Artificial Intelligence 1: planning in the real world

Lecturer: Tom Lenaerts
Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA)
Université Libre de Bruxelles

Outline

- Time, schedules and resources
- Hierarchical task network planning
- Non-deterministic domains
  - **Conditional planning**
  - **Execution monitoring and replanning**
  - **Continuous planning**
- Multi-agent planning

Time, schedules and resources

- Until know:
  - **what actions to do**
- Real-world:
  - **+ actions occur at certain moments in time.**
  - **+ actions have a beginning and an end.**
  - **+ actions take a certain amount of time.**
- Job-shop scheduling:
  - **Complete a *set of jobs*, each of which consists of a *sequence of actions*,**
  - **Where each action has a given *duration* and might require *resources*.**
  - **Determine a schedule that *minimizes the total time* required to complete all jobs (respecting resource constraints).**

Car construction example

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

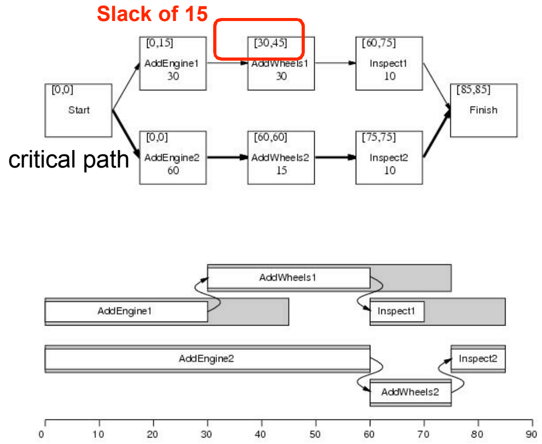
Goal(Done(C1) ∧ Done(C2))

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))*

Solution found by POP



Planning vs. scheduling

- How does the problem differ from a standard planning problem?
- When does an action start and when does it end?
  - **So next ot order (planning) duration is also considered**
    Duration(d)
- *Critical path method* is used to determine start and end times:
  - **Path = linear sequence from start to end**
  - **Critical path = path with longest total duration**
    - Determines the duration of the entire plan
    - Critical path should be executed without delay

ES and LS

- Earliest possible (ES) and latest possible (LS) start times.
- LS-ES = slack of an action
- for all actions determines the schedule for the entire problem.

  $ES(Start) = 0$

  $ES(B)=max_{A<B}\ ES(A) + Duration(A)$

  $LS(Finish)=ES(Finish)$

  $LS(A) = min_{A<B}\ LS(B) - Duration(A)$
- Complexity is $O(Nb)$ (given a PO)

Scheduling with resources

- Resource constraints = required material or objects to perform task
  - **Reusable resources**
    - A resource that is occupied during an action but becomes available when the action is finished.
    - Require extension of action syntax:

      Resource:R(k)
      - **k units of resource are required by the action.**
      - **Is a pre-requisite before the action can be performed.**
      - **Resource can not be used for k time units by other.**

Car example with resources

Init(Chassis(C1) ∧ Chassis(C2) ∧ Engine(E1,C1,30) ∧ Engine(E1,C2,60) ∧ Wheels(W1,C1,30) ∧ Wheels(W2,C2,15) ∧ EngineHoists(1) ∧ WheelStations(1) ∧ Inspectors(2))

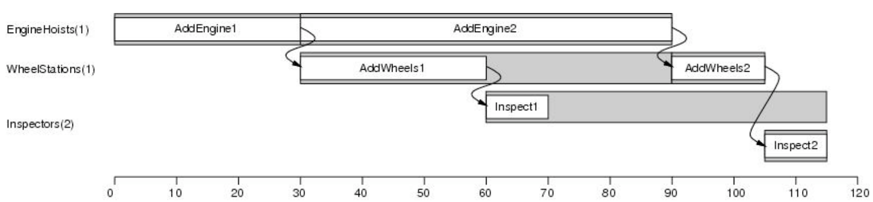
Goal(Done(C1) ∧ Done(C2))

Action(AddEngine(e,c,m)
PRECOND: *Engine(e,c,d) ∧ Chassis(c) ∧ ¬EngineIn(c)*
EFFECT: *EngineIn(c) ∧ Duration(d),*
RESOURCE*: EngineHoists(1))*

Action(AddWheels(w,c)
PRECOND: *Wheels(w,c,d) ∧ Chassis(c)*
EFFECT: *WheelsOn(c) ∧ Duration(d)*
RESOURCE*: WheelStations(1))*

Action(Inspect(c)
PRECOND: *EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)*
EFFECT: *Done(c) ∧ Duration(10)*
RESOURCE*: Inspectors(1))* aggregation

Car example with resources



Scheduling with resources

- Aggregation = group individual objects into quantities when the objects are undistinguishable with respect to their purpose.
  - **Reduces complexity**
- Resource constraints make scheduling problems more complicated.
  - **Additional interactions among actions**
- Heuristic: minimum slack algorithm
  - **Select an action with all pre-decessors scheduled and with the least slack for the earliest possible start.**

Hierarchical task network planning

- Reduce complexity ⇒ hierarchical decomposition
  - **At each level of the hierarchy a computational task is reduced to a *small* number of activities at the next lower level.**
  - **The computational cost of arranging these activities is low.**
- Hierarchical task network (HTN) planning uses a *refinement* of actions through decomposition.
  - **e.g. building a house = getting a permit + hiring a contractor + doing the construction + paying the contractor.**
  - **Refined until only primitive actions remain.**
- Pure and hybrid HTN planning.

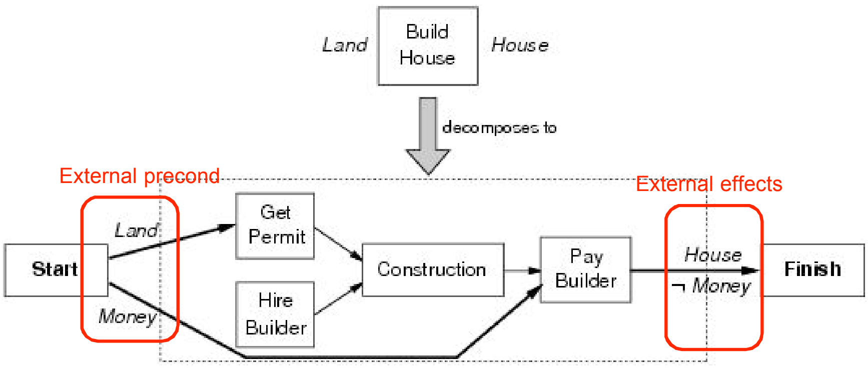
Representation decomposition

- General descriptions are stored in *plan library*.
  - **Each method = Decompos(a,d); a= action and d= PO plan.**
- See buildhouse example
- Start action supplies all preconditions of actions not supplied by other actions.

  =external preconditions
- Finish action has all effects of actions not present in other actions

  =external effects
  - Primary effects (used to achieve goal) vs. secondary effects

Buildhouse example



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

Properties of decomposition

- Should be correct implementation of action *a*
  - **Correct if plan *d* is complete and consistent PO plan for the problem of achieving the effects of *a* given the preconditions of *a*.**
- A decomposition is not necessarily unique.
- Performs information hiding:
  - **STRIPS action description of higher-level action hides some preconditions and effects**
  - **Ignore all internal effects of decomposition**
  - **Does not specify the intervals inside the activity during which preconditions and effects must hold.**
- Information hiding is essential to HTN planning.

Recapitulation of POP (1)

- 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* and
    - generates a successor plan for every possible consistent way of choosing action *A* that achieves *p*.
  - **Test goal**

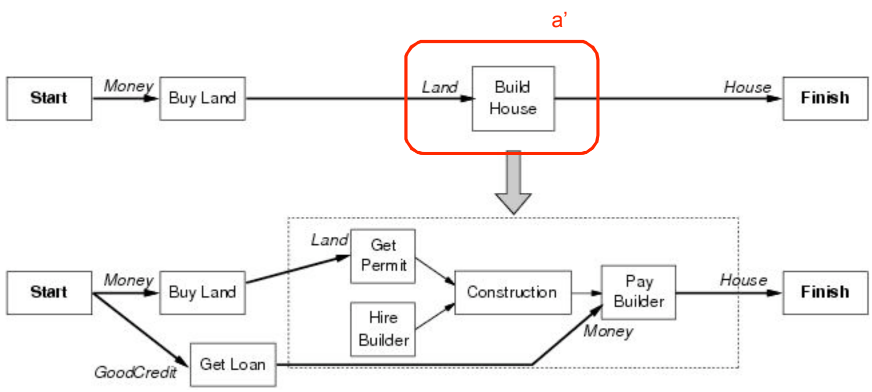
Recapitulation of POP (2)

- When generating successor plan:
  - **The causal link A--p->B and the ordering constraing 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.**

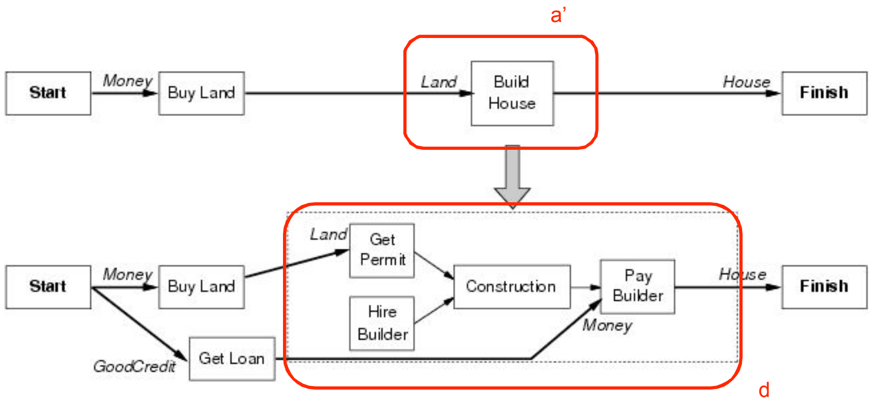
Adapting POP to HTN planning

- Remember POP?
  - **Modify the successor function: apply decomposition to current plan**
- **NEW** Successor function:
  - **Select non-primitive action *a'* in *P***
  - **For any *Decompose(a',d')* method in library where *a* and *a'* unify with substitution θ**
    - Replace *a'* with *d' = subst(θ,d)*

POP+HTN example



POP+HTN example



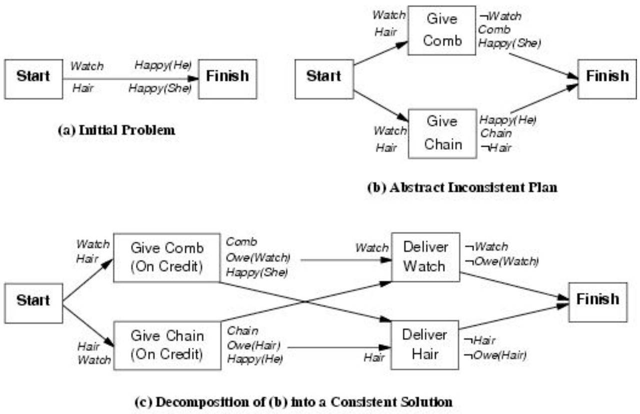
How to hook up *d* in *a'*?

- Remove action *a'* from *P* and replace with *dθ*
  - **For each step s in *d'* select an action that will play the role of *s* (either new *s* or existing *s'* from *P*)**
  - **Possibility of *subtask sharing***
- Connect ordering steps for *a'* to the steps in *d'*
  - **Put all constraints so that constraints of the form *B < a'* are maintained.**
  - **Watch out for too strict orderings !**
- Connect the causal links
  - **If B -p-> a' is a causal link in P, replace it by a set of causal links from B to all steps in d' with preconditions p that were supplied by the start step**
  - **Idem for a' -p-> C**

What about HTN?

- Additional modification to POP are necessary
- BAD news: pure HTN planning is undecidable due to recursive decomposition actions.
  - **Walk=make one step and walk**
- Resolve problems by
  - **Rule out recursion.**
  - **Bound the length of relevant solutions,**
  - **Hybridize HTN with POP**
- Yet HTN can be efficient (see motivations in book)

The Gift of magi



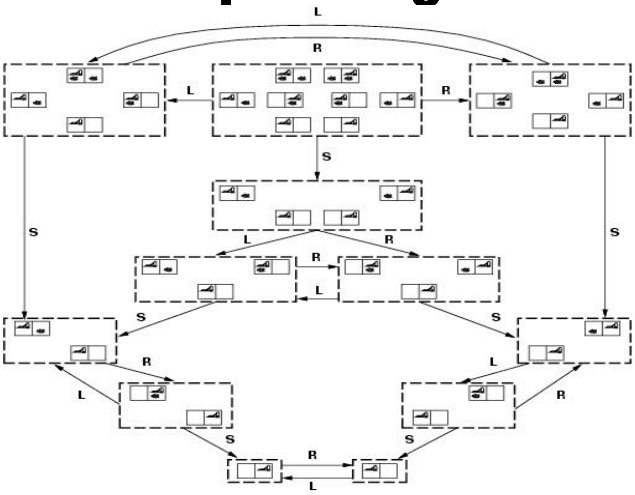
Non-deterministic domains

- So far: fully observable, static and deterministic domains.
  - **Agent can plan first and then execute plan with eyes closed**
- Uncertain environment: incomplete (partially observable and/or nondeterministic) and incorrect (differences between world and model) information
  - **Use percepts**
  - **Adapt plan when necessary**
- Degree of uncertainty defined by indeterminacy
  - **Bounded: actions can have unpredictable effects, yet can be listed in action description axioms.**
  - **Unbounded: preconditions and effects unknown or to large to enumerate.**

Handling indeterminacy

- 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 plan judge whether plan requires revision.**
- Continuous planning
  - **Planning active for a life time: adapt to changed circumstances and reformulate goals if necessary.**

Sensorless planning



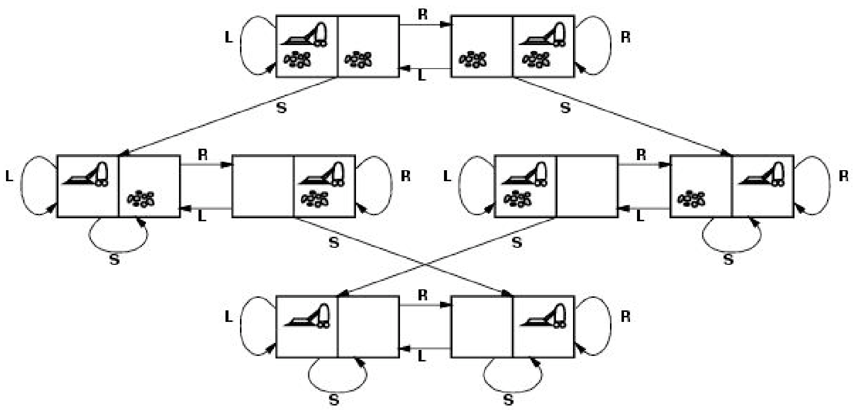
Abstract example

- Initial state = <chair,table, cans of paint, unknown colors>, goal state=<color(table) = color(chair)>
- Sensorless planning (conformant planning)
  - **Open any can of paint and apply it to both chair and table.**
- Conditional planning (Contingency planning)
  - **Sense color of table and chair, if they are the same then finish else sense labels paint if color(label) =color(Furniture) then apply color to othe piece else apply color to both**
- Execution monitoring and replanning
  - **Same as conditional and can fix errors (missed spots)**
- Continuous planning
  - **Can revise goal when we want to first eat before painting the table and the chair.**

Conditional planning

- Deal with uncertainty by checking the environment to see what is really happening.
- Used in fully observable and nondeterministic environments:
  - **The outcome of an action is unknown.**
  - **Conditional steps will check the state of the environment.**
  - **How to construct a conditional plan?**

Example, the vacuum-world



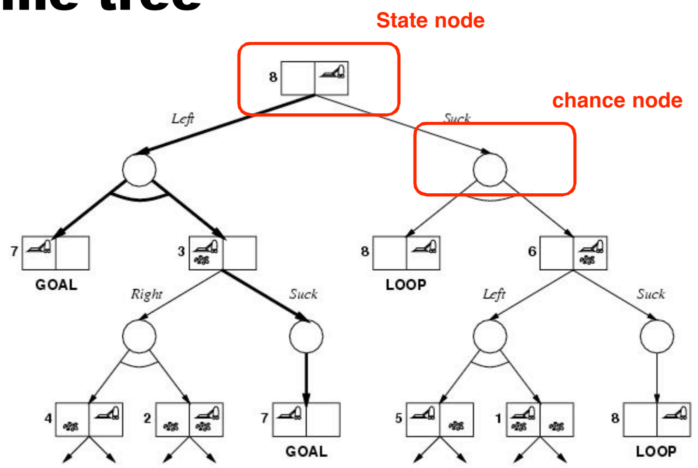
Conditional planning

- Actions: left, right, suck
- Propositions to define states: AtL, AtR, CleanL, CleanR
- How to include indeterminism?
  - **Actions can have more than one effect**
    - 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**
      Action(Left, PRECOND:AtR, EFFECT: AtL∨(AtL∧**when** cleanL: ¬cleanL)
    Both disjunctive and conditional

Conditional planning

- Conditional plans require conditional steps:
  - **If <*test*> then *plan_A* else *plan_B***
    if *AtL∧CleanL* **then** *Right* **else** *Suck*
  - **Plans become trees**
- Games against nature:
  - **Find conditional plans that work *regardless of which action outcomes actually occur.***
  - ***Assume vacuum-world***
    Initial state = AtR ∧ CleanL ∧ CleanR
    Double murphy: possibility of desposit dirt when moving to other square and possibility of despositing dirt when action is Suck.

Game tree



Solution of games against N.

- 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.**
- In previous example:

  [*Left*, **if** *AtL* ∧ *CleanL* ∧ *CleanR* **then** [] **else** *Suck*]
- For exact solutions: use minimax algorithm with 2 modifications:
  - **Max and Min nodes become OR and AND nodes**
  - **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*, [])

---

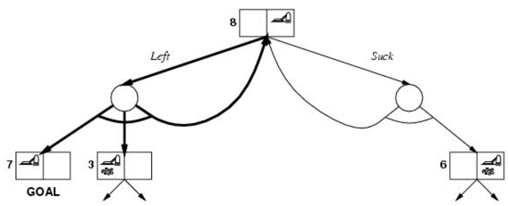
function OR-SEARCH(*state, problem, path*) **returns** *a conditional plan* or *failure*
if GOAL-TEST[*problem*](*state*) **then return** the empty plan
if *state* is on *path* **then return** failure
for *action,state_set* in SUCCESSORS[*problem*](*state*) **do**
plan ← AND-SEARCH(*state_set, problem,*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

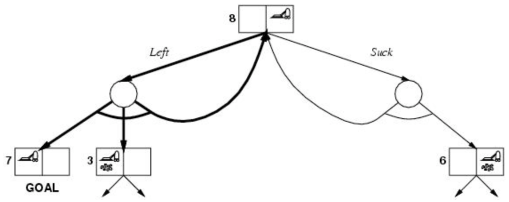
And-Or-search algorithm

- How does it deal with cycles?
  - **When a state that already is on the path appears, return failure**
    - No non-cyclic solution
  - **Ensures algorithm termination**
    - The algorithm does not check whether some state is already on some other path from the root.



And-Or-search algorithm

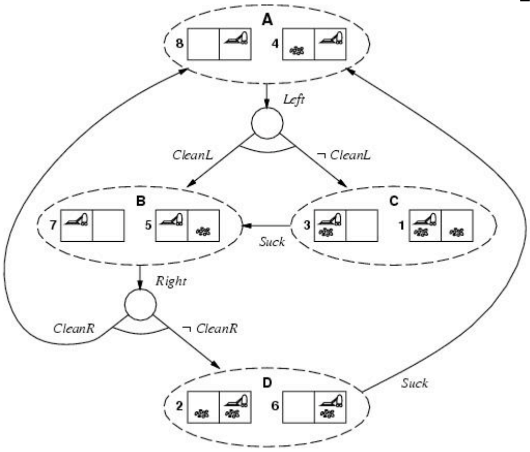
- Sometimes only a cyclic solution exists
  - **e.g. tripple murphy: sometimes the move is not performed**
    [*Left*, **if** *CleanL* **then** [] **else** *Suck*] is not a solution
  - **Use label to repeat parts of plan (*but infinite loops*)**
    [*L1*: *Left*, **if** *AtR* **then** *L1* **else if** *CleanL* **then** [] **else** *Suck*]



CP and partially observable env.

- Fully observable: conditional tests can ask any question and get an answer
- Partially observable???
  - **The agent has limited information about the environment.**
  - **Modeled by a state-set = belief states**
  - **E.g. assume vacuum agent which can not sense presence or absence of dirt in other squares than the one it is on.**
    - + alternative murphy: dirt can be left behind when moving to other square.
    - 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.*

PO: alternate double murphy



Belief states

- Representation?
  - **Sets of full state descriptions**

    $\{(AtR \wedge CleanR \wedge CleanL) \vee (AtR \wedge CleanR \wedge \neg CleanL)\}$
  - **Logical sentences that capture the set of possible worlds in the belief state (OWA)**

    $AtR \wedge CleanR$
  - **Knowledge propositions describing the agent's knowledge (CWA)**

    $\mathrm{K}(AtR) \wedge \mathrm{K}(CleanR)$

Belief states

- Choice 2 and 3 are equivalent (let's continue with 3)
- Symbols can appear in three ways in three ways: positive, negative or unknown: 3^n possible belief states for n proposition symbols.
  - **YET, set of belief sets is a power set of the phyiscal states which is much larger than 3^n**
  - **Hence 3 is restricted as representation**

    Any scheme capable of representing every possible belief state will require $O(2^n)$ bit to represent each one in the worst case.

    The current scheme only requires $O(n)$

Sensing in Cond. Planning

- How does it work?
  - **Automatic sensing**

    At every time step the agent gets all available percepts
  - **Active sensing**

    Percepts are obtained through the execution of specific *sensory actions*.

    checkDirt and *checkLocation*
- Given the representation and the sensing, action descriptions can now be formulated.

Monitoring and replanning

- Execution monitoring: check whether everything is going as planned.
  - **Unbounded indeterminancy: some unanticipated circumstances will arise.**
  - **A necessity in realistic environments.**
- Kinds of monitoring:
  - **Action monitoring: verify whether the next action will work.**
  - **Plan monitoring: verify the entire remaining plan.**

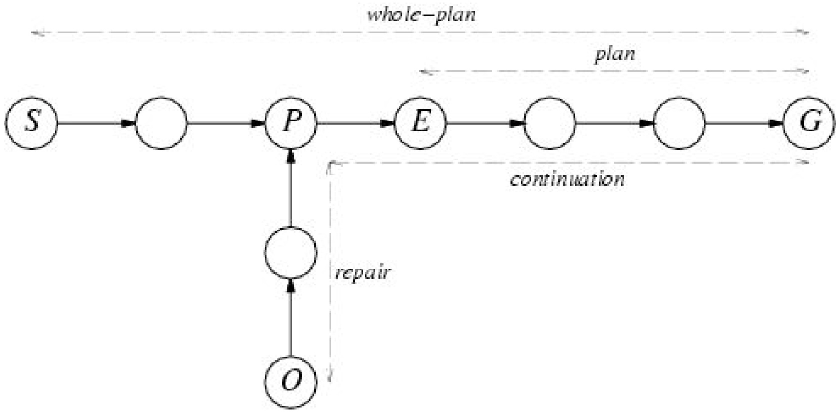
Monitoring and replanning

- When something unexpected happens: replan
  - **To avoid too much time on planning try to repair the old plan.**
- Can be applied in both fully and partially observable environments, and to a variety of planning representations.

Replanning-agent

function REPLANNING-AGENT(*percept*) **returns** *an action*
static: *KB*, a knowledge base (+ action descriptions)
plan, a plan initially []
whole_plan, a plan initially []
goal, a goal
TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*,*t*))
current ← STATE-DESCRIPTION(*KB*,*t*)
if *plan* = [] **then return** the empty 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 *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)*

Repair example



Repair example: painting

Init(Color(Chair, Blue) ∧Color(Table,Green) ∧ ContainsColor(BC,Blue) ∧ PaintCan(BC) ∧ ContainsColor(RC,Red) ∧ PaintCan(RC))

Goal(Color(Chair,x) ∧ Color(Table,x))

Action(Paint(object, color)

PRECOND: *HavePaint(color)*

EFFECT: *Color(object, color))*

Action(Open(can)

PRECOND: *PaintCan(can) ∧ ContainsColor(can,color)*

EFFECT: *HavePaint(color))*

[Start; Open(BC); Paint(Table,Blue), Finish]

Repair example: painting

- Suppose that the agent now perceives that the colors of table and chair are different
  - **Figure out point in whole_plan to aim for**

    Current state is identical as the precondition before *Paint*
  - **Repair action sequence to get there.**

    Repair =[] and plan=[*Paint*, *Finish*]
  - **Continue performing this new plan**

    Will loop until table and chair are perceived as the same.
- Action monitoring can lead to less intelligent behavior
  - **Assume the red is selected and there is not enough paint to apply to both chair and table.**
  - **Improved by doing *plan monitoring***

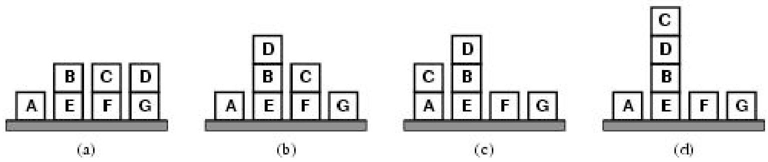
Plan monitoring

- Check the preconditions for success of the *entire* plan.
  - **Except those which are achieved by another step in the plan.**
  - **Execution of doomed plan is cut of earlier.**
- Limitation of replanning agent:
  - **It can not formulate new goals or accept new goals in addition to the current one**

Continuous planning.

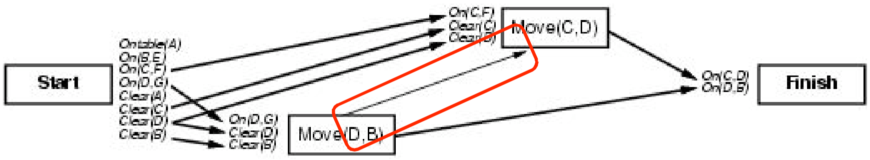
- Agent persists indefinitely in an environment
  - **Phases of goal formulation, planning and acting**
- Execution monitoring + planner as one continuous process
- Example:Blocks world
  - **Assume a fully observable environment**
  - **Assume partially ordered plan**

Block world example



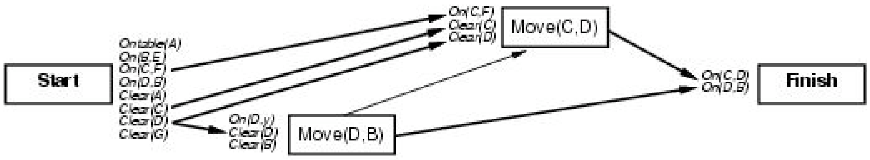
- Initial state (a)
- *Action(Move(x,y),*
  PRECOND: *Clear(x)* ∧ *Clear(y)* ∧ *On(x,z)*
  EFFECT: *On(x,y)* ∧ *Clear(z)* ∧ ¬*On(x,z)* ∧ ¬*Clear(y)*
- The agent first need to formulate a goal: *On(C,D)* ∧ *On(D,B)*
- Plan is created incrementally, return *NoOp* and check percepts

Block world example



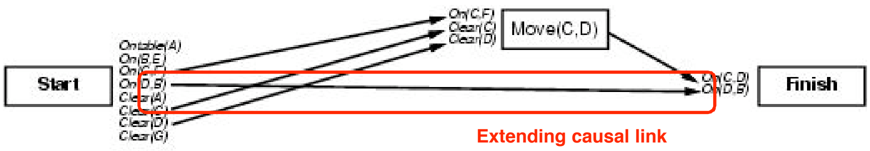
- Assume that percepts don't change and this plan is constructed
- Ordering constraint between *Move(D,B)* and *Move(C,D)*
- Start is label of current state during planning.
- Before the agent can execute the plan, nature intervenes:
  D is moved onto B

Block world example



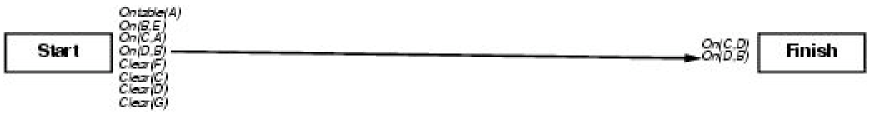
- Start contains now *On(D,B)*
- Agent perceives: *Clear(B)* and On(D,G) are no longer true
  - **Update model of current state (start)**
- Causal links from *Start* to *Move(D,B)* (*Clear(B)* and *On(D,G)*) no longer valid.
- Remove causal relations and two PRECOND of *Move(D,B)* are open
- Replace action and causal links to *Finish* by connecting *Start* to *Finish*.

Block world example



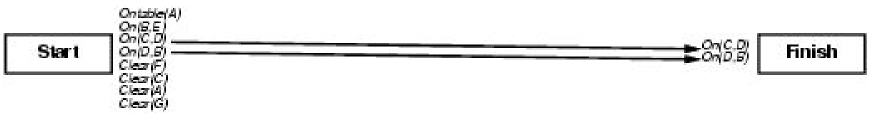
- *Extending*: whenever a causal link can be supplied by a previous step
- All redundant steps (*Move(D,B)* and its *causal links*) are removed from the plan
- Execute new plan, perform action *Move(C,D)*
  - ***This removes the step from the plan***

Block world example



- Execute new plan, perform action *Move(C,D)*
  - **Assume agent is clumsy and drops C on A**
- No plan but still an open PRECOND
- Determine new plan for open condition
- Again *Move(C,D)*

Block world example



- Similar to POP
- On each iteration find plan-flaw and fix it
- Possible flaws: Missing goal, Open precondition, Causal conflict, Unsupported link, Redundant action, Unexecuted action, unnecessary historical goal

Multi-agent planning

- So far we only discussed single-agent environments.
- Other agents can simply be added to the model of the world:
  - **Poor performance since agents are not indifferent ot other agents' intentions**
- In general two types of multiagent environments:
  - **Cooperative**
  - **Competitive**

Cooperation: Joint goals and plans

- Multi-planning problem: assume double tennis example where agents want to return ball.

Agents(A,B)

Init(At(A,[Left,Baseline]) ∧ *At(B,[Right, Net])* ∧ *Approaching(Ball,[Right, Baseline])* ∧ *PArtner(A,B)* ∧ *Partner(B,A))*

Goal(Returned(Ball) ∧ *At(agent,[x,Net]))*

Action(Hit(agent, Ball)

PRECOND: *Approaching(Ball,[x,y])* ∧ *At(agent,[x,y])* ∧ *Partner(agent, partner)* ∧ ¬*At(partner,[x,y])*

EFFECT: *Returned(Ball))*

Action(Go(agent,[x,y])

PRECOND: *At(agent,[a,b])*

EFFECT: *At(agent,[x,y])* ∧ ¬ *At(agent,[a,b]))*

Cooperation: Joint goals and plans

- A solution is a *joint-plan* consisting of actions for both agents.
- Example:

    A: [Go(A,[Right, Baseline]), Hit(A,Ball)]
    B: [NoOp(B), NoOp(B)]

  Or

    A: [Go(A,[Left, net), NoOp(A)]
    B: [Go(B,[Right, Baseline]), Hit(B, Ball)]
- *Coordination* is required to reach same joint plan

Multi-body planning

- Planning problem faced by a single centralized agent that can dictate action to each of several physical entities.
- Hence not truly multiagent
- Important: synchronization of actions
  - **Assume for simplicity that every action takes one time step and at each point in the joint plan the actions are performed simultaneously**

    [<Go(A,[Left,Net]), Go(B,[Right,Baseline]>;
    <NoOp(A), Hit(B, Ball)>]
  - **Planning can be performed using POP applied to the set of all possible joint actions.**
    - Size of this set???

Multi-body planning

- Alternative to set of all joint actions: add extra concurrency lines to action description
  - **Concurrent action**

    Action(Hit(A, Ball)
    CONCURRENT*: ¬Hit(B,Ball)*
    PRECOND: *Approaching(Ball,[x,y]) ∧ At(A,[x,y])*
    EFFECT: *Returned(Ball))*
  - **Required actions (carrying object by two agents)**

    Action(Carry(A, cooler, here, there)
    CONCURRENT*: Carry(B,cooler, here there)*
    PRECOND: *...)*
- Planner similar to POP with some small changes in possible ordering relations

Coordination mechanisms

- To ensure agreement on joint plan: use *convention*.
  - **Convention = a constraint on the selection of joint plans (beyond the constraint that the joint plan must work if the agents adopt it).**

    e.g. stick to your court or one player stays at the net.
- Conventions which are widely adopted= social laws e.g. language.
- Can be domain-specific or independent.
- Could arise through evolutionary process (flocking behavior).

Flocking example

- Three rules:
  - **Separation:**

    Steer away from neighbors when you get too close
  - **Cohesion**

    Steer toward the average position of neighbors
  - **Alignment**

    Steer toward average orientation (heading) of neighbors
- Flock exhibits *emergent behavior* of flying as a pseudo-rigid body.

Coordination mechanisms

- In the absence of conventions: Communication

  e.g. Mine! Or Yours! in tennis example

- The burden of arriving at a succesfull joint plan can be placed on
  - **Agent designer (agents are reactive, no explicit models of other agents)**
  - **Agent (agents are deliberative, model of other agents required)**

Competitive environments

- Agents can have conflicting utilities

  e.g. zero-sum games like chess
- The agent must:
  - **Recognise that there are other agents**
  - **Compute some of the other agents plans**
  - **Compute how the other agents interact with its own plan**
  - **Decide on the best action in view of these interactions.**
- Model of other agent is required
- YET, no commitment to joint action plan.