CMSC 722, AI Planning

Planning and Scheduling

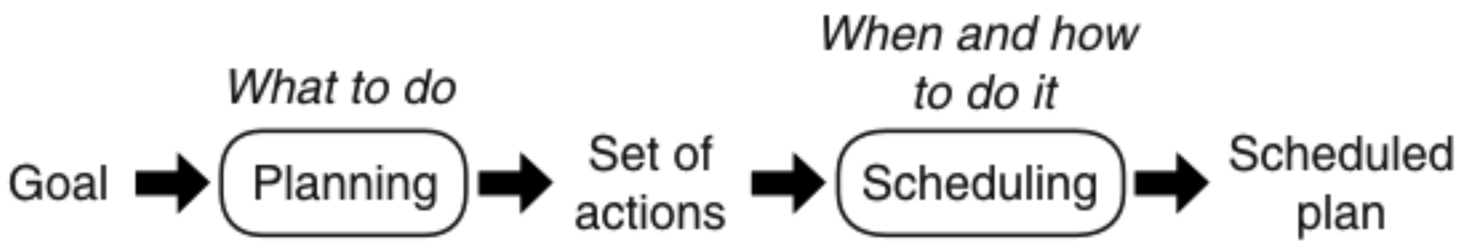
Dana S. Nau
University of Maryland

1:26 PM April 24, 2012

Scheduling

- Given:
  - actions to perform
  - set of resources to use
  - time constraints
    - e.g., the ones computed by the algorithms in Chapter 14
- Objective:
  - allocate times and resources to the actions
- What is a resource?
  - Something needed to carry out the action
  - Usually represented as a numeric quantity
  - Actions modify it in a *relative* way
  - Several concurrent actions may use the same resource

Planning and Scheduling



- Scheduling has usually been addressed separately from planning
  - E.g., the temporal planning in Chapter 14 didn't include scheduling
- Thus, will give an overview of scheduling algorithms
- In some cases, cannot decompose planning and scheduling so cleanly
  - Thus, will discuss how to integrate them

Scheduling Problem

- Scheduling problem
  - set of resources and their future availability
  - actions and their resource requirements
  - constraints
  - cost function
- Schedule
  - allocations of resources and start times to actions
    - » must meet the constraints and resource requirements

Actions

- Action a
  - resource requirements
    - » which resources, what quantities
  - usually, upper and lower bounds on start and end times
    - » Start time $s(a) \in [s_{min}(a), s_{max}(a)]$
    - » End time $e(a) \in [e_{min}(a), e_{max}(a)]$
- Non-preemptive action: cannot be interrupted
  - Duration $d(a) = e(a) - s(a)$
- Preemptive action: can interrupt and resume
  - Duration $d(a) = \sum_{i \in I} d_i(a) \leq e(a) - s(a)$
  - can have constraints on the intervals

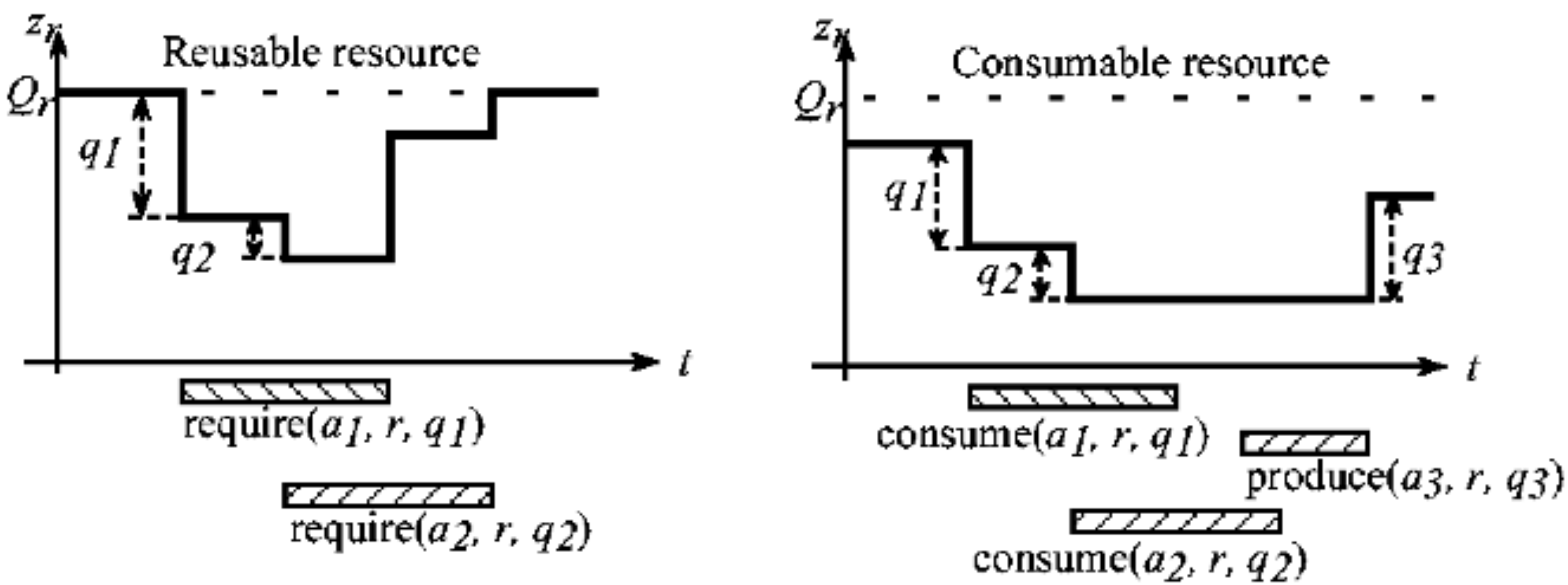
Reusable Resources

- A *reusable* resource is "borrowed" by an action, and released afterward
  - e.g., use a tool, return it when done
- Total capacity Q_i for r_i may be either discrete or continuous
  - Current level $z_i(t) \in [0,Q_i]$ is
    - » $z_i(t)$ = how much of r_i is currently available
- If action requires quantity q of resource $r_{i,}$
  - Then decrease z_i by q at time $s(a)$
    and increase z_i by q at time $e(a)$
- Example: five cranes at location l_i:
  - We might represent this as $Q_i = 5$
  - Two of them in use at time t: $z_i(t) = 5 - 2 = 3$

Consumable Resources

- A *consumable* resource is used up (or in some cases produced) by an action
  - e.g., fuel
- Like before, we have total capacity Q_i and current level $z_i(t)$
- If action requires quantity q of r_i
  - Decrease z_i by q at time $s(a)$
  - Don't increase z_i at time $e(a)$

- An action's resource requirement is a conjunct of assertions
  - consume(a,r_j,q_j) & …
- or a disjunct if there are alternatives
  - consume(a,r_j,q_j) ∨ …
- z_i is called the "resource profile" for r_i



Time constraints

- Bounds on start and end points of an action
  - absolute times
    - » e.g., a deadline: $e(a) \leq u$
    - » release date: $s(a) \geq v$
  - relative times
    - » latency: $u \leq s(b) – e(a) \leq v$
    - » total extent: $u \leq e(a) – s(a) \leq v$
- Constraints on availability of a resource
  - e.g., can only communicate with a satellite at certain times

Costs

- may be fixed
- may be a function of quantity and duration
  - e.g., a set-up cost to begin some activity,
    plus a run-time cost that's proportional to the amount of time
- e.g., suppose a follows b
  - cost $c_r(a,b)$ for a
  - duration $d_r(a,b)$, i.e., $s(b) \geq e(a) + d_r(a,b)$

- Objective: minimize some function of the various costs and/or end-times
  - the makespan or maximum ending time of the schedule, i.e., $f = max_i\{e(a_i) \mid a_i \in A\}$,
  - the *total weighted completion time*, i.e., $f = \Sigma_i w_i e(a_i)$, where the constant $w_i \in \Re^+$ is the weight of action a_i,
  - the maximum tardiness, i.e., $f = max\{\tau_i\}$, where the tardiness τ_i is the time distance to the deadline δ_{a_i} when the action a_i is late, i.e., $\tau_i = max\{0, e(a_i) - \delta_{a_i}\}$,
  - the total weighted tardiness, i.e., $f = \Sigma_i w_i \tau_i$,
  - the total number of late actions, i.e., for which $\tau_i > 0$,
  - the weighted sum of late actions, i.e., $f = \Sigma_i w_i u_i$, where $u_i = 1$ when action i is late and $u_i = 0$ when i meets its deadline,
  - the total cost of the schedule, i.e., the sum of the costs of allocated resources, of setup costs, and of penalties for late actions,
  - the peak resource usage, and
  - the total number of resources allocated.

Types of Scheduling Problems

- **Machine scheduling**
  - machine *i*: unit capacity (in use or not in use)
  - job *j*: partially ordered set of actions $a_{j1}, \ldots, a_{jk}$
  - schedule:
    - a machine *i* for each action a_{jk}
    - a time interval during which *i* processes a_{jk}
    - no two actions can use the same machine at once
  - actions in different jobs are completely independent
  - actions in the same job cannot overlap
    - e.g., actions to be performed on the same physical object

Single-Stage Machine Scheduling

- **Single-stage machine scheduling**
  - each job is a single action, and can be processed on any machine
  - identical parallel machines
    - processing time p_j is the same regardless of which machine
    - thus we can model all m machines as a single resource of capacity m
  - uniform parallel machines
    - machine i has speed(i); time for j is p_j/speed(i)
  - unrelated parallel machines
    - different time for each combination of job and machine

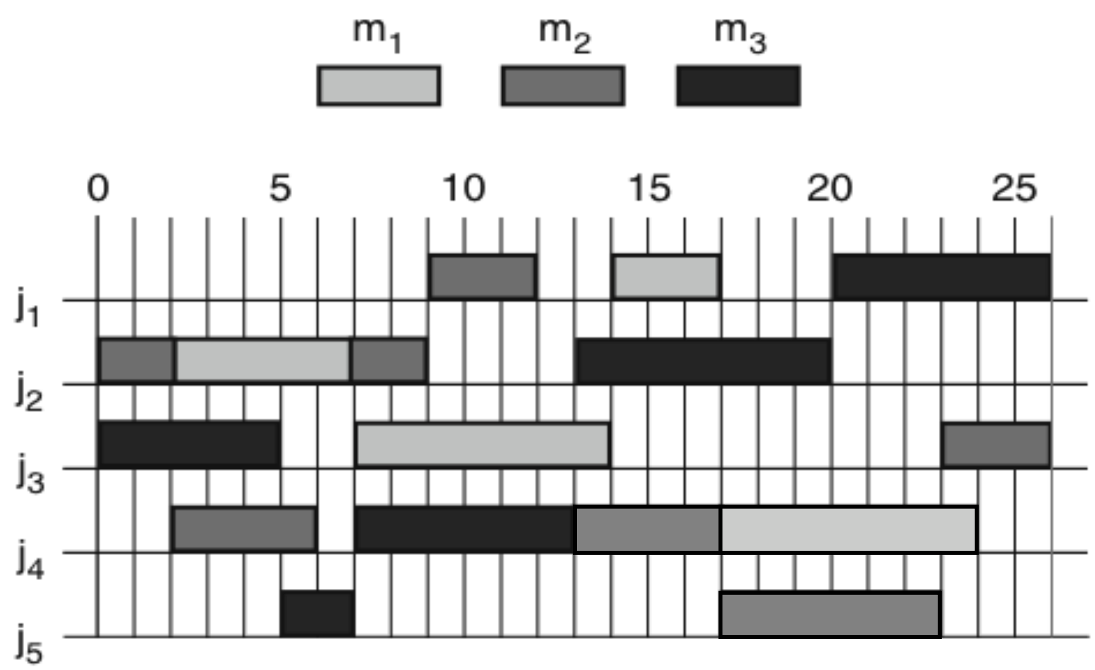
Multiple-Stage Scheduling

- **Multiple-stage** scheduling problems
  - job contains several actions
  - each requires a particular machine
  - **flow-shop** problems:
    - each job j consists of exactly m actions $\{a_{j1}, a_{j2}, \ldots, a_{jm}\}$
    - each a_{ji} needs to be done on machine i
    - actions must be done in order $a_{j1}, a_{j2}, \ldots, a_{jm}$
  - **open-shop** problems
    - like flow-shop, but the actions can be done in any order
  - **job-shop** problems (general case)
    - constraints on the order of actions, and which machine for each action

Example

- Job shop: machines m_1, m_2, m_3 and jobs $j_1, \ldots, j_5$

- j_1: $\langle m_2(3), m_1(3), m_3(6)\rangle$
    - *i.e.,* m_2 for 3 time units
      then m_1 for 3 time units
      then m_3 for 6 time units
- j_2: $\langle m_2(2), m_1(5), m_2(2), m_3(7)\rangle$
- j_3: $\langle m_3(5), m_1(7), m_2(3)\rangle$
- j_4: $\langle m_2(4), m_3(6), m_2(4), m_1(7)\rangle$
- j_5: $\langle m_3(2), m_2(6)\rangle$



Notation

- Standard notation for designating machine scheduling problems:

  $\alpha \mid \beta \mid \gamma$

  α = type of problem:

  - P (identical), U (uniform), R (unrelated) parallel machines
  - F (flow shop), O (open shop), J (job shop)

  β = job characteristics (deadlines, setup times, precedence constraints), empty if there are no constraints

  γ = the objective function

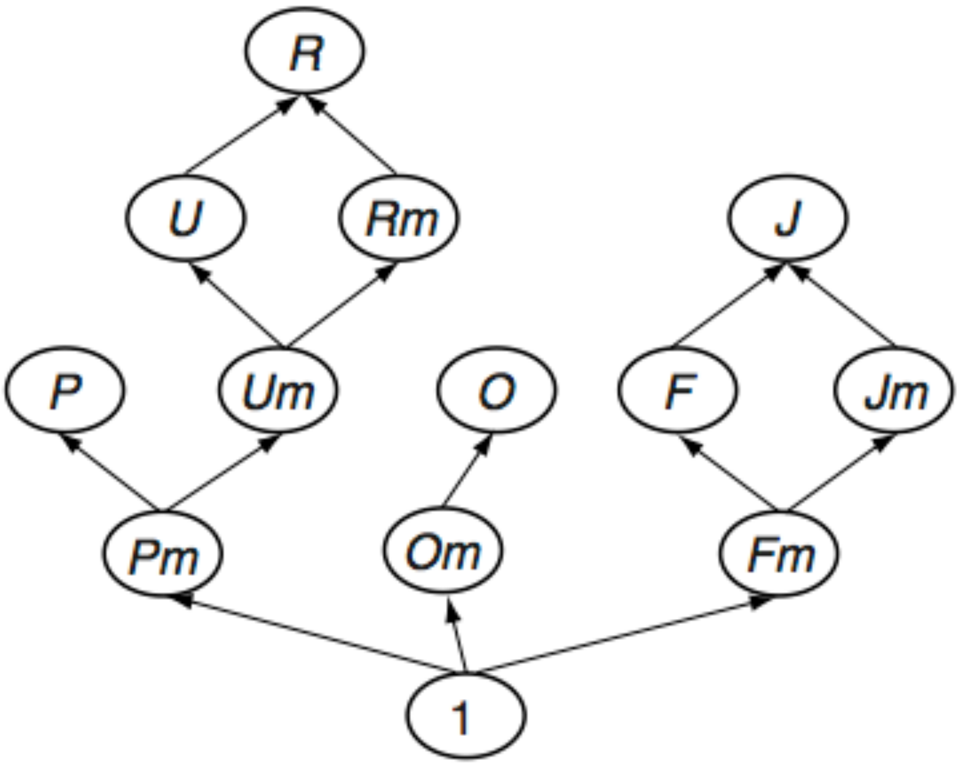
- Examples:
  - $Pm \mid \delta_j \mid \Sigma_j w_j e_j$
    - » m identical parallel machines, deadlines on jobs, minimize weighted completion time
  - $J \mid prec \mid makespan$
    - » job shop with arbitrary number of machines, precedence constrants between jobs, minimize the makespan

Complexity

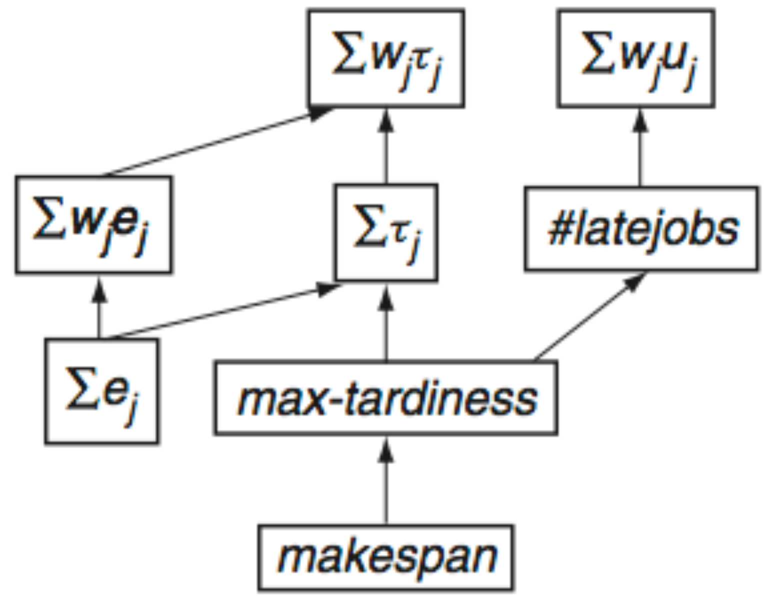
- Most machine scheduling problems are NP-hard
- Many polynomial-time reductions

Problem types
(α in the $\alpha|\beta|\gamma$ notation):

- P - identical parallel machines
- U - uniform parallel machines
- R - unrelated parallel machines
- F - flow shop
- O - open shop
- J - job shop



Reductions for α = type of problem



Reductions for γ = the objective function

Solving Machine Scheduling Problems

- Integer Programming (IP) formulations
  - n-dimensional space
  - Set of constraints C, all are linear inequalities
  - Linear objective function f
  - Find a point $p=(x_1,\ldots, x_n)$ such that
    - p satisfies C
    - p is integer-valued, i.e., every x_i is an integer
    - no other integer-valued point p' satisfies C and has $f(p') < f(p)$
- A huge number of problems can be translated into this format

- An entire subfield of Operations Research is devoted to IP
  - Several commercial IP solvers

IP Solvers

- Most IP solvers use *depth-first branch-and-bound*
  - Want a solution u that optimizes an objective function $f(u)$
  - Node selection is guided by a lower bound function $L(u)$
    - For every node u, $L(u) \leq \{f(v) : v \text{ is a solution in the subtree below } u\}$
    - Backtrack if $L(u) \geq f(u^*)$, where u^* = the best solution seen so far

procedure DFBB
  global $u^* \leftarrow$ fail; $f^* \leftarrow \infty$
  call search(r), where r is the initial node
  return (u^*, f^*)

procedure search(u)
  if u is a solution and $f(u) < f^*$
    then $u^* \leftarrow u$; $f^* \leftarrow f(u)$
  else if u has no unvisited children or $L(u) \geq f^*$
    then do nothing
  else call search(v), where $v = \text{argmin}\{L(v) : v \text{ is a not-yet-visited child of } u\}$

$L(u)$ very similar to A*'s heuristic function $f(u) = g(u) + h(u)$

Main difference: L isn't broken into f's two components g and h

A* can be expressed as a *best-first* branch-and-bound procedure

Planning as Scheduling

- Some planning problems can be modeled as machine-scheduling problems
- Example: modified version of DWR
  - m identical robots, several distinct locations
  - job: container-transportation(c,l,l')
    - » go to l, load c, go to l', unload c
      - All four tasks to be done by the same robot (which can be any robot)
  - release dates, deadlines, durations
  - setup time t_{ijk} if robot i does job j after performing job k
  - minimize weighted completion time

Let's ignore this for a moment

class $P|r_j\delta_j t_{ikj}|\Sigma_j w_j e_j$, where r_j, δ_j, and t_{ikj} denote respectively the release date, the deadline and the setup times of job j. □

- Can generalize the example to allow cranes for loading/unloading, and arrangement of containers into piles
- **Problem**: the machine-scheduling model can't handle the part I said to ignore
  - Can specify a *specific* robot r_i for each job j_i, but can't leave it unspecified

Limitations

- Some other characteristics of AI planning problems that don't fit machine scheduling
  - Precedence constraints on ends of jobs
    - Beyond the standard classes
    - Hard in practice for scheduling problems
      - How to control the end times of actions?
    - Could avoid this if we allow containers to be in any order within a pile
  - We have ignored some of the resource constraints
    - E.g., one robot in a location at a time

Discussion

- Overall, machine scheduling is too restricted to handle all the needs of planning
- But it is very well studied
  - Heuristics and techniques that can be useful for planning with resources

Integrating Planning and Scheduling

- Extend the chronicle representation to include resources
  - finite set $Z=\{z_1,\ldots,z_m\}$ of resource variables
    - z_i is the resource profile for resource i
- Like we did with other state variables, will use function-and-arguments notation to represent resource profiles
  - cranes(l) = number of cranes available at location l
- Will focus on reusable resources
  - resources are borrowed but not consumed

Temporal Assertions

- Resource variable z whose total capacity is Q
- A *temporal assertion* on z is one of the following:
  - Decrease z by amount q at time t: $z@t : -q$
  - Increase z by amount q at time t: $z@t : +q$
  - Use amount q of z during $[t,t')$: $z@[t,t') : q$
    - » Equivalent to $z@t : -q \wedge z@t' : +q$

- Consuming a resource is like using it *ad infinitum:*
  - $z@t : -q$ is equivalent to $z@[t,\infty) : q$

- Producing a resource is like having a higher initial capacity $Q' = Q + q$ at time 0, and using q of it during $[0, t)$:
  - $z@t : +q$ is equivalent to $z@0 : +q$ & $z@[0,t) : q$

Resource Capacity

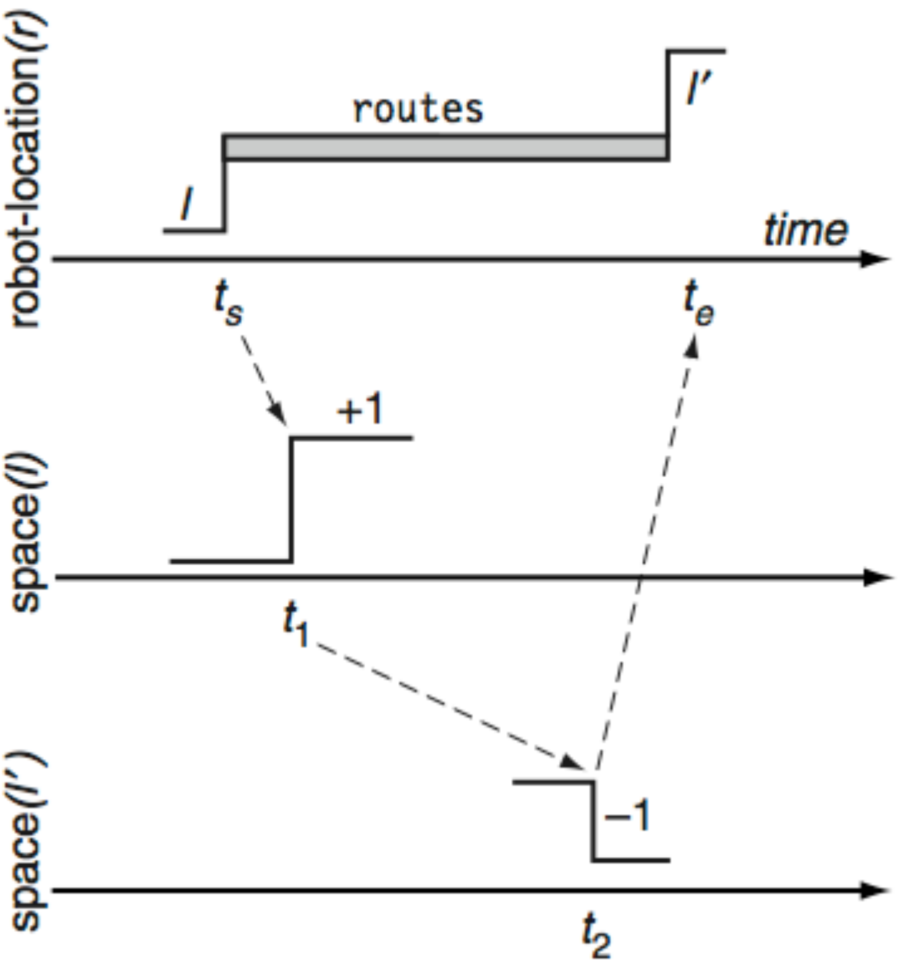
- Also need to specify total capacity of each resource
  - E.g., suppose we modify DWR so that locations can hold multiple robots
  - Need to specify how many robots each location can hold

- One way: fixed total capacity Q: maximum number of spots at each location
  - E.g., $Q = 12$ means each location has at most 12 spots
  - If location loc1 has only 4 spots, then we've specified 8 more spots than it actually has
  - To make the 8 nonexistent spots unavailable, assert that they're in use
    - The initial chronicle will contain space(loc1)@[0,∞):8

- Another way: make Q depend on the location
  - Q(loc1) = 4, Q(loc2) = 12, …

Example

- DWR domain, but locations may hold more than one robot
  - Resource variable space(l) = number of available spots at location l
  - Each robot requires one spot



$$
\begin{aligned}
\text{move}(t_s, t_e, t_1, t_2, r, l, l') \\
= \{&\text{robot-location}(r)@t_s : (l, \text{routes}),\\
&\text{robot-location}(r)@[t_s, t_e] : \text{routes},\\
&\text{robot-location}(r)@t_e : (\text{routes}, l'),\\
&\text{space}(l)@t_1 : +1,\\
&\text{space}(l')@t_2 : -1,\\
&t_s < t_1 < t_2 < t_e,\\
&\text{adjacent}(l, l')\ \}
\end{aligned}
$$

Possibly Intersecting Assertions

- Assume distinct resources are completely independent
  - Using a resource z does not affect another resource z'
  - Every assertion about a resource concerns just one resource
- Don't need consistency requirements for assertions about different resource variables, just need them for assertions about the same variable
- Let $\Phi = (F,C)$ be a chronicle
  - Suppose $z@[t_i,t_i'):q_i$ and $z@[t_j,t_j'):q_j$ be two temporal assertions in F
    - » both are for the same resource z
- $z@[t_i,t_i'):q_i$ and $z@[t_j,t_j'):q_j$ are *possibly intersecting*
  - iff $[t_i,t_i')$ and $[t_i,t_i')$ are possibly intersecting
  - iff C does not make them disjoint
    - » i.e., C does not entail $t_i' \leq t_j$ nor $t_j' \leq t_i$
- Similar if there are than two assertions about z

Conflict and Consistency

- Intuitively, R_z is conflicting if it is possible for R_z to use more than z's total capacity Q.

Definition 15.2 A set R_z of temporal assertions about the resource variable z is *conflicting* iff there is a possibly intersecting set of assertions $\{z@[t_i, t'_i):q_i \mid i \in I\} \subseteq R_z$ such that $\sum_{i \in I} q_i > Q$. ■

- To see if R_z possibly intersects, it's sufficient to see if each pair of assertions in R_z possibly intersects:

Proposition 15.1 *A set R_z of temporal assertions on the resource variable z is conflicting iff there is a subset $\{z@[t_i, t'_i):q_i \mid i \in I\} \subseteq R_z$ such that every pair $i, j \in I$ is possibly intersecting, and $\sum_{i \in I} q_i > Q$.*

- A chronicle is *consistent* if
  - Temporal assertions on state variables are consistent, in the sense specified in Chapter 14
  - No conflicts among temporal assertions

Planning Problems

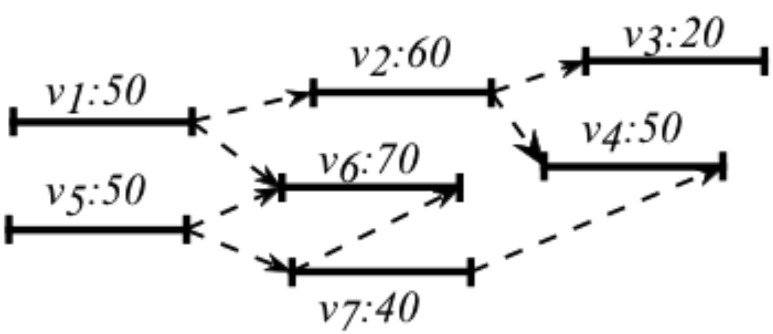
- Suppose we're only trying to find a feasible plan, not an optimal one
  - Then except for the resources, our definitions of planning domain, planning problem, etc. are basically the same as in Chapter 14

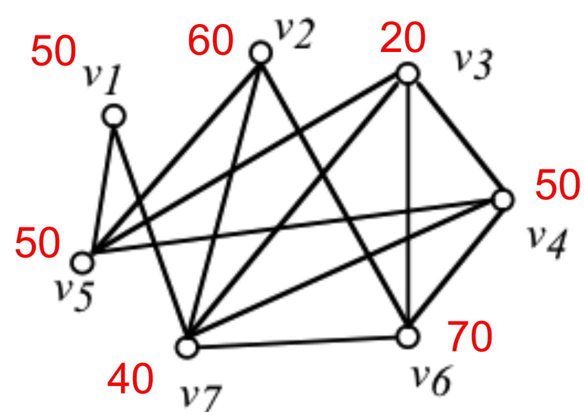
- Recall that in Chapter 14 we had two kinds of flaws
  - Open goals
  - Threats
- We now have a third kind of flaw
  - A *resource conflict flaw* for a resource variable z in a chronicle Φ is a set of conflicting temporal assertions for z in Φ

- Given a resource conflict flaw, what are all the possible ways to resolve it?

PIA Graphs

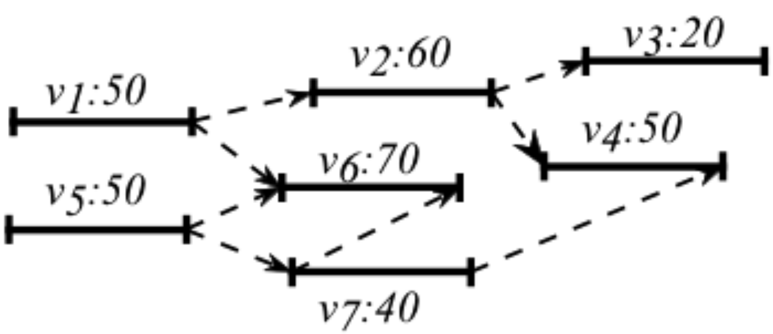
- Let $R_z = \{z@[t_1,t_1'){:}q_1, \ldots, z@[t_n,t_n'){:}q_n\}$ be all temporal assertions about z in a chronicle (F,C)
- The *Possibly Intersecting Assertions (PIA)* graph is $H_z = (V,E)$, where:
  - V contains a vertex v_i for each assertion $z@[t_i,t_i'){:}q_i$
  - E contains an edge (v_i,v_j) for each pair of intervals $[t_i,t_i')$, $[t_j,t_j')$ that possibly intersect
- Example:
  - $R_z = \{\ z@[t_1,t_1'){:}50,\ z@[t_2,t_2'){:}60,\ z@[t_3,t_3'){:}20,\ z@[t_4,t_4'){:}50,$
    $z@[t_5,t_5'){:}50,\ z@[t_6,t_6'){:}70,\ z@[t_7,t_7'){:}40\}.$
  - C contains $t_i < t_i'$ for all i, and also contains
    $t_1' < t_2,\quad t_1' < t_6,\quad t_2' < t_3,\quad t_2' < t_4,\quad t_5' < t_6,\quad t_5' < t_7,\quad t_7 < t_6',\quad t_7' < t_4'$

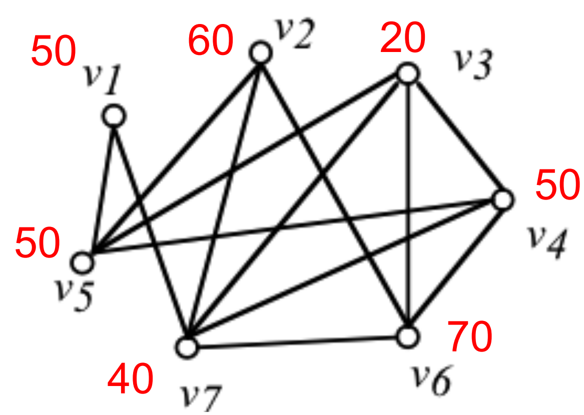




Minimal Critical Sets

- *Minimal Critical Set (MCS)*: a subset U of V such that
  - U is an over-consuming clique
  - No proper subset of U is over-consuming
- Example, continued:
  - $$R_z = \{ \quad z@[t_1, t'_1){:}50, \quad z@[t_2, t'_2){:}60, \quad z@[t_3, t'_3){:}20, \quad z@[t_4, t'_4){:}50, \quad z@[t_5, t'_5){:}50, \quad z@[t_6, t'_6){:}70, \quad z@[t_7, t'_7){:}40\}.$$
  - Suppose z's capacity is Q=100
- $\{v_1, v_5\}$ is a clique, but is not over-consuming
- $\{v_3, v_4, v_6, v_7\}$ is an over-consuming clique, but is not minimal
- $\{v_6, v_7\}$, $\{v_4, v_6\}$, and $\{v_3, v_4, v_7\}$ are *minimal critical sets* (MCSs) for z





Finding Every Minimax Critical Set

MCS-expand(p)
    for each $v_i \in$ pending(p) do
        add a new node m_i successor of p
        pending(m_i) $\leftarrow \{v_j \in$ pending(p) $\mid j < i$ and $(v_i, v_j) \in E\}$
        clique(m_i) $\leftarrow$ clique(p) $\cup \{v_i\}$
        if clique(m_i) is over-consuming than MCS $\leftarrow$ MCS $\cup$ clique(m_i)
        else if pending(m_i) $\neq \emptyset$ than MCS-expand(m_i)
end

- Assume the set of vertices is $V = \{v_1, \ldots, v_n\}$
- Depth-first search; each node p is a pair (clique(p), pending(p))
    - clique(p) is the current clique
    - pending(p) is the set of candidate vertices to add to clique(p)
- Initially, $p = (\varnothing, V)$
- Two kinds of leaf nodes:
    - clique(p) is not over-consuming but pending(p) is empty => dead end
    - clique(p) is over-consuming => found an MCS

MCS-expand(p)

  for each $v_i \in$ pending(p) do

    add a new node m_i successor of p

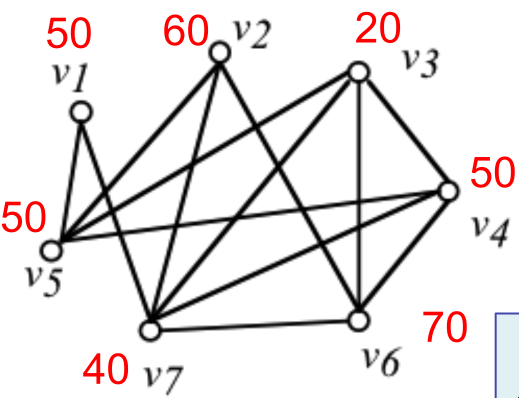
    pending(m_i) $\leftarrow \{v_j \in$ pending(p) $\mid j < i$ and $(v_i, v_j) \in E\}$

    clique(m_i) $\leftarrow$ clique(p) $\cup \{v_i\}$

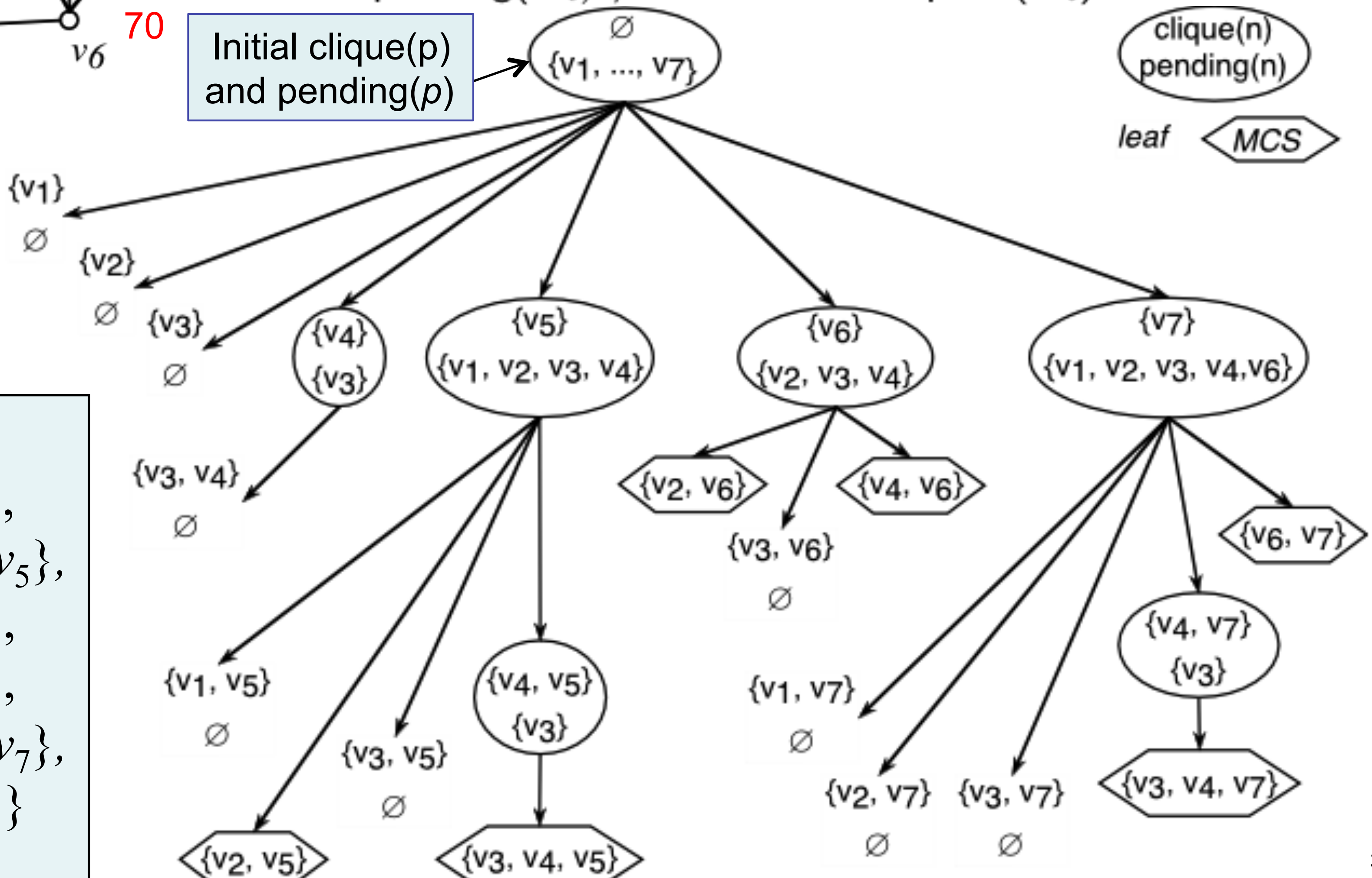
    if clique(m_i) is over-consuming than MCS $\leftarrow$ MCS $\cup$ clique(m_i)

    else if pending(m_i) $\neq \emptyset$ than MCS-expand(m_i)

vertices "below" v_i that are adjacent to v_i



Initial clique(p) and pending(p)



- MCS = $\{\{v_2,v_5\},$
  $\{v_3,v_4,v_5\},$
  $\{v_2,v_6\},$
  $\{v_4,v_6\},$
  $\{v_3,v_4,v_7\},$
  $\{v_6,v_7\}\}$

Resolving Resource-Conflict Flaws

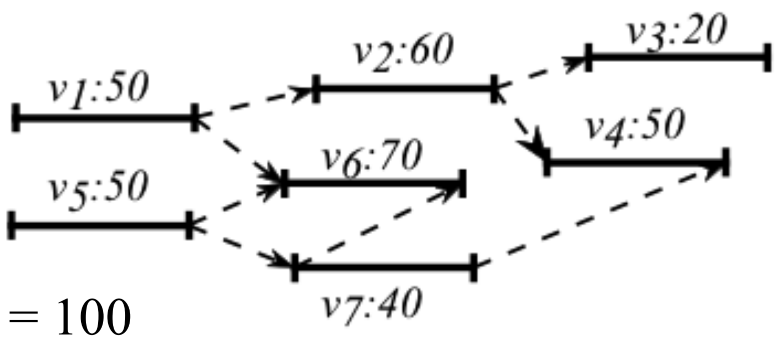
- Suppose $U = \{z@[t_i,t_i'):q_i : i \text{ in } I\}$ is a minimal critical set for z in a chronicle $\Phi=(F,C)$
  - For every pair of assertions $r_i = z@[t_i,t_i'):q_i$ and $r_j = z@[t_j,t_j'):q_j$ in I, let c_{ij} be the constraint $t_i' \leq t_j$ (i.e., c_{ij} makes r_i precede r_j)
- Each c_{ij} is a possible resolver of the resource conflict
  - If we add c_{ij} to C it will make $[t_i,t_i')$ and $[t_j,t_j')$ disjoint

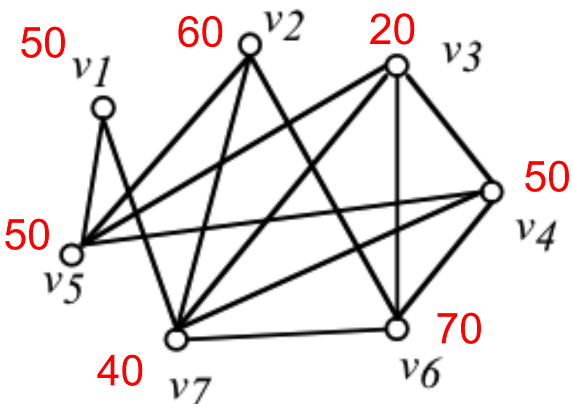
    => U won't be a clique any more
  - Various subsets of U may be cliques
    - » But none of them is overconsuming, since U is a *minimal* critical set
- If U is the only MCS in R_z, then adding c_{ij} makes R_z non-conflicting
- If R_z contains several MCSs, add one constraint to C for each MCS in R_z

Continuing the Previous Example …

$$R_z = \{ \quad z@[t_1, t_1'):50, \quad z@[t_2, t_2'):60, \quad z@[t_3, t_3'):20, \quad z@[t_4, t_4'):50,$$
$$z@[t_5, t_5'):50, \quad z@[t_6, t_6'):70, \quad z@[t_7, t_7'):40\}.$$

C contains $t_1' < t_2$, $t_1' < t_6$, $t_2' < t_3$, $t_2' < t_4$, $t_5' < t_6$, $t_5' < t_7$, $t_7 < t_6'$, $t_7' < t_4'$, and $t_i < t_i'$ for all i





- Recall that
  - Capacity is $Q = 100$
  - Each v_i starts at t_i and ends at t_i'
  - The MCSs are $\{\{v_2,v_5\}, \{v_3,v_4,v_5\}, \{v_2,v_6\}, \{v_4,v_6\}, \{v_3,v_4,v_7\}, \{v_6,v_7\}\}$
- For the MCS $U = \{v_3, v_4, v_7\}$, there are six possible resolvers:

$$t_3' \le t_4, \quad t_4' \le t_3, \quad t_3' \le t_7, \quad t_7' \le t_3, \quad t_4' \le t_7, \quad t_7' \le t_4$$

  - $t_4' \le t_7$ is inconsistent with C because C contains $t_7' < t_4'$
  - $t_4' \le t_3$ is over-constraining because it implies $t_7' \le t_3$
- Thus the only resolvers for U that we need to consider are
  - $\{ t_3' \le t_4, \quad t_3' \le t_7, \quad t_7' \le t_3, \quad t_7' \le t_4\}$

More about Over-Constraining Resolvers

- In general, a set of resolvers r' is *equivalent* to r if both
  - $r' \cup C$ entails r
  - $r \cup C$ entails r'
- There is a unique minimal set of resolvers r' that is equivalent to r
  - Desirable because it produces a smaller branching factor in the search space
  - Can be found in time $O(|U|^3)$ by removing over-constraining resolvers

CPR$(\Phi, G, \mathcal{K}, \mathcal{M}, \pi)$
if $G = \mathcal{K} = \mathcal{M} = \emptyset$ then return(π)
perform the three following steps in any order
if $G \neq \emptyset$ then do
select any $\alpha \in G$
if $\theta(\alpha/\Phi) \neq \emptyset$ then return(CPR$(\Phi, G - \{\alpha\}, \mathcal{K}\{\cup\, \theta(\alpha/\Phi\}), \mathcal{M}, \pi)$)
else do
$relevant \leftarrow \{a \mid a$ applicable to Φ and has a provider for $\alpha\}$
if $relevant = \emptyset$ then return(failure)
nondeterministically choose $a \in relevant$
$\mathcal{M}' \leftarrow$ the update of $\mathcal{M}$ with respect to $\Phi \cup (\mathcal{F}(a), \mathcal{C}(a))$
return(CPR$(\Phi \cup (\mathcal{F}(a), \mathcal{C}(a)), G \cup \mathcal{F}(a), \mathcal{K} \cup \{\theta(a/\Phi)\}, \mathcal{M}', \pi \cup \{a\})$)
if $\mathcal{K} \neq \emptyset$ then do
select any $C \in \mathcal{K}$
$threat\text{-}resolvers \leftarrow \{\phi \in C \mid \phi$ consistent with $\Phi\}$
if $threat\text{-}resolvers = \emptyset$ then return(failure)
nondeterministically choose $\phi \in threat\text{-}resolvers$
return(CPR$(\Phi \cup \phi, G, \mathcal{K} - C, \mathcal{M}, \pi)$)
if $\mathcal{M} \neq \emptyset$ then do
select $U \in \mathcal{M}$
$resource\text{-}resolvers \leftarrow \{\phi$ resolver of $U \mid \phi$ is consistent with $\Phi\}$
if $resource\text{-}resolvers = \emptyset$ then return(failure)
nondeterministically choose $\phi \in resource\text{-}resolvers$
$\mathcal{M}' \leftarrow$ the update of $\mathcal{M}$ with respect to $\Phi \cup \phi$
return(CPR$(\Phi \cup \phi, G, \mathcal{K}, \mathcal{M}', \pi)$)

Three main steps:

- solve open-goal flaws
- solve threat flaws
- solve resource-conflict flaws