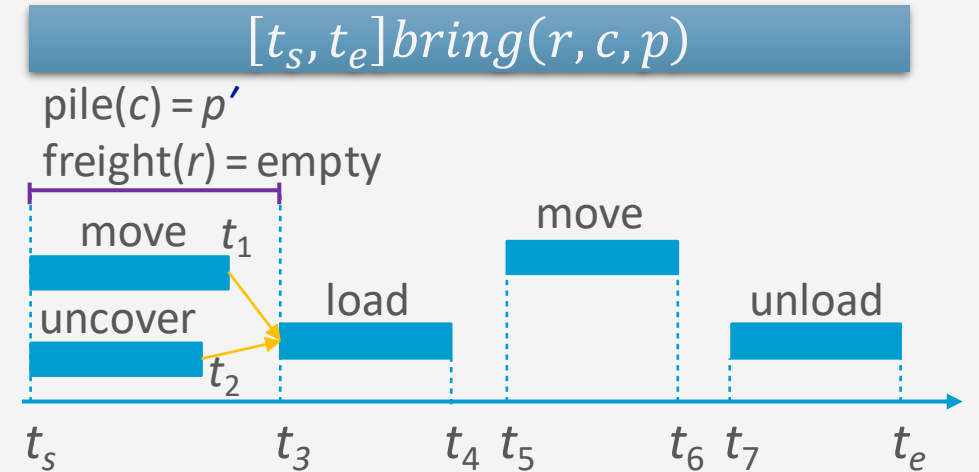


Automated Planning and Acting Temporal Models



Content: Planning and Acting

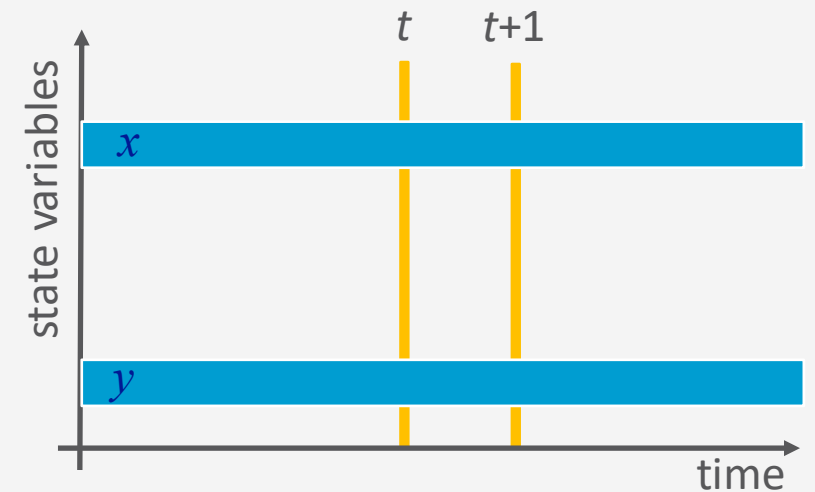
1. With **Deterministic** Models
2. Planning and Acting with **Temporal** Models
 - a. Temporal Representation
 - b. Planning with Temporal Refinement Methods
 - c. Constraint Management
 - d. Acting with Temporal Models
3. With **Nondeterministic** Models
4. With **Probabilistic** Models
5. By **Decision Making**
 - A. Foundations
 - B. Extensions
 - C. Structure
6. With **Human-awareness**

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 (\neq 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

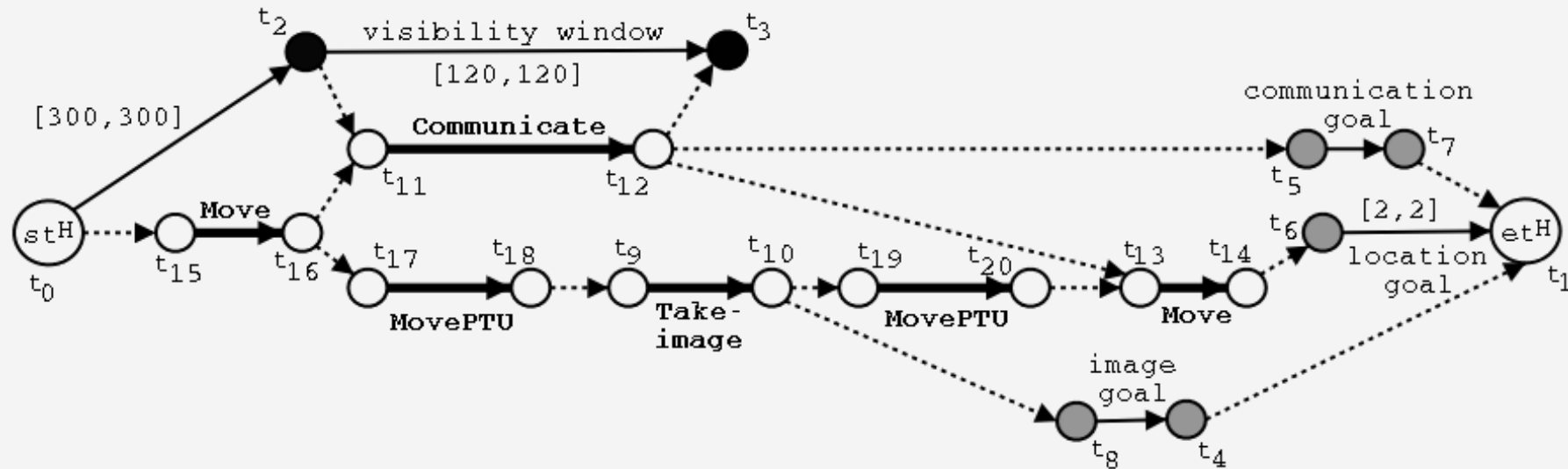
Timelines

- Up to now, *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
 - $t = 1, 2, 3, \dots$
 - For each state variable x , a **timeline**
 - Sequence of 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 per the Book

4.2 Representation

- Timelines
- Actions and tasks
- Chronicles

4.3 Temporal Planning

- Resolvers and flaws
- Search space

4.4 Constraint Management

- Consistency of object constraints and time constraints
- Controlling the actions when we do not know how long they will take

4.5 Acting with Temporal Models

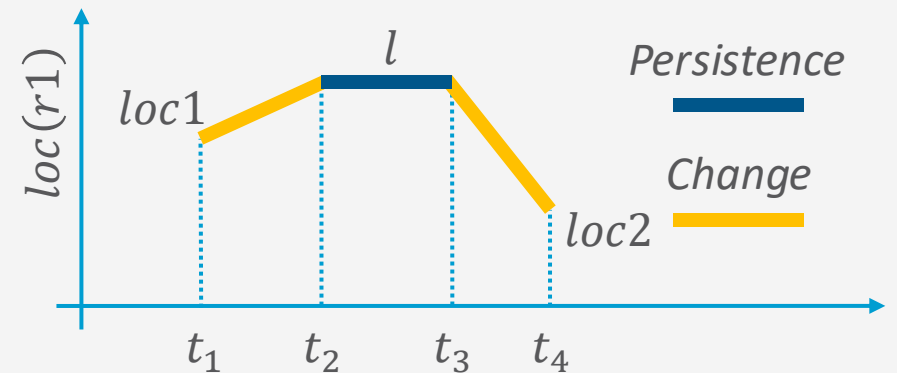
- Acting with atemporal refinement
- Dispatching
- Observation actions

Representation

- Quantitative model of time
 - Discrete: time points are integers
- Expressions:
 - time-point variables
 - t, t', t_2, t_j, \dots
 - simple constraints
 - $d \leq t' - t \leq d'$
- **Temporal assertion:**
 - Value of a state variable during a time interval
 - **Persistence:** $[t_1, t_2]x = v$ entails $t_1 < t_2$
 - **Change:** $[t_1, t_2]x : (v_1, v_2)$ entails $v_1 \neq v_2$ (and $t_1 < t_2$)

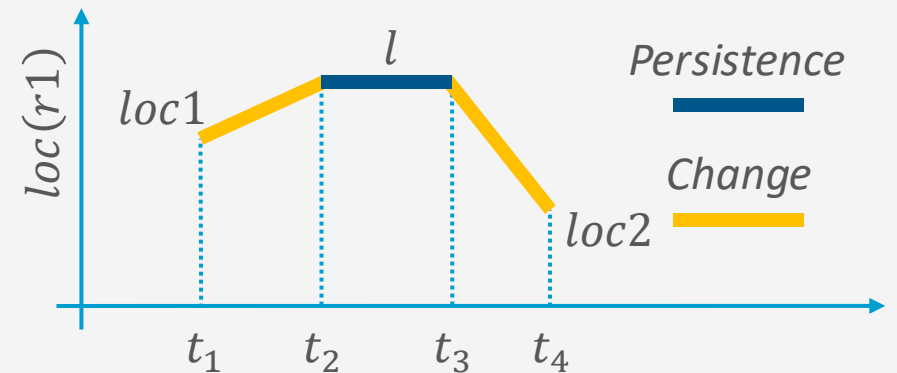
Timeline

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



Timeline

- **Instance** of $(\mathcal{T}, \mathcal{C})$ = temporal and object variables instantiated
- An instance is **consistent** if it satisfies all constraints in \mathcal{C} and does not specify two different values for a state variable at the same time
- A timeline is **secure** if its set of consistent instances is not empty



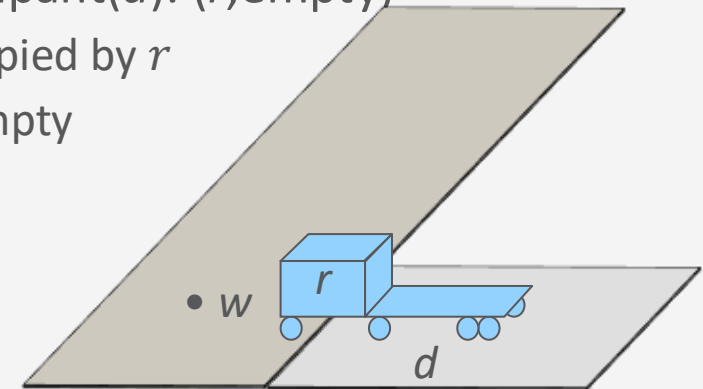
Actions

- Preliminaries:
 - Timelines $(\mathcal{T}_1, \mathcal{C}_1), \dots, (\mathcal{T}_k, \mathcal{C}_k)$ for k different state variables
 - Their **union**:
 - $(\mathcal{T}_1, \mathcal{C}_1) \cup \dots \cup (\mathcal{T}_k, \mathcal{C}_k) = (\mathcal{T}_1 \cup \dots \cup \mathcal{T}_k, \mathcal{C}_1 \cup \dots \cup \mathcal{C}_k)$
 - If
 - every $(\mathcal{T}_i, \mathcal{C}_i)$ is secure, and
 - no pair of timelines $(\mathcal{T}_i, \mathcal{C}_i)$ and $(\mathcal{T}_j, \mathcal{C}_j)$ has any unground variables in common
 - then
 - $(\mathcal{T}_1 \cup \dots \cup \mathcal{T}_k, \mathcal{C}_1 \cup \dots \cup \mathcal{C}_k)$ is also secure
- **Action** or **primitive task** (or just *primitive*):
 - a triple $(head, \mathcal{T}, \mathcal{C})$
 - *head* is the name and arguments
 - $(\mathcal{T}, \mathcal{C})$ is the union of a set of timelines
 - Primitive at the planning level
 - May be further refined at the acting level

Actions

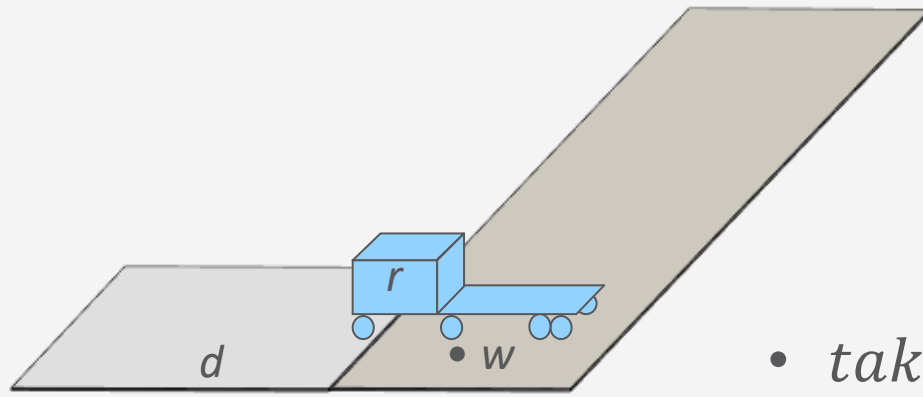
- $leave(r, d, w)$
 - Robot r leaves dock d , goes to adjacent waypoint w
- $leave(r, d, w)$
 assertions:
 $[t_s, t_e] loc(r): (d, w)$
 $[t_s, t_e] occupant(d): (r, empty)$
 constraints:
 $t_e \leq t_s + \delta_1$
 $adj(d, w)$
- $loc(r)$ changes to w with delay $\leq \delta_1$
 - Dock d becomes empty

- Two additional conventional parameters
 - Starting time t_s
 - Ending time t_e
- No separate preconditions and effects
 - Change assertion expresses both precondition and effect
 - E.g., $[t_s, t_e] occupant(d): (r, empty)$
 - Pre: d is occupied by r
 - Effect: d is empty



Actions

- $enter(r, d, w)$



- Robot r enters dock 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:

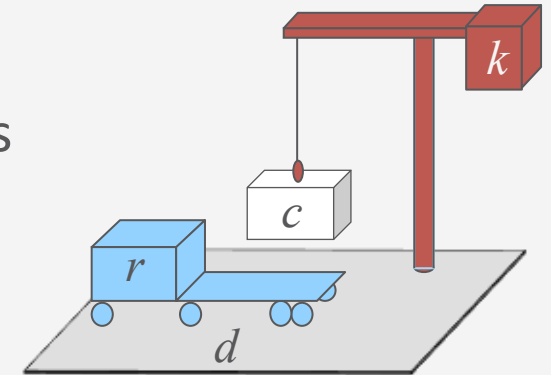
$t_e \leq t_s + \delta_2$

$adj(d, w)$

- $loc(r)$ changes to d with delay $\leq \delta_2$
- Dock d becomes occupied by r

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

- Action: crane k takes container c from r on dock d



book omits d

$take(k, c, r, d)$ // crane k , container c , robot r , dock d

assertions:

$[t_s, t_e] \text{ pos}(c): (r, k)$ // where c is

$[t_s, t_e] \text{ grip}(k): (\text{empty}, c)$ // what k 's gripper is holding

$[t_s, t_e] \text{ freight}(r): (c, \text{empty})$ // what r is carrying

$[t_s, t_e] \text{ loc}(r) = d$ // where r is

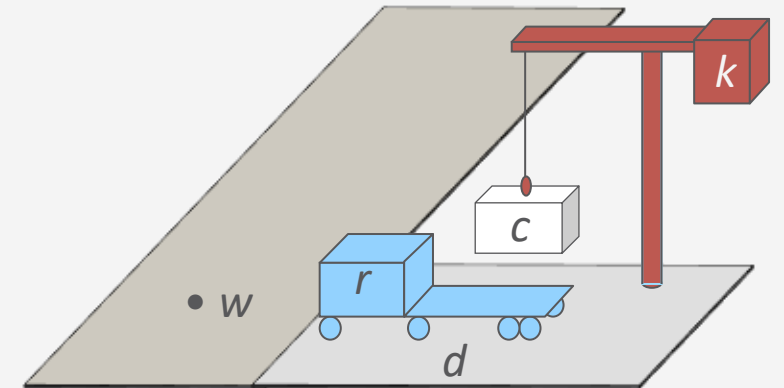
constraints:

$attached(k, d)$

Actions in the Example

- $leave(r, d, w)$ robot r leaves dock d to an adjacent waypoint w
- $enter(r, d, w)$ r enters d from an adjacent w
- $take(k, c, r, d)$ crane k takes cont. c from r at d
- $navigate(r, w, w')$ r navigates from w to w'
- $stack(k, c, p)$ k stacks c on top of pile p
- $unstack(k, c, p)$ k takes c from top of p
- $put(k, c, r, d)$ k puts c onto r at d

book omits d

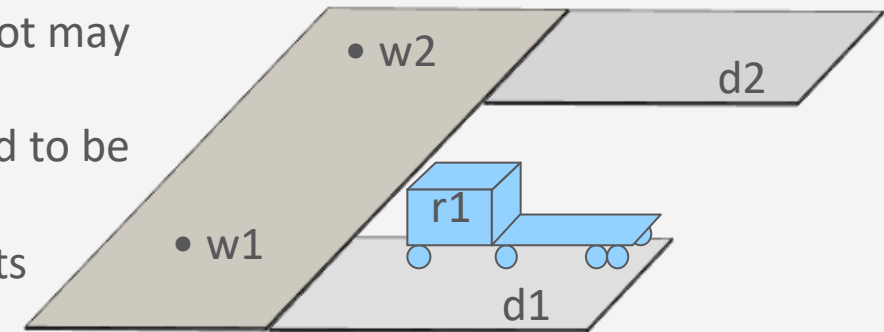
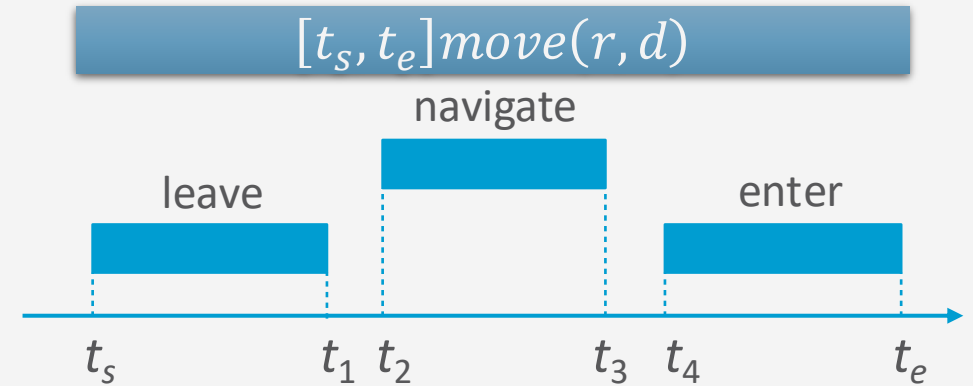


Tasks and Methods

- Methods implement a given task using refinements
 - Refinements can be actions or tasks again
 - Example only with actions as refinements
 - Task: move robot r to dock d , i.e., $[t_s, t_e]move(r, d)$

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

- d' becomes empty during $[t_s, t_1]$
 → Another robot may enter it after t_1
- d does not need to be empty until t_4
 → When r starts entering it



Tasks and Methods

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

- Method:

$m\text{-uncover}(c, p, k, d, p')$

task: $\text{uncover}(c, p)$

refinement: $[t_s, t_1] \text{unstack}(k, c', p)$ // action

$[t_2, t_3] \text{stack}(k, c', p')$ // action

$[t_4, t_e] \text{uncover}(c, p)$ // recursion

assertions: $[t_s, t_s+1] \text{pile}(c) = p$

$[t_s, t_s+1] \text{top}(p) = c'$

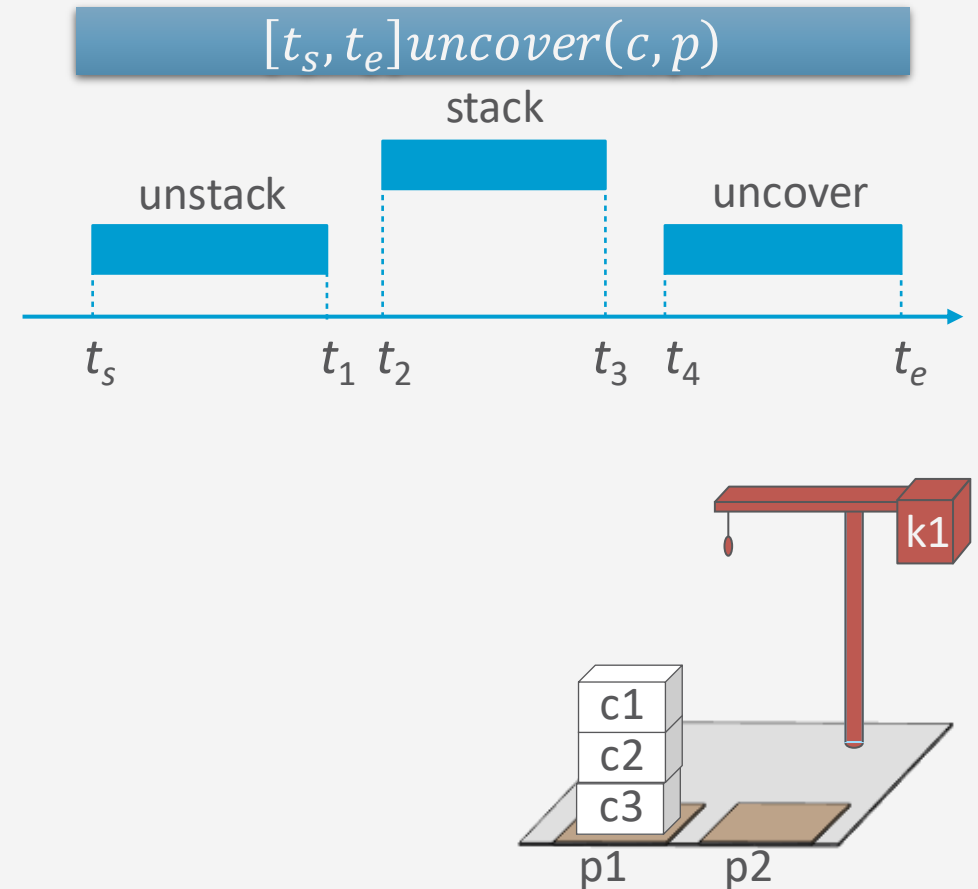
$[t_s, t_s+1] \text{grip}(k) = \text{empty}$

constraints: $\text{attached}(k, d), \text{attached}(p, d),$

$\text{attached}(p', d),$

$p \neq p', c' \neq c,$

$t_1 \leq t_2, t_3 \leq t_4$



Tasks and Methods

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

- Method:

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

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: $[t_s, t_3] pile(c) = p'$

$[t_s, t_3] freight(r) = empty$

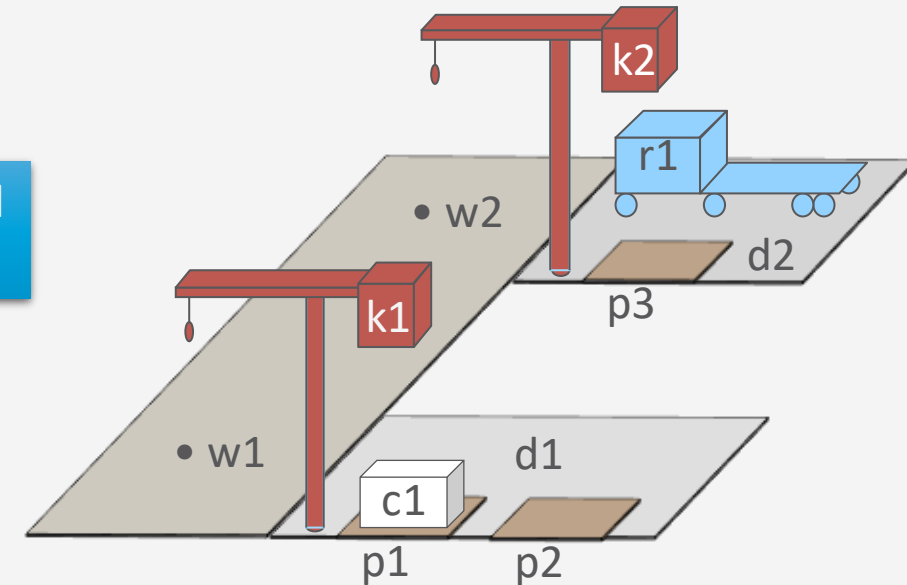
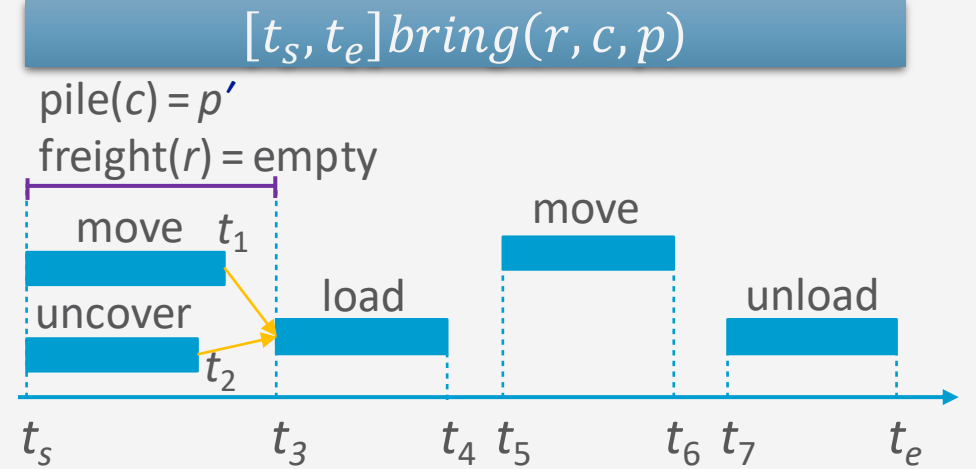
constraints: $attached(p', d')$, $attached(p, d)$, $d \neq d'$

$attached(k', d')$, $attached(k, d)$, $k \neq k'$

$t_1 \leq t_3$, $t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$

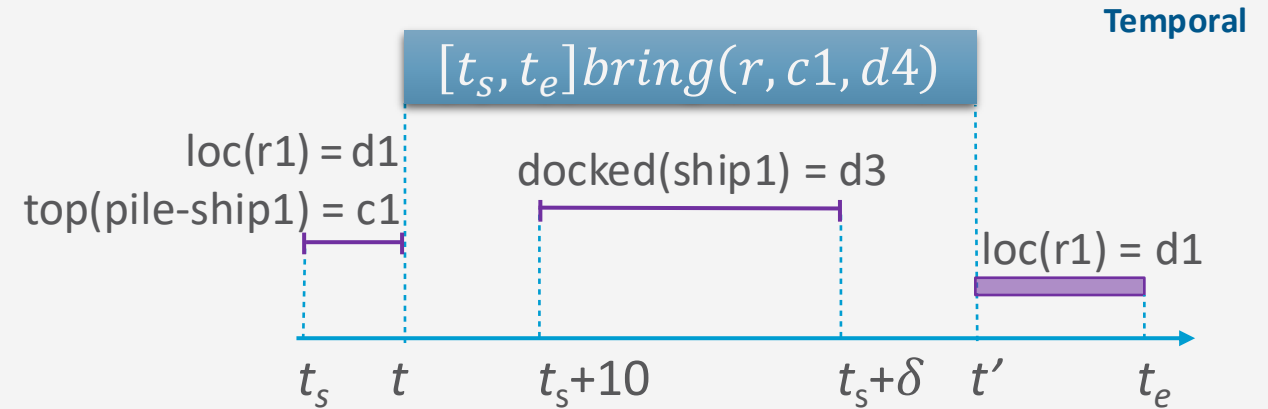
Refine into *unstack*
and *put* primitives

Refine into *take* and
stack primitives



Chronicles: Unions of Timelines

- Chronicle $\phi = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
 - \mathcal{A} : temporally qualified actions and tasks
 - \mathcal{S} : *a priori* supported assertions
 - \mathcal{T} : temporally qualified assertions
 - \mathcal{C} : constraints
- ϕ can include
 - Current state, future predicted events
 - Tasks to perform
 - Assertions and constraints to satisfy
- ϕ can represent
 - Planning problem
 - Plan or partial plan



ϕ_0 :

tasks: $[t, t']$ 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: $t_s = 0 < t < t' < t_e, 20 \leq \delta \leq 30$

Intermediate Summary

- Timelines
 - Temporal assertions (change, persistence), constraints
 - Conflicts, consistency, security, causal support
- Chronicle: union of several timelines
 - Consistency, security, causal support
- Actions represented by chronicles
 - No separate preconditions and effects
 - Preconditions \Leftrightarrow need for causal support

Outline per the Book

4.2 Representation

- Timelines
- Actions and tasks
- Chronicles

4.3 Temporal Planning

- Resolvers and flaws
- Search space

4.4 Constraint Management

- Consistency of object constraints and time constraints
- Controlling the actions when we do not know how long they will take

4.5 Acting with Temporal Models

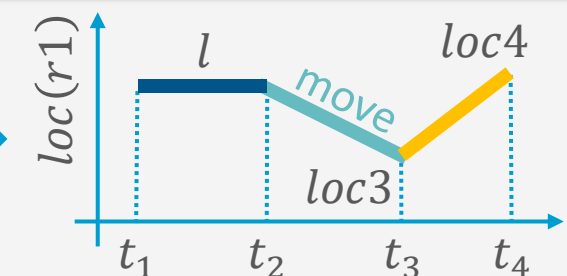
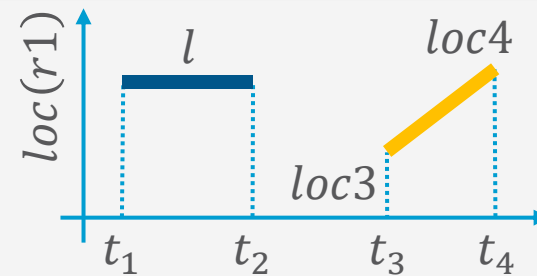
- Acting with atemporal refinement
- Dispatching
- Observation actions

Planning

- Planning problem:
 - Chronicle ϕ_0 that has some flaws
 - Analogous to flaws in PSP
- Add new assertions, constraints, actions to resolve the flaws

ϕ_0 : tasks: *(none)*
 supported: *(none)*
 assertions: $[t_1, t_2] \text{ loc}(r1) = l$
 $[t_3, t_4] \text{ loc}(r1) : (\text{loc3}, \text{loc4})$
 constraints: $\text{adj}(\text{loc3}, w1)$
 $\text{adj}(w1, \text{loc3})$
 $\text{adj}(\text{loc4}, w2)$
 $\text{adj}(w2, \text{loc4})$
 $\text{connected}(w1, w2)$

ϕ_1 : tasks: $[t_2, t_3] \text{ move}(r1, \text{loc3})$
 supported: *(none)*
 assertions: $[t_1, t_2] \text{ loc}(r1) = l$
 $[t_3, t_4] \text{ loc}(r1) : (\text{loc3}, \text{loc4})$
 constraints: $\text{adj}(\text{loc3}, w1)$
 $\text{adj}(w1, \text{loc3})$
 $\text{adj}(\text{loc4}, w2)$
 $\text{adj}(w2, \text{loc4})$
 $\text{connected}(w1, w2)$

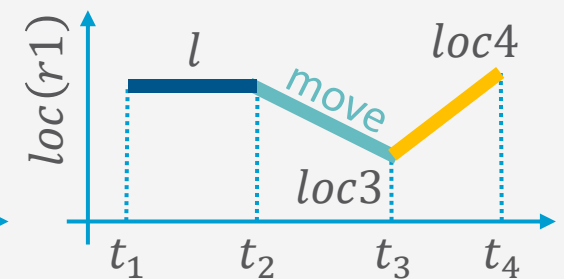
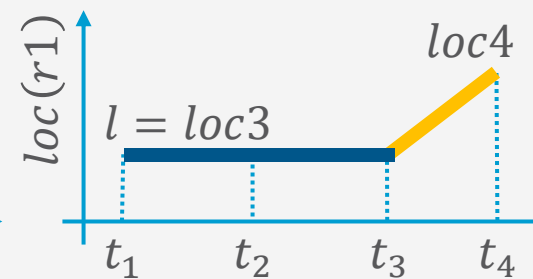
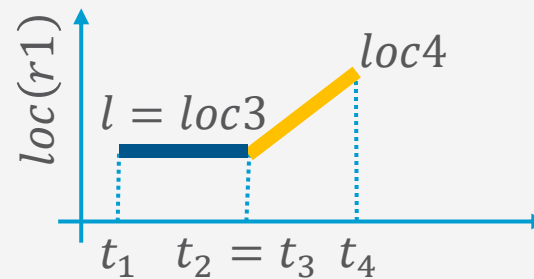
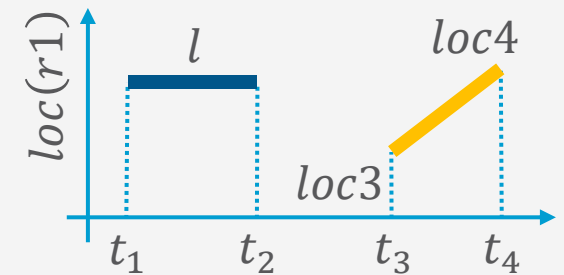


Flaws (1)

1. Temporal assertion α that is not *causally supported*

- What causes $r1$ to be at $loc3$ at time t_3 ?
- *Resolvers*:
 - Add constraints to support α from an assertion in ϕ
 - $l = loc3, t_2 = t_3$
 - Add a new persistence assertion to support α
 - $l = loc3, [t_2, t_3]loc(r1) = loc3$
 - Add a new task or action to support α
 - $[t_2, t_3]move(r1, loc3)$
 - Refining it will produce support for α

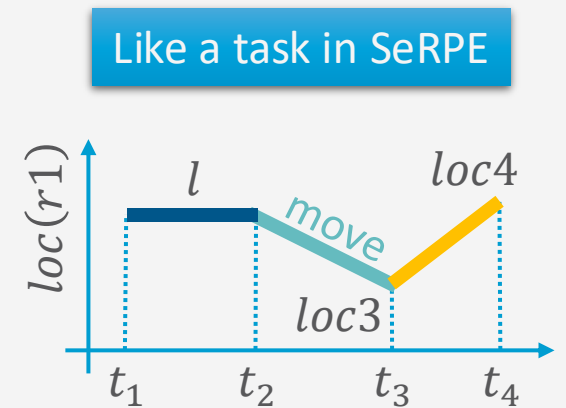
Like an open goal in PSP



Flaws (2)

2. Non-refined task

- *Resolver*: refinement method m
 - Applicable if it matches the task + its constraints are consistent with ϕ 's
- Applying the resolver:
 - Modify ϕ by replacing the task with m
- Example: $[t_2, t_3]move(r1, loc3)$
 - Refinement will replace it with something like
 - $[t_2, t_5]leave(r1, l, w)$
 - $[t_5, t_6]navigate(r1, w, w')$
 - $[t_6, t_3]enter(r1, loc3, w')$
 - Plus constraints



Flaws (3)

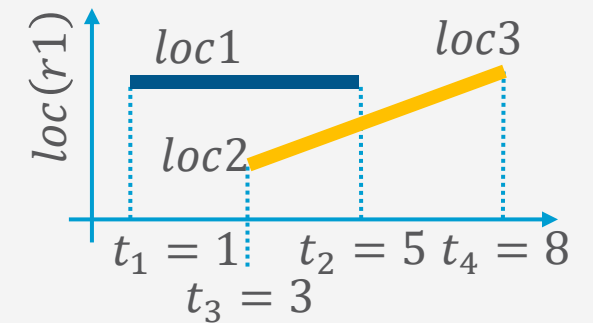
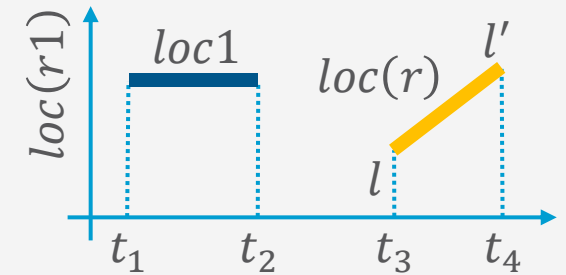
3. A pair of possibly-conflicting temporal assertions

- Temporal assertions α and β **possibly conflict** if they can have inconsistent instances
 - Example: $[t_1, t_2]loc(r1) = loc1$, $[t_3, t_4]loc(r) : (l, l')$

$\downarrow \downarrow$
 $[1, 5]loc(r1) = loc1$

$\downarrow \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
 $[3, 8]loc(r1) : (loc2, loc3)$
- Resolvers:** **separation** constraints
 - $r \neq r1$ or $t_2 < t_3$ or $t_4 < t_1$ or
 - $t_2 = t_3, r = r1, l = loc1$
 - Also provides causal support for $[t_3, t_4]loc(r) : (l, l')$
 - $t_4 = t_1, r = r1, l' = loc1$
 - Also provides causal support for $[t_1, t_2]loc(r1) = loc1$

Like a threat in PSP



Planning Algorithm

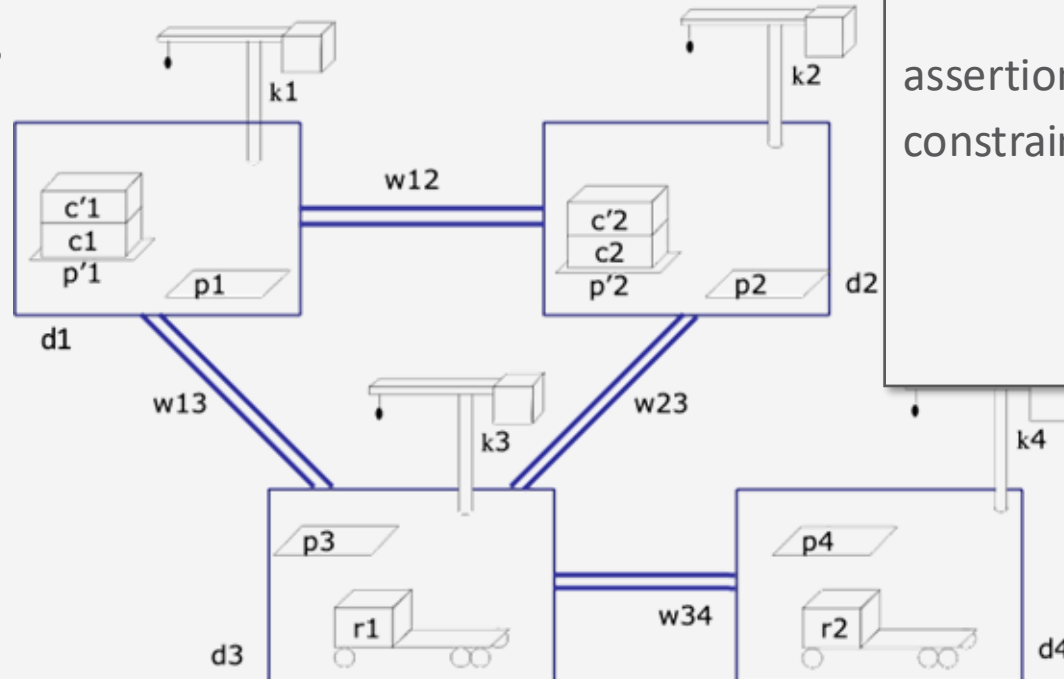
- Like PSP
 - Repeatedly selects flaws and chooses resolvers
- Input
 - Chronicle $\phi = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
- If resolving all flaws possible, at least one nondeterministic execution trace will do so
- In a deterministic implementation
 - Selecting a resolver ρ is a backtracking point
 - Selecting a flaw is not
 - (As in PSP)

```
TemPlan( $\phi$ )           // recursive version (book)
  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( $\phi$ )           // iterative version
  loop
    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

- $\phi = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
 - Establishes state-variable values at time $t = 0$
 - Flaws: two unrefined tasks
 - $\text{bring}(r, c1, p3)$
 - $\text{bring}(r', c2, p4)$



ϕ_0 : tasks: $\text{bring}(r, c1, p3)$
 $\text{bring}(r', c2, p4)$

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: *(none)*

constraints:

$\text{adj}(d1, w12)$

$\text{adj}(d1, w13)$

...

Example

- Flaws: two unrefined tasks
 - $\text{bring}(r,c1,p3)$, $\text{bring}(r',c2,p4)$
- Refinement for both:

```

m-bring( $r,c,p,p',d,d',k,k'$ )
  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( $k,d$ ),  $k \neq k'$ 
               $t_1 \leq t_3$ ,  $t_2 \leq t_3$ ,  $t_4 \leq t_5$ ,  $t_6 \leq t_7$ 

```

```

 $\phi_0$ : tasks: bring( $r,c1,p3$ )
              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: (none)

constraints:
            adj( $d1,w12$ )
            adj( $d1,w13$ )
            ...

```

Method Instance

- Instantiate $c = c1$ and $p = p3$ to match *bring*($r, c1, p3$)
 - p', d, d', k, k' instantiated to match book
 - Needed later to satisfy action preconditions

```

m-bring( $r, c1, p3, p'1, d3, d1, k3, k1$ )
  task: bring( $r, c1, p3$ )
  refinement: [ $t_s, t_1$ ] 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( $k3, r, c1, p3$ )
  assertions: [ $t_s, t_3$ ] pile( $c1$ ) =  $p'1$ 
               [ $t_s, t_3$ ] freight( $r$ ) = empty
  constraints: attached( $p'1, d1$ ),
               attached( $p3, d3$ ),  $d3 \neq d1$ 
               attached( $k1, d1$ ),
               attached( $k3, d3$ ),  $k3 \neq k1$ 
                $t_1 \leq t_3, t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7$ 

```

```

 $\phi_0$ : tasks: bring( $r, c1, p3$ )
              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: (none)
constraints:
             adj( $d1, w12$ )
             adj( $d1, w13$ )
             ...

```

Modified Chronicle

- Changes to ϕ_0
 - Removed *bring*(*r*, *c1*, *p3*)
 - Added 5 tasks, 2 assertions, 10 constraints
- Flaws
 - 6 unrefined tasks, 2 unsupported assertions

ϕ_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*)
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: $t_s < t_1 \leq t_3$, $t_s < t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$,
adj(*d1*, *w12*),
adj(*d1*, *w13*),
...

Method Instance

- Instantiate $r = r', c = c2, p = p4$ to match $bring(r', c2, p4)$
- p', d, d', k, k' instantiated to match book again

```

m-bring(r',c2,p4,p'2,d4,d2,k4,k2)
  task: bring(r',c2,p4)
  refinement: [t_s,t_1] move(r',d2)
              [t_s,t_2] uncover(c2,p'2)
              [t_3,t_4] load(k2,r',c2,p'2)
              [t_5,t_6] move(r', d4)
              [t_7,t_e] unload(k4,r',c2,p4)
  assertions: [t_s,t_3] pile(c2) = p'2
              [t_s,t_3] freight(r') = empty
  constraints: attached(p'2,d2),
              attached(p4,d4), d4 ≠ d2
              attached(k2,d2),
              attached(k4,d4), k4 ≠ k2
              t_1 ≤ t_3, t_2 ≤ t_3, t_4 ≤ t_5, t_6 ≤ t_7
  
```

```

φ1: 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)
            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_5 < t_1 ≤ t_3, t_5 < t_2 ≤ t_3, t_4 ≤ t_5, t_6 ≤ t_7,
            adj(d1,w12),
            adj(d1,w13),
            ...
  
```

Modified Chronicle

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

ϕ_2 : 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($k3, r, c1, p3$)
 $[t'_s, t'_1]$ 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: $t_s < t_1 \leq t_3, t_s < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$
 $t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$
adj($d1, w12$),
adj($d1, w13$), ...

Supporting the Assertions

- 3 ways to support

$$[t_s, t_3] \text{pile}(c1) = p'1$$

1. Constrain $t_s = 0$,
use $[0] \text{pile}(c1) = p'1$
2. Add persistence $[0, t_s] \text{pile}(c1) = p'1$
3. Add new action $[t_8, t_s] \text{stack}(k1, c1, p'1)$

Will any of them also
provide support for
 $[t_s, t_3] \text{freight}(r) = \text{empty}$
?

ϕ_2 : tasks: $[t_s, t_1] \text{move}(r, d1)$
 $[t_s, t_2] \text{uncover}(c1, p'1)$
 $[t_3, t_4] \text{load}(k1, r, c1, p'1)$
 $[t_5, t_6] \text{move}(r, d3)$
 $[t_7, t_e] \text{unload}(k3, r, c1, p3)$
 $[t'_s, t'_1] \text{move}(r', d2)$
 $[t'_s, t'_2] \text{uncover}(c2, p'2)$
 $[t'_3, t'_4] \text{load}(k4, r', c2, p'2)$
 $[t'_5, t'_6] \text{move}(r', d4)$
 $[t'_7, t'_e] \text{unload}(k2, r', c2, p'2)$

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

...

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

constraints: $t_s < t_1 \leq t_3, t_s < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$
 $t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$
 $\text{adj}(d1, w12),$
 $\text{adj}(d1, w13), \dots$

Supporting the Assertions

- 3 ways to support

$$[t_s, t_3] \text{pile}(c1) = p'1$$

1. Constrain $t_s = 0$,
use $[0] \text{pile}(c1) = p'1$

- To support

$$[0, t_3] \text{freight}(r) = \text{empty}$$

1. Constrain $r = r1$,
use $[0] \text{freight}(r1) = \text{empty}$

ϕ_2 : tasks: $[0, t_1]$ move($r, d1$)
 $[0, t_2]$ 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: $[0] \text{loc}(r1) = d3$

$[0] \text{freight}(r1) = \text{empty}$

$[0] \text{pile}(c1) = p'1$

...

assertions: $[0, t_3] \text{pile}(c1) = p'1$
 $[0, t_3] \text{freight}(r) = \text{empty}$

$[t'_s, t'_3] \text{pile}(c2) = p'2$

$[t'_s, t'_1] \text{freight}(r') = \text{empty}$

constraints: $0 < t_1 \leq t_3, 0 < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$
 $t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$

adj($d1, w12$),

adj($d1, w13$), ...

Supporting the Assertions

- 3 ways to support

$$[t_s, t_3] \text{pile}(c1) = p'1$$

1. Constrain $t_s = 0$,
use $[0] \text{pile}(c1) = p'1$

- To support

$$[0, t_3] \text{freight}(r) = \text{empty}$$

1. Constrain $r = r1$,
use $[0] \text{freight}(r1) = \text{empty}$

ϕ_2 : tasks: $[0, t_1]$ move($r1, d1$)
 $[0, t_2]$ uncover($c1, p'1$)
 $[t_3, t_4]$ load($k1, r1, c1, p'1$)
 $[t_5, t_6]$ move($r1, d3$)
 $[t_7, t_e]$ unload($k3, r1, c1, p3$)
 $[t'_s, t'_1]$ 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: $[0] \text{loc}(r1) = d3$

$[0] \text{freight}(r1) = \text{empty}$

$[0] \text{pile}(c1) = p'1$

...

$[0, t_3] \text{pile}(c1) = p'1$

$[0, t_3] \text{freight}(r1) = \text{empty}$

assertions: $[t'_s, t'_3] \text{pile}(c2) = p'2$

$[t'_s, t'_1] \text{freight}(r') = \text{empty}$

constraints: $0 < t_1 \leq t_3, 0 < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$

$t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$

adj($d1, w12$),

adj($d1, w13$), ...

Supporting the Assertions

- To support

$$[t'_s, t'_3]pile(c2) = p'2$$

1. Add persistence condition $[0, t'_s]pile(c2) = p'2$
2. Constrain $t'_s = 0$
3. Add new action $stack(k2, c2, p'2)$

ϕ_2 : tasks: $[0, t_1]$ move(r1,d1)
 $[0, t_2]$ uncover(c1,p'1)
 $[t_3, t_4]$ load(k1,r1,c1,p'1)
 $[t_5, t_6]$ move(r1,d3)
 $[t_7, t_e]$ unload(k3,r1,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

...

$[0, t_3]$ pile(c1) = p'1
 $[0, t_3]$ freight(r1) = empty

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

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

Supporting the Assertions

- To support

$$[t'_s, t'_3] \text{pile}(c2) = p'2$$

- Add persistence condition $[0, t'_s] \text{pile}(c2) = p'2$

- To support

$$[t'_s, t'_1] \text{freight}(r') = \text{empty}$$

- Constrain $r' = r2$

add persistence condition

$$[0, t'_s] \text{freight}(r2) = \text{empty}$$

ϕ_2 : tasks: $[0, t_1]$ move(r1,d1)
 $[0, t_2]$ uncover(c1,p'1)
 $[t_3, t_4]$ load(k1,r1,c1,p'1)
 $[t_5, t_6]$ move(r1,d3)
 $[t_7, t_e]$ unload(k3,r1,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 ...

$[0, t_3]$ 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

assertions: $[t'_s, t'_1]$ freight(r') = empty

constraints: $0 < t_1 \leq t_3, 0 < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$

$t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$

adj(d1,w12),

adj(d1,w13), ...

Supporting the Assertions

- To support

$$[t'_s, t'_3] \text{pile}(c2) = p'2$$

- Add persistence condition $[0, t'_s] \text{pile}(c2) = p'2$

- To support

$$[t'_s, t'_1] \text{freight}(r') = \text{empty}$$

- Constrain $r' = r2$

add persistence condition

$$[0, t'_s] \text{freight}(r2) = \text{empty}$$

- All assertions currently supported
- Remaining flaws: unrefined tasks

ϕ_2 : tasks: $[0, t_1]$ move(r1,d1)
 $[0, t_2]$ uncover(c1,p'1)
 $[t_3, t_4]$ load(k1,r1,c1,p'1)
 $[t_5, t_6]$ move(r1,d3)
 $[t_7, t_e]$ unload(k3,r1,c1,p3)
 $[t'_s, t'_1]$ move(r2,d2)
 $[t'_s, t'_2]$ uncover(c2,p'2)
 $[t'_3, t'_4]$ load(k4,r2,c2,p'2)
 $[t'_5, t'_6]$ move(r2,d4)
 $[t'_7, t'_e]$ unload(k2,r2,c2,p'2)

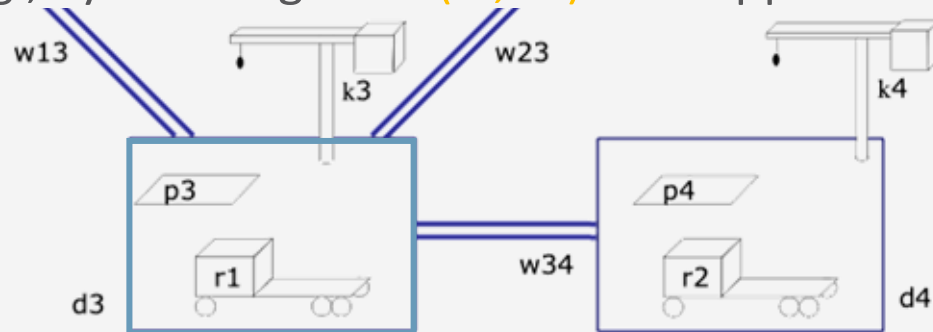
supported: $[0]$ loc(r1)=d3
 $[0]$ freight(r1)=empty
 $[0]$ pile(c1)=p'1 ...
 $[0, t_3]$ pile(c1) = p'1
 $[0, t_3]$ freight(r1) = empty
 $[0, t'_s]$ pile(c2)=p'2
 $[t'_s, t'_3]$ pile(c2) = p'2
 $[0, t'_s]$ freight(r2)=empty
 $[t'_s, t'_1]$ freight(r2) = empty

assertions: (none)

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

Example of Conflicts

- Refining tasks into actions will produce possibly-conflicting assertions
 - $\text{move}(r2,d4)$ must go from $d2$ through $d3$
 - Conflict: $\text{occupant}(d3)=r1, \text{occupant}(d3)=r2$
- Resolvers:
 - Separation constraints to ensure $r2$ only goes through $d3$ while $r1$ away from $d3$
 - E.g., by ensuring $\text{move}(r1,d1)$ has happened



ϕ_2 : tasks: $[0,t_1]$ $\text{move}(r1,d1)$
 $[0,t_2]$ $\text{uncover}(c1,p'1)$
 $[t_3,t_4]$ $\text{load}(k1,r1,c1,p'1)$
 $[t_5,t_6]$ $\text{move}(r1,d3)$
 $[t_7,t_e]$ $\text{unload}(k3,r1,c1,p3)$
 $[t'_s,t'_1]$ $\text{move}(r2,d2)$
 $[t'_s,t'_2]$ $\text{uncover}(c2,p'2)$
 $[t'_3,t'_4]$ $\text{load}(k4,r2,c2,p'2)$
 $[t'_5,t'_6]$ $\text{move}(r2,d4)$
 $[t'_7,t'_e]$ $\text{unload}(k2,r2,c2,p'2)$

supported: $[0]$ $\text{loc}(r1)=d3$
 $[0]$ $\text{freight}(r1)=\text{empty}$
 $[0]$ $\text{pile}(c1)=p'1 \dots$
 $[0,t_3]$ $\text{pile}(c1) = p'1$
 $[0,t_3]$ $\text{freight}(r1) = \text{empty}$
 $[0,t'_s]$ $\text{pile}(c2)=p'2$
 $[t'_s,t'_3]$ $\text{pile}(c2) = p'2$
 $[0,t'_s]$ $\text{freight}(r2)=\text{empty}$
 $[t'_s,t'_1]$ $\text{freight}(r2) = \text{empty}$
 assertions: *(none)*

constraints: $0 < t_1 \leq t_3, 0 < t_2 \leq t_3, t_4 \leq t_5, t_6 \leq t_7,$
 $t'_s < t'_1 \leq t'_3, t'_s < t'_2 \leq t'_3, t'_4 \leq t'_5, t'_6 \leq t'_7,$
 $\text{adj}(d1,w12), \text{adj}(d1,w13), \dots$

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
 - We discuss it next

TemPlan(ϕ)

```
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( $\phi$ )
```

PSP(Σ, π)

```
loop
    if Flaws( $\pi$ ) =  $\emptyset$  then
        return  $\pi$ 
    arbitrarily select  $f \in$  Flaws( $\pi$ )
     $R \leftarrow$  {all feasible resolvers for  $f$ }
    if  $R = \emptyset$  then
        return failure
    nondeterministically choose  $\rho \in R$ 
     $\pi \leftarrow \rho(\pi)$ 
return  $\pi$ 
```

Intermediate Summary

- Planning problems
 - Three kinds of flaws and their resolvers:
 - tasks (that need to be refined),
 - causal support (for assertions),
 - security (of instantiations)
 - Partial plans, solution plans
- Planning: TemPlan
 - Like PSP but with tasks, temporal assertions, temporal constraints

Outline per the Book

4.2 Representation

- Timelines
- Actions and tasks
- Chronicles

4.3 Temporal Planning

- Resolvers and flaws
- Search space

4.4 Constraint Management

- Consistency of object constraints and time constraints
- Controlling the actions when we do not know how long they will take

4.5 Acting with Temporal Models

- Acting with atemporal refinement
- Dispatching
- Observation actions

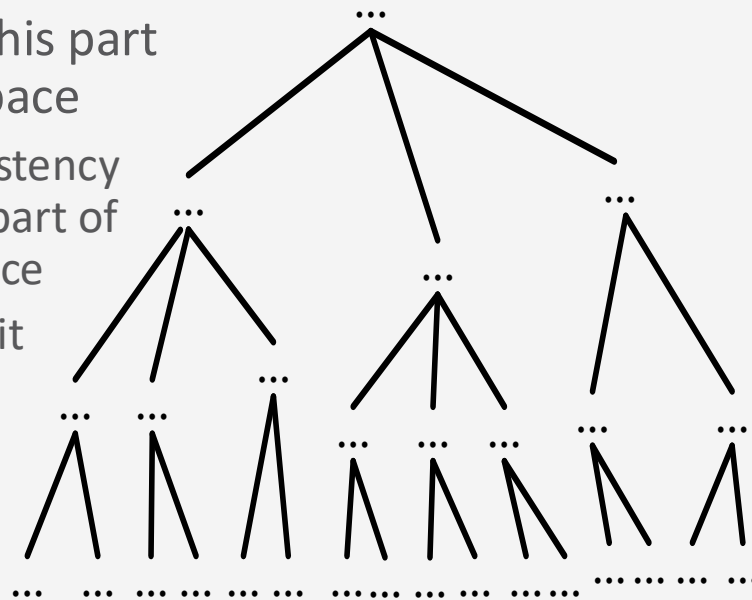
Constraint Management

- Each time TemPlan applies a resolver, it modifies $(\mathcal{T}, \mathcal{C})$
- Some resolvers will make $(\mathcal{T}, \mathcal{C})$ inconsistent

- No solution in this part of the search space

- Detect inconsistency → prune this part of the search space

- Do not detect it → waste time looking for a solution



- Analogy: PSP checks simple cases of inconsistency
- E.g., cannot create a constraint $a < b$ if there is already a constraint $b < a$
- Ignores more complicated cases
- Example:
 - $c_1, c_2, c_3 \in Containers = \{c1, c2\}$
 - Threats involving c_1, c_2, c_3
 - For resolvers, suppose PSP chooses
 - $c_1 \neq c_2, c_2 \neq c_3, c_1 \neq c_3$
 - No solutions in this part of the search space, but PSP searches it anyway

Constraint Management in TemPlan

- At various points, check consistency of \mathcal{C}
 - If \mathcal{C} is inconsistent, then $(\mathcal{T}, \mathcal{C})$ is inconsistent
 - Can prune this part of the search space
- If \mathcal{C} is consistent, then $(\mathcal{T}, \mathcal{C})$ may or may not be consistent
 - Example:
 - $\mathcal{T} = \{[t_1, t_2]loc(r1) = loc1, [t_3, t_4]loc(r1) = loc2\}$
 - $\mathcal{C} = (t_1 < t_3 < t_4 < t_2)$
 - Gives $loc(r1)$ two values during $[t_3, t_4]$

An instance is **consistent** if

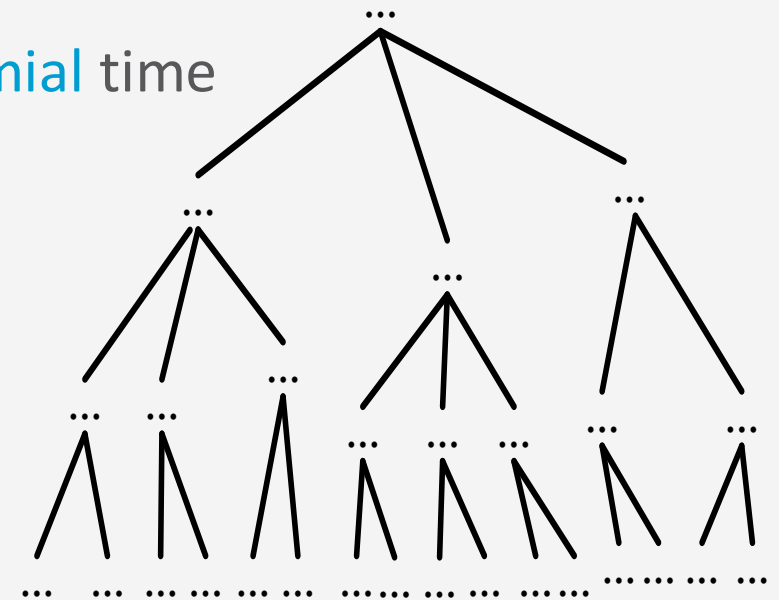
- it satisfies all constraints in \mathcal{C} and
- does not specify two different values for a state variable at the same time

Consistency of \mathcal{C}

- \mathcal{C} contains two kinds of constraints
 - **Object** constraints
 - $loc(r) \neq l_2, \quad l \in \{loc3, loc4\}, \quad r = r1, \quad o \neq o'$
 - **Temporal** constraints
 - $t_1 < t_3, \quad a < t, \quad t < t', \quad a \leq t' - t \leq b$
- Assume object constraints are independent of temporal constraints and vice versa
 - Exclude things like $t < f(l, r)$ with some function f
- Then two separate subproblems:
 1. Check consistency of object constraints
 2. Check consistency of temporal constraints
- \mathcal{C} is consistent iff both are consistent

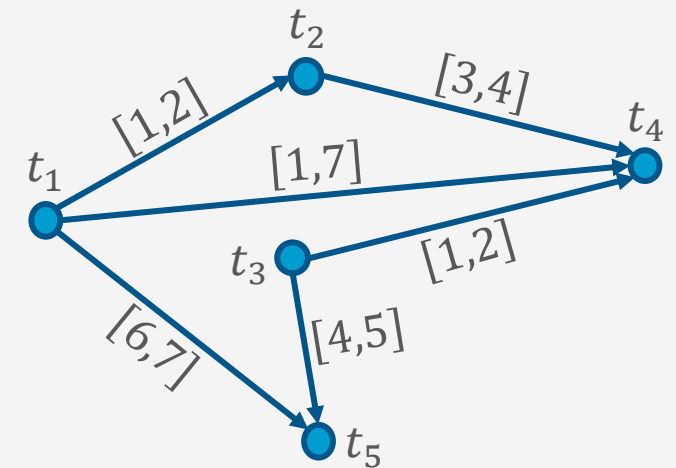
Object Constraints

- Constraint-satisfaction problem – NP-complete
- Can write an algorithm that is **complete** but runs in **exponential** time
 - If there is an inconsistency, always finds it
 - Might prune a lot, but spends lots of time at each node
- Instead, use a technique that is **incomplete** but takes **polynomial** time
 - Detects some inconsistencies but not others
 - Runs much faster, but prunes fewer nodes



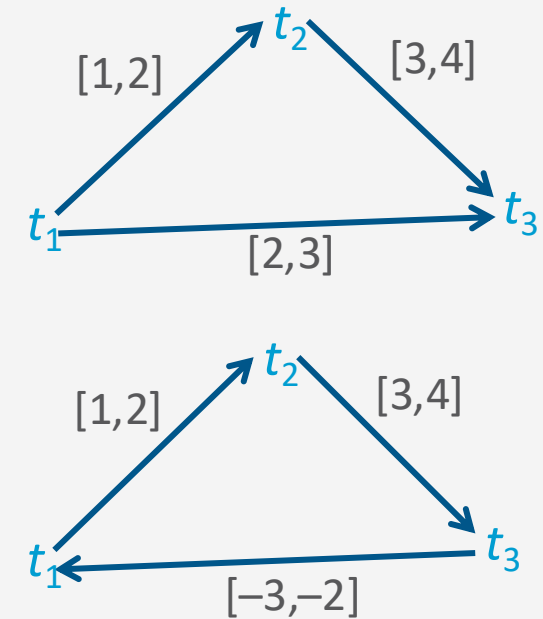
Time Constraints: Representation

- Simple Temporal Networks (STNs)
 - Networks of constraints on time points
- Synthesise an STN incrementally starting from ϕ_0
 - TemPlan can check time constraints in time $O(n^3)$
- Incrementally instantiated at acting time
- Kept consistent throughout planning and acting



Simple Temporal Networks

- STN: a pair $(\mathcal{V}, \mathcal{E})$, where
 - $\mathcal{V} = \{\text{a set of temporal variables } t_1, \dots, t_n\}$
 - $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges
- Each edge (t_i, t_j) is labelled with an interval $[a, b]$
 - Shorthand: represents constraint $a \leq t_j - t_i \leq b$
 - Equivalently, $-b \leq t_i - t_j \leq -a$
- Representing unary constraints
 - Dummy variable $t_0 = 0$
 - Edge (t_0, t_i) labelled with $[a, b]$ represents $a \leq t_i - 0 \leq b$



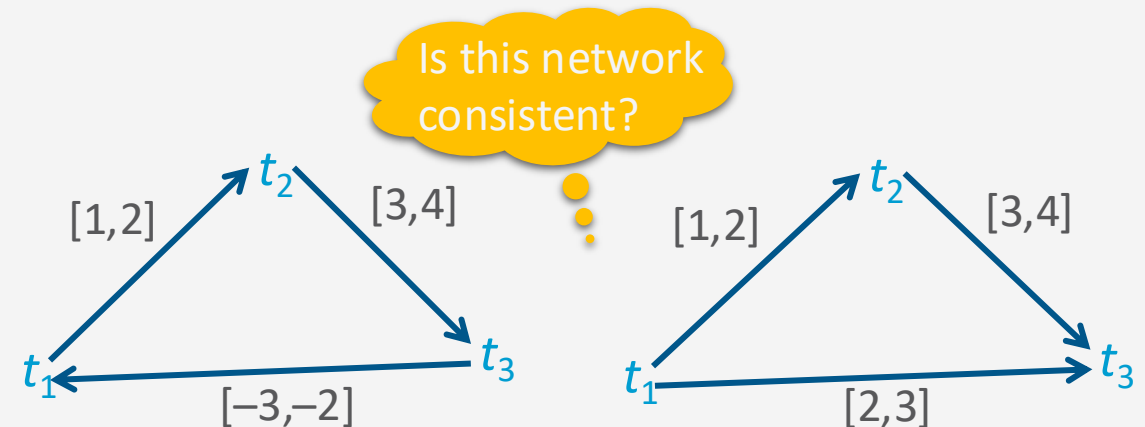
Simple Temporal Networks

- STN: a pair $(\mathcal{V}, \mathcal{E})$, where
 - $\mathcal{V} = \{\text{a set of temporal variables } t_1, \dots, t_n\}$
 - $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges
- Each edge (t_i, t_j) is labelled with an interval $[a, b]$
 - Shorthand: represents constraint $a \leq t_j - t_i \leq b$
 - Equivalently, $-b \leq t_i - t_j \leq -a$
- Representing unary constraints
 - Dummy variable $t_0 = 0$
 - Edge (t_0, t_i) labelled 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

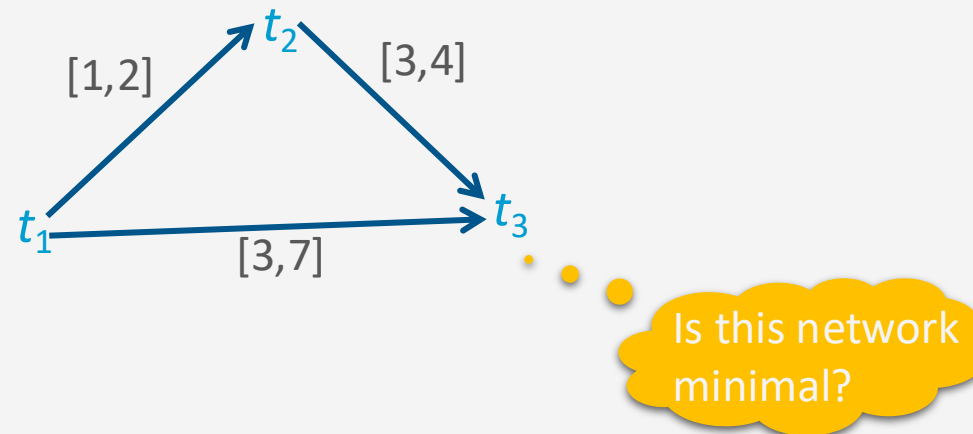
Book says:

- **Solution**
 - Integer value for each t_i
- **Consistent:**
 - Has a solution
 - *All constraints satisfied*



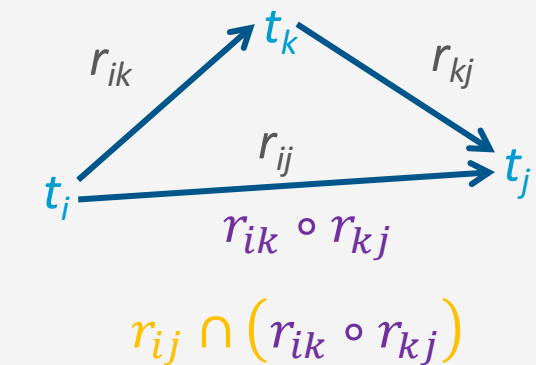
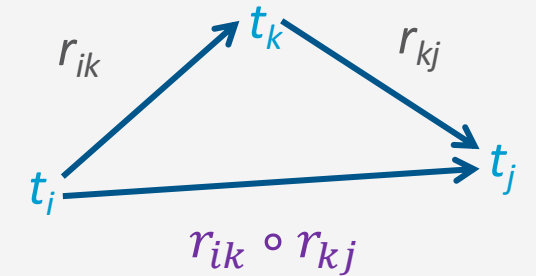
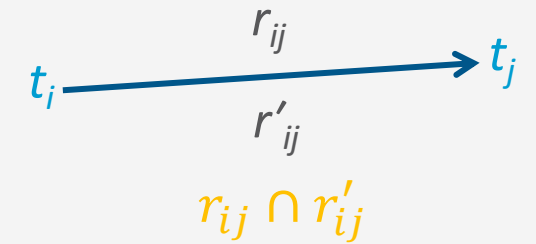
Time Constraints

- Minimal STN:
 - For every edge (t_i, t_j) with label $[a, b]$
 - For every $t \in [a, b]$
 - There is at least one solution such that $t_j - t_i = t$
 - Cannot make any of the time intervals shorter without excluding some solutions



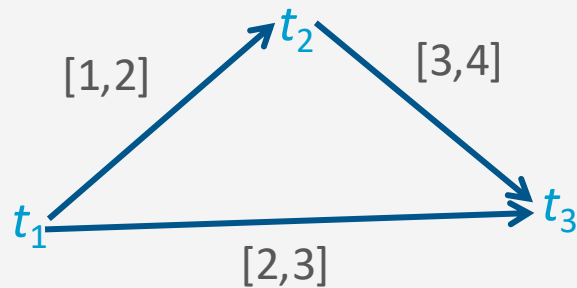
Operations on STNs

- 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, \circ
 - $t_k - t_i \in r_{ik} = [a_{ik}, b_{ik}]$
 - $t_j - t_k \in r_{kj} = [a_{kj}, b_{kj}]$
 - Infer $t_j - t_i \in r_{ik} \circ r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}]$
 - Reasoning: add up shortest and longest times
- Consistency checking
 - Three constraints r_{ik}, r_{kj}, r_{ij} are consistent only if $r_{ij} \cap (r_{ik} \circ r_{kj}) \neq \emptyset$

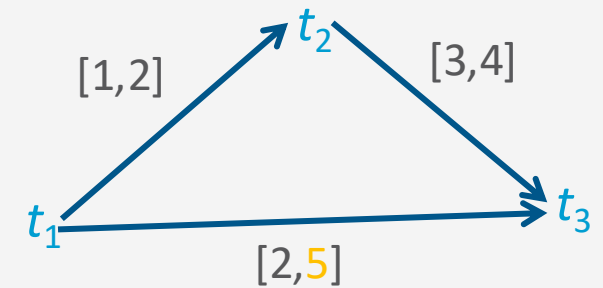


Two Examples

- 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
 - $r'_{13} = r_{12} \circ r_{23} = [1,2] \circ [3,4] = [4,6]$
- Cannot 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
 - $\mathcal{V} = \{t_1, t_2, t_3\}$
 - $\mathcal{E} = \{r_{12} = [1,2], r_{23} = [3,4], r_{13} = [2,5]\}$
- Composition (as before)
 - $r'_{13} = r_{12} \circ r_{23} = [4,6]$
- $(\mathcal{V}, \mathcal{E})$ is consistent
 - $r_{13} \cap r'_{13} = [2,5] \cap [4,6] = [4,5]$
- Minimal network with $r_{13} = [4,5]$

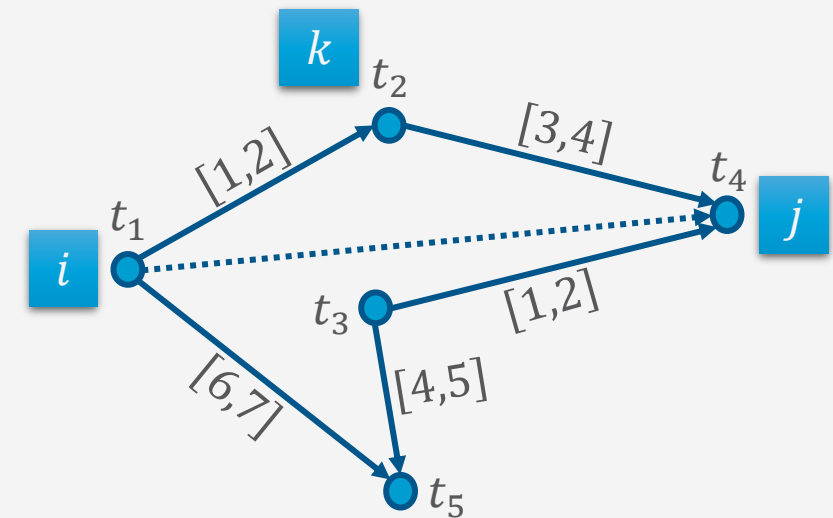


Operations on STNs

- PC (*Path Consistency*) algorithm:
 - Consistency checking on all triples
 - Input: STN $(\mathcal{V}, \mathcal{E})$
 - If an edge has no constraint, use $[-\infty, +\infty]$
 - n constraints $\rightarrow n^3$ triples \rightarrow time $O(n^3)$
- Example: $k = 2, i = 1, j = 4$
 - $r_{12} = [1, 2]$
 - $r_{24} = [3, 4]$
 - $r_{14} = [-\infty, \infty]$
 - $r_{12} \circ r_{24} = [1 + 3, 2 + 4] = [4, 6]$
 - $r_{14} \leftarrow [\max(-\infty, 4), \min(\infty, 6)] = [4, 6]$

```

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

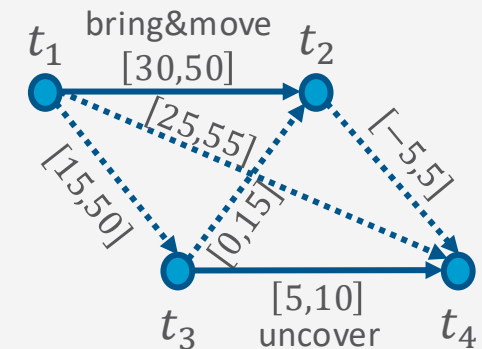


Operations on STNs

- PC makes network minimal
 - Shrinks each r_{ij} to exclude values not in any solution
 - Doing so, it detects inconsistent networks
 - $r_{ij} = [a_{ij}, b_{ij}]$ empty \rightarrow inconsistent
- Graph: dashed lines
 - Constraints that were shrunk
- Can modify PC to make it incremental
 - Input
 - A consistent, minimal STN
 - A new constraint r'_{ij}
 - Incorporate r'_{ij} in time $O(n^2)$

```

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



Pruning TemPlan's search space

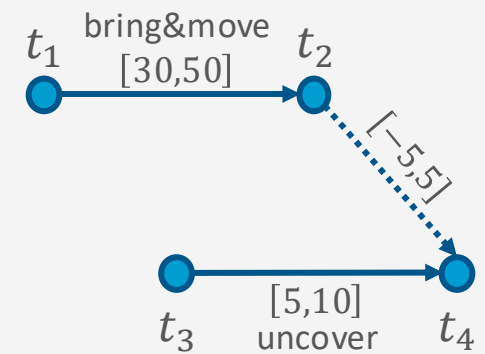
- Take the time constraints in \mathcal{C}
 - Write them as an STN
 - Use PC to check whether STN is consistent
 - If it is inconsistent, TemPlan can backtrack

Controllability

Constraint Management with Uncertain Durations

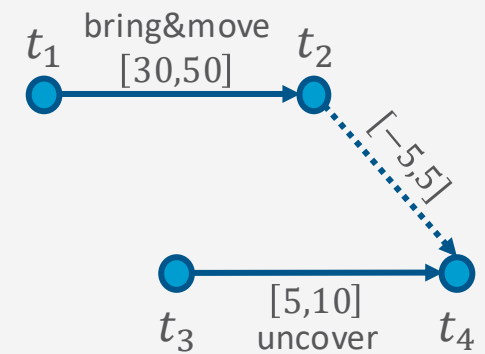
Controllability

- Suppose TemPlan gives you a chronicle and you want to execute it
 - Constraints on time points
 - Need to reason about these to decide when to start each action



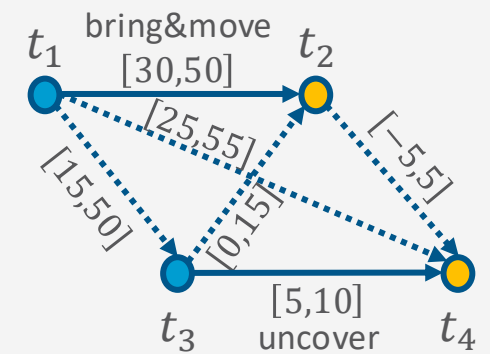
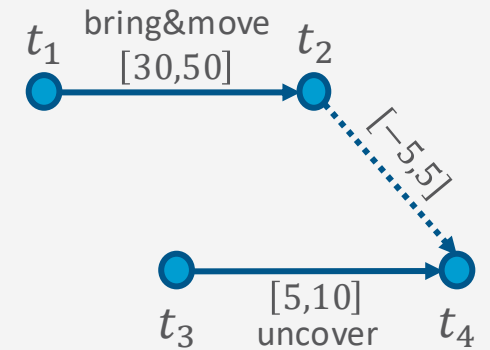
Controllability

- 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: **synchronisation constraint**
 - Do not 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

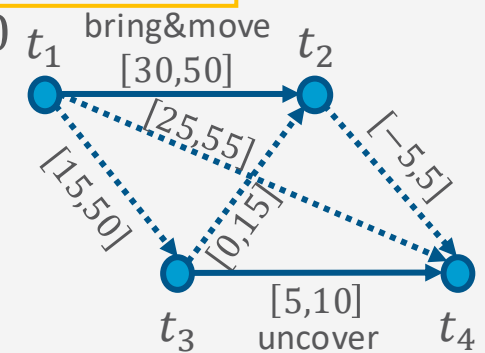
- Suppose we run PC
 - PC returns a minimal and consistent network
 - There *exist* time points that satisfy all the constraints
- Would work if we could choose all four time points
 - But we cannot choose t_2 and t_4
- t_1 and t_3 are **controllable**
 - Actor can control when each action starts
- t_2 and t_4 are **contingent**
 - Cannot 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

- Cannot guarantee all constraints satisfied
- Start bring&move at time $t_1 = 0$
- Suppose the durations are
 - bring&move 30, uncover 10
 - $t_2 = t_1 + 30 = 30$
 - $t_4 = t_3 + 10$
 - $t_4 - t_2 = t_3 - 20$
 - Constraint r_{24} :
 - $-5 \leq t_4 - t_2 \leq 5$
 - $-5 \leq t_3 - 20 \leq 5$
 - $15 \leq t_3 \leq 25$
- Must start uncover at $t_3 \leq 25$

- But if we start uncover at $t_3 \leq 25$, neither action has finished yet
 - We do not yet know how long they will take
- Durations might instead be
 - bring&move 50, uncover 5
 - $t_2 = t_1 + 50 = 50$
 - $t_4 = t_3 + 5 \leq 25 + 5 = 30$
 - $t_4 - t_2 \leq 30 - 50 = -20$
 - Violates r_{24}



STNUs

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

STNUs

- 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 \mathcal{V} (starting times) that guarantee that constraints are satisfied

Three kinds of controllability

- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is **strongly controllable** if the actor can choose values for \mathcal{V} such that success will occur for all values of $\tilde{\mathcal{V}}$ that satisfy $\tilde{\mathcal{E}}$
 - Actor can choose the values for \mathcal{V} offline
 - The right choice will work regardless of $\tilde{\mathcal{V}}$
- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is **weakly controllable** if the actor can choose values for \mathcal{V} such that success will occur for *at least one* combination of values for $\tilde{\mathcal{V}}$
 - Actor can choose the values for \mathcal{V} only if the actor knows in advance what the values of $\tilde{\mathcal{V}}$ will be

Three kinds of controllability

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

Dynamic Execution

- For $t = 0, 1, 2, \dots$
 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

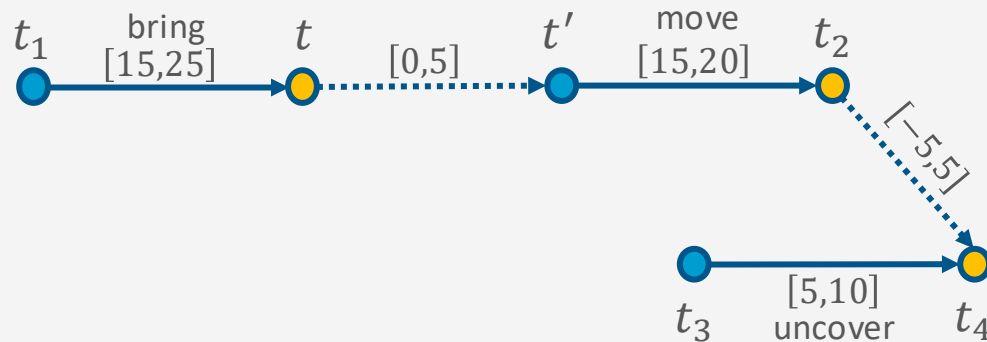
$r_{ij} = [l, u]$ is violated
if t_i and t_j have values
and $t_j - t_i \notin [l, u]$

Dynamic Execution

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

Example

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



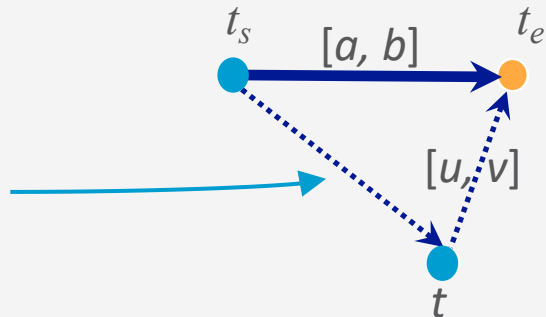
- Actor's dynamic execution strategy
 - Trigger t_1 at whatever time you want
 - Wait and observe t
 - Trigger t' at any time from t to $t + 5$
 - Trigger $t_3 = t' + 10$
 - For every $t_2 \in [t' + 15, t' + 20]$ and $t_4 \in [t_3 + 5, t_3 + 10]$
 - $t_4 \in [t' + 15, t' + 20]$
 - So, $t_4 - t_2 \in [-5, 5]$
 - Thus, all constraints are satisfied

Dynamic Controllability Checking

- For a chronicle $\phi = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
 - Temporal constraints in \mathcal{C} correspond to an STNU
 - Adapt TemPlan to test not only consistency but also dynamic controllability (*) of the STNU
 - If we detect cases where it is not dynamically controllable, then backtrack
- * Use PC as well
 - If $\text{PC}(\mathcal{V} \cup \tilde{\mathcal{V}}, \mathcal{E} \cup \tilde{\mathcal{E}})$ reduces a contingent constraint, $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ not dyn. controllable \Rightarrow Can prune this branch
 - Otherwise: unknown if $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ dynamically controllable
 - Only **necessary**, **not sufficient** condition
 - Two options
 - Continue down this branch and backtrack later if necessary
 - Extend PC to detect more cases where $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is not dynamically controllable (additional constraint propagation rules)

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 $[a', b'] \circ [u, v] = [a, b]$
 - $[a' + u, b' + v] = [a, b]$
 - $a' = a - u, b' = b - v$



Should be $[a', b']$
if I am not mistaken

Conditions	Propagated constraint
$t_s \xRightarrow{[a,b]} t_e, t \xrightarrow{[u,v]} t_e, u \geq 0$	$t_s \xrightarrow{[b',a']} t$
$t_s \xRightarrow{[a,b]} t_e, t \xrightarrow{[u,v]} t_e, u < 0, v \geq 0$	$t_s \xrightarrow{\langle t_e, b' \rangle} t$
$t_s \xRightarrow{[a,b]} t_e, t_s \xrightarrow{\langle t_e, u \rangle} t$	$t_s \xrightarrow{[\min\{a, u\}, \infty]} t$
$t_s \xrightarrow{\langle t_e, b \rangle} t, t' \xrightarrow{[u,v]} t$	$t_s \xrightarrow{\langle t_e, b' \rangle} t'$
$t_s \xrightarrow{\langle t_e, b \rangle} t, t' \xRightarrow{[u,v]} t, t_e \neq t$	$t_s \xrightarrow{\langle t_e, b-u \rangle} t'$

\Rightarrow contingent

\rightarrow controllable

$a' = a - u, b' = b - v$

Extended Version of PC

- We want a fast algorithm that TemPlan can run at each node, to decide whether to backtrack
- There is 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

Conditions	Propagated constraint
$t_s \xRightarrow{[a,b]} t_e , t \xrightarrow{[u,v]} t_e , u \geq 0$	$t_s \xrightarrow{[b',a']} t$
$t_s \xRightarrow{[a,b]} t_e , t \xrightarrow{[u,v]} t_e , u < 0 , v \geq 0$	$t_s \xrightarrow{\langle t_e, b' \rangle} t$
$t_s \xRightarrow{[a,b]} t_e , t_s \xrightarrow{\langle t_e, u \rangle} t$	$t_s \xrightarrow{[\min\{a, u\}, \infty]} t$
$t_s \xrightarrow{\langle t_e, b \rangle} t , t' \xrightarrow{[u,v]} t$	$t_s \xrightarrow{\langle t_e, b' \rangle} t'$
$t_s \xrightarrow{\langle t_e, b \rangle} t , t' \xRightarrow{[u,v]} t , t_e \neq t$	$t_s \xrightarrow{\langle t_e, b-u \rangle} t'$

 \Rightarrow contingent

 \rightarrow controllable

 $a' = a - u, b' = b - v$

Intermediate Summary

- Constraint management
 - Consistency of object constraints
 - Constraint-satisfaction problem
 - Consistency of time constraints
 - STN, solution, minimality, consistency
 - PC
- Controllability
 - STNU, controllable, contingent
 - Dynamic controllability

Outline per the Book

4.2 Representation

- Timelines
- Actions and tasks
- Chronicles

4.3 Temporal Planning

- Resolvers and flaws
- Search space

4.4 Constraint Management

- Consistency of object constraints and time constraints
- Controlling the actions when we do not know how long they will take

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

- TemPlan's
action template
(descriptive model)

```
leave(r,d,w)  
  assertions: [ts,te] loc(r): (d,w)  
              [ts,te] occupant(d): (r,empty)  
  constraints:  $t_e \leq t_s + \delta_1$   
              adj(d,w)
```

- RAE's
refinement method
(operational model)

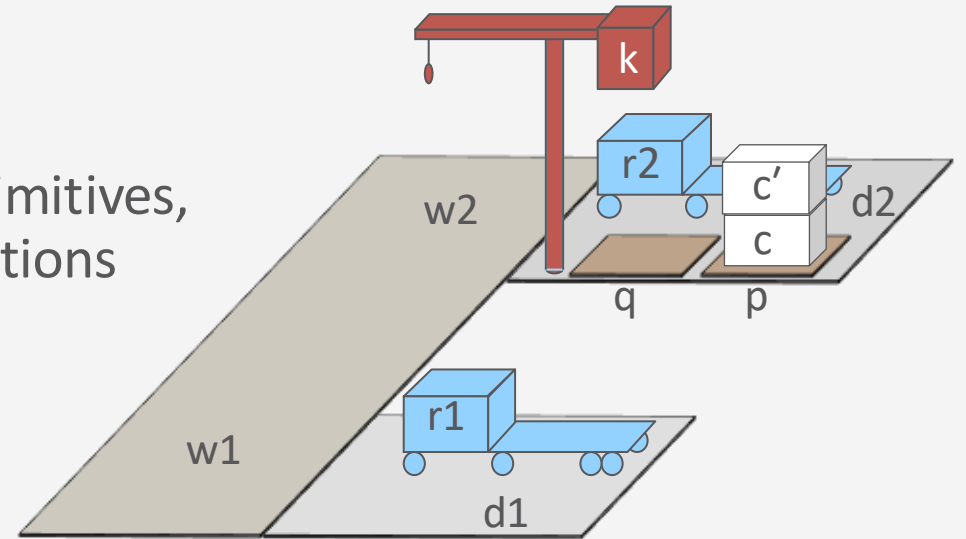
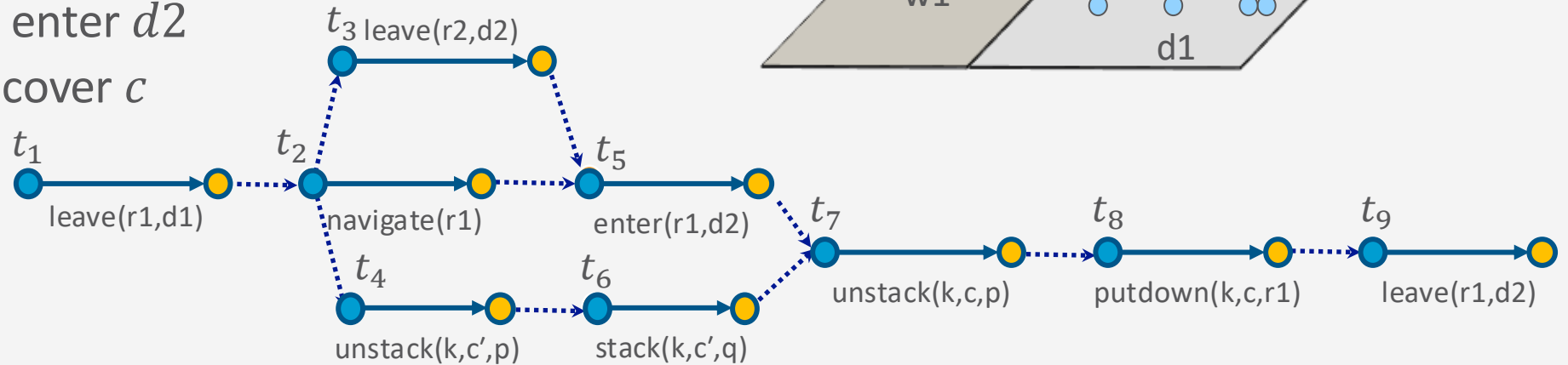
```
m-leave(r,d,w,e)  
  task: leave(r,d,w)  
  pre: loc(r)=d, adj(d,w), exit(e,d,w)  
  body: until empty(e)  
        wait(1)  
        goto(r,e)
```

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., synchronise 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, it 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 $r1$



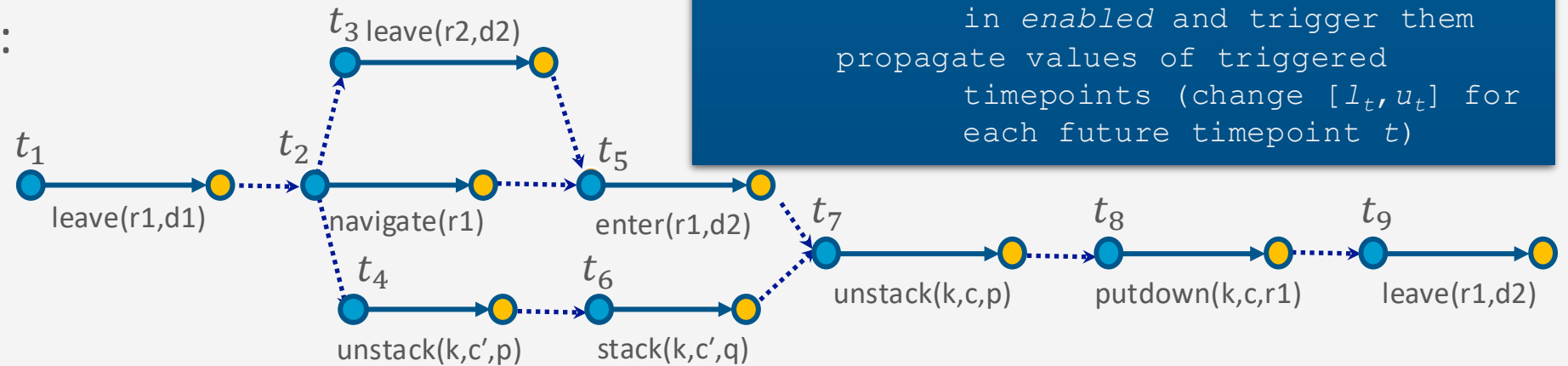
Dispatching

- Let $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ be a controllable STNU that is **grounded**
 - Different from a grounded expression in logic
 - At least one time point t^* is instantiated
 - 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 is alive
 - For every precedence constraint $t' < t$, t' has occurred
 - For every wait constraint $\langle t_e, \alpha \rangle$, t_e has occurred or α has expired
 - α has expired if t_s has occurred and $t_s + \alpha \leq now$

```
Dispatch ( $\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}}$ )  
  initialise the network  
  while there are time points in  $\mathcal{V}$  that  
    have not been triggered do  
    update now  
    update the time points in  $\tilde{\mathcal{V}}$  that have  
      been newly observed  
    update enabled  
    trigger every  $t \in enabled$  s.t.  $now = u_t$   
    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 , enables t_3, t_4
- Trigger t_3 soon enough to allow $enter(r1, d2)$ 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 ($\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}}$)

initialise the network

while there are time points in \mathcal{V} that have not been triggered **do**

 update *now*

 update the time points in $\tilde{\mathcal{V}}$ that have been newly observed

 update *enabled*

 trigger every $t \in enabled$ s.t. $now = u_t$

 arbitrarily choose other time points

 in *enabled* and trigger them

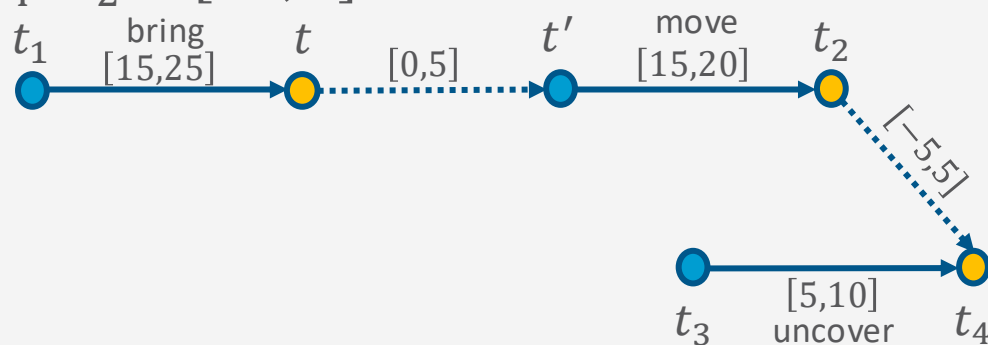
 propagate values of triggered

 timepoints (change $[l_t, u_t]$ for

 each future timepoint t)

Example from Slide 61

- 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 $t_4 - t_2 \in [-5, 5]$



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

initialise the network

while there are time points in \mathcal{V} that have not been triggered **do**

 update *now*

 update the time points in $\tilde{\mathcal{V}}$ that have been newly observed

 update *enabled*

 trigger every $t \in \text{enabled}$ s.t. $\text{now} = u_t$

 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)

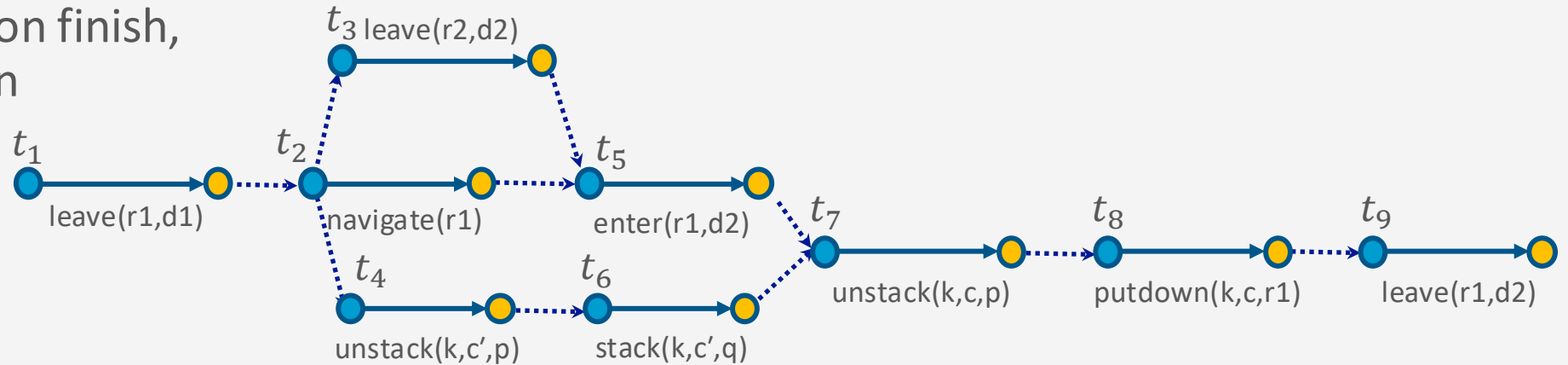
Dispatching

- Propagation step most costly one
 - $O(n^3)$
 - n the number of remaining future time points in network
- Ideally propagation fast enough to allow iterations and updates of *now* consistent with temporal granularity of plan

```
Dispatch ( $\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}}$ )  
  initialise the network  
  while there are time points in  $\mathcal{V}$  that  
    have not been triggered do  
    update now  
    update the time points in  $\tilde{\mathcal{V}}$  that have  
      been newly observed  
    update enabled  
    trigger every  $t \in \text{enabled}$  s.t.  $\text{now} = u_t$   
    arbitrarily choose other time points  
      in enabled and trigger them  
    propagate values of triggered  
      timepoints (change  $[l_t, u_t]$  for  
      each future timepoint  $t$ )
```

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



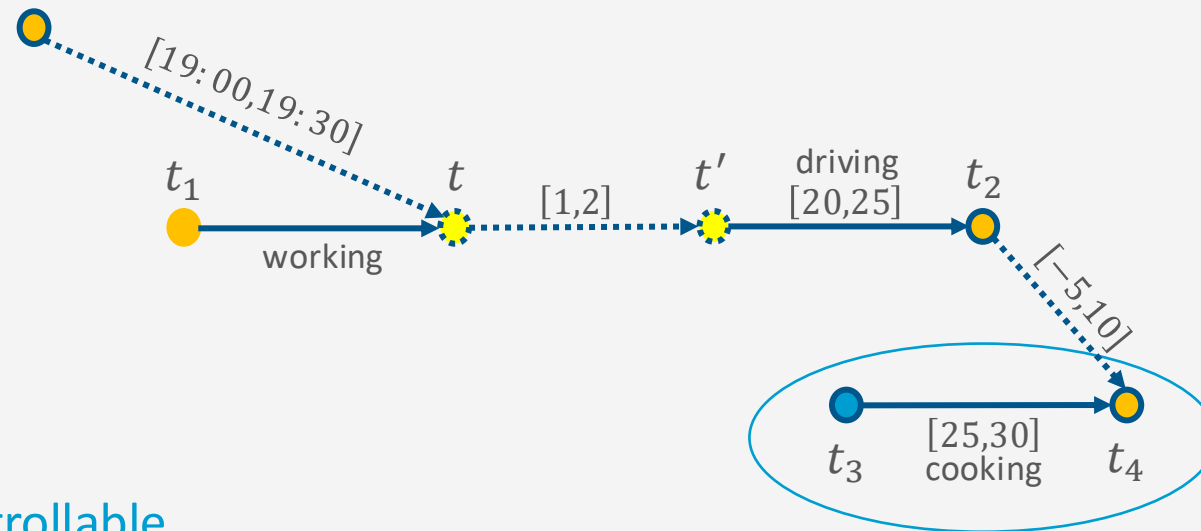
Partial Observability

- 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)
 - STNU where the contingent time points are given by a set of invisible and a set of observable timepoints
 - POSTNU = STNU if Invisible = \emptyset
 - Dynamically controllable?



Observation Actions

- Example



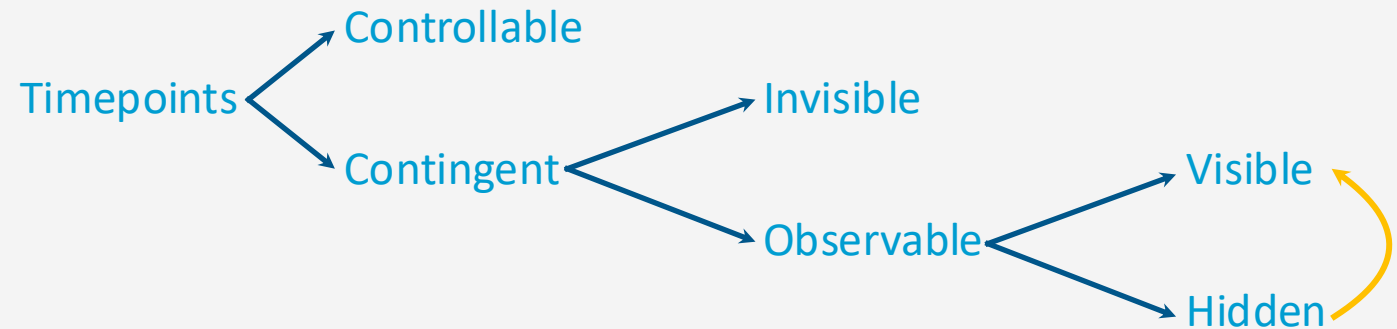
- Controllable
- Contingent
 - Invisible
 - observable

Dynamic Controllability

- 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
- Check dynamic controllability
 - Map an POSTNU to an STNU by deleting invisible time points and adding corresponding constraints on controllable and observable time points
 - Check dynamic controllability of the mapped STNU
 - E.g., using the extended PC algorithm
 - More details in the paper

Dynamic Controllability

- 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 \neq visible
 - Observable means it will be known **when observed**
 - It can be temporarily **hidden**
 - Aim: Find out which time points need to be observed for the plan to be dynamically controllable (details in paper)



Intermediate Summary

- Acting
 - Atemporal refinement
 - eRAE
 - Dispatching
 - Alive, enabled
 - Deadline failures
 - Partial observability
 - Invisible, observable (hidden/visible)

Outline per the Book

4.2 Representation

- Timelines
- Actions and tasks
- Chronicles

4.3 Temporal Planning

- Resolvers and flaws
- Search space

4.4 Constraint Management

- Consistency of object constraints and time constraints
- Controlling the actions when we do not know how long they will take

4.5 Acting with Temporal Models

- Acting with atemporal refinement
- Dispatching
- Observation actions

⇒ Next: Planning and Acting with Nondeterministic Models