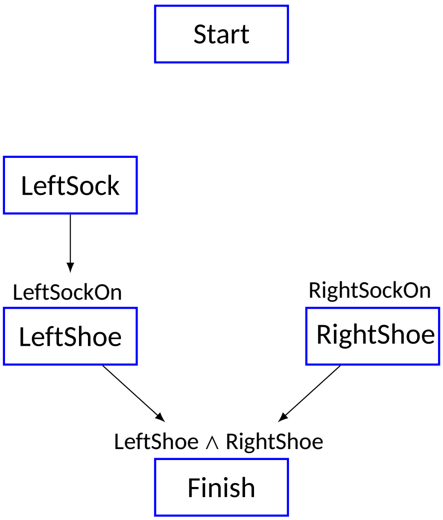
Shocks

- Initial state : ∅
- Goal state: LeftShoeOn ∧ RightShoeOn
- Action - LeftSock
  - Precondition: ∅
  - Effect: LeftSockOn
- Action - RightSock
  - Precondition: ∅
  - Effect: RightSockOn
- Action - LeftShoe
  - Precondition: LeftSockOn
  - Effect: LeftShoeOn
- Action - RightSock
  - Precondition: RightSockOn
  - Effect: RightShoeOn



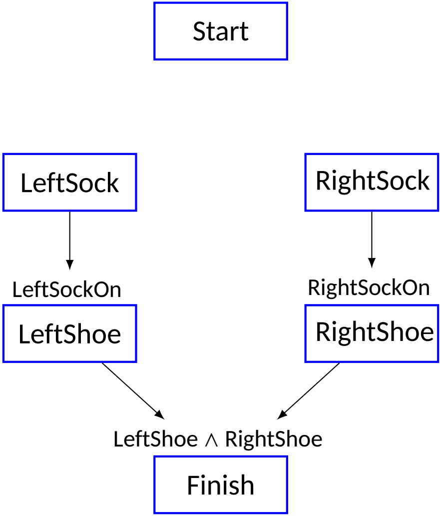
Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**



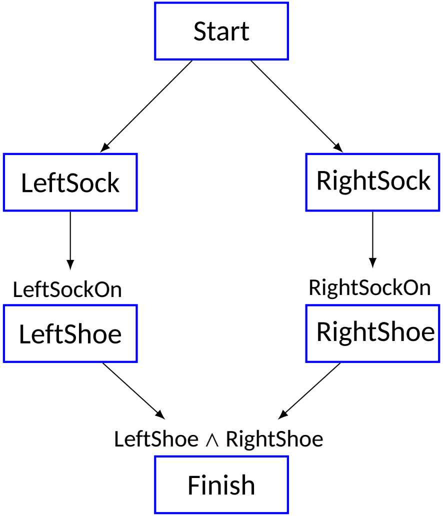
Shocks

- **Initial state : ∅**
- **Goal state: LeftShoeOn ∧ RightShoeOn**
- **Action - LeftSock**
  - **Precondition: ∅**
  - **Effect: LeftSockOn**
- **Action - RightSock**
  - **Precondition: ∅**
  - **Effect: RightSockOn**
- **Action - LeftShoe**
  - **Precondition: LeftSockOn**
  - **Effect: LeftShoeOn**
- **Action - RightSock**
  - **Precondition: RightSockOn**
  - **Effect: RightShoeOn**



Shocks

- Initial state : ∅
- Goal state: LeftShoeOn ∧ RightShoeOn
- Action - LeftSock
  - Precondition: ∅
  - Effect: LeftSockOn
- Action - RightSock
  - Precondition: ∅
  - Effect: RightSockOn
- Action - LeftShoe
  - Precondition: LeftSockOn
  - Effect: LeftShoeOn
- Action - RightSock
  - Precondition: RightSockOn
  - Effect: RightShoeOn



Partial order planning

- Basic idea: Make choices only that are relevant for solving the current part of the problem
- Least commitment choices
  - Ordering - Leave actions unordered, unless they must be sequential
  - Binding - Leave variable unbound, unless needed to unify with conditions being achieved
  - Actions - Usually not subjected to least commitment

Milk, Bananas, Book

Initial State:
Action: Start
Effect: At(Home) $\wedge$ Sells(BS,Book) $\wedge$ Sells(M,Milk) $\wedge$ Sells(M,Bananas)

Goal State:
Action: Finish
Precondition: Have(Book) $\wedge$ Have(Milk) $\wedge$ Have(Bananas) $\wedge$ At(Home)

Action: Go(y)
Precondition: At(x)
Effect: At(y) $\wedge$ $\neg$At(x)

Action: Buy(x)
Precondition: At(y) $\wedge$ Sells(y,x)
Effect: Have(x)

Milk, Bananas, Book

$\boxed{\text{Start}}$

At(Home) ∧ Sells(BS,Book) ∧ Sells(M,Milk) ∧ Sells(M,Bananas)

Milk, Bananas, Book

Start

At(Home) ∧ Sells(BS,Book) ∧ Sells(M,Milk) ∧ Sells(M,Bananas)

Have(Book) ∧ Have(Milk) ∧ Have(Bananas) ∧ At(Home)

Finish

Milk, Bananas, Book

Start

At(Home) ∧ Sells(BS,Book) ∧ Sells(M,Milk) ∧ Sells(M,Bananas)

At(BS) ∧ Sells(BS,Book)

Buy(book)

Have(Book) ∧ Have(Milk) ∧ Have(Bananas) ∧ At(Home)

Finish

Milk, Bananas, Book

Start

At(Home) ∧ Sells(BS,Book) ∧ Sells(M,Milk) ∧ Sells(M,Bananas)

At(BS) ∧ Sells(BS,Book)

Buy(book)

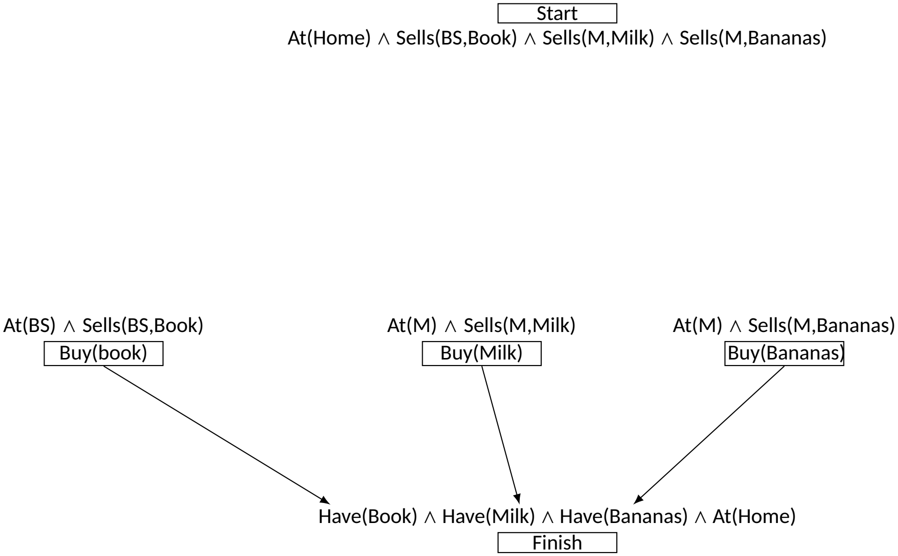
At(M) ∧ Sells(M,Milk)

Buy(Milk)

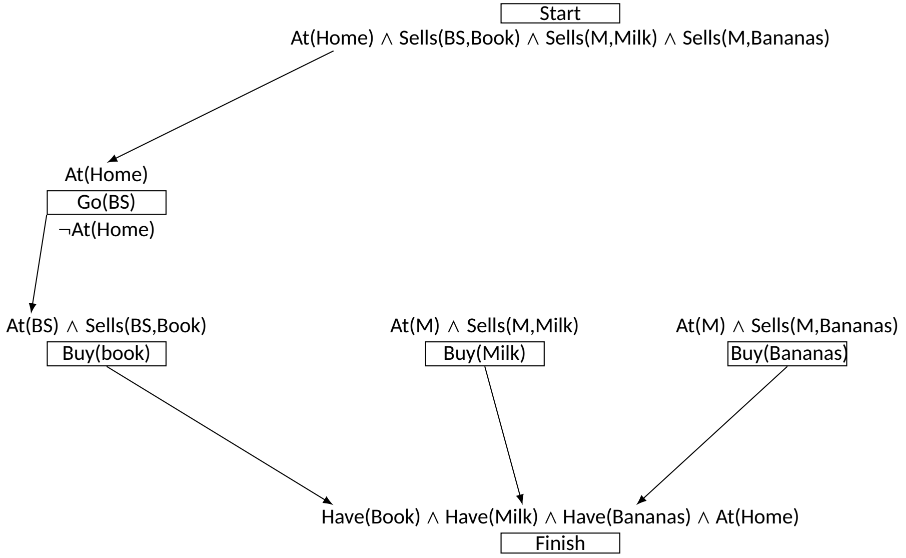
Have(Book) ∧ Have(Milk) ∧ Have(Bananas) ∧ At(Home)

Finish

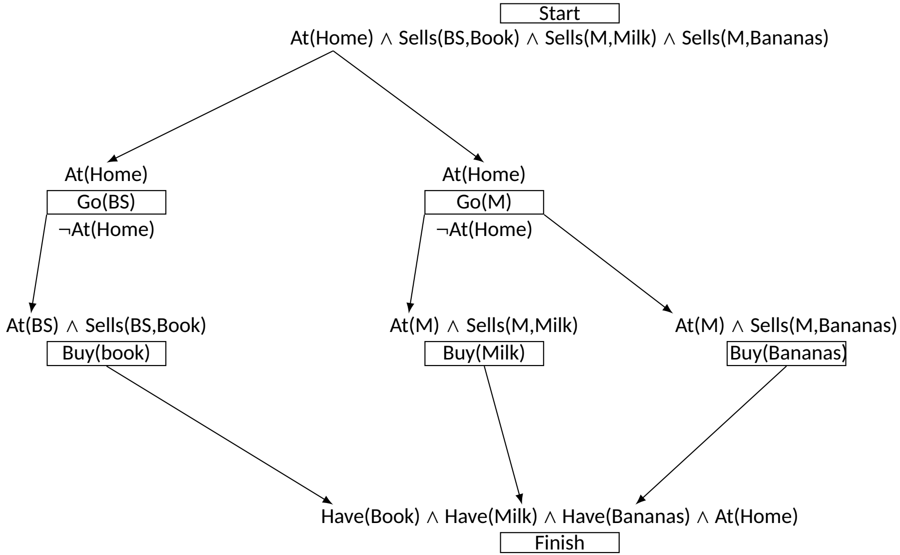
Milk, Bananas, Book



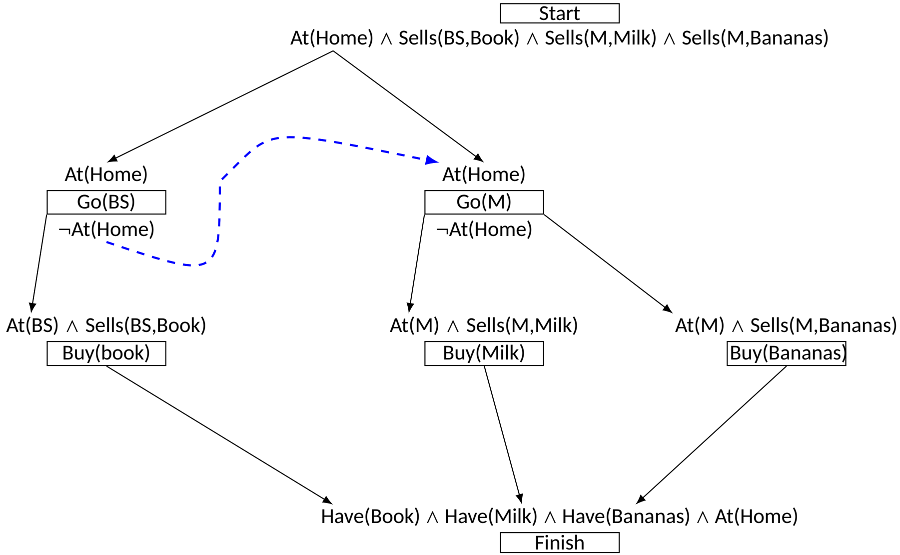
Milk, Bananas, Book



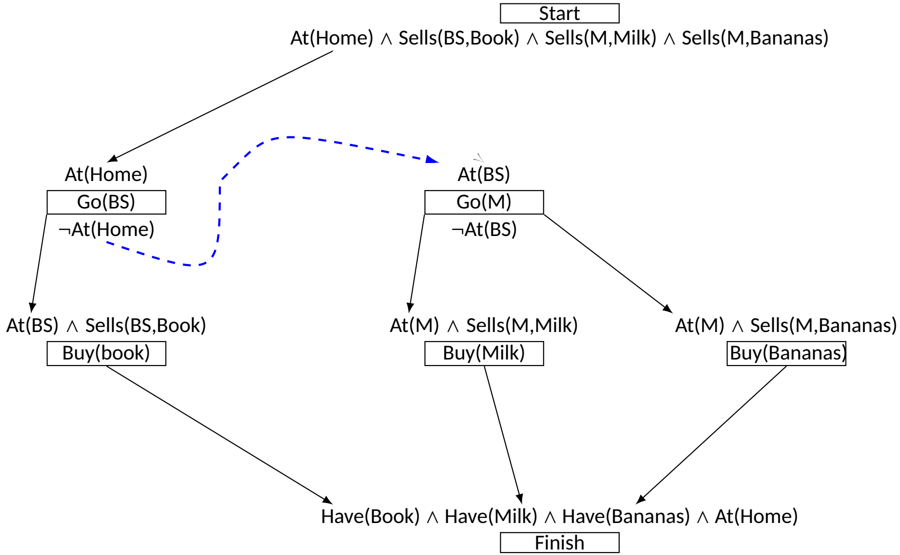
Milk, Bananas, Book



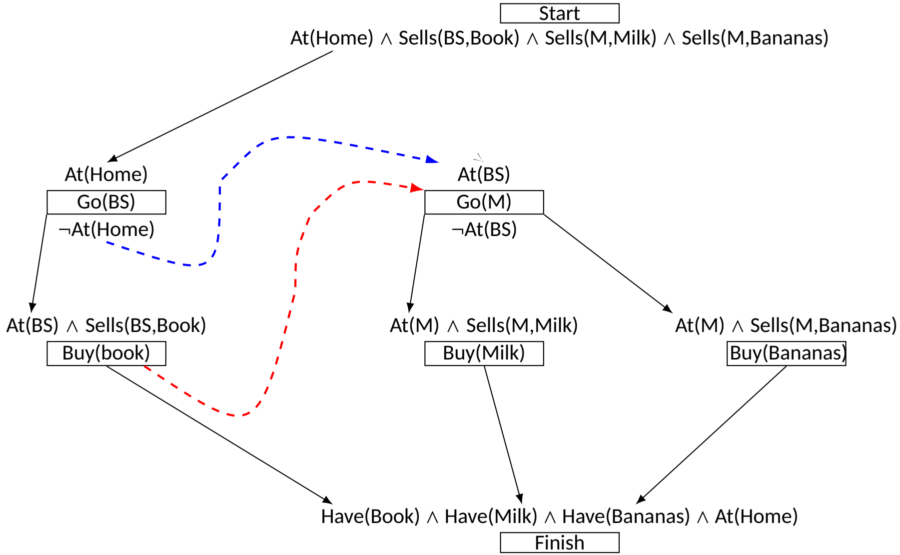
Milk, Bananas, Book



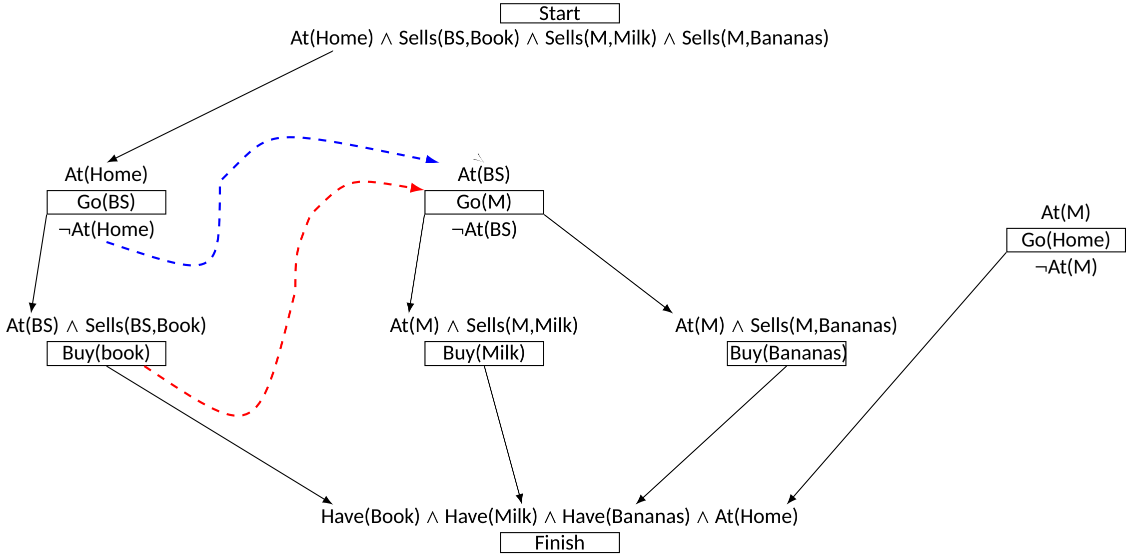
Milk, Bananas, Book



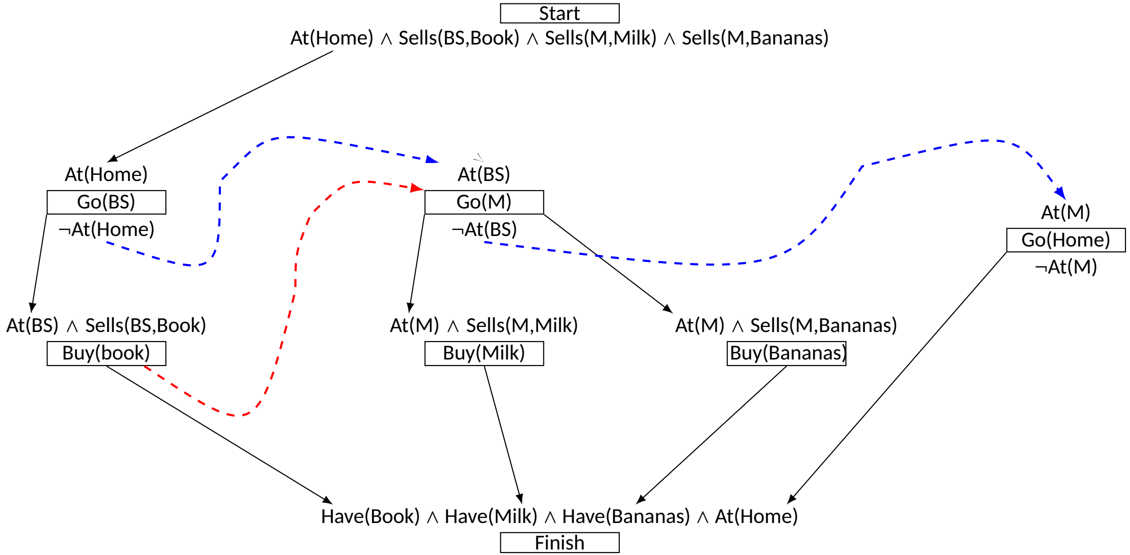
Milk, Bananas, Book



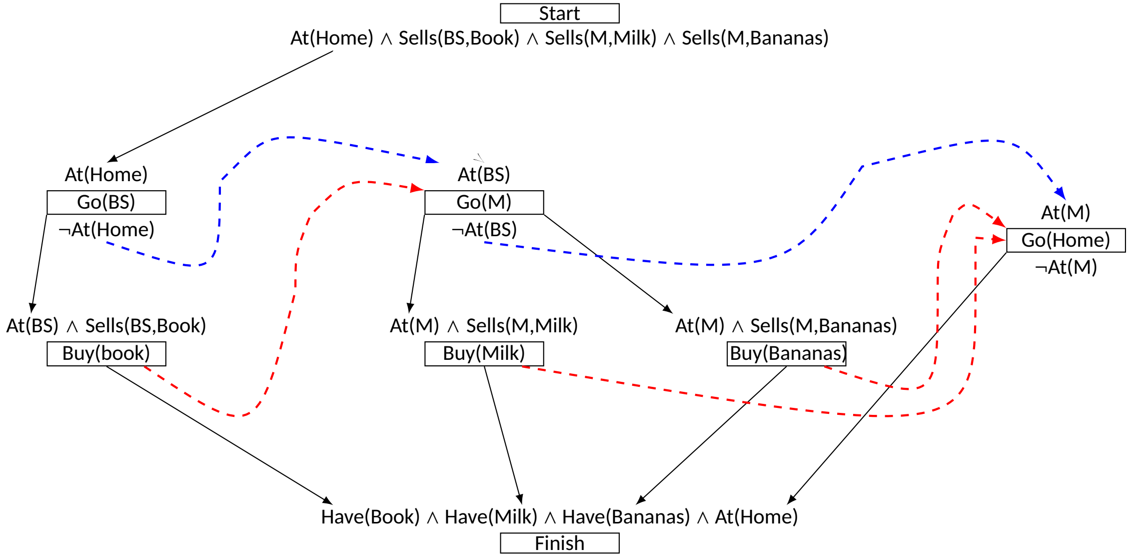
Milk, Bananas, Book



Milk, Bananas, Book



Milk, Bananas, Book



Planning Graphs

- Consists of a sequence of levels that correspond to time steps in the plan
- Each level contains a set of actions and a set of literals that could be true at that time step depending on the actions taken in previous time steps
- For every +ve and -ve literal C, we add a persistence action with precondition C and effect C

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) $\wedge$ Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0

Have(Cake)

¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0 A_0

Have(Cake)

¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0 A_0

Have(Cake)

$\boxed{\textbf{Eat(Cake)}}$

¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0 A_0

Have(Cake)

Eat(Cake)

¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0 A_0 S_1

Have(Cake)

Eat(Cake)

¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

S_0 A_0 S_1

Have(Cake)

¬Have(Cake)

$\boxed{\textbf{Eat(Cake)}}$

Eaten(Cake)

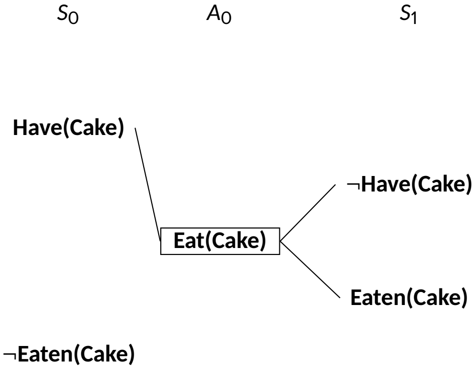
¬Eaten(Cake)

Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

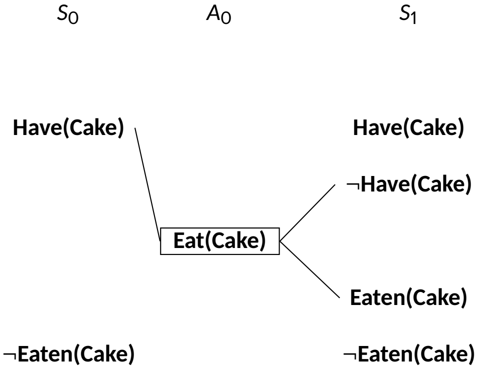


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

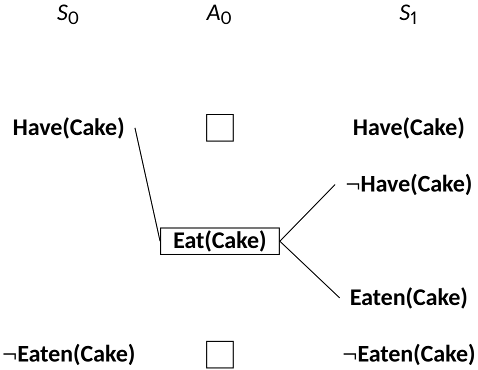


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

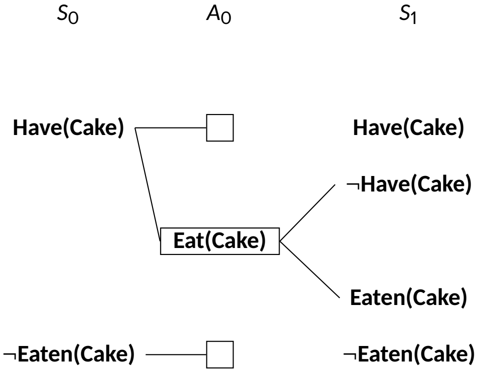


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

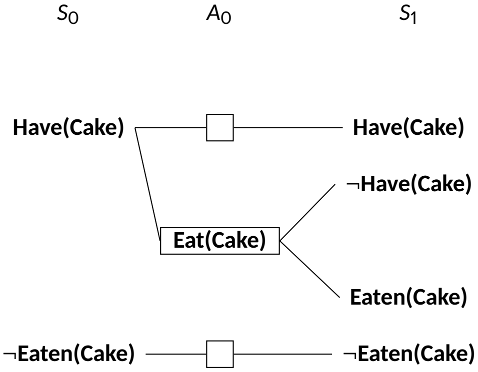


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

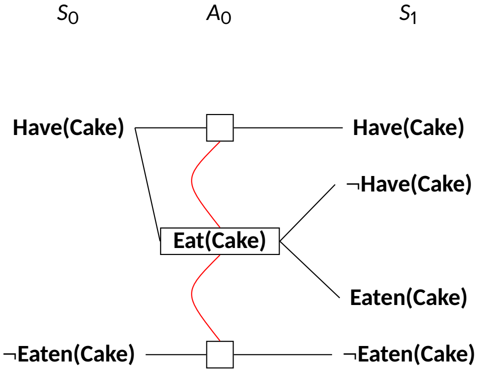


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

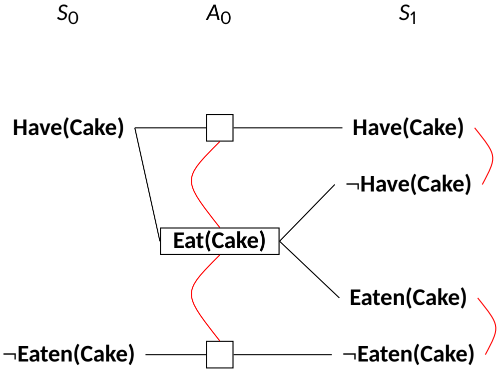


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

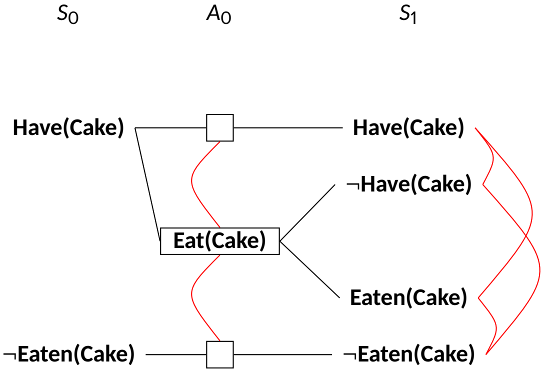


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

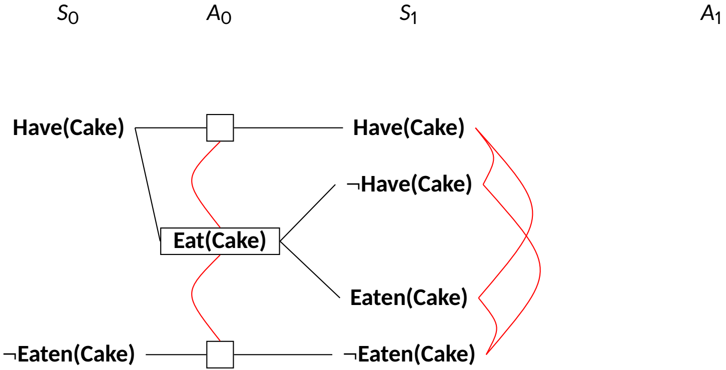


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) $\wedge$ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) $\wedge$ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

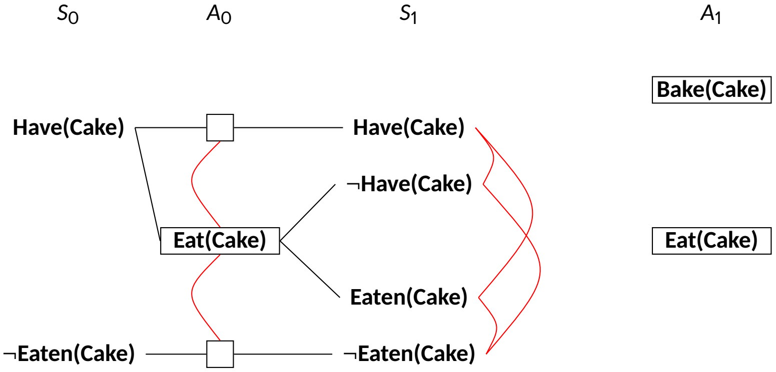


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

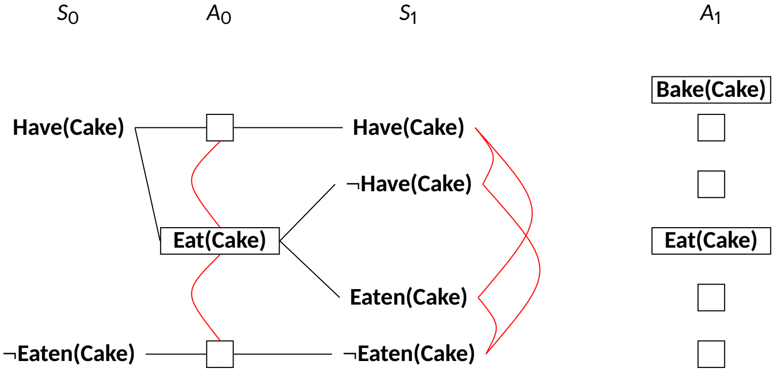


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

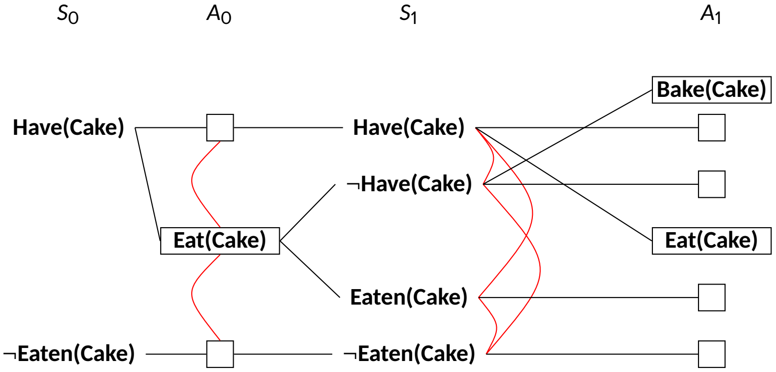


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

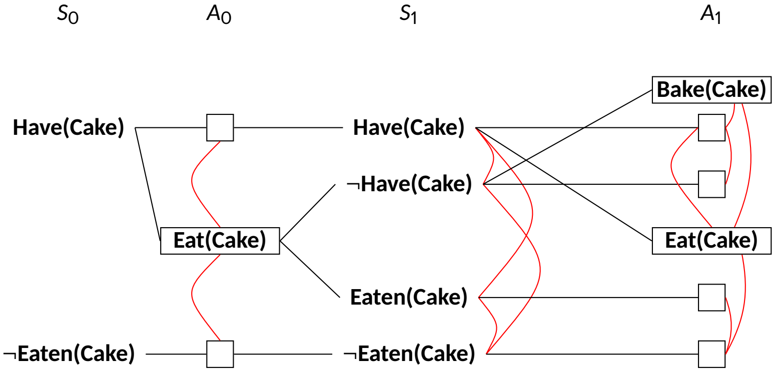


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

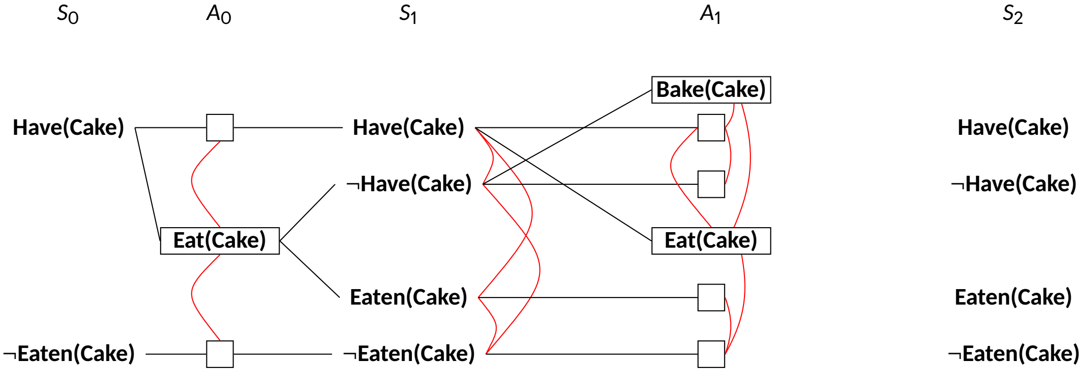


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

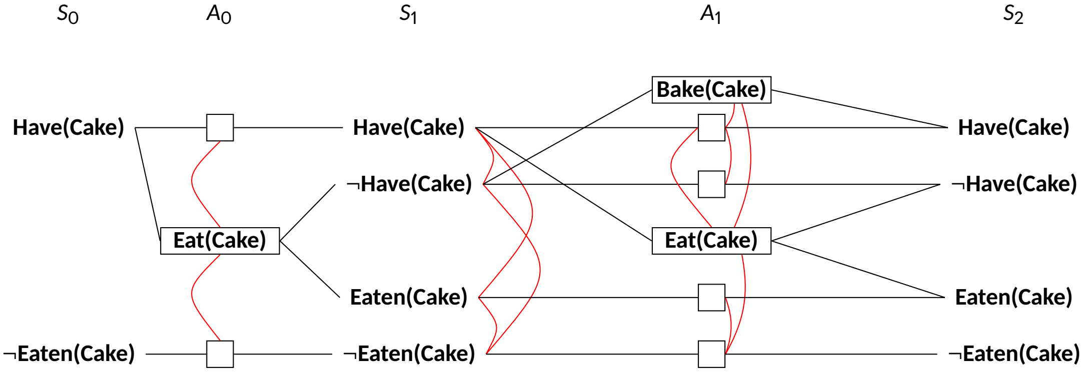


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)

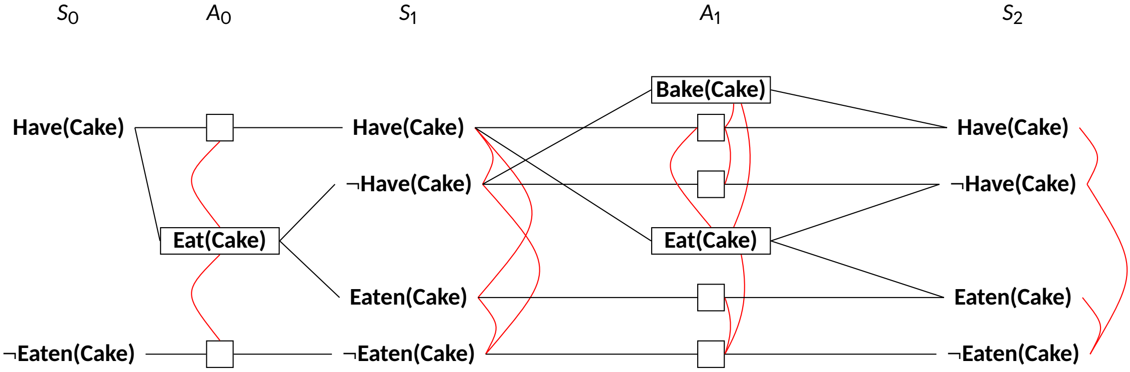


Planning Graph

Start: Have(Cake)
Finish: Have(Cake) ∧ Eaten(Cake)

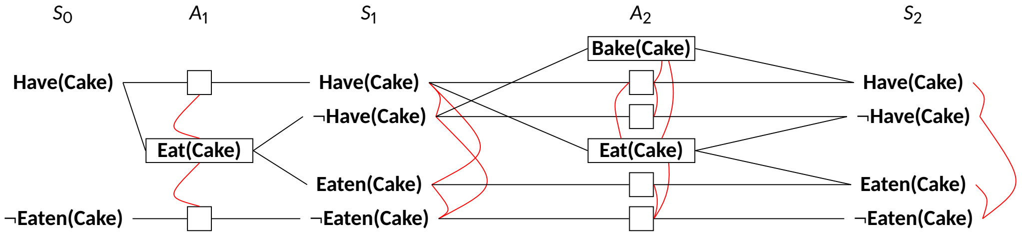
Action: Eat(Cake)
Precondition: Have(Cake)
Effect: Eaten(Cake) ∧ ¬Have(Cake)

Action: Bake(Cake)
Precondition: ¬Have(Cake)
Effect: Have(Cake)



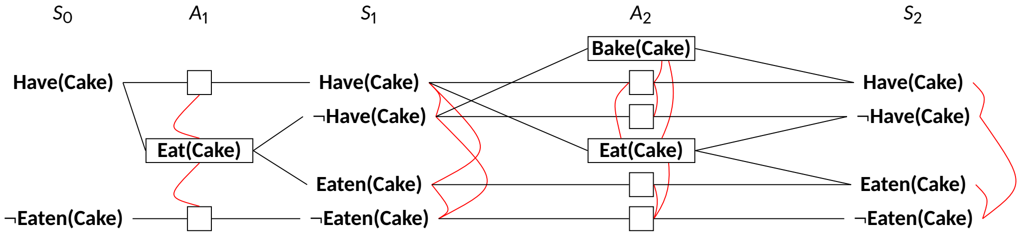
Mutex actions

- Mutual exclusion relation exists between two actions if
  - Inconsistent effects - once action negates an effect of the other
    - Eat(Cake) causes ¬Have(Cake) and Bake(Cake) causes Have(Cake)
  - Interference - one of the effects of one action is the negation of a precondition of the other
    - Eat(Cake) causes ¬Have(Cake) and the persistence of Have(Cake) needs Have(Cake)
  - Competing needs - one of the preconditions of one action is mutually exclusive with a precondition of the other
    - Bake(Cake) needs ¬Have(Cake) and Eat(Cake) needs Have(Cake)



Mutex literals

- **Mutual exclusion relation exists between two literals if**
  - **One is the negation of the other, OR**
  - **Each possible pair of actions that could achieve the two literals is mutually exclusive (inconsistent support)**



GraphPLAN algorithm

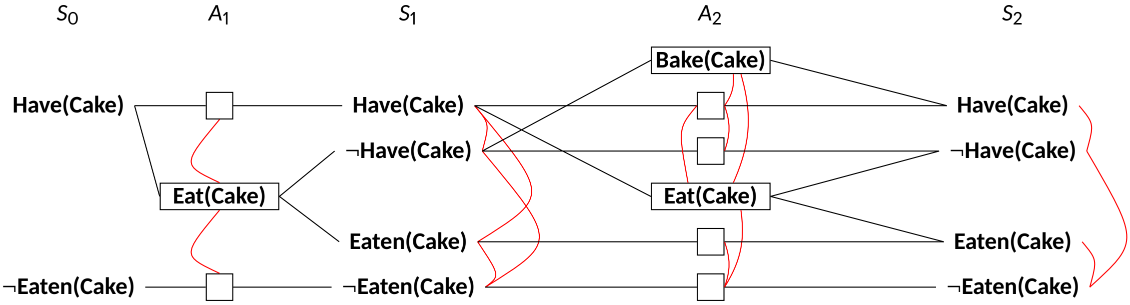
```
Function GraphPlan
graph ← Initial-Planning-Graph( problem )
goals ← Goals[ problem ]
do
    if goals are all non-mutex in last level of graph then do
       solution ← Extract-Solution( graph )
       if solution ← failure then return solution
       else if No-Solution-Possible (graph )
          then return failure
    graph ← Expand-Graph( graph, problem )
```

Termination

- Termination when no plan exists
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotonically

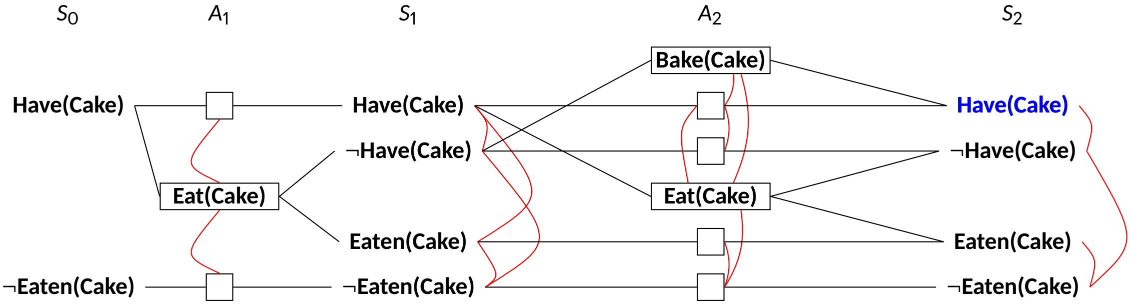
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



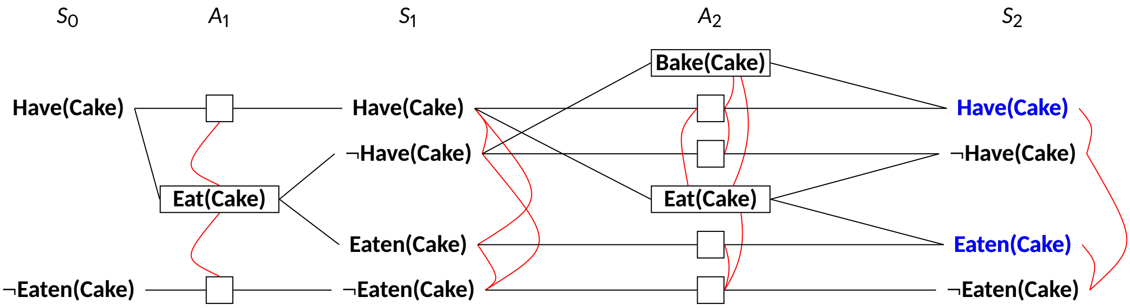
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



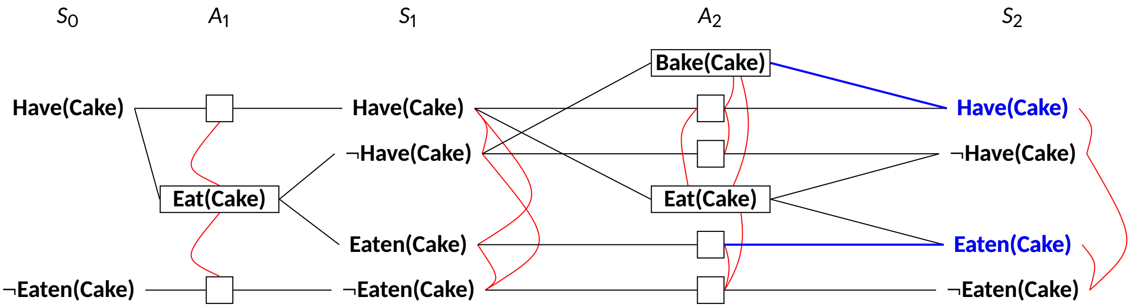
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



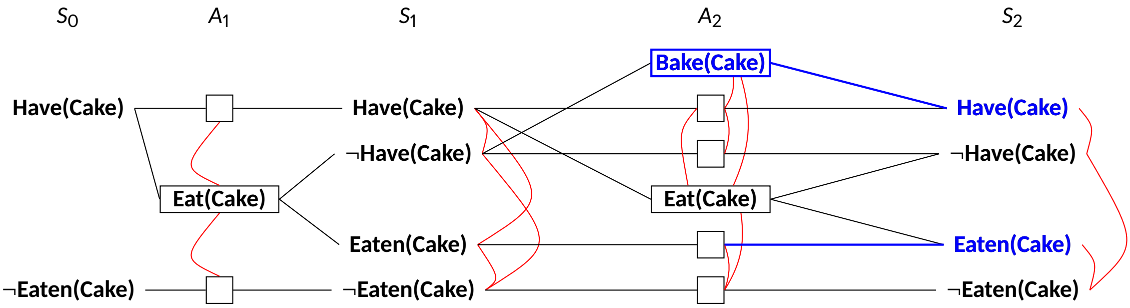
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



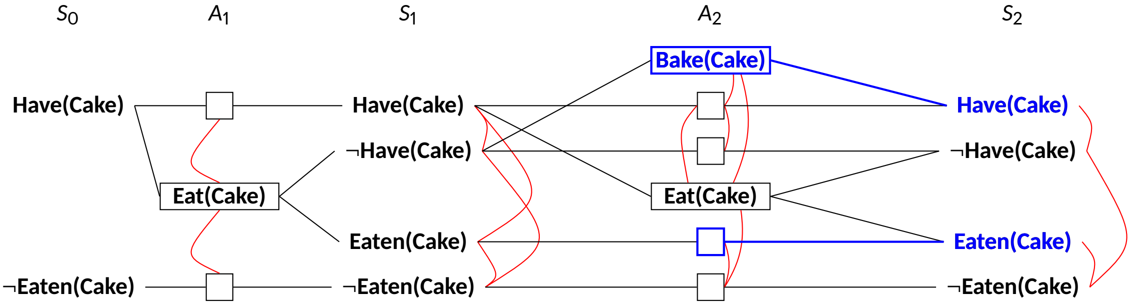
Finding the plan

- **Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes**
- **Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.**



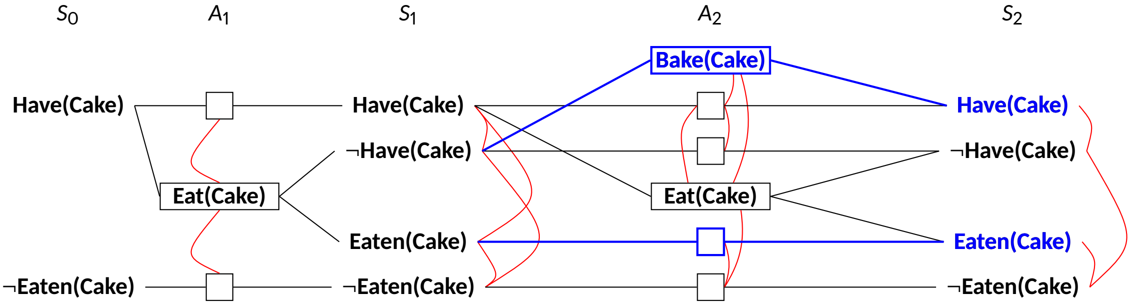
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



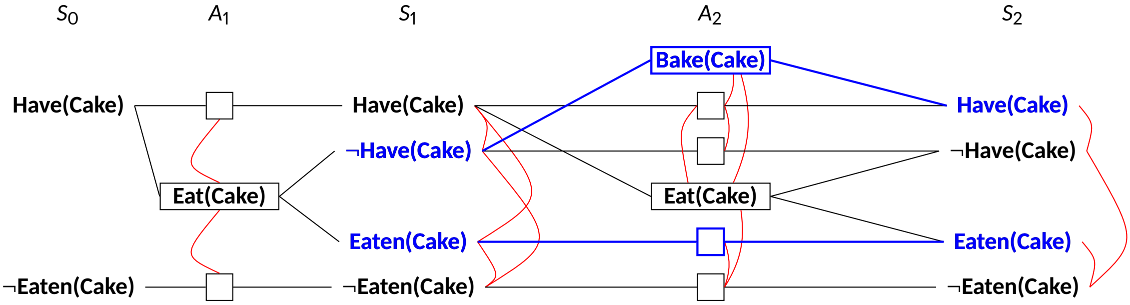
Finding the plan

- **Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes**
- **Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.**



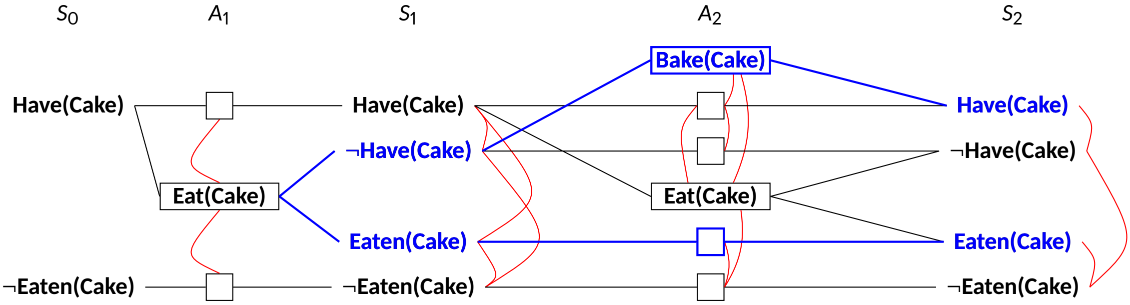
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



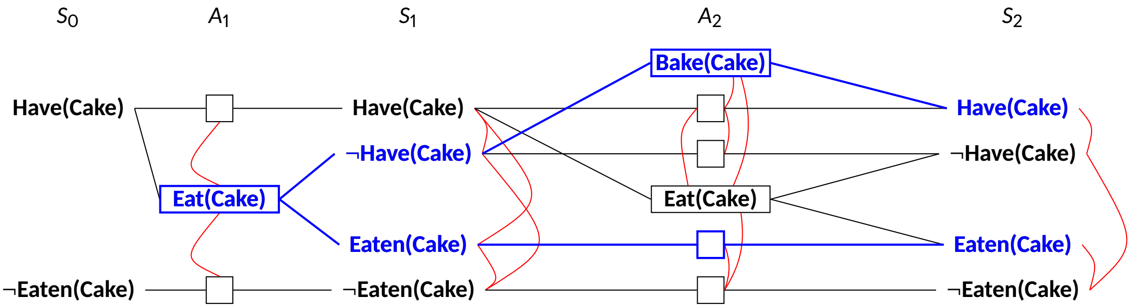
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



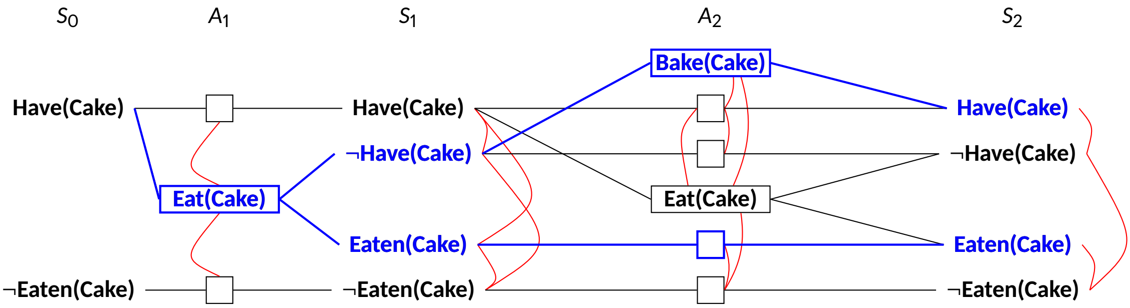
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



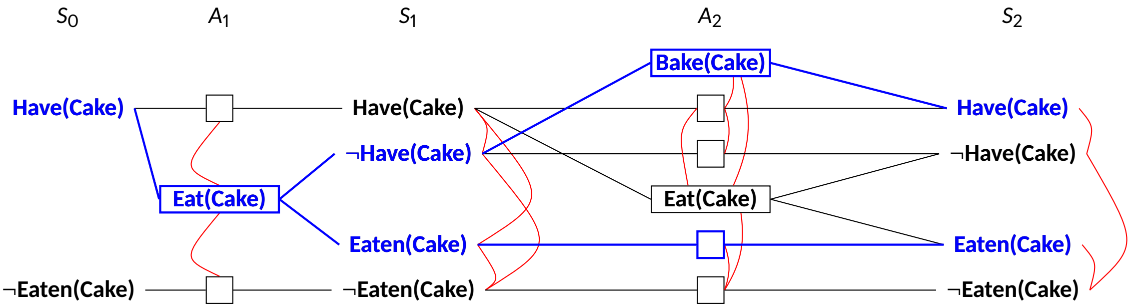
Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



Finding the plan

- Once a world is found having all goal predicates without mutexes, the plan can be extracted by solving a constraint satisfaction problem (CSP) for resolving the mutexes
- Creating the planning graph can be done in polynomial time, but planning is known to be a PSPACE-complete problem. The hardness is in the CSP.



Planning with Propositional Logic

- The planning problem is translated into a CNF satisfiability problem
- The goal is asserted to hold at a time step T, and clauses are included for each time step up to T.
- If the clauses are satisfiable, then a plan is extracted by examining the actions that are true.
- Otherwise, we increment T and repeat

- Constructing formulas to encode bounded planning problems into satisfiability problems
  - If f is a fluent $At(M)$, we write $At(M, i)$ as f_i, i denotes time stamp
  - If a is an action $Move(A, B)$, we write $Move(A, B, i)$ as a_i.
  - Notations: PC - precondition, E - effects, E^+ - effects in the +ve form, E^- - effect in the -ve form, s_0 - start state, g - goal state, g^+ - literals in +ve form in goal state, g^- - literals in -ve form in goal state, A - set of actions

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**
- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0 \right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0 \right)$

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**
- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0 \right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0 \right)$
- **Goal state:**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$

- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**
- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**
  - $C_3 : a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1}\right)$

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0 \right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0 \right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T \right) \wedge \left(\bigwedge_{f \in g^-} f_T \right)$
- **Action**
  - $C_3 : a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1} \right)$

- **An action changes only the fluents that are in its effects.**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**
  - $C_3 : a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1}\right)$

- **An action changes only the fluents that are in its effects.**
  - $C_4 : \left(\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{\{a \in A | f_i \in E^+(a_i)\}} a_i\right)\right) \wedge$
    $\left(f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{\{a \in A | f_i \in E^-(a_i)\}} a_i\right)\right)$
  - **Explanatory frame axioms - set of propositions that enumerate the set of actions that could have occurred in order to account for a state change.**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**
  - $C_3 : a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1}\right)$

- **An action changes only the fluents that are in its effects.**
  - $C_4 : \left(\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{\{a \in A | f_i \in E^+(a_i)\}} a_i\right)\right) \wedge$
    $\left(f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{\{a \in A | f_i \in E^-(a_i)\}} a_i\right)\right)$
  - **Explanatory frame axioms - set of propositions that enumerate the set of actions that could have occurred in order to account for a state change.**
- **Complete exclusion axiom - only one action occurs at each step.**

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1: \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2: \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**
  - $C_3: a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1}\right)$

- **An action changes only the fluents that are in its effects.**
  - $C_4: \left(\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{\{a \in A \mid f_i \in E^+(a_i)\}} a_i\right)\right) \wedge$
    $\left(f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{\{a \in A \mid f_i \in E^-(a_i)\}} a_i\right)\right)$
  - **Explanatory frame axioms - set of propositions that enumerate the set of actions that could have occurred in order to account for a state change.**
- **Complete exclusion axiom - only one action occurs at each step.**
  - $C_5: \neg a_i \vee \neg b_i$

SAT encoding

- **Formula is built with these five kinds of sets of formulas:**

- **Initial state:**
  - $C_1 : \left(\bigwedge_{f \in s_0} f_0\right) \wedge \left(\bigwedge_{f \notin s_0} \neg f_0\right)$
- **Goal state:**
  - $C_2 : \left(\bigwedge_{f \in g^+} f_T\right) \wedge \left(\bigwedge_{f \in g^-} f_T\right)$
- **Action**
  - $C_3 : a_i \implies \left(\bigwedge_{p \in PC(a)} p_i \wedge \bigwedge_{e \in E(a)} e_{i+1}\right)$

- **An action changes only the fluents that are in its effects.**
  - $C_4 : \left(\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{\{a \in A \mid f_i \in E^+(a_i)\}} a_i\right)\right) \wedge \left(f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{\{a \in A \mid f_i \in E^-(a_i)\}} a_i\right)\right)$
  - **Explanatory frame axioms - set of propositions that enumerate the set of actions that could have occurred in order to account for a state change.**
- **Complete exclusion axiom - only one action occurs at each step.**
  - $C_5 : \neg a_i \vee \neg b_i$
- **Need to check satisfiability of** $C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$

Excercise

- **Consider a simple example where we have one robot r and two locations l_1 and l_2. Let us suppose that the robot can move between the two locations. In the initial state, the robot is at l_1; in the goal state, it is at l_2. The operator that moves the robot is: Action: $move(r, l, l')$, Precond: $At(r, l)$, Effects: $At(r, l')$, $\neg At(r, l)$. In this planning problem, a plan of length 1 is enough to reach the goal state. Write the constraints.**

Summary

- Search involving logic along with change of state
- We looked into planning problem where the environment is fully observable, deterministic and static
- We looked into planning graph and SAT based planning
- Application domains - robotics, autonomous systems, etc.

Thank you!