Planning and Scheduling with Time and Resources

Section 11.1
Outline

- Scheduling problems vs. planning problems
- Scheduling with time constraints
- Scheduling with resource constraints

Additional references used for the slides:
Planning vs. scheduling

Planning
- Involves choice of actions
- Cannot deal with time and resource constraints

Scheduling
- Can easily represent time and resource constraints
- Cannot deal with action choices

Most real world problems are optimization problems that involve continuous time, resources, metric quantities, and a complex mixture of action choices and ordering decisions.
### Planning vs. Scheduling

<table>
<thead>
<tr>
<th>Planning problem</th>
<th>Scheduling problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state, goals</td>
<td>set of jobs (possibly partially ordered)</td>
</tr>
<tr>
<td>action descriptions</td>
<td>temporal constraints on jobs (EST, LFT, duration)</td>
</tr>
<tr>
<td>Synthesize a sequence of actions</td>
<td>resource constraints</td>
</tr>
<tr>
<td></td>
<td>Assign optimal start times and resources</td>
</tr>
</tbody>
</table>
Dealing with time

- **EST**: earliest start time
- **LFT**: latest finish time
- **duration**
- **CPM**: critical path method. A *path* is a sequence of actions that depend on each other. A *critical path* is the longest path. Delaying it would delay the entire plan.
Example

Init (Chassis(C₁) ∧ Chassis(C₂) ∧ 
Engine(E₁,C₁,30) ∧ Engine(E₂,C₂,60) ∧ 
Wheels(W₁,C₁,30) ∧ Wheels(W₂,C₂,15)) 
Goal(Done(C₁) ∧ Done(C₂))

Action(AddEngine(e,c),
  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) ∧ EngineIn(c)
  Effect: WheelsOn(c) ∧ Duration(d))

Action(Inspect(c),
  Precond: EngineIn(c) ∧ WheelsOn(c) ∧ Chassis(c)
  Effect: Done(c) ∧ Duration(10))
Dealing with resources

- **reusable resource**: is occupied during an action, and is freed afterwards
- **aggregation of resources**: group indistinguishable resources into quantities
- **Minimum slack algorithm**: a greedy algorithm
Example

\[ \text{Init (Chassis}(C_1) \land \text{Chassis}(C_2) \land \text{Engine}(E_1,C_1,30) \land \]
\[ \quad \text{Engine}(E_2,C_2,60) \land \text{Wheels}(W_1,C_1,30) \land \text{Wheels}(W_2,C_2,15) \land \]
\[ \quad \text{EngineHoists}(1) \land \text{WheelStations}(1) \land \text{Inspectors}(2)) \]

\[ \text{Goal(Done}(C_1) \land \text{Done}(C_2)) \]

Action(AddEngine(e,c),
\quad \text{PRECOND: Engine}(e,c,d) \land \text{Chassis}(c) \land \neg \text{EngineIn}(c)
\quad \text{EFFECT: EngineIn}(c) \land \text{Duration}(d)
\quad \text{RESOURCE: EngineHoists}(1))

Action(AddWheels(w,c),
\quad \text{PRECOND: Wheels}(w,c,d) \land \text{Chassis}(c) \land \text{EngineIn}(c)
\quad \text{EFFECT: WheelsOn}(c) \land \text{Duration}(d)
\quad \text{RESOURCE: WheelStations}(1))

Action(Inspect(c),
\quad \text{PRECOND: EngineIn}(c) \land \text{WheelsOn}(c) \land \text{Chassis}(c)
\quad \text{EFFECT: Done}(c) \land \text{Duration}(10)
\quad \text{RESOURCE: Inspectors}(1))
Each can do its own job. The big question is how best to couple them to avoid inter-module trashing.

The second big question is which planners are most suitable for coupling.