CMU 15-781

Lecture 6: Planning II

Teacher:
Gianni A. Di Caro

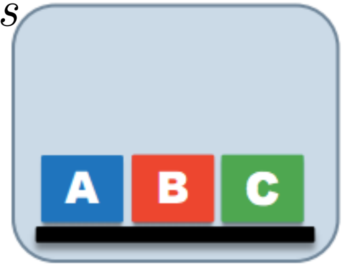
Recap: Classical Planning

- **Factored representation:** A state of the world is represented by a collection of variables → Exploit structure, sub-goaling / divide-and-conquer, domain-independent heuristics
- **PDDL / STRIPS:** Language expressive enough to describe a wide variety of problems, but restrictive enough to allow efficient algorithms to operate over it
- **State:** Conjunction of *literals*

Recap: Classical Planning

- **State:** Conjunction of *literals*
  - *Propositional literals:* Poor $\wedge$ Unknown
  - *Ground first order literals*: At(Plane$_1$, Rome) $\wedge$ At(Plane$_2$, Tokyo) ~~At(x, Rome) $\wedge$ At(y, Tokyo)~~
  - *Function-free:* ~~At(Father(Tom), NY)~~ → At(Alex, NY) $\wedge$ Father(Alex, Tom)
  - Closed-world assumption: Any condition which is not mentioned in the state is assumed to be *false*

The world is represented through a set of *features/objects* (e.g., planes, people, cities) and each literal states a *fact* that attributes "values" to features



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

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

Recap: Classical Planning

- **Goals:** A conjunction of literals, $\mathrm{At}(P_1, \mathrm{JFK}) \wedge \mathrm{At}(P_2, \mathrm{SFO})$, that may also contain variables, such as $\mathrm{At}(p, \mathrm{JFK}) \wedge \mathrm{Plane}(p)$, meaning that the goal is to have *any* plane at JFK

- The aim is to reach a state that *entails* a goal: OnTable(A) $\wedge$ OnTable(B) $\wedge$ OnTable(D) $\wedge$ On(C, D) $\wedge$ Clear(A) $\wedge$ Clear(B) $\wedge$ Clear(C) satisfies the goal to stack C on D

- → **A goal g is a conjunction of *sub-goals*!**
  $\mathbf{g} = g_1 \wedge g_2 \wedge \ldots \wedge g_n$

- Goals are reached through *sequence of actions* (the plan)

Recap: Classical Planning

- **Actions:** *Preconditions + Effects (Postconditions)*
- **Action schema:** a number of different actions that can be derived by universal quantification of the variables, e.g., an action schema to fly a plane from one location to another:

  $Action(Fly(p, from, to),$

  PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

  EFFECT: $\neg At(p, from) \wedge At(p, to))$
- An action is applicable in state s if s *entails the preconditions*
- The literals negated by the effect of a are removed from s, while the positive literals resulting from a are added to s

Recap: Classical Planning

- RESULT(s,a) = $(s$ – DELETE$(a))$ $\cup$ ADD(a)

- **Action schema:**

  $Action(Name(p_1, p_2, \ldots, p_n),$

  PRECONDITIONS: $L_1(p) \wedge L_2\ (p) \wedge \ldots \wedge L_m(p)$

  ADD-LIST: $\{A_1(p), A_2(p), \ldots, A_q\ (p)\}$

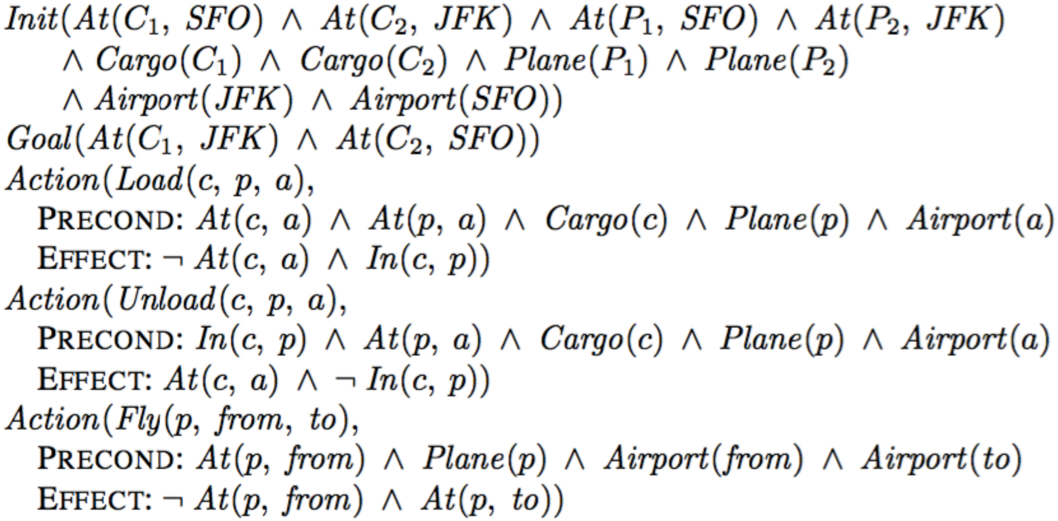
  DELETE-LIST: $\{L_i(p), L_j(p) \wedge \ldots \wedge L_k(p)\}$

RECAP: CLASSICAL PLANNING

- **Planning domain:** *Set of Action schemas (+ Set of Predicates)*
- **Planning problem (instance):** *Planning domain + Initial state + Goal + Set of Objects (world features)*
- **Solution of the planning problem:** A sequence of actions that, starting from the initial state, end in a state *s* that entails the goal

Air cargo transportation problem (from R&N)

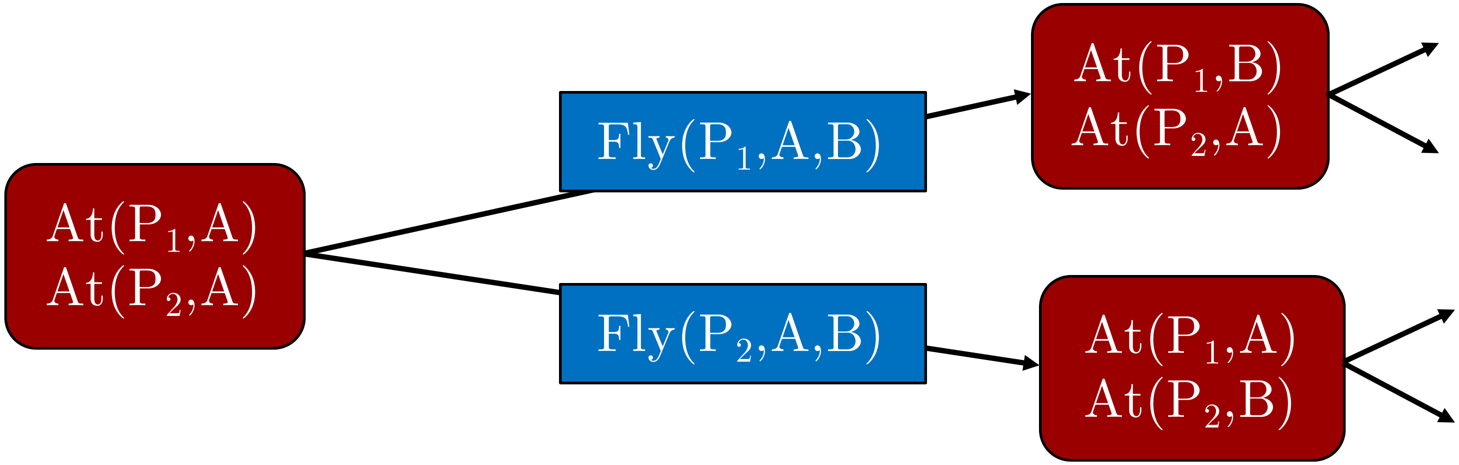
- *Predicates:* At, Cargo, Plane, Airport, In
- *Objects:* C_1, C_2, P_1, P_2, SFO, JFK
- *Actions:* Load, Unload, Fly



$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land\ Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land\ Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
$\quad$ PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
$\quad$ PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
$\quad$ PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p, from) \land At(p, to))$

Planning as search

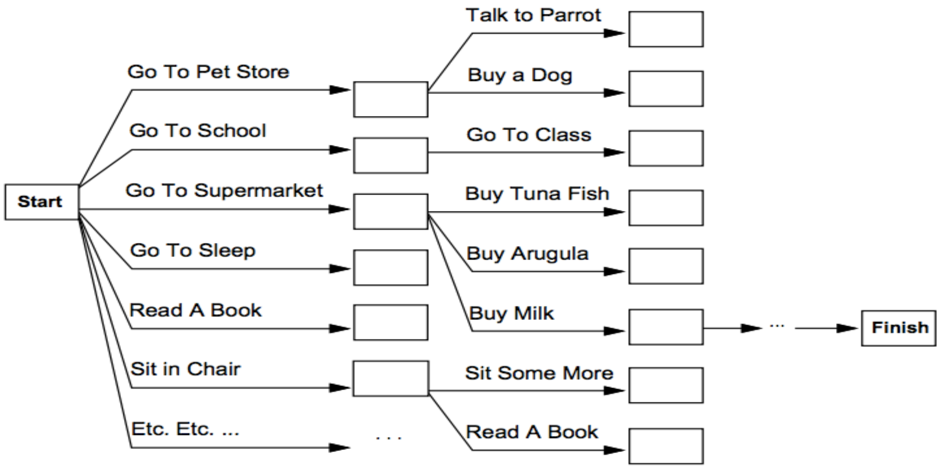
- (Forward) Search from initial state to goal
- Can use *standard search techniques*, including heuristic search



(FORWARD) STATE-SPACE SEARCH

- In absence of function symbols, the state space of a planning problem is finite → Any graph search algorithm that is complete will be a *complete planning algorithm*
- *Irrelevant action problem:* All applicable actions are considered at each state!
- The resulting *branching factor b* is typically large and the *state space is exponential in b* → Needs for good **heuristics**!

At home →
get milk, bananas and a cordless drill
→ return home



(Forward) State-space search

- *Air Cargo Example*
- Initial state: 10 airports, each airport has 5 planes and 20 pieces of cargo
- Goal: transport all the cargos at airport A to airport B
- Solution: load the 20 pieces of cargo at A into one of the planes at A and fly it to B
- Avg Branching factor b: each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded), or loaded into any plane at its airport (if it is unloaded)
- Number of states to explore: $O(b^d)$ ~ 2000^{41}

Find a Heuristic: Relax the problem

- **Define a Relaxed problem:**
  - (Potentially) Easy to solve
  - The solution gives admissible heuristics for A*
- **Relaxation: Remove all preconditions from actions**
- → Every action will always be applicable, and any literal (sub-goal) can be achieved in one step
- → *Adding edges to the graph:* including forbidden actions
- → $h(x) =$ The number of steps required to get to the goal is the number of unsatisfied goals from current state x?

Domain-Independent Heuristic

- $h(x)$ = The number of steps required to solve a conjunction of goals is the number of unsatisfied goals from current state x?
- Impossible to derive such a heuristic with atomic states! The successor function is a black box, here we exploit the structure of the representation
- The heuristic is **domain-independent!**
- With atomic states, in general only *domain-specific* heuristics are possible

Heuristic: Ignore preconditions

- Complications, that could made the heuristic function $h(x)$ not admissible:

  a. Some operations achieve multiple goals

  b. Some operations undo the effects of others

- Poll 1: To get an admissible heuristic, ignore preconditions and, in addition ignore:

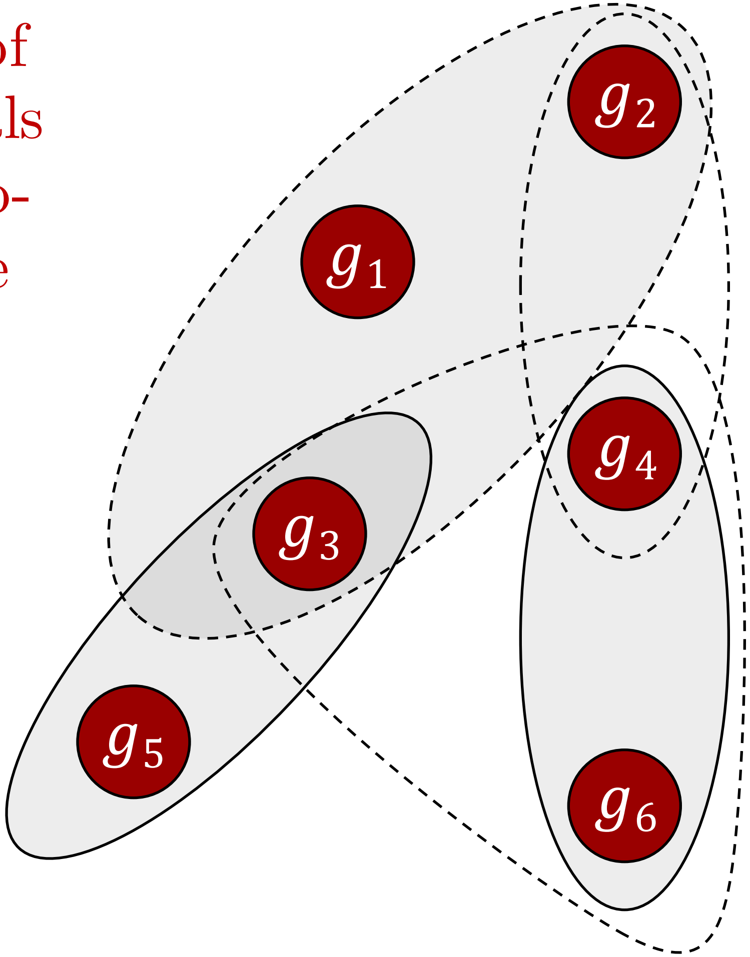
  1. Just a

  [✓ 2.] Just b

  3. Both a and b

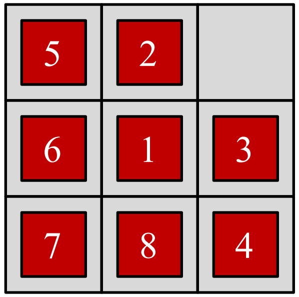
Ignore preconditions & Non-Goal Effects

- To avoid b. remove all the effects of actions, except those that are literals $g_{i,}$ $i=1,\ldots,n,$ in the goal g (i.e., sub-goals) → Exploit factored structure
- $h(x)$ = the min number of actions such that the union of their effects contains all n sub-goals g_i → Admissible
- Computing $h(x)$ = solving a Set Cover problem: NP-hard!
- Greedy log n approximation:
  - Admissibility is lost!



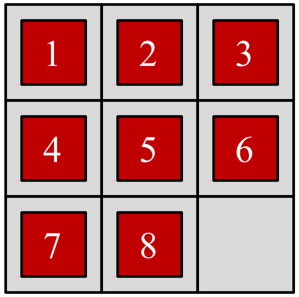
IGNORE (SPECIFIC) PRECONDITIONS

- Ignore specific preconditions to derive *domain-specific* heuristics
- Sliding block puzzle, $move(t, s_1, s_2)$ action:
- $\mathrm{On}(t, s_1) \wedge \mathrm{Blank}(s_2) \wedge \mathrm{Adjacent}(s_1, s_2) \Rightarrow \mathrm{On}(t, s_2) \wedge \mathrm{Blank}(s_1) \wedge \neg \mathrm{On}(t, s_1) \wedge \neg \mathrm{Blank}(s_2)$
- Consider two options for removing specific preconditions from $move()$
  a. Removing $\mathrm{Blank}(s_2) \wedge \mathrm{Adjacent}(s_1, s_2)$
  b. Removing $\mathrm{Blank}(s_2)$
- Poll 2: Match option to heuristic:
  1. a↔ ∑Manhattan, b↔#misplaced tiles
  [✓ 2.] a↔#misplaced tiles, b↔ ∑Manhattan
  3. b↔#misplaced tiles, a is inadmissible
  4. b↔ ∑Manhattan, a is inadmissible



5	2	
6	1	3
7	8	4

Example state

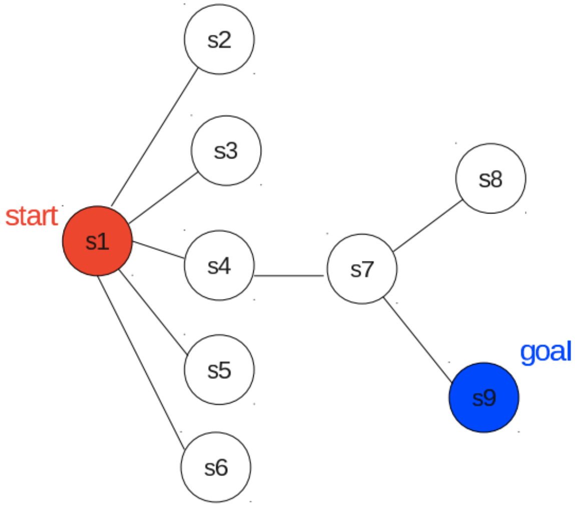


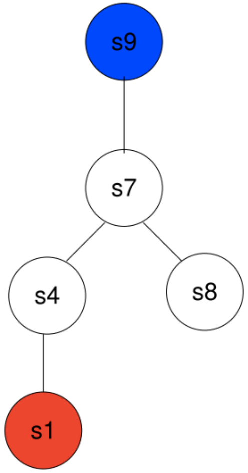
1	2	3
4	5	6
7	8	

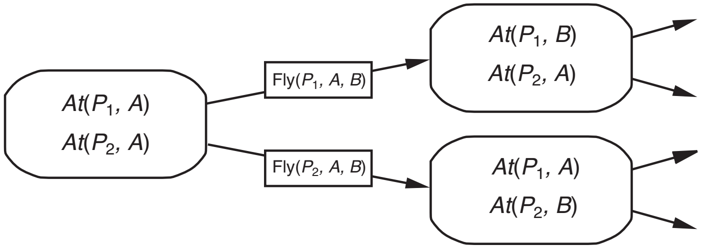
Goal state

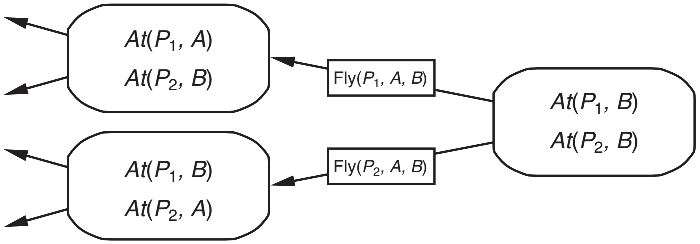
Backward State-space search

- Searching from a goal state to the initial state (**regression**)
- We only need to consider actions that are relevant to the goal (or current state) → **Relevant-state search**
- This can makes a strong reduction in branching factor, such that it could be more efficient than forward (progression) search
- "*Imagine trying to figure out how to get to some small place with few traffic connections from somewhere with a lot of traffic connections*"









Backward State-space search

- Regression from a (goal) state g over the action a gives state g'
  - $g' = (g - \text{ADD}(a)) \cup \text{Preconditions}(a)$

- DEL(a) doesn't appear: we don't know whether the literals negated by DEL(a) were true or not before a, therefore nothing can be said about them
- Variables can be included, such that a *set* of states is defined:
  - Goal $\text{At}(C_2, \text{SFO}) \rightarrow \text{Unload}(C_2, \text{p}, \text{SFO}) \rightarrow \text{g}' = \text{In}(C_2,\text{p}) \wedge \text{At}(\text{p}, \text{SFO}) \wedge \text{Cargo}(C_2) \wedge \text{Plane}(\text{p}) \wedge \text{Airport}(\text{SFO})$

Backward State-space search

- How to select actions?
- Relevant actions only
  - **Have an effect which is in the set of (current) goal literals**

    Goal: $At(C_1, JFK) \wedge At(C_2, SFO) \rightarrow Unload(C_2, p, SFO)$ is relevant, $Fly(p, JFK, SFO)$ is not relevant
- Consistent actions only
  - **Have no effect which negates an element of the goal**

    Goal: $A \wedge B \wedge C$, action a with effect $A \wedge B \wedge \neg C$ is not relevant

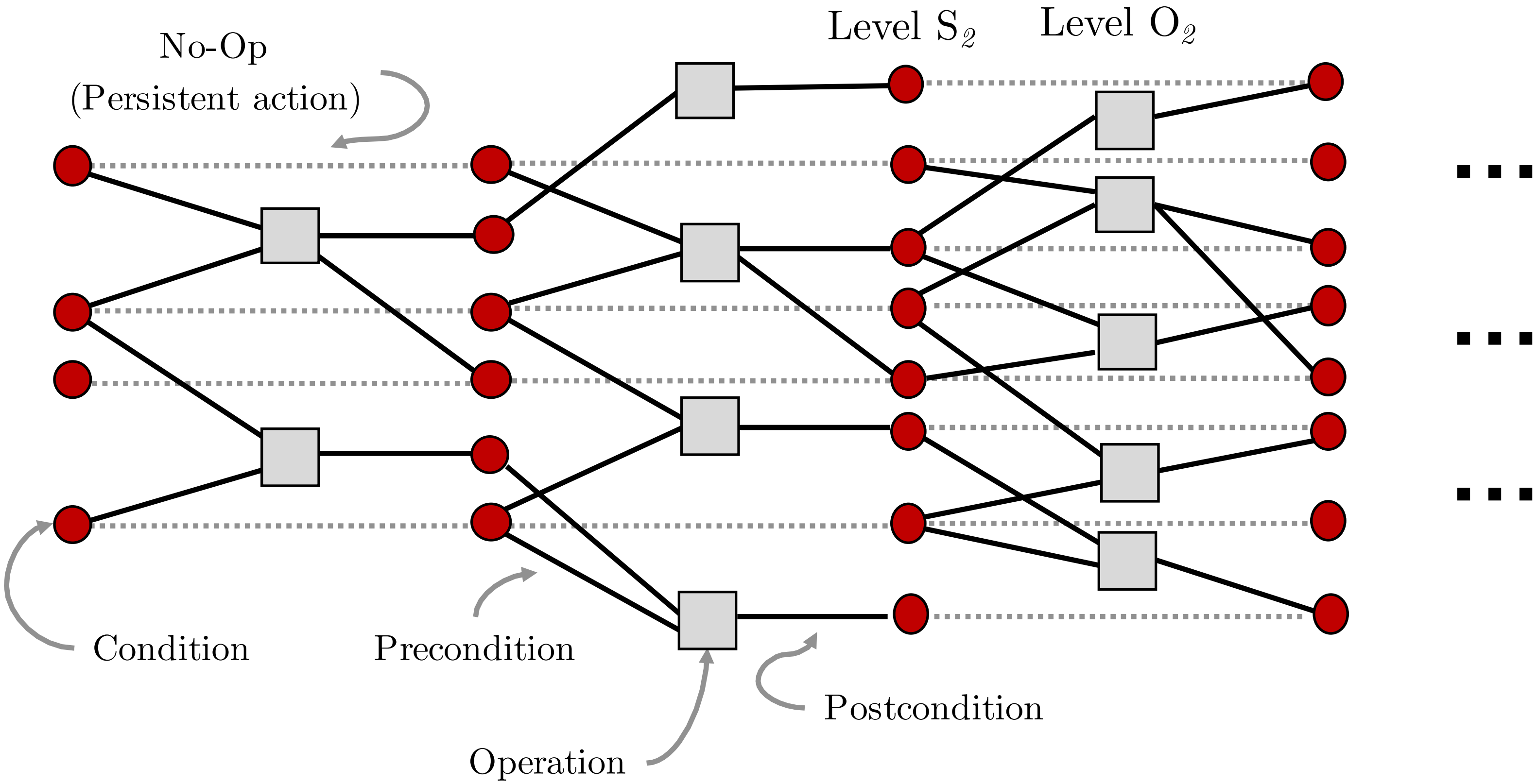
PLANNING GRAPHS

- Graph-based data structure representing a polynomial-size/time approximation of the exponential search tree
- Can be used to automatically produce good heuristic estimates (e.g., for A*)
- Can be used to search for a solution using the GRAPHPLAN algorithm

Planning graphs

- Leveled graph: vertices organized into levels/stages, with edges only between levels
- Two types of *vertices* on alternating levels:
  - Conditions
  - Operations
- Two types of *edges*:
  - Precondition: from condition to operation
  - Postcondition: from operation to condition

Generic planning graph*



* Slide based on Brafman

Planning graph construction

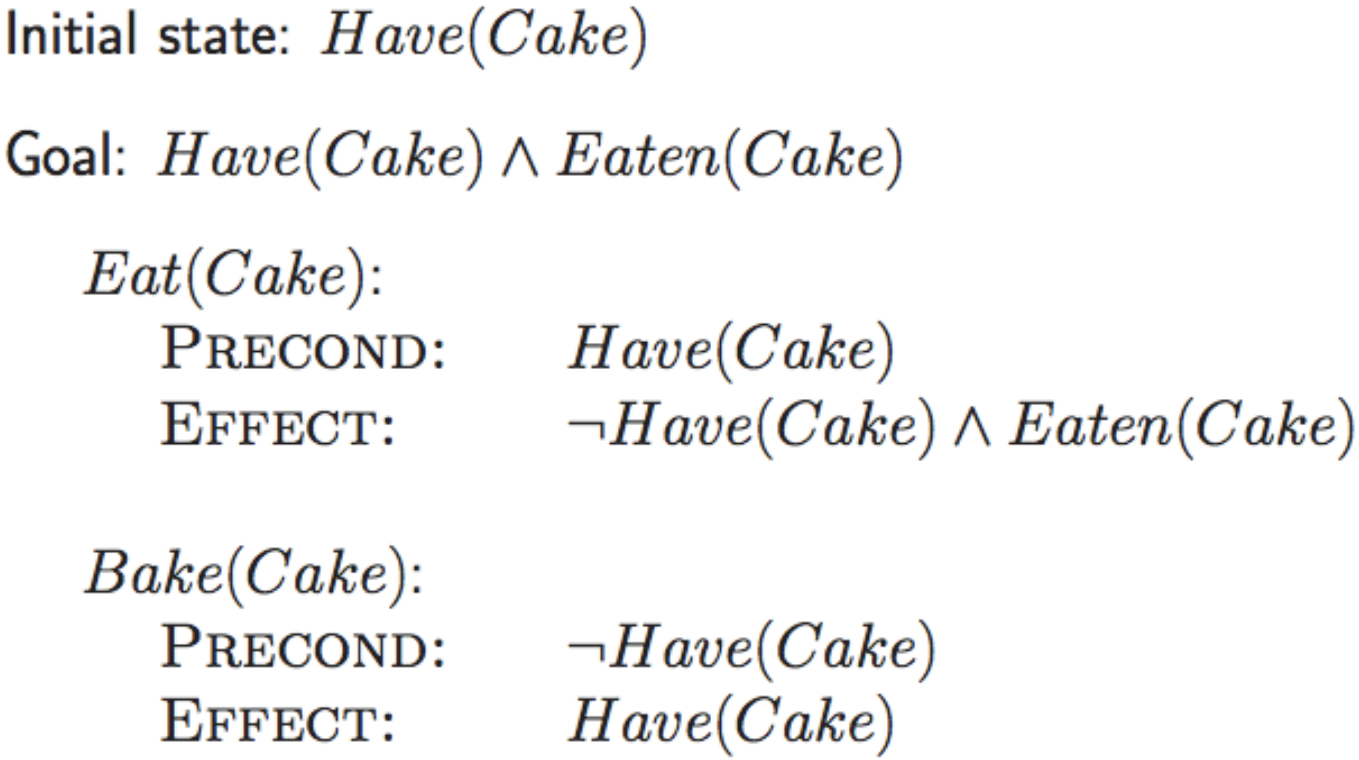
- S_0 contains all the conditions that hold in initial state
- Add operation to level O_i if its preconditions appear in level S_i
- Add condition to level S_i if it is the effect of an operation in level O_{i-1} (*no-op action* also possible)
- **Idea:** S_i contains all conditions that *could* hold at stage i; O_i contains all operations that *could* have their preconditions satisfied at time i
- Can optimistically estimate how many steps it takes to reach a goal: it includes all possible operations and preconditions that could hold, multiple actions could be executed (in parallel) at each stage (time step)

Mutual exclusion links

- The graph also *records conflicts* between actions or conditions: two operations or conditions are mutually exclusive (mutex) if no valid plan can contain both at the same time
- A bit more formally:
  - Two operations are mutex if their preconditions or postconditions are mutex
  - Two conditions are mutex if one is the negation of the other, or all actions that achieve them are mutex
- Even more formally…

A running example

- "*Have cake and eat cake too*" problem



Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

$Eat(Cake)$:
    Precond: $Have(Cake)$
    Effect: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
    Precond: $\neg Have(Cake)$
    Effect: $Have(Cake)$

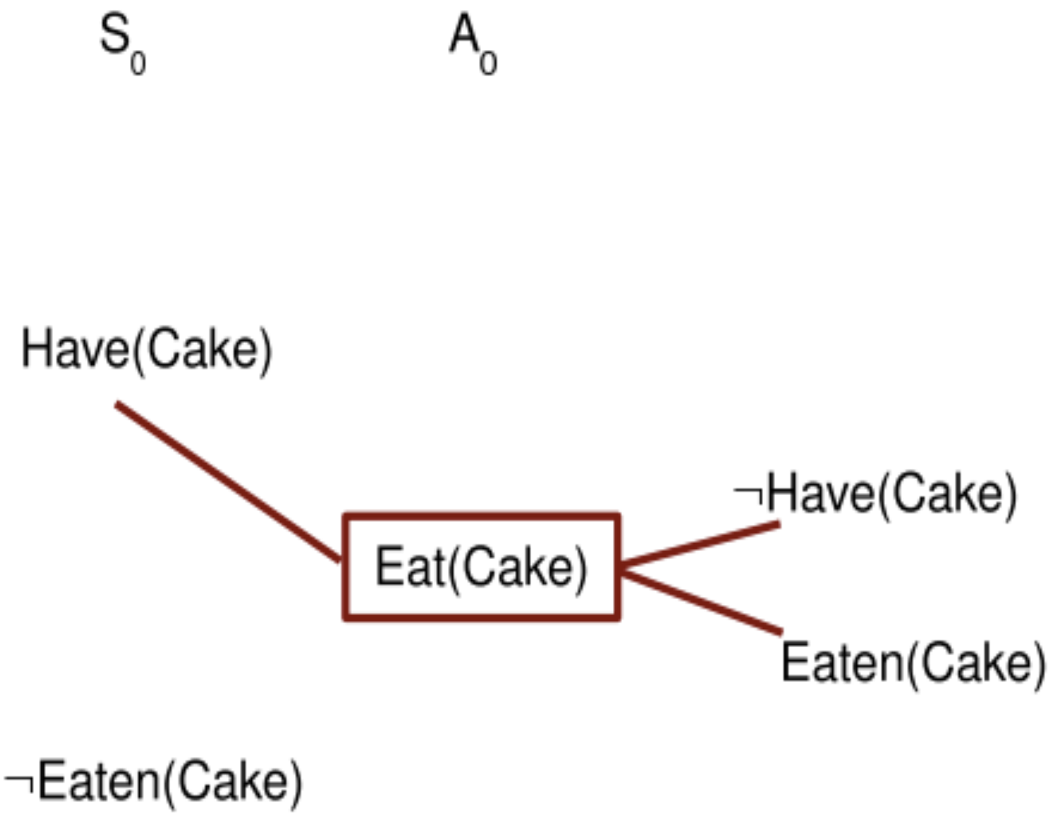
A running example

Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

$Eat(Cake)$:
- PRECOND: $Have(Cake)$
- EFFECT: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
- PRECOND: $\neg Have(Cake)$
- EFFECT: $Have(Cake)$



S_0

A_0

Have(Cake)

Eat(Cake)

¬Have(Cake)

Eaten(Cake)

¬Eaten(Cake)

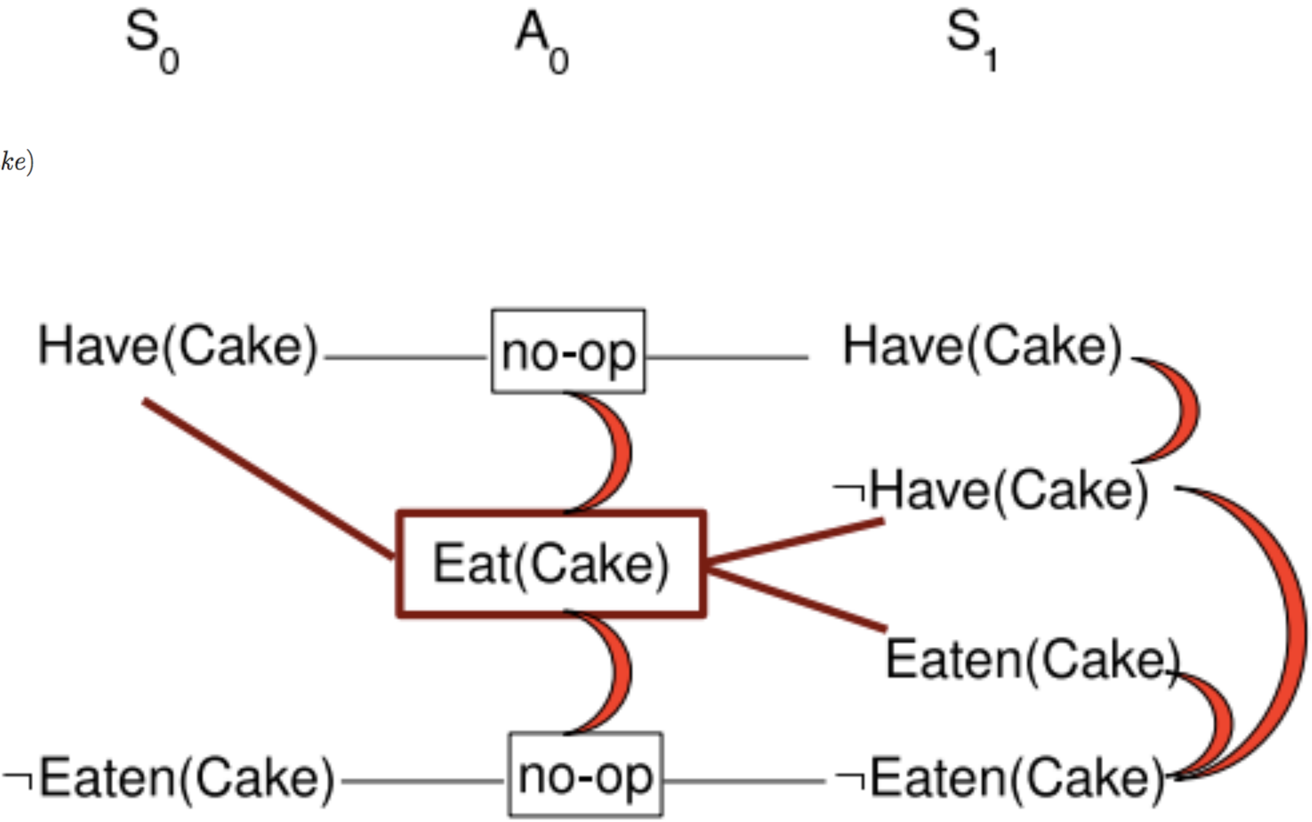
A RUNNING EXAMPLE

Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

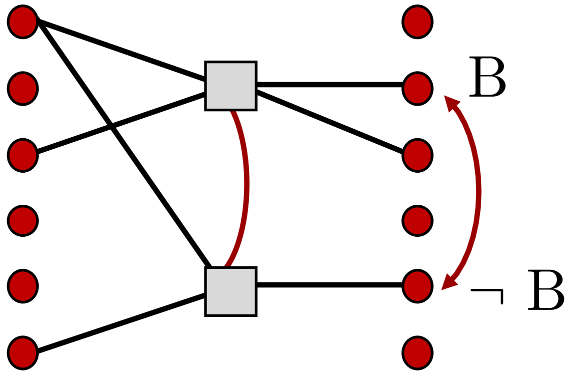
$Eat(Cake)$:
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$

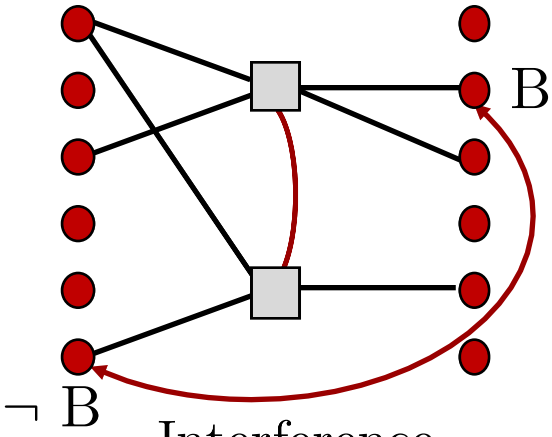


Mutex cases*

- Inconsistent postconditions (two ops): one operation negates the effect of the other, *Eat(Cake)* and no-op *Have(Cake)*
- Interference (two ops): a postcondition of one operation negates a precondition of other, *Eat(Cake)* and no-op *Have(Cake)* (issue in parallel execution, the order should not matter but here it would)



Inconsistent Postconditions

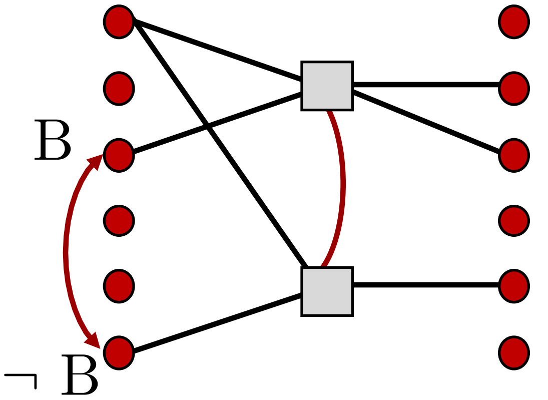


Interference

* Slide based on Brafman

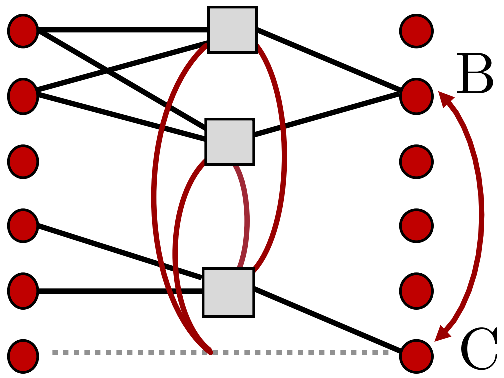
Mutex cases*

- Competing needs (two ops): a precondition of one operation is mutex with a precondition of the other, *Bake(Cake)* and *Eat(Cake)*



Competing Needs

- Inconsistent support (two conditions): each possible pair of operations that achieve the two conditions is mutex, *Have(Cake)* and *Eaten(Cake)*, are mutex in S_1 but not in S_2 because they can be achieved by *Bake(Cake)* and *Eaten(Cake)*



Inconsistent Support

* Slide based on Brafman

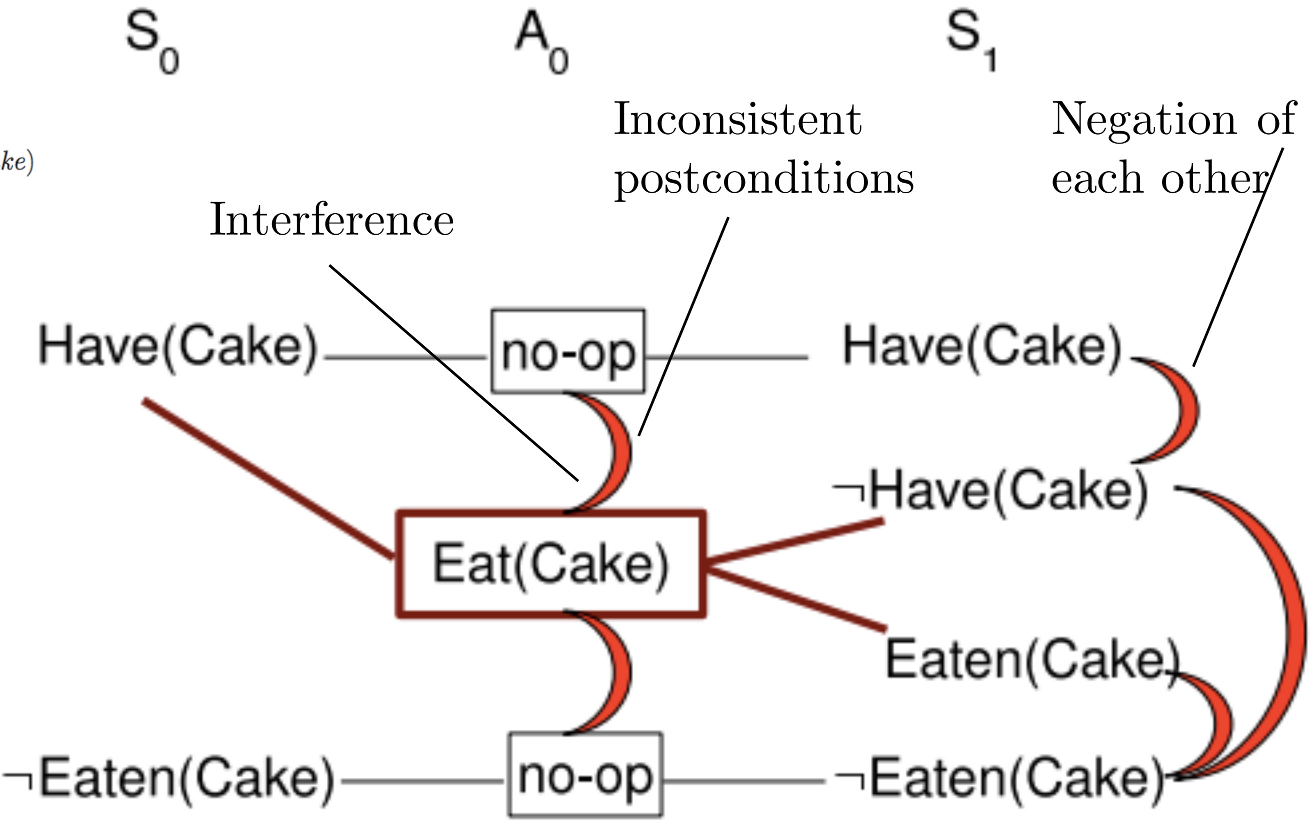
A RUNNING EXAMPLE

Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

$Eat(Cake)$:
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$



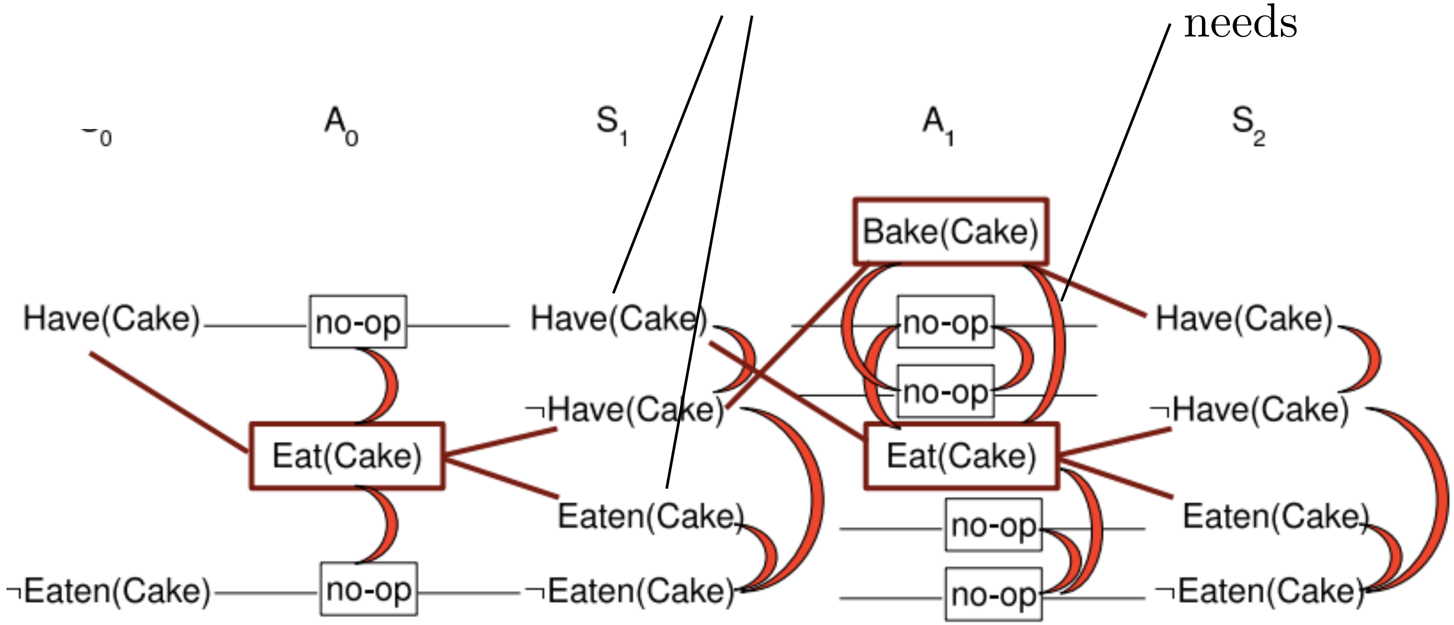
A running example

Initial state: $Have(Cake)$

Goal: $Have(Cake) \land Eaten(Cake)$

$Eat(Cake)$:
PRECOND: $Have(Cake)$
EFFECT: $\neg Have(Cake) \land Eaten(Cake)$

$Bake(Cake)$:
PRECOND: $\neg Have(Cake)$
EFFECT: $Have(Cake)$



Inconsistent support

Competing needs

Planning Graphs

To be continued ...