# Spezielle Themen der Künstlichen Intelligenz

5. Termin: Planning

Dr. Stefan Kopp
Center of Excellence „Cognitive Interaction Technology“
AG Sociable Agents



## Planning

- The Planning problem
- Planning with State-space search
- Partial-order planning
- Planning graphs



## What is Planning?

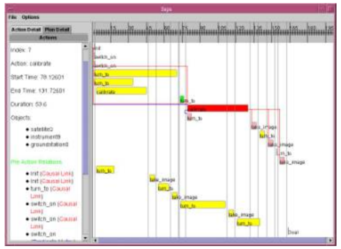
Generate sequences of actions to perform tasks and achieve objectives

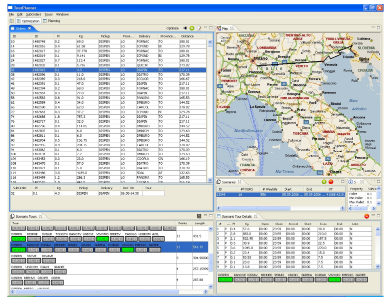
Search for solution over abstract space of action sequences

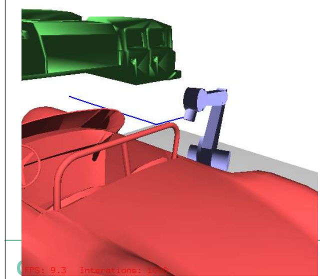
Used to assists humans in many practical applications

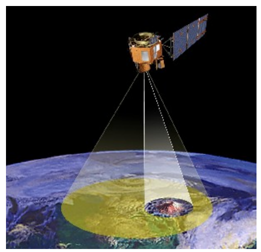
- design and manufacturing
- military operations
- games
- space exploration
- scheduling
- ...

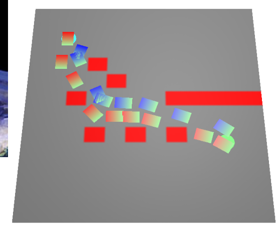












## Problem: difficulty of the real world

Assume a problem-solving agent using *some* search method, needs to build on answers to...

- 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?
  - TSP: O(n!) vs. O((n/k)!*k), if k equal subparts
  - most real-world problems are *nearly* decomposable



## Problem: language of planning

What is a good language to describe a planning problem?

- expressive enough to describe a wide variety of problems, with numerous states and how those change upon actions
- restrictive enough to allow algorithms to operate on it
- algorithms should be able to exploit logical structure of the problem

STRIPS and ADL

- STRIPS = Stanford Research Institute Problem Solver
- ADL = Action Description Language

PDDL (Planning domain description language)

- standardize languages to make the international Planning Competitions possible (ICP/ICAPS, 1998-)
- contains STRIPS, ADL and more



## Language features

Representation of states

- Decompose the world in logical conditions and represent a state as a conjunction of positive literals
  - Propositional literals: *Poor* ∧ *Unknown*
  - First order (FO), grounded and function-free: *At(Plane1, Melbourne)* ∧ *At(Plane2, Sydney)*
- Closed world assumption: any conditions not mentioned in a state are assumed to be false

Representation of goals

- Partially specified state, represented as a conjunction of positive ground literals
- A goal is satisfied by state *s*, if *s* contains (at least) all the literals in the goal



## Language features

Representations of actions

- Action = PRECOND + EFFECT, e.g. flying a plane:

  *Action(Fly(p, from, to),*
  *PRECOND: At(p,from)* ∧ *Plane(p)* ∧ *Airport(from)* ∧ *Airport(to)*
  *EFFECT: ¬AT(p,from)* ∧ *At(p,to))*

= action schema for which *p, from, to* are instantiated

- *Action name* and *parameter list* of variables
- *Precondition*: conjunction of function-free literals
- *Effect*: conjunction of function-free literals; literal *P* is asserted to be true in the resulting state, not P is false

# Classical problems of symblic KR

Frame problem

- specifying only what is changed by actions does not allow to conclude, in logic, that other conditions are *not changed*
- can be solved by adding so-called *frame axioms*
  - specify that all conditions not affected by the action are not changed
- different solutions in different formalisms

Qualification problem

- impossibility of listing all the *preconditions* required for an action to have its intended effect

Ramification problem

- how to represent what happens implicitly due to an action?



# Language semantics

How do actions affect states?

- An action is applicable in any state that satisfies preconditions
- FO action schema applicability involves unification, i.e. a substitution $\theta$ for the variables in the PRECOND
  - State:
    *At(P1,JFK) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)*
  - Satisfies precondition of *Action(Fly(p, from, to)*:
    *At(p,from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)*
    with $\theta$ ={*p/P1,from/JFK,to/SFO*}
  - Thus the action is applicable



# Language semantics

How do actions affect states?

The resulting state *s'* of action *a* in state *s* is the same as *s*, except

- any positive literal *P* in the effect of *a* is added to *s'*
- any negative literal ¬*P* in the effect of *a* is removed from *s'*
- Example: After *Fly(P1, JFK, SFO)* current state becomes
  *At(P1,SFO) ∧ At(P2,SFO) ∧ Plane(P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO)*

STRIPS assumption to avoid the representational frame problem:
Implicit assumption that every literal in *s* that is *not* in the effects remains unchanged



# Expressiveness and extensions

STRIPS is simplified

- important limit: function-free literals
- allows for turning action schemas into propositional action representations without variables (by universal insertion)
- function symbols lead to *infinitely* many states and actions

*Extension:* Action Description language (ADL)

- positive and negative literals
- quantified variables and conj.+discj. in goals
- conditional effects „when P: E"
- equality predicate, variables with types

*Action(Fly(p:Plane, from: Airport, to: Airport),*
*PRECOND: At(p,from) ∧ (from ≠ to)*
*EFFECT: ¬At(p,from) ∧ At(p,to))*

# Example: air cargo transport

*Init(At(C1, SFO) ∧ At(C2,JFK) ∧ At(P1,SFO) ∧ At(P2,JFK) ∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane (P1) ∧ Plane(P2) ∧ Airport(JFK) ∧ Airport(SFO))*

*Goal(At(C1,JFK) ∧ At(C2,SFO))*

*Action(Load(c,p,a)*

PRECOND: *At(c,a) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: *¬At(c,a) ∧In(c,p))*

*Action(Unload(c,p,a)*

PRECOND: *In(c,p) ∧At(p,a) ∧Cargo(c) ∧Plane(p) ∧Airport(a)*

EFFECT: *At(c,a) ∧ ¬In(c,p))*

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

PRECOND: *At(p,from) ∧Plane(p) ∧Airport(from) ∧Airport(to)*

EFFECT: *¬ At(p,from) ∧ At(p,to))*

***Solution:***

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



# Example: Spare tire problem

*Init(At(Flat, Axle) ∧ At(Spare,trunk))*

*Goal(At(Spare,Axle))*

*Action(Remove(Spare,Trunk)*

PRECOND: *At(Spare,Trunk)*

EFFECT: *¬At(Spare,Trunk) ∧ At(Spare,Ground))*

*Action(Remove(Flat,Axle)*

PRECOND: *At(Flat,Axle)*

EFFECT: *¬At(Flat,Axle) ∧ At(Flat,Ground))*

*Action(PutOn(Spare,Axle)*

PRECOND: *At(Spare,Groundp) ∧¬At(Flat,Axle)*

EFFECT: *At(Spare,Axle) ∧ ¬Ar(Spare,Ground))*

*Action(LeaveOvernight*

PRECOND:

EFFECT: *¬ At(Spare,Ground) ∧ ¬ At(Spare,Axle) ∧ ¬ At(Spare,trunk) ∧ ¬ At(Flat,Ground) ∧ ¬ At(Flat,Axle) )*

This example goes beyond STRIPS: negative literal in pre-condition



# Planning with state-space search

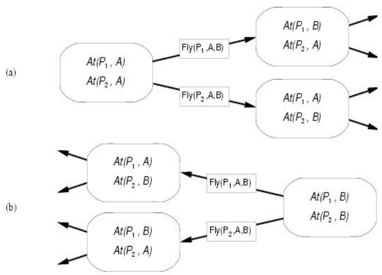
Both forward and backward search possible, because preconds and effects are given

## Progression planners

- Forward state-space search, start from initial state
- Follow effects of possible actions in a given state

## Regression planners

- Backward state-state search, start from goal state
- Follow preconditions that must have been true in the previous state





# 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 = constant, each action costs +1

State space is finite

- any graph search that is complete is a complete planning algorithm
- too inefficient to be practical
  - irrelevant actions are considered
  - good heuristic required for efficient search

# Regression algorithm

How to determine predecessors states from which an action leads to goal?

- Goal state = $At(C1, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
- Relevant action for first conjunct: $Unload(C1,p,B)$
- Works only if pre-conditions are satisfied -> add conj.
- Previous state= $In(C1, p) \wedge At(p, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
  - subgoal $At(C1,B)$ should not be present in this state anymore

Important that actions do not undo desired literal (consistent)

Can use any standard search algorithm, but needs a good admissible heuristics

Main advantage: only relevant actions are considered

- Often much lower branching factor than forward search
- In FO case, satisfaction might require a substitution



# Partial-order planning (POP)

Progression and regression planning are totally ordered plan searches

- yield strictly linear sequences of actions
- cannot take advantage of problem decomposition!
- decisions must be made on how to sequence actions on all the subproblems

Better: Least commitment strategy

- delay choice during search until really necessary
- keep flexibility in order of actions, and during plan construction



# Very simple example: Put on shoes

*Goal(RightShoeOn ∧ LeftShoeOn)*
*Init()*
*Action(RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)*
*Action(RightSock, EFFECT: RightSockOn)*
*Action(LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)*
*Action(LeftSock, EFFECT: LeftSockOn)*

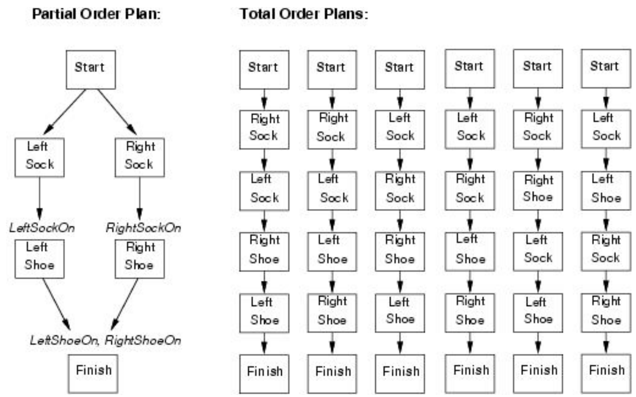
Planner

- plan two independent action sequences
  (1) leftsock, leftshoe
  (2) rightsock, rightshoe
- no need to commit itself to an order



# Partial-order planning

Any planning algorithm that can place two actions into a plan without saying which comes first is a POP



# POP as a search problem

Search states are (mostly unfinished) plans

- The empty plan contains only start and finish actions

Each plan has 4 components:

- set of actions that make up the steps of the plan
- set of ordering constraints: A<B (A before B)
  - cycles (A<B, B<A) represent contradictions!
- set of causal links
  - „A achieves p for B" $A \xrightarrow{p} B$
  - Plan not extended by adding action C if its effect is ¬p and if it could come after A and before B
- set of open preconditions
  - Not achieved by some action in the plan



# 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

A consistent plan with no open preconditions is a solution

- every linearization is a total solution

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

- benefit in non-deterministic, non-cooperative environments



# Solving POP search problems

Assume propositional planning problems

- The initial plan contains **Start** and **Finish**, the ordering constraint **Start < Finish**, no causal links, all the preconditions in **Finish** are open
- Successor function:
  - picks one open precondition $p$ on an action $B$
  - generates a successor plan for every possible consistent way of choosing action $A$ that achieves $p$
    - causal link *A-p->B* and ordering constraint *A<B* added to the plan; if A new, also add constraints *start<A* and *A<B*
    - resolve conflicts between link(s) and action(s) by constraining actions to occur outside protected intervals
- Test goal: check whether no open preconditions left
- Search refines the plan gradually, from incomplete/vague to complete/correct plans

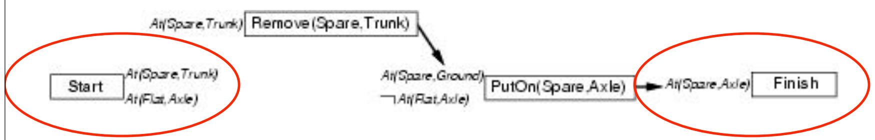


# Example: Mounting spare tire

*Init(At(Flat, Axle) ∧ At(Spare,trunk))*
*Goal(At(Spare,Axle))*
*Action(Remove(Spare,Trunk)*
PRECOND: *At(Spare,Trunk)*
EFFECT: *¬At(Spare,Trunk) ∧ At(Spare,Ground))*
*Action(Remove(Flat,Axle)*
PRECOND: *At(Flat,Axle)*
EFFECT: *¬At(Flat,Axle) ∧ At(Flat,Ground))*
*Action(PutOn(Spare,Axle)*
PRECOND: *At(Spare,Ground) ∧¬At(Flat,Axle)*
EFFECT: *At(Spare,Axle) ∧ ¬Ar(Spare,Ground))*
*Action(LeaveOvernight*
PRECOND:
EFFECT: *¬ At(Spare,Ground) ∧ ¬ At(Spare,Axle) ∧ ¬ At(Spare,trunk) ∧ ¬ At(Flat,Ground) ∧ ¬ At(Flat,Axle) )*

(*LeaveOvernight* ➔ bad neighborhood, all tires will disappear)

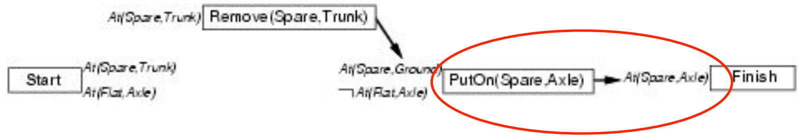
# Solving the problem




Intial plan: Start with EFFECTS and Finish with PRECOND.



# Solving the problem




Pick an open precondition: *At(Spare, Axle)*
Only *PutOn(Spare, Axle)* is applicable
Add causal link: $PutOn(Spare,Axle) \xrightarrow{At(Spare,Axle)} Finish$
Add constraint : *PutOn(Spare, Axle) < Finish*

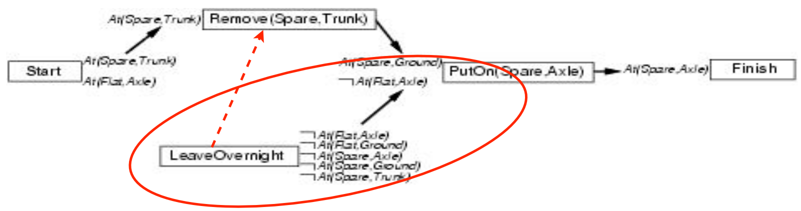


# Solving the problem


Pick an open precondition: *At(Spare, Ground)*
Only *Remove(Spare, Trunk)* is applicable
Add causal link: $Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$
Add constraint : *Remove(Spare, Trunk) < PutOn(Spare,Axle)*

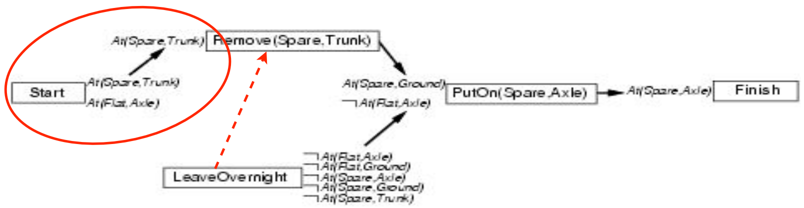


# Solving the problem




Pick other open precondition: *not At(Flat, Axle)*
*LeaveOverNight* is applicable, add causal link to *PutOn(Spare,Axle)*
conflict with $Remove(Spare,Trunk) \xrightarrow{At(Spare,Ground)} PutOn(Spare,Axle)$
To resolve, add constraint: *LeaveOverNight < Remove(Spare,Trunk)*

# Solving the problem




Pick open precondition: *At(Spare, Trunk)*

Only *Start* is applicable

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

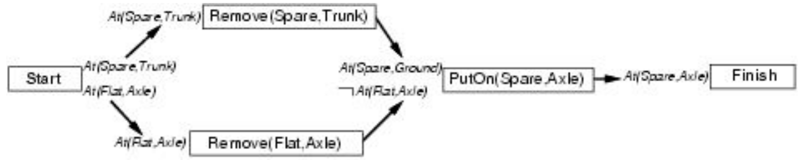
Conflict with effect *At(Spare,Trunk)* of *LeaveOverNight*

- **No re-ordering solution possible**

Backtrack (chronological)



# Solving the problem




Remove *Start, LeaveOverNight* and causal links

Choose *Remove(Flat,Axle)* for precond *not At(Flat, Axle)*, add link

Choose *Start* for precondition *At(Spare, Trunk)*, add link

Finish because *Start* meets also precond *At(Flat,Axle)*, no conflicts



# Planning graphs (PG)

Data structure to achieve better heuristic estimates

- A solution can also be directly extracted using GRAPHPLAN

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 actions at preceding step
  - *Actions* = all those that *could* have their preconds satisfied at that time step, depending on literals that actually hold



# Planning graphs (PG)

"Could"?

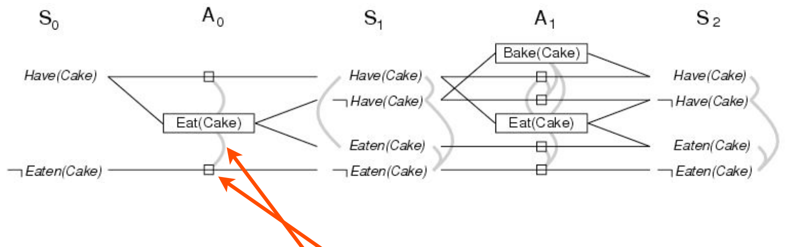
- Graph records only a restricted subset of possible negative interactions among actions; optimistic about number of steps for a literal to become true

Works only for propositional problems, no variables

- Example problem

*Init(Have(Cake))*
*Goal(Have(Cake) ∧ Eaten(Cake))*
*Action(Eat(Cake),*
*PRECOND: Have(Cake), EFFECT: ¬Have(Cake) ∧ Eaten(Cake))*
*Action(Bake(Cake),*
*PRECOND: ¬ Have(Cake), EFFECT: Have(Cake))*

# Cake example



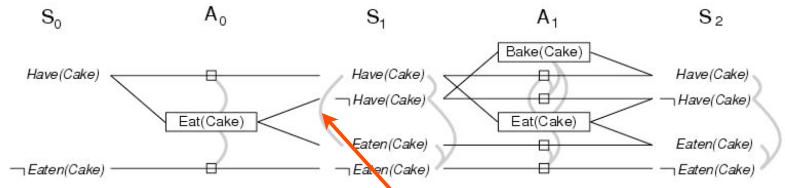

Start at state *S0*, determine action level *A0* and level *S1*

- A0: all actions that could occur in S0, i.e. whose preconditions are satisfied in S0
- Links connect A0 actions with preconds in S0 and effects in S1
- For every literal L, „in action" represented by **persistence actions** with precond and effect L
- Conflicts between actions represented by **mutex links**

***define what remains true because no action alters it; cf. frame problem in situation calculus***



# Cake example




*S1* contains literals that result from any subset of actions in *A0*

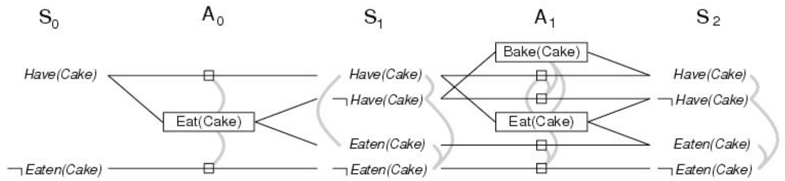
- conflicts between literals also represented by **mutex links**, e.g. either *Have(Cake)* or *Eaten(Cake)* true, depending on choice of actions on *A0*
- *S1* defines multiple states at the same time, mutex links define this set of states

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*** with

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

A **mutex relation** holds between ***two literals*** when

- one is the negation of the other, or
- each possible action pair that could achieve them is mutex'ed (inconsistent support)



# PG and heuristic estimation

PG's provide valuable information about the problem

- literal that does not appear in the final level of the graph *cannot* be achieved by any plan
- If a literal appears, there is a plan that *possibly* achieves it (no binary mutex'es, but could still be from >2 actions)

Can use PG to estimate cost heuristic

- level of appearance $S_i$ as cost estimate of achieving a goal literal = level cost of the goal
- But several actions can occur at each level
  - compute heuristic with serial planning graph: restrict to one action at a time step
- appraoches to estimates the costs of a conjuction of goals:
  - max-level (maximum level cost of any of the goals; admissible)
  - level sum (sum of the level costs of goals; inadmissible but works quite well)
  - set-level (level at which all literals in the conjunctive goal appear in the PG as pairwise not mutually exclusive)

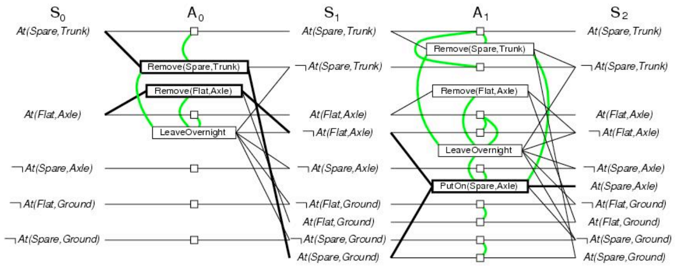
# The GRAPHPLAN Algorithm

Now, how to extract a solution directly from the PG?

**function** GRAPHPLAN(*problem*) **return** *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*)



# GRAPHPLAN example




Initially the plan consist of 5 literals from the initial state (➔ *S0*)

- no goal literal *At(Spare,Axle)*

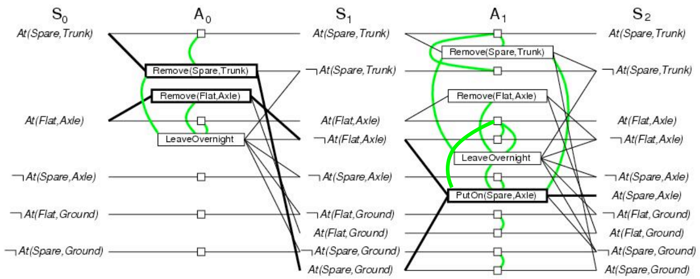
Add actions with precond satisfied by EXPAND-GRAPH (➔ *A0*)

- Also add persistence actions and mutex relations
- Add the effects at level *S1*

Repeat until goal is in level *Si*, then EXTRACT-SOLUTION (➔ S2)



# GRAPHPLAN example

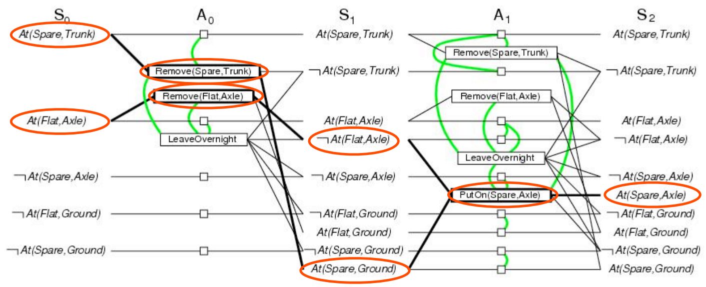



EXPAND-GRAPH also looks for mutex relations

- Inconsistent effects: E.g. *Remove(Spare,Trunk)* and *LeaveOverNight*
- Interference: E.g. *Remove(Flat,Axle)* and *LeaveOverNight*
- Competing needs: E.g. *PutOn(Spare,Axle)* and *Remove(Flat,Axle)*
- Inconsistent support: E.g. in S2, *At(Spare,Axle)* and *At(Flat,Axle)*



# GRAPHPLAN example




In *S2*, the goal literal exists and is not mutex with any other

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

Can use Boolean CSP or a search process

- Initial state = last level of PG and goals of planning problem
- Actions = set of non-conflicting actions in *A(i-1)* that cover goals in state; new state at level *S(i-1)* with goals = preconds of selected actions
- Goal = reach state at level *S0* such that all goals are satisfied

# Another way: planning with propositional logic

Planning can also be done by proving theorem in situation calculus
Here: test the *satisfiability* of a logical sentence:

$$\textit{initial state} \wedge \textit{all possible action descriptions} \wedge \textit{goal}$$

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 planning is unsolvable the sentence will be unsatisfiable

See Russel & Norvig, pp. 402-406



# Next week

So far: fully observable, static and deterministic domains.

- Agent can plan first and then execute plan with eyes closed

Uncertain environments

- partially observable
- nondeterministic
- incorrect information (differences between world and model)