CS 5100: Foundations of Artificial Intelligence

AI Planning

Prof. Amy Sliva
October 13, 2011

Outline

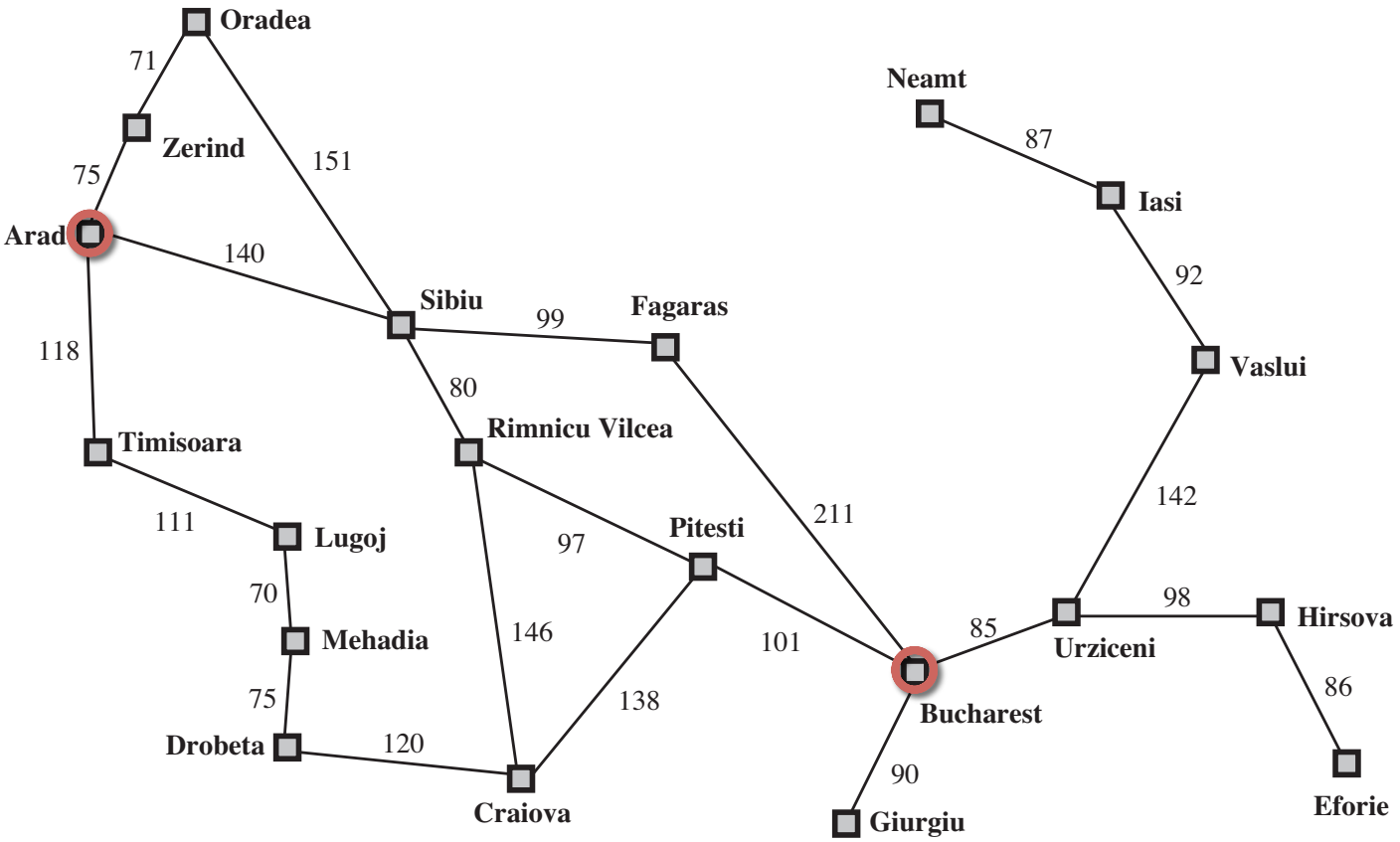
- Review A* Search
- AI Planning
  - State space search
  - Planning graphs
  - Situation calculus

Best-first search

- Use an **evaluation function** $f(n)$ for each node
  - Estimate of "desirability"
  - Includes a **heuristic function** to add more knowledge to the problem
    $h(n)$ = estimated cost of cheapest path from state at n to the goal
    - Heuristics are problem-specific
- Expand **most desirable** unexpanded node
- Implementation
  - Order nodes in the frontier in decreasing order of desirability

Heuristic for path-finding

- Romania example: estimate the straight-line distance from each node to Bucharest (the goal)



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

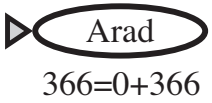
A* Search

- Avoid expanding paths that are already expensive

- **Evaluation function** $f(n) = g(n) + h(n)$
  - $g(n)$ = cost of path so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

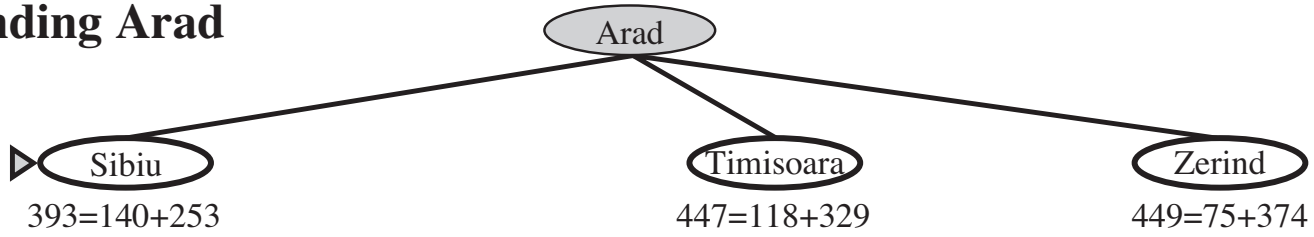
A* Search

(a) The initial state

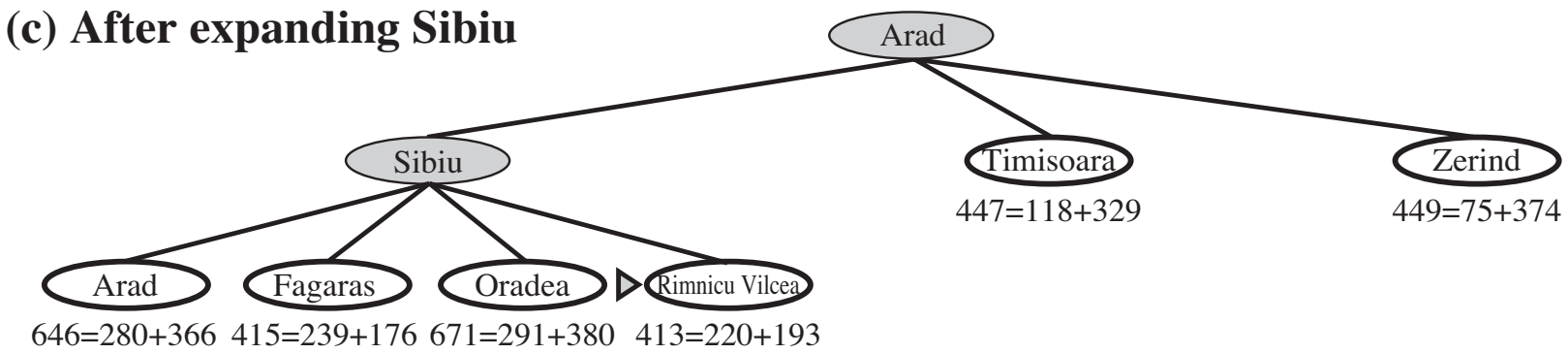


A* Search

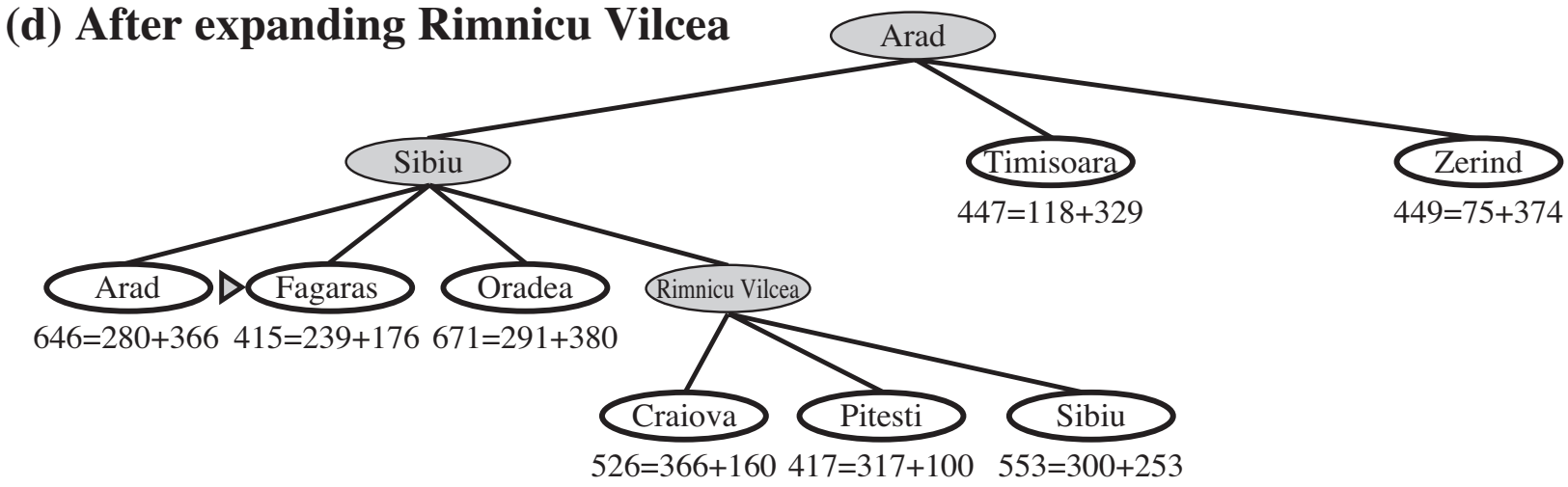
(b) After expanding Arad



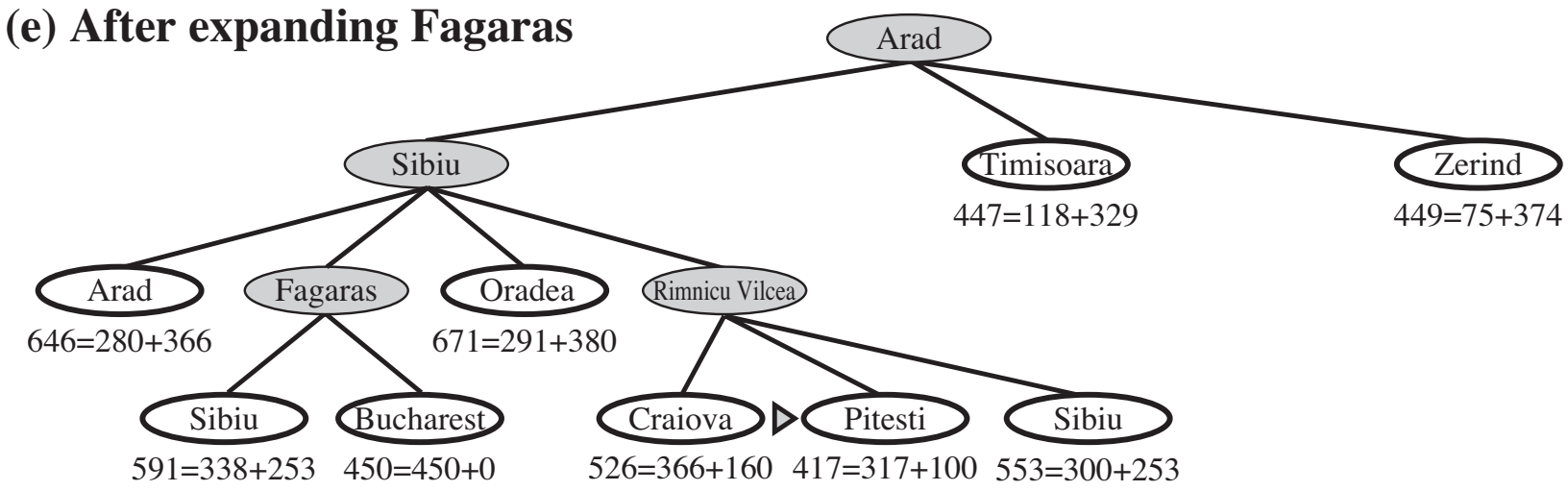
A* Search


(c) After expanding Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193

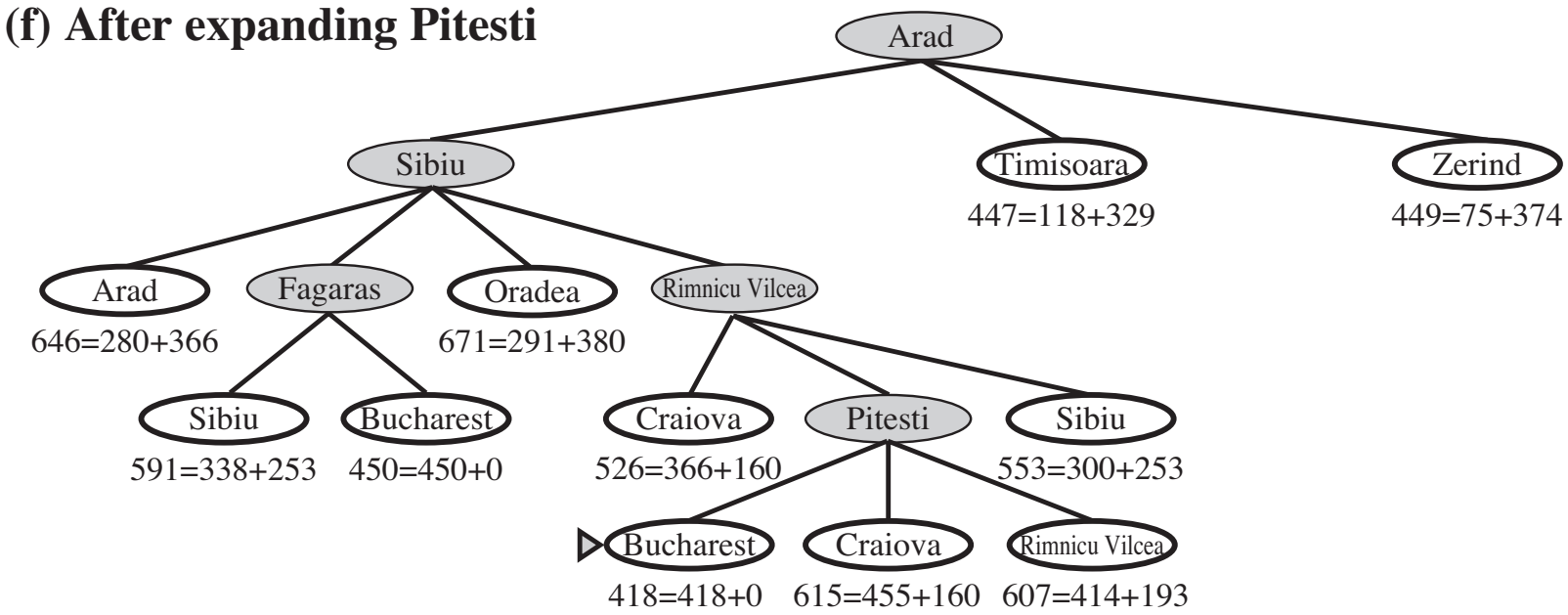
A* Search


(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Search


(e) After expanding Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* Search


(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

AI Planning

- Description of a **planning problem**
  - Way to describe the world
  - Initial state
  - Goal description
  - Set of possible **actions** to change the world
- Result: generate a **plan**
  - Sequence of actions for **changing** the initial state into a goal state
- Note similarity to state space search
- Planning extends to more complex worlds

Applications of planning

- Mobile robots
  - Initial motivator of the field, and still being developed
- Simulated environments
  - Goal directed agents for training or games
- Web and grid environments
  - Intelligent web “bots”
  - Workflows on a computational grid
- Managing crisis situations
  - E.g., Forest fires, oil spills, urban evacuations, in factories, …
- And many more!
  - Automation in factories, UAVs, playing bridge, military logistics, …

Planning is representing **change**

- As an agent considers actions it needs to
  - Know how an action will change the world
  - Keep track of the history of world states (avoid loops)
- 3 planning approaches
  - STRIPS with **state space search**
  - STRIPS with **planning graph**
  - **Situation calculus**

Assumptions in "classical planning"

- Discrete time
  - Instantaneous actions
- Deterministic effects
- Omniscience
- Sole agent of change
- Goals of attainment (not avoidance)
  - Negated goals are possible with slight expansion to "classic" planning

- Uses a **factored representation**
  - State of the world is collection of variables

Language for planning representation

- STRIPS—highly influential representation for actions
  - Planning Domain Definition Language (PDDL) is STRIPS variant
- Instead of transition function *T*: *state* × *action* → *new state* uses **planning operators** to achieve goals and subgoals
  - **Preconditions**—list of propositions to be true
  - Delete list—list of propositions that will *become* false
  - Add list—list of propositions that will *become* true

  } **Effects**
- Subset of first-order logic

- More efficient to capture known strategies than searching state space

Example planning problem

- **Initial state**
  $at(home), \neg have(beer), \neg have(chips)$
- **Goal**
  $at(home), have(beer), have(chips)$

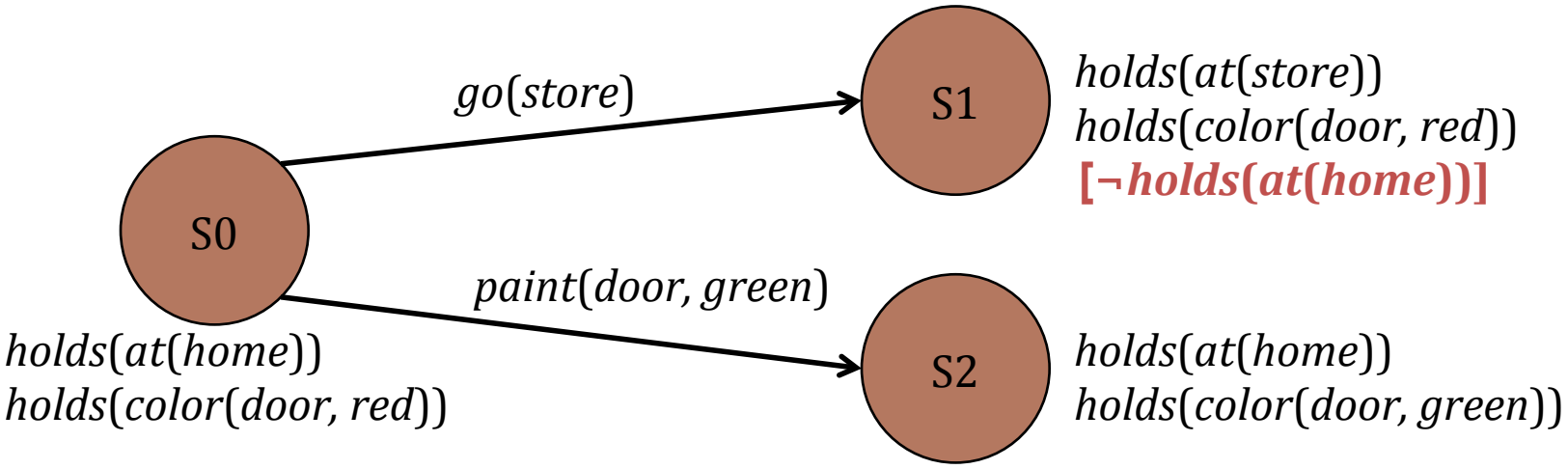
- **Operators**

$Buy(X)$:
Pre: $at(store)$
Add: $have(X)$

$Go(X,Y)$:
Pre: $at(X)$
Del: $at(X)$
Add: $at(Y)$

States of the world

- States are assertions of agent's beliefs
  - Given by **fluents**—ground (no variables), functionless logical atoms representing aspects of the world that change
- Contain partial descriptions
  - **Set** (or conjunction) of fluents that are true
- State space graph of world states and actions



Frame problem (again!)

- I go from home to the store—new situation *S'*
- In *S'*
  - The store still sells chips
  - My age is still the same
  - Los Angeles is still the largest city in California…

- How can we efficiently represent everything that doesn't change?
  - **STRIPS/PDDL provides a good solution for simple actions**

Ramification problem

- I go from home to the store—new situation *S'*
- In *S'*
  - I am now in Marina del Rey
  - The number of people in the store went up by 1
  - The contents of my pockets are now in the store…

- Do we want to say all that in the action definition?

Solutions to the frame and ramification problems

- Some facts are **inferred** with a world state
  - e.g., the number of people in the store
- All other facts—e.g., at(home)—**persist** between states unless changed
  - Remain unless explicitly on delete list

- A challenge for knowledge engineer not to make mistakes!

Blocks world example

Start State Goal State

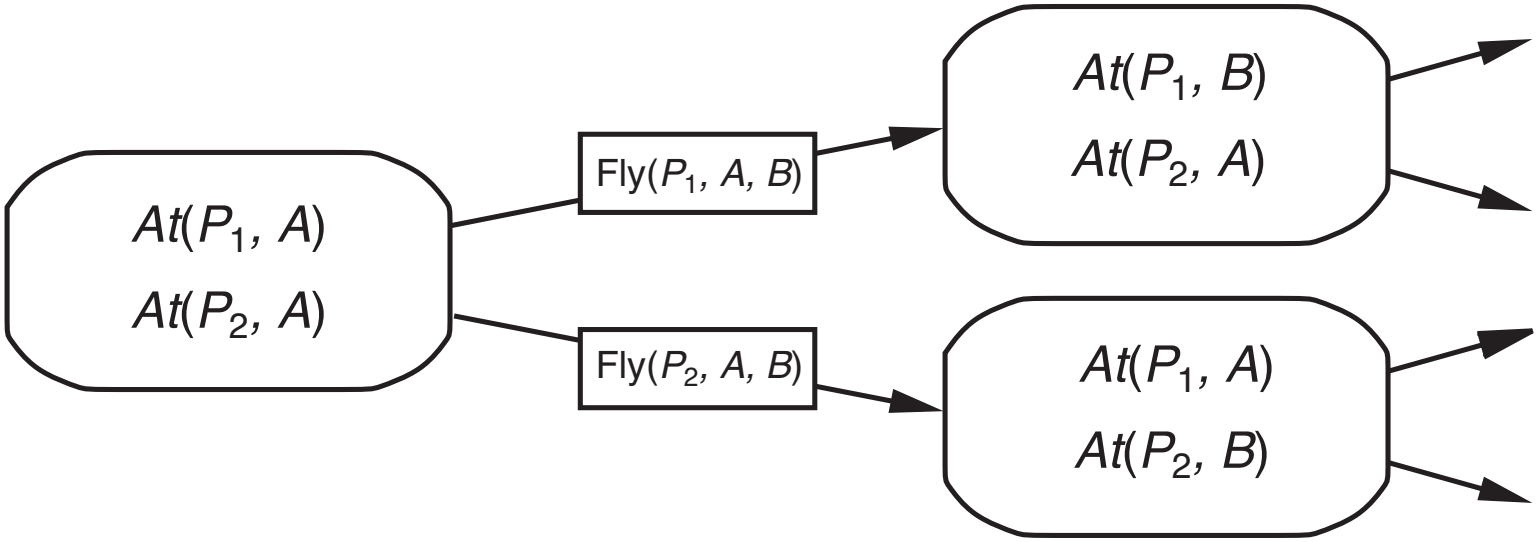
- **Initial state**
  $On(A, table)$, $On(C,A)$, $On(B, table)$, $clear(B)$, $clear(C)$
- **Goal**
  $On(A, B)$, $On(B,C)$
- "Naïve" planning algorithm output—**Sussman anomaly**
  [$Put(C, table)$, $Put(A, B)$ **goal 1 accomplished**, $Put(A, table)$, $Put(B, C)$ **both goals accomplished**] DONE!!

Planning as a search problem

- Nodes—world states
- Arcs—actions
- Solution—path from initial state to one that satisfies goal
  - Initial state is fully specified
  - There may be many goal states
- **Total-order** planning
- Search algorithms
  - Progression—forward search
  - Regression—backward search

Progression

- Search from **initial state** to the **goal**



Progression algorithm

```
ProgWS(state, goals, actions, path)
  if state satisfies goals then return path
  else a = choose(actions) s.t.
         preconditions(a) satisfied in state

  if no such a then return failure

  else return
         ProgWS(apply(a, state), goals,
                  actions,concatenate(path, a))
```

- **First call:** ProgWS(initState, G, Actions, ())

Progression example

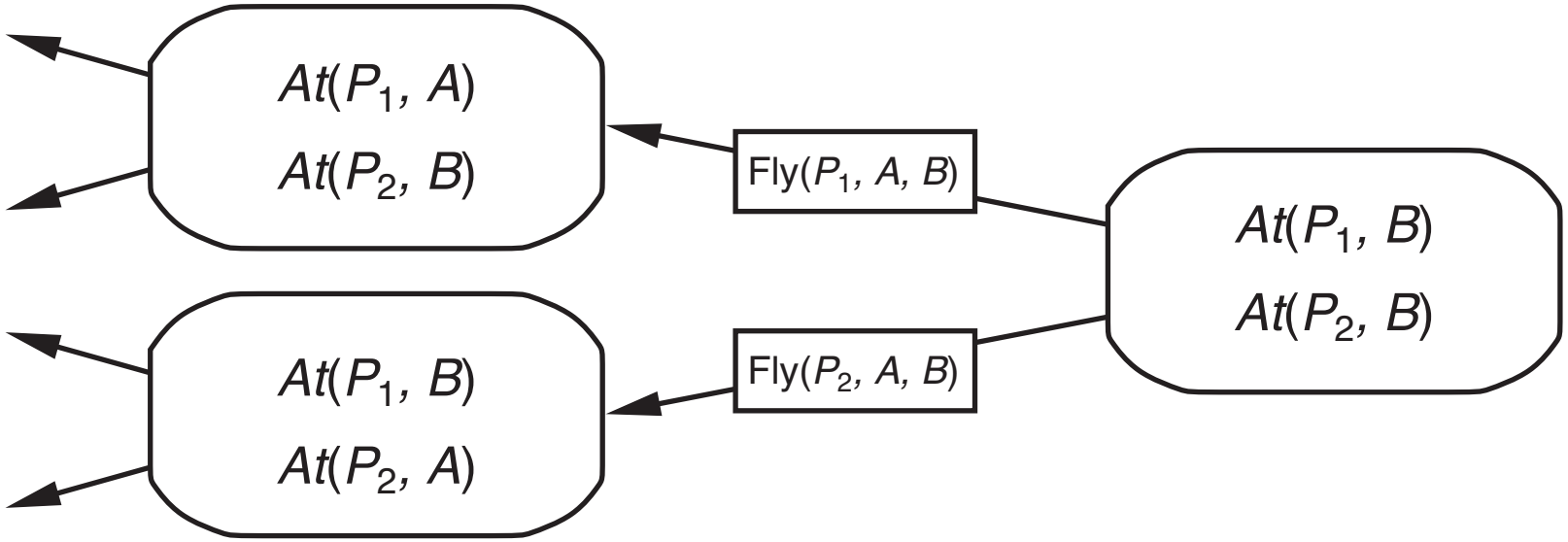
- **Initial state**
  $On(A, table)$, $On(C,A)$, $On(B, table)$, $clear(B)$, $clear(C)$
- **Goal**
  $On(A, B)$, $On(B,C)$
- Progression execution
  1. P(I, G, BlocksWorldActions, ())
  2. P(S1, G, BWA, (move-C-from-A-to-table))
  3. P(S2, G, BWA, (move-C-from-A-to-table, move-B-from-table-to-C))
  4. P(S3, G, BWA, (move-C-from-A-to-table, move-B-from-table-to-C, move-A-from-table-to-B))

  G ⊆ S3 → Success!

Non-Deterministic Choice!

Regression

- Start at the goal and apply actions **backward** until reach initial state
  - Only considers actions that are **relevant** to the goal



Regression algorithm

```
RegWS(state, current-goals, actions, path)
  if state satisfies current-goals then return path
  else a = choose(actions) s.t. some effect of a
    satisfies one of current-goals

  if no such a then return failure #unachievable
  if some effect of a contradicts some of current-
    goals then return failure #inconsistent state

  CG' = current-goals – effects(a) + preconditions(a)

  if current-goals ⊂ CG' then return failure #useless

  return RegWS(state, CG', actions,
    concatenate(a,path))
```

- **First call:** RegWS(initState, G, Actions, ())

Regression example

- **Initial state**
  $On(A, table)$, $On(C,A)$, $On(B, table)$, $clear(B)$, $clear(C)$
- **Goal**
  $On(A, B)$, $On(B,C)$
- Regression execution
  1. R(I, G, BlocksWorldActions, ())
  2. R(I, ((clear A) (on-table A) (clear B) (on B C)), BWA, (move-A-from-table-to-B))
  3. R(I, ((clear A) (on-table A) (clear B) (clear C), (on-table B)), BWA, (move-B-from-table-to-C, move-A-from-table-to-B))
  4. R(I, ((on-table A) (clear B) (clear C) (on-table B) (on C A)), BWA, (move-C-from-A-to-table, move-B-from-table-to-C, move-A-from-table-to-B))

Non-Deterministic Choice!

current-goals ⊆ I → Success!

Progression vs. regression

- Both algorithms are:
  - **Sound**—the result plan is valid
  - **Complete**—if a valid plan exists, they find one
- Non-deterministic choice→ search problem!
  - Uninformed: BFS, DFS, iterative deepening, etc…
  - Heuristic: A*
- Regression is often faster because **focused** by goals
- Progression often has more **accurate heuristics** because uses full state

Planning graphs

- Directed graph organized as **levels**
  - Nodes—fluents and actions
    - Alternating levels of states and actions
    - All literals/actions that could occur at each level
      A_i contains actions **applicable** in S_i, S_{i+1} contains literals resulting from A_i
  - Edges—preconditions, effects, and conflicts
- Provide more accurate heuristics
  - First level where literal appears estimates difficulty of getting there from initial state
- Search graph for planning solution—GRAPHPLAN algorithm

Persistence and mutex

- **Persistence** actions (no-ops)
  - Literal persists if not negated—no-op action gets it to next graph level
- **Mutual exclusion** (mutex) links—edges recording conflicts between actions
  - Inconsistent effects: one action negates an effect of the other
  - Interference: one of the effects of an action is a negation of the precondition of another
  - Competing needs: precondition of one action is mutually exclusive with precondition of another
- Mutex between literals (state level)
  - One literal is negation of another at the same level
  - Inconsistent support: actions that support the literals are mutex

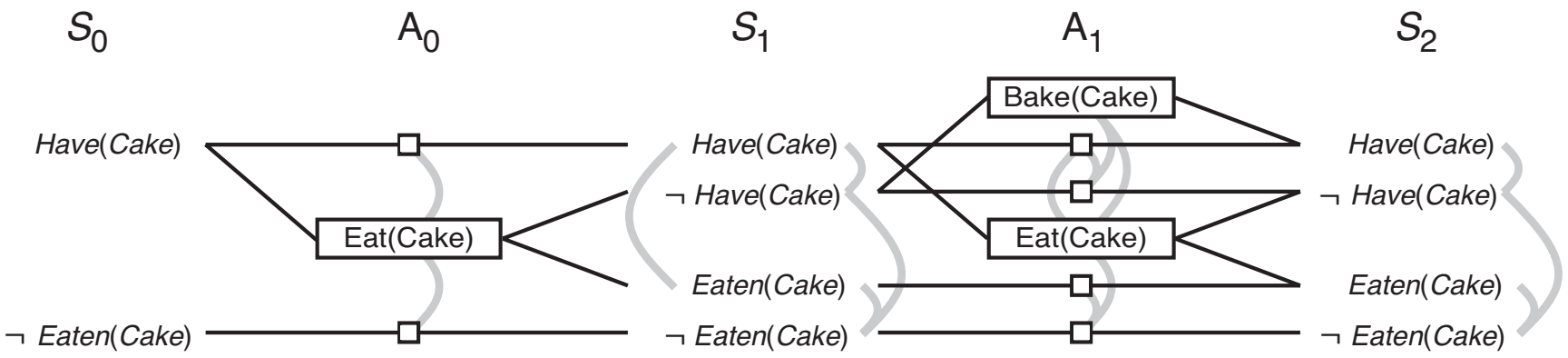
Example: Have your cake and eat it too!

- **Initial state**
  $Have(Cake)$
- **Goal**
  $Have(Cake), Eaten(Cake)$
- **Operators**

$Eat(Cake)$:
Pre: $Have(Cake)$
Eff: $\neg Have(Cake), Eaten(Cake)$

$Bake(Cake)$:
Pre: $\neg Have(Cake)$
Eff: $Have(Cake)$

Example: Have your cake and eat it too!



- Add states/actions until graph **levels off**
  - Two consecutive levels are identical

GRAPHPLAN Algorithm

```
function GRAPHPLAN(problem) returns solution or failure

  graph  ← INITIAL-PLANNING-GRAPH(problem)
  goals  ←  GOALS[problem]

  loop do
    if goals all non-mutex in last level of graph then do
      solution ← EXTRACT-SOLUTION (graph, goals, LENGTH(graph))

        if solution ≠ failure then return solution
        else if NO-SOLUTION-POSSIBLE(graph) then return failure

    graph  ← EXPAND-GRAPH(graph, problem)
```

- Builds planning graph until goals show up as non-mutex
- Searches for a plan that solves the problem
  - Regression search from goals

The situation calculus

- Planning using logic and resolution
  - FOL representation of the planning problem
  - Resolution to find a solution
- **Key idea:** represent a snapshot of the world, called a "situation" explicitly
- **Fluents**—statements that are true or false in any given situation, e.g., "I am at home"
- Actions map situations to situations

Blocks world example

- A move action:
  Move(*x, loc*)
- Use of the Result function:
  Result(*Move*(*x, loc*), *state*) → the state resulting from doing the *Move* action
- An axiom about moving:
  ∀*x* ∀*loc* ∀*s* [*At*(*x, loc, Result*(*Move*(*x, loc*), *s*))]
  If you move some object to a location, then in the resulting state that object is at that location
- *At*(*B1*, *Table*, *S0*)
- *At*(*B1, Top*(*B2*), *Result*(*Move*(*B1, Top*(*B2*)), *S0*))
  using the axiom

Monkeys and bananas problem

- The **monkey-and-bananas problem** is faced by a **monkey** standing under some **bananas** which are hanging out of reach from the ceiling. There is a box in the corner of the room that will enable the **monkey** to reach the **bananas** if he climbs on it.

- Use situation calculus to represent this problem and solve it using resolution theorem proving

Representation of Monkey/banana problem

- **Fluents**
  - $At(x, loc, s)$
  - $On(x, y, s)$
  - $Reachable(x, Bananas, s)$
  - $Has(x, y, s)$
- **Other predicates**
  - $Moveable(x)$, $Climbable(x)$
  - $Can\text{-}move(x)$
- **Actions**
  - $Climb\text{-}on(x, y)$
  - $Reach(x, y)$
  - $Move(x, loc)$
  - $Push(x, y, loc)$

Constants

BANANAS
MONKEY
BOX
S0
CORNER
UNDER-BANANAS

Monkey/bananas axioms

1. $\forall x1, s1\ [\ Reachable(x1, BANANAS, s1) \Rightarrow$
   $Has(x1, BANANAS, Result(Reach(x1, BANANAS), s1))\ ]$

   If a person (or monkey!) can reach the bananas ,then the result of reaching them is to have them

2. $\forall s2\ [At(BOX, UNDER\text{-}BANANAS, s2) \wedge On(MONKEY, BOX, s2) \Rightarrow$
   $Reachable(MONKEY, BANANAS, s2)$

   If a box is under the bananas and the monkey is on the box then the monkey can reach the bananas

3. $\forall x3,loc3,s3\ [\ Can\text{-}move(x3) \Rightarrow$
   $At(x3, loc, Result(Move(x3, loc3), s3))\ ]$

   The result of moving to a location is to be at that location

Monkey/bananas axioms (cont.)

4. $\forall x4, y4, s4\ [\exists loc4\ [At(x4, loc4, s4) \wedge At(y4, loc4, s4)] \wedge Climbable(y4) \Rightarrow On(x4, y4, Result(Climb\text{-}on(x4, y4), s4))]$
   The result of climbing on an object is to be on the object

5. $\forall x5, y5, loc5, s5\ [\exists loc\ [At(x5, loc0, s) \wedge At(y5, loc0, s5)\ ] \wedge Moveable(y5) \Rightarrow At(y5, loc5, Result(Push(x5, y5, loc5), s5))$
   The result of x pushing y to a location is y is at that location

6. $<same> \Rightarrow At(x6, loc6, Result(Push(x6, y6, loc6), s6))\ ]$
   The result of x pushing y to a location is x is at that location

Monkey/bananas problem (cont.)

- **Initial state** (S0)
  F1. *Moveable*(*BOX*)
  F2. *Climbable*(*BOX*)
  F3. *Can-move*(*MONKEY*)
  F4. *At*(*BOX, CORNER, S0*)
  F5. *At*(*MONKEY, UNDER-BANANAS, S0*)
- **Goal**
  Has(*MONKEY, BANANAS, s*)
- **How to solve?**
  - Convert to clause form
  - Apply resolution to prove
    Has(*MONKEY, BANANAS*, ***Result*(*Reach*(. . .), *Result*(. .) . .), *S0***)
    which gives the plan in reverse order
  - Don't forget to standardize variables!
  - For the proof we need two additional **frame axioms**…

Frame axioms for monkey/bananas problem

7. $\forall x, y, loc, s\ [\ At(x, loc, s) \Rightarrow$
   $At(x, loc, Result(Move(y, loc), s))\ ]$

   The location of an object does not change as a result of someone moving to the same location

8. $\forall x, y, loc, s\ [\ At(x, loc, s) \Rightarrow$
   $At(x, loc, Result(Climb\text{-}on(y, x), s))\ ]$

   The location of an object does not change as a result of someone climbing on it

Limitation of situation calculus

- **Frame problem** (it's back again!!)
- I go from home to the store—new situation *S'*
- In *S'*
  - The store still sells chips
  - My age is still the same
  - Los Angeles is still the largest city in California…

- How can we efficiently represent everything that doesn't change?
  Successor state axioms…

Successor state axioms

- Normally, things stay true from one state to the next—unless an action **changes** them

  $At(p, loc, Result(a, s))$ iff $a = Go(p, x)$

  or $[At(p, loc, s)$ and $a \mathrel{!}= Go(p, y)]$

- We need one or more of these for every fluent
- Now we can use theorem proving (or possibly backward chaining) to deduce a plan
  - Still not very practical
- Need for **effective heuristics** for situation calculus