Spezielle Themen der Künstlichen Intelligenz

6. Termin: Planning & Uncertainty

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

Recap‘ planning

Formulating planning problems: STRIPS, ADL

Planning as state-space search

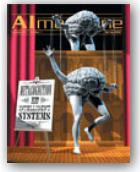
- progression: start ➔ goal, follow action effects
- regression: goal ➔ start, follow action preconditions

Partial order planning

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

Planning graphs

- single graph structure with *all possible worlds and plans*
- mutex relations between actions or literals
- used to extract solution, estimate cost heuristics



Time, schedules and resources

Until know: what actions to do?

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: Interactive Scheduling

Planning vs. scheduling

How does the scheduling problem differ from a standard planning problem?

- need to determine when an action should start and when it should end
- need to consider order (planning) *and* duration

Critical path method to determine start and end times

- Path = linear action sequence from start to end
- Critical path = path with longest total duration
  - determines the duration of the entire plan
  - should be executed without delay

Example: Car construction scheduling

Init(Chassis(C1) ∧ Chassis(C2) ∧ Engine (E1,C1,30) ∧ Engine(E1,C2,60) ∧ Wheels(W1,C1,30) ∧ Wheels (W2,C2,15))

Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e,c,m)

PRECOND: *Engine(e,c,d) ∧ Chassis(c) ∧ ¬EngineIn(c)*

EFFECT: *EngineIn(c) ∧ Duration(d))*

Action(AddWheels(w,c)

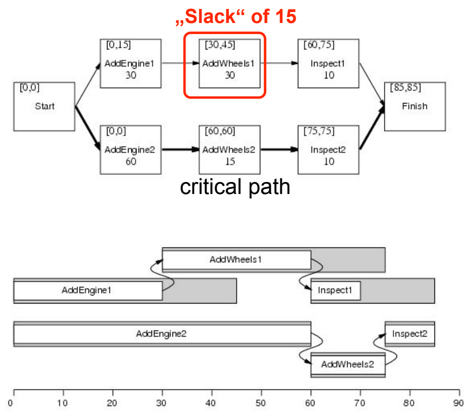
PRECOND: *Wheels(w,c,d) ∧ Chassis(c)*

EFFECT: *WheelsOn(c) ∧ Duration(d))*

Action(Inspect(c)

PRECOND: *EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)*

EFFECT: *Done(c) ∧ Duration(10))*



Hierarchical task network planning

How to cope with hugely complex problems?

⇒ exploit hierarchical structure of the problem domain

⇒ hierarchical decomposition

- at each level a computational task is reduced to a small number of less complex activities at the next lower level
- the computational cost of arranging these activities is low

Hierarchical task network (HTN) planning: action refinement through decomposition

- Building a house = getting a permit + hiring a contractor + doing the construction + paying the contractor
- Refined until only primitive actions remain

„Hybrid HTN“: combine HTN with POP

Representing action decomposition

General descriptions stored in plan library

- Each method:
  Decompose(a,d) = action ***a*** can be decomposed into PO plan ***d***

Start action supplies all preconditions of actions not enabled as effects of other actions in the plan

= external preconditions

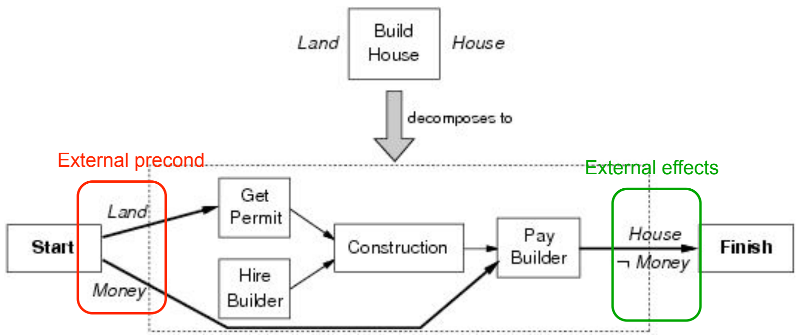
Finish action has (as preconditions) all effects of actions in the plan not negated by other actions

= external effects

- primary effects (used to achieve goal) vs. secondary effects

Buildhouse example

Build-house decomposed into plan with 4 lower level actions



Buildhouse example

Action(Buyland, PRECOND: *Money,* EFFECT: *Land* ∧ ¬*Money)*

Action(GetLoan, PRECOND: *Goodcredit,* EFFECT: *Money* ∧ *Mortgage)*

Action(BuildHouse, PRECOND: ***Land***, EFFECT: ***House****)*

Action(GetPermit, PRECOND: ***Land***, EFFECT: *Permit)*

Action(HireBuilder, EFFECT: *Contract)*

Action(Construction, PRECOND: *Permit* ∧ *Contract,* EFFECT: *HouseBuilt* ∧ ¬*Permit),*

Action(PayBuilder, PRECOND: ***Money*** ∧ *HouseBuilt,* EFFECT: ¬***Money*** ∧ ***House*** ∧ ¬ *Contract),*

Decompose(BuildHouse,

*Plan ::*STEPS*{ S1: GetPermit, S2:HireBuilder, S3:Construction, S4:PayBuilder}*

ORDERINGS*: {Start <S1<S3<S4<Finish, Start<S2<S3},*

LINKS

$$\left\{ \begin{array}{l} Start \xrightarrow{Land} S1, Start \xrightarrow{Money} S4, S1 \xrightarrow{Permit} S3, S2 \xrightarrow{Contract} S3, \\ S3 \xrightarrow{HouseBuilt} S4, S4 \xrightarrow{house} Finish, S4 \xrightarrow{\neg Money} Finish \end{array} \right\}$$

Properties of decomposition

Should be a correct implementation of action **a**

- Correct if plan **d** is a complete and consistent PO plan for the problem of achieving effects of **a**, given the preconditions of **a**

Not necessarily unique

Performs information hiding

- higher-level action description hides preconditions+effects
- ignores all internal effects of decomposition
- does not specify intervals inside the activity during which preconditions and effects must hold

Information hiding is essential to HTN planning

- reduces complexity in reasoning about abstract actions

Hybrid HTN planning: adopt POP

Recall POP

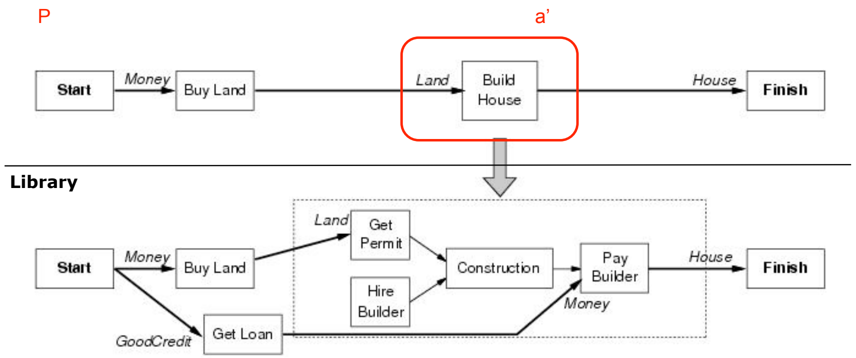
- Start with empty plan Start<Finish, open preconds of Finish
- Successfor function: picks one open precondition p of action B and generates successor plan for *every possible consistent way* of choosing action A that achieves p

Now, modify the successor function to apply decomposition to the current plan

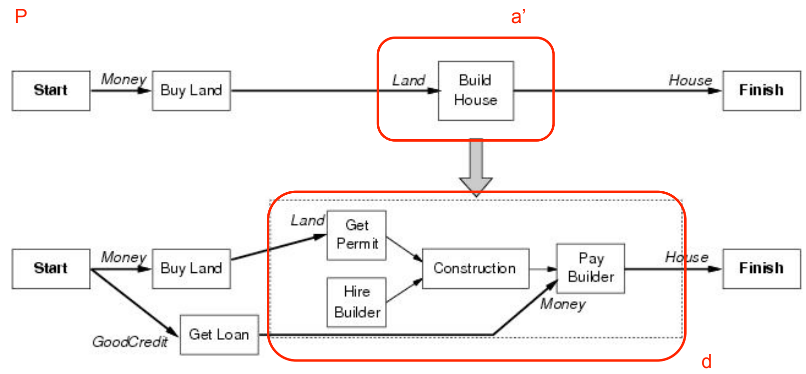
- Select *non-primitive* action **a'** in current partial plan **P**

For any **Decompose(a,d)** method in library, where **a** and **a'** unify with substitution θ, do: Replace **a'** with **d'** $= subst(\theta,$**d**$)$

POP+HTN example



POP+HTN example



How to hook up **d** in **a'**?

1. Replace action **a'** in **P** with **dθ**

2. Connect ordering steps for **a'** to steps in **d'**
   - Maintain constraints of the form **B<a'** in **P** for steps **s** of **d'**
   - Watch out for too strict orderings, e.g. simply setting **B<s** for every step **s** with **Start<s**
   - Record reasons for constraints and relax as possible

3. Connect causal links
   - If **B-p->a'** is a causal link in **P**, replace by a set of causal links from **B** to all steps in **d'** with preconds **p** supplied by **Start** step
     - Ex: BuyLand-Land->BuildHouse replaced by BuyLand-Land->GetPermit
   - Analog for **a'-p->C**
     - Ex: PayBuilder-House->Finish

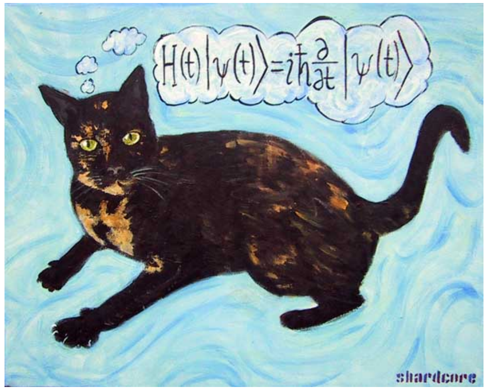
Discussion of HTN planning

Problem: decomposition becomes undecidable when recursive actions can be taken, can be coped with by bounding recursion and using POP

Complexity: *d* possible decompositions into *k* actions at next level, *n* primitive actions
- $1+k+k^2+...+k^{\log_k (n-1)}=(n-1)/(k-1)$ internal decomp nodes
- that is, one can have $d^{(n-1)/(k-1)}$ possible decomposition trees
- efficient only with *d* small, *k* large = small lib of long decomp's

In practice: almost all large-scale planners are HTN-based
- allows human expert to provide crucial knowledge
- Example: O-PLAN used e.g. for HITACHI production plans

What if one cannot be certain?



Uncertain domains

So far, observable, static, deterministic domains

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

But, in reality we have uncertain environments

- incomplete: partially observable, non-deterministic
- incorrect, incomplete information: world and beliefs may differ

Degree of uncertainty depends on indeterminacy

- Bounded: actions can have unpredictable effects, but these can be listed in action description axioms
- Unbounded: preconditions and effects are unknown or too large to enumerate

Handling indeterminacy in planning

Sensorless planning (conformant planning)

- Find plan that achieves goal *in all possible circumstances* (regardless of initial state and action effects)

Conditional planning (contingency planning)

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

Execution monitoring and replanning

- While constructing and executing a plan, judge whether plan requires revision

Continuous planning

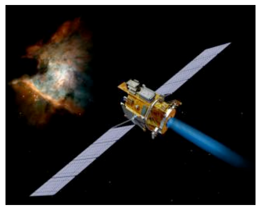
- Planner persists over time: adapt plan to changed circumstances, reformulate goals if necessary

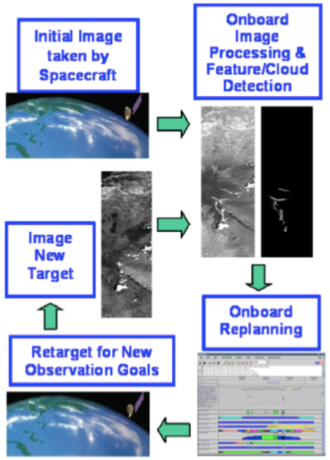
Example: A.I. in space

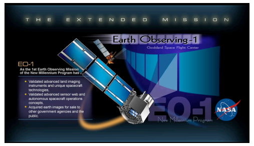
http://www.aaai.org/aitopics/pmwiki/pmwiki.php/AITopics/Astronomy

"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."

- from NASA's DEEP SPACE 1 - REMOTE AGENT site







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.“

Continuous Activity Scheduling Planning Execution and Replanning

http://ai.jpl.nasa.gov/public/projects/casper/

Problem with batch planning for spacecraft control:

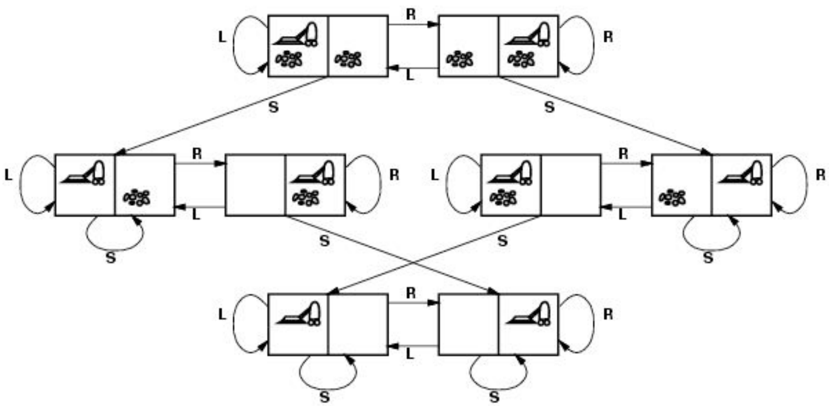
- constructing a plan is computationally intensive and onboard computational resources are typically quite limited
  - Planner on-board the New Millennium Deep Space One mission: ~4 hours to produce a 3 day operations plan (with 25% of CPU load)
- under changing conditions, need to increase the time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

Approch: continuous planning and replanning

- 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

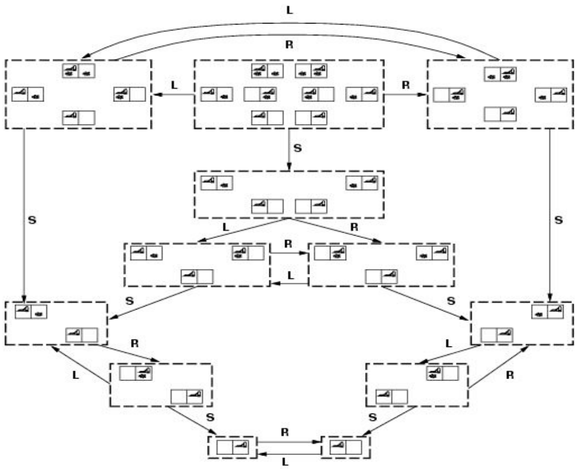
Recall: the vacuum-world



Sensorless planning

Find plan that works *without* sensory info

- plan over sets of world states
- employ coercive actions



Conditional planning

Deal with uncertainty by checking what is really happening at predetermined points

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

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

Build plan with conditional steps that check 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*

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

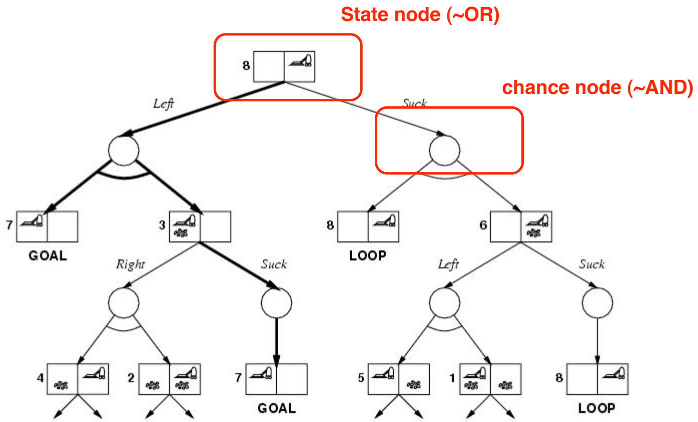
- If **<test>** then **plan_A** else **plan_B**
- Example: **if** *AtL∧CleanL* **then** *Right* **else** *Suck*
- plans become (game) trees

„Games Against Nature"

- goal: find conditional plans that work, regardless of which action outcomes actually occur
- assume vacuum-world:

  Initial state = AtR ∧ CleanL ∧ CleanR
- „double murphy" cleaner:
  possibility of despositing dirt when moving to other square, and
  possibility of despositing dirt when action is Suck

„Game tree“



Solution of „games against nature“

Solution is a subtree that
- has a goal node at every leaf
- specifies *one* action at each of its *state nodes*
- includes *every* outcome branch at each of the *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 state node
  - **AND**: plan is nested series of if-then-else steps
- Algorithm returns conditional plan instead of single move

And-Or-search algorithm

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan or failure
  return OR-SEARCH(INITIAL-STATE[problem], problem, [])
```

```
function OR-SEARCH(state, problem, path) returns a conditional plan or failure
  if GOAL-TEST[problem](state) then return the empty plan
  if state is on path then return failure  /* detected circle */
  for action,state_set in SUCCESSORS[problem](state) do
    plan ← AND-SEARCH(state_set, problem, [state | plan] )
    if plan ≠ failure then return [action | plan]
  return failure
```

```
function AND-SEARCH(state_set, problem, path) returns a conditional plan or failure
  for each s_i in state_set do
    plan_i ← OR-SEARCH(s_i, problem,path )
    if plan = failure then return failure
  return [ if s_1 then plan_1 else if s_2 then plan_2 else … if s_{n-1} then plan_{n-1} else plan_n]
```

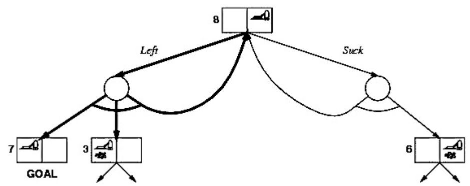

Problems with cycles

When a state already appeared on the path, return failure
- ensures algorithm termination

But sometimes only cyclic solutions exist
- Consider „tripple murphy“: += sometimes a move is not performed

  [*Left*, **if** *CleanL* **then** [] **else** *Suck*] is *not* a solution
- Must repeat parts of the plan until it works, but may run into infinite loops; need labels to denote portions of the plan

  [*L1*: *Left*, **if** *AtR* **then** *L1* **else if** *CleanL* **then** [] **else** *Suck*]



Partially observable environment

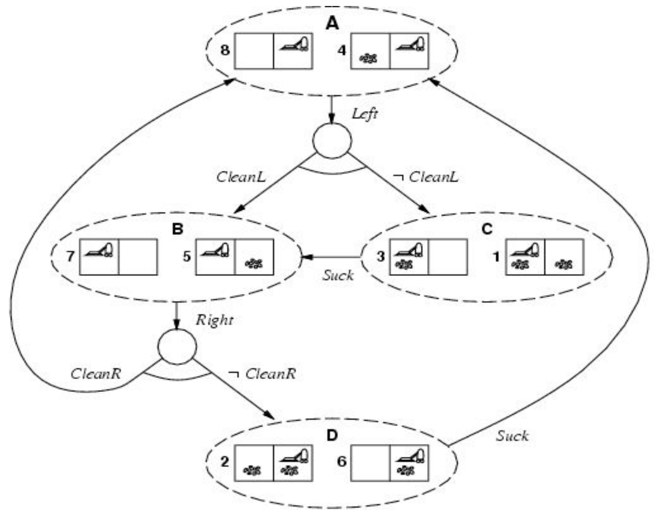
What if the agent has limited information about the current state of the environment?

Modeled as a set of possible states = the agent's belief states

- E.g. assume a vacuum agent that
  - cannot sense presence or absence 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

Plan on AND-OR-graph of belief states



Conditional Planning on belief states

Note: Belief state is always fully observable!

- can use AND-OR-GRAPH-SEARCH on belief states

Representation of belief states? 3 choices:

- sets of full state descriptions, easy but expensive

  {(*AtR*∧*CleanR*∧*CleanL*) ∨ (*AtR*∧*CleanR*∧¬*CleanL*)}
- logical sentences that capture the set of possible worlds in the belief state (*open-world assumption*)

  AtR ∧ *CleanR*
- knowledge propositions describing what the agent knows (*closed-world assumption*: the rest is assumed false)

  K(*AtR*) ∧ K(*CleanR*)

Choice 2 and 3 are roughly equivalent, let's continue with 3

How to „feed" knowledge propositions?

Sensing in Conditional Planning

- Automatic sensing: At every time step the agent gets all available percepts
- Active sensing: Percepts are obtained through specific sensory actions that must be planned for

  ➔ additional actions *checkDirt* and *checkLocation*

Given the knowledge proposition representation and the sensing, action descriptions can now be formulated in STRIPS:

Action(Left, PRECOND: AtR,
EFFECT: K(AtL)∧¬K(AtR) ∧
when CleanR: ¬K(CleanR) ∧
when CleanL: K(CleanL) ∧
when ¬CleanL: K(¬CleanL)).

Assumes automatic sensing, otherwise required explicit sensory actions.

Complexity of Conditional Planning

Harder than classical planning problems. Why?

NP problems: exponentional number of candidates, but each candidate solution can be checked in polynomial time

- true for classical plans

Conditional Plan: exponentional number of candidates, each of which contains multiple states; must check for all possible states whether some 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

Realistic world: unbounded indeterminancy ➔ some unanticipated circumstances will 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

Replanning-agent

Action monitoring

```
function REPLANNING-AGENT(percept) returns an action
   static: KB, a knowledge base (+ action descriptions)
         plan, a plan initially []   /* remaining unexecuted plan segment */
         whole_plan, a plan initially []
         goal, a goal
   TELL(KB, MAKE-PERCEPT-SENTENCE(percept,t))
   current ← STATE-DESCRIPTION(KB,t)
   if plan = [] then   /* create initial plan */
         whole_plan ← plan ← PLANNER(current, goal, KB)
   if PRECONDITIONS(FIRST(plan)) not currently true in KB then
         candidates ← SORT(whole_plan, ordered by distance to current)
      /* find state s in old plan, closest to current, and a plan current->s */
         find state s in candidates such that
            failure ≠ repair ← PLANNER(current, s, KB)
         continuation ← the tail of whole_plan starting at s
         whole_plan ← plan ← APPEND(repair, continuation)
   return POP(plan)
```


Discussion: monitoring & replanning

Algorithm 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

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

Trends: interactive planning

Off-load some planning decisions onto the user

Allow for incremental planning with incessant input by the user

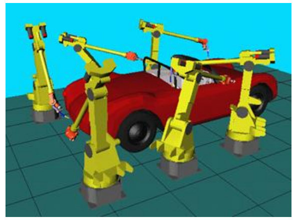
Trends: multi-agent planning

Planning in environments with other agents

- cooperative
- competitive

May simply add other agents to own world model

- Problem: agents need to be treated differently



Cooperation: Joint goals and plans

Multi-agent planning: e.g. double tennis where agents want to return ball

- agent as parameter of actions: *At(A,[Left,Baseline])*
- solution: joint plan with actions for each agent and committment of each agent
- coordination required for agents to reach same joint plan
- at coordination problems, communicate!

Multi-body planning: one agent plans actions of everybody using slightly extended POP

- world not static, need to plan synchronization
- plan joint actions: *<Go(A,[Left,Net]), Go(B,[Right,Baseline]>*
- ...or add concurrent action conditions to actions

Cooperation to ensure agreement on joint plan

- Convention: constraint on the selection of joint plans
- Communication

Next week(s)

How to model uncertain knowledge and reasoning about it?

- Probability theory
- Degrees of belief
- Bayesian (belief) networks, influence diagrams, graphical probabilistic models, ...
- Inference in Bayesian networks