Automated Planning

Outline

I. Planning domain definition language (PPDL)

II. Example domains

III. Algorithms for classical planning

* Figures are from the textbook site.

I. Classical Planning

The task of finding *a sequence of actions* to accomplish a goal in a discrete, deterministic, static, fully observable environment.

I. Classical Planning

The task of finding *a sequence of actions* to accomplish a goal in a discrete, deterministic, static, fully observable environment.

Planning approaches learned so far :

- Problem solving via informed search
- Propositional logical agent (for the wumpus world)

I. Classical Planning

The task of finding *a sequence of actions* to accomplish a goal in a discrete, deterministic, static, fully observable environment.

Planning approaches learned so far :

- Problem solving via informed search
- Propositional logical agent (for the wumpus world)

Their limitations:

- ♠ Requirement of ad hoc heuristics for a new domain
- ♠ Explicit representation of an exponentially large state space

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables

single predicate

time varying aspect

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables | single predicate | time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables | single predicate | time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables | single predicate | time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

- Database semantics: any unmentioned fluents are false.

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables | single predicate | time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

- Database semantics: any unmentioned fluents are false.

- Action schema: a family of ground actions

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables | single predicate | time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

- Database semantics: any unmentioned fluents are false.

- Action schema: a family of ground actions

$Action(Fly(P_1, SFO, JFK),$
  PRECOND: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
  POSTCOND: $\neg At(P_1, SFO) \wedge At(P_1, JFK))$

Planning Domain Definition Language (PDDL)

- State: a conjunction of ground atomic fluents

no variables — single predicate — time varying aspect

$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$ // a state in package delivery

- Database semantics: any unmentioned fluents are false.

- Action schema: a family of ground actions

$Action(Fly(P_1, SFO, JFK),$
PRECOND: $At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$
POSTCOND: $\neg At(P_1, SFO) \wedge At(P_1, JFK))$

An action a is *applicable* in state s if s entails the precondition of a.

Initial State and Goal

- Result of a in s

$$\text{RESULT}(s,a) = (\underbrace{s - \text{DEL}(a)}_{\text{delete list}}) \cup \underbrace{\text{ADD}(a)}_{\text{add list}}$$

  - Remove the negative literals in the action's effects.
  - Add the positive literals in the action's effects.

Initial State and Goal

- Result of a in s

delete list   add list

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

  - Remove the negative literals in the action's effects.
  - Add the positive literals in the action's effects.

Action(*Fly*(P_1, *SFO*, *JFK*),
  PRECOND: *At*(P_1, *SFO*) ∧ *Plane*(P_1) ∧ *Airport*(*SFO*) ∧ *Airport*(*JFK*)
  POSTCOND: ¬*At*(P_1, *SFO*) ∧ *At*(P_1, *JFK*))

Initial State and Goal

- Result of a in s

delete list   add list

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

  - Remove the negative literals in the action's effects.
  - Add the positive literals in the action's effects.

Action(*Fly*(P_1, *SFO*, *JFK*),
  PRECOND: *At*(P_1, *SFO*) ∧ *Plane*(P_1) ∧ *Airport*(*SFO*) ∧ *Airport*(*JFK*)
  POSTCOND: ¬*At*(P_1, *SFO*) ∧ *At*(P_1, *JFK*)

$\Downarrow$

Remove *At*(P_1, *SFO*) and add *At*(P_1, *SFK*).

Initial State and Goal

- **Result** of a in s

$$\text{Result}(s, a) = (s - \overset{\text{delete list}}{\text{DEL}(a)}) \cup \overset{\text{add list}}{\text{ADD}(a)}$$

  - Remove the negative literals in the action's effects.
  - Add the positive literals in the action's effects.

Action(*Fly*(P_1, *SFO*, *JFK*),
PRECOND: *At*(P_1, *SFO*) ∧ *Plane*(P_1) ∧ *Airport*(*SFO*) ∧ *Airport*(*JFK*)
POSTCOND: ¬*At*(P_1, *SFO*) ∧ *At*(P_1, *JFK*)

$\Downarrow$

Remove *At*(P_1, *SFO*) and add *At*(P_1, *SFK*).

- **Initial state**: conjunction of ground fluents

Initial State and Goal

- Result of a in s

delete list add list

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

  - Remove the negative literals in the action's effects.
  - Add the positive literals in the action's effects.

Action(*Fly*(P_1, *SFO*, *JFK*),
  PRECOND: *At*(P_1, *SFO*) ∧ *Plane*(P_1) ∧ *Airport*(*SFO*) ∧ *Airport*(*JFK*)
  POSTCOND: ¬*At*(P_1, *SFO*) ∧ *At*(P_1, *JFK*)

$\Downarrow$

Remove *At*(P_1, *SFO*) and add *At*(P_1, *SFK*).

- Initial state: conjunction of ground fluents
- Goal: conjunction of literals (positive or negative) possibly with variables

At(C_1, *SFK*) ∧ ¬*At*(C_2, *SFO*) ∧ *At*(p, *SFK*)

II. Example 1: Air Cargo Transport

- Three actions: *Load*, *Unload*, and *Fly*.
- Predicate *In*(c, p): cargo c is inside plane p.
- Predicate *At*(x, a): object x (either plane or cargo) is at airport a.

II. Example 1: Air Cargo Transport

- Three actions: *Load*, *Unload*, and *Fly*.
- Predicate *In*(c, p): cargo c is inside plane p.
- Predicate *At*(x, a): object x (either plane or cargo) is at airport a.

PPDL:

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

Example 1 (cont'd)

- When a plane flies from one airport to another, all the cargo inside the plane goes with it.
  - PDDL does not have ∀, instead we say that a piece of cargo ceases to be *At* anywhere when it is *In* a plane.
  - Namely, the cargo only becomes *At* the new airport when it is unloaded.

Example 1 (cont'd)

- When a plane flies from one airport to another, all the cargo inside the plane goes with it.
  - PDDL does not have ∀, instead we say that a piece of cargo ceases to be *At* anywhere when it is *In* a plane.
  - Namely, the cargo only becomes *At* the new airport when it is unloaded.

Plan:

[*Load*$(C_1, P_1,$ *SFO*$)$, *Fly*$(P_1,$ *SFO*, *JFK*$)$, *Unload*$(C_1, P_1,$ *JFK*$)$,
Load$(C_2, P_2,$ *JFK*$)$, *Fly*$(P_1,$ *JFK*, *SFO*$)$, *Unload*$(C_2, P_2,$ *SFO*$)$]

Example 2: The Spare Tire Problem

- Goal: mount a good spare tire properly onto the car’s axle.
- Initial state: a flat tire on the axle and a good tire in the trunk.

Example 2: The Spare Tire Problem

- Goal: mount a good spare tire properly onto the car's axle.
- Initial state: a flat tire on the axle and a good tire in the trunk.
- Four actions:
  - removing the spare tire from the trunk.
  - removing the flat tire from the axle.
  - putting the spare tire on the axle.
  - leaving the car unattended overnight.

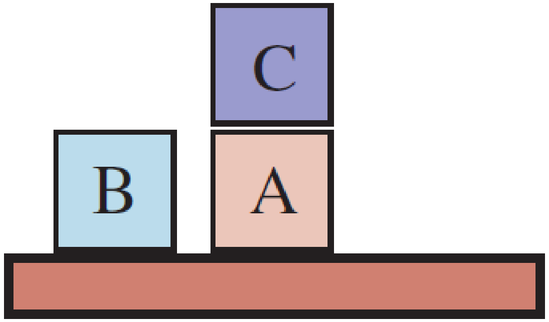
Example 2 (cont'd)

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
  PRECOND: $At(obj, loc)$
  EFFECT: $\neg At(obj, loc) \wedge At(obj, Ground))$
$Action(PutOn(t, Axle),$
  PRECOND: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Spare, Axle)$
  EFFECT: $\neg At(t, Ground) \wedge At(t, Axle))$
$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
    $\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk))$

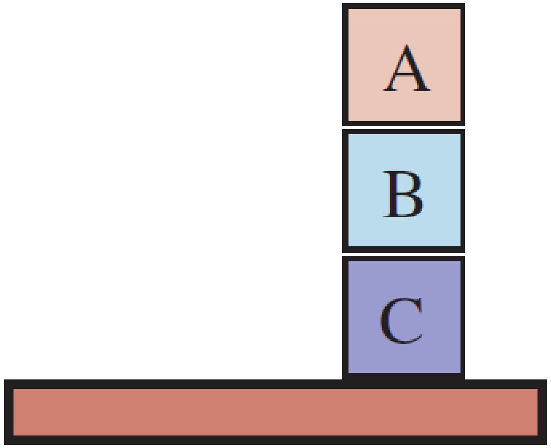
// Tires will disappear because the car is parked in a bad
// neighborhood.

Example 3: The Blocks World

- Identical cube-shaped blocks sit on a table.
- Blocks can be stacked one on top of another.
- A robotic arm can pick up a block one at a time (thus it cannot pick up a block with another one on top of it.)
- The robotic arm can place the block either on the table or on top of another block.



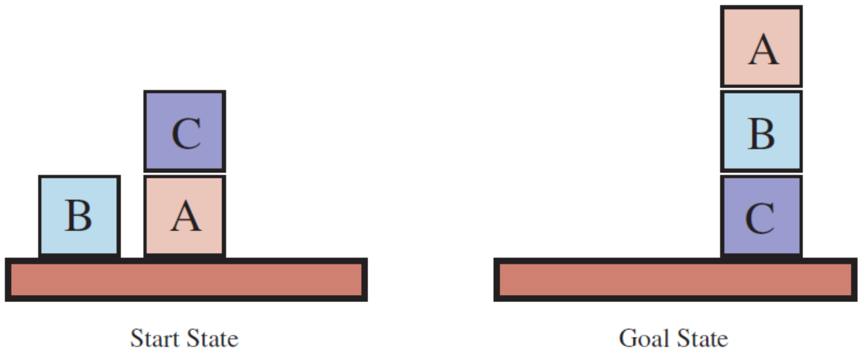
Start State



Goal State

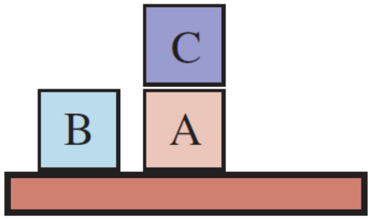
Building a Three-Block Tower

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$

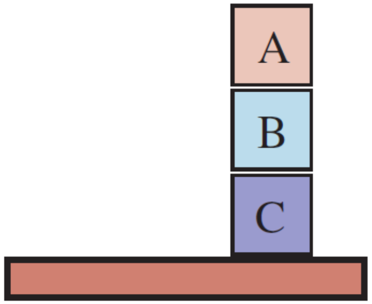


Building a Three-Block Tower

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$ // move block b from the top of x to the top of y.
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



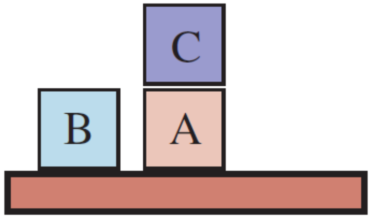
Start State



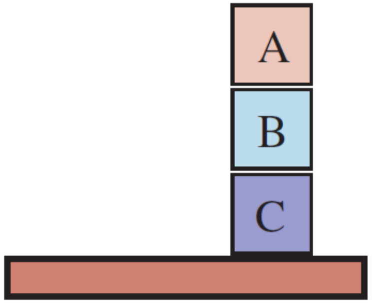
Goal State

Building a Three-Block Tower

$Init(On(A, Table) \land On(B, Table) \land On(C, A)$
$\quad \land\ Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A, B) \land On(B, C))$
$Action(Move(b, x, y),$ // move block b from the top of x to the top of y.
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$\qquad (b \neq x) \land (b \neq y) \land (x \neq y),$ // neither x nor y can be the table (which does not need
$\quad$ EFFECT: $On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$ // to be clear before or after
$Action(MoveToTable(b, x),$ // the move).
$\quad$ PRECOND: $On(b, x) \land Clear(b) \land Block(b) \land Block(x),$
$\quad$ EFFECT: $On(b, Table) \land Clear(x) \land \neg On(b, x))$



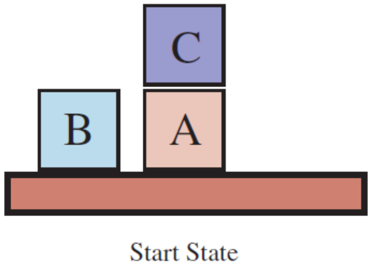
Start State

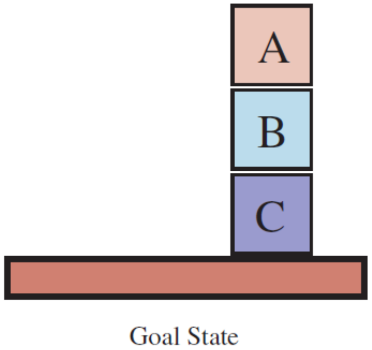


Goal State

Building a Three-Block Tower

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C) \wedge Clear(Table))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$ // move block b from the top of x to the top of y.
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$ // neither x nor y can be the table (which does not need
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$ // to be clear before or after
$Action(\boxed{MoveToTable}(b, x),$ // the move).
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge Block(x),$
$\quad$ EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$





Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

At(*obj*,*loc*)

variables

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

At(*obj*,*loc*)

variables

- Unify the current state with the preconditions of every action schema.

$\Downarrow$

Apply the substitution θ to yield a ground action.

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

At(*obj*,*loc*)

variables

- Unify the current state with the preconditions of every action schema.

$\Downarrow$

Apply the substitution θ to yield a ground action.

- Each schema may unify in multiple ways.

At(*obj*,*loc*)

At(*Flat*,*Axle*) *At*(*Spare*,*Trunk*)

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

At(*obj*,*loc*)

variables

- Unify the current state with the preconditions of every action schema.

$\Downarrow$

Apply the substitution θ to yield a ground action.

- Each schema may unify in multiple ways.

At(*obj*,*loc*)

At(*Flat*,*Axle*) *At*(*Spare*,*Trunk*)

$\theta = \{obj/Flat,\ loc/Axle\}$

Planning: Forward State-Space Search

Any of the heuristic search algorithms is applicable.

- Actions(s), applicable actions in a state s, are grounded instantiations of the *action schemas* from replacing variables with constant values.

At(*obj*,*loc*)

variables

- Unify the current state with the preconditions of every action schema.

$\Downarrow$

Apply the substitution θ to yield a ground action.

- Each schema may unify in multiple ways.

At(*obj*,*loc*)

At(*Flat*,*Axle*) *At*(*Spare*,*Trunk*)

$\theta = \{obj/Flat,\ loc/Axle\}$ $\theta = \{obj/Spare,\ loc/Trunk\}$

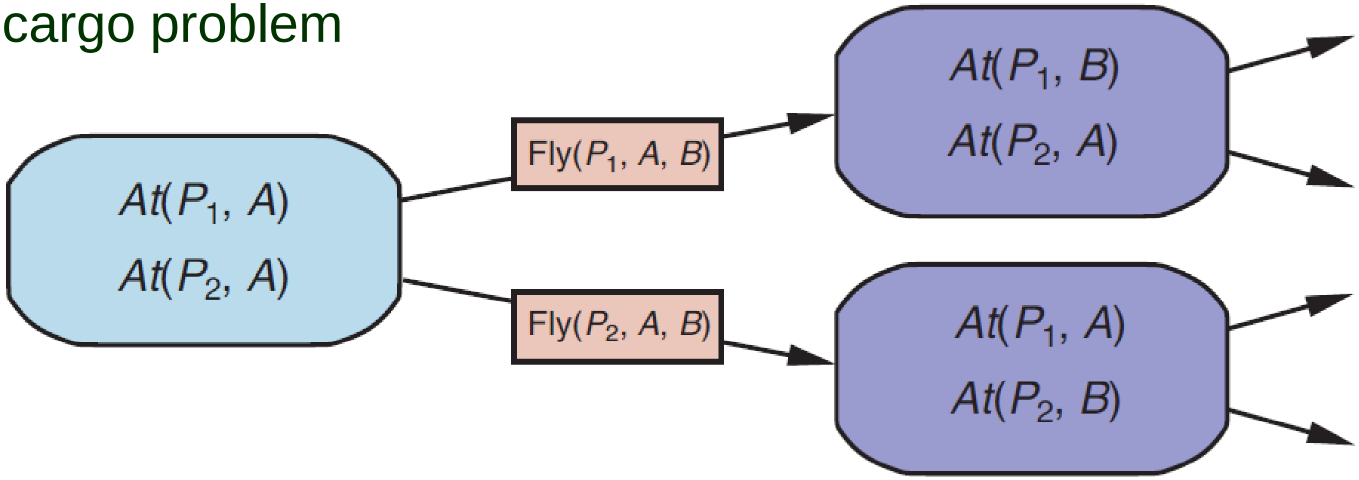
Combinatorial Explosion

State space can get too big if we match a state with every action in every possible way!

Combinatorial Explosion

State space can get too big if we match a state with every action in every possible way!

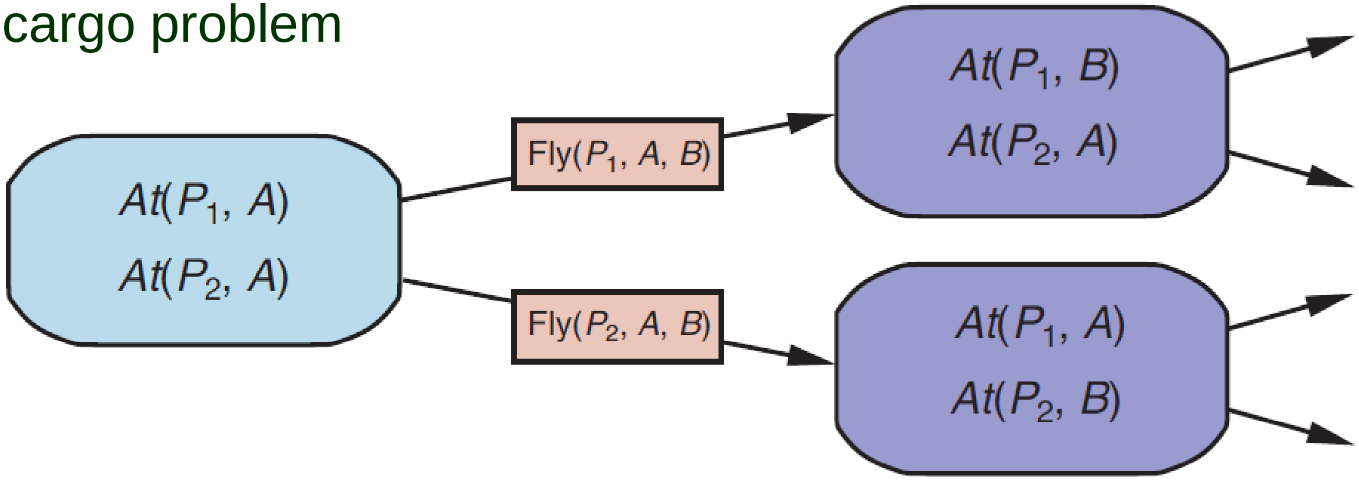
Airport cargo problem



Combinatorial Explosion

State space can get too big if we match a state with every action in every possible way!

Airport cargo problem

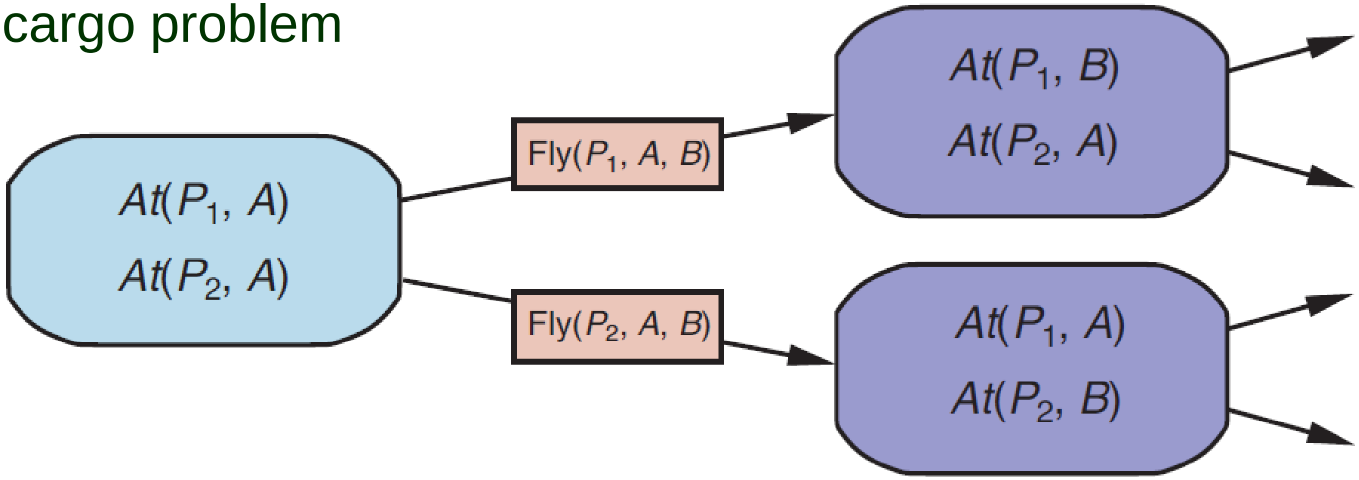


- 10 airports, 5 planes and 20 pieces of cargo per airport initially.
- Goal: move all the cargos at airport A to airport B.

Combinatorial Explosion

State space can get too big if we match a state with every action in every possible way!

Airport cargo problem



- 10 airports, 5 planes and 20 pieces of cargo per airport initially.
- Goal: move all the cargos at airport A to airport B.
- Huge average branching factor.
  - ♣ Each of the 50 planes can fly to 9 other airports.
  - ♣ Each of the 200 pieces of cargo can be either unloaded from or loaded into any plane at its airport.

Backward (Regression) Search

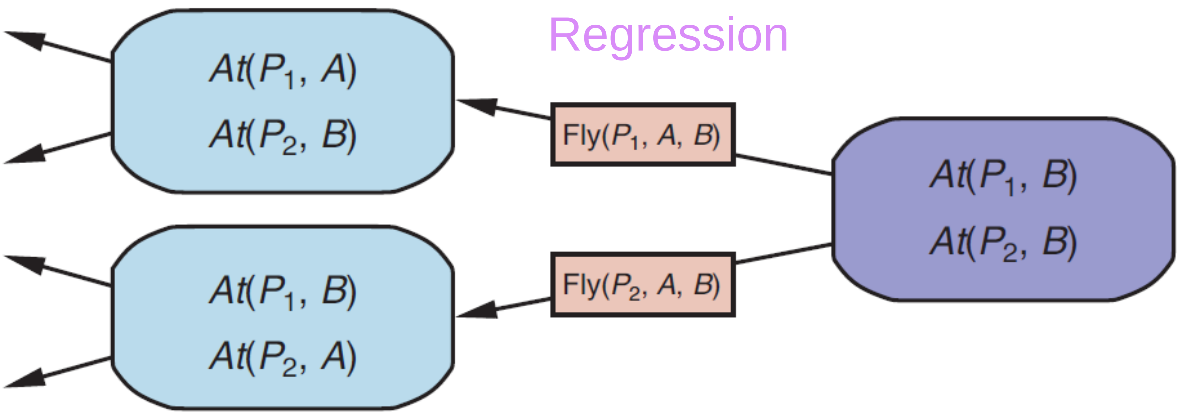
Consider *relevant* not just applicable actions.

unifying with one of the goal literals,
but not negating any part of the goal.

Backward (Regression) Search

Consider *relevant* not just applicable actions.

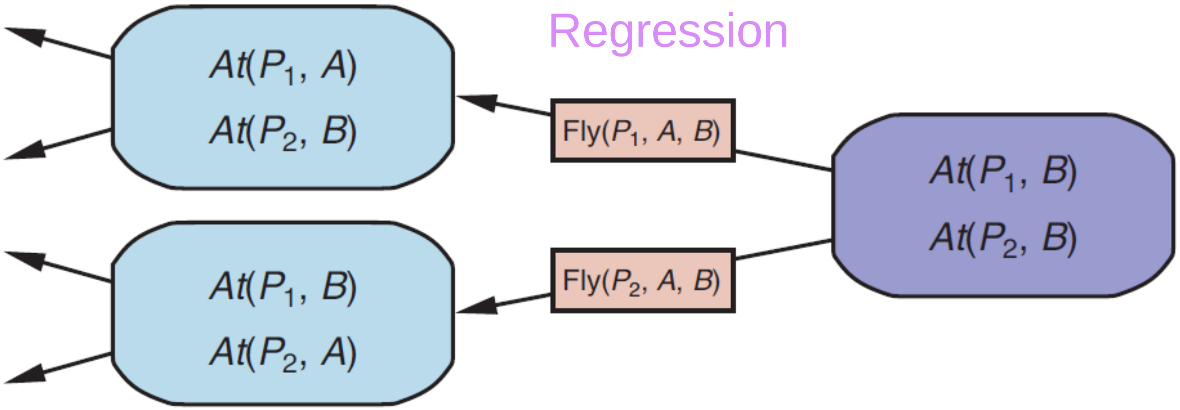
unifying with one of the goal literals,
but not negating any part of the goal.



Backward (Regression) Search

Consider *relevant* not just applicable actions.

unifying with one of the goal literals,
but not negating any part of the goal.



Given a goal g and an action a, regression from g over a yields a state g':

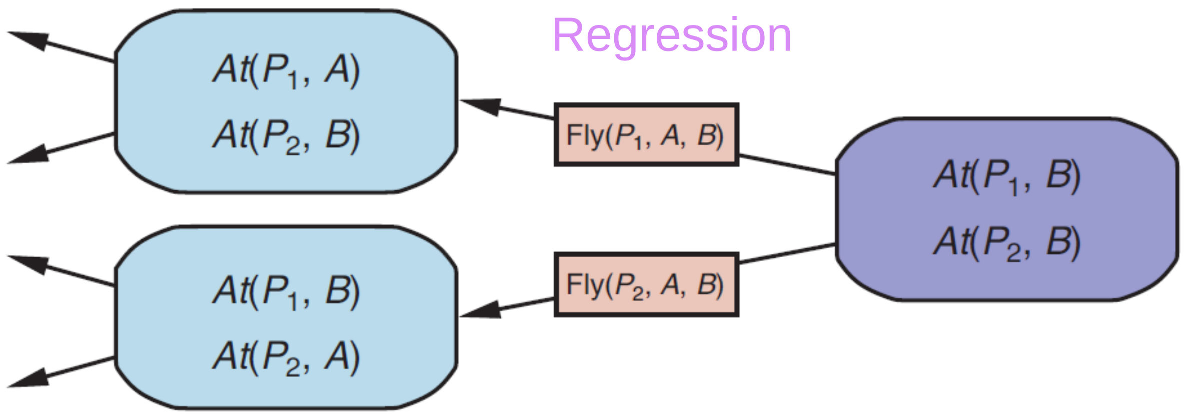
$$\text{Pos}(g') = (\text{Pos}(g) - \text{Add}(a)) \cup \text{Pos}(\mathit{Precond}(a))$$

positive literals

Backward (Regression) Search

Consider *relevant* not just applicable actions.

unifying with one of the goal literals,
but not negating any part of the goal.



Given a goal g and an action a, regression from g over a yields a state g':

positive literals

$$\text{Pos}(g') = (\text{Pos}(g) - \text{Add}(a)) \cup \text{Pos}(Precond(a))$$

negative literals

$$\text{Neg}(g') = (\text{Neg}(g) + \text{Del}(a)) \cup \text{Neg}(Precond(a))$$

Example

Goal: $g = At(C_2, SFO)$

$Action(Unload(c,\ p,\ a),$
  $\textsc{Precond}: In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
  $\textsc{Effect}: At(c,\ a) \wedge \neg In(c,\ p))$

Example

Goal: $g = At(C_2, SFO)$

$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\boxed{At(c, a)} \land \neg In(c, p))$

Example

Goal: $g = At(C_2, SFO)$

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

$\theta = \{c/C_2, a/SFO\}$

$At(C_2, SFO)$

Example

Goal: $g = At(C_2, SFO)$

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

$At(C_2, SFO) \longrightarrow \theta = \{c/C_2, a/SFO\}$

Unification under θ yields a new goal:

$g' = In(C_2, p') \wedge At(p', SFO) \wedge Cargo(C_2) \wedge Plane(p') \wedge Airport(SFO)$

name standardization
not to conflict p

Example

Goal: $g = At(C_2, SFO)$

$Action(Unload(c,\ p,\ a),$
  PRECOND: $In(c,\ p) \land At(p,\ a) \land Cargo(c) \land Plane(p) \land Airport(a)$
  EFFECT: $\boxed{At(c,\ a)} \land \neg In(c,\ p))$

$At(C_2, SFO)$

$\longrightarrow \theta = \{c/C_2, a/SFO\}$

Unification under θ yields a new goal:

$g' = In(C_2, p') \land At(p', SFO) \land Cargo(C_2) \land Plane(p') \land Airport(SFO)$

name standardization
not to conflict p