Reasoning and Decision-Making under Uncertainty

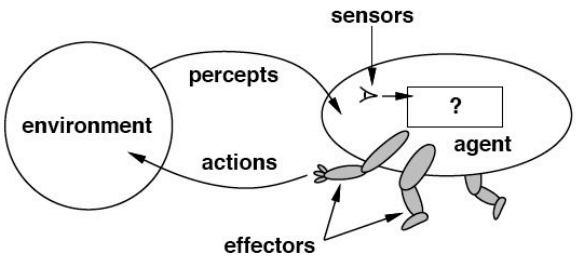
2. Session: Robust planning

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

CITEC

Sociable Agents

Sources of uncertainty in A.I. systems



- incomplete knowledge about the world
- idealized representation of this knowledge
- conflicting information
- local and detached inferences, deductive & abductive reasoning
- indeterminacy of the world (dynamics, low predictability)

Given all this, how still to decide what to do?

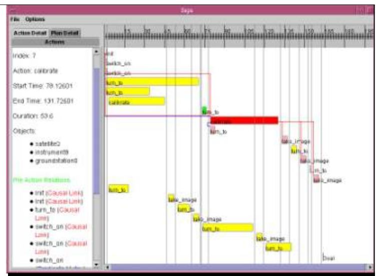
What is Planning?

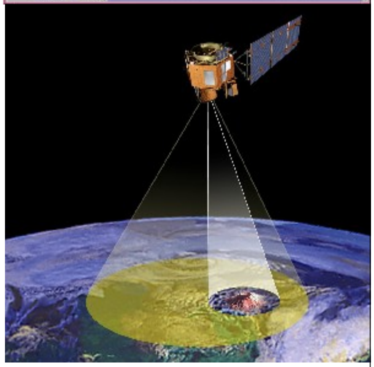
Decide sequences of actions to perform tasks and to achieve objectives

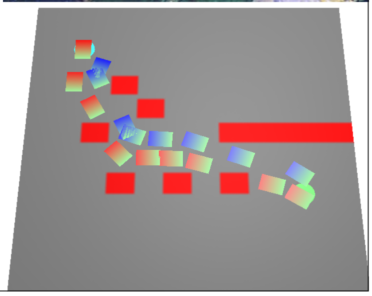
Search for solution over abstract space of possible action sequences

Used in many practical applications

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







Planning as formal search problem

To plan, an agent needs to build on assumptions about

- which actions are relevant
  - exhaustive search vs. backward search
- when actions are possible and what effects they bring about
  - pre-conditions and post-conditions
- what a good heuristic function is
  - estimate of „cost“ of an action sequence
  - problem-dependent vs. -independent
- how the problem may be decomposed
  - <u>*Example:*</u> Traveling-Salesman in O(n!) vs. O((n/k)!*k), if k equal subparts

Planning as formal search problem

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, being able to exploit logical structure of the problem

Standard logics-based languages

- STRIPS = Stanford Research Institute Problem Solver
- ADL = Action Description Language
- PDDL = Planning domain description language
  - standardized language for int. Planning Competitions (ICP/ICAPS, 1998-)
  - encompasses STRIPS, ADL, and many more

Planning languages

Representation of states

- Decompose the world into logical conditions, represent a state as a conjunction of positive literals
  - Propositional literals: $Poor \wedge Unknown$
  - First order (FO), grounded, function-free: $At(Plane1, Melbourne) \wedge At(Plane2, Sydney)$
- 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

Planning languages

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

Expressiveness and extensions

STRIPS is simplified to be efficient

- important limit: function-free literals
- action schemas as 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, conjunction and disjunction 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: spare tire problem in ADL

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)

Three classical problems

Frame problem

- specifying only the changes through actions, does not allow to formally conclude that other conditions have 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

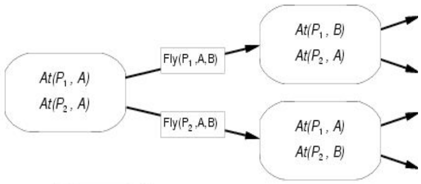
Qualification problem

- impossibility of listing all *preconditions* required for an action to have its intended effect, i.e., to check everything that can *prevent* an action from being successful

Ramification problem

- impossibility of listing all direct and indirect effects of an action

Planning as state-space search



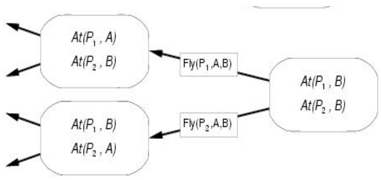
Progression planner:

- 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

Planning as state-space search



Regression planner

- Starting from goal state, follow preconditions that must have been true in previous state
- Works only if pre-conditions are satisfied -> add conj.
- Actions must not undo desired literals (consistent)
- *Example*:
  - Goal state = $At(C1, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
  - Relevant action for first conjunct: $Unload(C1,p,B)$
  - Previous state= $In(C1, p) \wedge At(p, B) \wedge At(C2, B) \wedge \ldots \wedge At(C20, B)$
    - subgoal $At(C1,B)$ not present in this state anymore

Main advantage: only relevant actions are considered

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

Partial-order planning (POP)

Progression and regression planning are totally ordered plan searches

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

Better: Least commitment strategy

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

POP as a plan-based search

Search states = plans (mostly unfinished)

- 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 within 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

- advantage in non-deterministic, non-cooperative environments

Solving POP search problems

Search refines the plan gradually, from incomplete/vague to complete/correct plans:

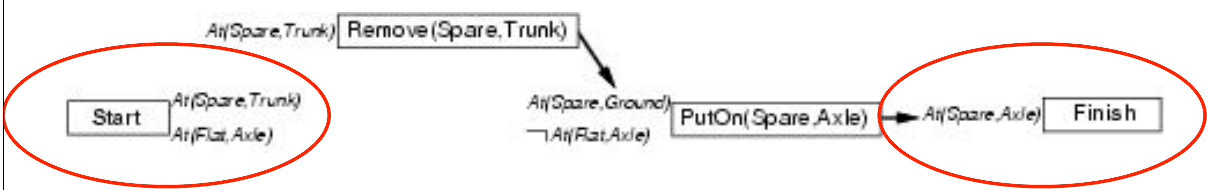
- The initial plan: {**Start, Finish**}, **Start < Finish**, no causal links, all preconditions in **Finish** are open
- Successor function:
  - picks one open precondition p on an action B
  - generates a successor plan for every possible consistent way of choosing action A that achieves p
    - causal 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

Example: Mounting spare tire in POP

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))*

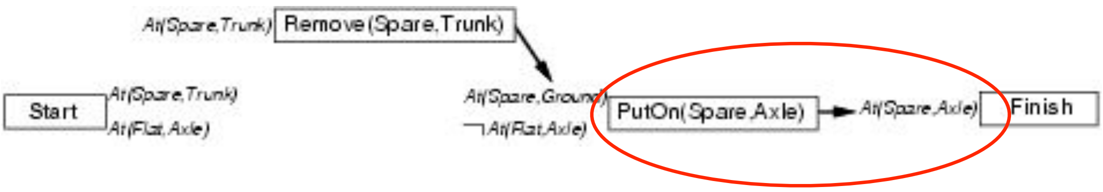
(➔ bad neighborhood, all tires will disappear)

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

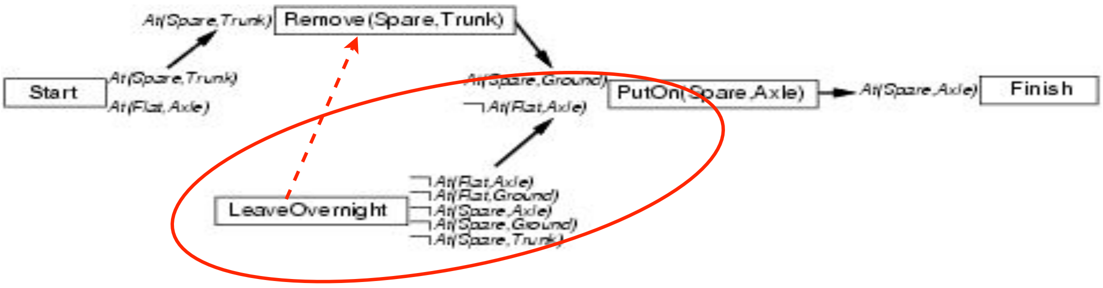
Pick an open precondition: *At(Spare, Ground)*

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



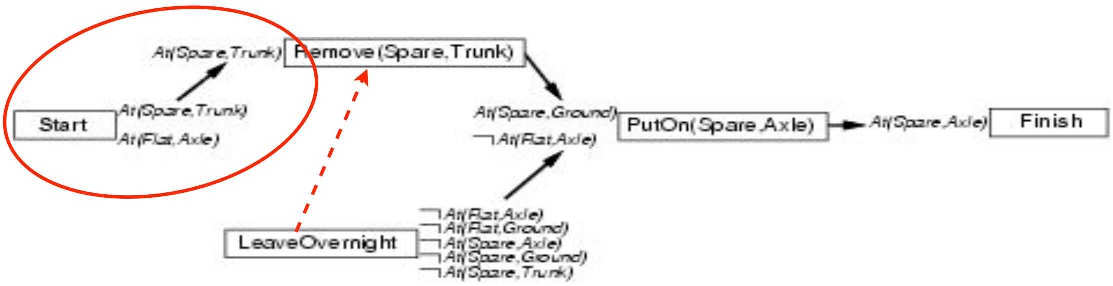
Pick other open precondition: *not At(Flat, Axle)*

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)*

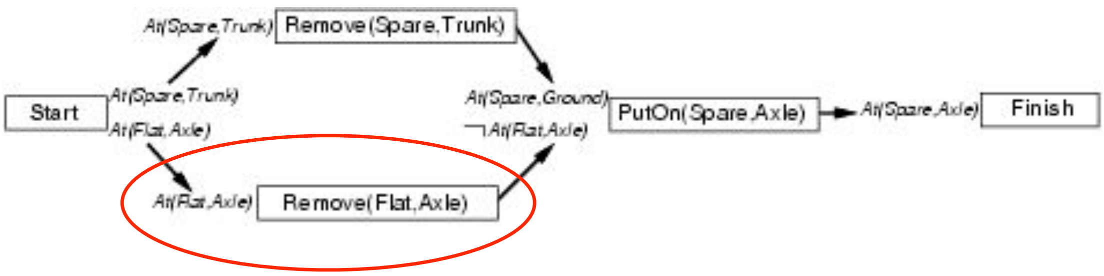
Only *Start* is applicable

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

Conflict with effect *not At(Spare,Trunk)* of *LeaveOverNight*

- No re-ordering solution possible
- Backtrack (chronological)

Solving the problem



Remove *Start, LeaveOverNight* and causal links

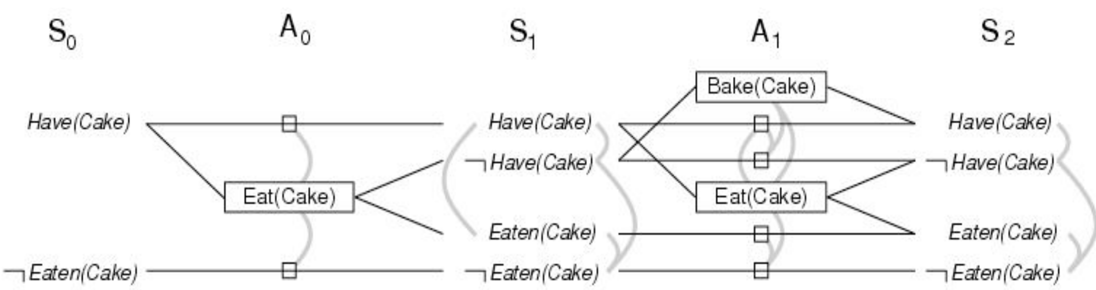
Choose *Remove(Flat,Axle)* for precond *not At(Flat, Axle),* add link

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 with all possible worlds and plans (sets of literals, actions, mutex links), used to achieve better heuristic estimates



Solution plan can be directly extracted from PG

- *Standard algorithm:* GRAPHPLAN (cf. Russel & Norvig)

Summary -- so far

Formulating planning problems: STRIPS, ADL

Planning as search over world states

- progression: start ➔ goal, follow action effects
- regression: goal ➔ start, follow action preconditions

Partial order planning as search over plans

- refine partial plans with least commitment
- order constraints, causal links

Planning graphs

- single graph structure with *all possible worlds and plans*
- used to extract solution, estimate cost heuristics

Time, schedules and resources

Until know: what actions to select?

Real-world:

- + actions have a beginning and an end time
- + actions have a certain duration
- + actions consume certain resources

Job-shop scheduling problem

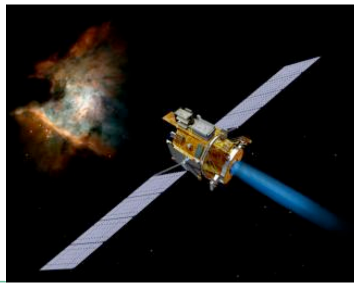
- complete a set of jobs, each consisting of sequence of actions
- each action has duration and requires resources
- determine a schedule that minimizes total time to complete all jobs (respecting resource constraints)

Example: A.I. in space

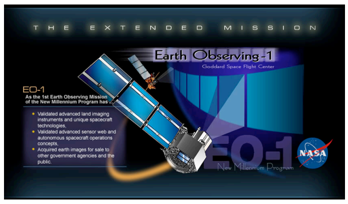
NASA's DEEP SPACE 1 - REMOTE AGENT

- Known as Remote Agent, the software operated NASA's Deep Space 1 spacecraft during two experiments that started on Monday, May 17, 1999. For two days Remote Agent ran on the on-board computer of Deep Space 1.

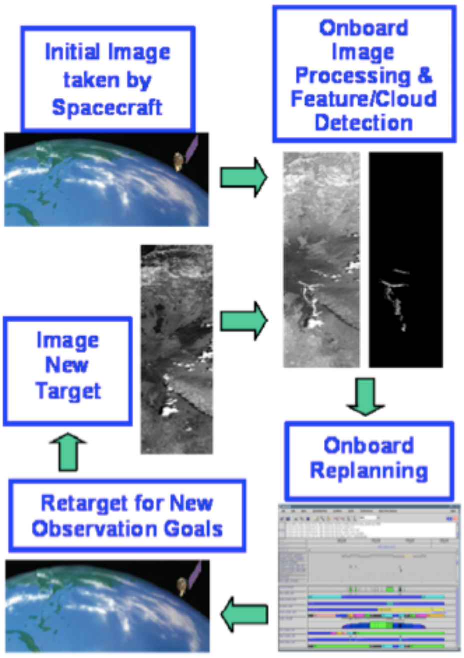
"It's one small step in the history of space flight. But it was one giant leap for computer-kind, with a state of the art artificial intelligence system being given primary command of a spacecraft."



Example: A.I. in space



http://eo1.gsfc.nasa.gov/



*„The **Earth Observing One** spacecraft, launched Nov. 2000, has been under the control of AI software for several years - experimentally since 2003 and since November 2004 as the primary operations system.*

This software includes: model-based planning and scheduling, procedural execution, and event detection software learned by support vector machine (SVM) techniques. It has enabled a 100x increase in the mission science return per data downlinked and a >$1M/year reduction in operations costs.“

How to make plans under uncertainty?

Sensorless planning (conformant planning)

- Find plan that achieves goal *in all possible circumstances*, i.e. in all possible world states

Conditional planning (contingent planning)

- Construct *conditional* plan with different branches for possible contingencies

Execution monitoring and replanning

- While constructing and executing a plan, judge whether the plan requires revision and re-plan if needed

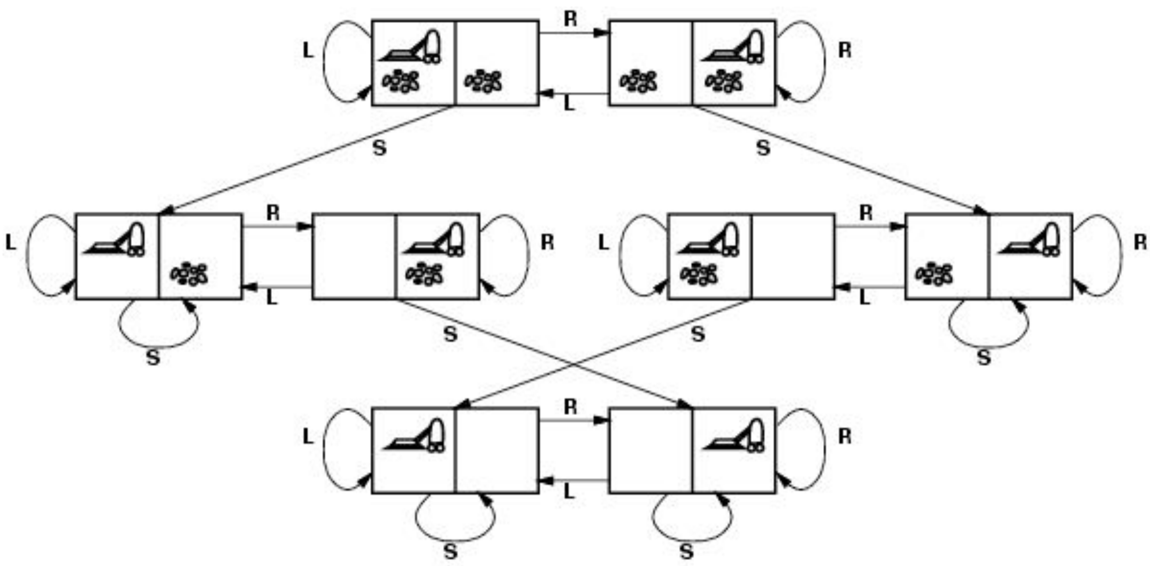
Continuous planning

- Planner persists over time, adapts plan to changed circumstances, reformulates goals if necessary

Incomplete information

Unreliable actions

Recall: the vacuum-world



Conditional planning

Let‘s start with fully observable, but non-deterministic environments

- current state is always known
- outcome of an action is unknown (but present)

Idea:

- deal with uncertainty by checking what is really happening at predetermined points
- build plan with conditional (contingent) steps that probe state of the environment

Problem: How to construct such a conditional plan?

Conditional planning

STRIPS-like description

- Actions: *left, right, suck*
- States: conjunction of *AtL, AtR, CleanL, CleanR*

Now, how to include indeterminism?

- actions can have disjunctive effects (more than one)
  - E.g. moving left sometimes fails

    Action(Left, PRECOND: AtR, EFFECT: AtL)

    ... becomes ...

    Action(Left, PRECOND: AtR, EFFECT: AtL ∨ *AtR)*
- actions can have conditional effects *when <cond.>: effect*

    Action(Left, PRECOND: AtR, EFFECT: AtL ∨ *(AtL* ∧ *when CleanL: ¬CleanL)* ∨ ...

Conditional planning

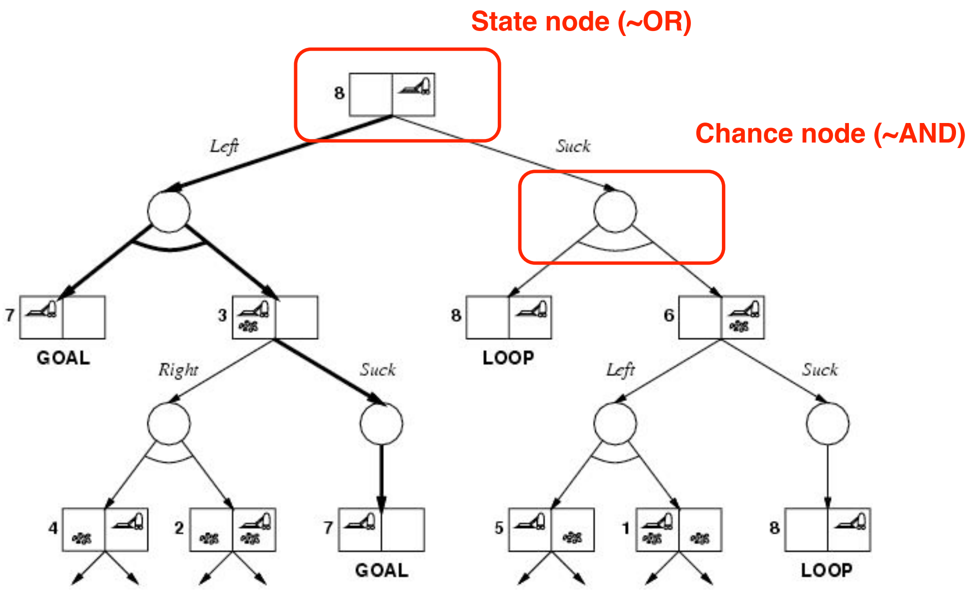
Conditional plans require conditional steps

- If **<test>** then **plan_A** else **plan_B**
- _Example_: **if** $AtL \wedge CleanL$ **then** *Right* **else** *Suck*
- plans become game trees

„Games Against Nature“

- _Goal_: find conditional plans that work, regardless of which action outcomes actually occur, i.e. cope with all possible outcomes of indeterminate actions
- _Example_: vacuum-world
  - with $Initial\ state = AtR \wedge CleanL \wedge CleanR$
  - possibility of depositing dirt when moving to other square
  - possibility of depositing dirt when action is *Suck* („double murphy“ cleaner)

„Game tree“



Solution of „games against nature“

Solution plan is a subtree (no longer a sequence) with

- goal node at every leaf
- one action at each of its state nodes
- every outcome branch at each of its chance nodes

Example: solution in previous example (bold lines)

[*Left*, **if** *AtL* ∧ *CleanL* ∧ *CleanR* **then** [] **else** *Suck*]

For exact solutions use **minimax** algorithm with two modifications

- Max and Min nodes become **OR** and **AND nodes**
  - **OR**: plan is just the action selected at that state node
  - **AND**: plan is nested series of if-then-else steps testing all branches
- Algorithm returns conditional plan instead of single move

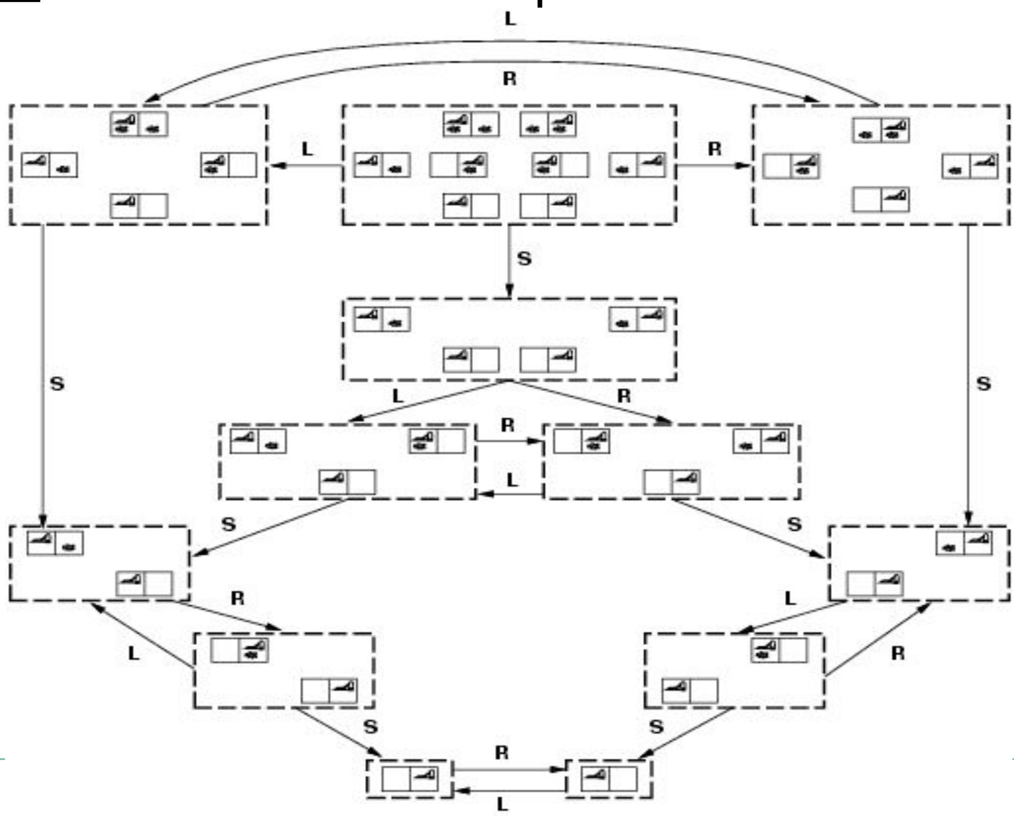
Planning with incomplete information

What if the agent has limited information about the current state of the (partially observable) environment?

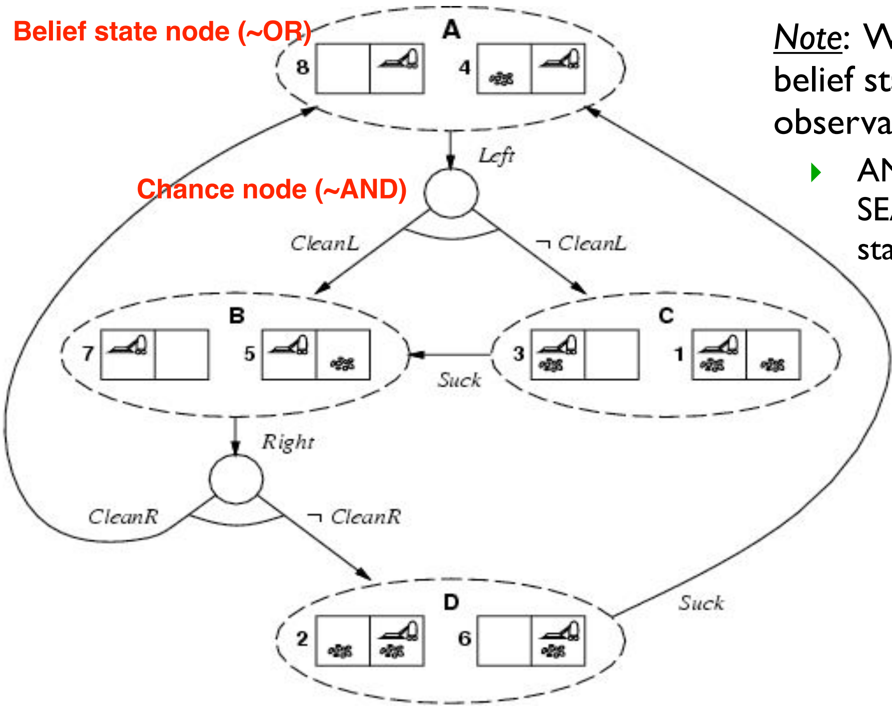
search state = world state → search state = *set of* world states „Belief state“

- Planning as heuristic search in belief state space
- *Example*: assume a vacuum agent that
  - cannot sense presence of dirt in other squares than the one it is on
  - can leave behind dirt when moving to other square
  - Cyclic solution in fully observable world: *keep moving left and right, sucking dirt whenever it appears until both squares are clean and I'm in square left*

Example: Actions in belief state space



Conditional Planning on belief states



Note: We assume that the belief state is always fully observable!

- AND-OR-GRAPH-SEARCH over belief states

Complexity of Conditional Planning

Note: Condition planning is much harder than (already very complex) classical planning problems. Why?

NP problems: exponential number of candidates, but each candidate solution can be checked in polynomial time (true for classical plans)

Conditional Plan: exponential number of candidates, each of which contains multiple states; must check for all possible states whether a path exists that satisfies the goals (cannot be done in polynomial time)

Way out: ignore some contingencies, handle others only when they actually occur

Monitoring & replanning

Even worse: realistic world has unbounded indeterminancy
➔ some unanticipated circumstances will likely arise

Monitor whether everything is going as planned and replan when something unexpected happens

- *action vs. plan monitoring*: verify next action vs. entire remaining plan
- replan by *repairing* old plan, find way back to old plan

Advantages:

- allows to start out with easy plans
- works in both fully and partially observable environments, and with a variety of planning representations

Example: Continuous Activity Scheduling Planning Execution and Replanning   http://ai.jpl.nasa.gov/public/projects/casper/

Problem with batch planning for spacecraft control:

- constructing a plan is computationally intensive and onboard computational resources are typically quite limited
  - Planner on-board the New Millennium Deep Space One mission: ~4 hours to produce a 3 day operations plan (with 25% of CPU load)
- under changing conditions, need to increase the time for which the spacecraft has a consistent plan

Approach: continuous planning and re-planning

- current goal set, a plan, a current state, expected future state
- incremental update invokes planner to maintain consistent plan
- iterative plan repair techniques

Example: Autonomous navigation on Mars

Discussion

Problem: Monitoring & re-planning can lead to less intelligent behavior

- E.g. resource problems would not be detected before an action execution failed

Better: plan monitoring

- check *always* all preconds of entire remaining plan, which are not achieved by another step in the plan
- can also take advantage of serendipity (accidental success)

What if in partially observable environments?

- checking all preconds is difficult, if not impossible
- check only important, fallible, and perceivable variables -> uncertainty remains!

Complete in environments without dead ends, short-coming: time demands of replanning

Summary

Deciding on next action(s) requires planning

- decision-making and planning
- planning as state-based search over plans
- representation of states and actions, classical problems
- partial order planning & planning graphs

Challanging domains require dedicated, very complex formalisms

- conditional (contingent) planning ➔ plans become trees
- sensorless (conformant) planning ➔ world states become belief states
- monitoring & replanning
- very costly, uncertainty remains

Next week(s)

How to model uncertain knowledge and reasoning about it?

- Probabilistic turn
- Bayesian interpretation of prob‘s -- degrees of belief
- Graphical models (networks) to model influence (in-/dependence)
- Probabilistic reasoning (inferences) and decision-making