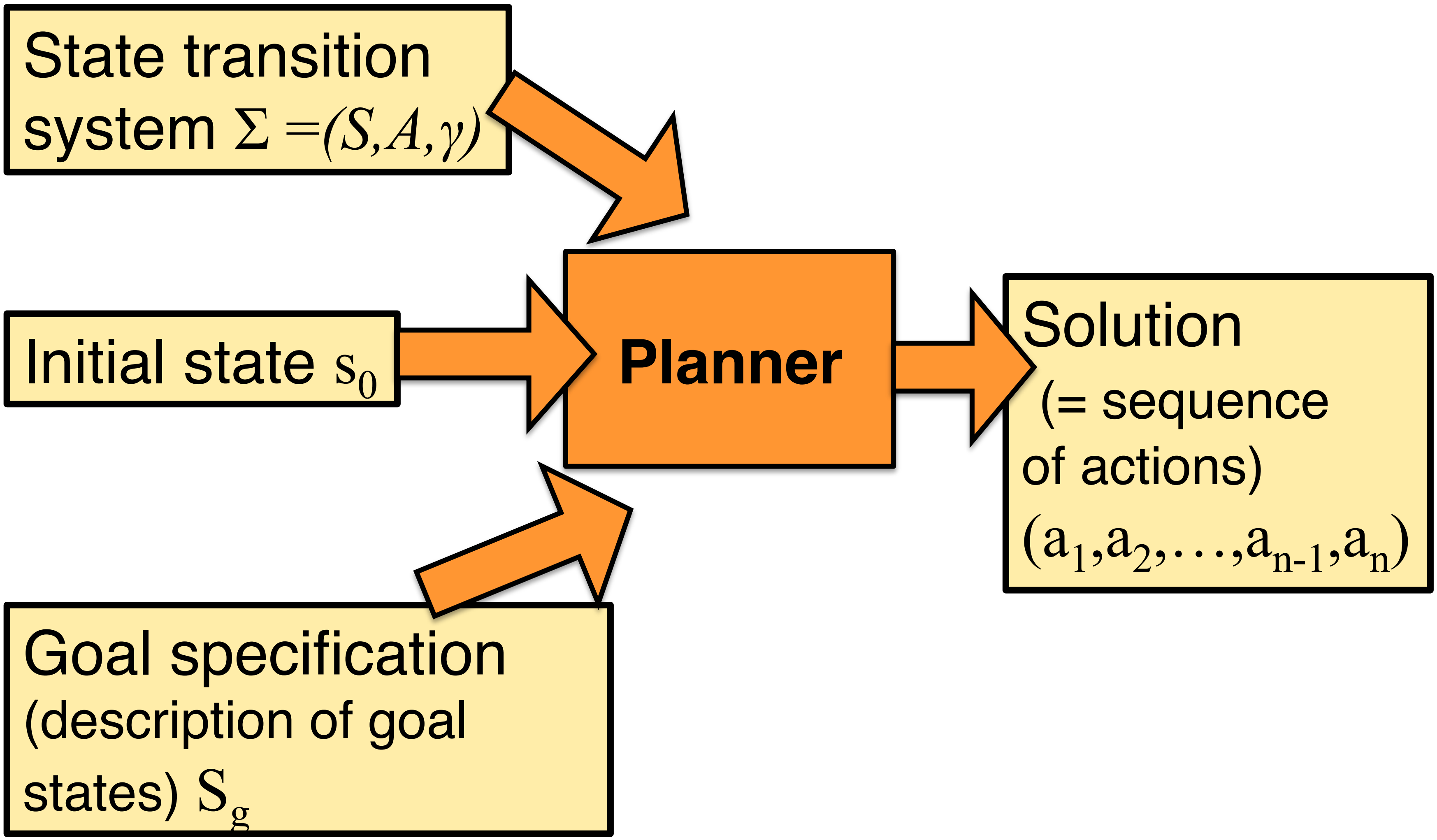
CS440/ECE448: Intro to Artificial Intelligence

# Lecture 12: Planning algorithms

Prof. Julia Hockenmaier
juliahmr@illinois.edu

http://cs.illinois.edu/fa11/cs440

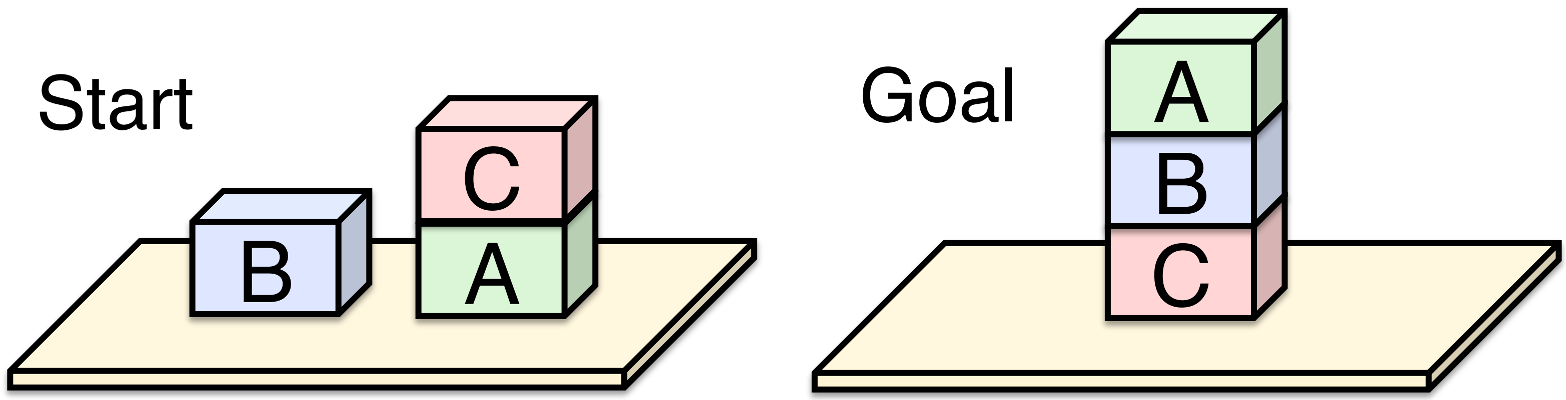
# Classical Planning



# Review: representations for planning

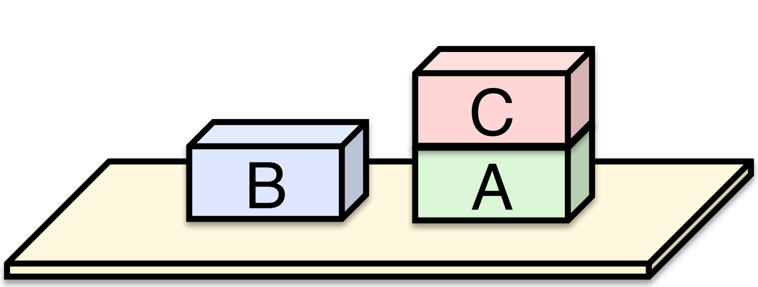
| Situation Calculus | Strips |
| --- | --- |
| Specify fluents | Specify fluents |
| *Add*-set | *Add*-set |
| *Persist*-set | *Delete*-set |
| By default fluents are deleted | By default fluents persist |

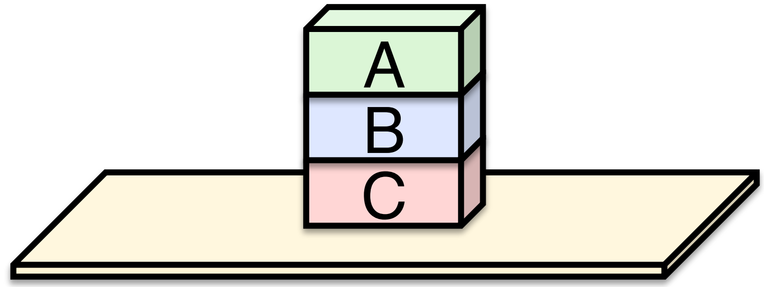
# Sussman anomaly




Start: $On(C,A)$

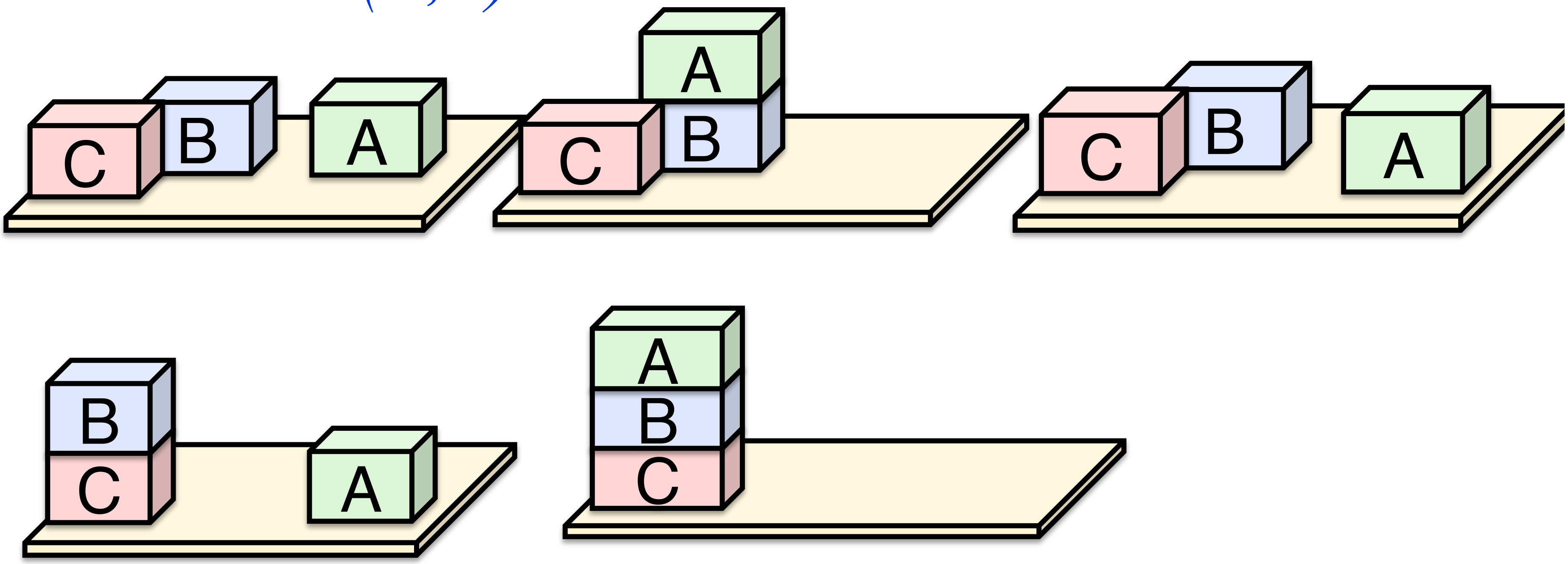
Goal: $On(A,B) \wedge On(B,C)$

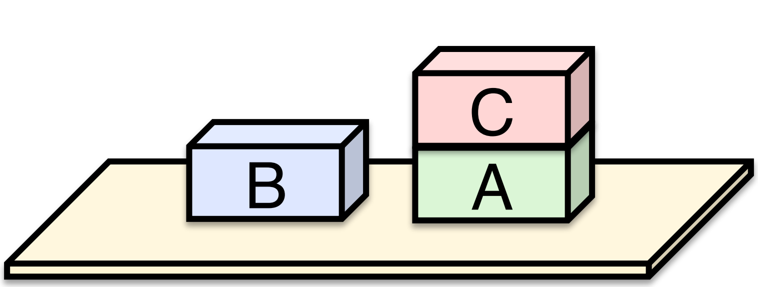


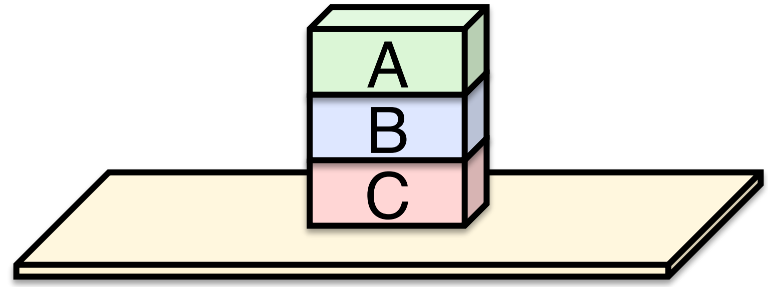




Start: $On(C,A)$ Goal: $On(A,B) \land On(B,C)$

**Solve $On(A,B)$ first:**



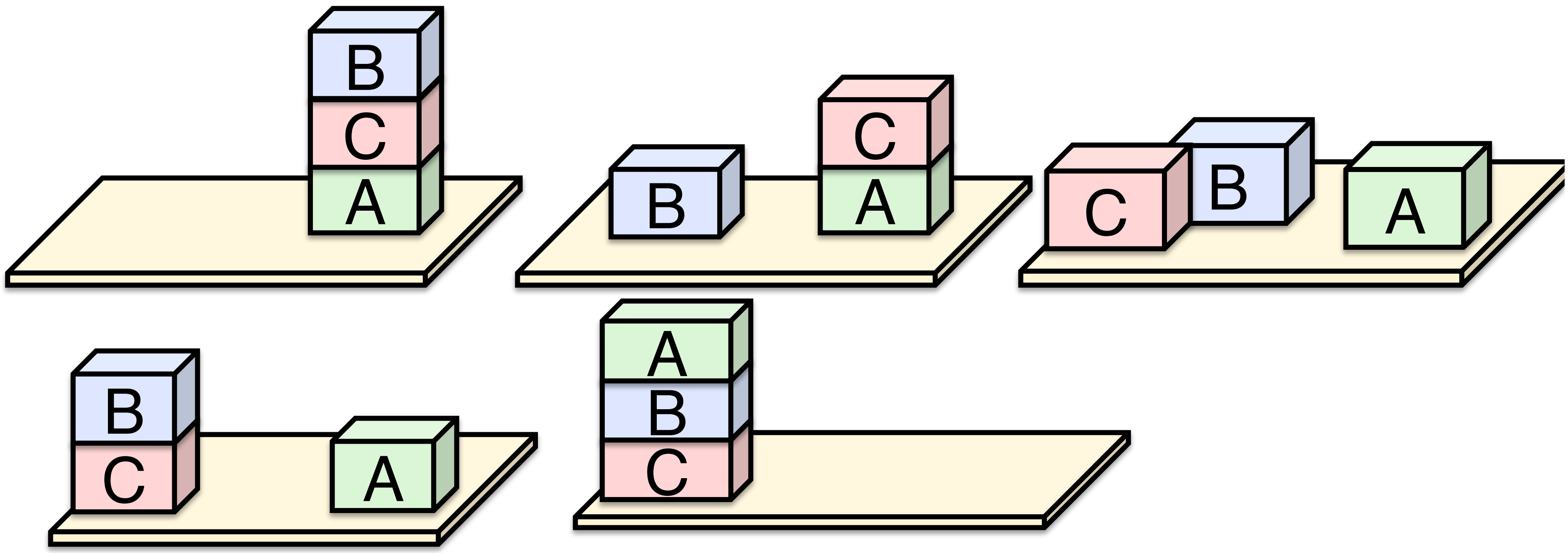


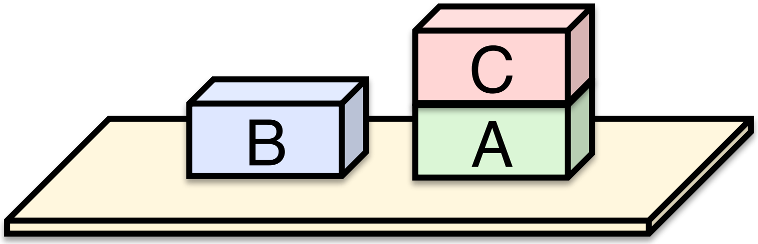



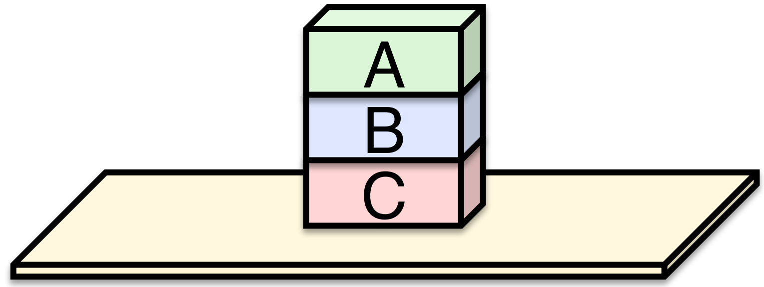

Start: $On(C,A)$

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

## **Solve $On(B,C)$ first:**



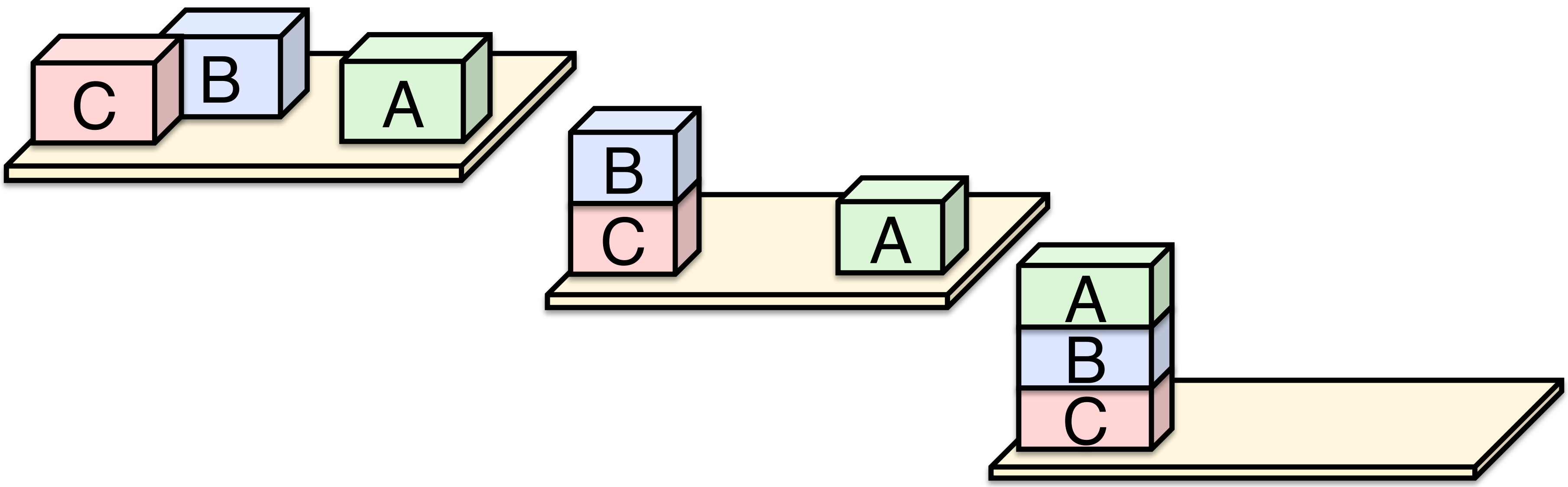



Start: $On(C,A)$

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

Most efficient solution requires **interleaved planning:**



# Planning algorithms

**State space search** (DFS, BFS, etc.)

Nodes = states; edges = actions;

**Heuristics** (make search more efficient)

Compute h() using relaxed version of the problem

**Plan space search** (refinement of partial plans)

Nodes = partial plans; edges: fix flaws in plan

**SATplan** (encode plan in propositional logic)

Solution = *true* variables in a model for the plan

**Graphplan** (reduce search space to planning graph)

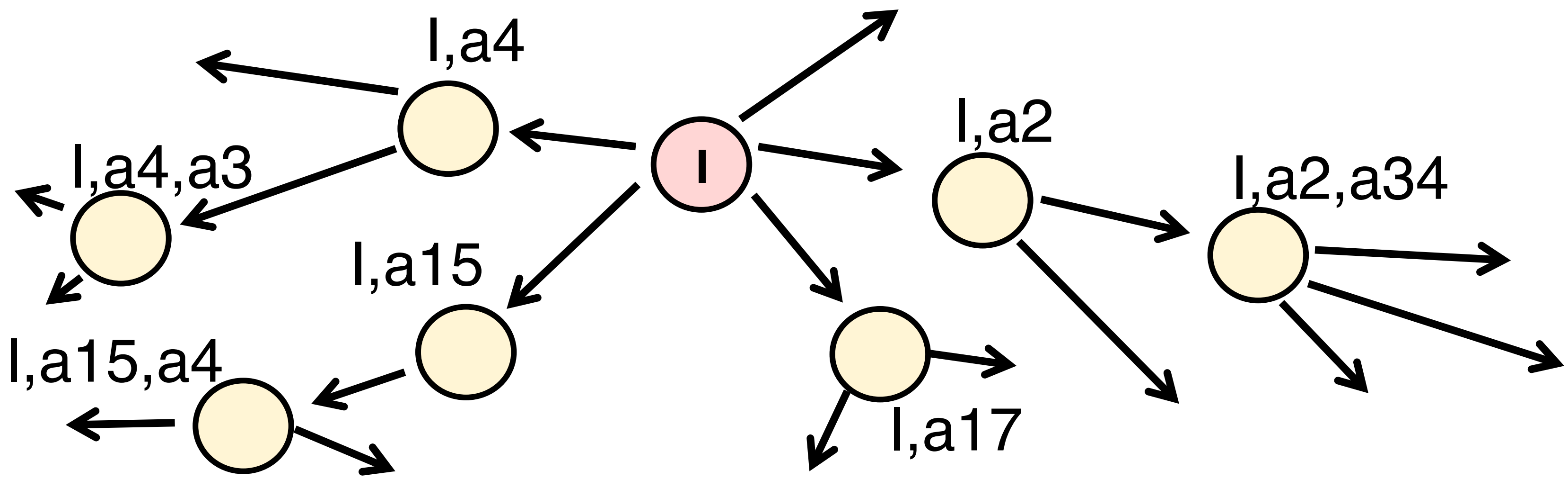
Planning graph: levels = literals and actions

# State space search

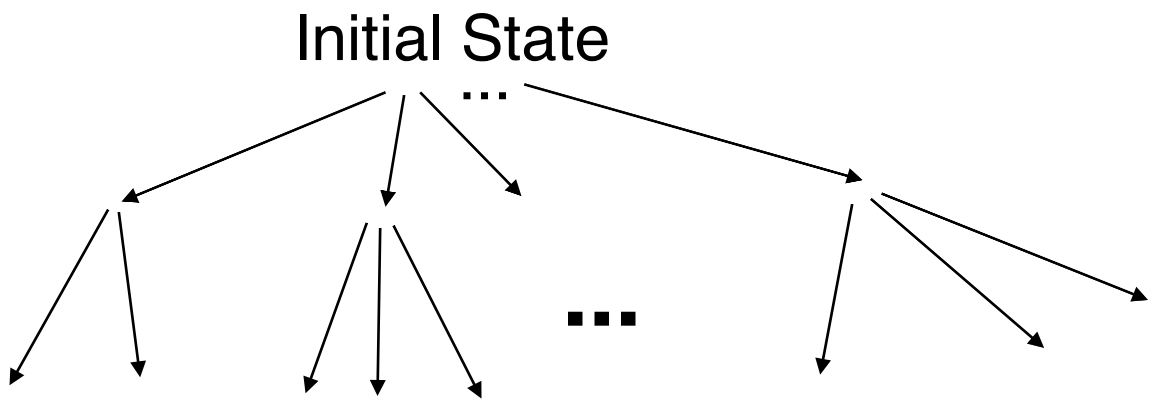
# Planning as state space search

Search tree:

– Nodes: states
– Root:  initial state
– Edges: actions (ground instances of operators
– Solutions: paths from initial state to goal.



# Forward search




Breadth-first forward search is sound and complete, but may require lots of memory

Depth-first forward search can be better in practice (needs graph-search to be complete)

Problem: branching factor is very large (need good heuristic: which actions may lead to goal?)

# DFS and loops: iterative deepening

Loops ($s_i \rightarrow \ldots \rightarrow s_i$) in the search graph lead to infinite branches in the search tree.

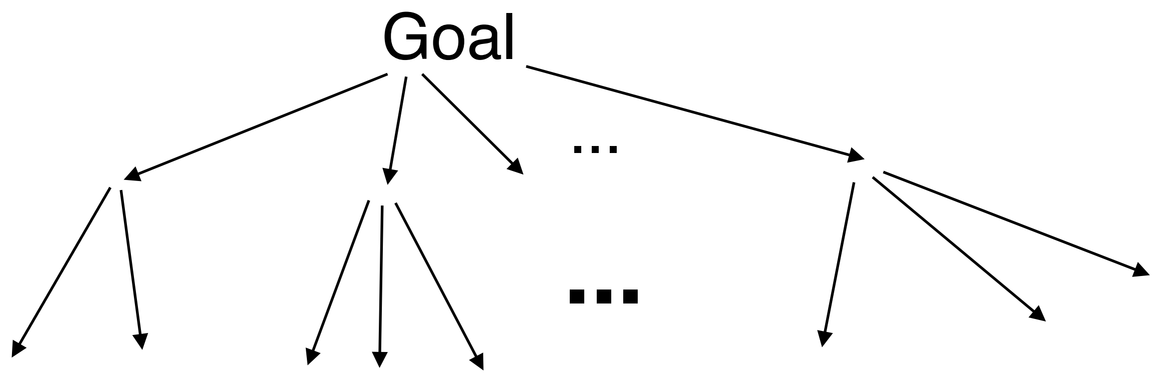
The tree-search variant of DFS never terminates if it goes down an infinite branch

Remedy (iterative deepening):

– Try to find solution of length $l$ with DFS

– If this fails, $l := l + \Delta$; try again.

# Backward search




Start with goal; 'undo' actions until initial state

Edges = inverse actions $a^{-1}$:

$$Result(a^{-1},s) = s \setminus effects(a) \cup precond(a)$$

Large branching factor: many possible actions; not every $Result(a^{-1}, s)$ leads back to initial state

# Plan-space search

# State space search is inefficient

Observation 1: many actions are independent.

e.g.: *move(A,B,C)* and *move(D,E,F)*

We don't want to have to commit to specific order.

Observation 2: naïve backward search requires fully instantiated actions. We often don't know/care how to instantiate variables.

*move(A,B,?C)* is prerequisite for *move(B,D,E)*, but we don't care which C A moves to.

# Searching plan-space

States in plan-space are partial plans: = sets of partially instantiated actions with constraints on precedence and variable (in)equality

$move(A,B,x) < move(B,y,z); \;\; z \neq A$

Solution is (partially ordered) complete plan with fully instantiated actions

# Searching plan-space

1. Start with the empty plan
   = *{start state, goal state}*

2. Iteratively refine current plan to resolve flaws
   (refine = add new actions and constraints)

3. Solution = a plan without flaws

# Flaws: open goals

Open goal: a precondition that is not yet met

$move(A,B,x)$ requires $clear(x)$

Solution*:*

add new action and precedence constraint:

$move(A,B,x)$ needs to be preceded by
some new action *A* with effect $clear(x)$.

# Flaws: threats

*C* is a threat if *A* is a precondition for *B*
and *C* undoes the effect of *A.*

*move(A,B,x)* establishes precondition *clear(B)* for *move (C,D,B)*. *move(E,F,B)* undoes *clear(B)*

Solution: add new precedence constraint: *move(E,F,B)* has to precede *move(A,B,x)*, or follow *move(C,D,B)*.

# SATplan

# SATplan: basic idea

We can encode a plan of fixed length *n*
(*n* time steps required for solution)
as a formula in propositional logic.

There is a solution if this formula is satisfiable. Use existing tools (SAT solvers)

If there is no solution of length *n*, try *n+1.*

# From plans to propositional logic

– **Fluents** are ground literals:
$clear(B)^t$: block *B* is clear at time *t*

– **Actions** are ground implications:
$(preconditions^t \wedge action^t) \rightarrow effect^{t+1}$

Operator $move(x,y,z)$:
*PRE: on(x,y), clr(z) EFFECT: on(x,z),clr(y)*
Action $move(A,B,C)^{23}$
$(on(A,B)^{23} \wedge clr(C)^{23} \wedge move(A,B,C)^{23}) \rightarrow on(A,C)^{24} \wedge clr(B)^{24}$

# Graphplan

# Basic idea

1. Create an easier, relaxed problem P':
   - can be solved in polynomial time
   - relax = remove some restrictions
   - Solutions to P' $\supset$ solutions to P
2. Solve this relaxed problem P'
3. Search among solutions to P' for solutions to P

Iterative deepening: try to solve P' in 1,…,n steps

# Basic idea

**Relaxed problem P'**: can we satisfy some necessary precondition for the goal of P in *k* steps? (N.B.: this only solves P if *all* necessary preconditions can be achieved in *k* steps)

**Solve P':** build a planning graph of depth *(2)k*

**Solve P:** do backward search on the planning graph to extract solution to P. If this fails, set k to k+1.

# Planning graph: nodes

Two kinds of levels alternate:

**State level $S_i$:**

Node at level $S_i$: a ground fluent (pos. or neg. literal) which may hold *i* actions after $S_0$

**Action level $A_i$:**

Node at level $A_i$: a ground action (incl. *noop*) whose preconditions might be satisfied at $S_i$

# Have your cake and eat it too!

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

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

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

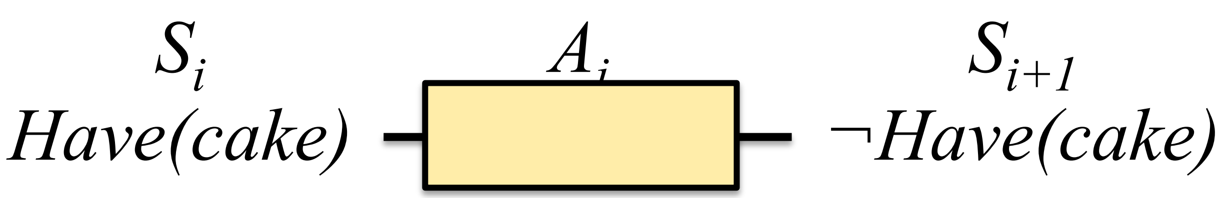
# Preconditions/effects: edges *between* levels

**From state level $S_i$ to action level $A_i$:**

the node (fluent) at level $S_i$ is a precondition for the node (action) at level $A_i$

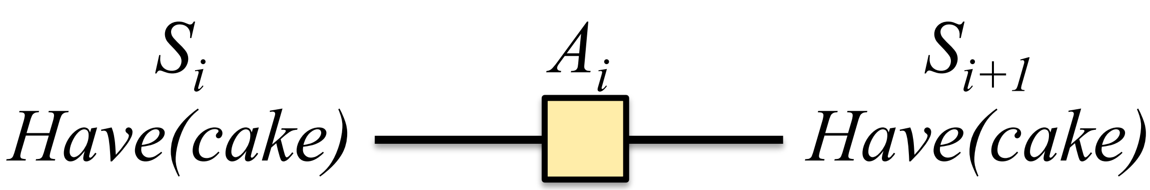
**From action level $A_i$ to state level $S_{i+1}$**

the node (fluent) at level $S_{i+1}$ is an effect of the node (action) at level $A_i$



# Persistence (no-op) actions

For each literal $C_i$ at state level $S_i$:

- add a **persistence action $\mathbf{Noop_C}$** at action level $A_i$
- Link from **$\mathbf{C_i}$ to $\mathbf{Noop_C}$**
- Also add **$\mathbf{C_{(i+1)}}$** to state level $S_{i+1}$
- Link from **$\mathbf{Noop_C}$ to $\mathbf{C_{i+1}}$**
- 

# Mutually exclusive actions: *mutex* links *within* action levels

–Inconsistent effects:
effect of $A_1$ negates effect of $A_2$
*eat(cake)* negates *bake(cake)*

–Interference:
effect of $A_1$ negates precondition of $A_2$
*eat(cake)* interferes with no-op for *have(cake)*

–Competing needs:
preconditions of $A_1$, $A_2$ are *mutex*
*eat(cake)* competes with *bake(cake)*

# Mutually exclusive fluents: edges *within* state levels

*Mutex* within **state level $S_i$:** *F1* and F2 cannot hold at the same time.

- Negation:
  *have(Cake)* negates ¬*have(Cake)*
- Inconsistent support:
  all actions a1 to achieve F1 are mutex with any action a2 that achieves F2.
  *have(Cake)* and *eaten(Cake)*

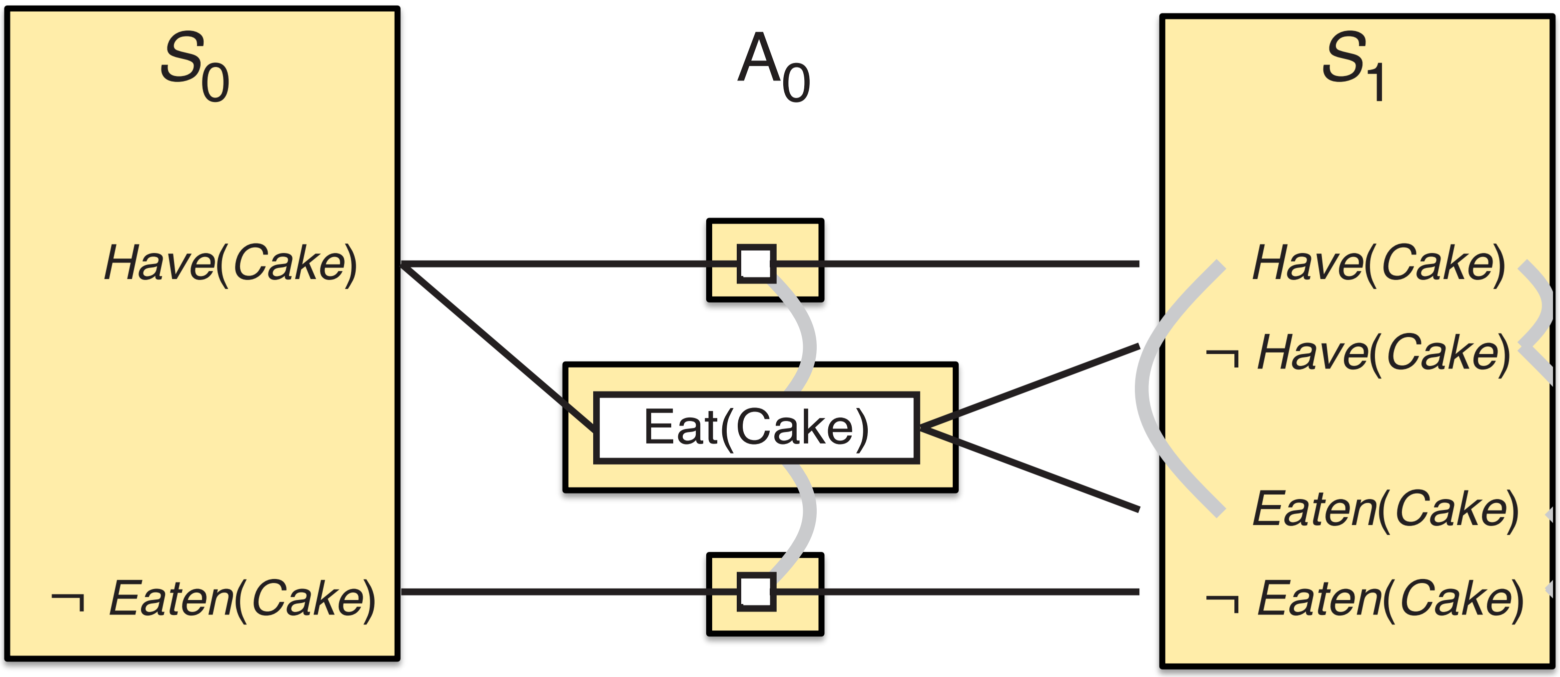
# The initial planning graph

**Action** ***Eat(Cake)***

PC: *Have(Cake)* EF: $\neg$*Have(Cake)* $\wedge$ *Eaten(Cake)*

**Action** ***Bake(Cake)***

PC.:$\neg$*Have(Cake)* EF: *Have(Cake)*



# The size of the planning graph

The original planning problem P
has $l$ literals and $a$ actions

Size of planning graph with *n levels*:
polynomial in $l$ and $a$

– State levels: at most $l$ nodes and $l^2$ *mutex* links
– Action levels: at most $a+l$ nodes, $(a+l)^2$ *mutex;*

# Solution extraction

If all literals in *goal(P)* hold at $S_n$ and are not *mutex* with each other, we may be able to find a solution by backward search:

- States in backward search tree: conjunction of (a subset of the) literals at $S_i$
- Actions in backward search tree: conflict-free subset of actions at $A_{i-1}$
- Goals of $S_n$: all literals in *goal(P)*
- At each state level $S_i$, select

# The search tree for solution extraction

- **State**: conjunction of (subset of the) literals at $S_i$
- **Initial state:** all literals in *goal(P)*
- **Goal state:** all literals in *init(P)*

- **Actions**: a conflict-free subset **a** of actions at $A_{i-1}$
- **Result of a set of actions from** $A_{i-1}$:
  the conjunction of all literals in $S_{i-1}$
  that are preconditions for some action in **a**

# Heuristics for planning

# Using the planning graph to estimate heuristics

What is the cost of a goal literal *g*?

**Minimum cost (level cost)**

= minimum number of steps required to achieve *g*

= first state level at which *g*

What is the cost of a conjunction of goal literals?

**Max level cost:** max. of level costs of each literal

**Level sum:** sum of level costs (inadmissible)

**Set level cost:** level at which all are true and each pair of literals is non *mutex*

# Today's key concepts

**State space search** (DFS, BFS, etc.)
Nodes = states; edges = actions;
**Heuristics** (make search more efficient)
Compute h() using relaxed version of the problem
**Plan space search** (refinement of partial plans)
Nodes = partial plans; edges: fix flaws in plan
**SATplan** (encode plan in propositional logic)
Solution = *true* variables in a model for the plan
**Graphplan** (reduce search space to planning graph)
Planning graph: levels = literals and actions