

CISC 6525

Assignment Project Exam Help

<https://powcoder.com>
Planning

Add WeChat powcoder

Chapter 10

Planning

- The Planning problem [Assignment Project Exam Help](#)
- Planning with State-space search
- Partial-order planning <https://powcoder.com>
- Planning graphs [Add WeChat powcoder](#)
- Planning with propositional logic
- Analysis of planning approaches

What is Planning

Generate sequence of actions to achieve a goal.

- States, actions and goals

Search for solution over abstract space of plans.

Assignment Project Exam Help

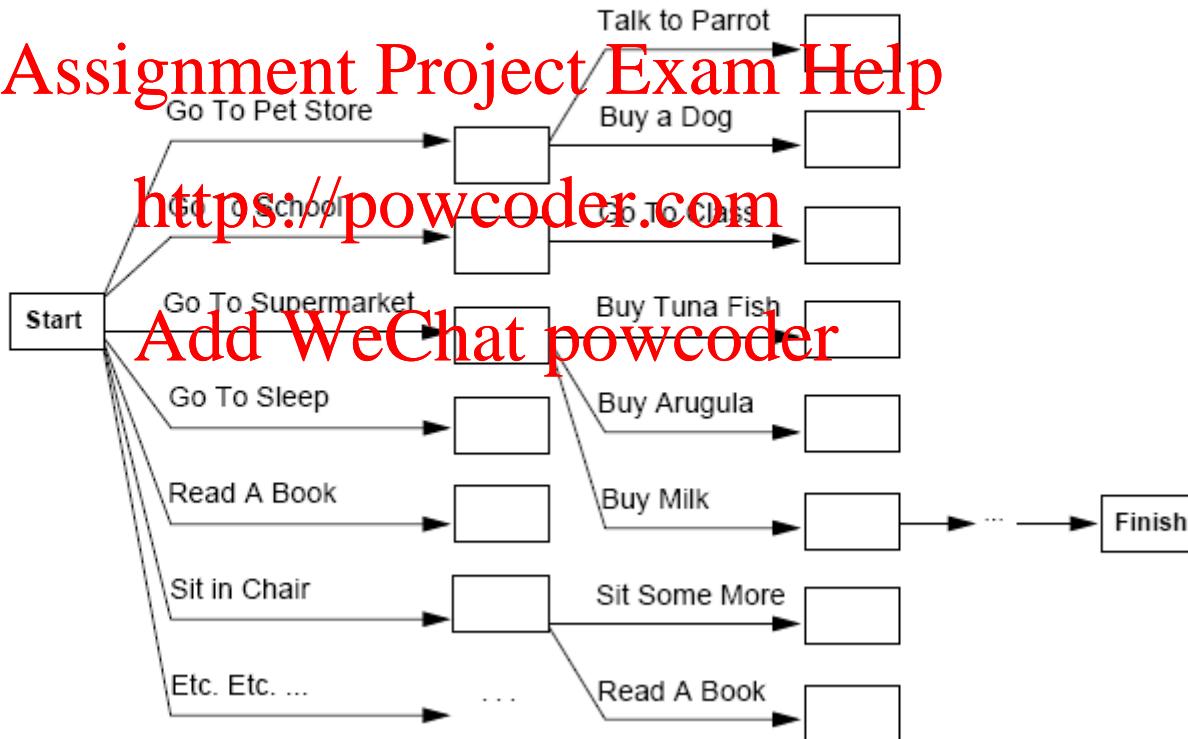
The *classical planning* environment:
fully observable, deterministic, finite, static
and discrete. **Add WeChat powcoder**

Assists humans in practical applications

- design and manufacturing
- military operations
- games
- space exploration

Planning as Search

Consider the task *get milk, bananas, and a cordless drill*. Standard search algorithms seem to fail miserably:



Difficulty of real world problems

Assume a problem-solving agent
using some search method ...

- Which actions are relevant?
 - Exhaustive search vs. backward search
- What is a good heuristic function?
 - Good estimate of the cost of the state?
 - Problem-dependent vs, -independent
- How to decompose the problem?
 - Most real-world problems are ***nearly*** decomposable.

Planning language features

Factored Representation of states

- Decompose the world in logical conditions and represent a state as a *conjunction of positive literals*.
 - Propositional literals: $Poor \wedge Unknown$
 - FO-literals (grounded and function-free):
 $At(Plane1, Melbourne) \wedge At(Plane2, Sydney)$
- Database Semantics(anything not mentioned is false; different names refer to different things)

Representation of goals

- Partially specified state and represented as a *conjunction of positive ground literals*
- A goal is *satisfied* if the state contains all literals in goal.

Planning language features

Representations of actions

- Action = ~~Assignment~~ Project Exam Help

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))

- = action schema (p, from, to need to be instantiated)

- Action name and parameter list
- Precondition (conj. of function-free literals)
- Effect (conj of function-free literals and P is True and not P is false)
- Add-list vs delete-list in Effect

Logic vs. Set Theory

Logic:

$$(a \in \text{ACTIONS}(s)) \Leftrightarrow s \models \text{PRECOND}(a),$$

<https://powcoder.com>

Set theory:

Add WeChat powcoder

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

Language semantics

How do actions affect states?

- An action is applicable if any state that satisfies the precondition.
- For FO action schema, applicability involves a substitution δ for the variables in the PRECOND.

$At(P1, JFK) \wedge At(P2, SFO) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO)$

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

With $\delta = \{p/P1, from/JFK, to/SFO\}$

Thus the action is applicable.

Language semantics

The result of executing action a in state s is the state s'

- s' is same as s except
 - Any positive literal P in the effect of a is added to s'
 - Any negative literal $\neg P$ is removed from s'

EFFECT: $\neg \text{At}(p, \text{from}) \wedge \text{At}(p, \text{to})$

$\text{At}(P1, \text{SFO}) \wedge \text{At}(P2, \text{SFO}) \wedge \text{Plane}(P1) \wedge \text{Plane}(P2) \wedge \text{Airport}(\text{JFK}) \wedge \text{Airport}(\text{SFO})$

Add WeChat powcoder

- STRIPS assumption:
every literal NOT in the effect remains unchanged

Avoids representational frame problem
(i.e. what remains unchanged by an action)

Expressiveness and extensions

STRIPS is simplified

- Important limit: function-free literals
 - Allows for propositional representation
 - Function symbols lead to infinitely many states and actions

Assignment Project Exam Help

<https://powcoder.com>

Recent extension: Action description language (ADL)

Action(*Fly(p:Plane, from: Airport, to: Airport)*)
PRECOND: At(*p,from*) \wedge (*from* \neq *to*)
EFFECT: \neg At(*p,from*) \wedge At(*p,to*)

Add WeChat powcoder

Standardization : Planning domain definition language (PDDL)

many versions, can have negative literals in PRECOND and EFFECT

Example: air cargo transport

Init(At(C1, SFO) \wedge At(C2, JFK) \wedge At(P1, SFO) \wedge At(P2, JFK) \wedge
Cargo(C1) \wedge Cargo(C2) \wedge Plane(P1) \wedge Plane(P2) \wedge Airport(JFK) \wedge Airport(SFO))

Goal(At(C1, JFK) \wedge At(C2, SFO))

Action(Load(c,p,a))

Assignment Project Exam Help

PRECOND: At(c,a) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)

EFFECT: \neg At(c,a) \wedge In(c,p))

<https://powcoder.com>

Action(Unload(c,p,a))

Add WeChat powcoder

PRECOND: In(c,p) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)

EFFECT: At(c,a) \wedge \neg In(c,p))

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))

[Load(C1,P1,SFO), Fly(P1,SFO,JFK), Unload(C1,P1,JFK),
Load(C2,P2,JFK), Fly(P2,JFK,SFO), Unload(C2,P2,SFO)]

Example: Spare tire problem

Init(At(Flat, Axle) \wedge At(Spare, trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk))

PRECOND: At(Spare, Trunk)

EFFECT: \neg At(Spare, Trunk) \wedge At(Spare, Ground))

Assignment Project Exam Help

Action(Remove(Flat, Axle))

PRECOND: At(Flat, Axle)

<https://powcoder.com>

EFFECT: \neg At(Flat, Axle) \wedge At(Flat, Ground))

Action(PutOn(Spare, Axle))

Add WeChat powcoder

PRECOND: At(Spare, Ground) \wedge \neg At(Flat, Axle)

EFFECT: At(Spare, Axle) \wedge \neg At(Spare, Ground))

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))

This example goes beyond STRIPS: negative literal in pre-condition (ADL/PDDL description)

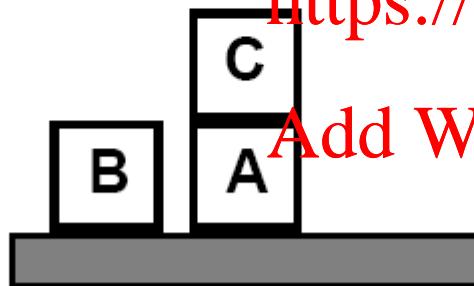
Blocks World

Example: Blocks world

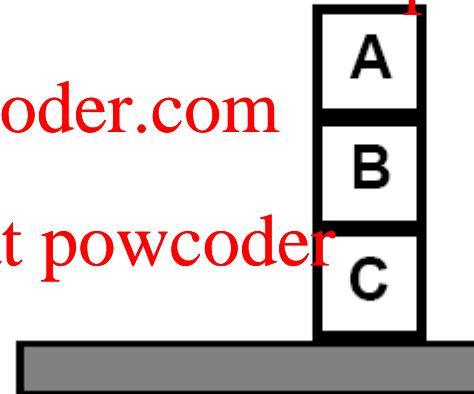
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



Start State



Goal State

Example: Blocks world

Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(A) \wedge Clear(B) \wedge Clear(C))

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

Assignment Project Exam Help

Action(Move(b, x, y))

PRECOND: On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b)
 $\wedge (b \neq x) \wedge (b \neq y) \wedge (x \neq y)$

EFFECT: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))

Action(MoveToTable(b, x))

PRECOND: On(b, x) \wedge Clear(b) \wedge Block(b) \wedge ($b \neq x$)

EFFECT: On($b, Table$) \wedge Clear(x) \wedge \neg On(b, x))

Spurious actions are possible: Move(B,C,C)

Planning Algorithms: Planning with state-space search

Both forward and backward search possible

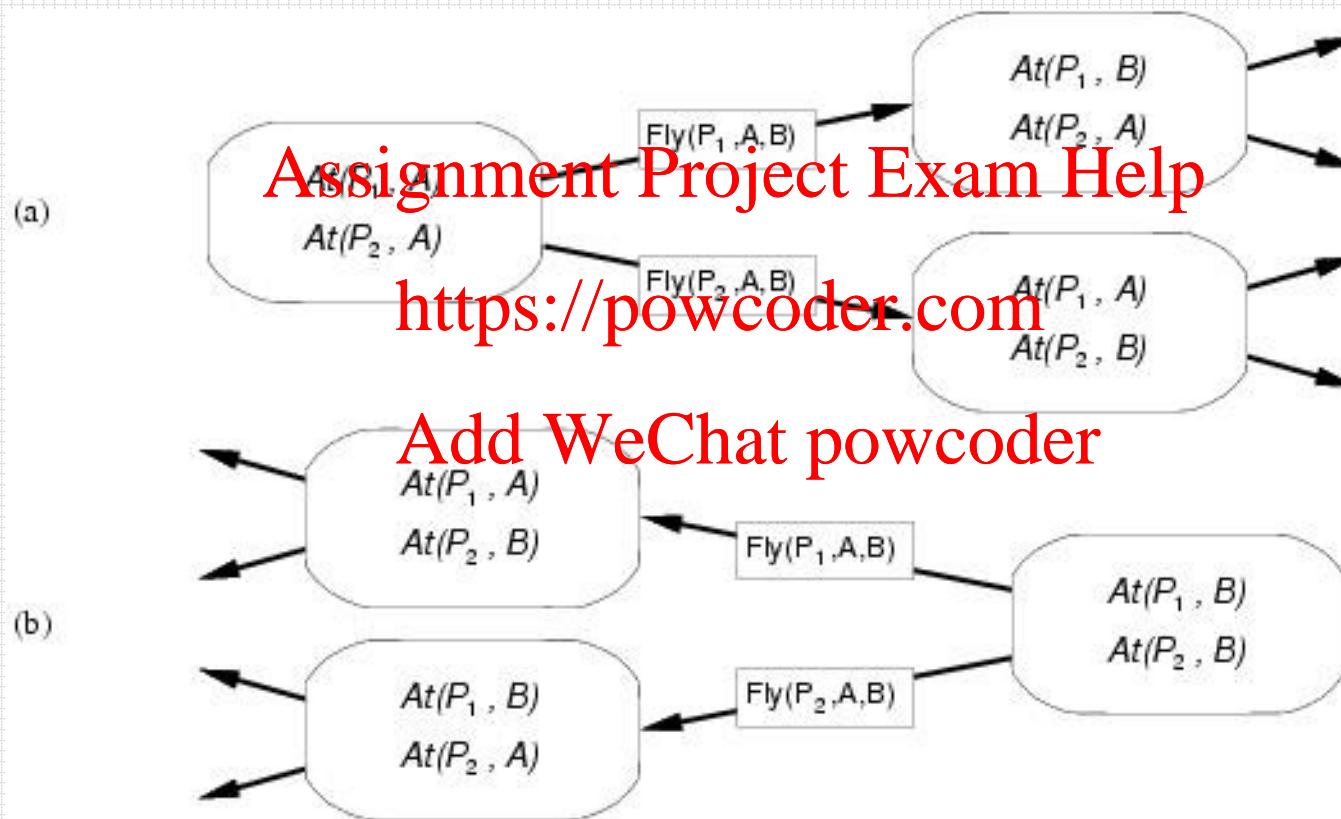
Progression planners

- **forward** state-space search
<https://powcoder.com>
- Consider the effect of all possible actions in a given state

Regression planners

- **backward** state-space search
- To achieve a goal, what must have been true in the previous state.

Progression and regression



Progression algorithm

Formulation as state-space search problem:

- Initial state = initial state of the planning problem
 - Literals not appearing are false
- Actions = those whose preconditions are satisfied
 - Add positive effects, delete negative
- Goal test = does the state satisfy the goal
- Step cost = each action costs 1

No functions ... any graph search that is complete is a complete planning algorithm.

- E.g. A*

Potentially inefficient:

- (1) irrelevant action problem
- (2) good heuristic required for efficient search

Regression algorithm

How to determine predecessors?

- What are the states from which applying a given action leads to the goal?

Goal state = $A(C1, B) \wedge A(C2, B) \wedge \dots \wedge A(C20, B)$

Relevant action for first conjunct: *Unload(C1, p, B)*

Works only if pre-conditions are satisfied.

Previous state= *In(C1, p, B) \wedge At(C2, B) \wedge ... \wedge At(C20, B)*

Subgoal *At(C1, B)* should not be present in this state.

- Actions must not undo desired literals (**consistent**)

Add WeChat powcoder

Main advantage: only relevant actions are considered.

- Often much lower branching factor than forward search.
- Deals in sets of states, so more difficult to find a good heuristic.

Regression algorithm

General process for predecessor construction

- Give a goal description G
- Let A be an action that is relevant and consistent
- The predecessors is as follows:
 - Any positive effects of A that appear in G are deleted.
 - Each precondition literal of A is added , unless it already appears.

Any standard search algorithm can be added to perform the search.

Termination when predecessor satisfied by initial state.

- In FO case, satisfaction might require a substitution.

Heuristics for state-space search

Neither progression or regression are very efficient without a good heuristic

Assignment Project Exam Help

- How many actions are needed to achieve the goal?
- Exact solution is NP hard, find a good estimate
<https://powcoder.com>

Two approaches to find admissible heuristic:

- The optimal solution to the relaxed problem.
 - Remove all preconditions from actions
- The subgoal independence assumption:

The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.

Partial-order planning

Progression and regression
planning are totally ordered plan
search forms.

<https://powcoder.com>

- They cannot take advantage of problem decomposition.
 - Decisions must be made on how to sequence actions on all the subproblems

Least commitment strategy:

- Delay choice during search

Shoe example

Goal(RightShoeOn ∧ LeftShoeOn)

Init()

Action(RightShoe, PRECOND: RightSockOn)

EFFECT: RightShoeOn)

Action(RightSock, PRECOND:

EFFECT: RightSockOn)

Action(LeftShoe, PRECOND: LeftSockOn)

EFFECT: LeftShoeOn)

Action(LeftSock, PRECOND:

EFFECT: LeftSockOn)

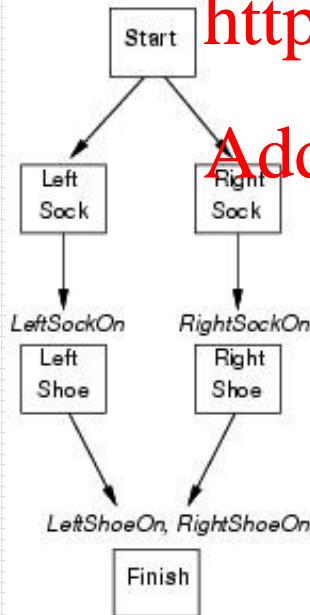
Planner: combine two action sequences (1)leftsock,
leftshoe (2)rightsock, rightshoe

Partial-order planning(POP)

Any planning algorithm that can place two actions into a plan without deciding which comes first is a PO plan.

Assignment Project Exam Help

Partial Order Plan:



Total Order Plans:



<https://powcoder.com>

Add WeChat powcoder

POP as a search problem

States are (mostly unfinished) plans.

- The empty plan contains only start and finish actions.

Assignment Project Exam Help
Each plan has 4 components:

- A set of actions (steps of the plan)
- A set of ordering constraints: $A < B$ (A before B)
 - Cycles represent contradictions.
- A set of causal links $A \xrightarrow{p} B$
 - The plan may not be extended by adding a new action C that conflicts with the causal link. (if the effect of C is $\neg p$ and if C could come after A and before B)
- A set of open preconditions.
 - If precondition is not achieved by action in the plan.

Example of final plan

Actions={Rightsock, Rightshoe, Leftsock, Leftshoe,
Start, Finish}

Assignment Project Exam Help

Orderings={Rightsock < Rightshoe; Leftsock <
Leftshoe}

Add WeChat powcoder

Links={Rightsock -*Rightsockon->* Rightshoe,
Leftsock -*Leftsockon->* Leftshoe,
Rightshoe -*Rightshoeon->* Finish, ...}

Open preconditions={}

POP as a search problem

A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.

<https://powcoder.com>

A consistent plan with no open preconditions is a *solution*.

A partial order plan is executed by repeatedly choosing *any* of the possible next actions.

- This flexibility is a benefit in non-cooperative environments.

Solving POP

Assume propositional planning problems:

- The initial plan contains *Start* and *Finish*, the ordering constraint $\text{Start} < \text{Finish}$, no causal links, all the preconditions in *Finish* are open.
- Successor function :
 - picks one open precondition p on an action B and
 - generates a successor plan for every possible consistent way of choosing action A that achieves p .
- Test goal

Enforcing consistency

When generating successor plan:

- The causal link $A -p > B$ and the ordering constraint $A < B$ is added to the plan.
 - If A is new also add start $< A$ and $A < B$ to the plan
- Resolve conflicts between new causal link and all existing actions
- Resolve conflicts between action A (if new) and all existing causal links.

Process summary

Operators on partial plans

- Add link from existing Plan to open precondition.
- Add a step to fulfill an open condition.
- Order one step w.r.t another to remove possible conflicts

Gradually move from incomplete/vague plans
to complete/correct plans

*Backtrack if an open condition is
unachievable or if a conflict is irresolvable.*

Example: Spare tire problem

Init(At(Flat, Axle) \wedge At(Spare, trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk)

PRECOND: At(Spare, Trunk)

EFFECT: \neg At(Spare, Trunk) \wedge At(Spare, Ground))

Action(Remove(Flat,Axle)

PRECOND: At(Flat,Axle)

EFFECT: \neg At(Flat,Axle) \wedge At(Flat,Ground))

Action(PutOn(Spare,Axle)

PRECOND: At(Spare,Groundp) \wedge \neg At(Flat,Axle)

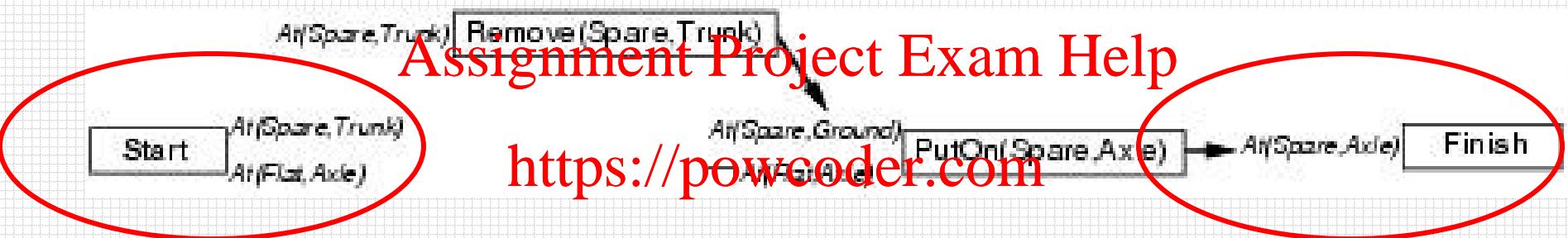
EFFECT: At(Spare,Axle) \wedge \neg Ar(Spare,Ground))

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))*

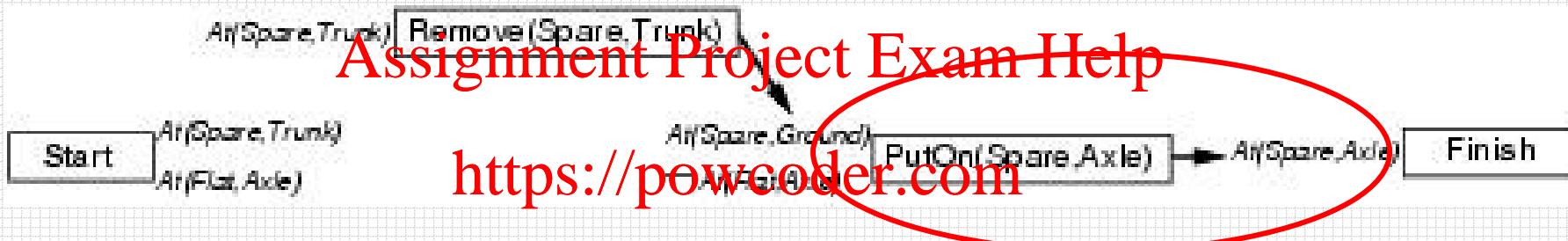
Solving the problem



Add WeChat powcoder

Initial plan: Start with EFFECTS and Finish with PRECOND.

Solving the problem



Add WeChat powcoder

Initial plan: Start with EFFECTS and Finish with PRECOND.

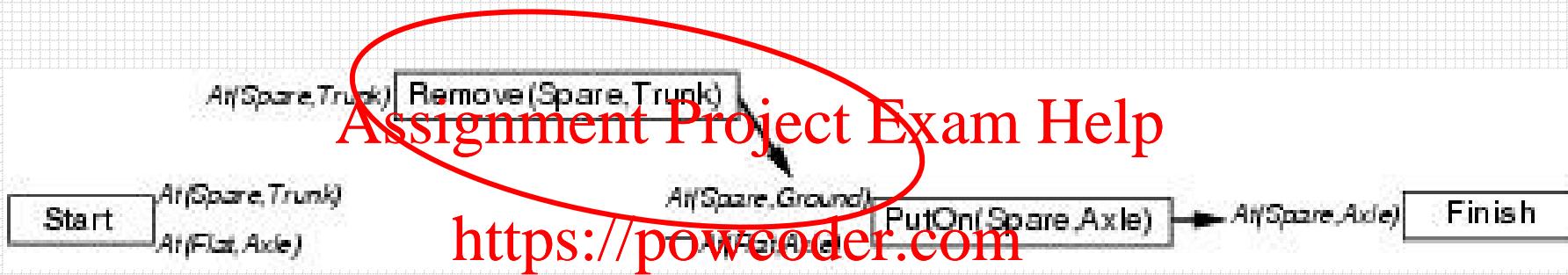
Pick an open precondition: *At(Spare, Axle)*

Only *PutOn(Spare, Axle)* is applicable

Add causal link: *PutOn(Spare, Axle)* $\xrightarrow{\text{At(Spare, Axle)}}$ *Finish*

Add constraint : *PutOn(Spare, Axle)* < *Finish*

Solving the problem



Add WeChat powcoder

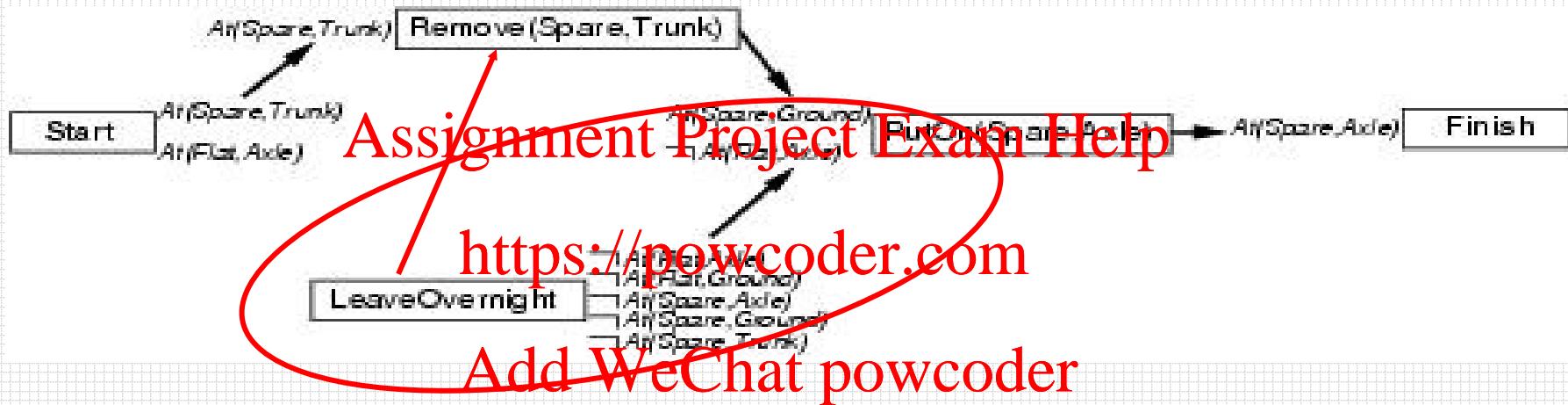
Pick an open precondition: *At(Spare, Ground)*

Only *Remove(Spare, Trunk)* is applicable

Add causal link: *Remove(Spare, Trunk)* $\xrightarrow{\text{At(Spare,Ground)}}$ *PutOn(Spare, Axle)*

Add constraint : *Remove(Spare, Trunk)* < *PutOn(Spare, Axle)*

Solving the problem



Pick an open precondition: $\neg At(Flat, Axle)$

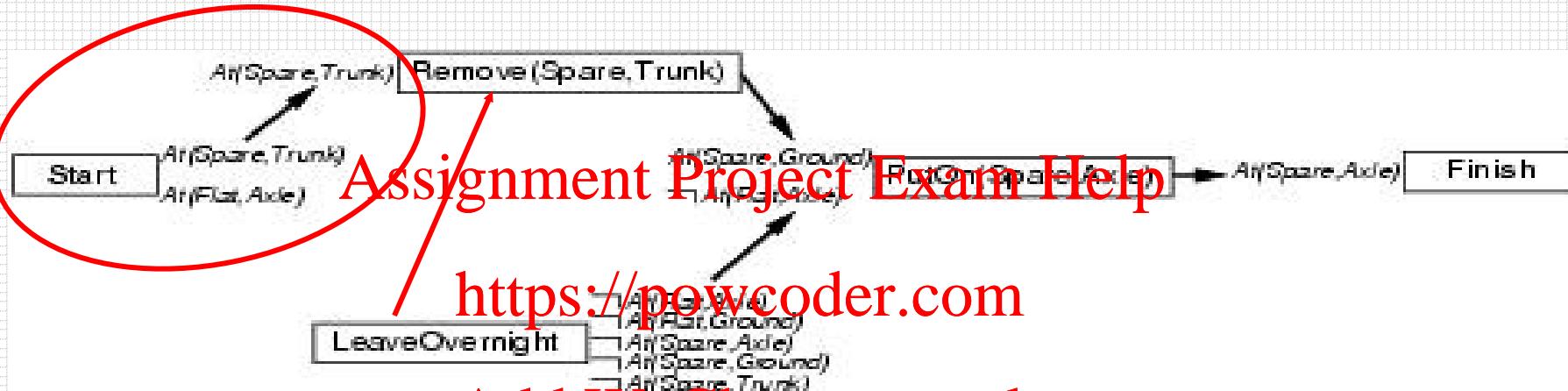
LeaveOverNight is applicable

conflict: *LeaveOverNight* also has the effect $\neg At(Spare, Ground)$

$Remove(Spare, Trunk) \xrightarrow{At(Spare, Ground)} PutOn(Spare, Axle)$

To resolve, add constraint : $LeaveOverNight < Remove(Spare, Trunk)$

Solving the problem



Pick an open precondition: $At(Spare, Trunk)$

Only *Start* is applicable

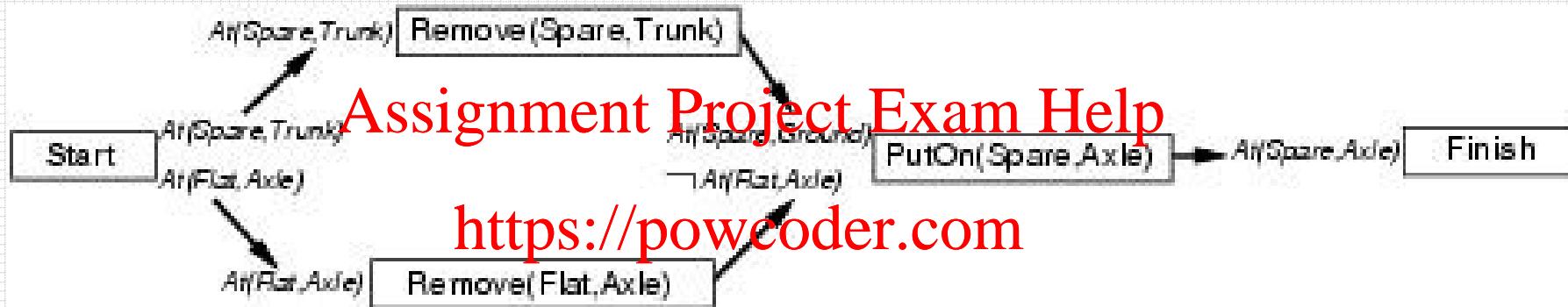
Add causal link: $Start \xrightarrow{At(Spare,Trunk)} Remove(Spare,Trunk)$

Conflict: of causal link with effect $At(Spare,Trunk)$ in *LeaveOverNight*

- No re-ordering solution possible.

backtrack

Solving the problem



Add WeChat powcoder
Remove *LeaveOverNight*, *Remove(Spare, Trunk)* and causal links

Repeat step with *Remove(Spare,Trunk)*

Add also *RemoveFlatAxe* and finish

Some details ...

What happens when a first-order representation that includes variables is used?

- Complicates the process of detecting and resolving conflicts.
- Can be resolved by introducing inequality constraint.

Planning graphs

Used to achieve better heuristic estimates.

- A solution can also directly be extracted using GRAPHPLAN.

Assignment Project Exam Help
https://powcoder.com

Consists of a sequence of levels that correspond to time steps in the plan.

- Level 0 is the initial state.
- Each level consists of a set of literals and a set of actions.
 - *Literals* = all those that *could* be true at that time step, depending upon the actions executed at the preceding time step.
 - *Actions* = all those actions that *could* have their preconditions satisfied at that time step, depending on which of the literals actually hold.

Planning graphs

“Could”?

- Records **Assignment Project Exam Help** only a restricted subset of possible negative interactions among actions.

They work only for propositional problems.
<https://powcoder.com>

Example: Add WeChat powcoder

Init(Have(Cake))

Goal(Have(Cake) \wedge Eaten(Cake))

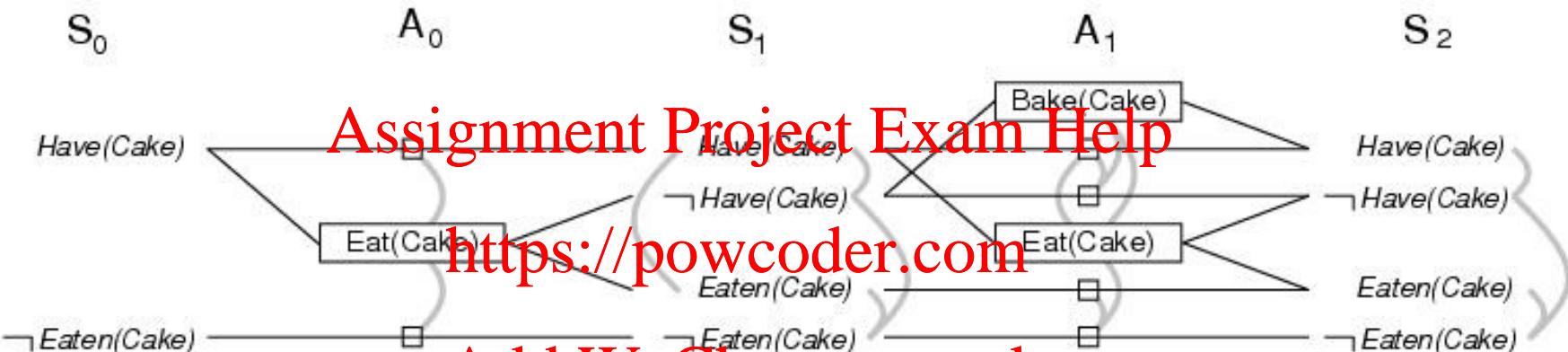
Action(Eat(Cake), PRECOND: Have(Cake))

EFFECT: \neg Have(Cake) \wedge Eaten(Cake))

Action(Bake(Cake), PRECOND: \neg Have(Cake))

EFFECT: Have(Cake))

Cake example



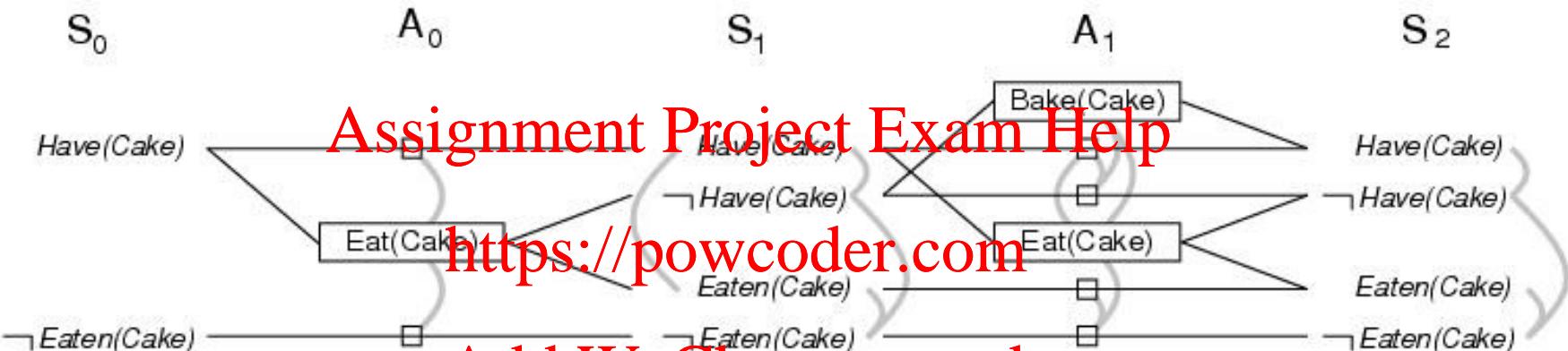
Start at level S_0 and determine action level A_0 and next level S_1 .

- $A_0 >>$ all actions whose preconditions are satisfied in the previous level.
- Connect precond and effect of actions $S_0 \rightarrow S_1$
- Inaction is represented by *persistence actions*.

Level A_0 contains the actions that could occur

- Conflicts between actions are represented by *mutex links*

Cake example



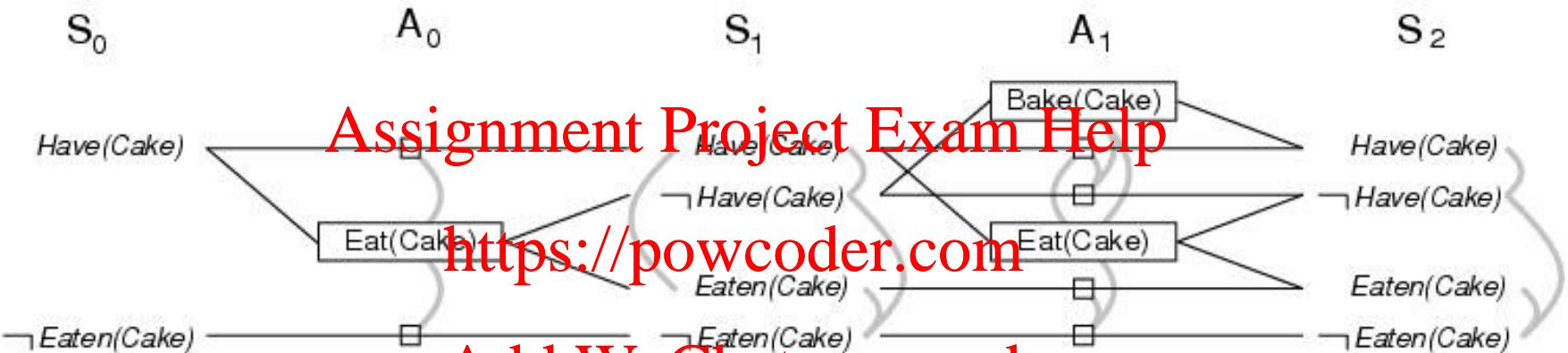
Level S_1 contains all literals that could result from picking any subset of actions in A_0

- Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
- S_1 defines multiple states and the mutex links are the constraints that define this set of states.

Continue until two consecutive levels are identical: leveled off

- Or contain the same amount of literals (explanation follows later)

Cake example



A mutex relation holds between **two actions** when:

- *Inconsistent effects*: one action negates the effect of another.
- *Interference*: one of the effects of one action is the negation of a precondition of the other.
- *Competing needs*: one of the preconditions of one action is mutually exclusive with the precondition of the other.

A mutex relation holds between **two literals** when (*inconsistent support*):

- If one is the negation of the other OR
- if each possible action pair that could achieve the literals is mutex.

PG and heuristic estimation

PG's provide information about the problem

- A literal that does not appear in the final level of the graph cannot be achieved by any plan.
 - Useful for backward search ($\text{cost} = \text{inf}$).
- Level of appearance can be used as cost estimate of achieving any goal literals = *level cost*.
- Small problem: several actions can occur
 - Restrict to one action using serial PG (add mutex links between every pair of actions, except persistence actions).
- Cost of a conjunction of goals? Max-level, sum-level and set-level heuristics.

PG is a relaxed problem.

The GRAPHPLAN Algorithm

How to extract a solution directly from the PG

Assignment Project Exam Help
<https://powcoder.com>

```
function GRAPHPLAN(problem) return solution or failure
  graph  $\leftarrow$  INITIAL-PLANNING-GRAFH(problem)
  goals  $\leftarrow$  GOALS[problem]
  loop do
    if goals all non-mutex in last level of graph then do
      solution  $\leftarrow$  EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
      if solution  $\neq$  failure then return solution
    else if NO-SOLUTION-POSSIBLE(graph) then return failure
    graph  $\leftarrow$  EXPAND-GRAFH(graph, problem)
```

Example: Spare tire problem

Init(At(Flat, Axle) \wedge At(Spare, trunk))

Goal(At(Spare, Axle))

Action(Remove(Spare, Trunk))

PRECOND: *At(Spare, Trunk)*

EFFECT: *\neg At(Spare, Trunk) \wedge At(Spare, Ground)*)

Action(Remove(Flat, Axle))

PRECOND: *At(Flat, Axle)*

EFFECT: *\neg At(Flat, Axle) \wedge At(Flat, Ground)*)

Action(PutOn(Spare, Axle))

PRECOND: *At(Spare, Ground) \wedge ~~At(Flat, Axle)~~*

EFFECT: *At(Spare, Axle) \wedge \neg At(Spare, Ground)*)

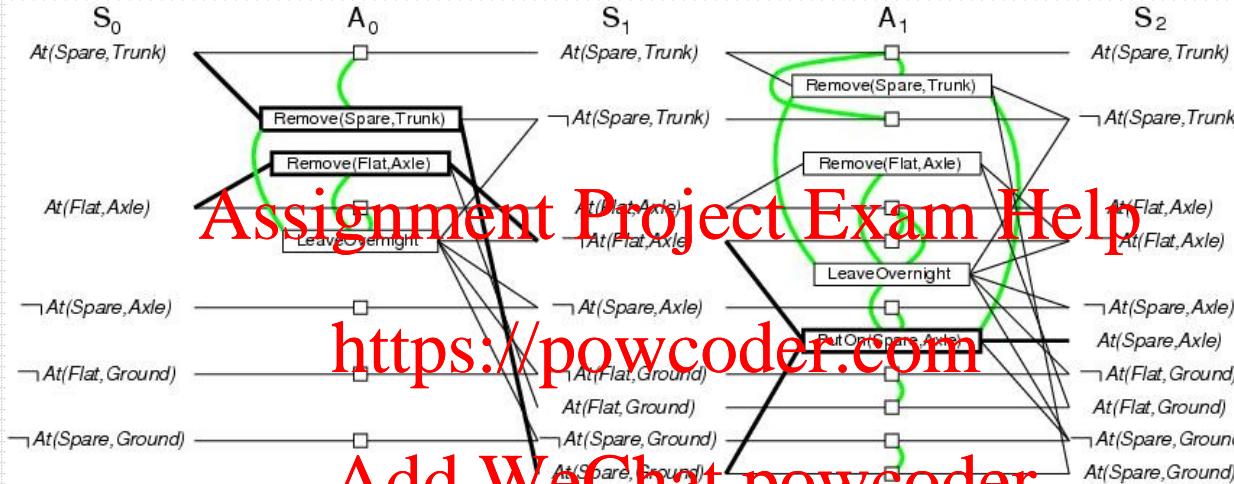
Action(LeaveOvernight)

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))*

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)

GRAPHPLAN example



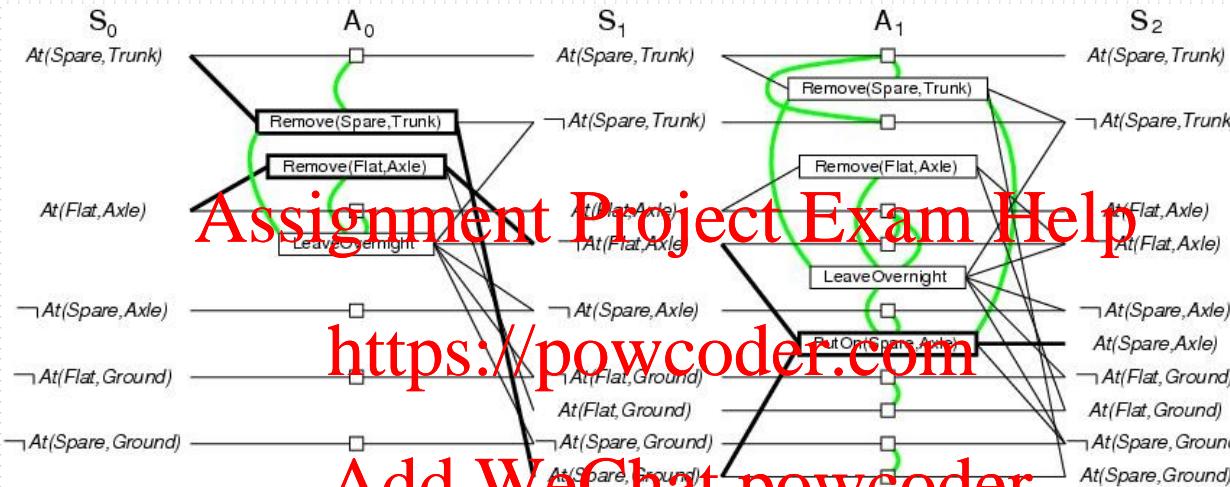
Initially the plan consist of 5 literals from the initial state and the CWA literals (S_0).

Add actions whose preconditions are satisfied by EXPAND-GRAFH (A_0)
Also add persistence actions and mutex relations.

Add the effects at level S_1

Repeat until goal is in level S_i

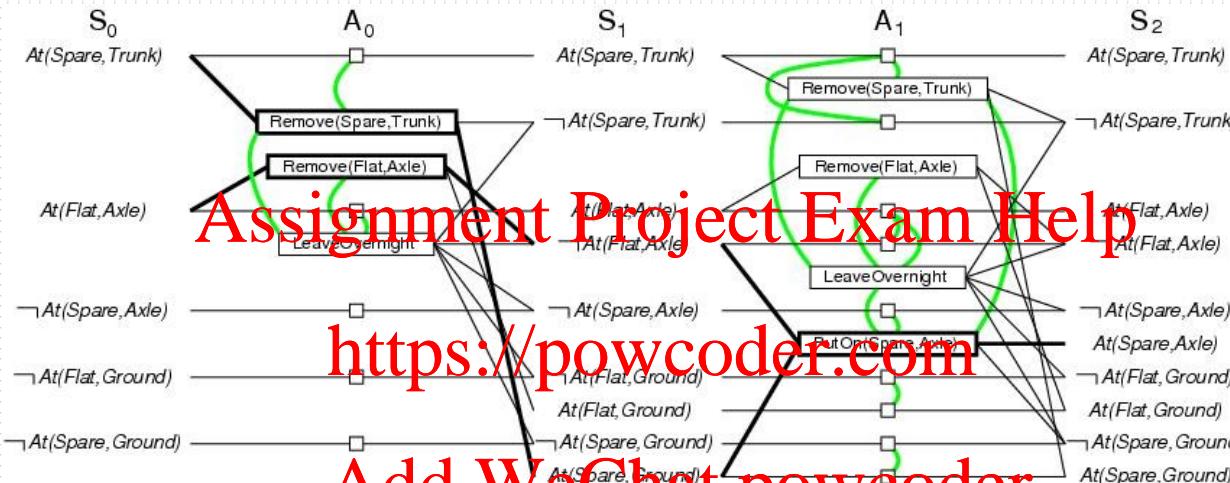
GRAPHPLAN example



EXPAND-GRAF also looks for mutex relations

- Inconsistent effects
 - E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare,Ground) and **not** At(Spare, Ground)
- Interference
 - E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and **not** At(Flat,Axle) as EFFECT
- Competing needs
 - E.g. PutOn(Spare,Axle) and Remove(Flat, Axle) due to At(Flat.Axle) and **not** At(Flat, Axle)
- Inconsistent support
 - E.g. in S_2 , At(Spare,Axle) and At(Flat,Axle)

GRAPHPLAN example



Add WeChat powcoder

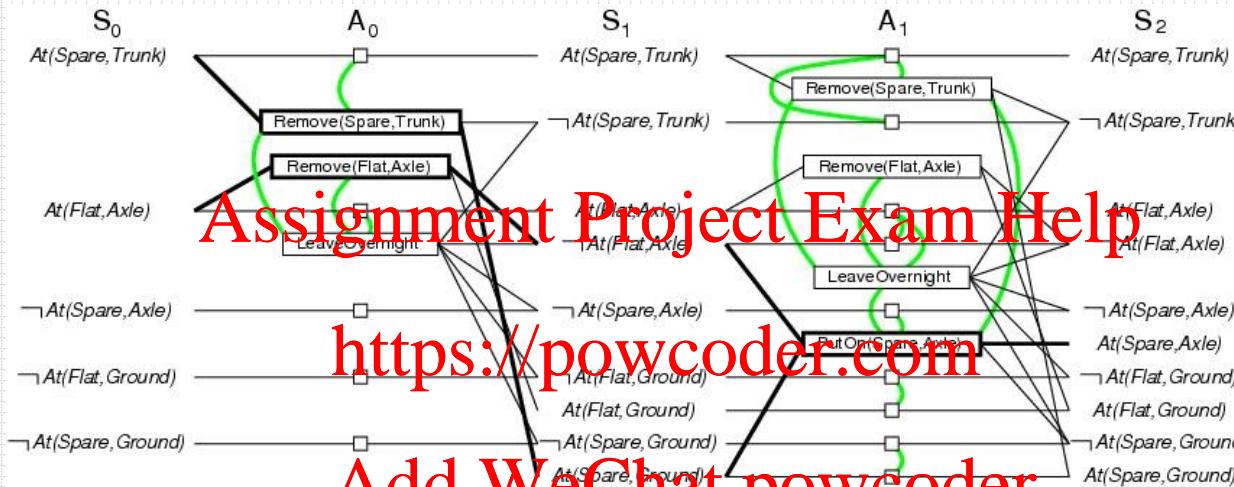
In S_2 , the goal literals exist and are not mutex with any other

- Solution might exist and EXTRACT-SOLUTION will try to find it

EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:

- Initial state = last level of PG and goal goals of planning problem
- Actions = select any set of non-conflicting actions that cover the goals in the state
- Goal = reach level S_0 such that all goals are satisfied
- Cost = 1 for each action.

GRAPHPLAN example



Termination? YES

PG are monotonically increasing or decreasing:

- Literals increase monotonically
- Actions increase monotonically
- Mutexes decrease monotonically

Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off !

Planning with propositional logic

Planning can be done by proving theorem in situation calculus.

Here: test the *satisfiability* of a logical sentence:

Assignment Project Exam Help

initial state \wedge all possible action descriptions \wedge goal
<https://powcoder.com>

Sentence contains propositions for every action occurrence.

- A model will assign true to the actions that are part of the correct plan and false to the others
- An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that the goal is true.
- If the planning is unsolvable the sentence will be unsatisfiable.

SATPLAN algorithm

function SATPLAN(*problem*, T_{max}) **return** *solution* or failure

inputs: *problem*, a planning problem

T_{max} , an upper limit to the plan length

for $T = 0$ **to** T_{max} **do** <https://powcoder.com>

cnf, mapping \leftarrow TRANSLATE-TO-SAT(*problem, T*)
assignment \leftarrow SAT-SOLVER(*cnf*)

if *assignment* is not null **then**

return EXTRACT-SOLUTION(*assignment, mapping*)

return failure

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

Distinct propositions for assertions about each time step

Assignment Project Exam Help

- Superscripts denote the time step
 $At(P1, SFO)^0 \wedge At(P2, JFK)^0$
- No CWA thus specify which propositions are not true
 $\neg At(P1, SFO)^0 \wedge \neg At(P2, JFK)^0$
- Unknown propositions are left unspecified.

The goal is associated with a particular time-step

- But which one?

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

How to determine the time step where
the goal will be reached?

- Start at $T=0$ <https://powcoder.com>
 - Assert $\text{At}(P1, SFO)^0 \wedge \text{At}(P2, JFK)^0$
- Failure .. Try [Add WeChat powcoder](#)
 - Assert $\text{At}(P1, SFO)^1 \wedge \text{At}(P2, JFK)^1$
- ...
- Repeat this until some minimal path length is reached.
- Termination is ensured by T_{max}

$cnf, mapping \leftarrow \text{TRANSLATE-TO_SAT}(problem, T)$

How to encode actions into PL?

- Propositional versions of successor-state axioms
 $At(P1, JFK)^1 \Leftrightarrow$
 $(At(P1, JFK)^0 \rightarrow (Fly(P1, SFO, JFK)^0 \wedge At(P1, SFO)^0)) \vee$
 $(Fly(P1, SFO, JFK)^0 \wedge At(P1, SFO)^0)$
- Such an axiom is required for each plane, airport and time step
- If more airports add another way to travel than additional disjuncts are required

Once all these axioms are in place, the satisfiability algorithm can start to find a plan.

assignment \leftarrow SAT-SOLVER(*cnf*)

Multiple models can be found

They are NOT satisfactory: (for $T=1$)

$$Fly(P1, SFO, JFK)^0 \wedge Fly(P1, JFK, SFO)^0 \wedge Fly(P2, JFK, SFO)^0$$

The second action is infeasible

Yet the plan IS a model of the sentence

Add WeChat powcoder

initial state \wedge *all possible action descriptions* \wedge *goal*¹

Avoiding illegal actions: pre-condition axioms

$$Fly(P1, SFO, JFK)^0 \Rightarrow At(P1, JFK)$$

Exactly one model now satisfies all the axioms where the goal is achieved at $T=1$.

assignment \leftarrow SAT-SOLVER(*cnf*)

A plane can fly at two destinations at once

They are NOT satisfactory: (for T=1)

$Fly(P1, SFO, JFK)^0 \wedge Fly(P2, JFK, SFO)^0 \wedge Fly(P2, JFK, LAX)^0$

The second action is infeasible

Yet the plan allows spurious relations

<https://powcoder.com>

Avoid spurious solutions: action-exclusion axioms

$\neg(Fly(P2, JFK, SFO)^0 \wedge Fly(P2, JFK, LAX)^0)$

Prevents simultaneous actions

Loss of flexibility since plan becomes totally ordered : no actions are allowed to occur at the same time.

- Restrict exclusion to preconditions

Analysis of planning approach

Planning is an area of great interest
within **AI** Assignment Project Exam Help

- Search for solution <https://powcoder.com>
- Constructively prove existence of solution

Add WeChat powcoder

Biggest problem is the combinatorial explosion in states.

Efficient methods are under research

- E.g. divide-and-conquer