Chapter 4
Deliberation with Temporal Domain Models

Section 4.4: Constraint Management
Section 4.5: Acting with Temporal Models

Dana S. Nau
University of Maryland
Outline

✓ Introduction
✓ Representation
✓ Temporal planning

● 4.4 Constraint Management
  ➢ Consistency of object constraints and time constraints
  ➢ Controlling the actions when we don’t know how long they’ll take

● 4.5 Acting with temporal models
Constraint Management

- Each time TemPlan applies a resolver, it modifies \((T, C)\)
  - Some resolvers will make \((T, C)\) inconsistent
    - No solution in this part of the search space
  - Detect inconsistency => prune this part of the search space
  - Don’t detect it => waste time looking for a solution

- Analogy: PSP checked simple cases of inconsistency
  - E.g., can’t create a constraint \(a < b\) if there’s already a constraint \(b < a\)
  - Ignored more complicated cases

- Example:
  - \(c_1, c_2, c_3 \in Containers = \{c1, c2\}\)
  - Suppose that to resolve 3 threats, PSP chooses these resolvers:
    - \(c_1 \neq c_2, c_2 \neq c_3, c_1 \neq c_3\)
    - No solutions in this part of the search space, but PSP searches it anyway
Constraint Management in TemPlan

- At various points, check consistency of $C$
  - If $C$ is inconsistent, then $(T, C)$ is inconsistent
    - Can prune this part of the search space
  - If $C$ is consistent then $(T, C)$ may or may not be consistent

- Example of a case where $C$ is consistent but $(T, C)$ isn’t:
  - $T = \{[t_1, t_2] \text{loc}(r1)=\text{loc1},$
    $[t_3, t_4] \text{loc}(r1)=\text{loc2}\}$
  - $C = (t_1 < t_3 < t_4 < t_2)$
  - Gives $\text{loc}(r1)$ two values during $[t_3, t_4]$
Consistency of $C$

- $C$ contains two kinds of constraints
  - Object constraints
    - $\text{loc}(r) \neq l_2, \ l \in \{\text{loc3, loc4}\}, \ r = r_1, \ o \neq o'$
  - Temporal constraints
    - $t_1 < t_3, \ a < t, \ t < t', \ a \leq t' - t \leq b$

- Assume object constraints are independent of temporal constraints and vice versa
  - exclude things like $t < f(l, r)$

- Then two separate subproblems
  - (1) check consistency of object constraints
  - (2) check consistency of temporal constraints
  - $C$ is consistent iff both are consistent
Object Constraints

- Constraint-satisfaction problem (CSP) – NP-hard
- Can write an algorithm that’s *complete* but runs in exponential time
  - If there’s an inconsistency, always finds it
  - Might do a lot of pruning, but spend lots of time at each node

- Instead, use a constraint-satisfaction technique that’s incomplete but takes *polynomial* time
  - arc consistency, path consistency
- Detects some inconsistencies but not others
  - Runs much faster, but prunes fewer nodes
Time Constraints

To represent time constraints:

- Simple Temporal Networks (STNs)
  - Networks of constraints on time points

- Synthesize incrementally them starting from $\phi_0$
  - Templan can check time constraints in time $O(n^3)$

- Incrementally instantiated at acting time
- Kept consistent throughout planning and acting
Time Constraints

- **Simple Temporal Network (STN):**
  - a pair \((V, E)\), where
    - \(V = \{\text{a set of temporal variables } \{t_1, \ldots, t_n\}\}
    - \(E \subseteq V^2\) is a set of arcs
  - Each arc \((t_i, t_j)\) is labeled with an interval \([a, b]\)
    - Represents constraint \(a \leq t_j - t_i \leq b\)
    - Equivalently, \(-b \leq t_i - t_j \leq -a\)
  - Representing unary constraints: dummy variable \(t_0 = 0\)
    - Arc \(r_{0i} = (t_0, t_i)\) labeled with \([a, b]\) represents \(a \leq t_i - 0 \leq b\)

- **Solution** to an STN: integer value for each \(t_i\), all constraints satisfied
- **Consistent STN:** has a solution
- **Minimal STN:** for every arc \((t_i, t_j)\) with label \([a, b]\), for every \(t \in [a, b]\)
  - there’s at least one solution such that \(t_j - t_i = t\)
- **Shorthand:** instead of \(a \leq t_j - t_i \leq b\), write \(r_{ij} = [a_{ij}, b_{ij}]\)
Operations on STNs

- Intersection, \( \cap \)
  
  \[ t_j - t_i \in r_{ij} = [a_{ij}, b_{ij}] \]
  
  \[ t_j - t_i \in r'_{ij} = [a'_{ij}, b'_{ij}] \]
  
  Infer \( t_j - t_i \in r_{ij} \cap r'_{ij} = [\max(a_{ij}, a'_{ij}), \min(b_{ij}, b'_{ij})] \)

- Composition, \( \cdot \)
  
  \[ t_k - t_i \in r_{ik} = [a_{ik}, b_{ik}] \]
  
  \[ t_j - t_k \in r_{kj} = [a_{kj}, b_{kj}] \]
  
  Infer \( t_j - t_i \in r_{ik} \cdot r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}] \)
  
  Reason: shortest and longest times for the two intervals

- Consistency checking
  
  \( r_{ik}, r_{kj}, r_{ij} \) are consistent if \( r_{ij} \cap (r_{ik} \cdot r_{kj}) \neq \emptyset \)
Two Examples

- STN \((\mathcal{V}, \mathcal{E})\), where
  - \(\mathcal{V} = \{t_1, t_2, t_3\}\)
  - \(\mathcal{E} = \{r_{12}=[1,2], r_{23}=[3,4], r_{13}=[2,3]\}\)
- Composition:
  - \(r'_{13} = r_{12} \cdot r_{23} = [4,6]\)
- Can’t satisfy both \(r_{13}\) and \(r'_{13}\)
  - \(r_{13} \cap r'_{13} = [2,3] \cap [4,6] = \emptyset\)
- \((\mathcal{V}, \mathcal{E})\) is inconsistent

- STN \((\mathcal{V}, \mathcal{E})\), where
  - \(\mathcal{V} = \{t_1, t_2, t_3\}\)
  - \(\mathcal{E} = \{r_{12}=[1,2], r_{23}=[3,4], r_{13}=[2,5]\}\)
- As before, \(r'_{13} = [4,6]\)
- This time, \((\mathcal{V}, \mathcal{E})\) is consistent
  - \(r_{13} \cap r'_{13} = [4,5]\)
- Minimal network: change \(r_{13}\) to \([4,5]\)
Operations on STNs

PC($V,E$):
for $1 \leq k \leq n$ do
  for $1 \leq i < j \leq n$, $i \neq k$, $j \neq k$ do
    $r_{ij} \leftarrow r_{ij} \cap [r_{ik} \cdot r_{kj}]$
    if $r_{ij} = \emptyset$ then
      return inconsistent

- **PC (Path Consistency) algorithm:**
  - Consistency checking on all triples
  - If an arc has no constraint, use $[-\infty, +\infty]$
  - $n$ constraints
    - => $n^3$ triples
    - => time $O(n^3)$

- **Example:** $k = 2$, $i = 1$, $j = 2$
  - $r_{12} = [1,2]$
  - $r_{24} = [3,4]$
  - $r_{14} = [-\infty, \infty]$
  - $r_{12} \cdot r_{24} = [1+3, 2+4] = [4,6]$
  - $r_{14} \leftarrow [\max(-\infty,4), \min(\infty,6)] = [4,6]$
Operations on STNs

PC(\(V, E\)):
\[
\text{for } 1 \leq k \leq n \text{ do }
\]
\[
\text{for } 1 \leq i < j \leq n, \ i \neq k, \ j \neq k \text{ do }
\]
\[
r_{ij} \leftarrow r_{ij} \cap [r_{ik} \cdot r_{kj}]
\]
\[
\text{if } r_{ij} = \emptyset \text{ then }
\]
\[
\text{return inconsistent}
\]

- Detects inconsistent networks
  - \(r_{ij} = [a_{ij}, b_{ij}]\) empty => inconsistent
- Makes STN minimal
  - Shrinks each \(r_{ij}\) to exclude values that aren’t in any solution
- Can modify it to make it incremental
  - Input: a consistent, minimal STN, and a new constraint \(r'_{ij}\)
  - Incorporate \(r'_{ij}\) in time \(O(n^2)\)

Example: \(k = 2, i = 1, j = 2\)
\[
r_{12} = [1, 2]
\]
\[
r_{24} = [3, 4]
\]
\[
r_{14} = [-\infty, \infty]
\]
\[
r_{12} \cdot r_{24} = [1+3, 2+4] = [4, 6]
\]
\[
r_{14} \leftarrow \text{[max}(-\infty, 4), \text{min}(\infty, 6)] = [4, 6]
\]
Pruning TemPlan’s search space

- Take the time constraints in $C$
  - Write them as an STN
  - Use Path Consistency to check whether STN is consistent
  - If it’s inconsistent, TemPlan can backtrack
Controllability

- Section 4.4.3 of the book
- Suppose TemPlan gives you a chronicle and you want to execute it
  - Constraints on time points
  - Need to reason about these in order to decide when to start each action
Controllability

- Solid lines: duration constraints
  - Robot will do bring\&move, will take 30 to 50 time units
  - Crane will do uncover, will take 5 to 10 time units

- Dashed line: synchronization constraint
  - Don’t want either the crane or robot to wait long
  - At most 5 seconds between the two ending times

- Objective
  - Choose time points that will satisfy all the constraints
Controllability

- Suppose we run PC
- PC returns a minimal and consistent network
- There exist time points that satisfy all the constraints
- Would work if we could choose all four time points
  - But we can’t choose $t_2$ and $t_4$

- $t_1$ and $t_3$ are controllable
  - Actor can control when each action starts
- $t_2$ and $t_4$ are contingent
  - Can’t control how long the actions take
  - Random variables that are known to satisfy the duration constraints
    - $t_2 \in [t_1+30, t_1+50]$
    - $t_4 \in [t_3+5, t_3+10]$
Controllability

- Can’t guarantee that all of the constraints will be satisfied
- Start bring&move at time $t_1 = 0$
- Suppose the durations are
  - bring&move 30, uncover 10
    - $t_2 = 0 + 30 = 30$
    - $t_4 = t_3 + 10$
    - $t_4 - t_2 = t_3 - 20$
- Constraint $-5 \leq t_4 - t_2 \leq 5$
  - $-5 \leq t_3 - 20 \leq 5$
- Need to start uncover at $t_3 \leq 25$
  - If $t_3 > 25$ then $t_4 - t_2 > 5$
- But if we start uncover at $t_3 \leq 25$, neither action has finished yet
  - We don’t yet know how long they’ll take
- Might instead get this:
  - bring&move 50, uncover 5
    - $t_2 = 0 + 50 = 50$
    - $t_4 = t_3 + 5 \leq 25 + 5 = 30$
    - $t_4 - t_2 \leq 30 - 50 = -20$
**STNUs**

- **STNU (Simple Temporal Network with Uncertainty):**
  - A 4-tuple \((V, \tilde{V}, E, \tilde{E})\)
    - \(V = \{\text{controllable time points}\} = \{\text{starting times of actions}\}\)
    - \(\tilde{V} = \{\text{contingent time points}\} = \{\text{ending times of actions}\}\)
    - \(E = \{\text{controllable constraints}\}, \tilde{E} = \{\text{contingent constraints}\}\)

- Controllable and contingent constraints:
  - Synchronization between two starting times: controllable
  - Duration of an action: contingent
  - Synchronization between ending points of two actions: contingent
  - Synchronization between end of one action, start of another:
    - Controllable if the new action starts after the old one ends
    - Contingent if the new action starts before the old one ends

- Want a way for the actor to choose time points in \(V\) (starting times) that guarantee that the constraints are satisfied
Three kinds of controllability

- \((V, \tilde{V}, E, \tilde{E})\) is strongly controllable if the actor can choose values for \(V\) such that success will occur for all values of \(\tilde{V}\) that satisfy \(\tilde{E}\)
  - Actor can choose the values for \(V\) offline
  - The right choice will work regardless of \(\tilde{V}\)

- \((V, \tilde{V}, E, \tilde{E})\) is weakly controllable if the actor can choose values for \(V\) such that success will occur for at least one combination of values for \(\tilde{V}\)
  - Actor can choose the values for \(V\) only if the actor knows in advance what the values of \(\tilde{V}\) will be

- Dynamic controllability:
  - Game-theoretic model: actor vs. environment
  - A player’s strategy: a function \(\sigma\) telling what to do in every situation
    - Choices may differ depending on what has happened so far
  - \((V, \tilde{V}, E, \tilde{E})\) is dynamically controllable if \(\exists\) strategy for actor that will guarantee success regardless of the environment’s strategy

Two player, zero sum, extensive form, imperfect information
Dynamic Execution

For $t = 0, 1, 2, \ldots$

1. Actor chooses an unassigned set of variables $\mathcal{V}_t \subseteq \mathcal{V}$ that all can be assigned the value $t$ without violating any constraints in $\mathcal{E}$
   - $\approx$ actions the actor chooses to start at time $t$

2. Simultaneously, environment chooses an unassigned set of variables $\tilde{\mathcal{V}}_t \subseteq \tilde{\mathcal{V}}$ that all can be assigned the value $t$ without violating any constraints in $\tilde{\mathcal{E}}$
   - $\approx$ actions that finish at time $t$

3. Each chosen time point $v$ is assigned $v \leftarrow t$

4. Failure if any of the constraints in $\mathcal{E} \cup \tilde{\mathcal{E}}$ are violated
   - There might be violations that neither $\mathcal{V}_t$ nor $\tilde{\mathcal{V}}_t$ caused individually

5. Success if all variables in $\mathcal{V} \cup \tilde{\mathcal{V}}$ have values and no constraints are violated

- Dynamic execution strategies $\sigma_A$ for actor, $\sigma_E$ for environment
  - $\sigma_A(h_{t-1}) = \{\text{what events in } \mathcal{V} \text{ to assign } = t, \text{ given } h_{t-1}\}$
  - $\sigma_E(h_{t-1}) = \{\text{what events in } \tilde{\mathcal{V}} \text{ to trigger at time } t, \text{ given } h_{t-1}\}$
    - $h_t = h_{t-1} \cdot (\sigma_A(h_{t-1}) \cup \sigma_E(h_{t-1}))$
  - $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is dynamically controllable if $\exists \sigma_A$ that will guarantee success $\forall \sigma_E$

\[ r_{ij} = [l, u] \text{ is violated if } t_i \text{ and } t_j \text{ have values and } t_j - t_i \notin [l, u] \]
Example

- Instead of a single bring&move task, two separate bring and move tasks

- Actor’s dynamic execution strategy
  - trigger $t_1$ at whatever time you want
  - wait and observe $t$
  - trigger $t'$ at any time from $t$ to $t + 5$
  - trigger $t_3 = t' + 10$
  - for every $t_2 \in [t' + 15, t' + 20]$ and every $t_4 \in [t_3 + 5, t_3 + 10]$:
    - $t_4 \in [t' + 15, t' + 20]$
    - so $t_4 - t_3 \in [-5, 5]$
  - So all the constraints are satisfied
Dynamic Controllability Checking

- For a chronicle $\phi = (A, S_T, T, C)$
  - Temporal constraints in $C$ correspond to an STNU
  - Put code into TemPlan to keep the STNU dynamically controllable
- If we detect cases where it isn’t dynamically controllable, then backtrack
- If $PC(\mathcal{V} \cup \mathcal{V}', \mathcal{E} \cup \mathcal{E}')$ reduces a contingent constraint then $(\mathcal{V}, \mathcal{V}', \mathcal{E}, \mathcal{E}')$ isn’t dynamically controllable
  - $\Rightarrow$ can prune this branch
- If it doesn’t reduce any contingent constraints, we don’t know whether $(\mathcal{V}, \mathcal{V}', \mathcal{E}, \mathcal{E}')$ is dynamically controllable
- Two options
  - Either continue down this branch and backtrack later if necessary, or
  - Extend $PC$ to detect more cases where $(\mathcal{V}, \mathcal{V}', \mathcal{E}, \mathcal{E}')$ isn’t dynamically controllable

**PC($\mathcal{V}, \mathcal{E}$):**

for $1 \leq k \leq n$ do

for $1 \leq i < j \leq n$, $i \neq k$, $j \neq k$ do

$r_{ij} \leftarrow r_{ij} \cap [r_{ik} \cdot r_{kj}]$

if $r_{ij} = \emptyset$ then

return inconsistent
Additional Constraint Propagation Rules

- Case 1: $u \geq 0$
  - $t$ must come before $t_e$
  - Add a composition constraint $[a', b']$
  - Find $[a', b']$ such that $[a', b'] \cdot [u, v] = [a, b]$
    - $[a'+u, b'+v] = [a, b]$
    - $a' = a - u, \ b' = b - v$

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Propagated constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e, \ t \xrightarrow{[u,v]} t_e, \ u \geq 0$</td>
<td>$t_s \xrightarrow{[b',a']} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e, \ t \xrightarrow{[u,v]} t_e, \ u &lt; 0, \ v \geq 0$</td>
<td>$t_s \xrightarrow{\langle t_e, b' \rangle} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e, \ t_s \xrightarrow{\langle t_e, u \rangle} t$</td>
<td>$t_s \xrightarrow{\min{a, u}, \infty} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e, b \rangle} t, \ t' \xrightarrow{[u,v]} t$</td>
<td>$t_s \xrightarrow{\langle t_e, b' \rangle} t'$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e, b \rangle} t, \ t' \xrightarrow{[u,v]} t, \ t_e \neq t$</td>
<td>$t_s \xrightarrow{\langle t_e, b-u \rangle} t'$</td>
</tr>
</tbody>
</table>

contingent $\Rightarrow$ controllable $\rightarrow$

$a' = a - u, \ b' = b - v$
Additional Constraint Propagation Rules

- Case 2: $u < 0$ and $v \geq 0$
  - $t$ may be either before or after $t_e$
- Add a wait constraint
  - Wait until either $t_e$ occurs or current time is $t_s + b - v$, whichever comes first

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Propagated constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \land t \xrightarrow{[u,v]} t_e, u \geq 0$</td>
<td>$t_s \xrightarrow{[b',a']} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \land t \xrightarrow{[u,v]} t_e, u &lt; 0, v \geq 0$</td>
<td>$t_s \xrightarrow{\langle t_e,b' \rangle} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \land t_s \xrightarrow{\langle t_e,u \rangle} t$</td>
<td>$t_s \xrightarrow{[\min{a,u},\infty]} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e,b \rangle} t, t' \xrightarrow{[u,v]} t$</td>
<td>$t_s \xrightarrow{\langle t_e,b' \rangle} t'$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e,b \rangle} t, t' \xrightarrow{[u,v]} t, t_e \neq t$</td>
<td>$t_s \xrightarrow{\langle t_e,b-u \rangle} t'$</td>
</tr>
</tbody>
</table>

contingent $\Rightarrow$ controllable $\rightarrow a' = a - u, b' = b - v$
**Extended Version of PC**

- We want a fast algorithm that TemPlan can run at each node, to decide whether to backtrack
- There’s an extended version of PC that runs in polynomial time, but it has high overhead
- Possible compromise: use ordinary PC most of the time
  - Run extended version occasionally, or at end of search before returning plan

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Propagated constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \text{, } t \xrightarrow{[u,v]} t_e \text{, } u \geq 0$</td>
<td>$t_s \xrightarrow{[b',a']} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \text{, } t \xrightarrow{[u,v]} t_e \text{, } u &lt; 0 \text{, } v \geq 0$</td>
<td>$t_s \xrightarrow{\langle t_e,b' \rangle} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e \text{, } t_s \xrightarrow{\langle t_e,u \rangle} t$</td>
<td>$t_s \xrightarrow{[\min{a,u},\infty]} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e,b \rangle} t \text{, } t' \xrightarrow{[u,v]} t$</td>
<td>$t_s \xrightarrow{\langle t_e,b' \rangle} t'$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e,b \rangle} t \text{, } t' \xrightarrow{[u,v]} t \text{, } t_e \neq t$</td>
<td>$t_s \xrightarrow{\langle t_e,b-u \rangle} t'$</td>
</tr>
</tbody>
</table>
Outline

✓ Introduction
✓ Representation
✓ Temporal planning
✓ Consistency and controllability

● 4.5 Acting with temporal models
  ➢ Acting with atemporal refinement
  ➢ Dispatching
  ➢ Observation actions
Atemporal Refinement of Primitive Actions

- Templan’s action templates may correspond to compound tasks
  - In RAE, refine into commands with refinement methods

- Action template (descriptive model)
  \[ \text{leave}(r, d, w) \]
  \[
  \begin{align*}
  \text{assertions: } & [t_s, t_e] \text{loc}(r) := (d, w) \\
  & [t_s, t_e] \text{occupant}(d) := (r, \text{empty}) \\
  \text{constraints: } & t_e \leq t_s + \delta_1 \\
  & \text{adjacent}(d, w)
  \end{align*}
  \]

- Refinement method (operational model)

  \[ \text{m-leave}(r, d, w, e) \]
  \[
  \begin{align*}
  \text{task: } & \text{leave}(r, d, w) \\
  \text{pre: } & \text{loc}(r) = d, \text{adjacent}(d, w), \text{exit}(e, d, w) \\
  \text{body: } & \text{until empty}(e) \text{ wait}(1) \\
  & \text{goto}(r, e)
  \end{align*}
  \]
Atemporal Refinement of Primitive Actions

- Templan’s action templates may correspond to compound tasks
  - In RAE, refine into commands with refinement methods

- Action template (descriptive model)
  - unstack\((k, c, p)\)
    - assertions: …
    - constraints: …

- Refinement method (operational model)
  - m-unstack\((k, c, p)\)
    - task: unstack\((k, c, p)\)
    - pre: \(\text{pos}(c) = p\), \(\text{top}(p) = c\), \(\text{grip}(k) = \text{empty}\)
      - attached\((k, d)\), attached\((p, d)\)
    - body: locate-grasp-position\((k, c, p)\)
      - move-to-grasp-position\((k, c, p)\)
      - grasp\((k, c, p)\)
      - until firm-grasp\((k, c, p)\) ensure-grasp\((k, c, p)\)
      - lift-vertically\((k, c, p)\)
      - move-to-neutral-position\((k, c, p)\)
Discussion

- **Pros**
  - Simple online refinement with RAE
  - Avoids breaking down uncertainty of contingent duration
  - Can be augmented with temporal monitoring functions in RAE
    - E.g., watchdogs, methods with duration preferences

- **Cons**
  - Does not handle temporal requirements at the command level,
    - e.g., synchronize two robots that must act concurrently

- Can augment RAE to include temporal reasoning
  - Call it eRAe
  - One essential component: a *dispatching* function
Acting With Temporal Models

- Dispatching procedure: a dynamic execution strategy
  - Controls when to start each action
  - Given a dynamically controllable plan with executable primitives, triggers corresponding commands from online observations

- Example
  - robot r2 needs to leave dock d2 before robot r1 can enter d2
  - crane k needs to uncover c then put c onto r1

---

Example:

- robot r2 needs to leave dock d2 before robot r1 can enter d2
- crane k needs to uncover c then put c onto r1
Dispatching

- Let $(V, \tilde{V}, E, \tilde{E})$ be a controllable STNU that’s grounded
- Different from a grounded expression in logic
  - At least one time point $t$ is instantiated
- This bounds each time point $t$ within an interval $[l_t, u_t]$

Controllable time point $t$ in the future:
- $t$ is *alive* if current time $now \in [l_t, u_t]$
- $t$ is *enabled* if
  - it’s alive
  - for every precedence constraint $t' < t$, $t'$ has occurred
  - for every wait constraint $\langle t_e, \alpha \rangle$, $t_e$ has occurred or $\alpha$ has expired

Dispatch $(V, \tilde{V}, E, \tilde{E})$
- initialize the network
- while there are time points in $V$ that haven’t been triggered, do
  - update *now*
  - update the time points in $\tilde{V}$ that were triggered since the last iteration
  - update *enabled*
  - trigger every $t \in enabled$ such that $now = u_t$
  - arbitrarily choose other time points in *enabled*, and trigger them
  - propagate values of triggered timepoints (change $[l_t, u_t]$ for each future timepoint $t$)
**Example**

- trigger $t_1$, observe leave finish
- enable and trigger $t_2$, this enables $t_3$, $t_4$
- trigger $t_3$ soon enough to allow enter($r_1,d_2$) at time $t_5$
- trigger $t_4$ soon enough to allow stack($k,c'$) at time $t_6$
- rest of plan is linear: choose each $t_i$ after the previous action ends

---

**Dispatch($\mathcal{V},\mathcal{V},\mathcal{E},\mathcal{E}$)**

- initialize the network
- while there are time points in $\mathcal{V}$ that haven’t been triggered, do
  - update *now*
  - update the time points in $\mathcal{V}$ that were triggered since the last iteration
  - update *enabled*
  - trigger every $t \in \text{enabled}$ such that $\text{now} = u_t$
  - arbitrarily choose other time points in *enabled*, and trigger them
  - propagate values of triggered timepoints (change $[l_t, u_t]$ for each future timepoint $t$)

---

What about a more detailed example?
Example from slide 21

\[ \text{Dispatch}(\mathcal{V}, \mathcal{\bar{V}}, \mathcal{E}, \mathcal{\bar{E}}) = \]

- initialize the network
- while there are time points in \( \mathcal{V} \) that haven’t been triggered, do
  - update \textit{now}
  - update the time points in \( \mathcal{\bar{V}} \) that were triggered since the last iteration
  - update \textit{enabled}
  - trigger every \( t \in \text{enabled} \) such that \( \text{now} = u_t \)
  - arbitrarily choose other time points in \textit{enabled}, and trigger them
  - propagate values of triggered timepoints (change \([l_t, u_t]\) for each future timepoint \( t \))

- trigger \( t_1 \) at time 0
- wait and observe \( t \); this enables \( t \)
- trigger \( t' \) at any time from \( t \) to \( t+5 \)
- trigger \( t_3 \) at time \( t' + 10 \)
  \( t_2 \in [t' + 15, t' + 20] \)
  \( t_4 \in [t_3 + 5, t_3 + 10] = [t' + 15, t' + 20] \)
- so \( t_4 - t_3 \in [-5, 5] \)
Example from slide 21

- trigger $t_1$ at time 0
- wait and observe $t$; this enables $t$
- trigger $t'$ at any time from $t$ to $t+5$
- trigger $t_3$ at time $t' + 10$
  
  $t_2 \in [t' + 15, t' + 20]$
  $t_4 \in [t_3 + 5, t_3 + 10] = [t' + 15, t' + 20]$
  
  so $t_4 - t_3 \in [-5, 5]$

To use the extended PC algorithm here, I think we would need additional rule(s)
• Current ones seem inapplicable

Dispatch($\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}}$)
- initialize the network
- while there are time points in $\mathcal{V}$ that haven’t been triggered, do
  - update $now$
  - update the time points in $\tilde{\mathcal{V}}$ that were triggered since the last iteration
  - update $enabled$
  - trigger every $t \in enabled$ such that $now = u_t$
  - arbitrarily choose other time points in $enabled$, and trigger them
  - propagate values of triggered timepoints (change $[l_{t}, u_{t}]$ for each future timepoint $t$)

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Propagated constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e$, $t \xrightarrow{[u,v]} t_e$, $u \geq 0$</td>
<td>$t_s \xrightarrow{[b',a']} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e$, $t \xrightarrow{[u,v]} t_e$, $u &lt; 0$, $v \geq 0$</td>
<td>$t_s \xrightarrow{\langle t_e, b' \rangle} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{[a,b]} t_e$, $t_s \xrightarrow{\langle t_e, u \rangle} t$</td>
<td>$t_s \xrightarrow{[\min{a,u}, \infty]} t$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e, b \rangle} t$, $t \xrightarrow{[u,v]} t$</td>
<td>$t_s \xrightarrow{\langle t_e, b' \rangle} t'$</td>
</tr>
<tr>
<td>$t_s \xrightarrow{\langle t_e, b \rangle} t$, $t \xrightarrow{[u,v]} t$, $t_e \neq t$</td>
<td>$t_s \xrightarrow{\langle t_e, b - u \rangle} t'$</td>
</tr>
</tbody>
</table>
Deadline Failures

- Suppose something makes it impossible to start an action on time
- Do one of the following:
  - stop the delayed action, and look for new plan
  - let the delayed action finish; try to repair the plan by resolving violated constraints at the STNU propagation level
    - e.g., accommodate a delay in `navigate` by delaying the whole plan
  - let the delayed action finish; try to repair the plan some other way

\[
\begin{align*}
&\text{navigate}(r1) \\
&\text{leave}(r1,d1) \\
&\text{stack}(k,c',q) \\
&\text{unstack}(k,c,p) \\
&\text{stack}(k,c',q) \\
&\text{navigate}(r1) \\
&\text{enter}(r1,d2) \\
&\text{unstack}(k,c) \\
&\text{putdown}(k,c,r1) \\
&\text{leave}(r1,d2)
\end{align*}
\]
Partial Observability

- Tacit assumption: all occurrences of contingent events are observable
  - Observation needed for dynamic controllability
  - In general not all events are observable
- POSTNU (Partially Observable STNU)

- Dynamically controllable?
Observation Actions

- Controllable
- Contingent
  - Invisible
  - Observable

Actions:
- working
- driving
- cooking

Times:
- $t_0$: [19:00, 19:30]
- $t_1$: [1, 2]
- $t'$: [20, 25]
- $t_2$: [-5, 10]
- $t_3$: [25, 30]
- $t_4$
A POSTNU is dynamically controllable if

- there exists an execution strategy that chooses future controllable points to meet all the constraints, given the observation of past visible points
- Observable ≠ visible
- Observable means it will be known *when observed*
- It can be temporarily hidden
Outline

✓ Introduction
✓ Representation
✓ Temporal planning
✓ Consistency and controllability
✓ Acting with executable primitives

● Summary
Summary

- Managing constraints in TemPlan: like CSPs
  - Temporal constraints: STNs, PC algorithm (path consistency)

- Acting
  - Dynamic controllability
  - STNUs
  - RAE and eRAE
  - Dispatching