Overview

I Artificial Intelligence
II Problem Solving
III Knowledge, Reasoning, Planning
    7. Logical Agents
    8. First-Order Logic
    9. Inference in First-Order Logic
    10. Knowledge Representation
    11. Automated Planning
IV Uncertain Knowledge and Reasoning
V Machine Learning
VI Communicating, Perceiving, and Acting
VII Conclusions

Automated Planning

„Je planmäßiger die Menschen vorgehen, desto wirksamer trifft sie der Zufall“ (F. Dürrematt)

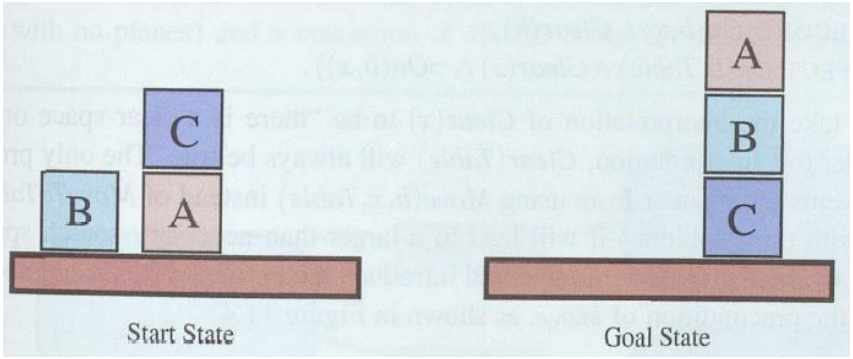
„Ja, mach nur einen Plan! Sei nur ein großes Licht! Und mach dann noch’nen zweiten Plan. Gehn tun sie beide nicht.“ (B. Brecht)

- Goal: Planning a course of actions to reach a goal
  - In contrast to search, planning uses a **factored representation language**

- Definition of Classical Planning
- Algorithms for Classical Planning
- Heuristics for Planning
- Hierarchical Planning
- Planning, Acting in Nondeterministic Domains
- Time, Schedules, and Resources
- Analysis of Planning Approaches

Definition of Classical Planning

- Classical planning finds a sequence of actions to reach a goal in a discrete, deterministic, static, fully observable environment
- Example (right):
  - Input: initial state, goal state, actions
  - Output: Sequence of actions



- Uses factored representation: Planning Domain Definition Language (PDDL)
  - Subset of first-order logic (different versions)
  - Does not need domain-specific knowledge
  - Uses database semantics (closed world assumption, unique name assumption)
  - Does allow variables in actions and in the goal only
  - States without variables, negation, discunction, function symbols

Examples for Representation

- **Action schema**: Family of ground actions consisting of a name, variables, precondition, effect
  - Precondition and effect: Conjunction of literals (positive or negated atomic sentences)

  Action (Fly (p, from, to)

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

  EFFECT: $\neg$At(p,from) $\wedge$ At(p,to))
- Possible resulting ground action:

  Action (Fly (P_1, SFO, JFK)

  PRECOND: At(P_1, SFO) $\wedge$ Plane(P_1) $\wedge$ Airport(SFO) $\wedge$ Airport(JFK)

  EFFECT: $\neg$At(P_1, SFO) $\wedge$ At(P_1, JFK))
- **State**: Conjunction of ground atomic fluents: literals without variables; may change over time
  - Propositional, e.g. „Poor $\wedge$ Unknown“
  - With predicates, e.g. „In (LKW1, Melbourne) $\wedge$ In (LKW2, Sydney)"
    - But **not** e.g. „In (x, y)“ or „In (Father(Fred), Sydney)“
- **Goal**: Like PRECOND

Example Domain: Air Cargo Transportation

$$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$$
$$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$$
$$\land Airport(JFK) \land Airport(SFO))$$
$$Goal(At(C_1, JFK) \land At(C_2, SFO))$$

$$Action(Load(c, p, a),$$
$$\text{PRECOND}: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT}: \neg At(c, a) \land In(c, p))$$
$$Action(Unload(c, p, a),$$
$$\text{PRECOND}: In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT}: At(c, a) \land \neg In(c, p))$$
$$Action(Fly(p, from, to),$$
$$\text{PRECOND}: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$$
$$\text{EFFECT}: \neg At(p, from) \land At(p, to))$$

Solution:

Load (C_1, P_1, SFO)

Fly (P_1, SFO, JFK)

Unload (C_1, P_1, JFK)

Load (C_2, P_2, JFK)

Fly (P_2, JFK, SFO)

Unload (C_2, P_2, SFO)

Example Domain: Blocks World

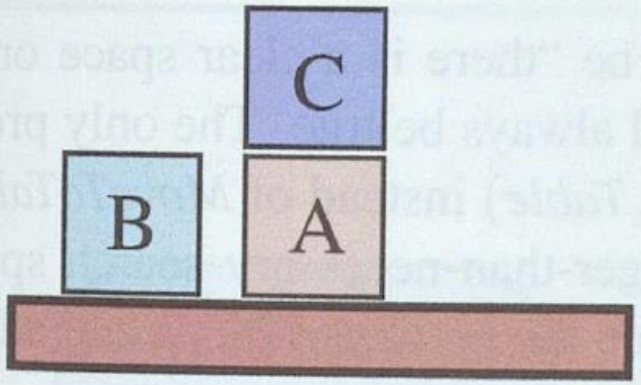
$Init(On(A,Table) \land On(B,Table) \land On(C,A)$
$\land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table))$
$Goal(On(A,B) \land On(B,C))$
$Action(Move(b,x,y),$
PRECOND: $On(b,x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land$
$(b \neq x) \land (b \neq y) \land (x \neq y),$
EFFECT: $On(b,y) \land Clear(x) \land \neg On(b,x) \land \neg Clear(y))$
$Action(MoveToTable(b,x),$
PRECOND: $On(b,x) \land Clear(b) \land Block(b) \land Block(x),$
EFFECT: $On(b,Table) \land Clear(x) \land \neg On(b,x))$

Solution: MoveToTable (C, A)

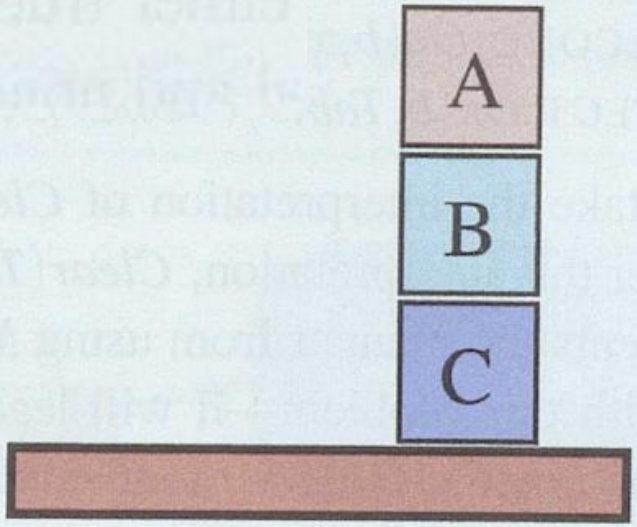
Move (B, Table, C)

Move (A, Table, B)

Frank Puppe



Start State



Goal State

Example Domain: Spare Tire Problem

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$
PRECOND: $At(obj, loc)$
EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$
$Action(PutOn(t, Axle),$
PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \land \neg At(Spare, Axle)$
EFFECT: $\neg At(t, Ground) \land At(t, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

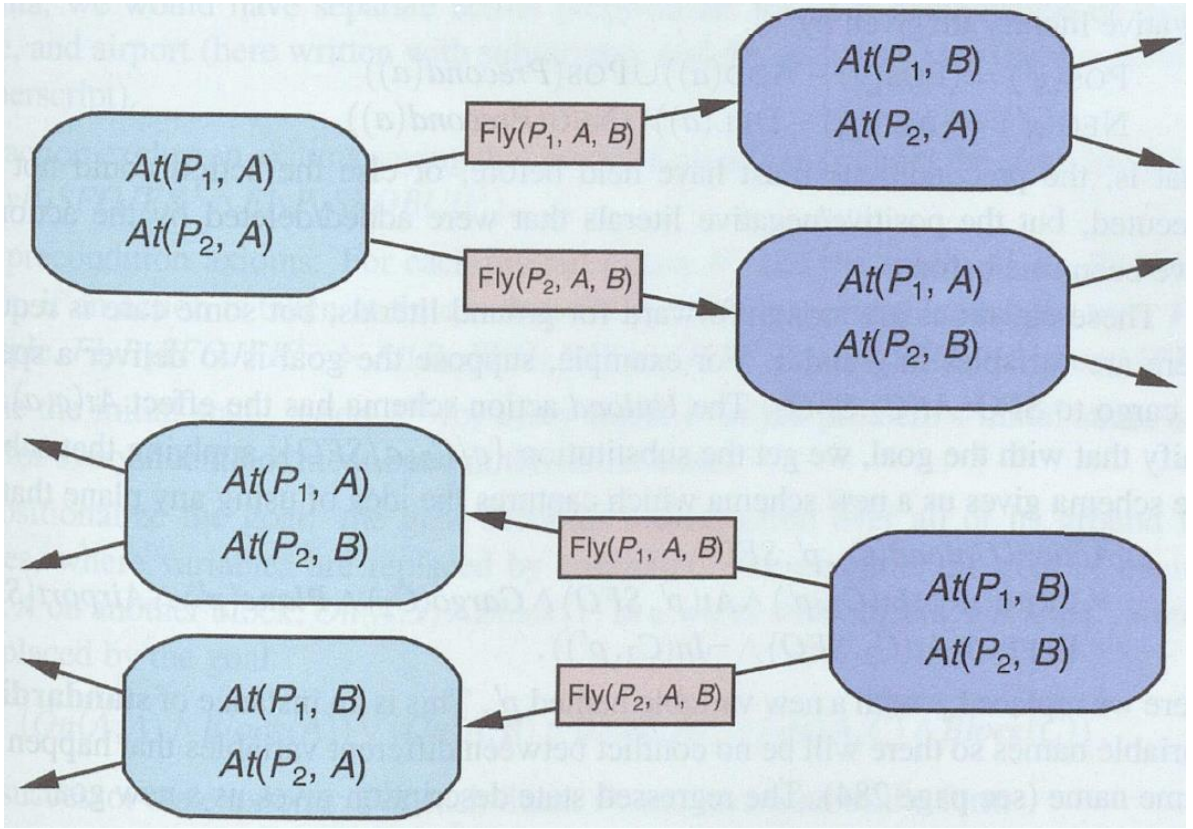
Solution: Remove (Flat, Axle)
Remove (Spare, Trunk)
Puton (Spare, Axle)

Designing Actions

- When formulating actions schemes, special care is necessary to represent exactly what is intended, e.g.
  - Blocks world: Move (block, x, y): If we move a block on another block, the other block is not clear, i.e. it is not possible to put a second block on it. But if we move a block on the table, it is still possible to move further blocks on the table
    - ➢ Two actions for Move and MoveToTable necessary!
  - Air cargo transportation: If several pieces of cargo are loaded in a plane, they should fly with the plane. However we cannot state that in PDDL, because we lack an universal quantifier.
    - ➢ Therefore we differentiate between an „at-predicate“ for cargo being at an airport and an „in-predicate“, when it is in a plane.

Algorithms for Classical Planning (1)

- Forward state-space search: Each instantiation (unification) of the variables in the precondition of an action with constants in the state yields one ground action
  - ➢ huge branching factor
- Backward search: relevant action schemes are those, whose action part unifies with the goal state and the unification must be propagated to the preconditions of the action schemes
  - ➢ usually much smaller branching factor



Algorithms for Classical Planning (2)

- **Planning as Boolean satisfiability (SATPlan):** Translate PDDL problem description into propositional form and use efficient SAT problem solver (e.g. WalkSAT).
  - Propositionalize the actions: Replace the variables with constants, e.g. Unload (c, p, a), generate all combinations of cargos, planes and airports. For each combination, the different time steps must be generated (e.g. as supersripts)!
  - Add action exclusion axioms, saying that no two actions can occur at the same time, e.g. $\neg(\text{FlyP}_1\text{SFOJFK}^1 \wedge \text{FlyP}_1\text{SFOBUH}^1)$
  - Add precondition axioms (when the propositionalized actions can be executed)
  - Define the initial state (with time step 0)
  - Propositionalize the goal (similar to the actions, where the goal is a large disjunction)
  - Add successor-state axioms (to come from one time step to the next)
- Although it seems unplausible, the resulting very large translation can often be solved with modern SAT problem solvers.

Other Classical Planning Approaches

- They resemble more the way humans would plan:
  - Try to avoid too much search
  - However, are currently not competive
  - Might be useful, if humans need to understand and check the plan (e.g. for Mars rovers)
- Main approaches:
  - Planning graph: Generates constraints from solving a simplified planning problem
  - Situation calculus: Describe and solve planning problems in first-order logic
  - Encode problem as constraint satisfaction problem (similar to the SAT-problem)
  - Partial-order planning: While the approaches above construct totally ordered plans, partial-order planning takes advantage if there are inpependant subproblems
    - Construct plans for the subproblems and connect them using constraints about the ordering

Heuristics for Planning (1)

- Neither forward nor backward search is efficient without a good heuristic function
- Requirements:
  - Heuristics should be derived automatically
  - Heuristics should be admissable (i.e. they underestimate the costs for reaching the goal)
- Main ideas:
  - Derive admissable heuristics from a relaxed problem
    - Ignore-precondition heuristic
    - Drop selected preconditions
    - Ignore-delete-list heuristic
  - Symmetry reduction
  - State abstraction

Heuristics for Planning (2)

- **Ignore-precondition heuristic:** Drops all preconditions from actions
  - Number of steps required is similar to the number of unsatisfied goals
    - But: some actions may achieve multiple goals
    - But: Some actions may undo the effect of others
  - If ignoring the second exception, we count the minimum number of actions (without preconditions) reaching the goal by using a set-covering algorithm.
    - Although set covering in itself is a NP-hard problem, a simple greedy algorithm might overestimate the length with a factor of log (n), with n being the number of literals in the goal.
    - The heuristic might produces slightly longer plans compared to the optimal plan

Heuristics for Planning (3)

- **Drop selected preconditions** of actions like in the 8-puzzle, which can be encoded as planning problem with a single action scheme „Slide“:

  Action: (Slide (t, s1, s2),

  PRECOND: On (t, s_1) $\wedge$ Tile (t) $\wedge$ Blank (s_2) $\wedge$ Adjacent (s_1, s_2)

  Effect: On (t, s_2) $\wedge$ Blank (s_1) $\wedge$ $\neg$On(t, s_1) $\wedge$ $\neg$Blank (s_2))

  - If we remove the preconditions Blank (s_2) $\wedge$ Adjacent (s_1, s_2), than a tile can move in one action to any space
  - If we remove only the preconditions Blank (s_2), than we get a more accurate heuristic (Manhattan-distance heuristic)
- Such heuristics can be derived automatically from the factored problem description!

Heuristics for Planning (4)

- **Ignore-delete-list heuristic**:
  - Assumption: All goals and preconditions contain only positive literals
    - If the assumption is not valid, the problem can be rewritten accordingly.
  - If we remove all negative literals from effects, we can make monotonic progress towards the goal (and take the length of this easier problem as heuristic for the original problem)
    - Use an efficient hill-climbing algorithm to solve the easier problem (again, it might be not optimal as required for admissability, but the error usually is small)

Domain-independant Pruning

- **Symmetry reduction:**
  - Often many similar actions exist, from which only one needs to be considered
    - Example: Blocks World: Place Block A on top of a three block tower
      - Place any block y on top of a block x on the table and A on top of y
  - General: Prune out consideration of symmetric branches of the search tree except one
- **Preferred action:**
  - If we have a solution to a relaxed problem, we can choose the actions of the relaxed problem as preferred action (attention: might be suboptimal)
- **State abstraction:**
  - Ignore some fluents in state description, which are not critical for achieving the goal
- **Pattern database:** Store useful patterns of plans (endgame database in chess)

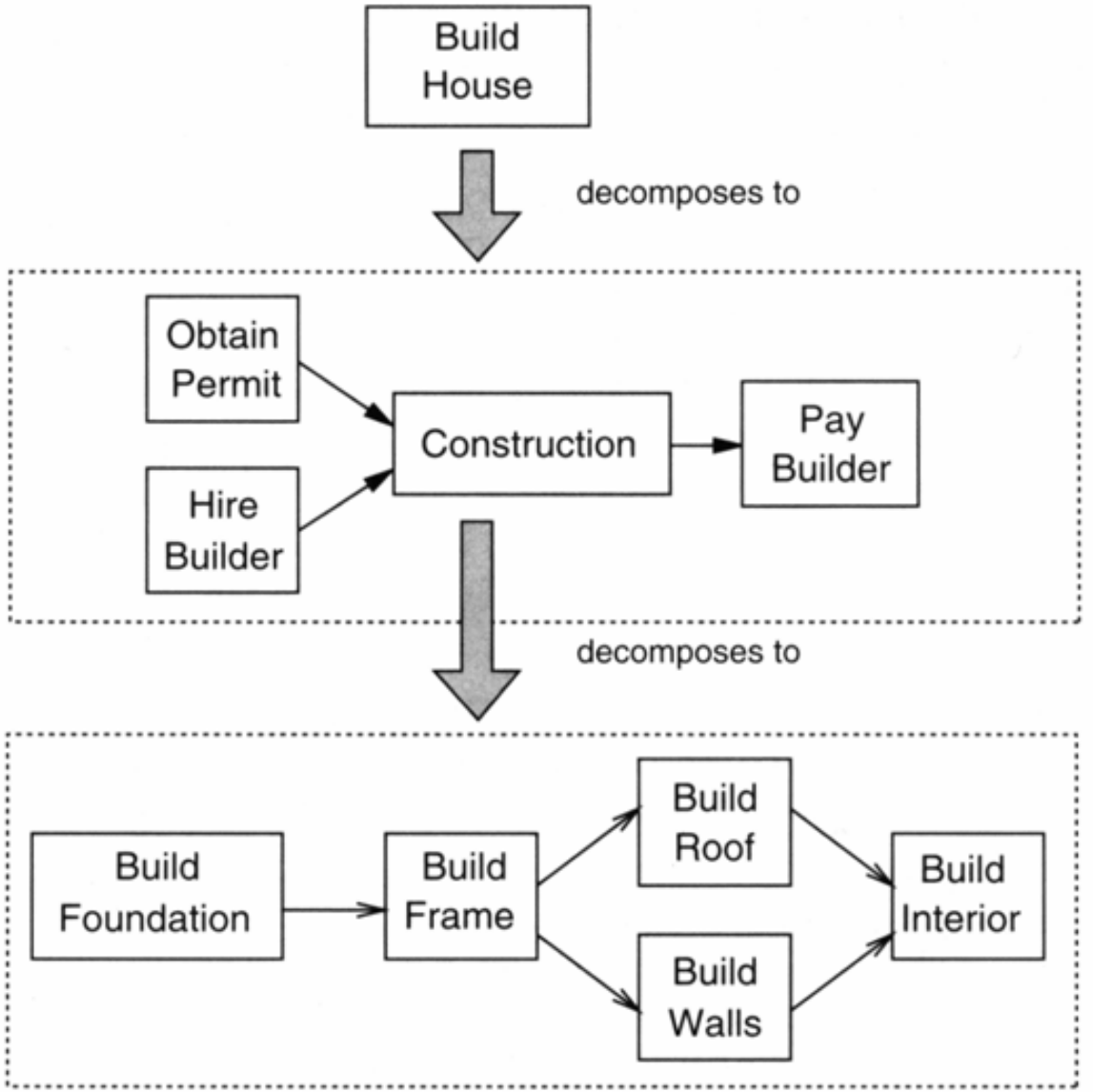
Decomposition Ideas

- **Serializable subgoals**:
  - Def: Order of subgoals, where the achievement of one subgoal does not undo previously achieved subgoals
    - Example: Blocks World: Building a special tower of blocks
- **Subgoal independence assumption**:
  - Assumption: The cost of solving a conjunction of subgoals is approximated by the sum of solving each subgoal independently
    - Is usually not an admissable heuristic (may estimate costs too high, if actions for two subgoals are the same)

Hierarchical Planning

Plan first on a high level and then refine high-level actions (recursively),

- Example: Build House (right)
- Key concepts:
  - Distinction between hig-level actions (HLA) and primitive actions



Hierarchical Planning (2)

- Example: Planning a trip
- Plan with HLAs
  - Go (Home, San Francisco)
  - Fly (San Francisco, Frankfurt)
  - Go (Frankfurt, Rothenburg o.d.T)
  - Visit Rothenburg o.d.T
  - Go (Rothenburg, o.d.T Frankfurt)
  - Fly (Frankfurt, San Francisco)
  - Go (San Francisco, Home)
- Refinements:
  - Go (Home, San Francisco): see right
  - Navigate ([a,b], [x,y]): primitive action; see right

Refinement(*Go*(*Home*, *SFO*),
  STEPS: [*Drive*(*Home*, *SFOLongTermParking*),
    Shuttle(*SFOLongTermParking*, *SFO*)])
Refinement(*Go*(*Home*, *SFO*),
  STEPS: [*Taxi*(*Home*, *SFO*)])

Refinement(*Navigate*([*a*, *b*], [*x*, *y*]),
  PRECOND: $a = x \wedge b = y$
  STEPS: [])
Refinement(*Navigate*([*a*, *b*], [*x*, *y*]),
  PRECOND: *Connected*([*a*, *b*], [*a* − 1, *b*])
  STEPS: [*Left*, *Navigate*([*a* − 1, *b*], [*x*, *y*])])
Refinement(*Navigate*([*a*, *b*], [*x*, *y*]),
  PRECOND: *Connected*([*a*, *b*], [*a* + 1, *b*])
  STEPS: [*Right*, *Navigate*([*a* + 1, *b*], [*x*, *y*])])
...

Hierarchical Planning (3)

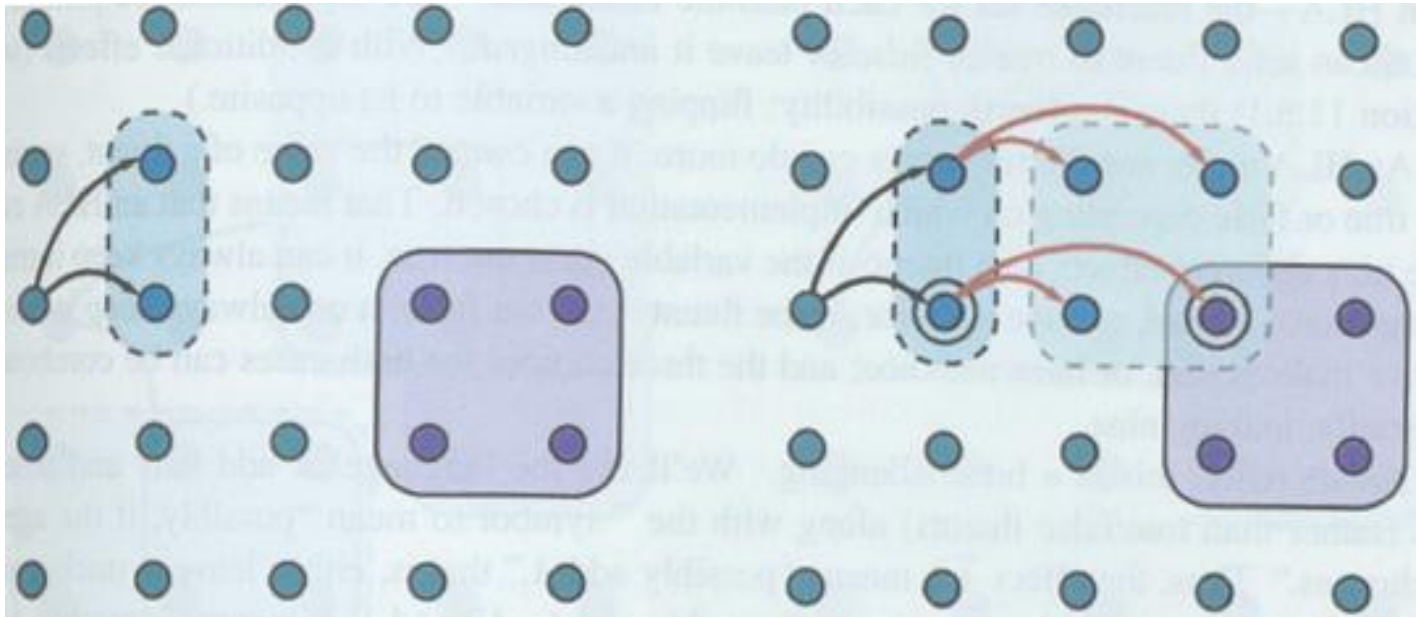
- Hierarchical Planning can be viewed as solving an AND-OR-Tree
  - For a goal (HLA), there can be various solutions (plans), i.e. choosing one is sufficient (OR)
  - The refinement of a HLA usually involves several plan steps (AND)
  - Finally, all actions must be primitive
- Standard search algorithms can be applied
  - AND-OR-Search
  - Breadth-first and depth-first (with a criteria, when the goal is reached)
- There are huge savings compared to non-hierarchical planning
  - Non-hierarchical: Final plan with d actions and b allowable actions in each state: $O(b^d)$
  - Hierarchical: If each non-primitive action has r possible refinements, each into k actions at the next level: $O(r^{(d-1)/(k-1)})$

Hierarchical Planning (4)

- The key to hierarchical planning is a plan library containing known methods for implementing complex, high level actions.
- Can be incrementally constructed by generalizing plans.

Hierarchical Planning (5)

- Up to now, we still consider all primitive plans based on stepwise refinement with hierarchical planning
- It would be much more efficient, if we can check the feasability of a high level plan without the need of refinement.
  - **Downward refinement property** of HLA descriptions: At least one implementation of a HLA must fulfil the goal
    - Easy to prove, if there is only one refinement
    - More difficult, if there are several refinements;
      - **Reachable set** (at least one refinement achieves the goal)
        - **Angelic semantics:** One can choose the best refinement



Hierarchical Planning (6)

- Reachable set is not sufficient:
  - The combination of many refinements might not reach the goal, although the goal was in the reachable set.
  - Differentiation necessary:
    - Optimistic description $REACH^{+}$ (state, hla): „Union“ of all reachable sets
    - Pessimistic description $REACH^{-}$ (state, hla): „Intersection“ of all reachable sets
    - If optimistic description include goal, but pessimistic does not, there is uncertainty
      - ➢ Refine that part of the plan to reduce the uncertainty
      - ➢ Similar to human planning, which strongly depends on hierachical planning and only refines critical parts of the plan
      - ➢ Very efficient

Planning and Acting in Nondeterministic Domains

- Agent does not know, in what state is it
  - Needs some kind of belief state
    - Belief state based on a factored representation
- Example painting problem:
  - A chair and a table should have the same color. The agent has cans with colors and can perform the actions open (can) and paint (x, can). But it cannot percept the color of the chair, table or the cans.
- Three kinds of Tasks:
  - **Sensorloss planning** (conformant planning) for environments with no observations
  - **Contingency planninng** for partially observable and nonterministic environments
  - **Online planning** and **replanning** for unknown environments

Sensorless Planning

- The agent knows some facts about the world, which does not change
- The agent has no percepts
  - Therefore the closed world assumption cannot be used, where everything is false, what the agent does not know. Instead the open world assumption is necessary
  - Nevertheless, the agent can solve problems
  - e.g. in the painting problem:
    - Object (table) $\wedge$ Object (Chair) $\wedge$ Can (C_1) $\wedge$ Can (C_2)
    - Objects have colors (although the agent has no percepts for them): $\forall$ x $\exists$ c Color (x,c)
      - Normalized as: b_0 = Color (x, C(x))
    - Solution: Open (Can_1), Paint (Chair, Can_1), Paint (Table, Can_1)

Sensorless Planning: Belief State

- The initial belief state is a conjunction of literals (1-CNF formula).
- The belief state is updated by the add and delete list of actions simply by adding or deleting literals
  - Unknown literals not mentioned in the action remain unknown and are not mentioned in the belief state.
- Example: Open (Can_1), Paint (Chair, Can_1), Paint (Table, Can_1)
  - Let [Object (table) $\wedge$ Object (Chair) $\wedge$ Can (C_1) $\wedge$ Can (C_2)] = Context
  - Intial belief state b_0: Context $\wedge$ Color (x, C(x))
  - b_1 after Open (Can_1): Context $\wedge$ Color (x, C(x)) $\wedge$ Open (Can_1)
  - b_2 after Paint (Chair, Can_1): Context $\wedge$ Color (x, C(x)) $\wedge$ Open (Can_1) $\wedge$ Color (Chair, C (Can_1))
  - b_3 after Paint (Table, Can_1): Context $\wedge$ Color (x, C(x)) $\wedge$ Open (Can_1) $\wedge$ Color (Chair, C (Can_1)) $\wedge$ Color (Table, C (Can_1))
- Belief state starts as 1-CNF and remains 1-CNF
  - Valid only, if the effect of actions does not depend on the state

Conditional effects

- The effect of actions depends on the state
  - e.g. in vacuum world Action „Suck“ depends on current location (AtL = Agent in square L):
    - Action Suck, EFFECT: when AtL: CleanL $\wedge$ when AtR: CleanR
    - Or
      - Action SuckL: PRECOND: AtL; EFFECT: CleanL
      - Action SuckR: PRECOND: AtR; EFFECT: CleanR
- Conditional effects or unconditional effects with preconditions, which cannot be checked would result in huge belief state
- Solution: Remain 1-CNF belief state and perform actions to reduce uncertainty:
  - e.g. for cleaning two squares R and L in sensorless vacuum world: [Right, Suck, Left, Suck]
  - Belief states at start (b_0) and after each of the four action action (b_1 – b_4):
    - b_0 = true, b_1 = AtR; b_2 = AtR $\wedge$ CleanR; b_3 = AtL $\wedge$ CleanR; b_4 = AtL $\wedge$ CleanR $\wedge$ CleanL

Possible Search Approaches to Sensorless Planning

- Start with a Belief set is in 1-CNF (i.e. conjunction of literals)
- Choose a possible sequence of actions with A* search and a heuristic function based on subsets of the belief state and simplifications of the actions
- Prove, whether an action sequence reaches the goal with SATPlan

Contingent Planning

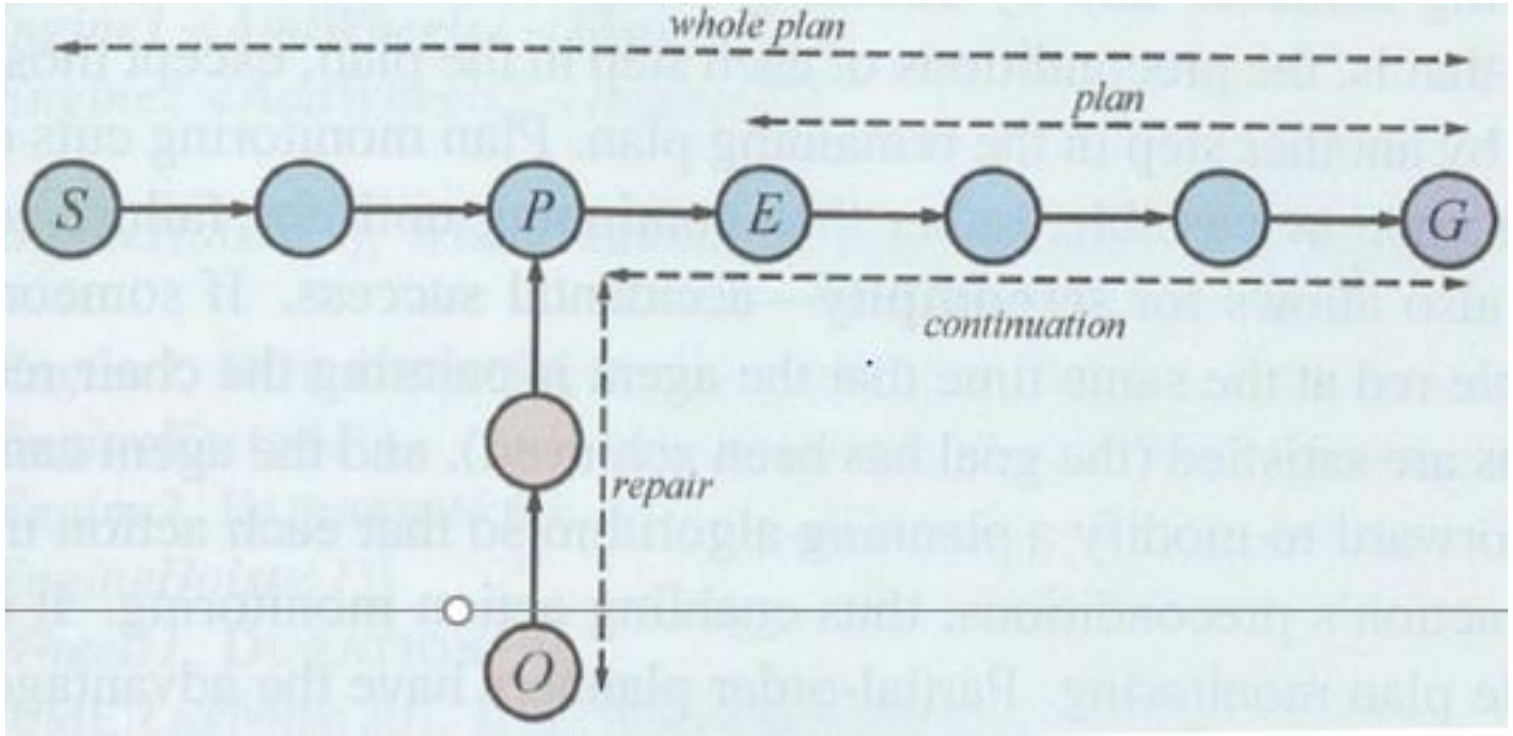
- Idea: Conditional branching based on percepts to deal with partial observability or non-determinism
- Example: Solution for Painting world:

$$
\begin{array}{l}
[LookAt(Table), LookAt(Chair),\\
\quad \textbf{if } Color(Table, c) \wedge Color(Chair, c) \textbf{ then } NoOp\\
\quad\quad \textbf{else } [RemoveLid(Can_1), LookAt(Can_1), RemoveLid(Can_2), LookAt(Can_2),\\
\quad\quad\quad \textbf{if } Color(Table, c) \wedge Color(can, c) \textbf{ then } Paint(Chair, can)\\
\quad\quad\quad \textbf{else if } Color(Chair, c) \wedge Color(can, c) \textbf{ then } Paint(Table, can)\\
\quad\quad\quad \textbf{else } [Paint(Chair, Can_1), Paint(Table, Can_1)]]]
\end{array}
$$

- Plan generation by an AND-OR forward search over belief states

Online Planning (1)

- Checking during plan execution, whether preconditions of plan steps are still valid.
  - If not, replanning
- Three levels of monitoring before executing an action:
  - Action monitoring: Checking the preconditions of the current plan step (action)
  - Plan monitoring: Checking the remaining plan, whether it still will reach the goal
  - Goal monitoring: Checking, if there are better goals are better paths to the goal
- Example:
  - The agent executes the whole plan from S to G until it expects to be in state E, but observes that it is actually in state O
  - The agent replans for a minimal repair.



Online Planning (2)

- Trade off between amount of monitoring and plan execution
  - Monitoring everything can be quite inefficient
- Plan monitoring allows for serendipity, i.e. accidental access
  - e.g. if someone else already has solved the problem, i.e. the goal is reached.
- With faulty models, replanning might lead in dead ends
  - This is true also, if a seemingly non-deterministic action is not actually random, but depends on preconditions unknown to the agent.
  - Learning necessary

Times, Schedules and Resources

- Till now: Classical planning about what to do in what order, but without time constraints
  - **Scheduling**
- Standard approach: Plan first, schedule later.
  - Planning: Select actions with ordering constraints
  - Scheduling: Add temporal information so that plan meets resource & deadline constraints
- **Job-shop scheduling problem**:
  - Set of **jobs**
  - Each job has a collection of **actions** with ordering constraints among them
  - Each action has a **duration** and a set of **resource constraints**
    - Resources are either **consumable** or **reusable**
- Solution to job-shop scheduling problem:
  - Specifies start time for each action satisfying ordering and resource constraints
  - Cost function (opimization criteria): total duration of plan (**makespan**)

Example for Job-Shop Scheduling Problem

$$Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},$$
$$\{AddEngine2 \prec AddWheels2 \prec Inspect2\})$$

$$Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))$$

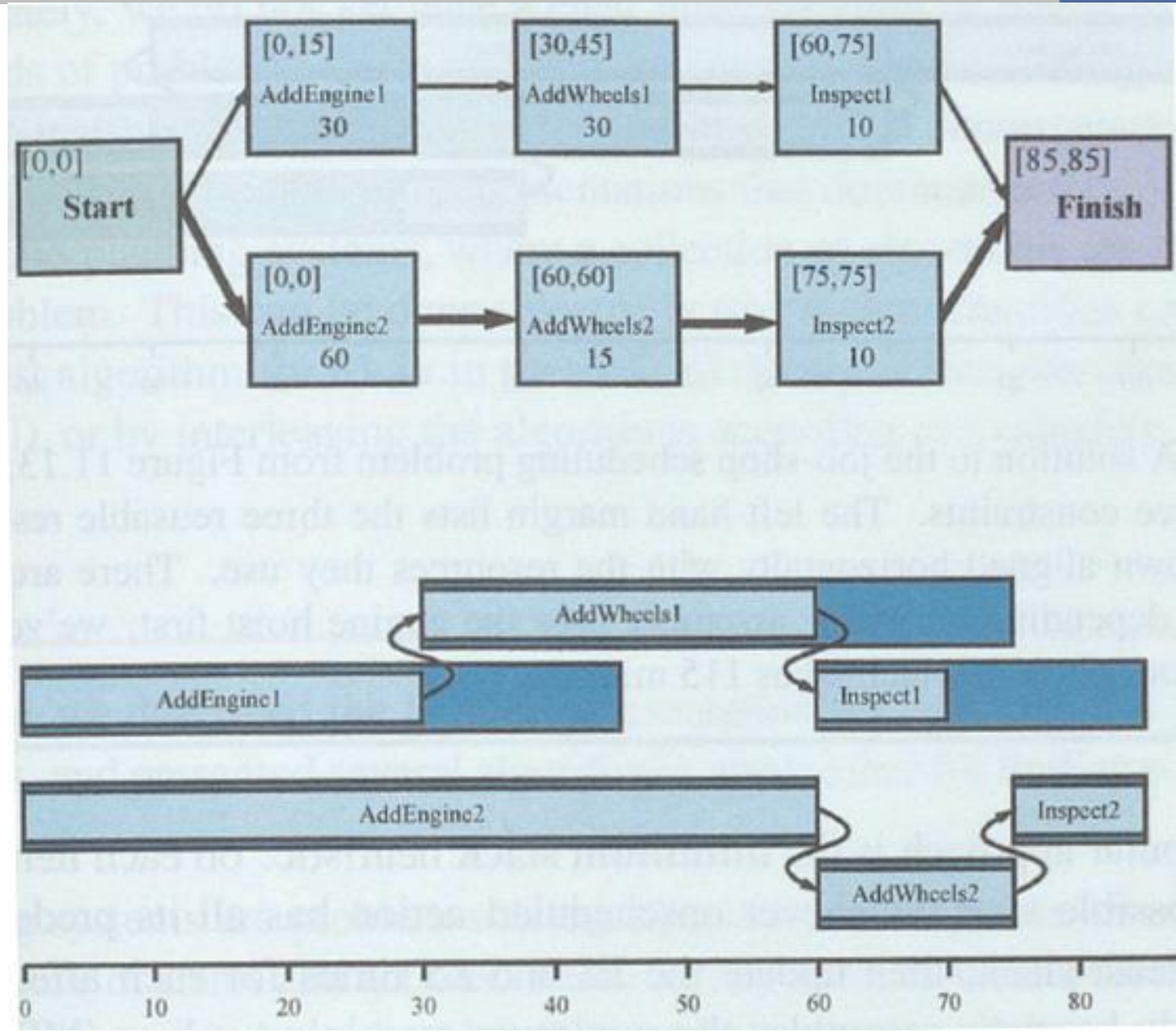
$$Action(AddEngine1, \text{DURATION}:30,$$
$$\text{USE}:EngineHoists(1))$$
$$Action(AddEngine2, \text{DURATION}:60,$$
$$\text{USE}:EngineHoists(1))$$
$$Action(AddWheels1, \text{DURATION}:30,$$
$$\text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$$
$$Action(AddWheels2, \text{DURATION}:15,$$
$$\text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$$
$$Action(Inspect_i, \text{DURATION}:10,$$
$$\text{USE}:Inspectors(1))$$

Algorithm for Job-Shop Scheduling Problems

Critical path method (CPM):

- Critical path is path in partial order plan whose duration is longest
  - ➢ Delaying any action on critical path increases total duration (critical path with bold arrows)
  - ➢ Other action have a time window in which they can excecuted
    - Time window is specified by earliest and latest possible start time (ES and LS), e.g. for „AddEngine1 [0, 15]
    - **Slack:** Difference between ES and LS (time margin of action)

Example right without ressource constraints



Algorithm for Computing Time Windows of Actions

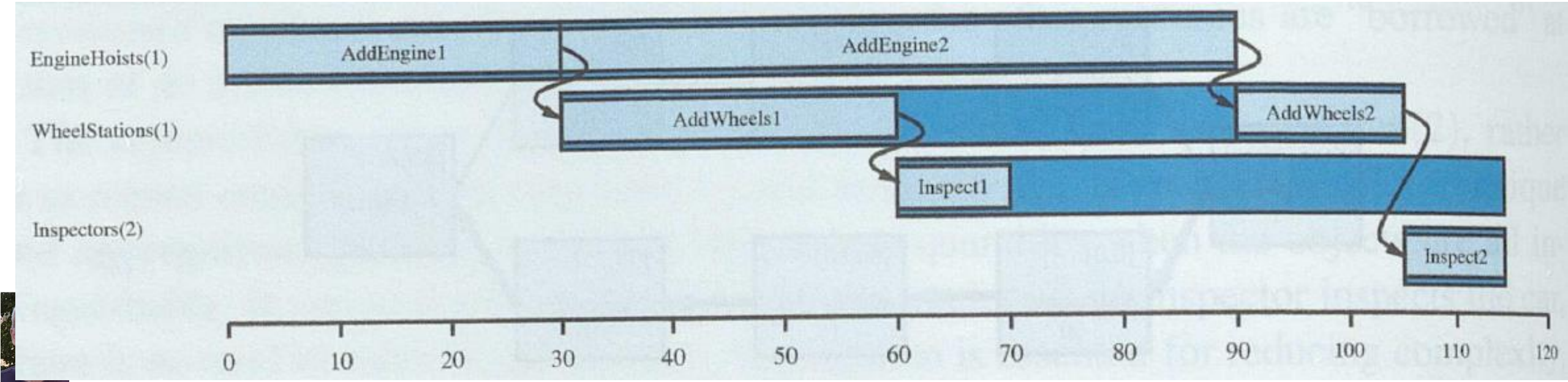
- Dynamic programming algorithm
  - A and B are actions, „A $\lceil$ B“ means that A precedes B

  - Compute ES and LS of each action seperately in a similar manner.
  - ES of action B can be computed, if all actions A coming immediately before B have an ES
  - ES (Start) = 0
  - ES (B) = $\max_{A \lceil B}$ ES(A) + Duration (A): Search latest beginning predecessor of B named A

  - LS(Finish) = ES(Finish)
  - LS(A) = $\min_{B \rfloor A}$ LS(B) – Duration(B): Search latest ending successor of A named B
- Complexity: O(Nb)
  - with N = number of actions and b = maximal branching factor into or out of an action

Introducing Resource Constraints

- Constraints in example: Only one engine station and one wheel station, i.e. engines cannot be done in parallel (as well as wheels).
- Choosing one engine before the other necessary
  - Equivalent to a disjunction and increases complexity to NP-hard
  - Popular greedy approach: **minimum slack heuristic**: On each iteration, schedule for the earliest possible start whichever unscheduled action has all its predecessors scheduled and has the least slack.
    - Would choose „AddEngine2“ in example below at first; resulting in a supoptimal plan



Planning Reflected

- Planning combines two major areas of AI: Search and logic
- Cross-fertilization of ideas resulted in real-world applications with millions of states and thousands of actions
- Main problem: Controlling combinatorial explosion
  - Identification of independant subproblems
  - Different powerful algorithms
    - SATPlan
    - Forward search with strong heuristics
  - Portfolio of planning systems: Apply a collection of algorithms to find the best solution