academic year: 2022-2023

Artificial Intelligence

Monica Mordonini

monica.mordonini@unipr.it

Automated Planning

Classical Planning

Problem solving

- **Classical Planning**
  - task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment
- We have seen
  - Search-based problem solver
  - Logical planner

- Both share two limitations.
- First, they both require ad hoc heuristics for each new domain: a heuristic evaluation function for search, and hand-written code for the agent.
- Second, they both need to explicitly represent an exponentially large state space

Characteristics of Real Problems

- They are complex:
  - Using the search method in the statespace or the logical planner without using another actor is not conceivable: the planner
  - The planner deals with the **scalability** of planning techniques

- But it requires methods for:
  - break the problem down.
  - apply the operators based on the part of the state of the problem that concerns them (frame problem).
  - to deal with incomplete knowledge and with the evolution not completely controllable and predictable of the state.

Characteristics of Real Problems

- How these problems can be addressed?
  - Considering all the possible plans.
  - Selecting a plan with a high probability of success.
- If you perform a plan action and the action fails?
  - Reschedule from the state in which the action failed.
  - Make local changes and keep the rest of the plan.

Characteristics of Real Problems

- Language of planning problems (PDDL family - Planning Domain Definition Language)
  - Representation of states
    - The world is broken down into logical conditions and a state is given as a conjunction of positive literals
    - (es, poor & unknown & sad)
  - Representation of the objectives
    - A partially specified state that must be satisfied by a state
    - (poor & unknown goal achieved in the poor & unknown & sad state)
  - Representation of actions
    - Specified by means of the preconditions that must apply before its execution and the effects that arise from it

Basic Functions of the Planners

- ✓ Choose the best operator to produce the action that will bring the agent closer to the target.
- ✓ Apply an operator and calculate the new status.
- ✓ Identify when the solution was found.
- ✓ Identify the blind alleys.
- ✓ Identify when the solution is almost correct and apply special techniques to make it correct.

Apply an Operator and Calculate the New State

- The representation of planning problems (objective action states) should allow algorithms to take advantage of the logical structure of the problem

- The key lies in finding a ***sufficiently expressive language*** to describe a great variety of problems, but <u>rather narrow enough to be used by efficient algorithms</u>

Apply an Operator and Calculate the New State

- STRIPS (STandford Research Institute Problem Solver)
  - Representation language of classical planners
- Representation of states
  - The world is broken up into logical actions
  - ✓ Closed world hypotheses (all conditions not mentioned in a state will be considered false)
- Representation of the objectives
  - Objective: partially specified statuso
  - ✓ A state s satisfies an objective if it contains all the conditions foreseen in the objective (possibly also others)
- Representation of actions

Apply an Operator and Calculate the New State

- STRIPS (STandford Research Institute Problem Solver)
  - Representation language of classical planners
- Representation of actions
  - Scheme of action: an action is specified by means of the preconditions that must apply before its execution and the effects that result
  - ✓ An action is applicable in any state that satisfies the precondition
  - ✓ STRIPS statement: literals not mentioned in the effect are never modified

Apply an Operator and Calculate the New State

- STRIPS (STandford Research Institute Problem Solver)
  - Representation language of classical planners

- *Solution of a planning problem*
  - It is constituted in its simplest form by a sequence of actions that, once performed, will lead to a state that satisfies the objective

Apply an Operator and Calculate the New State

An action schema for flying a plane from one location to another:

Action(Fly(p, from, to),

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

EFFECT:¬At(p, from)∧At(p, to))

- A set of action schemas serves as a definition of a planning domain.
- A specific problem within the domain is defined with the addition of an initial state and a goal

The block world

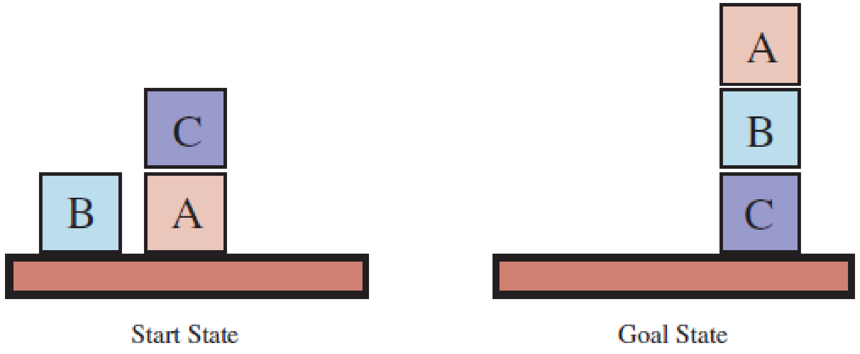


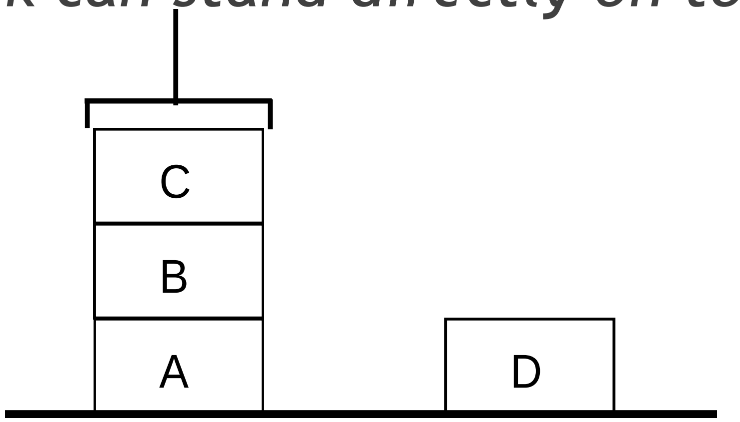
Figure 11.3 Diagram of the blocks-world problem in Figure 11.4.

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

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C,A), Move(B,Table,C), Move(A,Table,B)]$.

Example: The World of Blocks

Domain: set of cubic blocks. They can be stacked but only one block can stand directly on top of another.



- The state of the world is described by a set of predicates.
- The world can only be changed by the arm.

The World of Blocks in Strips

- The **predicates** are of the type:
  - on(B,A) ontable(A)
  - clear(D) armempty
- The **actions** are of the type:
  - unstack(C,B) stack(C,D)
  - pickup(D) putdown(D)

Apply an Operator and Calculate the New State

- Operators at STRIPS
  unstack(x,y)

  - Precondition: on(x,y) ∧ clear(x) ∧ armempty
  - Effect: on(x,y) ∧ armempty
  - Action: holding(x) ∧ clear(y)

Apply an Operator and Calculate the New State

- The description of a planning problem provides an obvious way to search from the initial state through the space of states, looking for a goal.

- A nice advantage of the declarative representation of action schemas is that we can also search backward from the goal, looking for the initial state.

- A third possibility is to translate the problem description into a set of logic sentences, to which we can apply a logical inference algorithm to find a solution.

Example domain: Air cargo transport

- The problem can be defined with three actions: Load, Unload, and Fly.
- The actions affect two predicates: In(c, p) means that cargo c is inside plane p, and At(x,a) means that object x (either plane or cargo) is at airport a.

$$
\begin{array}{l}
Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
\quad \land Airport(JFK) \land Airport(SFO)) \\
Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
Action(Load(c, p, a), \\
\quad \textsc{Precond}: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
\quad \textsc{Effect}: \neg At(c, a) \land In(c, p)) \\
Action(Unload(c, p, a), \\
\quad \textsc{Precond}: In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
\quad \textsc{Effect}: At(c, a) \land \neg In(c, p)) \\
Action(Fly(p, from, to), \\
\quad \textsc{Precond}: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
\quad \textsc{Effect}: \neg At(p, from) \land At(p, to))
\end{array}
$$

Figure 11.1 A PDDL description of an air cargo transportation planning problem.

Example domain: Air cargo transport –Search a solution

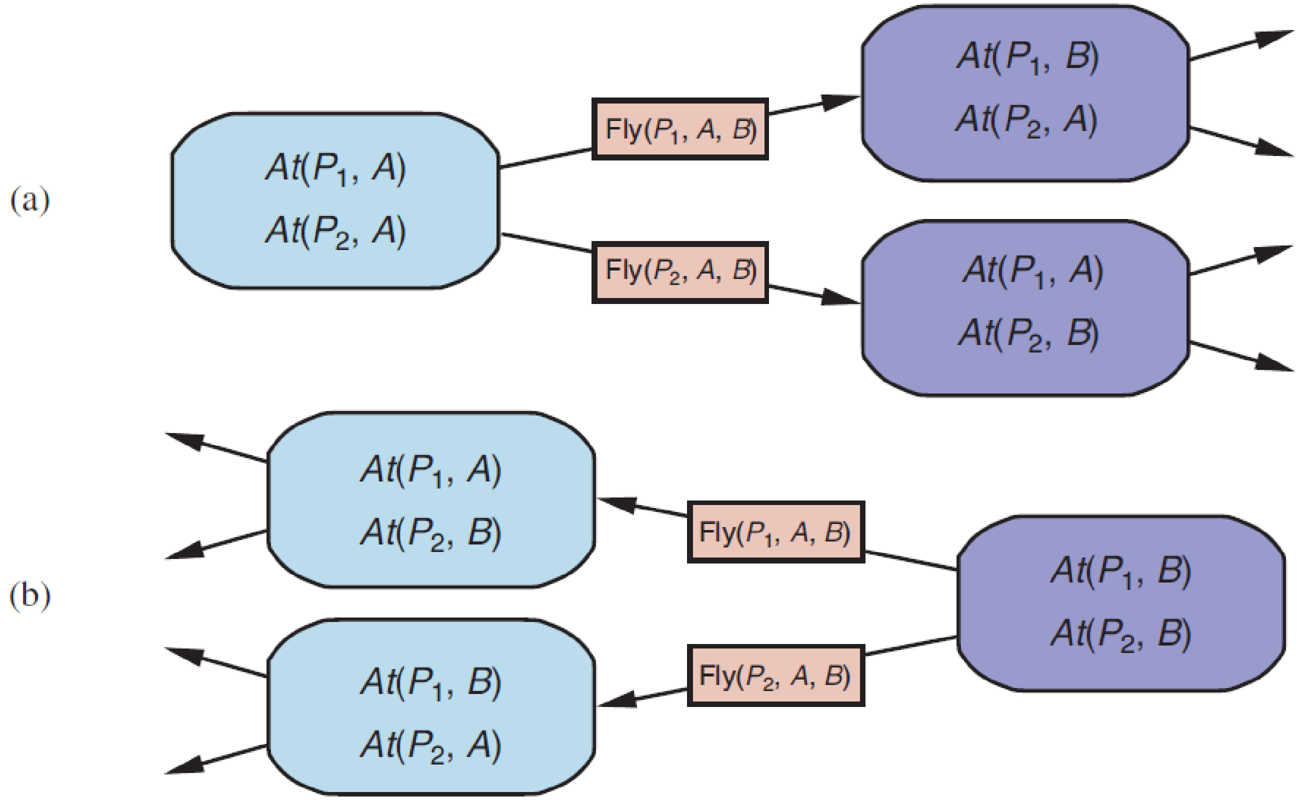


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) search through the space of ground states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through state descriptions, starting at the goal and using the inverse of the actions to search backward for the initial state.

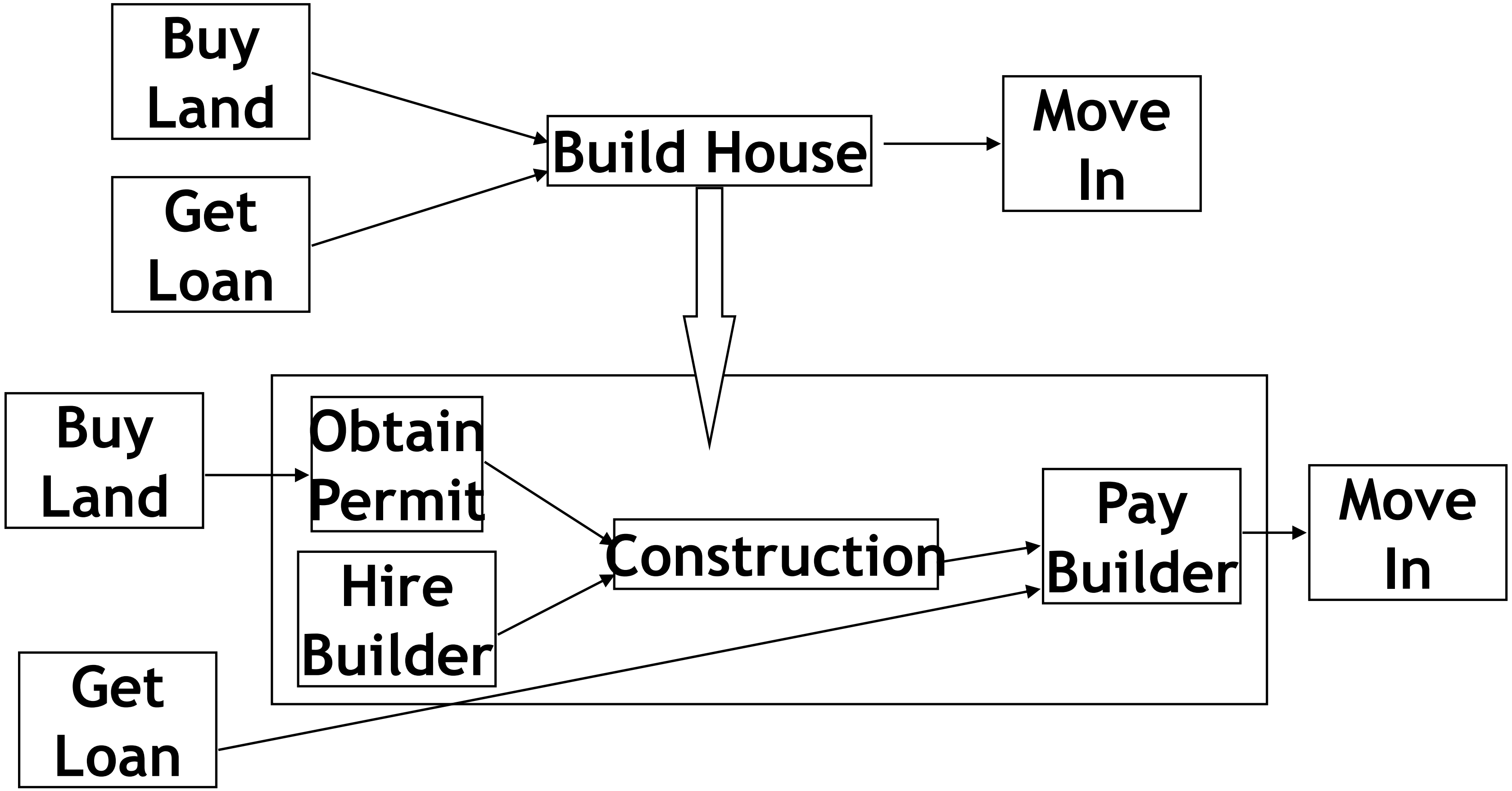
Hierarchical Planning

- There are more complex problems than the example in the world of blocks.
- There are problems that are given by the composition of several objectives (goals)
- **Hierarchical Planning** tries to tackle the problem of the ***different importance of goals.***
- It uses abstract operators that are decomposed into plans that implement the operator.
- We have a planning at different levels of abstraction.
- Each level of hierarchy is reduced to a small number of activities at the level immediately below, so that the cost of finding the best way to organize this activity is equally small

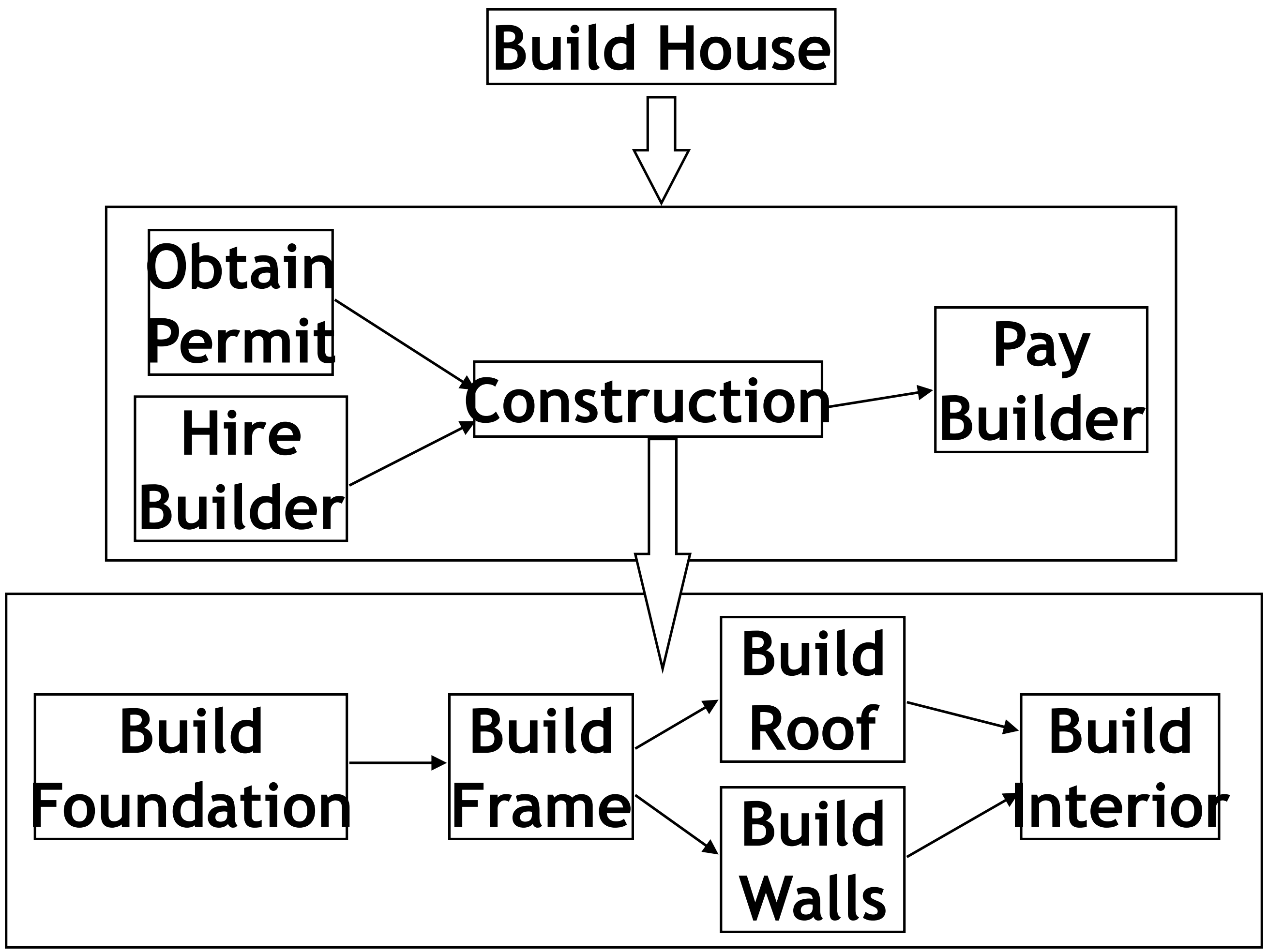
Hierarchical Planning

- A plan p correctly implements an operator o if:
  - p is a consistent plan.
  - Each effect of o must be achieved by a step of p.
  - Each precondition of a step of p must either be attained by another step of p or be a precondition of o.

Hierarchical Planning



Hierarchical Planning



Planning in the Real World

- This type of planning techniques are applicable to domains in which:
  - Each entity is known.
  - The properties of each entity are static and deterministic.
- In the real world they are not enough because:
  - There is an incomplete knowledge of the world.
  - You have a wrong knowledge of the world.
- To manage these situations you can use two different planning techniques:
  - Conditional planning.
  - Execution monitoring (execution monitoring).

Conditional Planning

- If the environment is not deterministic, the agent can not completely predict the result of his actions

- A conditional planning agent manages non-determinism by including at the time of construction of the conditional steps plan in which he will verify, during execution, the state of the environment

- Conditional planning therefore introduces sensorial actions to decide between sequences of alternative actions.

- The problem is therefore reduced to the construction of conditional plans

Conditional Planning

- Example: checking and changing a punctured wheel
- Target:
  - have a wheel mounted and inflated
- Actions:
  - fit a wheel,
  - remove a wheel,
  - inflate a wheel
- Initial state:
  - a wheel mounted but deflated
  - a spare wheel intact and swollen but in the trunk

Conditional Planning

- For example, if we have:
  Goal: $On(x) \wedge Inflated(x)$
  Operators: $Remove(x)$, $PutOn(x)$, $Inflate(x)$
  Init State: $Inflated(Spare) \wedge Intact(Spare)$
  $\wedge\ Off(Spare)$
  $\wedge\ On(Tire_1) \wedge Flat(Tire_1)$

- A previous planner will dial plan:
  [$Remove(Tire_1)$, $PutOn(Spare)$]

- Instead a conditional planner:
  if ($Intact(Tire_1)$) [$Inflate(Tire_1)$],
  [$Remove(Tire_1)$, $PutOnt(Spare)$]

Execution Control

- The control of the execution allows to reschedule when unexpected situations are encountered in the model of the world on which the plan has been defined.

- The operation of such a system can be described by the following steps:
  - As long as the preconditions of the next action of the plan are satisfied in the current state s, the action is executed.
  - So you look for a point p’ of the plane that is easily reachable from the current state s.
  - A plan from s to p’ is generated.

Conditional Planning Vs Execution Control

- Every action of a plan has the possibility of a malfunction and one can have a wrong knowledge of every state of the world.
- The number of conditions that must be introduced increases exponentially with the number of steps in the plan.
- At each step of the plan it may be necessary to reschedule (continuous planning).
- An acceptable solution is to integrate the two approaches by introducing conditions:
  - For events that have a high probability.
  - For events that despite having a low probability, are catastrophic.

Time, Schedules, and Resources

- Classical planning talks about what to do, in what order, but does not talk about time: how long an action takes and when it occurs.

- For example, in the airport domain we could produce a plan saying what planes go where, carrying what, but could not specify departure and arrival times.

- This is the subject matter of **scheduling.**

- The real world also imposes **resource constraints**: an airline has a limited number of staff, and staff who are on one flight cannot be on another at the same time

Time, Schedules, and Resources

- **"plan first, schedule later"**

- divide the overall problem into a planning phase in which actions are selected, with some ordering constraints, to meet the goals of the problem, and a later scheduling phase, in which temporal information is added to the plan to ensure that it meets resource and deadline constraints.

- This approach is common in real-world manufacturing and logistical settings, where the planning phase is sometimes automated, and sometimes performed by human experts.

Time, Schedules, and Resources

- **“job-shop scheduling problem”**

- a set of jobs, each of which has a collection of actions with ordering constraints among them.

- Each action has a duration and a set of resource constraints required by the action.

- A constraint specifies a type of resource (e.g., bolts, wrenches, or pilots), the number of that resource required, and whether that resource is consumable (e.g., the bolts are no longer available for use) or reusable (e.g., a pilot is occupied during a flight but is available again when the flight is over).

Time, Schedules, and Resources

$$
\begin{aligned}
&Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},\\
&\quad \{AddEngine2 \prec AddWheels2 \prec Inspect2\})
\end{aligned}
$$

$$Resources(EngineHoists(1),\ WheelStations(1),\ Inspectors(e2),\ LugNuts(500))$$

$$
\begin{aligned}
&Action(AddEngine1,\ \textsc{Duration}{:}30,\\
&\quad \textsc{Use}{:}EngineHoists(1))\\
&Action(AddEngine2,\ \textsc{Duration}{:}60,\\
&\quad \textsc{Use}{:}EngineHoists(1))\\
&Action(AddWheels1,\ \textsc{Duration}{:}30,\\
&\quad \textsc{Consume}{:}LugNuts(20),\ \textsc{Use}{:}WheelStations(1))\\
&Action(AddWheels2,\ \textsc{Duration}{:}15,\\
&\quad \textsc{Consume}{:}LugNuts(20),\ \textsc{Use}{:}WheelStations(1))\\
&Action(Inspect_i,\ \textsc{Duration}{:}10,\\
&\quad \textsc{Use}{:}Inspectors(1))
\end{aligned}
$$

Figure 11.13 A job-shop scheduling problem for assembling two cars, with resource constraints. The notation $A \prec B$ means that action A must precede action B.

Time, Schedules, and Resources

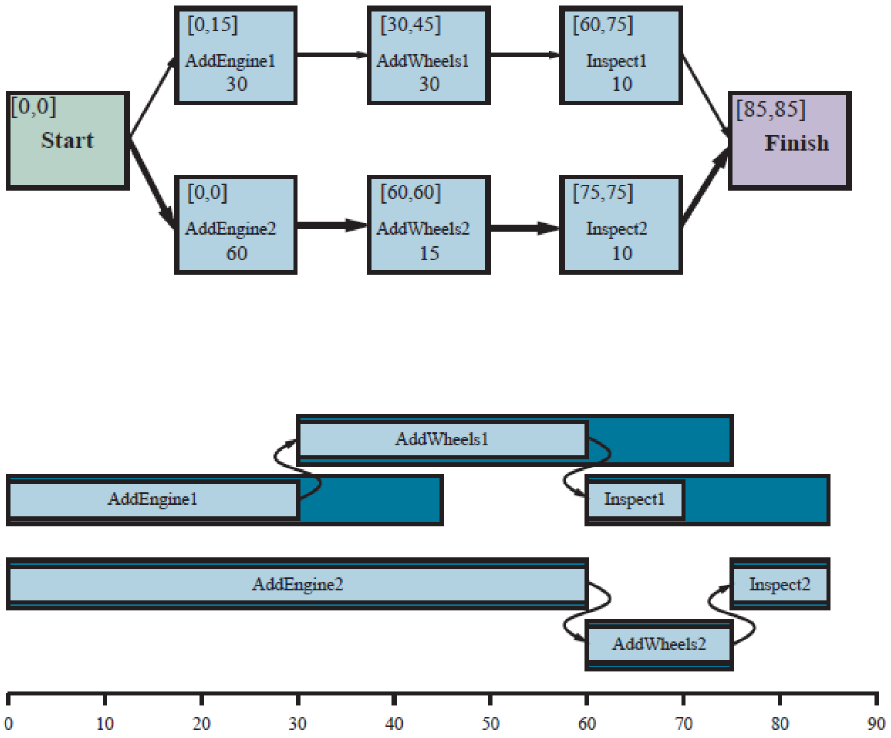


Figure 11.14 Top: a representation of the temporal constraints for the job-shop scheduling problem of Figure 11.13. The duration of each action is given at the bottom of each rectangle. In solving the problem, we compute the earliest and latest start times as the pair $[ES, LS]$, displayed in the upper left. The difference between these two numbers is the *slack* of an action; actions with zero slack are on the critical path, shown with bold arrows. Bottom: the same solution shown as a timeline. Grey rectangles represent time intervals during which an action may be executed, provided that the ordering constraints are respected. The unoccupied portion of a gray rectangle indicates the slack.

Time, Schedules, and Resources

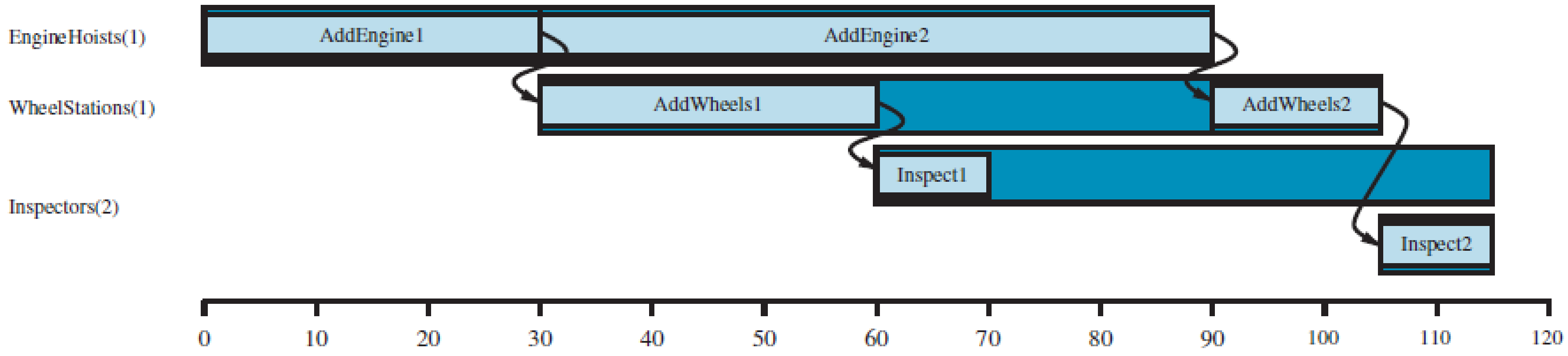


Figure 11.15 A solution to the job-shop scheduling problem from Figure 11.13, taking into account resource constraints. The left-hand margin lists the three reusable resources, and actions are shown aligned horizontally with the resources they use. There are two possible schedules, depending on which assembly uses the engine hoist first; we've shown the shortest-duration solution, which takes 115 minutes.