Chapter 4
Deliberation with Temporal Domain Models

Section 4.1: Introduction
Section 4.2: Temporal Representation
Section 4.3: Planning

Dana S. Nau
University of Maryland
Temporal Models

- Durations of actions
- Delayed effects and preconditions
  - e.g., resources borrowed or consumed during an action
- Time constraints on goals
  - relative or absolute
- Exogenous events expected to occur in the future
  - when?
- Maintenance actions:
  - maintain a property (≠ changing a value)
  - e.g., track a moving target, keep a spring latch in position
- Concurrent actions
  - interacting effects, joint effects
- Delayed commitment
  - instantiation at acting time
• LAAS/CNRS, Toulouse, France
• mid-1990s
RAX/PS

- Planning/control of DS1 spacecraft
- NASA Ames and JPL, 1999
T-ReX

- Planning/control of AUVs
- Monterey Bay Aquarium Research Institute, ≈ 2005-2010

Casper (NASA JPL)

- Planning/control of spacecraft
- NASA JPL, ongoing
Timelines

- Up to now, we’ve used a “state-oriented view”
  - Time is a sequence of states $s_0, s_1, s_2$
  - Instantaneous actions transform each state into the next one
  - No overlapping actions

- Switch to a “time-oriented view”
  - Sequence of integer time points
    - $t = 1, 2, 3, \ldots$
  - For each state variable $x$, a timeline
    - values during different time intervals
  - State at time $t = \{\text{state-variable values at time } t\}$
Timelines

- Sets of constraints on state variables and events
  - Reflect predicted actions and events
- Planning is constraint-based
Outline

✓ Introduction

• 4.2 Representation
  ➢ Timelines
  ➢ Actions and tasks
  ➢ Chronicles

• 4.3 Temporal planning

• 4.4 Constraint management

• 4.5 Acting with temporal models
Representation

- Quantitative model of time
  - Discrete: time points are integers

- Expressions:
  - time-point variables \( t, t', t_2, t_j, \ldots \)
  - simple constraints \( d \leq t' - t \leq d' \)

- Temporal assertion:
  - value of a state variable during a time interval
    - persistence: \([t_1, t_2] \ x = \nu\) entails \( t_1 < t_2 \)
    - change: \([t_1, t_2] \ x : (\nu_1, \nu_2)\) entails \( \nu_1 \neq \nu_2 \)
Timeline

- A pair \((\mathcal{T}, C)\)
  - partially predicted evolution of one state variable
  - \(\mathcal{T}\) : temporal assertions
    \[ [t_1, t_2] \text{loc}(r1) : (\text{loc1}, l) \]
    \[ [t_2, t_3] \text{loc}(r1) = l \]
    \[ [t_3, t_4] \text{loc}(r1) : (l, \text{loc2}) \]
  - \(C\) : constraints
    \[ t_1 < t_2 < t_3 < t_4 \]
    \[ l \neq \text{loc1} \]
    \[ l \neq \text{loc2} \]
- If we want to restrict \(\text{loc}(r1)\) during \([t_1, t_2]\)
  \[ [t_1, t_1+1] \text{loc}(r1) : (\text{loc1}, \text{route}) \]
  \[ [t_2-1, t_2] \text{loc}(r1) : (\text{route}, l) \]
  \[ [t_1+1, t_2-1] \text{loc}(r1) = \text{route} \]
Actions

• Preliminaries:
  - Timelines \((T_1, C_1), \ldots, (T_k, C_k)\) for \(k\) different state variables
  - Their union:
    - \((T_1, C_1) \cup \ldots \cup (T_k, C_k) = (T_1 \cup \ldots \cup T_k, C_1 \cup \ldots \cup C_k)\)
  - If every \((T_i, C_i)\) is secure, and
    - no pair of timelines \((T_i, C_i)\) and \((T_j, C_j)\) have any unground variables in common
    then \((T_1, C_1) \cup \ldots \cup (T_k, C_k)\) is also secure

• *Action or primitive task* (or just *primitive*):
  - a triple \((\text{head}, T, C)\)
    - *head* is the name and arguments
    - \((T, C)\) is the union of a set of timelines
Actions

- **leave**(\(r,d,w\))
  - robot \(r\) leaves dock \(d\),
  - goes to adjacent waypoint \(w\)

**leave**(\(r,d,w\))

assertions:

\[ [t_s,t_e] \text{ loc}(r): (d,w) \]
\[ [t_s,t_e] \text{ occupant}(d): (r,\text{empty}) \]

constraints:

\[ t_e \leq t_s + \delta_1 \]
\[ \text{adjacent}(d,w) \]

- loc\((r)\) changes to \(w\) with delay \(\leq \delta_1\)
- dock \(d\) becomes empty

- Two additional parameters
  - starting time \(t_s\), ending time \(t_e\)

- No separate preconditions and effects
  - preconditions \(\iff\) need for causal support
**Actions**

- enter\((r,d,w)\)
  - \(r\) enters \(d\) from an adjacent waypoint \(w\)

**enter\((r,d,w)\)**

assertions:

\[
[t_s, t_e] \ \text{loc}(r): (w,d) \\
[t_s, t_e] \ \text{occupant}(d): (\text{empty}, r)
\]

constraints:

\[t_e \leq t_s + \delta_2\]

- adjacent\((d,w)\)

- \(\text{loc}(r)\) changes to \(d\) with delay \(\leq \delta_2\)
- dock \(d\) becomes occupied by \(r\)

- Two additional parameters
  - starting time \(t_s\), ending time \(t_e\)

- No separate preconditions and effects
  - preconditions \(\Leftrightarrow\) need for causal support
Actions

- **take**($k,c,r,d$)
  - Action: crane $k$ takes container $c$ from $r$

- Two additional parameters
  - starting time $t_s$, ending time $t_e$

- No separate preconditions and effects
  - preconditions ⇔ need for causal support

**take**($k,c,r,d$)

**assertions:**

- $[t_s, t_e]$ pos($c$): ($r$, $k$)  // where container $c$ is
- $[t_s, t_e]$ grip($k$): (empty, $c$)  // what crane $k$’s gripper is holding
- $[t_s, t_e]$ freight($r$): ($c$, empty)  // what $r$ is carrying
- $[t_s, t_e]$ loc($r$) = $d$  // where $r$ is

**constraints:**

- attached($k,d$)
## Actions

- **leave**\( (r,d,w) \)  
  robot \( r \) leaves dock \( d \) to an adjacent waypoint \( w \)

- **enter**\( (r,d,w) \)  
  \( r \) enters \( d \) from an adjacent waypoint \( w \)

- **take**\( (k,c,r) \)  
  crane \( k \) takes container \( c \) from \( r \)

- **navigate**\( (r,w,w') \)  
  \( r \) navigates from waypoint \( w \) to \( w' \)

- **stack**\( (k,c,p) \)  
  crane \( k \) stacks container \( c \) on top of pile \( p \)

- **unstack**\( (k,c,p) \)  
  crane \( k \) takes a container \( c \) from top of \( p \)

- **put**\( (k,c,r) \)  
  crane \( k \) puts container \( c \) onto robot \( r \)

*Note: Book omits \( r \).*
Tasks and Methods

- Task: move robot $r$ to dock $d$
  - $[t_s, t_e] \text{ move}(r,d)$

- Method:
  m-move1($r,d,d',w,w'$)
  - task: move($r,d$)
  - refinement:
    - $[t_s, t_1] \text{ leave}(r,d',w')$
    - $[t_2, t_3] \text{ navigate}(r,w',w)$
    - $[t_4, t_e] \text{ enter}(r,d,w)$
  - assertions:
    - $[t_s, t_s+1] \text{ loc}(r) = d'$
  - constraints:
    - adjacent($d,w$),
    - adjacent($d',w'$), $d \neq d'$,
    - connected($w,w'$),
    - $t_1 \leq t_2$, $t_3 \leq t_4$

- $d'$ becomes empty during $[t_s, t_1]$
  - another robot may enter it after $t_1$
- $d$ doesn’t need to be empty until $t_4$
  - when $r$ starts entering it
Tasks and Methods

- **Task:** remove everything above container \(c\) in pile \(p\)
  - \([t_s, t_e]\) uncover\((c,p)\)

- **Method:**
  - m-uncover\((c,p,k,d,p')\)
    - task: uncover\((c,p)\)
    - refinement: \([t_s, t_1]\) unstack\((k,c',p)\)  // action
      \([t_2, t_3]\) stack\((k,c',p')\)  // action
      \([t_4, t_e]\) uncover\((c,p)\)  // recursive uncover
    - assertions: \([t_s, t_s+1]\) pile\((c) = p\)
      \([t_s, t_s+1]\) top\((p) = c'\)
      \([t_s, t_s+1]\) grip\((k) = \text{empty}\)
    - constraints: attached\((k,d)\), attached\((p,d)\),
      attached\((p',d)\),
      \(p \neq p', c' \neq c\),
      \(t_1 \leq t_2, t_3 \leq t_4\)
Tasks and Methods

- Task: robot $r$ brings container $c$ to pile $p$
  - $[t_s, t_e]$ bring($r, c, p$)

$m$-bring($r, c, p, p', d, d'$)
- task: bring($r, c, p$)
- refinement: $[t_s, t_1]$ move($r, d'$)
  - $[t_s, t_2]$ uncover($c, p'$)
  - $[t_3, t_4]$ load($k', r, c, p'$)
  - $[t_5, t_6]$ move($r, d$)
  - $[t_7, t_e]$ unload($k, r, c, p$)

assertions: $[t_s, t_3]$ pile($c$) = $p'$
- $[t_s, t_3]$ freight($r$) = empty

constraints: attached($p', d'$), attached($p, d$), $d \neq d'$
- attached($k', d'$), attached($k, d$), $k \neq k'$
- $t_1 \leq t_3$, $t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$
Chronicles

- Chronicle $\phi = (\mathcal{A}, S, \mathcal{T}, C)$
  - $\mathcal{A}$: temporally qualified actions and tasks
  - $S$: a priori supported assertions
  - $\mathcal{T}$: temporally qualified assertions
  - $C$: constraints

- $\phi$ can include
  - Current state, future predicted events
  - Tasks to perform
  - Assertions and constraints to satisfy

- Can represent
  - a planning problem
  - a plan or partial plan

$\phi_0$:

- **Tasks:**
  - $[t_0, t_1]$ bring($r; c1, d4$)

- **Supported:**
  - $[t_s]$ loc($r1$)=d1
  - $[t_s]$ loc($r2$)=d2
  - $[t_s+10, t_s+\delta]$ docked(ship1)=d3
  - $[t_s]$ top(pile-ship1)=c1
  - $[t_s]$ pos(c1)=pallet

- **Assertions:**
  - $[t_e]$ loc($r1$)=d1
  - $[t_e]$ loc($r2$)=d2

- **Constraints:**
  - $0 < t < t' < t_e$, $20 \leq \delta \leq 30$
Outline

✓ Introduction

✓ Representation

● 4.3 Temporal planning
  ➢ Resolvers and flaws
  ➢ Search space

● 4.4 Constraint management

● 4.5 Acting with temporal models
Planning

- Planning problem:
  - a chronicle $\phi_0$ that has some flaws
  - analogous to flaws in PSP
- Add new assertions, constraints, actions to resolve the flaws

$\phi_0$: tasks: \textit{(none)}
supported: \textit{(none)}
assertions: $[t_1,t_2]$ $\text{loc}(r1) = l$
  $[t_3,t_4]$ $\text{loc}(r1) : (\text{loc3,loc4})$
constraints: $\text{adjacent(loc3,w1)}$
  $\text{adjacent(w1,loc3)}$
  $\text{adjacent(loc4,w2)}$
  $\text{adjacent(w2,loc4)}$
  $\text{connected(w1,w2)}$

$\phi_0$: tasks: $[t_2,t_3]$ $\text{move(r1,loc3)}$
supported: \textit{(none)}
assertions: $[t_1,t_2]$ $\text{loc}(r1) = l$
  $[t_3,t_4]$ $\text{loc}(r1) : (\text{loc3,loc4})$
constraints: $\text{adjacent(loc3,w1)}$
  $\text{adjacent(w1,loc3)}$
  $\text{adjacent(loc4,w2)}$
  $\text{adjacent(w2,loc4)}$
  $\text{connected(w1,w2)}$
Flaws (1)

1. Temporal assertion $\alpha$ that isn’t *causally supported*
   - What causes $r1$ to be at $\text{loc3}$ at time $t_3$?

   - *Resolvers:*
     - Add constraints to support $\alpha$ from an assertion in $\phi$
       - $l = \text{loc3}, \ t_2 = t_3$
     - Add a new persistence assertion to support $\alpha$
       - $l = \text{loc3}, \ [t_2,t_3] \ loc(r1) = \text{loc3}$
     - Add a new task or action to support $\alpha$
       - $[t_2,t_3] \ \text{move}(r1,\text{loc3})$
         - refining it will produce support for $\alpha$ (see next slide)
2. Non-refined task

- **Resolver**: refinement method
  - Applicable if it matches the task and its constraints are consistent with \( \phi \)'s

- Applying the resolver:
  - Modify \( \phi \) by replacing the task with \( m \)

- Example: \([t_2, t_3]\) move(r1,loc3)
  - Refinement will replace it with something like
    - \([t_2, t_5]\) leave(r1,l,w)
    - \([t_5, t_6]\) navigate(r,w,w')
    - \([t_6, t_3]\) enter(r1,loc3,w')
    - plus constraints

Like a task in SeRPE
Flaws (3)

3. A pair of possibly-conflicting temporal assertions
   - temporal assertions $\alpha$ and $\beta$ *possibly conflict* if they can have inconsistent instances
     e.g., $[t_1,t_2] \text{loc}(r1) = \text{loc1},$ $[t_3,t_4] \text{loc}(r) : (l, l')$
     \[\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \downarrow \]
     instance: $[1,5] \text{loc}(r1) = \text{loc1},$ $[3,8] \text{loc}(r1) : (\text{loc2},\text{loc3})$
   
   - *Resolvers*: separation constraints
     - $r \neq r1$
     - $t_2 < t_3$
     - $t_4 < t_1$
     - $t_2 = t_3,$ $r = r1,$ $l = \text{loc1}$
       - Also provides causal support for $[t_3,t_4] \text{loc}(r) : (l, l')$
     - $t_4 = t_1,$ $r = r1,$ $l = \text{loc1}$
       - Also provides causal support for $[t_1,t_2] \text{loc}(r1) = \text{loc1}$

Like a threat in PSP
Planning Algorithm

- Like PSP in Chapter 2
  - Repeatedly selects flaws and chooses resolvers
- In the book, TemPlan uses recursion
  - Can be rewritten to use a loop
  - Just programming style, equivalent either way
- In a deterministic implementation
  - Selecting a resolver $\rho$ is a backtracking point
  - Selecting a flaw isn’t
- If it’s possible to resolve all flaws, at least one of the nondeterministic execution traces will do so

```
TemPlan($\phi, \Sigma$)

Flaws $\leftarrow$ set of flaws of $\phi$
if Flaws=$\emptyset$ then return $\phi$
arbitrarily select $f \in$ Flaws
Resolvers $\leftarrow$ set of resolvers of $f$
if Resolvers=$\emptyset$ then return failure
nondeterministically choose $\rho \in$ Resolvers
$\phi \leftarrow$ Transform($\phi, \rho$)
Templan($\phi, \Sigma$)
```

```
TemPlan($\phi, \Sigma$)

Flaws $\leftarrow$ set of flaws of $\phi$
if Flaws=$\emptyset$ then return $\phi$
arbitrarily select $f \in$ Flaws
Resolvers $\leftarrow$ set of resolvers of $f$
if Resolvers=$\emptyset$ then return failure
nondeterministically choose $\rho \in$ Resolvers
$\phi \leftarrow$ Transform($\phi, \rho$)
```
Example

- $\phi_0 = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
  - Establishes state-variable values at time $t = 0$
  - Flaws: two unrefined tasks
    - $\text{bring}(r,c_1,p_3), \text{bring}(r',c_2,p_4)$

\[ \phi_0: \text{tasks:} \begin{align*}
\text{bring}(r,c_1,p_3) \\
\text{bring}(r',c_2,p_4)
\end{align*} \]

supported: [0] $\text{loc}(r_1)=d_3$
- [0] $\text{freight}(r_1)=\text{empty}$
- [0] $\text{pile}(c_1)=p_1'$
- [0] $\text{pile}(c_1')=p_1'$
- [0] $\text{pos}(c_1)=\text{pallet}$
- [0] $\text{pos}(c_1')=c_1$

\[
\ldots
\]

assertions: (none)

constraints:

- $\text{adjacent}(d_1,w_{12})$
- $\text{adjacent}(d_1,w_{13})$

\[
\ldots
\]
Example

- $\phi_0 = (A, S, T, C)$
  - Establishes state-variable values at time $t = 0$
  - Flaws: two unrefined tasks

  - $\text{bring}(r,c1,p3)$, $\text{bring}(r',c2,p4)$

$m$-bring$(r,c,p,p',d,d',k,k')$

- task: $\text{bring}(r,c,p)$
- refinement: $[t_s, t_1]$ move$(r,d')$
  $[t_s, t_2]$ uncover$(c,p')$
  $[t_3, t_4]$ load$(k',r,c,p')$
  $[t_5, t_6]$ move$(r,d)$
  $[t_7, t_e]$ unload$(k,r,c,p)$

- assertions: $[t_s, t_3]$ pile$(c) = p'$
  $[t_s, t_3]$ freight$(r) = \text{empty}$

- constraints: attached$(p',d')$, attached$(p,d)$, $d \neq d'$
  attached$(k',d')$, attached$(k,d)$, $k \neq k'$
  $t_1 \leq t_3$, $t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$

$\phi_0$: tasks: $\text{bring}(r,c1,p3)$
  $\text{bring}(r',c2,p4)$

- supported: $[0]$ loc$(r1) = d3$
  $[0]$ freight$(r1) = \text{empty}$
  $[0]$ pile$(c1) = p'1$
  $[0]$ pile$(c1) = p'1$
  $[0]$ pos$(c1) = \text{pallet}$
  $[0]$ pos$(c'1) = c1$

- assertions: (none)
- constraints:
  - adjacent$(d1,w12)$
  - adjacent$(d1,w13)$

...
Method instance

- Instantiate $c = c_1$ and $p = p_3$ to match $\text{bring}(r, c_1, p_3)$
- I don’t think $p', d, d', k, k'$ should be instantiated yet, but I did it to match the example in the book
  - Needed later to satisfy action preconditions

\[ \phi_0: \text{tasks: bring}(r, c_1, p_3) \]
\[ \text{bring}(r', c_2, p_4) \]

supported:
- $[0]$ loc($r_1$) = $d_3$
- $[0]$ freight($r_1$) = empty
- $[0]$ pile($c_1$) = $p_1$
- $[0]$ pile($c_1'$) = $p_1$
- $[0]$ pos($c_1$) = pallet
- $[0]$ pos($c_1'$) = $c_1$

... 

assertions: (none)

constraints:
- adjacent($d_1$, $w_{12}$)
- adjacent($d_1$, $w_{13}$)

...
**Modified Chronicle**

- Changes to $\phi_0$
  - Removed $\text{bring}(r,c1,p3)$
  - Added 5 tasks, 2 assertions, 4 constraints
- Flaws
  - 6 unrefined tasks, 2 unsupported assertions

$\phi_1$: tasks:

- $[t_s,t_1]$ move($r$; $d1$)
- $[t_s,t_2]$ uncover($c1$; $p'1$)
- $[t_3,t_4]$ load($k1$; $r$; $c1$; $p'1$)
- $[t_5,t_6]$ move($r$; $d3$)
- $[t_7,t_e]$ unload($k3$; $r$; $c1$; $p3$)
- $\text{bring}(r',c2,p4)$

supported:

- [0] loc($r1$)=d3
- [0] freight($r1$)=empty
- [0] pile($c1$)=p’1
- [0] pile($c'1$)=p’1
- [0] pos($c1$)=pallet
- [0] pos($c'1$)=c1

... assertions:

- $[t_s,t_3]$ pile($c1$) = p’1
- $[t_s,t_3]$ freight($r$) = empty

constraints:

- $t_s < t_1 \leq t_3$, $0 < t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$,
- adjacent($d1$,w12),
- adjacent($d1$,w13),

...
Method instance

- Instantiate \( r=r' \), \( c=c2 \), \( p=p4 \) to match bring\((r',c2,p4)\)
- I don’t think \( p',d,d',k,k' \) should be instantiated yet, but I’ve done so to match the book
- Other variables renamed to avoid name conflicts

\( m\text{-bring}(r,c,p,p',d,d',k,k') \)

- **task**: bring\((r,c,p)\)
- **refinement**: 
  - \([t_s,t_1] \text{ move}(r,d')\)
  - \([t_s,t_2] \text{ uncover}(c,p')\)
  - \([t_3,t_4] \text{ load}(k',r,c,p')\)
  - \([t_5,t_6] \text{ move}(r,d)\)
  - \([t_7,t_r] \text{ unload}(k,r,c,p)\)
- **assertions**: 
  - \([t_s,t_3] \text{ pile}(c)=p'\)
  - \([t_s,t_3] \text{ freight}(r)=\text{empty}\)
- **constraints**: 
  - adjacent\((d1,w12),d2\)
  - adjacent\((d1,w13),d2\)
  - \( t_1 \leq t_3, t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7 \)
  - \( t'_1 \leq t'_3, t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7 \)

\( \phi_1 \): tasks:

- \([t_s,t_1] \text{ move}(r,d1)\)
- \([t_s,t_2] \text{ uncover}(c1,p'1)\)
- \([t_3,t_4] \text{ load}(k1,r,c1,p'1)\)
- \([t_5,t_6] \text{ move}(r,d3)\)
- \([t_7,t_r] \text{ unload}(k3,r,c1,p3)\)
- bring\((r',c2,p4)\)

- supported: 
  - \([0] \text{ loc}(r1)=d3\)
  - \([0] \text{ freight}(r1)=\text{empty}\)
  - \([0] \text{ pile}(c1)=p'1\)
  - \([0] \text{ pile}(c'1)=p'1\)
  - \([0] \text{ pos}(c1)=\text{pallet}\)
  - \([0] \text{ pos}(c'1)=c1\)

- assertions: 
  - \([t_s,t_3] \text{ pile}(c1)=p'1\)
  - \([t_s,t_3] \text{ freight}(r)=\text{empty}\)

- constraints: 
  - \( t_s \leq t_1 \leq t_3, 0 \leq t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7, t'_1 \leq t'_3, t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7 \)

Poll: is it OK to instantiate \( r=r' \)?
**Modified Chronicle**

- Changes:
  - Removed `bring(r',c2,p4)`
  - Added 5 tasks, 2 assertions, 4 constraints
- Flaws
  - 10 unrefined tasks, 4 unsupported assertions
- Next, work on these two assertions

### \( \phi_2 \):
- **tasks:**
  - \([t_s,t_1]\) move\((r;d_1)\)
  - \([t_s,t_2]\) uncover\((c1,p'1)\)
  - \([t_3,t_4]\) load\((k1,r;c1,p'1)\)
  - \([t_5,t_6]\) move\((r; d3)\)
  - \([t_7,t_e]\) unload\((k3,r;c1,p3)\)
  - \([t'_s,t'_1]\) move\((r',d2)\)
  - \([t'_s,t'_2]\) uncover\((c2,p'2)\)
  - \([t'_3,t'_4]\) load\((k4,r',c2,p'2)\)
  - \([t'_5,t'_6]\) move\((r',d4)\)
  - \([t'_7,t'_e]\) unload\((k2,r',c2,p'2)\)

- **supported:**
  - \([0]\) loc\((r1)=d3\)
  - \([0]\) freight\((r1)=empty\)
  - \([0]\) pile\((c1)=p'1\)

- **assertions**
  - \([t_s,t_3]\) pile\((c1)=p'1\)
  - \([t_s,t_3]\) freight\((r)=empty\)
  - \([t'_s,t'_3]\) pile\((c2)=p'2\)
  - \([t'_s,t'_1]\) freight\((r')=empty\)

- **constraints:**
  - \(t_s < t_1 \leq t_3, t_s < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,\)
  - \(t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,\)
  - adjacent\((d1,w12)\),
  - adjacent\((d1,w13)\), . . .
Supporting the assertions

- 3 ways to support \([t_s, t_3] \text{pile}(c1)=p'1\)
  1. Constrain \(t_s = 0\) and use \([0] \text{pile}(c1)=p'1\)
  2. Add persistence \([0,t_s] \text{pile}(c1)=p1\)
  3. Add new action \([t_8,t_s] \text{put}(r'',c1, p'1)\)

Poll: Will any of them also provide support for \([t_s, t_3] \text{freight}(r) = \text{empty}\)?
- Vote 1, 2, 3, or 4 (to mean none of them)

\(\phi_2:\) tasks:
\[
\begin{align*}
[t_s,t_1] & \text{move}(r;d1) \\
[t_s,t_2] & \text{uncover}(c1,p'1) \\
[t_3,t_4] & \text{load}(k1,r;c1,p'1) \\
[t_5,t_6] & \text{move}(r; d3) \\
[t_7,t_{e1}] & \text{unload}(k3;r;c1,p3) \\
[t'_s,t'_1] & \text{move}(r',d2) \\
[t'_s,t'_2] & \text{uncover}(c2,p'2) \\
[t'_3,t'_4] & \text{load}(k4,r';c2,p'2) \\
[t'_5,t'_6] & \text{move}(r',d4) \\
[t'_7,t'_{e1}] & \text{unload}(k2,r';c2,p'2)
\end{align*}
\]

supported: \([0] \text{loc}(r1)=d3\) \\
\([0] \text{freight}(r1)=\text{empty}\) \\
\([0] \text{pile}(c1)=p'1\)

\ldots

assertions:
\[
\begin{align*}
[t_s,t_3] & \text{pile}(c1) = p'1 \\
[t_s,t_3] & \text{freight}(r) = \text{empty} \\
[t'_s,t'_3] & \text{pile}(c2) = p'2 \\
[t'_s,t'_1] & \text{freight}(r') = \text{empty}
\end{align*}
\]

constraints:
\[
t_s < t_1 \leq t_3, t_s < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7, \\
t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,
\]
adjacent(d1,w12),
adjacent(d1,w13), \ldots
Supporting the assertions

- To support \([t_s, t_3]\) pile(c1) = p'1
  - Constrain \(t_s = 0\) and use [0] pile(c1) = p'1
- To support [0, t_3] freight(r) = empty, constrain \(r = r_1\)

Poll: Should \(0 < t_1\) and \(0 < t_2\) have been there already?
1. yes  
2. no

\(\phi_3\): tasks:
- [0, t_1] move(r1, d1)
- [0, t_2] uncover(c1, p'1)
- [0, t_4] load(k1, r1, c1, p'1)
- [t_5, t_6] move(r1, d3)
- [t_7, t_e] unload(k3, r1, c1, p1)
- [t'_s, t'_1] move(r', d2)
- [t'_s, t'_2] uncover(c2, p'2)
- [t'_3, t'_4] load(k4, r', c2, p'2)
- [t'_5, t'_6] move(r', d4)
- [t'_7, t'_e] unload(k2, r', c2, p'2)

supported:
- [0] loc(r1) = d3
- [0] freight(r1) = empty
- [0] pile(c1) = p'1

assertions:
- [t'_s, t'_3] pile(c2) = p'2
- [t'_s, t'_1] freight(r') = empty

constraints:
- \(0 < t_1 \leq t_3, 0 < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7, t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,\)
- adjacent(d1, w12), adjacent(d1, w13), ...
Supporting the assertions

- To support \([t_s',t_3']\) \(\text{pile}(c2)=p'2\)
  - Add persistence condition \([0,t_s']\) \(\text{pile}(c2)=p'2\)
  - Alternatives: constrain \(t_s'=0\) or add new action \(\text{put}(r'',c2, p'2)\)
- To support \([t_s',t_1']\) \(\text{freight}(r') = \text{empty}\)
  - Constrain \(r = r2\), add \([0,t_s']\) \(\text{freight}(r2)=\text{empty}\)

\[\phi_3: \text{tasks:} \]
\[\begin{align*}
[0,t_1] & \text{move}(r1,d1) \\
[0,t_2] & \text{uncover}(c1,p'1) \\
[t_3,t_4] & \text{load}(k1,r1,c1,p'1) \\
[t_5,t_6] & \text{move}(r1,d3) \\
[t_7,t_e] & \text{unload}(k3,r1,c1,p1) \\
[t_s',t_1'] & \text{move}(r2,d2) \\
[t_s',t_2'] & \text{uncover}(c2,p'2) \\
[t_3',t_4'] & \text{load}(k4,r2,c2,p'2) \\
[t_5',t_6'] & \text{move}(r2,d4) \\
[t_7',t_e'] & \text{unload}(k2,r2,c2,p'2)
\end{align*}\]

supported: 
\[\begin{align*}
[0] & \text{loc}(r1)=d3 \\
[0] & \text{freight}(r1)=\text{empty} \\
[0] & \text{pile}(c1)=p'1
\end{align*}\]
\[
\ldots
\]
\[\begin{align*}
[0,t_3] & \text{pile}(c1) = p'1 \\
[0,t_3] & \text{freight}(r1) = \text{empty} \\
[0,t_s'] & \text{pile}(c2)=p'2 \\
[t_s',t_3'] & \text{pile}(c2) = p'2 \\
[0,t_s'] & \text{freight}(r2)=\text{empty} \\
[t_s',t_1'] & \text{freight}(r2) = \text{empty}
\end{align*}\]

assertions: (none)

constraints: 
\[0<t_1\leq t_3, 0<t_2\leq t_3, t_4\leq t_5, t_6\leq t_7, 0<t_s'<t_1\leq t_3', 0<t_s'<t_2\leq t_3', t_4\leq t_5', t_6\leq t_7', \ldots\]
Example of conflicts

- Refining tasks into actions will produce possibly-conflicting assertions
  - move(r2,d4) must go through d3
  - Conflict: occupant(d3)=r1, occupant(d3)=r2
- Resolvers: separation constraints to ensure r2 only goes through d3 while r1 away from d3

\[ \phi_3: \text{tasks: } [0,t_1] \text{ move(r1,d1)} \]
\[ [0,t_2] \text{ uncover(c1,p'1)} \]
\[ [t_3,t_4] \text{ load(k1,r1,c1,p'1)} \]
\[ [t_5,t_6] \text{ move(r1,d3)} \]
\[ [t_7,t_\epsilon] \text{ unload(k3,r1,c1,p)} \]
\[ [t'_s,t'_1] \text{ move(r2,d2)} \]
\[ [t'_s,t'_2] \text{ uncover(c2,p'2)} \]
\[ [t'_3,t'_4] \text{ load(k4,r2,c2,p'2)} \]
\[ [t'_5,t'_6] \text{ move(r2,d4)} \]
\[ [t'_7,t'_\epsilon] \text{ unload(k2,r2,c2,p'2)} \]

supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p'1

... 
[0,t_3] pile(c1)=p'1
[0,t_3] freight(r1)=empty
[0,t'_s] pile(c2)=p'2
[t'_s,t'_3] pile(c2)=p'2
[0,t'_s] freight(r2)=empty
[t'_s,t'_1] freight(r2)=empty

assertions: (none)

constraints: 0< t_1 \leq t_3, 0< t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7, 
0< t'_s \leq t'_3, 0< t'_s \leq t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7, ...
Heuristics for Guiding TemPlan

- Flaw selection, resolver selection heuristics similar to those in PSP
  - Select the flaw with the smallest number of resolvers
  - Choose the resolver that rules out the fewest resolvers for the other flaws

- There is also a problem with constraint management
  - We ignored it when discussing PSP
  - Discuss it next

\[ \text{TemPlan}(\phi, \Sigma) \]
\[ Flaws \leftarrow \text{set of flaws of } \phi \]
if \( Flaws = \emptyset \) then return \( \phi \)
arbitrarily select \( f \in Flaws \)
\[ Resolvers \leftarrow \text{set of resolvers of } f \]
if \( Resolvers = \emptyset \) then return failure
nondeterministically choose \( \rho \in Resolvers \)
\[ \phi \leftarrow \text{Transform}(\phi, \rho) \]
\[ \text{Templan}(\phi, \Sigma) \]

\[ \text{PSP}(\Sigma, \pi) \]
loop
if \( Flaws(\pi) = \emptyset \) then return \( \pi \)
arbitrarily select \( f \in Flaws(\pi) \)
\[ R \leftarrow \{ \text{all feasible resolvers for } f \} \]
if \( R = \emptyset \) then return failure
nondeterministically choose \( \rho \in R \)
\[ \pi \leftarrow \rho(\pi) \]
return \( \pi \)
Summary of Sections 4.1, 4.2, 4.3

- **Timelines**
  - Temporal assertions (change, persistence), constraints
  - Conflicts, consistency, security, causal support
- **Chronicle**: union of several timelines
  - Consistency, security, causal support
- **Actions** represented by chronicles; preconditions $\Leftrightarrow$ causal support
- **Planning problems**
  - Three kinds of flaws and their resolvers:
    - Tasks, causal support, security
  - Partial plans, solution plans
- **Planning**: TemPlan
  - Like PSP but with tasks, temporal assertions, temporal constraints