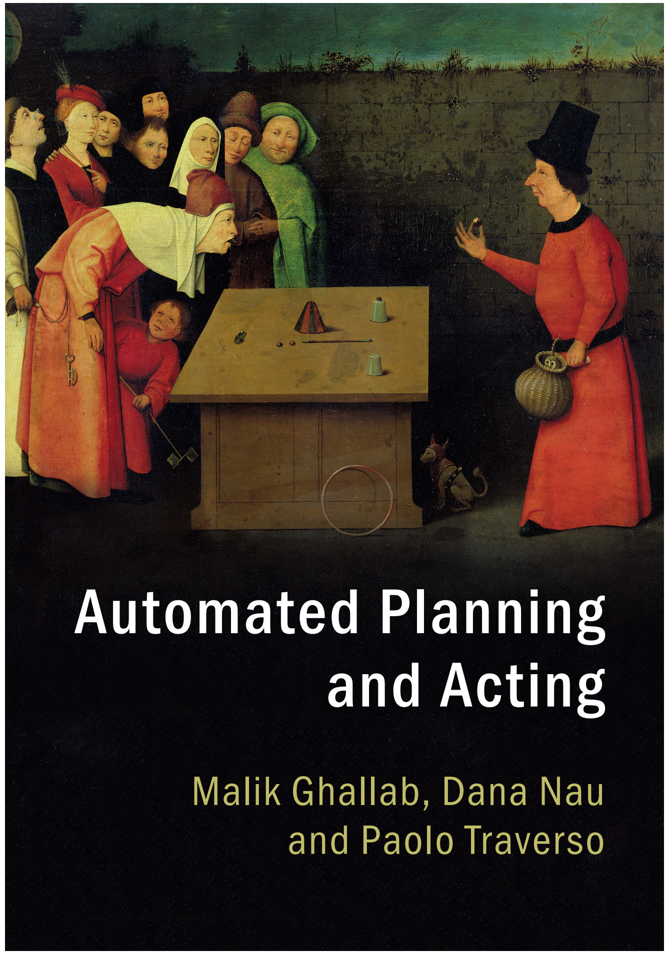
Last update: 5:03 PM, April 16, 2022

Chapter 4
Deliberation with Temporal Domain Models

Dana S. Nau
University of Maryland

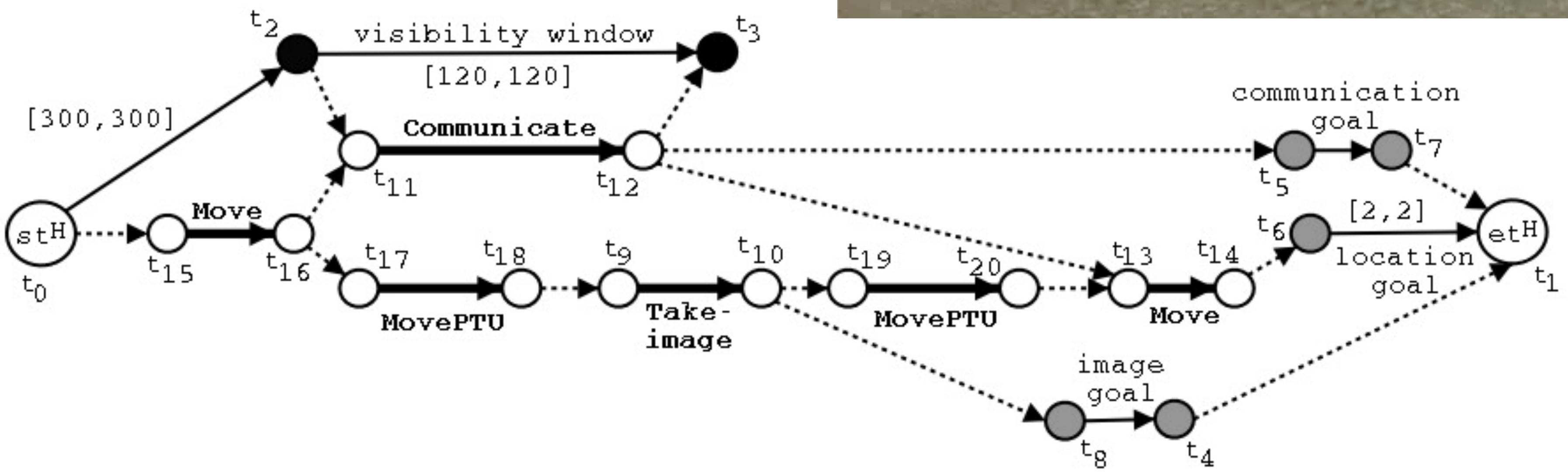


http://www.laas.fr/planning

IxTeT

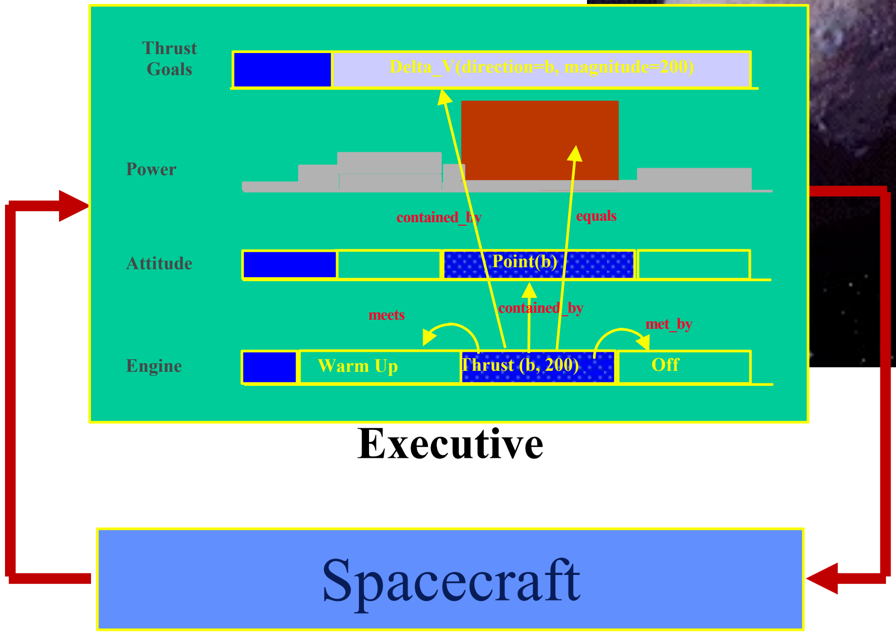
- LAAS/CNRS, Toulouse, France
- mid-1990s
- Video:

  https://www.cs.umd.edu/~nau/apa/ixtet.mov



RAX/PS

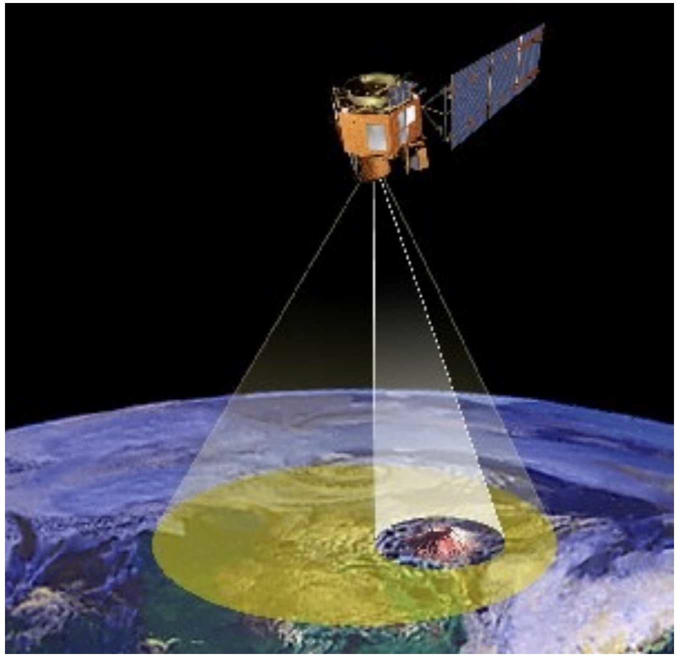
- Planning/control of DS1 spacecraft
- NASA Ames and JPL, 1999



T-ReX

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

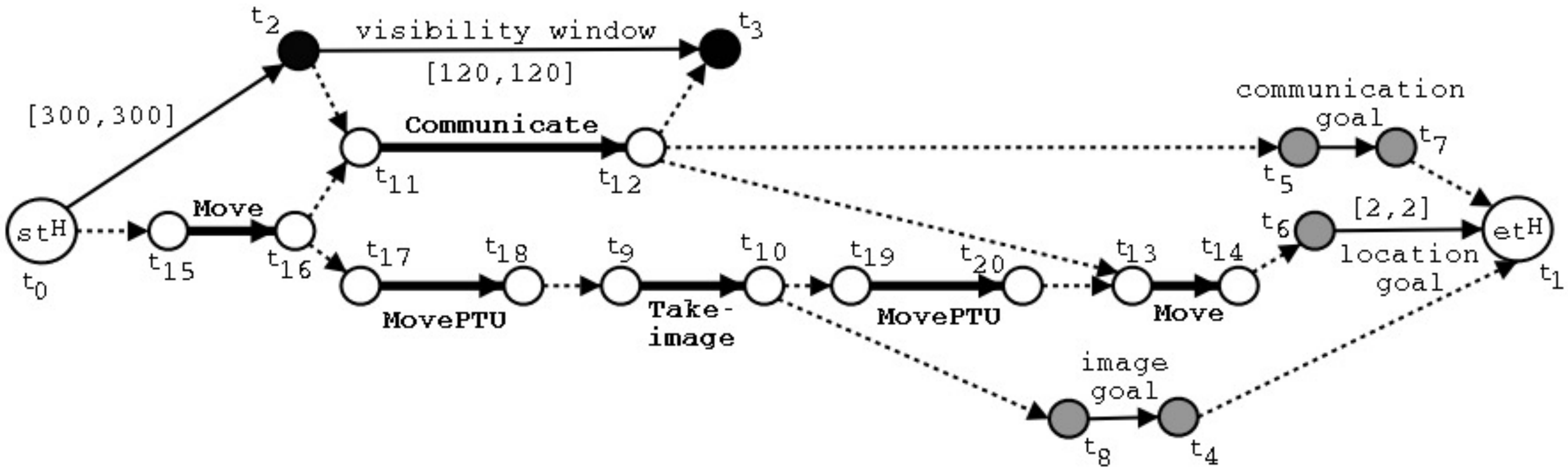
Casper (NASA JPL)



- Planning/control of spacecraft
- NASA JPL, ongoing

Temporal Models

- Constraints on state variables and events
  - Reflect predicted actions and events
- Actions have duration
  - preconditions and effects may occur at times other than start and end
- Time constraints on goals
  - relative or absolute
- Exogenous events expected to occur in the future
- Maintenance actions: maintain a property
  - e.g., track a moving target, keep a door closed
- Concurrent actions
  - interacting effects, joint effects
- Delayed commitment
  - instantiation at acting time



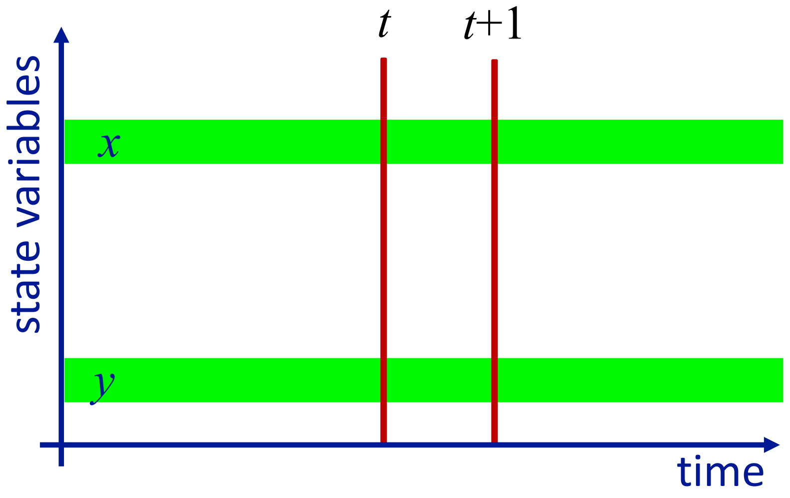
Outline

- ✓ Introduction
- **4.2 Representation**
  - Timelines
  - Actions and tasks
  - Chronicles
- 4.3 Temporal planning
- 4.4 Constraint management
- 4.5 Acting with temporal models

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"
  - Discrete: time points are integers
    - $t = 1, 2, 3, \ldots$
  - For each state variable x, a *timeline*
    - values of x during different time intervals
  - State at time t = {state-variable values at time t}

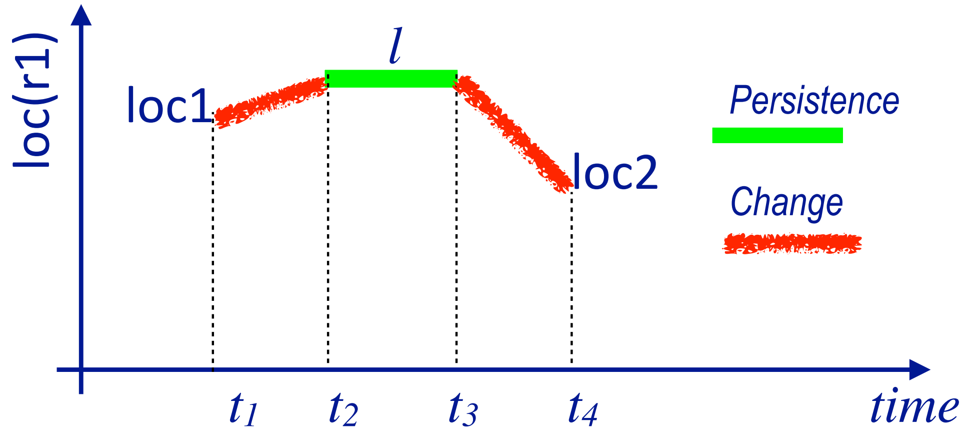


Timeline

- A pair $(\mathcal{T}, \mathcal{C})$
  - partially predicted evolution of one state variable
- $\mathcal{T}$: temporal assertions
  - $[t_1, t_2]$ loc(r1) : (loc1, l)
  - $[t_2, t_3]$ loc(r1) = l
  - $[t_3, t_4]$ loc(r1) : (l, loc2)
- $\mathcal{C}$: constraints
  - $t_1 < t_2 < t_3 < t_4$
  - $l \neq$ loc1
  - $l \neq$ loc2

persistence requires $t_1 \leq t_2$

change requires $t_3 \leq t_4$ and $l \neq$ loc2



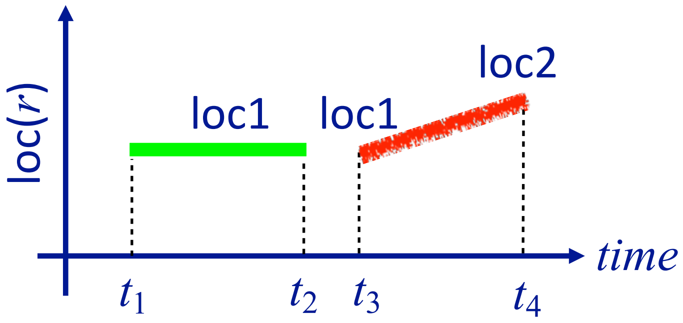
- If $\mathcal{T}$ contains $[t,t']\,x : (v,v')$ or $[t,t']\,x = v$ then $\mathcal{C}$ always contains $t \leq t'$
  - To keep the examples from getting cluttered, we'll often be sloppy and not write $t \leq t'$ explicitly

Consistency

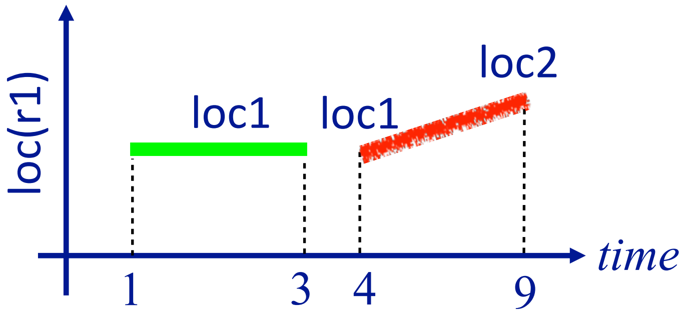
- Let $(\mathcal{T},\mathcal{C})$ be a timeline
- Let $(\mathcal{T}',\mathcal{C}')$ be a ground instance of $(\mathcal{T},\mathcal{C})$
  - $(\mathcal{T}',\mathcal{C}')$ is *consistent* if
    - $\mathcal{T}'$ satisfies $\mathcal{C}'$

      and
    - no state variable in $(\mathcal{T},\mathcal{C})$ has more than one value at a time
- $(\mathcal{T},\mathcal{C})$ is *consistent* if it has *at least one* consistent ground instance

- A consistent timeline:
  - $\mathcal{T}_1 = \{[t_1,t_2]$ loc(r)=loc1, $[t_3,t_4]$ loc(r):(loc1,loc2)$\}$
  - $\mathcal{C}_1 = \{t_1 < t_2 < t_3\}$

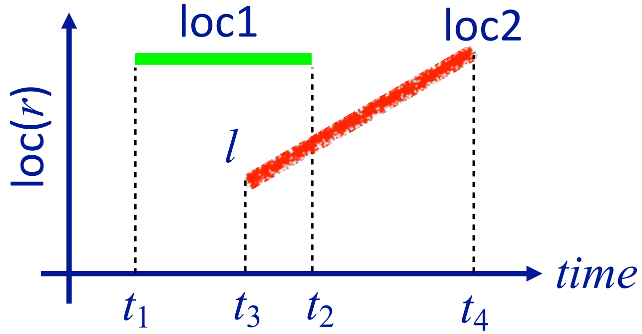


$(\mathcal{T}_1,\mathcal{C}_1)$



a ground instance of $(\mathcal{T}_1,\mathcal{C}_1)$

- **Poll**: is this timeline consistent?
  - $\mathcal{T}_2 = \{[t_1,t_2]$ loc(r)=loc1, $[t_3,t_4]$ loc(r):$(l$,loc2)$\}$
  - $\mathcal{C}_2 = \{t_1 < t_3 < t_2\}$

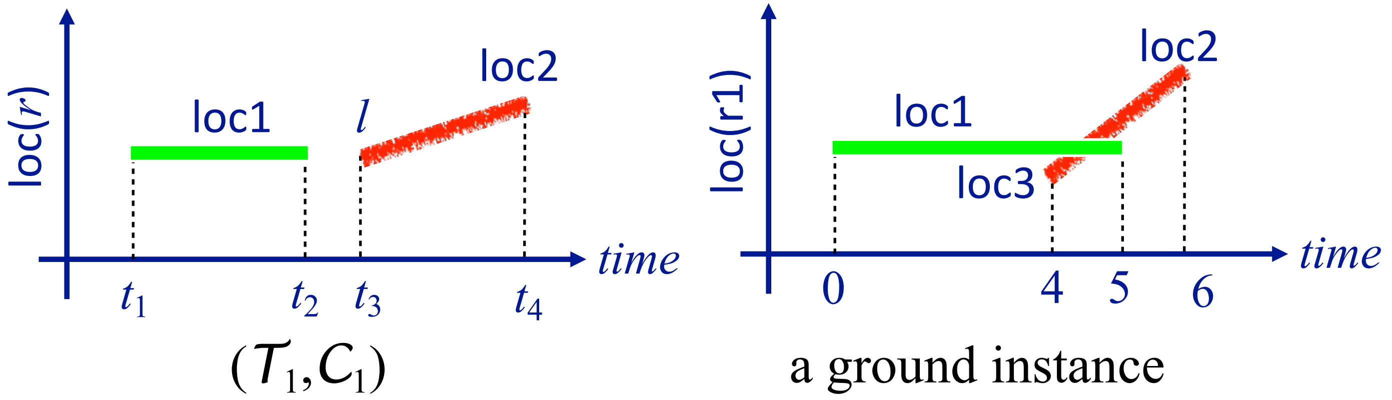


Security

- $(\mathcal{T},\mathcal{C})$ is *secure* if
  - it's consistent (has *at least one* consistent ground instance)
  - *every* ground instance that satisfies the constraints is consistent
- In PSP (Chapter 2), analogous to a partial plan that has no threats

- Can make a consistent timeline secure by adding *separation constraints*:
  - $r \neq$ r1
  - $t_2 < t_3$
  - $t_4 < t_1$
  - $t_2 = t_3$, r = r1 , l = loc1
  - $t_4 = t_1$, r = r1 , l = loc1
- Analogous to resolvers in PSP

- Not secure:
  - $\mathcal{T}_1$ = {[t_1,t_2] loc(r) = loc1, [t_3,t_4] loc(r) : (l,loc2)}
  - $\mathcal{C}_1$ = {$t_1 < t_2$, $t_3 < t_4$}



- Separation constraints:
  - $t_2 < t_3$
  - $t_2 = t_3$, l = loc1

Union of Multiple Timelines

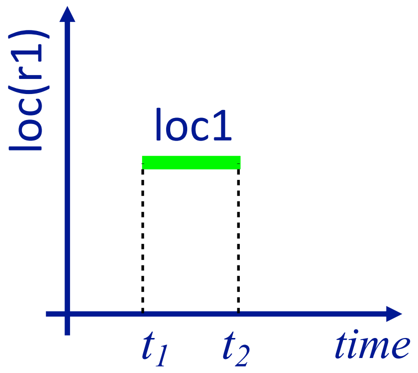
- Timelines for k different state variables, all of which are fully ground:
  - $(\mathcal{T}_1,\mathcal{C}_1)$, …, $(\mathcal{T}_k,\mathcal{C}_k)$

book omits this part

- Union is $(\mathcal{T},\mathcal{C})$:
  - $\mathcal{T} = \mathcal{T}_1 \cup \ldots \cup \mathcal{T}_k$
  - $\mathcal{C} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_k$

- If every $(\mathcal{T}_i,\mathcal{C}_i)$ is secure, then $(\mathcal{T}_1,\mathcal{C}_1) \cup \ldots \cup (\mathcal{T}_k,\mathcal{C}_k)$ is also secure

Causal support



- Consider the assertion $[t_1,t_2]$ loc(r1) = loc1
  - How did r1 get to loc1 in the first place?

- Let α be either $[t_1,t_2]\ x = v_1$ or $[t_1,t_2]\ x : (v_1, v_2)$
- *Causal support* for α
  - Information saying α is supported *a priori*
  - Or another assertion that produces $x = v_1$ at time t_1
    - $[t_0,t_1]\ x = v_1$
    - $[t_0,t_1]\ x : (v_0, v_1)$

- A timeline $\mathcal{T}$ is *causally supported* if every assertion α in $\mathcal{T}$ has a causal support

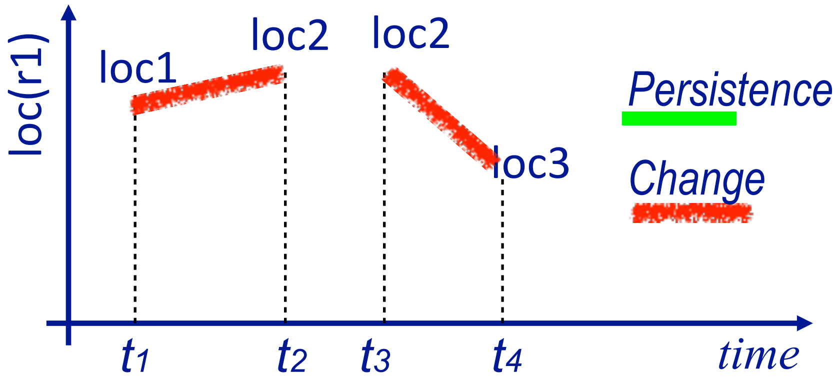
- Three ways to modify a timeline to add causal support …

Establishing causal support

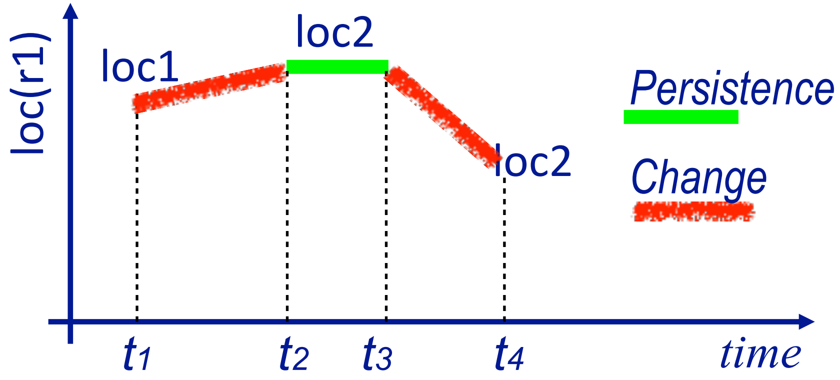
(1) Add a persistence assertion

$\mathcal{T}$ = {[t_1,t_2] loc(r1):(loc1,loc2),
[t_3,t_4] loc(r1):(loc2,loc3)}

$C = \{t_1 < t_2 < t_3 < t_4\}$



- Add [t_2,t_3] loc(r1) = loc2
  - Supported by the first temporal assertion
  - Supports the second one

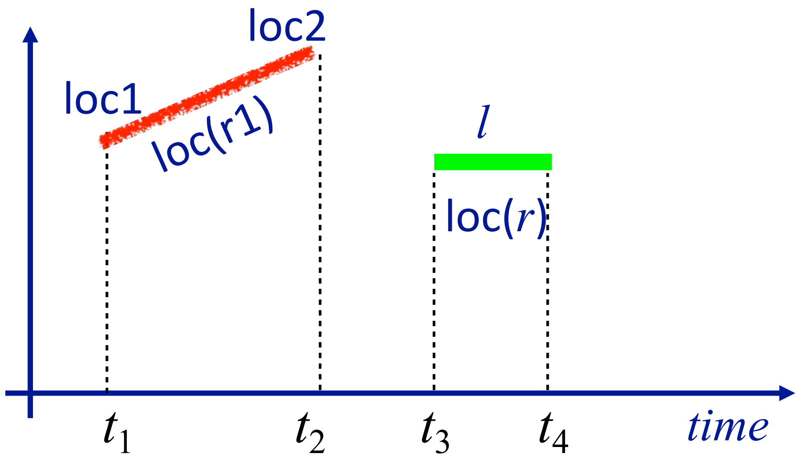


Establishing causal support

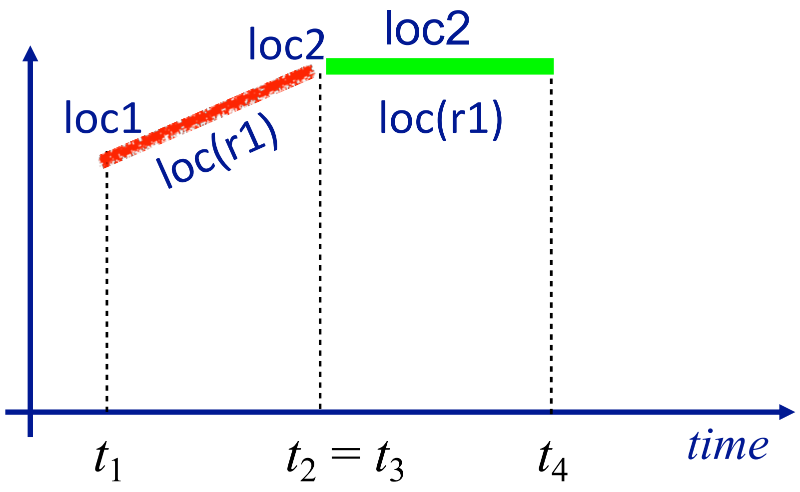
(2) Add constraints

$\mathcal{T} = \{[t_1,t_2]$ loc(r1):(loc1,loc2),
$[t_3,t_4]$ loc(r) = $l\}$

$C = \{t_1 < t_2,\ t_3 < t_4\}$



- Add $t_2 = t_3$, r = r1, l = loc2

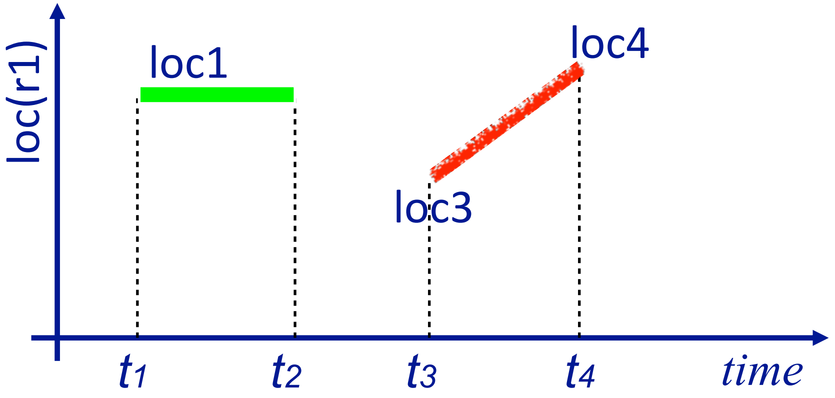


Establishing causal support

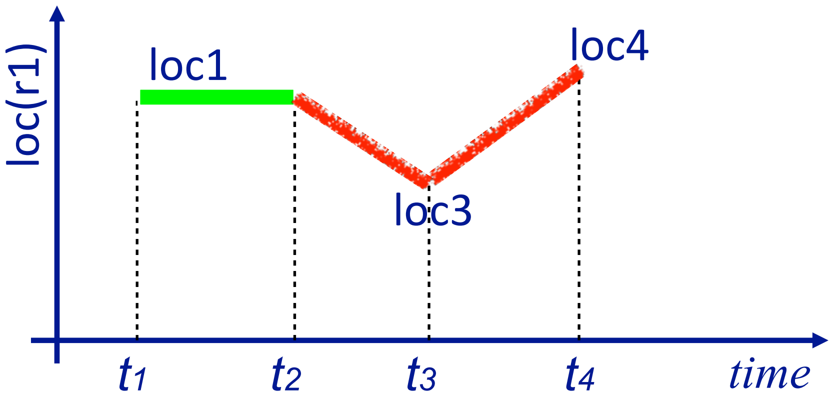
(3) Add a change assertion
(by adding an action)

$\mathcal{T} = \{[t_1,t_2]$ loc(r1) = loc1,
$[t_3,t_4]$ loc(r1):(loc3,loc4)$\}$

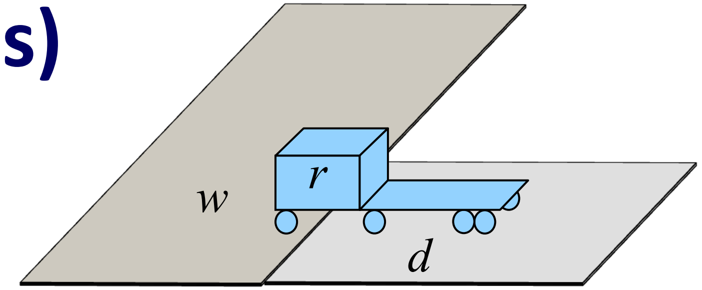
$\mathcal{C} = \{t_1 < t_2 < t_3 < t_4\}$



- Add an action that includes
  $[t_2,t_3]$ loc(r1):(loc1,loc3)



Primitive Tasks (Actions)



- *Action* or *primitive task* (or just *primitive*):
  - a triple (*head*, $\mathcal{T}$, $\mathcal{C}$)
    - *head* is the name and parameters
    - ($\mathcal{T}$, $\mathcal{C}$) is the union of a set of timelines
- Always two additional parameters
  - starting time t_s , ending time t_e
- In each temporal assertion in $\mathcal{T}$,
  - left endpoint is like a precondition
    - ⬄ need for causal support
  - right endpoint is like an effect

leave(r,d,w)

// robot r goes from loading dock d to waypoint w

assertions:

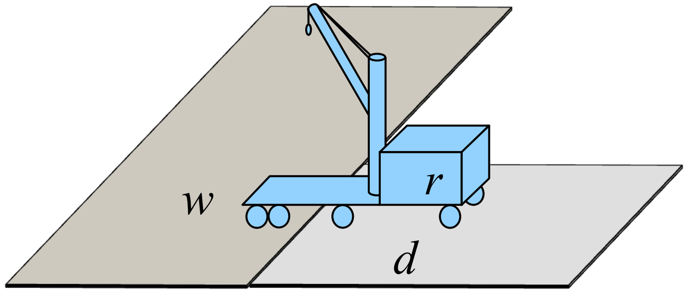
$[t_s,t_e]$ loc(r): (d,w)
$[t_s,t_e]$ occupant(d): (r,empty)

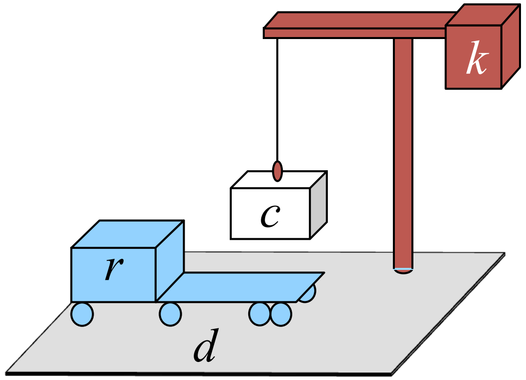
constraints:

$t_e \leq t_s + \delta_1$
adjacent(d,w)

- Action duration $t_e - t_s \leq \delta_1$

Primitive Tasks (Actions)





enter(r,d,w)

// robot r goes from waypoint w to loading dock d

assertions:

$[t_s,t_e]$ loc(r): (w,d)
$[t_s,t_e]$ occupant(d): (empty,r)

constraints:

$t_e \leq t_s + \delta_2$
adjacent(d,w)

- Action duration $t_e - t_s \leq \delta_2$
- Dock d becomes occupied by r

take(k,c,r,d)

// crane k takes container c from robot r

assertions:

$[t_s,t_e]$ pos(c): (r, k) // where c is
$[t_s,t_e]$ grip(k): (empty, c) // what's in k's gripper
$[t_s,t_e]$ freight(r): (c,empty) // what r is carrying
$[t_s,t_e]$ loc(r) = d // where r is

constraints:

attached(k,d)

Primitive Tasks (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 robot r
- put(k,c,r) — crane k puts container c onto robot r

- navigate(r,w,w') — r navigates from waypoint w to connected waypoint 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 pile p

c, c' - containers

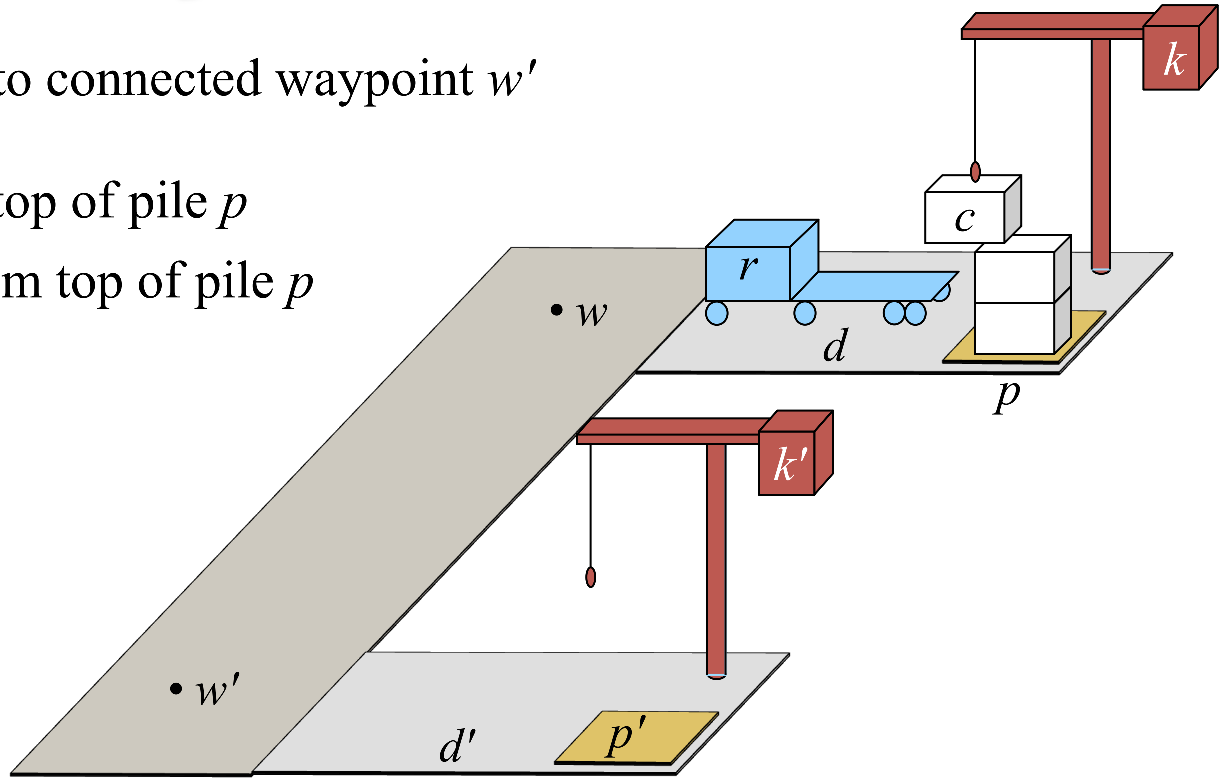
d, d' - loading docks

k, k' - cranes

p, p' - piles

r - robot

w, w' - waypoints



Tasks and Methods

- Task: move robot r to dock d
  - $[t_s, t_e]$ 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)

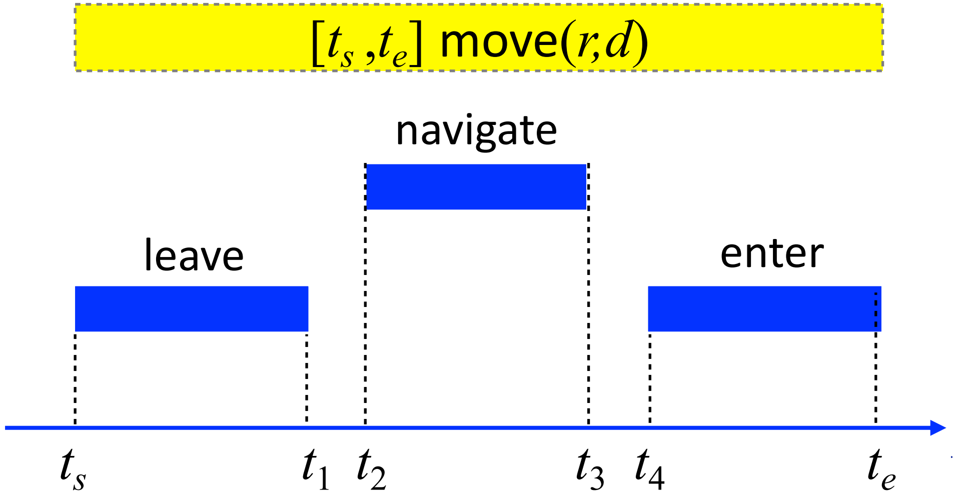
book omits r

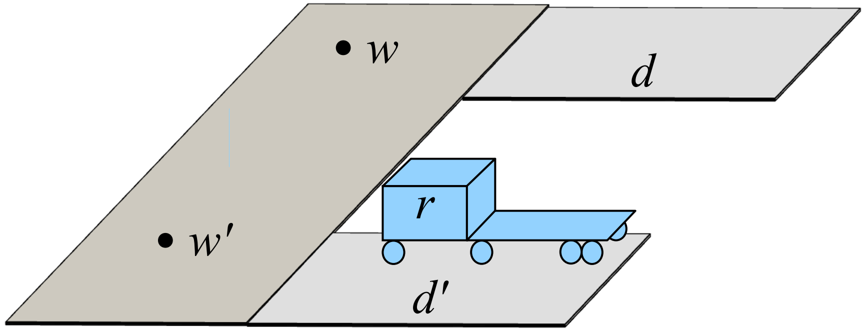
assertions:

$[t_s,t_s+1]$ loc$(r) = d'$

constraints:

adjacent(d,w),
adjacent(d',w'), $d \neq d'$,
connected(w,w'),
$t_1 \le t_2$, $t_3 \le t_4$





Tasks and Methods

- Task: move container c from pile p to robot r
  - $[t_s, t_e]$ load(k,r,c,p)
- Method:

  m-load1(k,r,c,p)

  task: load(k,r,c,p)

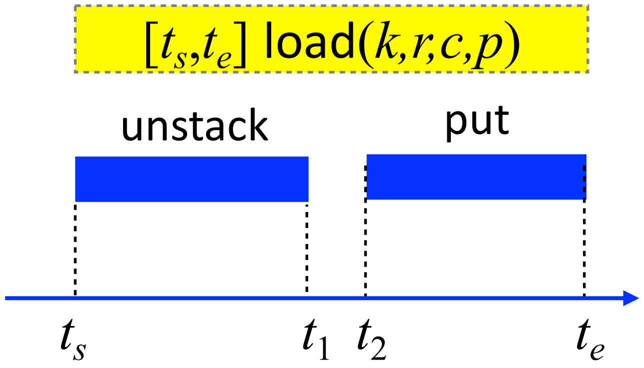
  refinement:

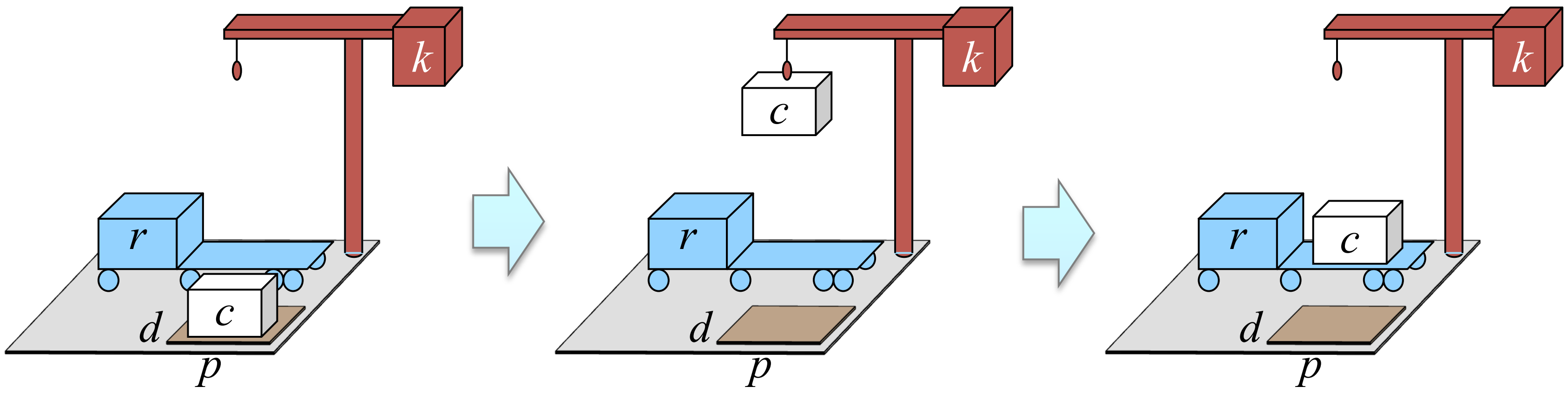
  $[t_s,t_1]$ unstack(k,c,p)
  $[t_2,t_e]$ put(k,c,r)

  assertions:

  constraints:

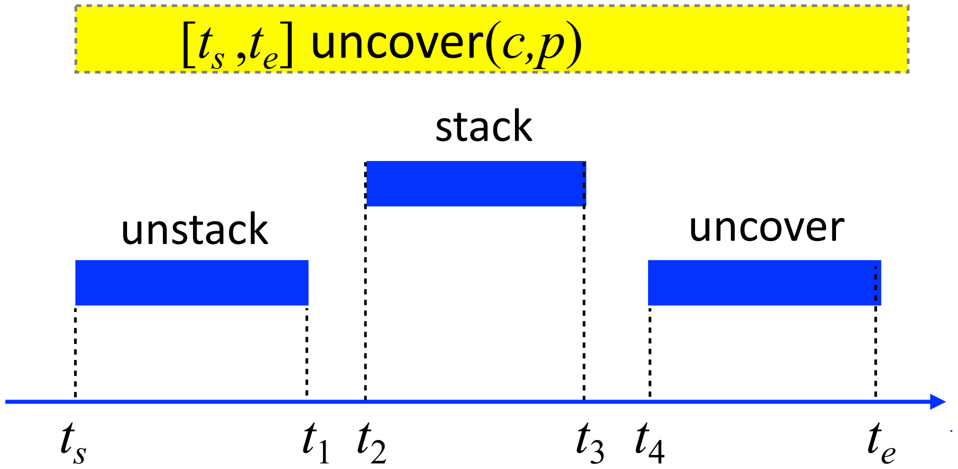
  $t_1 \leq t_2$





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')

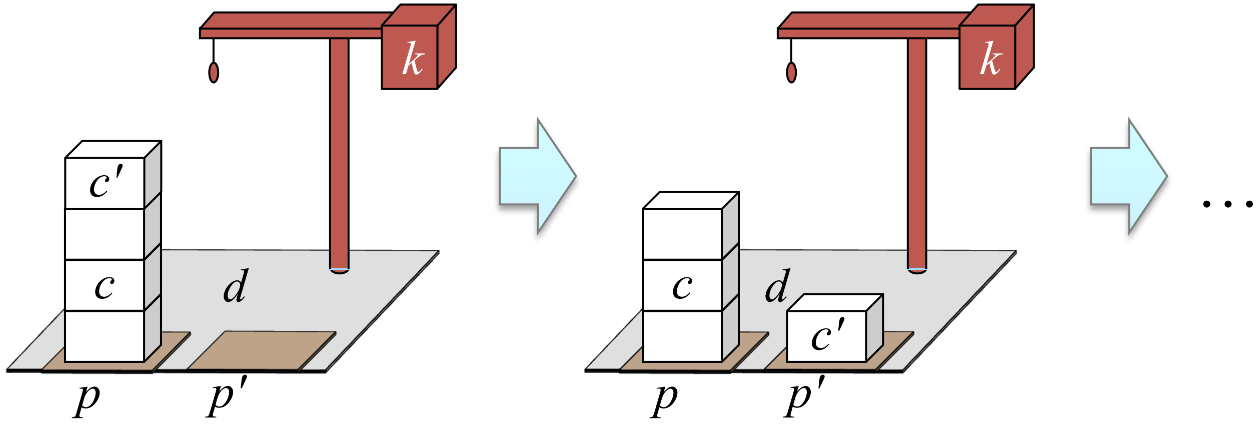
task: uncover(c,p)

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) = empty

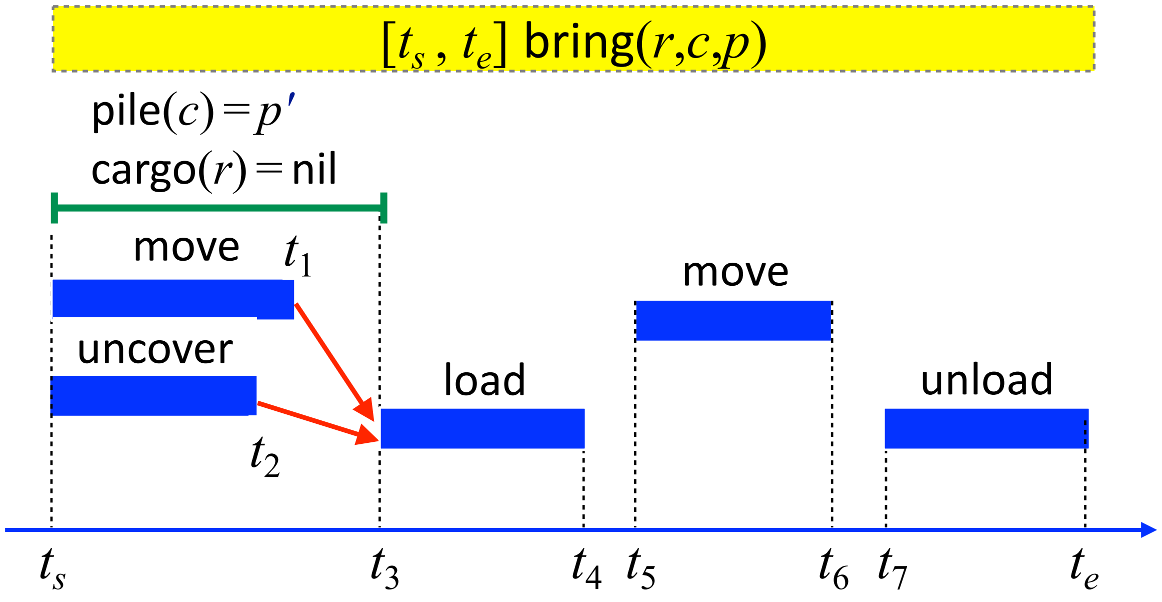
constraints: attached(k,d), attached(p,d), attached(p',d),
$p \neq p'$, $c' \neq c$, $t_1 \leq t_2$, $t_3 \leq t_4$

c = c3 - container
p = p1 - pile it's in
k = k1 - crane
d = d1 - loading dock
p' = p2 - offload pile



Tasks and Methods

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



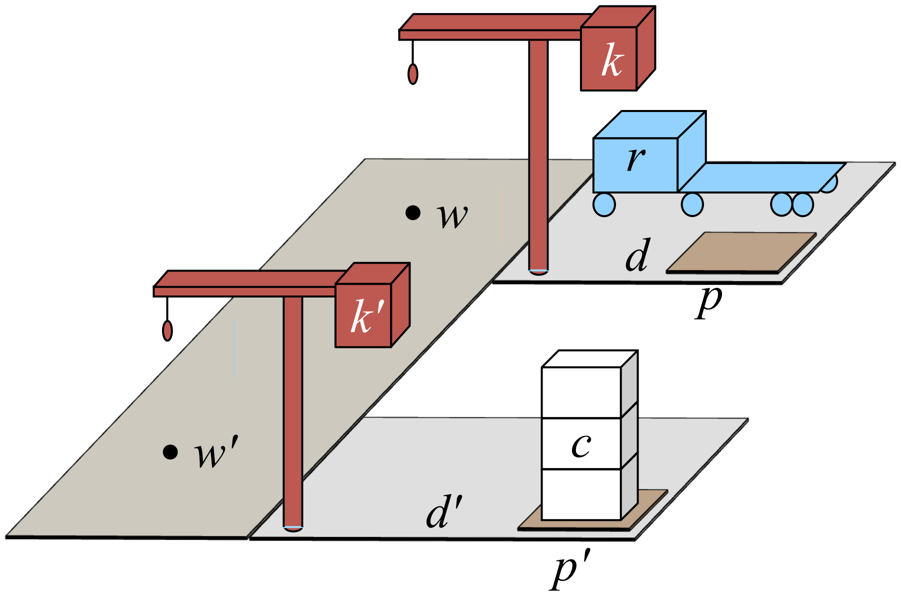
m-bring(r,c,p,p',d,d')

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 tasks
  - S : *a priori* supported assertions
  - $\mathcal{T}$: temporally qualified assertions
  - C: constraints
- ϕ 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

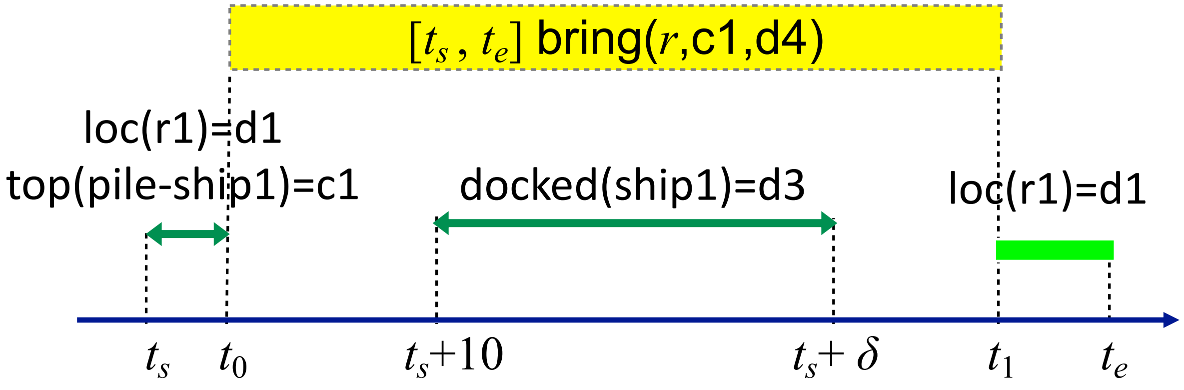
ϕ_0:

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: $t_s = 0 < t_0 < t_1 < t_e$, $20 \leq \delta \leq 30$



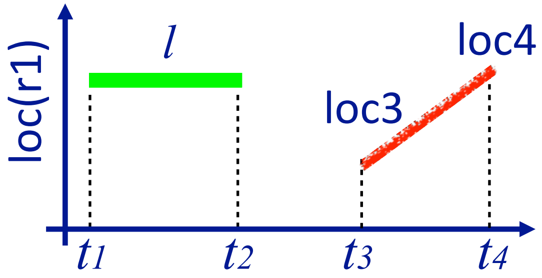
Outline

- ✓ Introduction
- ✓ Representation
- **4.3 Temporal planning**
  - Resolvers and flaws
  - Search space
- 4.4 Constraint management
- 4.5 Acting with temporal models

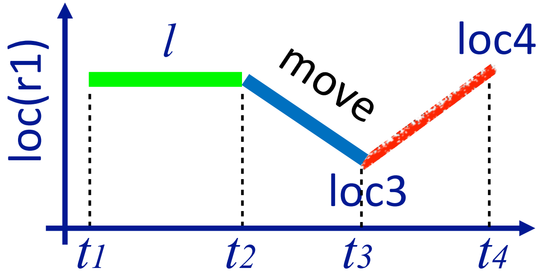
Planning

- Planning problem:
  - a chronicle ϕ_0 that has some *flaws*
    - analogous to flaws in PSP
- To resolve the flaws, add
  - assertions
  - constraints
  - actions

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



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

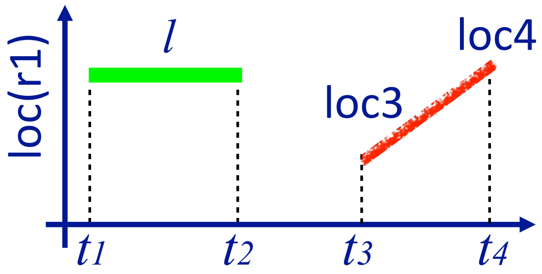


Flaws (1)

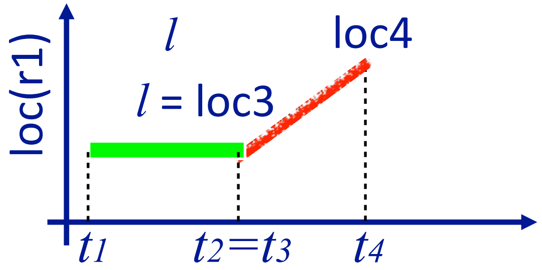
Like an open goal in PSP

1. Temporal assertion α that isn't causally supported

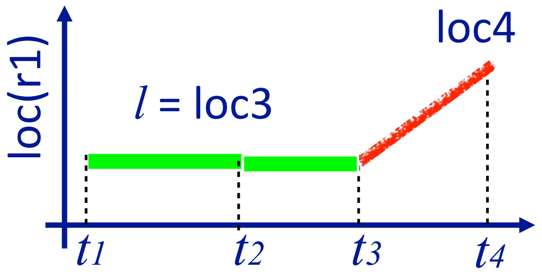
- What causes r1 to be at loc3 at time t_3?



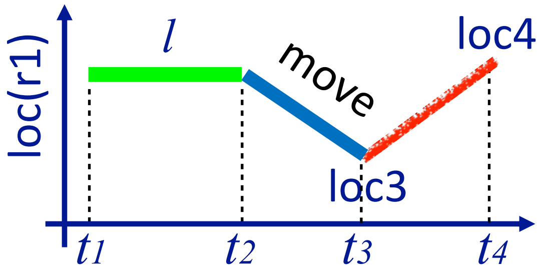
- *Resolvers*:
  - Add constraints* to support α from an assertion in ϕ
    - l = loc3, $t_2 = t_3$



  - Add a new persistence assertion* to support α
    - l = loc3, $[t_2,t_3]$ loc(r1) = loc3



  - Add an action or task to support α
    - $[t_2,t_3]$ move(r1,loc3)
      - refining it will produce an action that supports α (see next slide)



*where the book uses equality constraints, I'll use substitutions

Like a task in HTN planning

Flaws (2)

2. Non-refined task

- *Resolver*: refinement method
  - Applicable if it matches the task and its constraints are consistent with ϕ's
- Applying the resolver:
  - Modify ϕ by replacing the task with m

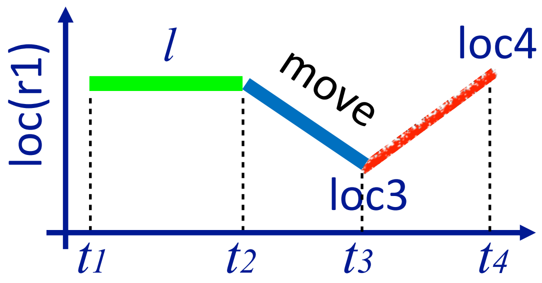
- Example: $[t_2,t_3]$ move(r1,loc3)
  - Refinement will replace it with something like

    $[t_2,t_5]$ leave(r1,l,w)

    $[t_5,t_6]$ navigate(r,w,w')

    $[t_6,t_3]$ enter(r1,loc3,w')

  plus constraints



- 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:

  adjacent(d,w),

  adjacent(d',w'), $d \neq d'$,

  connected(w,w'),

  $t_1 \leq t_2$, $t_3 \leq t_4$

Like a threat in PSP

Flaws (3)

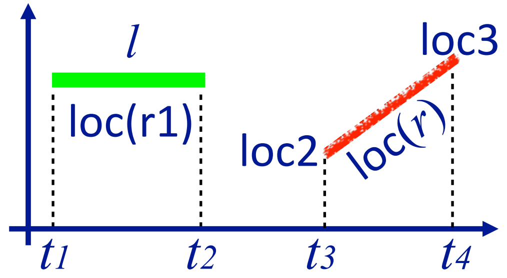
3. A pair of possibly-conflicting temporal assertions

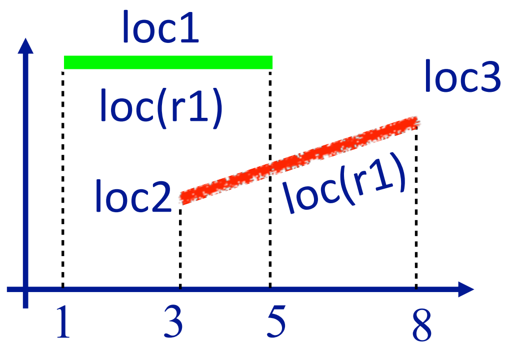
- temporal assertions α and β *possibly conflict* if they can have inconsistent instances

e.g., $[t_1,t_2]$ loc(r1) = loc1, $[t_3,t_4]$ loc(r) : (l, l')

↓ ↓ ↓ ↓ ↓ ↓ ↘

instance: [1,5] loc(r1) = loc1, [3,8] loc(r1) : (loc2,loc3)





- *Resolvers*: separation constraints
  - $r \neq$ r1
  - $t_2 < t_3$
  - $t_4 < t_1$
  - $t_2 = t_3$, 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

Planning Algorithm

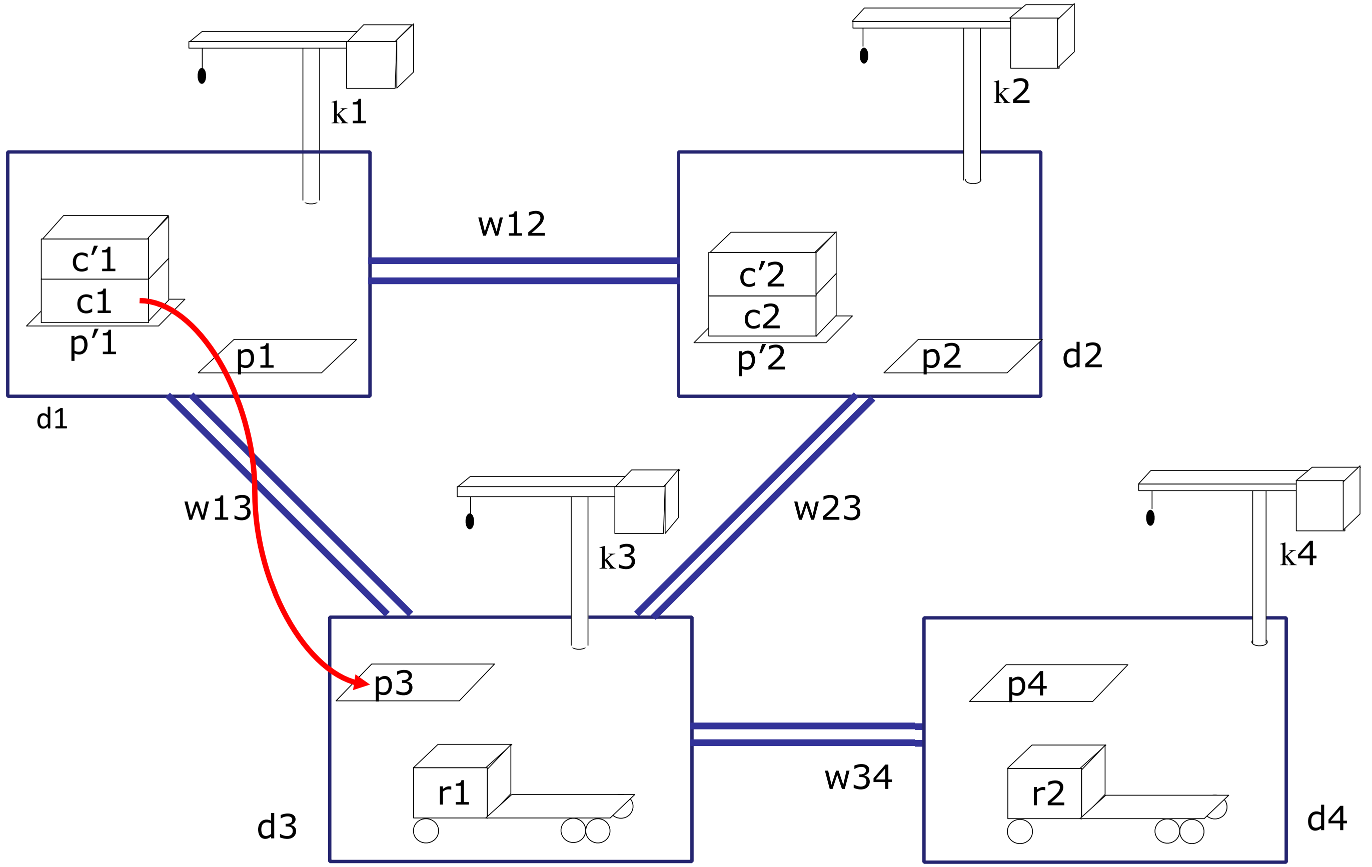
- Like PSP in Chapter 2
  - Repeatedly selects flaws and chooses resolvers
- In the book, TemPlan uses recursion
  - Can be rewritten to use a loop
  - Just programming style, equivalent either way
- Selecting a resolver ρ is an OR-branch
  - backtracking point
- Selecting a flaw f is an AND-branch
  - not a backtracking point, must eventually select all of them
- If it's possible to resolve all flaws, at least one of the nondeterministic execution traces will do so

TemPlan(ϕ, Σ)
  Flaws $\leftarrow$ set of flaws of ϕ
  if *Flaws*$=\varnothing$ then return ϕ
  arbitrarily select $f \in$ *Flaws*
  Resolvers $\leftarrow$ set of resolvers of f
  if *Resolvers*$=\varnothing$ then return failure
  nondeterministically choose $\rho \in$ *Resolvers*
  $\phi \leftarrow$ Transform(ϕ, ρ)
  Templan(ϕ, Σ)

TemPlan(ϕ, Σ)
  loop:
    Flaws $\leftarrow$ set of flaws of ϕ
    if *Flaws*$=\varnothing$ then return ϕ
    arbitrarily select $f \in$ *Flaws*
    Resolvers $\leftarrow$ set of resolvers of f
    if *Resolvers*$=\varnothing$ then return failure
    nondeterministically choose $\rho \in$ *Resolvers*
    $\phi \leftarrow$ Transform(ϕ, ρ)

Example

- $\phi_0 = (\mathcal{A},\mathcal{S},\mathcal{T},\mathcal{C})$
  - Establishes state-variable values at time $t = 0$
  - Flaws: two unrefined tasks
- Select bring(r,c1,p3) to resolve first



ϕ_0: tasks: bring(r,c1,p3)
bring(r',c2,p4)
supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1
[0] pile(c2)=p′2
. . .
assertions: *(none)*
constraints: adjacent(d1,w12), . . .

- Note:
  - [0] $x = v$ means [0,0] $x = v$

Method instance

- One relevant method
  - Instantiate $c \leftarrow$ c1 and $p \leftarrow$ p3 to match bring(r,c1,p3)

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

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

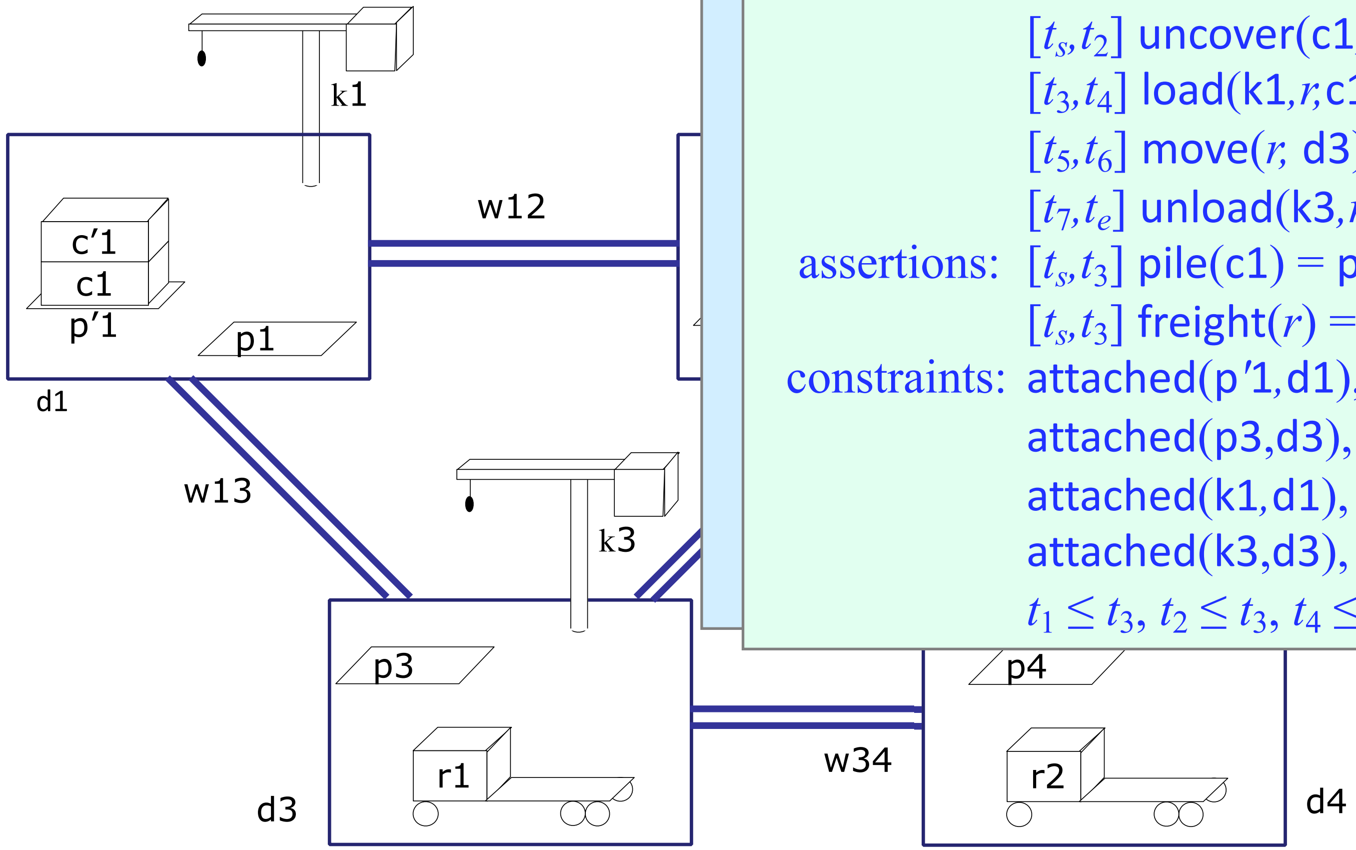
constraints: attached(p′1,d1),

attached(p3,d3), d3 $\neq$ d1

attached(k1,d1),

attached(k3,d3), k3 $\neq$ k1

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



ϕ_0: tasks: bring(r,c1,p3)

bring(r',c2,p4)

supported: [0] loc(r1)=d3

[0] loc(r2)=d4

[0] freight(r1)=empty

[0] freight(r2)=empty

[0] pile(c1)=p′1

[0] pile(c2)=p′2

. . .

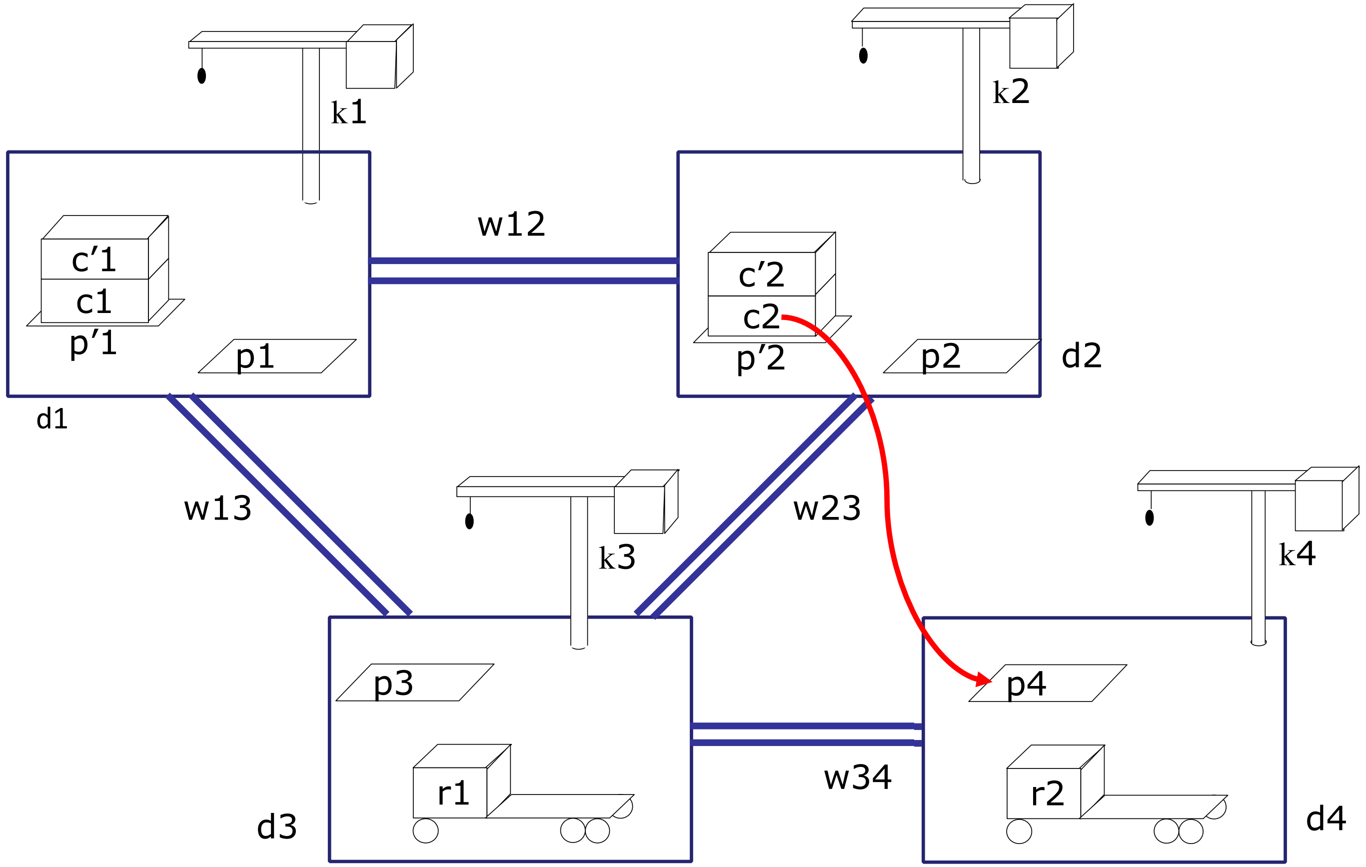
assertions: *(none)*

constraints: adjacent(d1,w12), . . .

Poll: did I instantiate p',d,d',k,k' prematurely?

Apply the method instance

- Changes to ϕ_0
  - Remove bring(r,c1,p3)
  - Add new tasks, assertions, constraints
- Flaws: 6 unrefined tasks, 2 unsupported assertions
- Select bring(r',c2,p4) to resolve next



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] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1
[0] pile(c2)=p′2
. . .

assertions: $[t_s,t_3]$ pile(c1) = p′1
$[t_s,t_3]$ freight(r) = empty

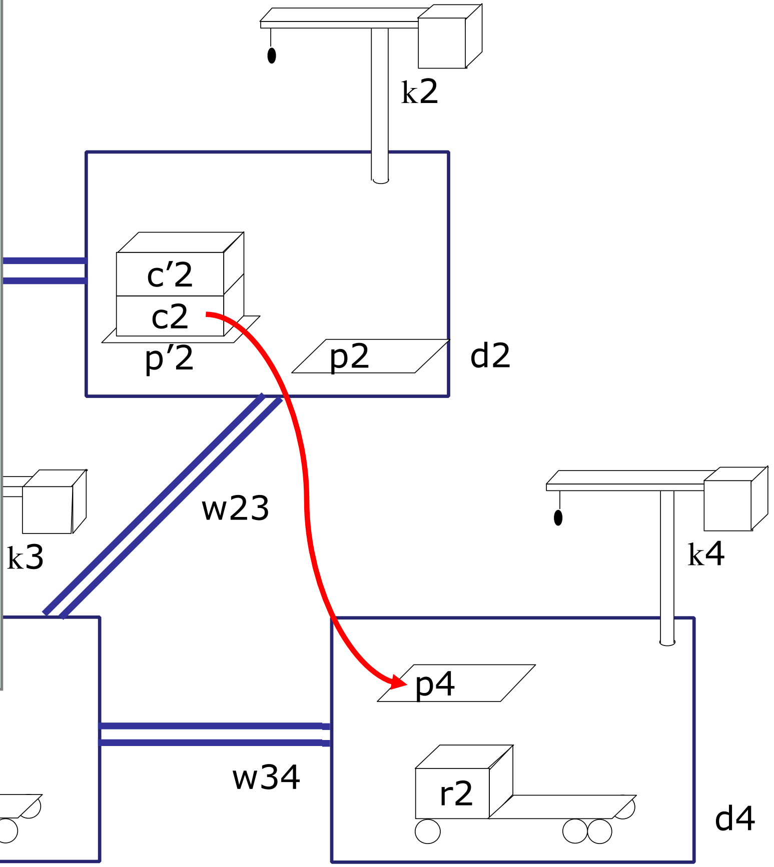
constraints: $t_s<t_1\le t_3,\ 0<t_2\le t_3,\ t_4\le t_5,\ t_6\le t_7$
adjacent(d1,w12), . . .

Method instance

- Instantiate $r \leftarrow r'$, $c \leftarrow$ c2, $p \leftarrow$ p4 to match bring(r',c2,p4)
- Rename timepoint variables t_s, t_e, t_1, …, t_7 to avoid name conflicts

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

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
$[t'_s,t'_1]$ freight(r') = empty
constraints: attached(p′2,d2),
attached(p4,d4), d4 ≠ d2
attached(k2,d2),
attached(k4,d4), k4 ≠ k2
$t'_1 \le t'_3,\ t'_2 \le t'_3,\ t'_4 \le t'_5,\ t'_6 \le t'_7$

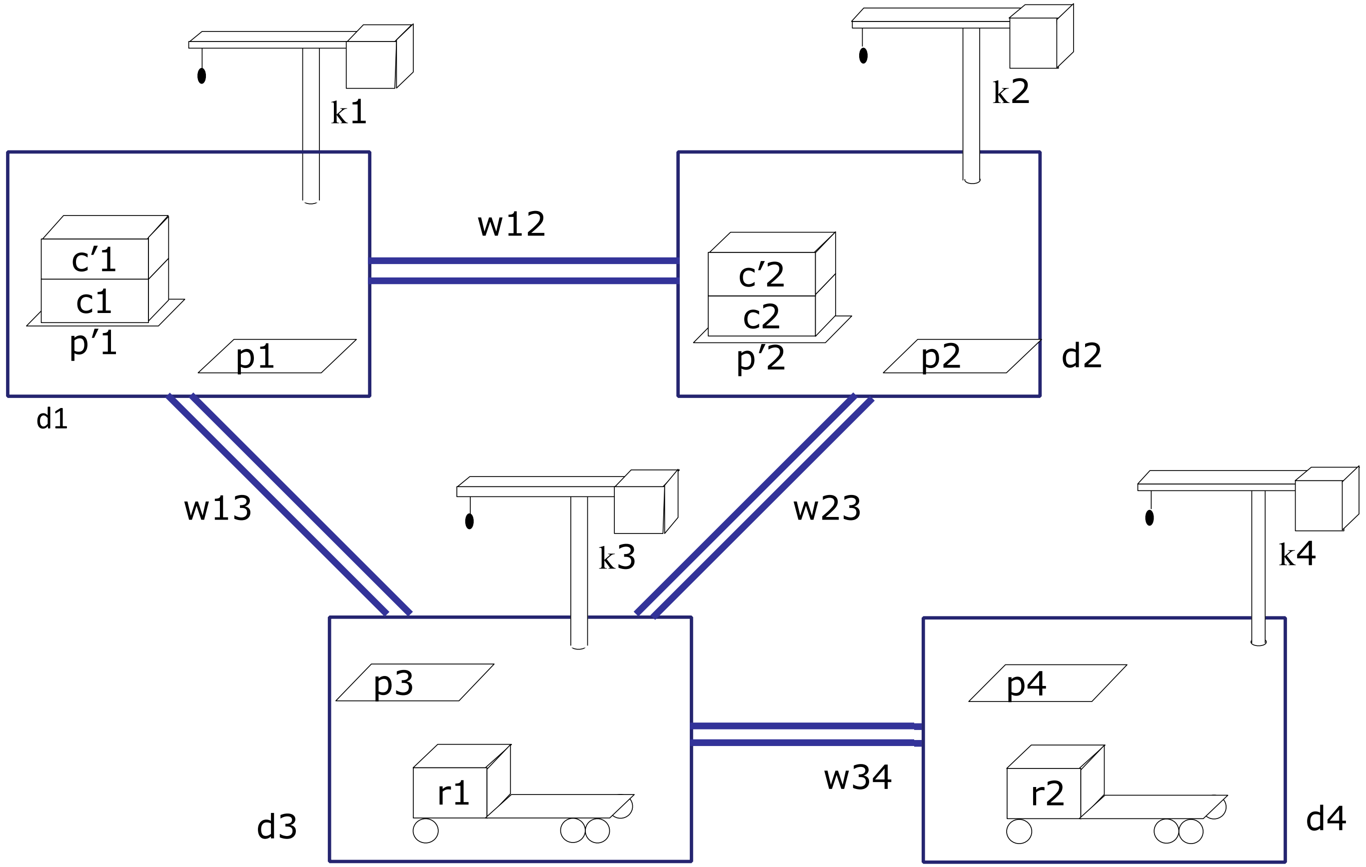


ϕ_1: tasks: $[t_s,t_1]$ move(r,d1)
$[t_s,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r,c1,p′1)
$[t_5,t_6]$ move(r, d3)
$[t_7,t_e]$ unload(k3,r,c1,p3)
bring(r',c2,p4)
supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1
[0] pile(c2)=p′2
. . .
assertions: $[t_s,t_3]$ pile(c1) = p′1
$[t_s,t_3]$ freight(r) = empty
constraints: $t_s < t_1 \le t_3,\ 0 < t_2 \le t_3,\ t_4 \le t_5,\ t_6 \le t_7$
adjacent(d1,w12), . . .

Poll: to get the method instance, I substituted $r \leftarrow r'$ in m-bring. Should I also substitute $r \leftarrow r'$ in ϕ_1?

Apply the method instance

- Changes:
  - Remove bring(r',c2,p4)
  - Add tasks, assertions, constraints
- Flaws: 10 unrefined tasks, 4 unsupported assertions
- Select $[t_s, t_3]$ pile(c1) = p′1 to resolve next



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] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1
[0] pile(c2)=p′2
. . .

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), . . .

Resolvers for a temporal assertion

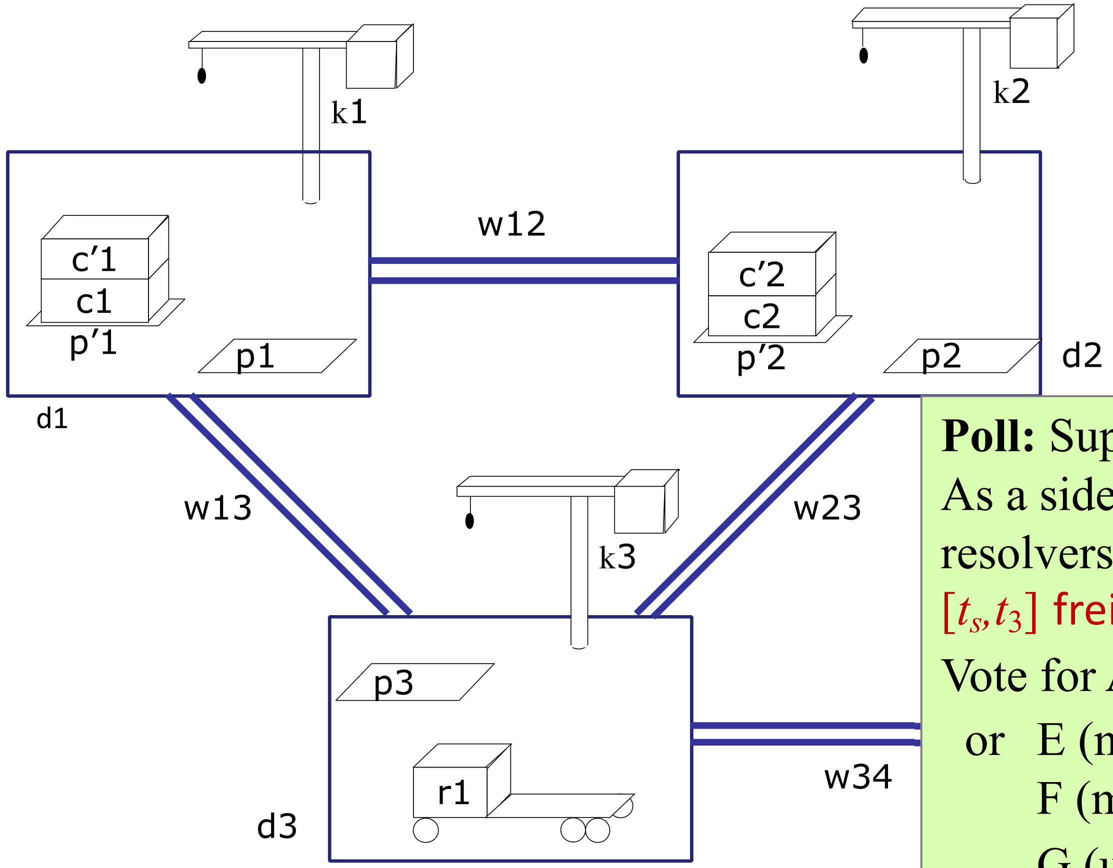
- Four ways to support $[t_s, t_3]$ pile(c1)=p′1

  A. Support from [0] pile(c1)=p′1 by constraining $t_s = 0$

  B. Support from [0] pile(c1)=p′1 by adding persistence $[0, t_s]$ pile(c1)=p1

  C. Add a task $[t_8, t_s]$ unload(k,r'',c1,p′1)

  D. Add a primitive task $[t_8, t_s]$ unstack(k,c1,p′1)



Poll: Suppose r = r1 here. As a side effect, which resolvers will also support $[t_s, t_3]$ freight(r1) = empty?

Vote for A, B, C, or D,
or E (none of them),
F (more than one),
G (unsure)

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)

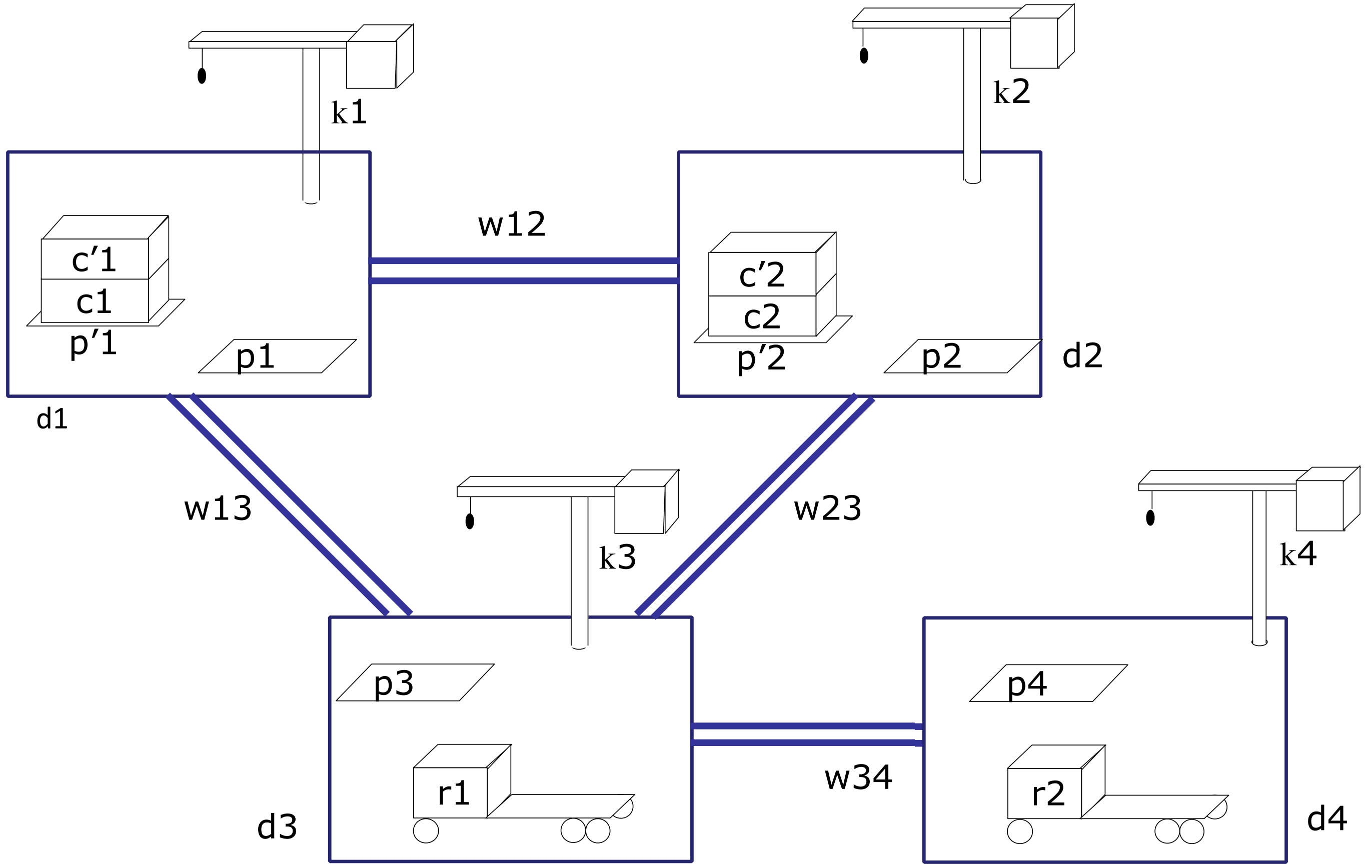
supported:
[0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1
[0] pile(c2)=p′2
. . .

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: $t_s < t_1 \le t_3$, $t_s < t_2 \le t_3$, $t_4 \le t_5$, $t_6 \le t_7$,
$t'_s < t'_1 \le t'_3$, $t'_s < t'_2 \le t'_3$, $t'_4 \le t'_5$, $t'_6 \le t'_7$
adjacent(d1,w12), . . .

Support the assertion

- Support $[t_s,t_3]$ pile(c1) = p′1 from [0] pile(c1)=p′1 by constraining $t_s = 0$
  - Move it to the "supported" list
- Next: support $[0,t_3]$ freight(r) = 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)

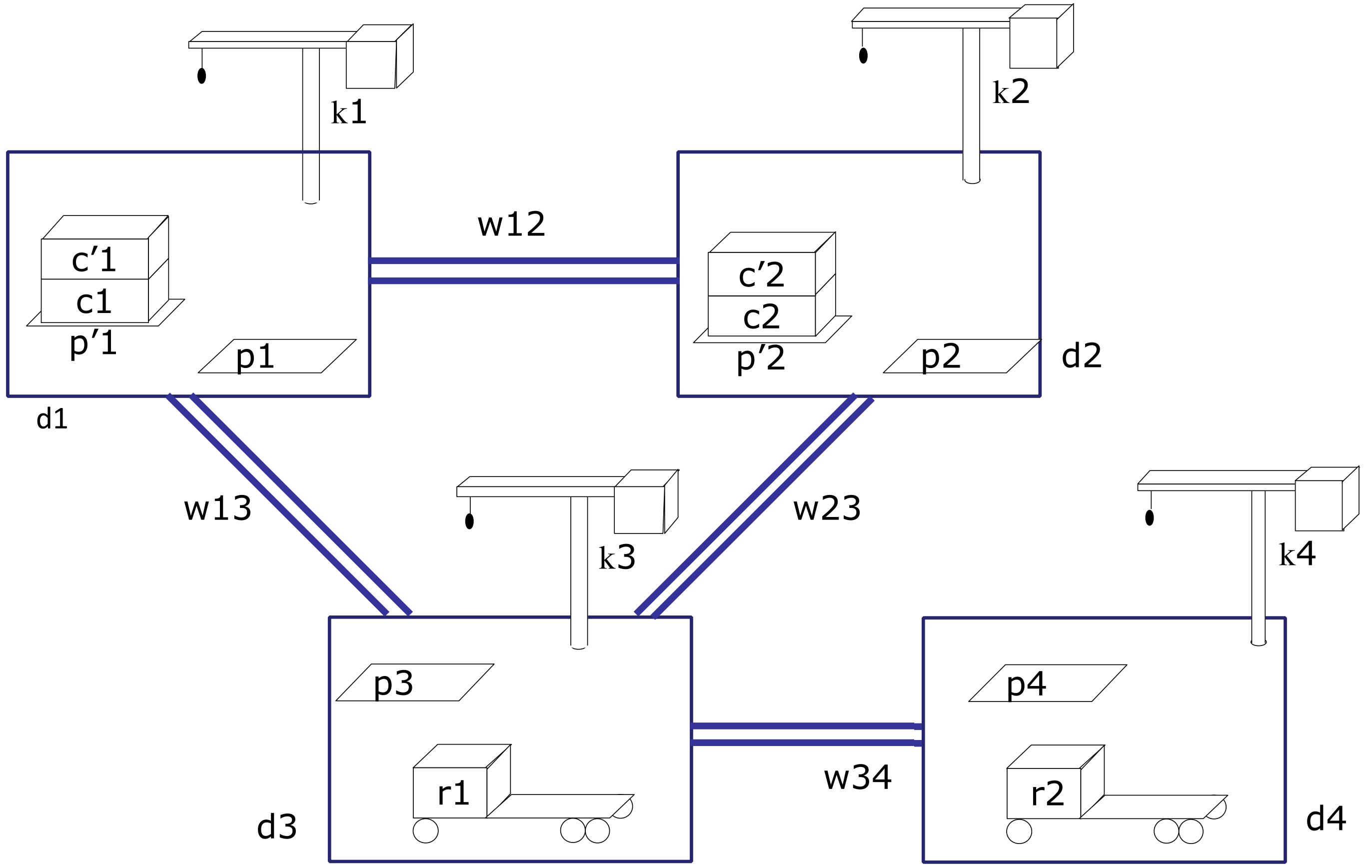
supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1, $[0,t_3]$ pile(c1)=p′1
[0] pile(c2)=p′2
. . .

assertions: $[0,t_3]$ freight(r) = empty
$[t'_s,t'_3]$ pile(c2) = p′2
$[t'_s,t'_1]$ freight(r') = empty

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

Support another assertion

- Support $[0,t_3]$ freight(r) = empty from [0] freight(r2)=empty by constraining r = r2
  - Move it to the "supported" list
- Next: support $[t'_s,t'_3]$ pile(c2)=p′2



ϕ_3: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,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)

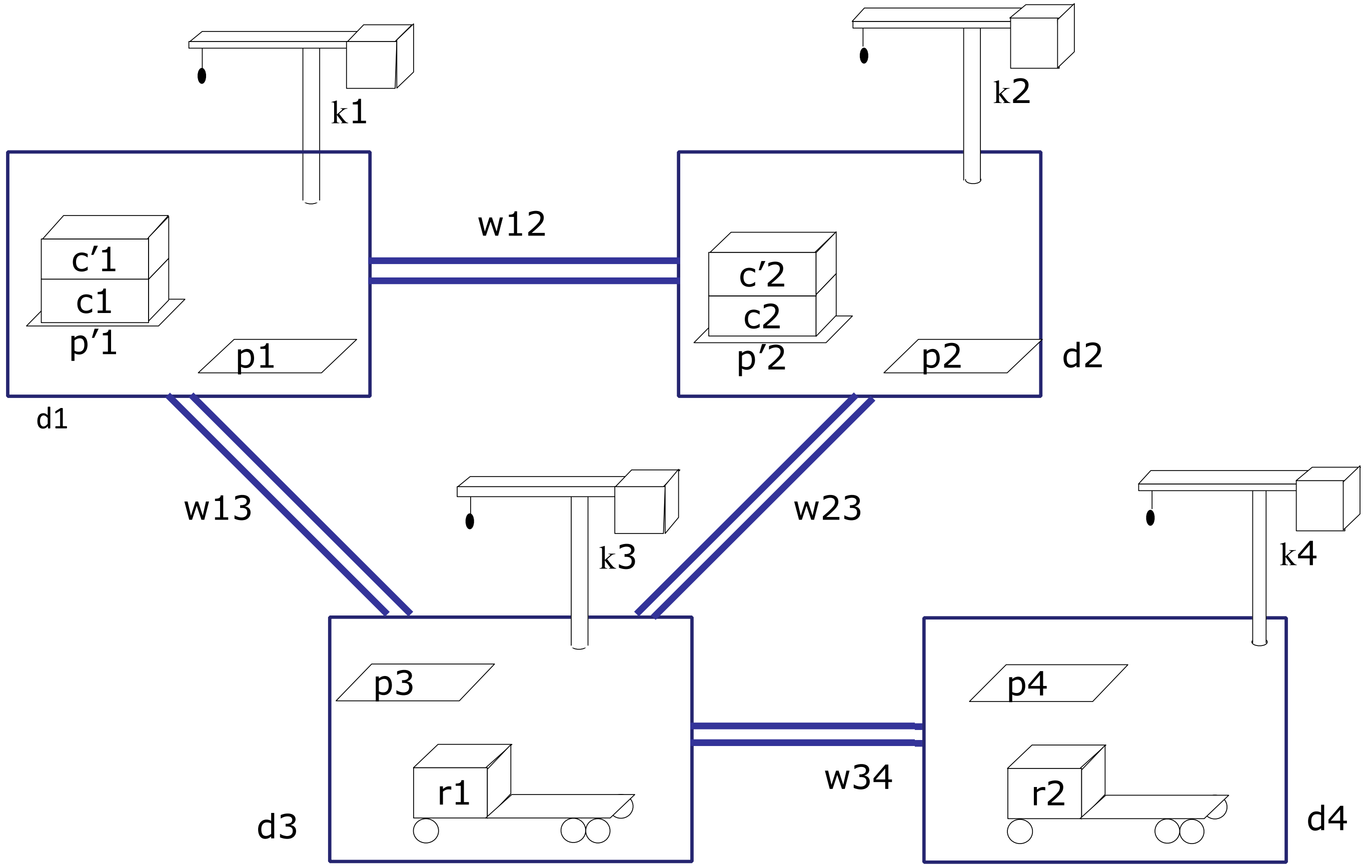
supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1, $[0,t_3]$ pile(c1)=p′1
[0] pile(c2)=p′2
$[0,t_3]$ freight(r2) = empty
. . .

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

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

Support another assertion

- Support $[t'_s, t'_3]$ pile(c2)=p′2 from [0] pile(c2)=p′2 by constraining $t'_s = 0$
  - Move it to the "supported" list
- Next: support $[t'_s, t'_1]$ freight(r') = empty



ϕ_3: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[0,t'_1]$ move(r',d2)
$[0,t'_2]$ 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)

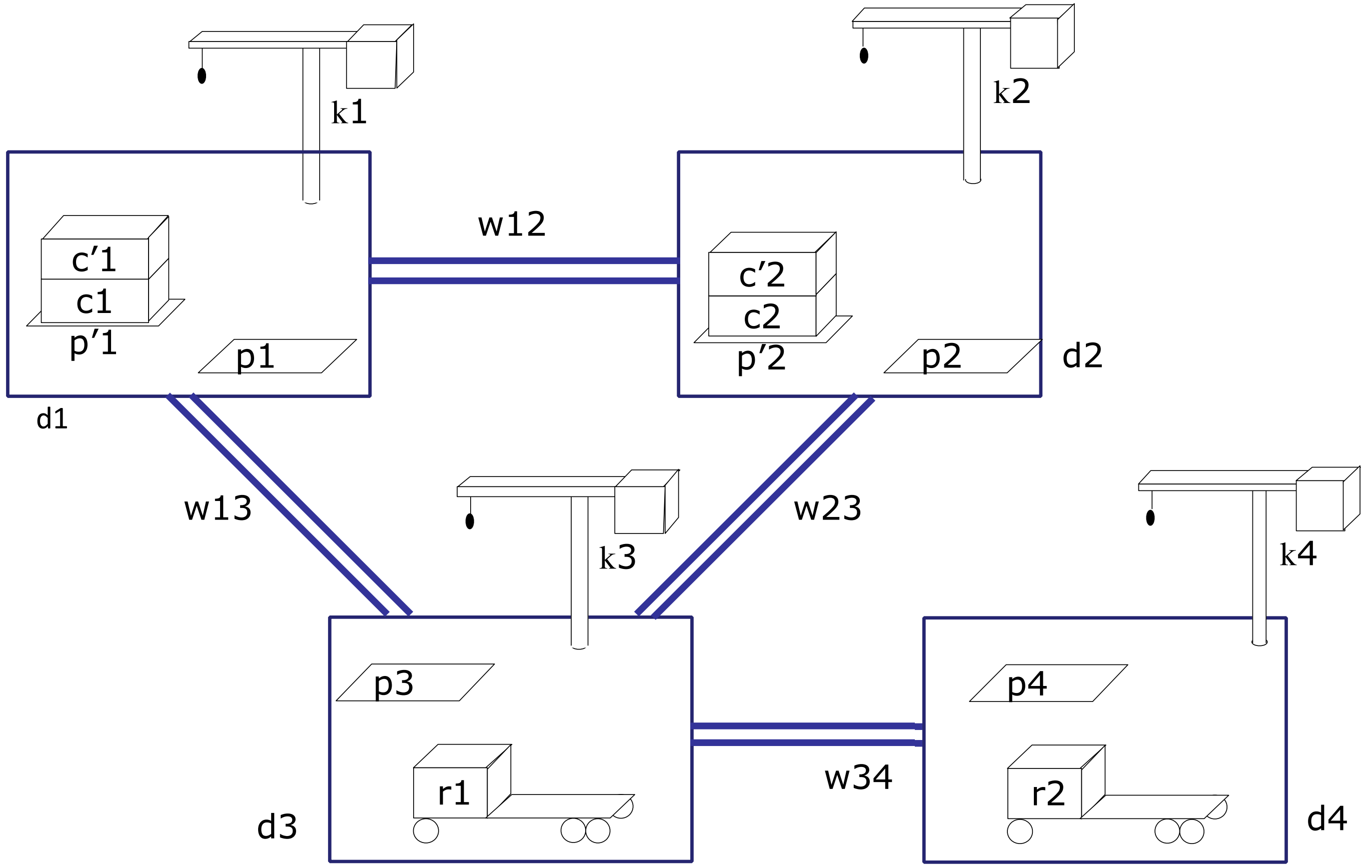
supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
[0] freight(r2)=empty
[0] pile(c1)=p′1, $[0,t_3]$ pile(c1)=p′1
[0] pile(c2)=p′2, $[0,t'_3]$ pile(c2)=p′2
$[0,t_3]$ freight(r2) = empty
. . .

assertions: $[0,t'_1]$ freight(r') = empty

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

Support another assertion

- Support $[t'_s, t'_1]$ freight(r') = empty from [0] freight(r1)=empty by constraining r' = r1
  - Move it to the "supported" list
- Next: refine task $[0, t_1]$ move(r1,d2)



ϕ_4: tasks: $[0, t_1]$ move(r2,d1)
$[0, t_2]$ uncover(c1,p′1)
$[t_3, t_4]$ load(k1,r2,c1,p′1)
$[t_5, t_6]$ move(r2,d3)
$[t_7, t_e]$ unload(k3,r2,c1,p3)
$[0, t'_1]$ move(r1,d2)
$[0, t'_2]$ uncover(c2,p′2)
$[t'_3, t'_4]$ load(k4,r1,c2,p′2)
$[t'_5, t'_6]$ move(r1,d4)
$[t'_7, t'_e]$ unload(k2,r1,c2,p′2)

supported: [0] loc(r1)=d3
[0] loc(r2)=d4
[0] freight(r1)=empty
$[0, t_3]$ freight(r2) = empty
[0] pile(c1)=p′1, $[0, t_3]$ pile(c1)=p′1
[0] pile(c2)=p′2, $[0, t'_3]$ pile(c2)=p′2
[0] freight(r2)=empty
$[0, t'_1]$ freight(r1) = empty
. . .

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

Method instance

- Instantiate $r \leftarrow$r1, $d \leftarrow$d2, $t_s \leftarrow$0, $t_e \leftarrow t_1$ to match $[0,t_1']$ move(r1,d3)
  - Rename timepoint variables $t_1, \ldots, t_4$ to avoid name conflicts

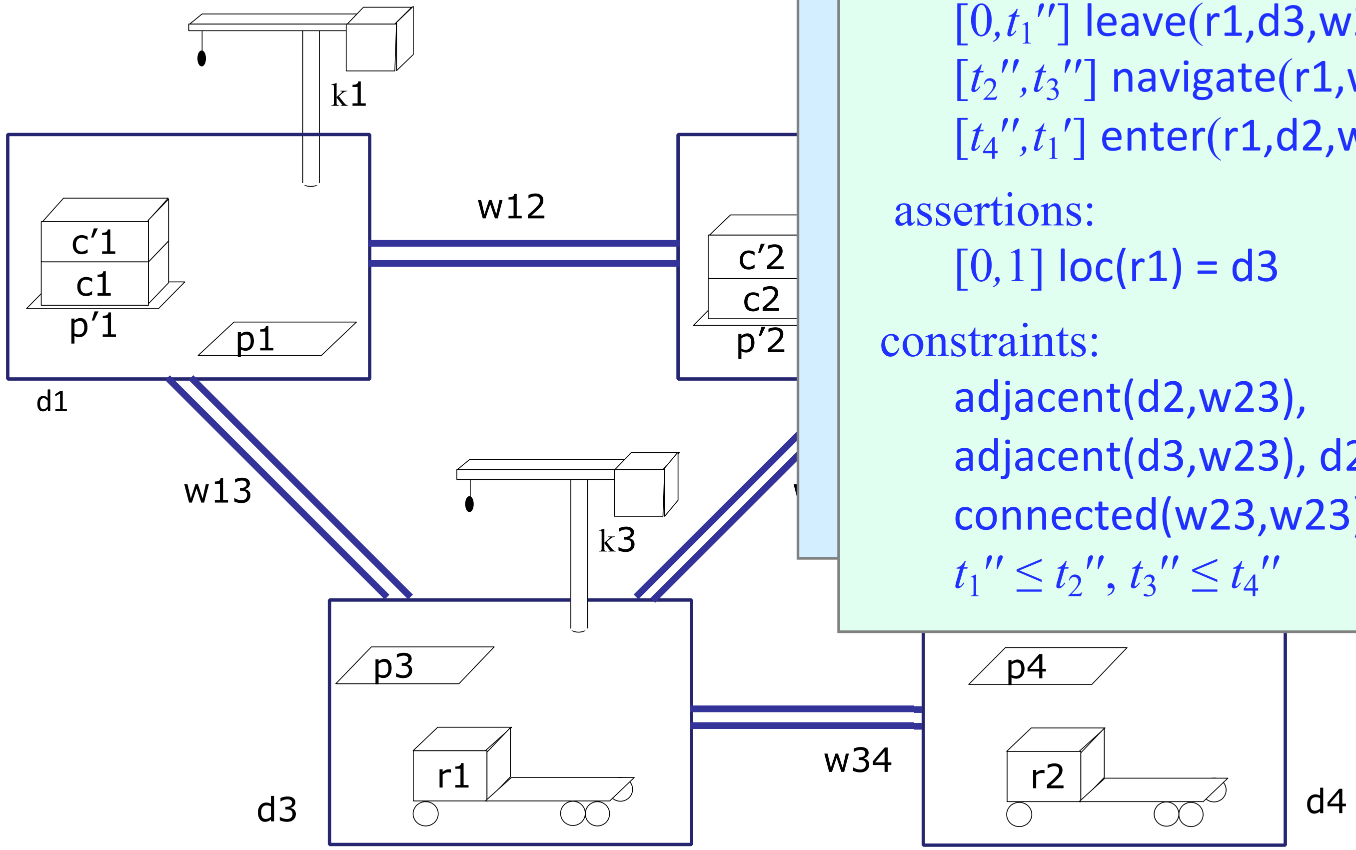
m-move1(r,d,d',w,w')

m-move1(r1,d2,d3,w23,w23)
task: move(r2,d1)

refinement:
- $[0,t_1'']$ leave(r1,d3,w23)
- $[t_2'',t_3'']$ navigate(r1,w23,w23)
- $[t_4'',t_1']$ enter(r1,d2,w23)

assertions:
- $[0,1]$ loc(r1) = d3

constraints:
- adjacent(d2,w23),
- adjacent(d3,w23), d2 ≠ d3,
- connected(w23,w23),
- $t_1'' \le t_2''$, $t_3'' \le t_4''$



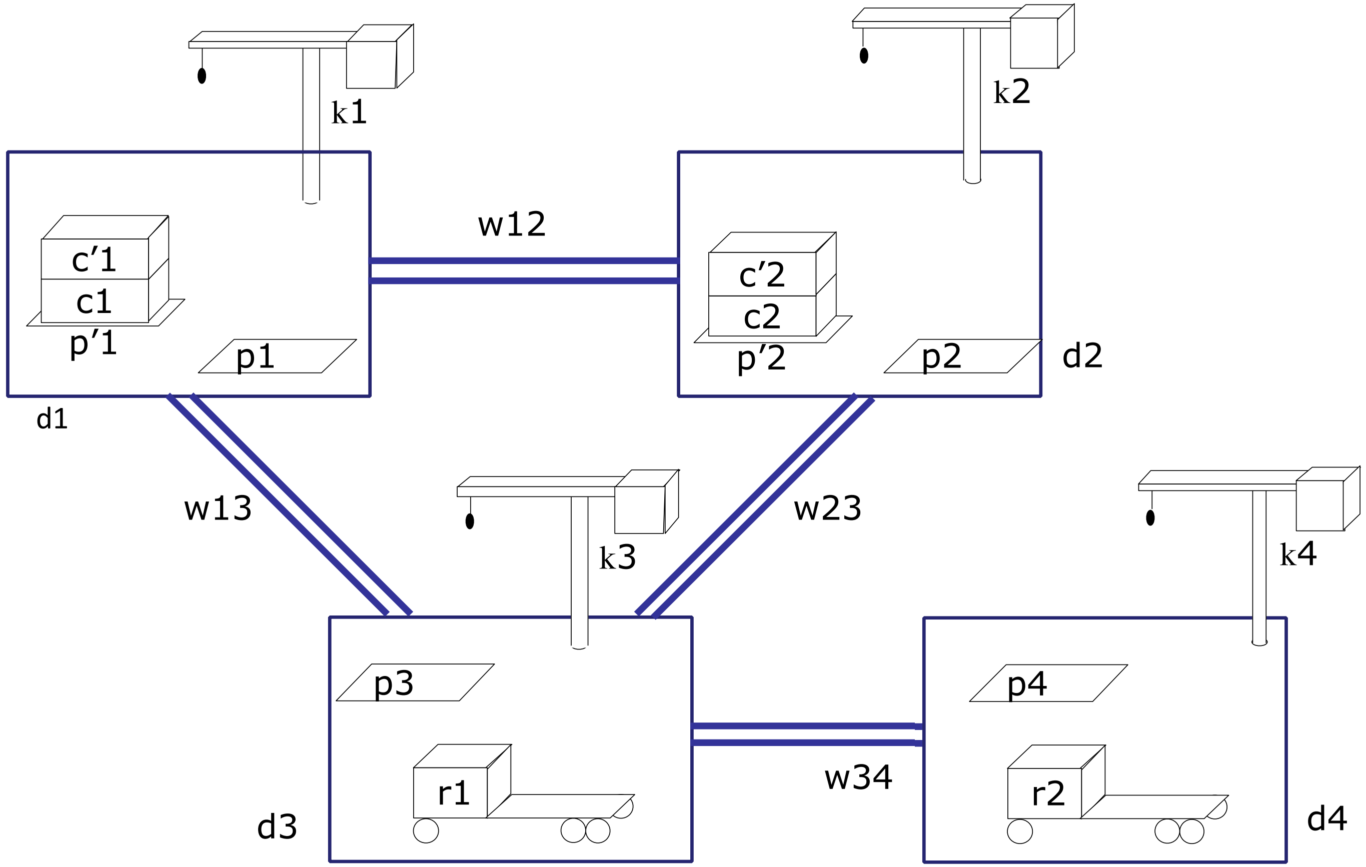
ϕ_4: tasks:
- $[0,t_1]$ move(r2,d1)
- $[0,t_2]$ uncover(c1,p'1)
- $[t_3,t_4]$ load(k1,r2,c1,p'1)
- $[t_5,t_6]$ move(r2,d3)
- $[t_7,t_e]$ unload(k3,r2,c1,p3)
- $[0,t'_1]$ move(r1,d2)
- $[0,t'_2]$ uncover(c2,p'2)
- $[t'_3,t'_4]$ load(k4,r1,c2,p'2)
- $[t'_5,t'_6]$ move(r1,d4)
- $[t'_7,t'_e]$ unload(k2,r1,c2,p'2)

supported:
- $[0]$ loc(r1)=d3
- $[0]$ loc(r2)=d4
- . . .

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

Modified chronicle

- Replace $[0,t'_1]$ move(r1,d2) with 3 tasks
- Assertion $[0,1]$ loc(r1) = d3 is already supported, add it to the “supported” list
- Add some constraints
- Flaws: 12 tasks
- Next: support $[0,t_1'']$ leave(r1,d3,w23)



ϕ_5: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[0,t_1'']$ leave(r1,d3,w23)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[t_4'',t_1']$ enter(r1,d2,w23)
$[0,t'_2]$ uncover(c2,p′2)
$[t'_3,t'_4]$ load(k4,r1,c2,p′2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p′2)

supported: $[0]$ loc(r1)=d3, $[0,1]$ loc(r1) = d3
$[0]$ loc(r2)=d4
. . .

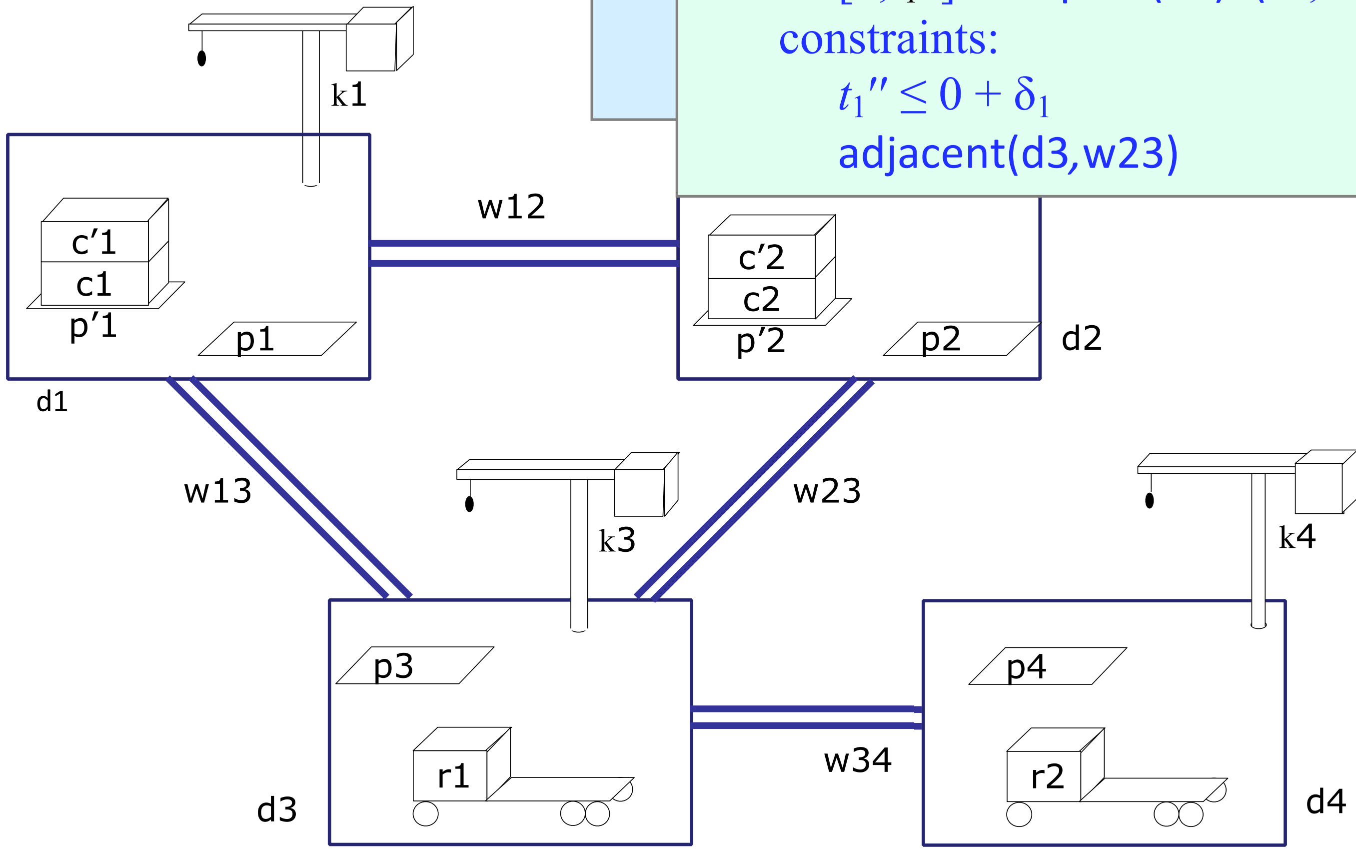
constraints: $0<t_1\le t_3$, $0<t_2\le t_3$, $t_4\le t_5$, $t_6\le t_7$,
$0<t'_1\le t'_3$, $0<t'_2\le t'_3$, $t'_4\le t'_5$, $t'_6\le t'_7$
$t_1''\le t_2''$, $t_3''\le t_4''$
adjacent(d1,w12), . . .

Primitive task instance

- Instantiate $r \leftarrow$r2, $d \leftarrow$d3, $t_s \leftarrow 0$, $t_e \leftarrow t_1''$ to match $[0, t_1'']$ leave(r1,d3,w23)

leave(r,d,w)

leave(r1,d3,w23)
  assertions:
    $[0, t_1'']$ loc(r1): (d3,w23)
    $[0, t_1'']$ occupant(d3): (r1,empty)
  constraints:
    $t_1'' \leq 0 + \delta_1$
    adjacent(d3,w23)



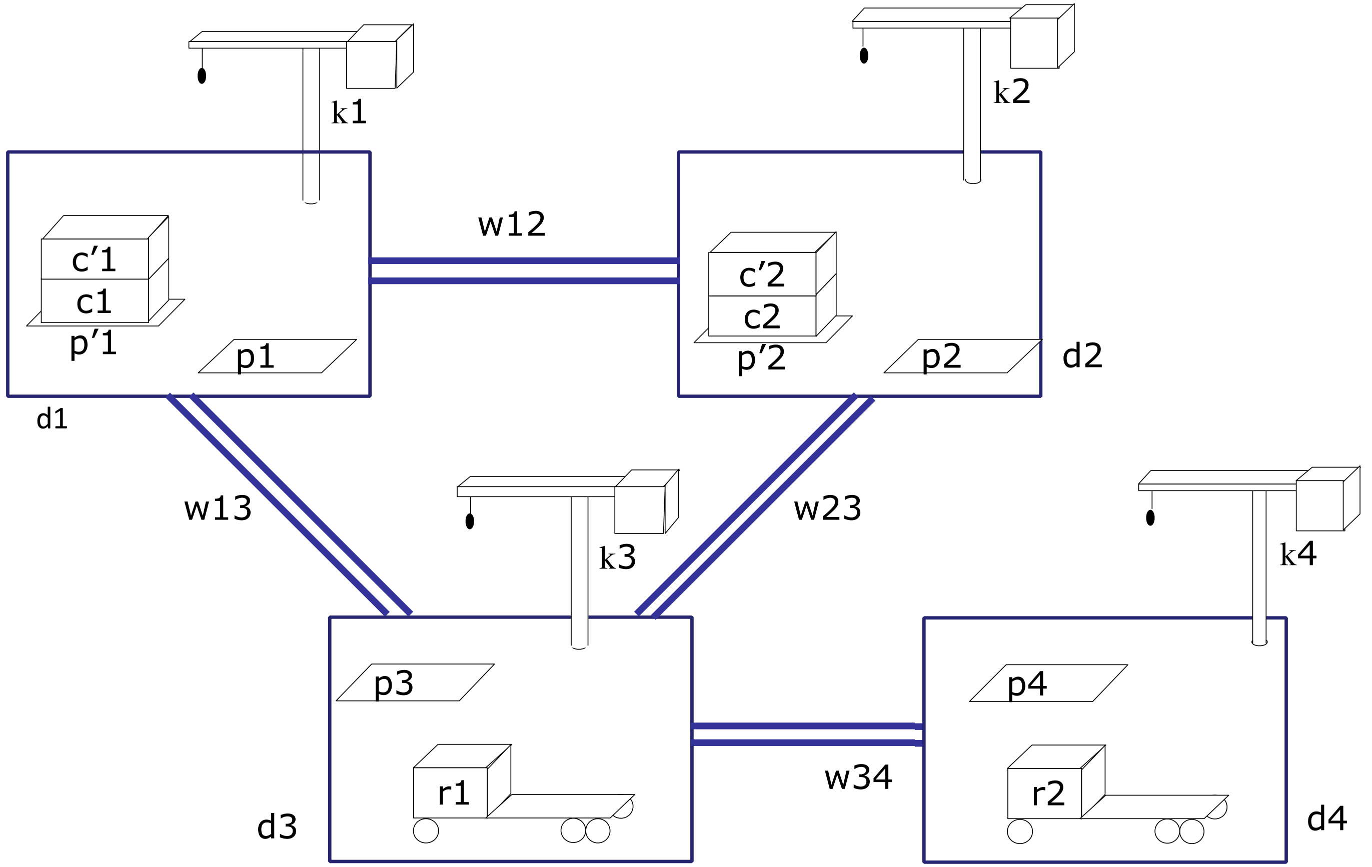
ϕ_6: tasks: $[0, t_1]$ move(r2,d1)
$[0, t_2]$ uncover(c1,p′1)
$[t_3, t_4]$ load(k1,r2,c1,p′1)
$[t_5, t_6]$ move(r2,d3)
$[t_7, t_e]$ unload(k3,r2,c1,p3)
$[0, t_1'']$ leave(r1,d3,w23)
$[t_2'', t_3'']$ navigate(r1,w23,w23)
$[t_4'', t_1']$ enter(r1,d2,w23)
$[0, t'_2]$ uncover(c2,p′2)
$[t'_3, t'_4]$ load(k4,r1,c2,p′2)
$[t'_5, t'_6]$ move(r1,d4)
$[t'_7, t'_e]$ unload(k2,r1,c2,p′2)

supported: $[0]$ loc(r1)=d3, $[0,1]$ loc(r1) = d3
$[0]$ loc(r2)=d4
. . .

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

Modified chronicle

- Removed $[0,t_1'']$ leave(r1,d2,w23), added 2 assertions, one constraint
- Next: support $[0,t_1'']$ enter(r1,d2,w23)



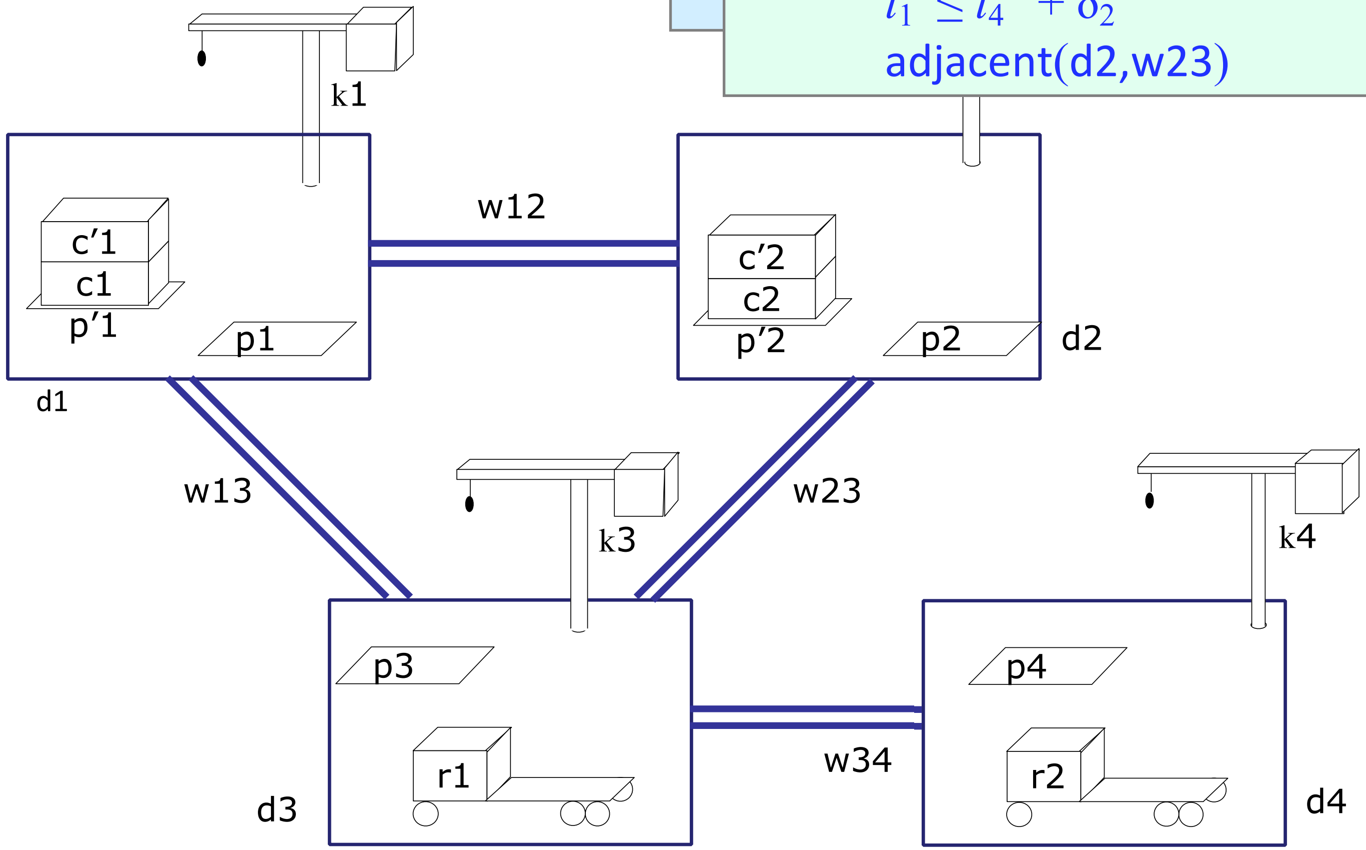
ϕ_6: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[t_4'',t_1']$ enter(r1,d2,w23)
$[0,t'_2]$ uncover(c2,p′2)
$[t'_3,t'_4]$ load(k4,r1,c2,p′2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p′2)
supported: [0] loc(r1)=d3, [0,1] loc(r1) = d3
[0] loc(r2)=d4
. . .
assertions: $[0,t_1'']$ loc(r1): (d3,w23)
$[0,t_1'']$ occupant(d3): (r1,empty)
constraints: $0<t_1\leq t_3$, $0<t_2\leq t_3$, $t_4\leq t_5$, $t_6\leq t_7$,
$0<t'_1\leq t'_3$, $0<t'_2\leq t'_3$, $t'_4\leq t'_5$, $t'_6\leq t'_7$
$t_1''\leq t_2''$, $t_3''\leq t_4''$
$t_1''\leq \delta_1$
adjacent(d1,w12), . . .

Primitive task instance

- Instantiate $r \leftarrow$ r1, $t_s \leftarrow t_4''$, $t_e \leftarrow t_1'$

enter(r,d,w)

enter(r1,d2,w23)
  assertions:
    $[t_4'',t_1']$ loc(r1): (w23,d2)
    $[t_4'',t_1']$ occupant(d2): (empty, r1)
  constraints:
    $t_1' \leq t_4'' + \delta_2$
    adjacent(d2,w23)



ϕ_6: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[t_4'',t_1']$ enter(r1,d2,w23)
$[0,t'_2]$ uncover(c2,p′2)
$[t'_3,t'_4]$ load(k4,r1,c2,p′2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p′2)

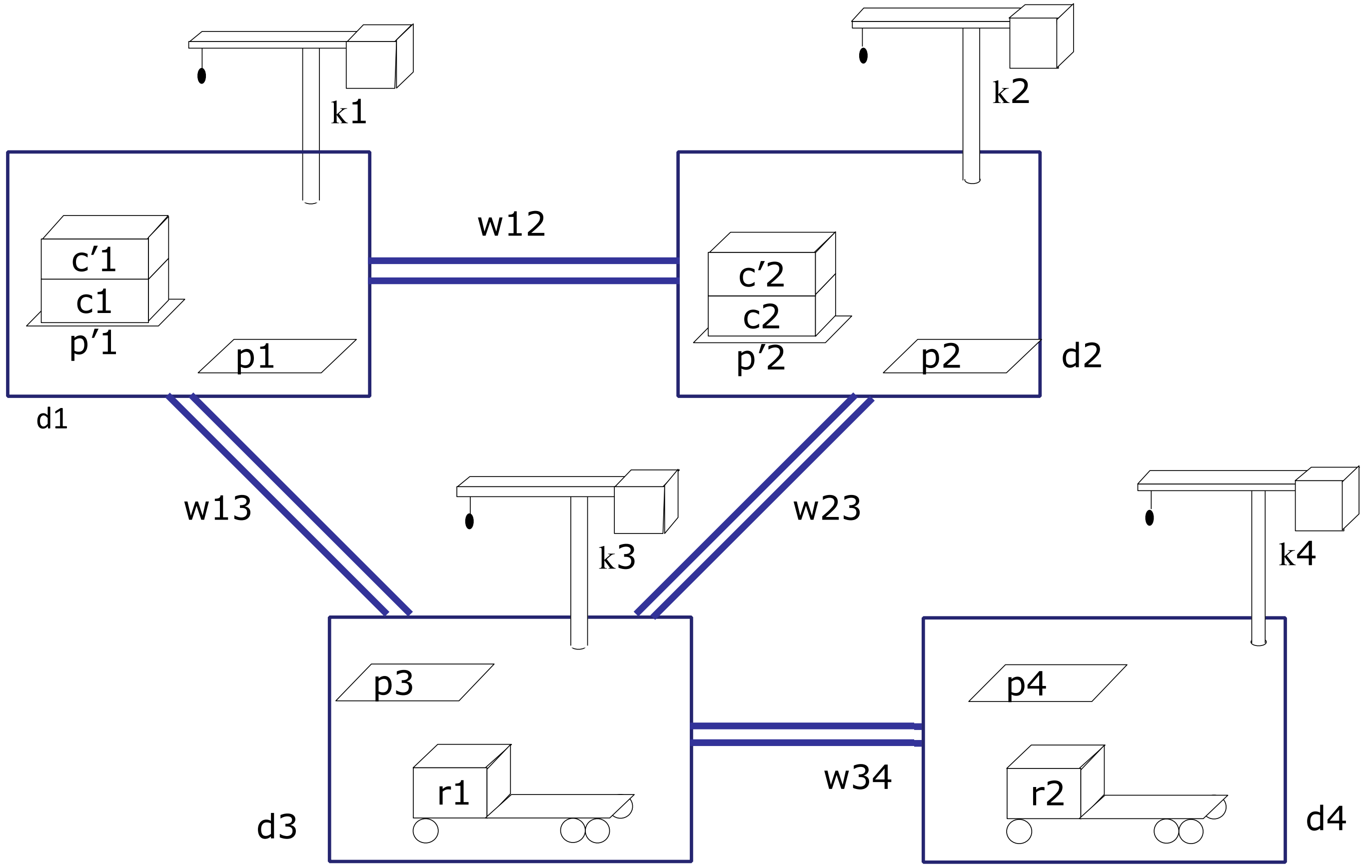
supported: $[0]$ loc(r1)=d3, $[0,1]$ loc(r1) = d3
$[0]$ loc(r2)=d4
. . .

assertions: $[0,t_1'']$ loc(r1): (d3,w23)
$[0,t_1'']$ occupant(d3): (r1,empty)

constraints: $0<t_1\leq t_3$, $0<t_2\leq t_3$, $t_4\leq t_5$, $t_6\leq t_7$,
$0<t'_1\leq t'_3$, $0<t'_2\leq t'_3$, $t'_4\leq t'_5$, $t'_6\leq t'_7$
$t_1''\leq t_2''$, $t_3''\leq t_4''$
$t_1'' \leq \delta_1$
adjacent(d1,w12), . . .

Modified chronicle

- Remove $[t_4'', t_1']$ enter(r1,d2,w23), add 2 assertions, one constraint
- Next: support $[t_2'', t_3'']$ navigate(r1,w23,w23)



ϕ_6: tasks: $[0, t_1]$ move(r2,d1)
$[0, t_2]$ uncover(c1,p′1)
$[t_3, t_4]$ load(k1,r2,c1,p′1)
$[t_5, t_6]$ move(r2,d3)
$[t_7, t_e]$ unload(k3,r2,c1,p3)
$[t_2'', t_3'']$ navigate(r1,w23,w23)
$[0, t'_2]$ uncover(c2,p′2)
$[t'_3, t'_4]$ load(k4,r1,c2,p′2)
$[t'_5, t'_6]$ move(r1,d4)
$[t'_7, t'_e]$ unload(k2,r1,c2,p′2)

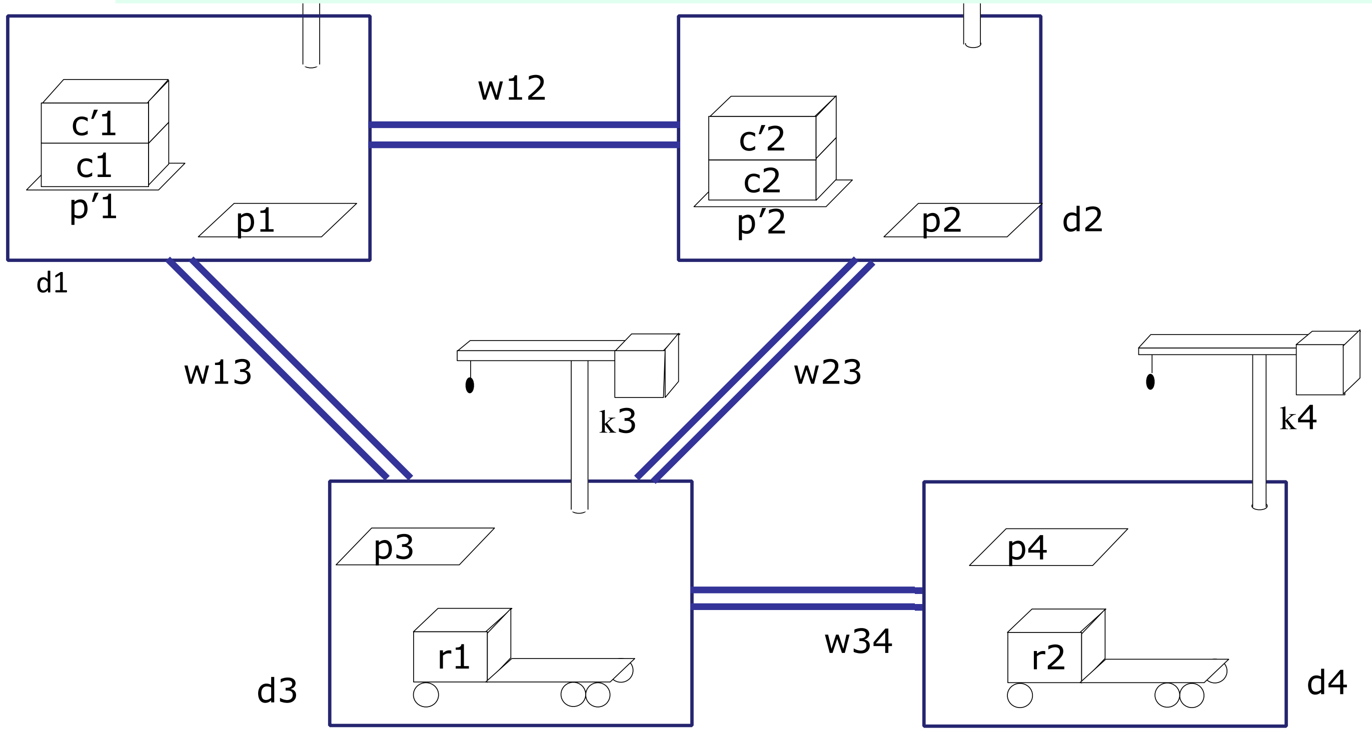
supported: [0] loc(r1)=d3, [0,1] loc(r1) = d3
[0] loc(r2)=d4
. . .

assertions: $[0, t_1'']$ loc(r1): (d3,w23)
$[0, t_1'']$ occupant(d3): (r1,empty)
$[t_4'', t_1']$ loc(r1): (w23,d2)
$[t_4'', t_1']$ occupant(d2): (empty,r1)

constraints: $0 < t_1 \le t_3$, $0 < t_2 \le t_3$, $t_4 \le t_5$, $t_6 \le t_7$,
$0 < t'_1 \le t'_3$, $0 < t'_2 \le t'_3$, $t'_4 \le t'_5$, $t'_6 \le t'_7$
$t_1'' \le t_2''$, $t_3'' \le t_4''$
$t_1'' \le \delta_1$, $t_1' \le t_4'' + \delta_2$
adjacent(d1,w12), . . .

Problems with navigation

- Book's description of navigate(r,w,w'):
  - r navigates from waypoint w to connected waypoint w'
- When are two waypoints connected?
  - (1) If there's a road network between them?
  - (2) If they're on the same road and adjacent?
  - (3) If they're on the same road (but not necessarily adjacent)?
- Is a waypoint connected to itself?



ϕ_6: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p'1)
$[t_3,t_4]$ load(k1,r2,c1,p'1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[0,t'_2]$ uncover(c2,p'2)
$[t'_3,t'_4]$ load(k4,r1,c2,p'2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p'2)

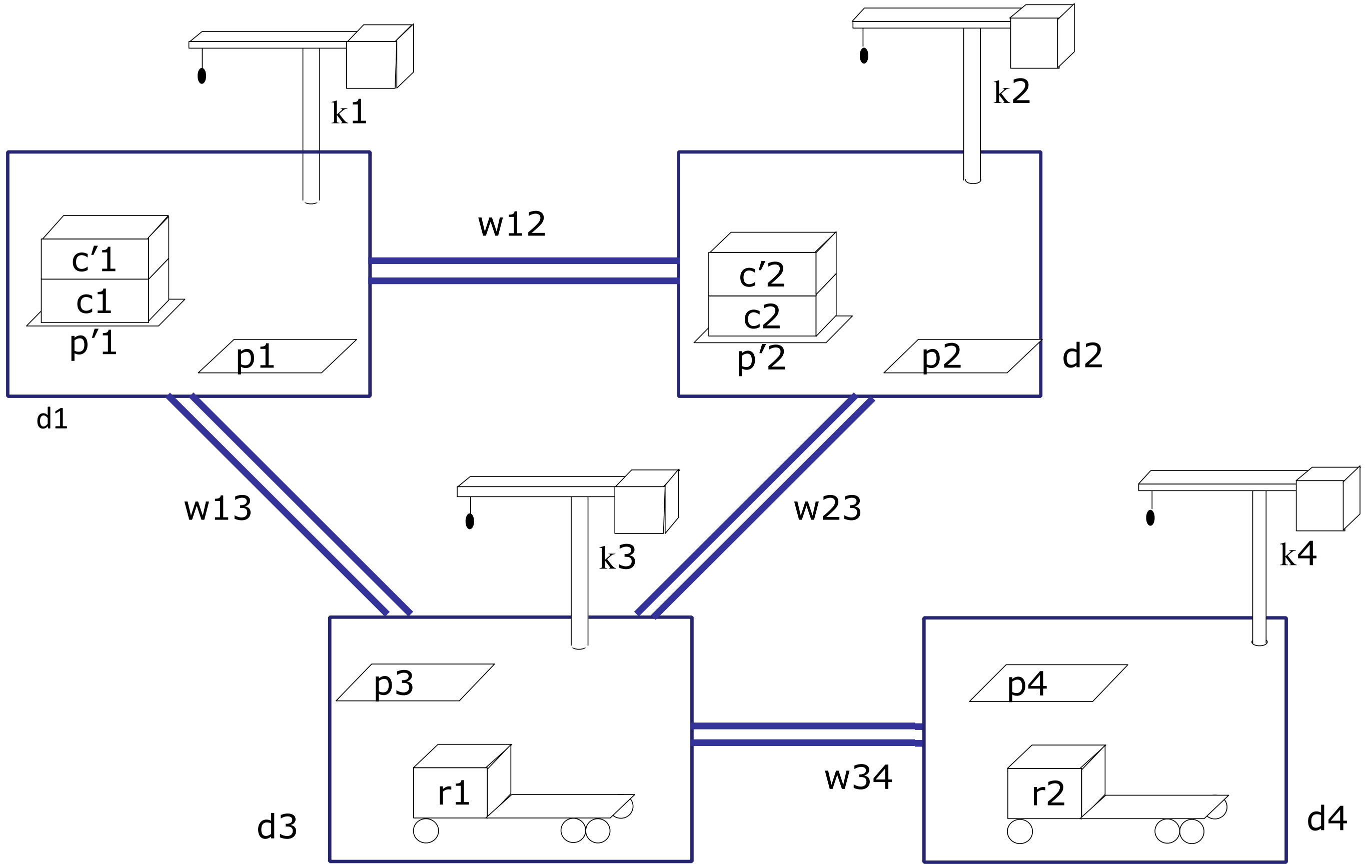
supported: $[0]$ loc(r1)=d3, $[0,1]$ loc(r1) = d3
$[0]$ loc(r2)=d4
. . .

assertions: $[0,t_1'']$ loc(r1): (d3,w23)
$[0,t_1'']$ occupant(d3): (r1,empty)
$[t_4'',t_1']$ loc(r1): (w23,d2)
$[t_4'',t_1']$ occupant(d2): (empty,r1)

constraints: $0<t_1 \le t_3$, $0<t_2 \le t_3$, $t_4 \le t_5$, $t_6 \le t_7$,
$0<t'_1 \le t'_3$, $0<t'_2 \le t'_3$, $t'_4 \le t'_5$, $t'_6 \le t'_7$
$t_1'' \le t_2''$, $t_3'' \le t_4''$
$t_1'' \le \delta_1$, $t_1' \le t_4''+\delta_2$
adjacent(d1,w12), . . .

Possibility (1)

- Suppose *w* is connected to *w*′ if a network of roads connects them
  - Can write navigate as a task, but not a primitive
  - Assertions and constraints depend on the route from *w* to *w*′
- Write a method whose body is a path-planning algorithm



ϕ_6: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[0,t'_2]$ uncover(c2,p′2)
$[t'_3,t'_4]$ load(k4,r1,c2,p′2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p′2)

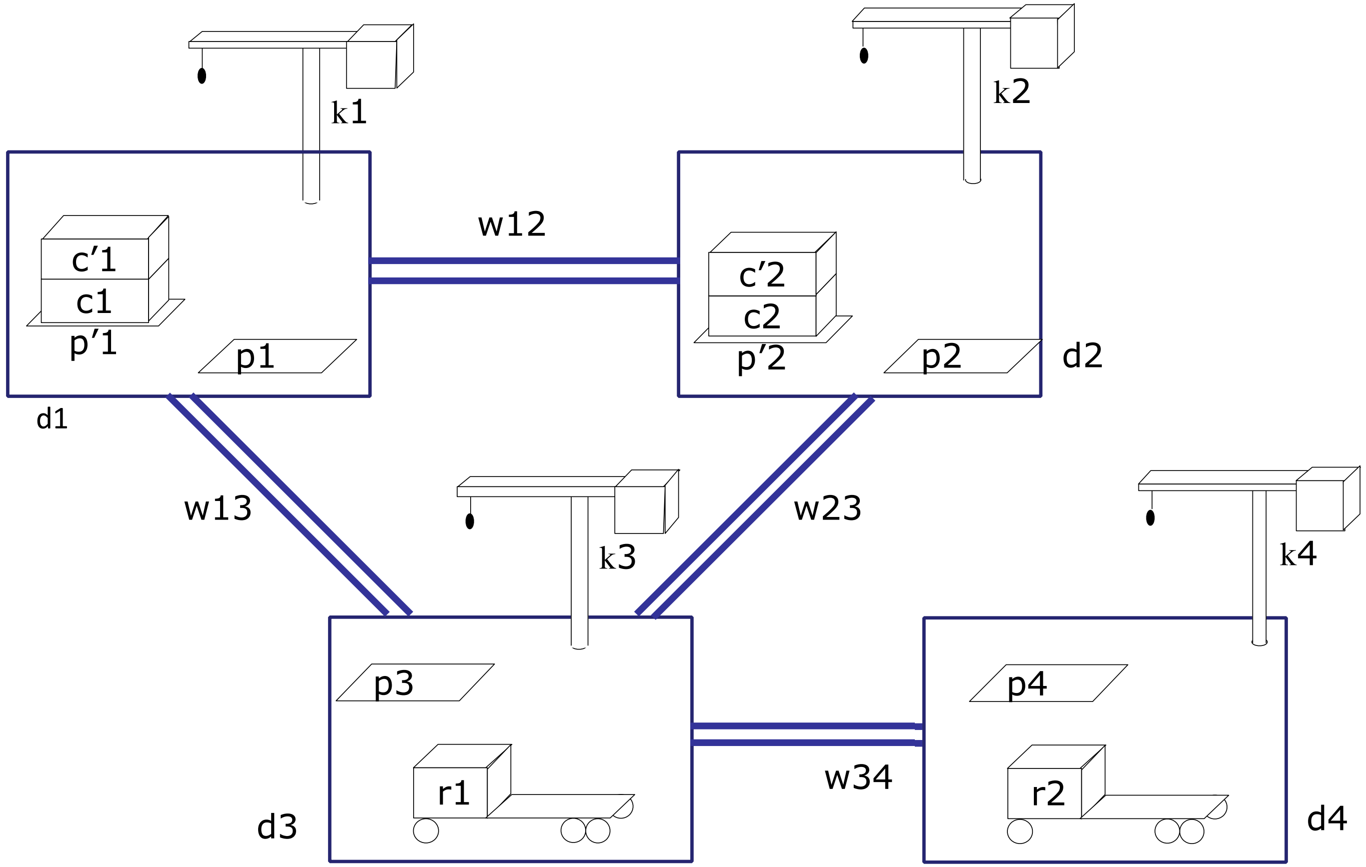
supported: $[0]$ loc(r1)=d3, $[0,1]$ loc(r1) = d3
$[0]$ loc(r2)=d4
. . .

assertions: $[0,t_1'']$ loc(r1): (d3,w23)
$[0,t_1'']$ occupant(d3): (r1,empty)
$[t_4'',t_1']$ loc(r1): (w23,d2)
$[t_4'',t_1']$ occupant(d2): (empty,r1)

constraints: $0<t_1\leq t_3$, $0<t_2\leq t_3$, $t_4\leq t_5$, $t_6\leq t_7$,
$0<t'_1\leq t'_3$, $0<t'_2\leq t'_3$, $t'_4\leq t'_5$, $t'_6\leq t'_7$
$t_1''\leq t_2''$, $t_3''\leq t_4''$
$t_1''\leq \delta_1$, $t_1'\leq t_4''+\delta_2$
adjacent(d1,w12), . . .

Possibility (2)

- Suppose w is connected to w' if they're adjacent and on the same road
  - Can write navigate as a task, but not a primitive
  - Assertions and constraints depend on how many waypoints from w to w'
- Write a method that goes to an adjacent waypoint then calls navigate again



ϕ_6: tasks: $[0,t_1]$ move(r2,d1)
$[0,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,r2,c1,p′1)
$[t_5,t_6]$ move(r2,d3)
$[t_7,t_e]$ unload(k3,r2,c1,p3)
$[t_2'',t_3'']$ navigate(r1,w23,w23)
$[0,t'_2]$ uncover(c2,p′2)
$[t'_3,t'_4]$ load(k4,r1,c2,p′2)
$[t'_5,t'_6]$ move(r1,d4)
$[t'_7,t'_e]$ unload(k2,r1,c2,p′2)

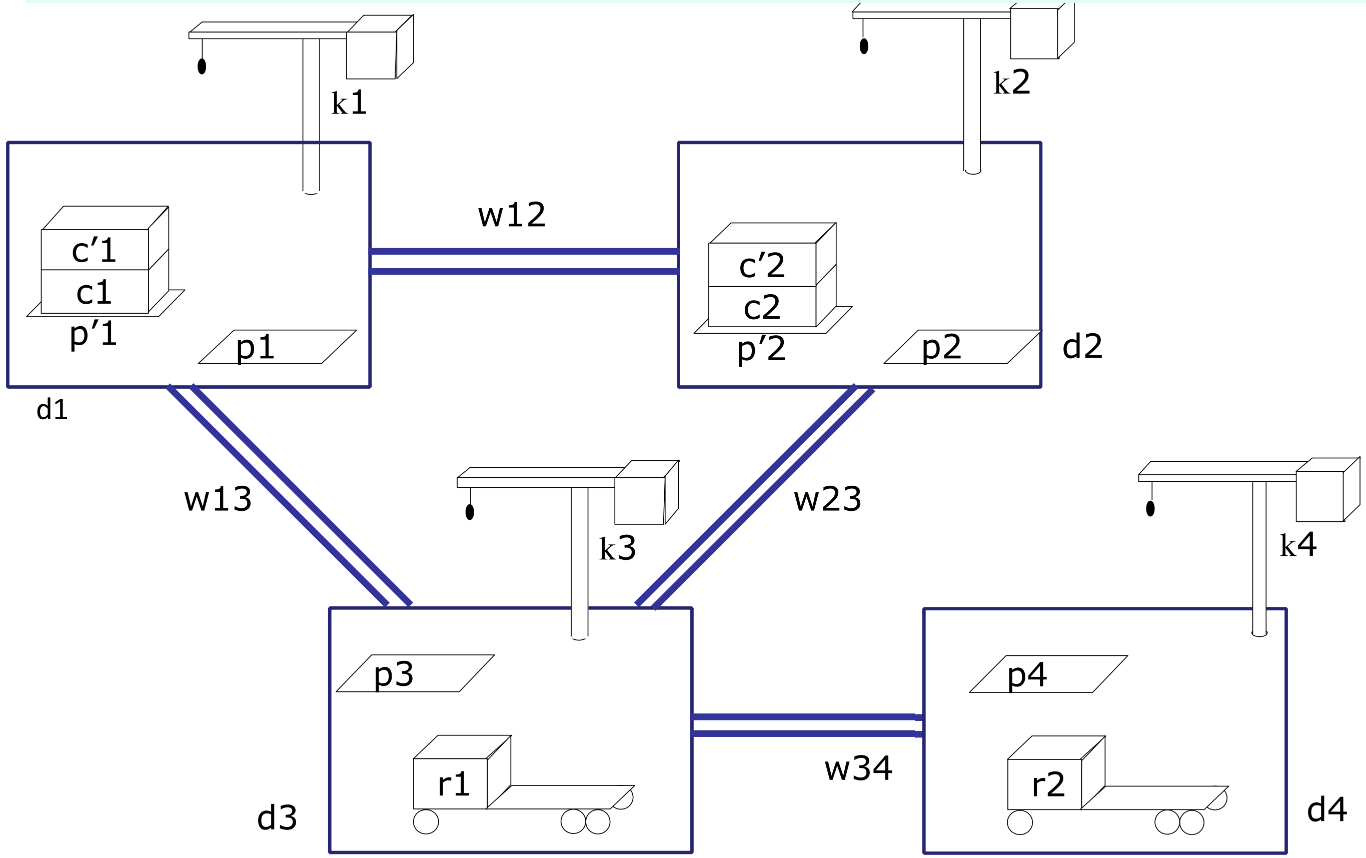
supported: [0] loc(r1)=d3, [0,1] loc(r1) = d3
[0] loc(r2)=d4
. . .

assertions: $[0,t_1'']$ loc(r1): (d3,w23)
$[0,t_1'']$ occupant(d3): (r1,empty)
$[t_4'',t_1']$ loc(r1): (w23,d2)
$[t_4'',t_1']$ occupant(d2): (empty,r1)

constraints: $0<t_1\leq t_3$, $0<t_2\leq t_3$, $t_4\leq t_5$, $t_6\leq t_7$,
$0<t'_1\leq t'_3$, $0<t'_2\leq t'_3$, $t'_4\leq t'_5$, $t'_6\leq t'_7$
$t_1''\leq t_2''$, $t_3''\leq t_4''$
$t_1''\leq \delta_1$, $t_1'\leq t_4''+\delta_2$
adjacent(d1,w12), . . .

Possibility (3)

- Suppose w is connected to w' if they're on the same road, but not necessarily adjacent
  - Then navigate is easy: loc(r) : (w,w')
- Works well if we redraw the figure to have two waypoints on each road
- With only one waypoint, a degenerate case: [t_2'',t_3''] loc(r1) : (w23,w23)



ϕ_6: tasks: [$0,t_1$] move(r2,d1)
[$0,t_2$] uncover(c1,p′1)
[t_3,t_4] load(k1,r2,c1,p′1)
[t_5,t_6] move(r2,d3)
[t_7,t_e] unload(k3,r2,c1,p3)
[t_2'',t_3''] navigate(r1,w23,w23)
[$0,t'_2$] uncover(c2,p′2)
[t'_3,t'_4] load(k4,r1,c2,p′2)
[t'_5,t'_6] move(r1,d4)
[t'_7,t'_e] unload(k2,r1,c2,p′2)

supported: [0] loc(r1)=d3, [0,1] loc(r1) = d3
[0] loc(r2)=d4
. . .

assertions: [$0,t_1''$] loc(r1): (d3,w23)
[$0,t_1''$] occupant(d3): (r1,empty)
[t_4'',t_1'] loc(r1): (w23,d2)
[t_4'',t_1'] occupant(d2): (empty,r1)

constraints: $0<t_1\le t_3$, $0<t_2\le t_3$, $t_4\le t_5$, $t_6\le t_7$,
$0<t'_1\le t'_3$, $0<t'_2\le t'_3$, $t'_4\le t'_5$, $t'_6\le t'_7$
$t_1''\le t_2''$, $t_3''\le t_4''$
$t_1''\le \delta_1$, $t_1'\le t_4''+\delta_2$
adjacent(d1,w12), . . .

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

TemPlan(ϕ, Σ)
    Flaws $\leftarrow$ set of flaws of ϕ
    if $\mathit{Flaws} = \varnothing$ then return ϕ
    arbitrarily select $f \in \mathit{Flaws}$
    Resolvers $\leftarrow$ set of resolvers of f
    if $\mathit{Resolvers} = \varnothing$ then return failure
    nondeterministically choose $\rho \in \mathit{Resolvers}$
    $\phi \leftarrow$ Transform(ϕ, ρ)
    Templan(ϕ, Σ)

PSP(Σ, π)
    loop
        if $\mathit{Flaws}(\pi) = \varnothing$ then return π
        arbitrarily select $f \in \mathit{Flaws}(\pi)$
        $R \leftarrow$ {all feasible resolvers for f}
        if $R = \varnothing$ then return failure
        nondeterministically choose $\rho \in R$
        $\pi \leftarrow \rho(\pi)$
    return π

Summary of Sections 4.1, 4.2, 4.3

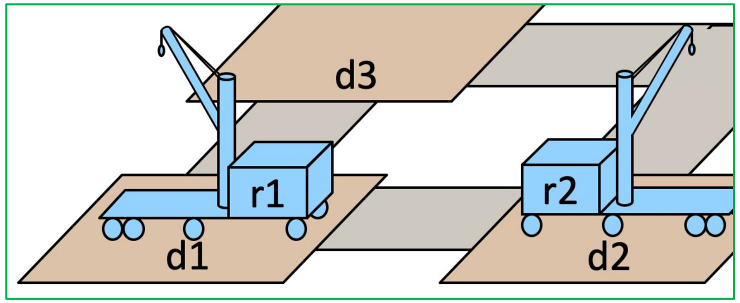
- Timelines
  - Temporal assertions (change, persistence), constraints
  - Conflicts, consistency, security, causal support
- Chronicle: union of several timelines
  - Consistency, security, causal support
- Actions represented by chronicles; preconditions ⇔ causal support
- Planning problems
  - three 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

Outline

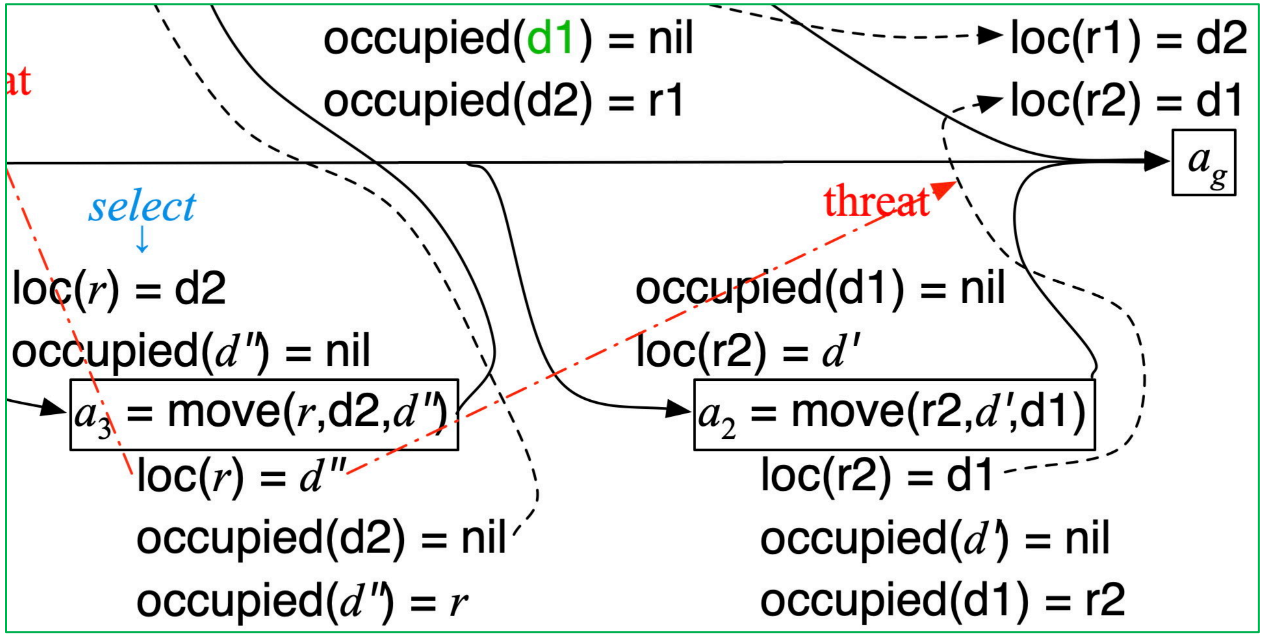
- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- **4.4 Constraint Management**
  - Consistency of object constraints and time constraints
  - Controlling the actions when we don’t know how long they’ll take
- 4.5 Acting with temporal models

Constraint Management

- Each time TemPlan applies a resolver, it modifies ($\mathcal{T}$,$\mathcal{C}$)
  - Some resolvers will make ($\mathcal{T}$,$\mathcal{C}$) inconsistent
    - No solution in this part of the search space
  - Would like to detect inconsistency, prune the search space
    - Otherwise we'll waste time looking for a solution



- Analogy: PSP checks simple cases of inconsistency
  - E.g., can't create a constraint $a \prec b$ if there's already a constraint $b \prec a$
- But PSP ignores more complicated cases
  - E.g., suppose there are three threats
    - To resolve them, suppose PSP chooses $d'' \neq$ d1 , $d'' \neq$ d2 , $d'' \neq$ d3
    - No solutions in this part of the search space, but PSP searches it anyway



Constraint Management in TemPlan

$\mathcal{T} = \{\ldots\}$

$C = (t_1 < t_2 < t_3 < t_1)$

- At various points, check consistency of C
  - If C is inconsistent, then $(\mathcal{T},C)$ is inconsistent
  - Can prune this part of the search space

- If C is consistent, $(\mathcal{T},C)$ may or may not be consistent
  - Example:
    - $\mathcal{T} = \{[t_1,t_2]$ loc(r1)=loc1,
      $[t_3,t_4]$ loc(r1)=loc2$\}$
    - $C = (t_1 < t_3 < t_4 < t_2)$
  - Gives loc(r1) two values during $[t_3,t_4]$

Consistency of C

- C contains two kinds of constraints
  - Object constraints
    - loc(r) $\neq l_2$, $l \in$ {loc3, loc4}, $r =$ r1, $o \neq o'$
  - Temporal constraints
    - $t_1 < t_3$, $a < t$, $t < t'$, $a \leq t' - t \leq b$

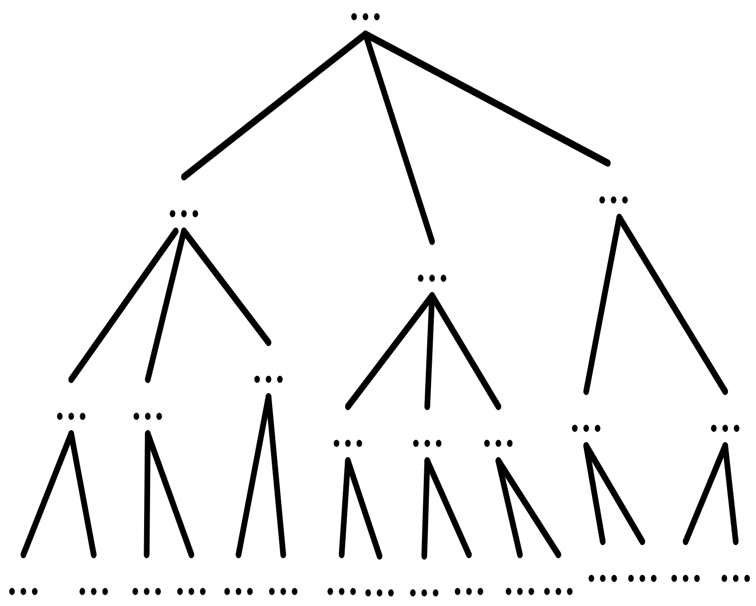
- Assume object constraints are independent of temporal constraints and vice versa
  - exclude things like $t <$ distance (r,r')

- 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 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

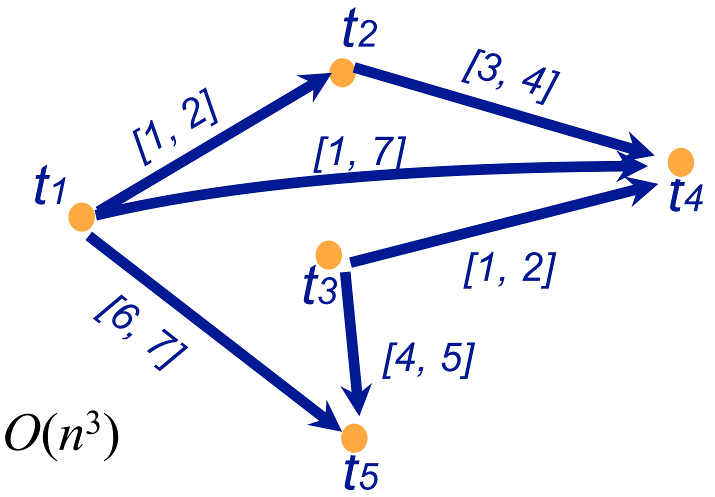


*See Russell & Norvig, Artificial Intelligence: A Modern Approach

Time Constraints

To represent time constraints:

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

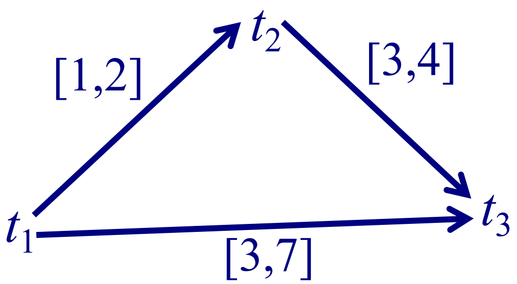


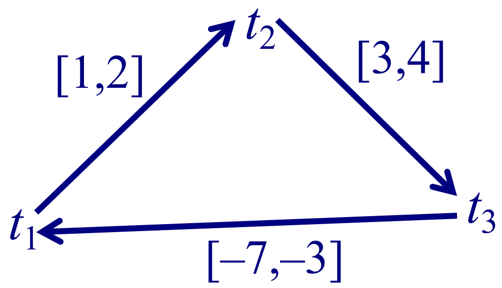
- Synthesize incrementally them starting from ϕ_0
  - Templan can check time constraints in time $O(n^3)$

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

Time Constraints

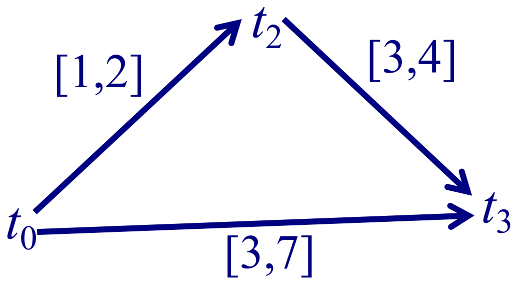
- *Simple Temporal Network* (STN):
- a pair $(\mathcal{V},\mathcal{E})$, where
  - $\mathcal{V}$ = {a set of temporal variables $\{t_1, \ldots, t_n\}$
  - $\mathcal{E} \subseteq \mathcal{V}^2$ is a set of arcs
- Each arc (t_i,t_j) is labeled with an interval $[a,b]$
  - Represents constraint $t_j - t_i \in [a, b]$
  - Or equivalently, $t_i - t_j \in [-b, -a]$





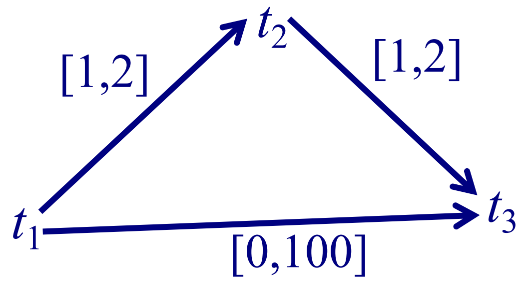
- Notation: instead of $t_j - t_i \in [a, b]$, write $r_{ij} = [a,b]$

- To represent unary constraints:
  - Dummy variable $t_0 \equiv 0$
  - Arc $r_{0i} = [a,b]$ represents $t_i - 0 \in [a,b]$



Time Constraints

- *Solution* to an STN:
  - any assignment of integer values to the time points such that all the constraints are satisfied
- *Consistent* STN: has a solution



- Solutions:
  - $(t_2–t_1,\ t_3–t_2,\ t_3–t_1) \in$
    $\{(1,1,2),\ (1,2,3),\ (2,1,3),\ (2,2,4)\}$

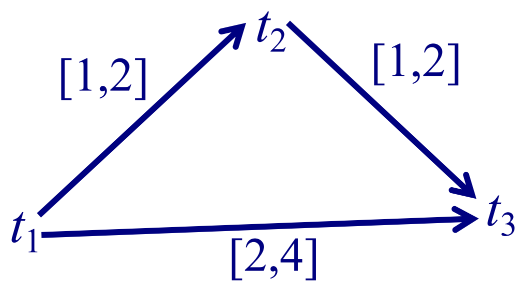
- *Minimal* STN:

  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 $t_j - t_i = t$

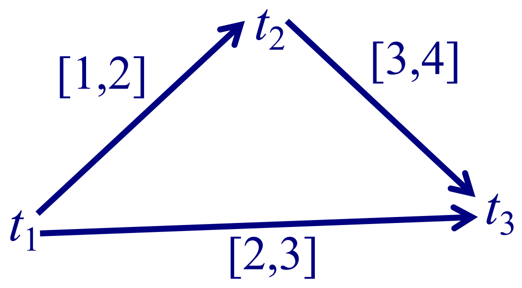
  - If we make any of the time intervals shorter, we'll exclude some solutions



Time Constraints

- *Solution* to an STN:
  - any assignment of integer values to the time points such that all the constraints are satisfied
- *Consistent* STN: has a solution

Poll: Is this network consistent?



- *Minimal* STN:

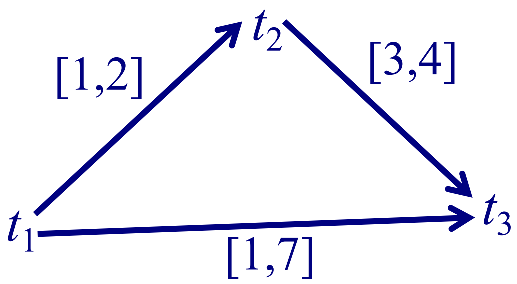
  for every arc (t_i,t_j) with label $[a,b]$,

  for every $t \in [a,b]$,

  there's at least one solution such that $t_j - t_i = t$

  - If we make any of the time intervals shorter, we'll exclude some solutions

Poll: Is this network minimal?



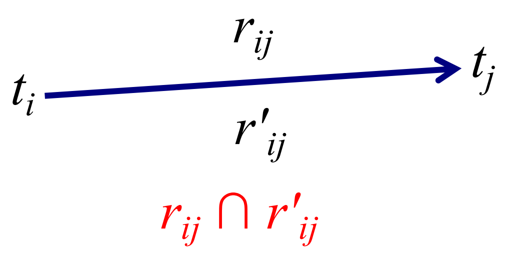
Operations on STNs

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



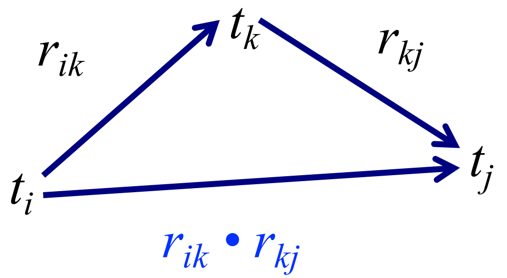
- Composition, $\bullet$

  $t_k - t_i \in r_{ik} = [a_{ik}, b_{ik}]$

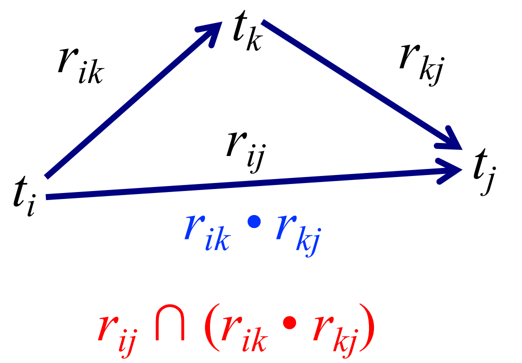
  $t_j - t_k \in r_{kj} = [a_{kj}, b_{kj}]$

Infer $t_j - t_i \in r_{ik} \bullet r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}]$

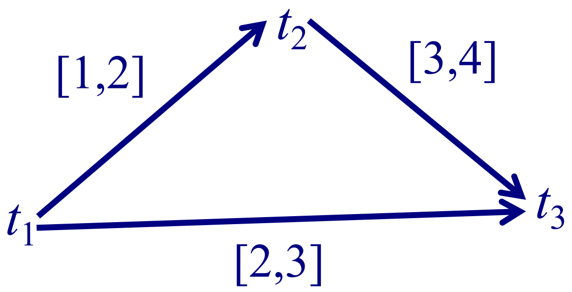
Reason: shortest and longest times for the two intervals



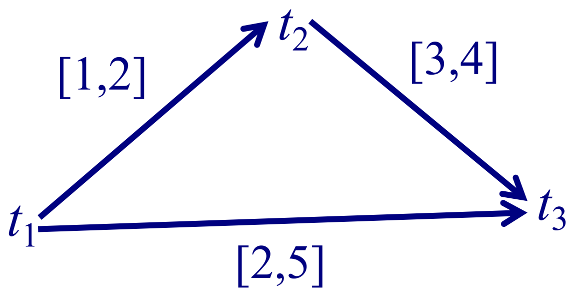
- Consistency checking
  - r_{ik}, r_{kj}, r_{ij} are consistent only if $r_{ij} \cap (r_{ik} \bullet r_{kj}) \neq \emptyset$
- Special case for networks with just three nodes
  - Consistent iff if $r_{ij} \cap (r_{ik} \bullet r_{kj}) \neq \emptyset$



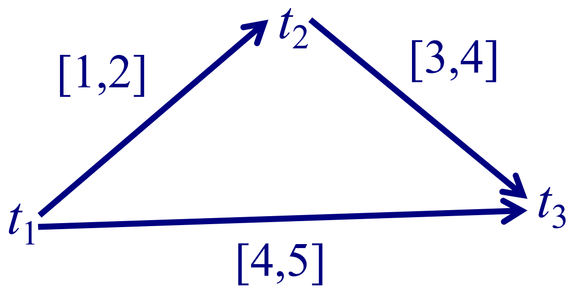
Two Examples



- 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:
  - $r'_{13} = r_{12} \bullet r_{23} = [4,6]$
- 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, $r'_{13} = [4,6]$
  - $r_{13} \cap r'_{13} = [4,5]$
- $(\mathcal{V},\mathcal{E})$ is consistent
  - $r_{13} \leftarrow [4,5]$ will make it minimal
  - Same solutions as above



- Let $d_{12} = t_2 - t_1 \in r_{12} = [1,2]$
  - $d_{23} = t_3 - t_2 \in r_{23} = [3,4]$
  - $d_{13} = d_{12} + d_{23}$
- Solution iff $d_{13} \in r_{13} = [2,5]$

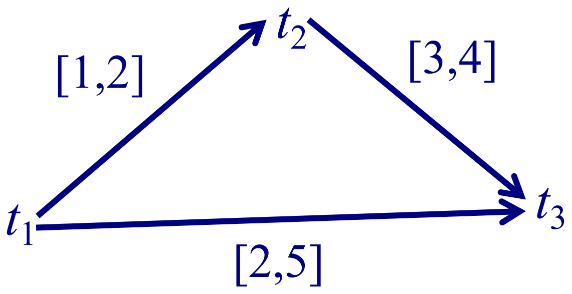
d_{12}	d_{23}	d_{13}	solution?
1	3	4	yes
1	4	5	yes
2	3	5	yes
2	4	6	no

- $d_{12} = 1, 2$
  - Can't shrink r_{12}
- $d_{23} = 3, 4$
  - Can't shrink r_{23}
- $d_{13} = 4, 5$
  - Shrink r_{13} to [4,5], get the same solutions

Operations on STNs

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

- If an arc has no constraint, use $[-\infty, +\infty]$
- PC (*Path Consistency*) algorithm:
  - For every $i < j$, iterates over r_{ij} once for each k
  - Each time, tries to reduce interval, checks consistency
  - i, j, k each go from 1 to n
    => time $O(n^3)$



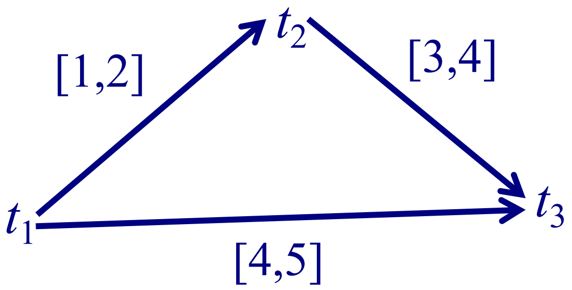
- For $n = 3$, PC tries these triples:
  - k=1, i=2, j=3
  - k=2, i=1, j=3
  - k=3, i=1, j=2

- k=1, i=2, j=3
  - $r_{ik} = r_{21} = [-2,-1]$
  - $r_{kj} = r_{13} = [2,5]$
  - $r_{ij} = r_{23} = [3,4]$
  - $r_{ik} \bullet r_{kj} = [-2+2, -1+5] = [0,4]$
  - $r_{23} = r_{ij} \leftarrow [\max(3,0), \min(4,4)] = [3,4]$
  - No change

- k=2, i=1, j=3 (see previous slide)
  - PC reduces $r_{ij} = r_{13}$ to [4,5]

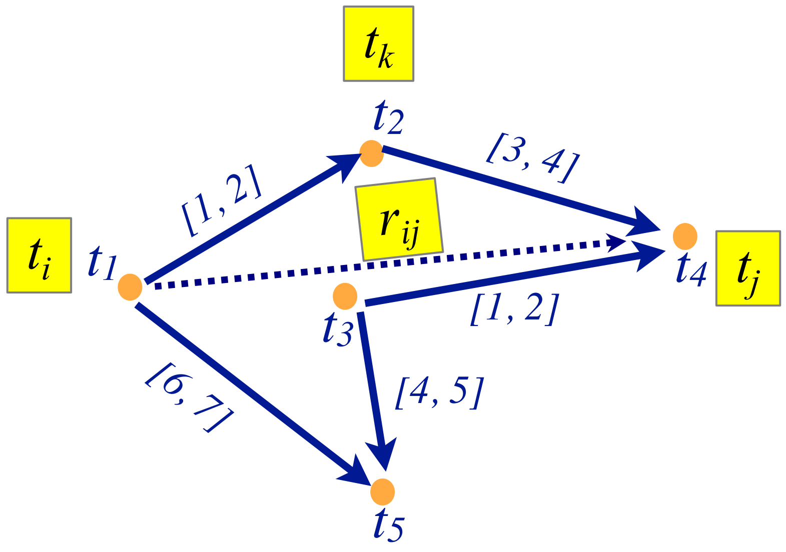
- k=3, i=1, j=2
  - $r_{ik} = r_{13} = [4,5]$
  - $r_{kj} = r_{32} = [-4,-3]$
  - $r_{ij} = r_{12} = [1,2]$
  - $r_{ik} \bullet r_{kj} = [4-4, 5-3] = [0,2]$
  - $r_{13} = r_{ij} \leftarrow [\max(1,-2), \min(2,2)] = [1,2]$
  - No change

- Minimal network:



Operations on STNs

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



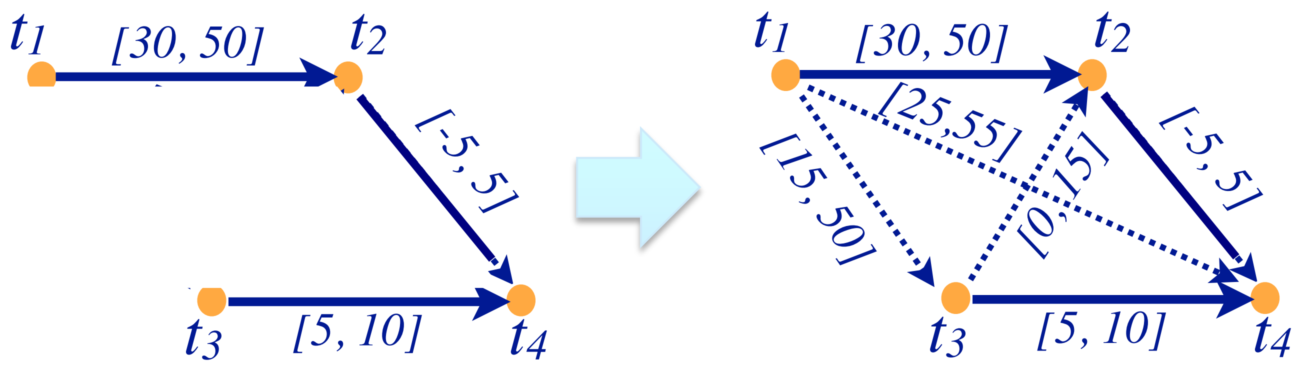
- i, j, k each go from 1 to n
  - $O(n^3)$ triples; I think exact value is $n(n–1)(n–2)/2$
    => time $O(n^3)$
- For $n = 4$, 12 triples:
  - k=1, i=2, j=3
  - k=1, i=2, j=4
  - k=1, i=3, j=4
  - k=2, i=1, j=3
  - k=2, i=1, j=4
  - k=2, i=3, j=4
  - k=3, i=1, j=2
  - k=3, i=1, j=4
  - k=3, i=2, j=4
  - k=4, i=1, j=2
  - k=4, i=1, j=3
  - k=4, i=2, j=3

- Example: $k = 2,\ i = 1,\ j = 4$

$$r_{ik} = r_{12} = [1,2]$$
$$r_{kj} = r_{24} = [3,4]$$
$$r_{ij} = r_{14} = [-\infty, \infty]$$
$$r_{ik} \bullet r_{kj} = [1+3, 2+4] = [4,6]$$
$$r_{14} = r_{ij} \leftarrow [\max(-\infty,4), \min(\infty,6)] = [4,6]$$

Operations on STNs

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



- PC makes network minimal
  - Reduces each r_{ij} to exclude values that aren't in any solution
- Also detects inconsistent networks
  - $r_{ij} = [a_{ij}, b_{ij}]$ empty => inconsistent

- Dashed lines: constraints shrunk from $[-\infty, \infty]$
- Can modify PC to make it incremental
  - Input:
    - a consistent, minimal STN
    - a new constraint r'_{ij}
  - Incorporate r'_{ij} in time $O(n^2)$

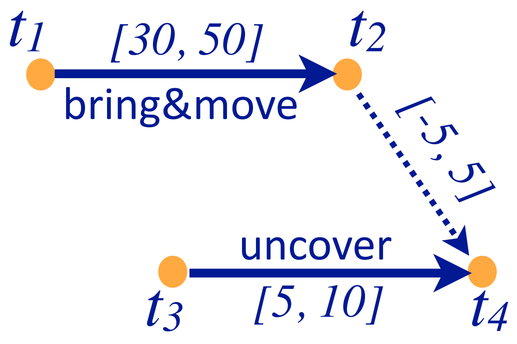
Pruning TemPlan's search space

- 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

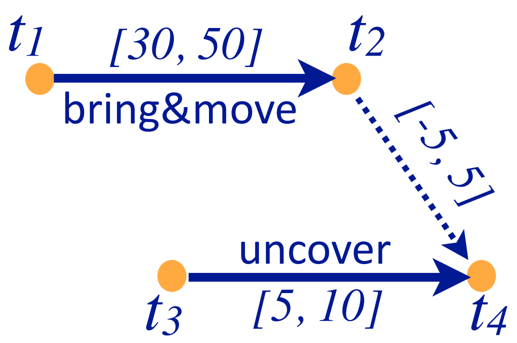
- Solid lines: duration constraints
  - Robot will do bring&move, will take 30 to 50 time units
  - Crane will do uncover, will take 5 to 10 time units
- Dashed line: synchronization constraint
  - At most 5 seconds between the two ending times



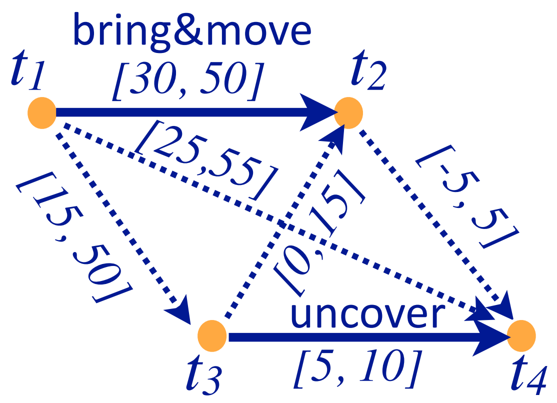
- Objective
  - Choose time points that will satisfy all the constraints

Controllability

- Suppose we run 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



- Actor can control when each action starts
  - t_1 and t_3 are *controllable*
- Can't control how long the actions take
  - t_2 and t_4 are *contingent*
  - 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

Suppose we start bring&move at time $t_1 = 0$

- Suppose the durations are bring&move 30, uncover 10

  Then

  $$t_2 = t_1 + 30 = 30$$

  $$t_4 = t_3 + 10$$

  so $t_4 - t_2 = t_3 - 20$

- 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 $t_3 \leq 25$

- Suppose the durations are bring&move 50, uncover 5

  Then

  $$t_2 = t_1 + 50 = 50$$

  $$t_4 = t_3 + 5$$

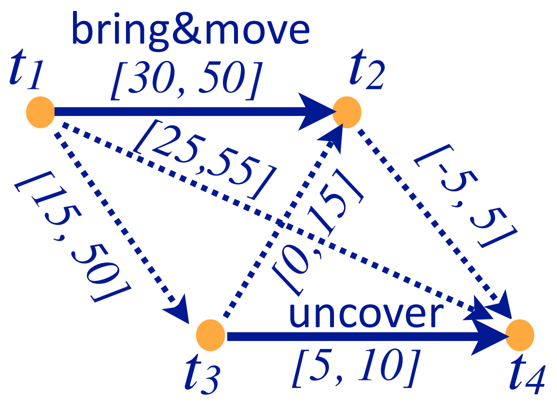
  so $t_4 - t_2 = t_3 - 45$

- Constraint r_{24}: $-5 \leq t_4 - t_2 \leq 5$

  $$-5 \leq t_3 - 45 \leq 5$$

  $$40 \leq t_3 \leq 50$$

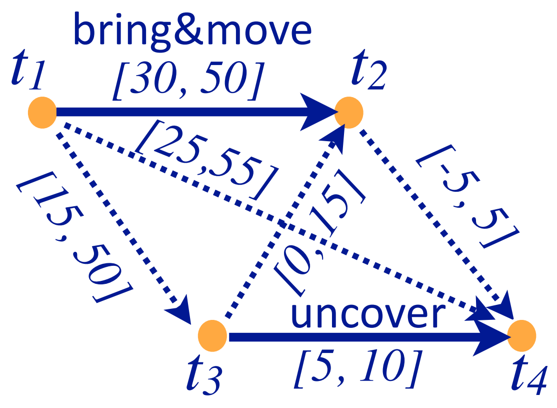
- Must start uncover at $t_3 \geq 40$



There's no t_3 that works in both cases

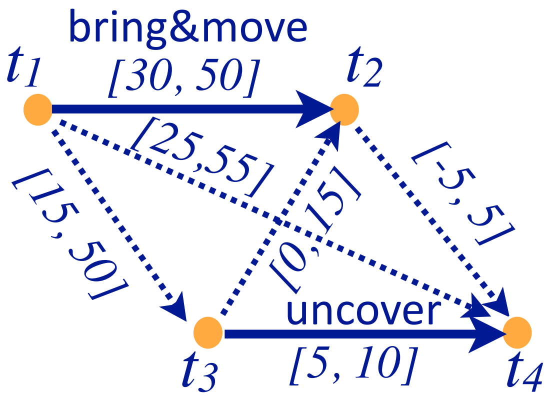
STNUs

- *STNU (Simple Temporal Network with Uncertainty):*
  - A 4-tuple $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$
    - $\mathcal{V}$ ={*controllable* time points}, e.g., starting times of actions
    - $\tilde{\mathcal{V}}$ ={*contingent* time points}, e.g., ending times of actions
    - $\mathcal{E}$ ={*controllable* constraints}, $\tilde{\mathcal{E}}$ ={*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 $\mathcal{V}$ (starting times) that guarantee that the constraints are satisfied



Poll. is r_{32} controllable?

Three kinds of controllability



- $(\mathcal{V},\tilde{V},\mathcal{E},\tilde{E})$ is *strongly controllable* if the actor can choose values for $\mathcal{V}$ such that success will occur for all values of $\tilde{V}$ that satisfy $\tilde{E}$
  - Actor can choose the values for $\mathcal{V}$ offline
  - The right choice will work regardless of $\tilde{V}$
- $(\mathcal{V},\tilde{V},\mathcal{E},\tilde{E})$ 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 $\tilde{V}$
  - Actor can choose the values for $\mathcal{V}$ 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 σ telling what to do in every situation
    - Choices may depend on what has happened so far
  - $(\mathcal{V},\tilde{V},\mathcal{E},\tilde{E})$ is *dynamically controllable* if ∃ strategy for actor that will guarantee success regardless of the environment's strategy

Two player, zero sum, extensive form, imperfect information game

Poll. Is the above STNU strongly controllable?

Poll. Is it weakly controllable?

Poll. Is it dynamically controllable?

Dynamic Execution

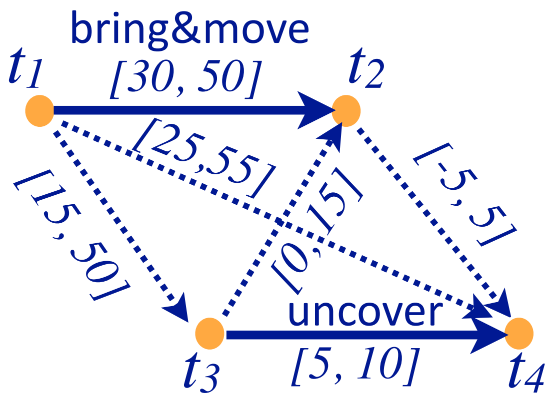
For $t = 0, 1, 2, \ldots$

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 $\mathcal{E}$
   - ≈ actions the actor chooses to start at time t
2. Simultaneously, environment chooses an unassigned set of variables $\tilde{V}_t \subseteq \tilde{V}$ that all can be assigned the value t without violating any constraints in $\tilde{E}$
   - ≈ 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 $\mathcal{E} \cup \tilde{E}$ are violated
   - There might be violations that neither $\mathcal{V}_t$ nor $\tilde{V}_t$ caused individually
5. Success if all variables in $\mathcal{V} \cup \tilde{V}$ have values and no constraints are violated

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

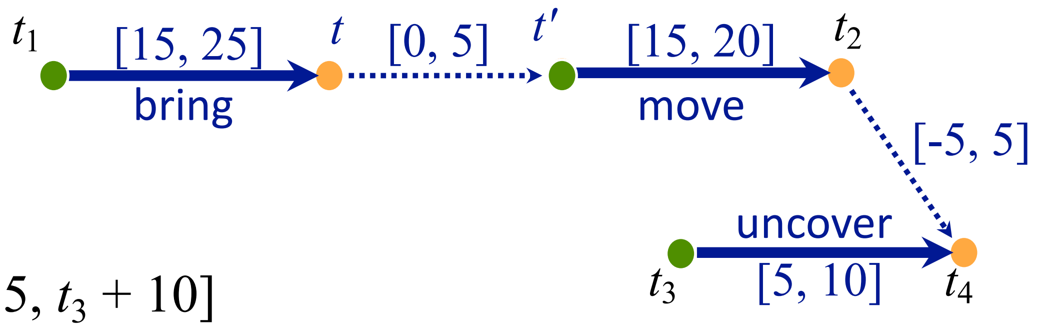
- *Dynamic execution strategy* σ_A for actor, σ_E for environment
  - $\sigma_A(h_{t-1})$ = {what events in $\mathcal{V}$ to trigger at time t, given h_{t-1}}
  - $\sigma_E(h_{t-1})$ = {what events in $\tilde{V}$ to trigger at time t, given h_{t-1}}
    - $h_t = h_{t-1} \,.\, (\sigma_A(h_{t-1}) \cup \sigma_E(h_{t-1}))$
  - $(\mathcal{V},\tilde{V},\mathcal{E},\tilde{E})$ 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
  - Then it's dynamically controllable

- 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



Poll. Is the above STNU strongly controllable?

Dynamic Controllability Checking

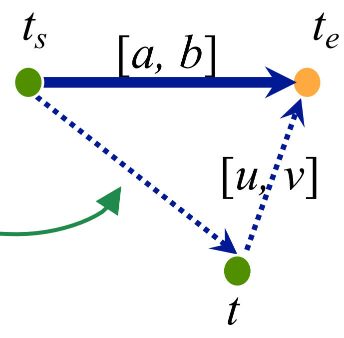
- For a chronicle $\phi = (\mathcal{A}, S_\mathcal{T}, \mathcal{T}, \mathcal{C})$
  - Temporal constraints in $\mathcal{C}$ correspond to an STNU
  - Put code into TemPlan to keep the STNU dynamically controllable
- If we detect cases where it isn't dynamically controllable, then backtrack
- If PC($\mathcal{V} \cup \tilde{V}, \mathcal{E} \cup \tilde{E}$) reduces a contingent constraint then $(\mathcal{V}, \tilde{V}, \mathcal{E}, \tilde{E})$ isn't dynamically controllable

  $\Rightarrow$ can prune this branch
- If it *doesn't* reduce any contingent contraints, we don't know whether $(\mathcal{V}, \tilde{V}, \mathcal{E}, \tilde{E})$ is dynamically controllable
- Two options
  - Either continue down this branch and backtrack later if necessary, or
  - Extend PC to detect more cases where $(\mathcal{V}, \tilde{V}, \mathcal{E}, \tilde{E})$ isn't dynamically controllable
    - additional constraint propagation rules

```
PC(V,E):
   for 1 ≤ i ≤ n, 1 ≤ j ≤ n, 1 ≤ k ≤ n,
         i ≠ j, i ≠ k, j ≠ k do
      r_ij ← r_ij ∩ [r_ik • r_kj]
      if r_ij = ∅ 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 $[a',b'] \bullet [u,v] = [a,b]$
  - $[a'+u,\ b'+v] = [a,b]$
  - $a' = a - u,\ \ b' = b - v$



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

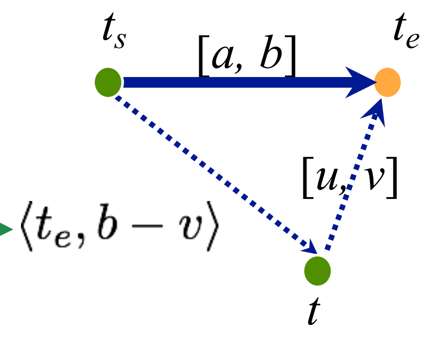
- I think this should be $[a', b']$

$\Rightarrow$ contingent
$\rightarrow$ controllable

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

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



- As before, let $b' = b - v$

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'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

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

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 primitive actions may correspond to compound tasks
  - In RAE, use refinement methods to refine them into commands
  - Templan's primitive action template (descriptive model)

$\text{leave}(r, d, w)$
- assertions: $[t_s, t_e]\text{loc}(r){:}(d, w)$
  $[t_s, t_e]\text{occupant}(d){:}(r, \text{empty})$
- constraints: $t_e \leq t_s + \delta_1$
  $\text{adjacent}(d, w)$

  - RAE's refinement method (operational model)

$\text{m-leave}(r, d, w, e)$
- task: $\text{leave}(r, d, w)$
- pre: $\text{loc}(r){=}d, \text{adjacent}(d, w), \text{exit}(e, d, w)$
- body: until $\text{empty}(e)$ $\text{wait}(1)$
  $\text{goto}(r, e)$

Atemporal Refinement of Primitive Actions

- Templan's primitive actions may correspond to compound tasks
  - In RAE, use refinement methods to refine them into commands
  - Templan's primitive action template (descriptive model)
  - RAE's refinement method (operational model)

unstack(k,c,p)
  assertions: …
  constraints: …

m-unstack(k, c, p)
  task: unstack(k, c, p)
  pre: pos(c)=p, top(p)=c, grip(k)=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)

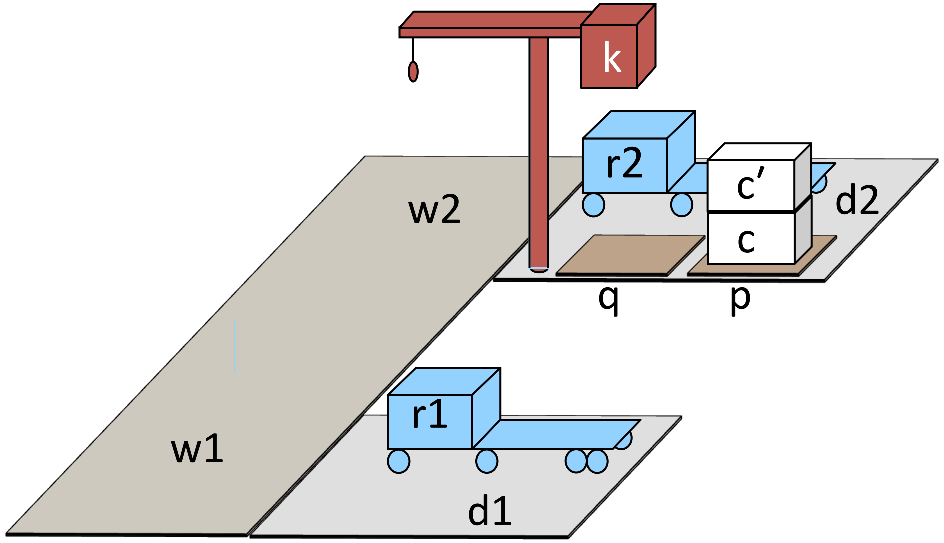
Discussion

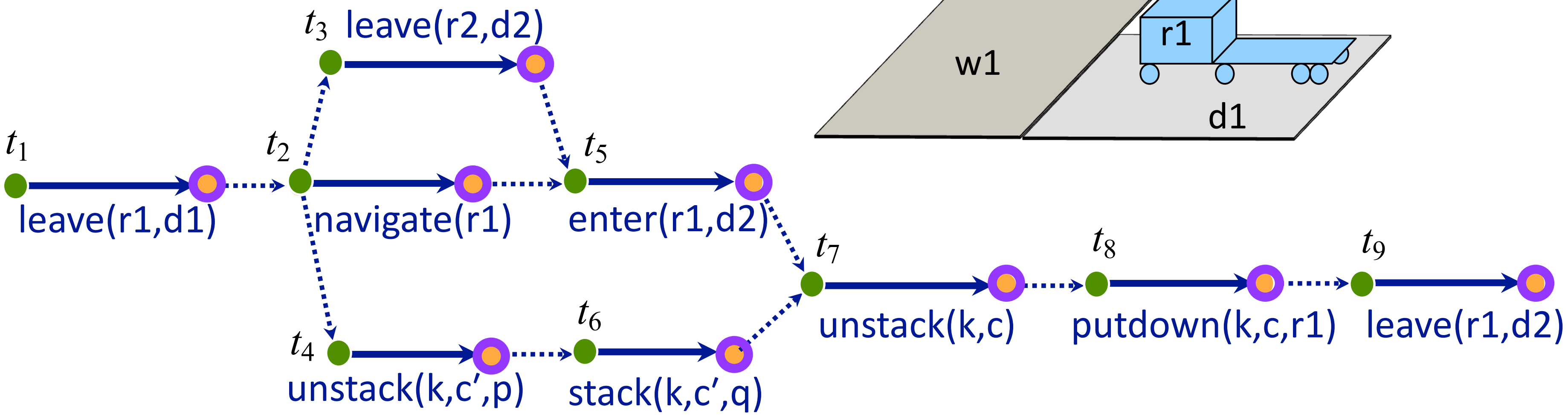
- Pros
  - Simple online refinement with RAE
  - Avoids breaking down uncertainty of contingent duration
  - Can be augmented with temporal monitoring functions in RAE
    - E.g., watchdogs, methods with duration preferences
- Cons
  - Does not handle temporal requirements at the command level,
    - 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



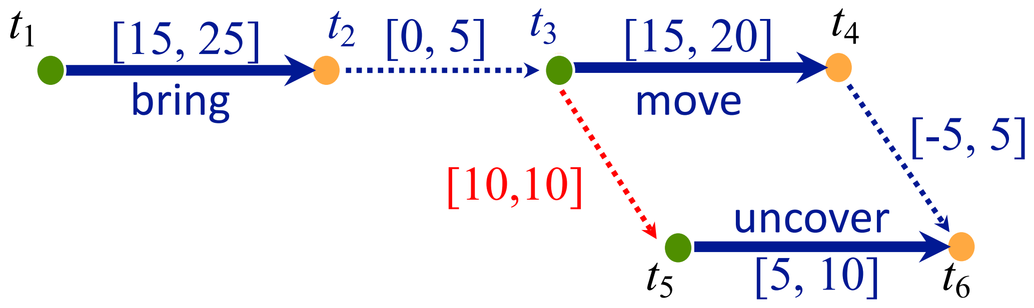


Dispatching

- Let $(\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}})$ be a controllable STNU that's *grounded*
- Different from a grounded expression in logic
  - At least one time point in $(\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}})$ is instantiated
- Bounds every time point t_i within an interval $[l_i,u_i]$

Controllable time point t in the future:

- t_i is *alive* if current time *now* $\in [l_i\,, u_i]$
- t_i is *enabled* if
  - it's alive
  - for every precedence constraint $t' < t_i$, t' has occurred
  - for every wait constraint $\langle t_e, \alpha \rangle$, t_e has occurred or α has expired



- Let $t_1 = 0$. Then:
  - $t_2 \in [15,25]$
  - $t_3 \in [t, t{+}5]$
  - $t_4 \in [t_3{+}15,\ t_3{+}20]$
  - $t_5 \in [t_4{+}10,\ t_4{+}10]$
  - $t_6 \in [t_5{+}5,\ t_5{+}10]$
- Suppose bring finishes at t=20, and *now* = 25. Then
  - $t3$

Dispatching

- Let $(\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}})$ be a controllable STNU that's *grounded*
- Different from a grounded expression in logic
  - At least one time point in $(\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}})$ is instantiated
- Bounds every time point t_i within an interval $[l_i, u_i]$

Controllable time point t in the future:

- t_i is *alive* if current time *now* $\in [l_i\, , u_i]$
- t_i is *enabled* if
  - it's alive
  - for every precedence constraint $t' < t_i$, t' has occurred
  - for every wait constraint $\langle t_e, \alpha\rangle$, t_e has occurred or α has expired

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 $t_i \in$ *enabled* such that *now* $= u_i$
  - arbitrarily choose other time points in *enabled*, and trigger them
  - propagate values of triggered timepoints (change $[l_j, u_j]$ for each future timepoint t_j)

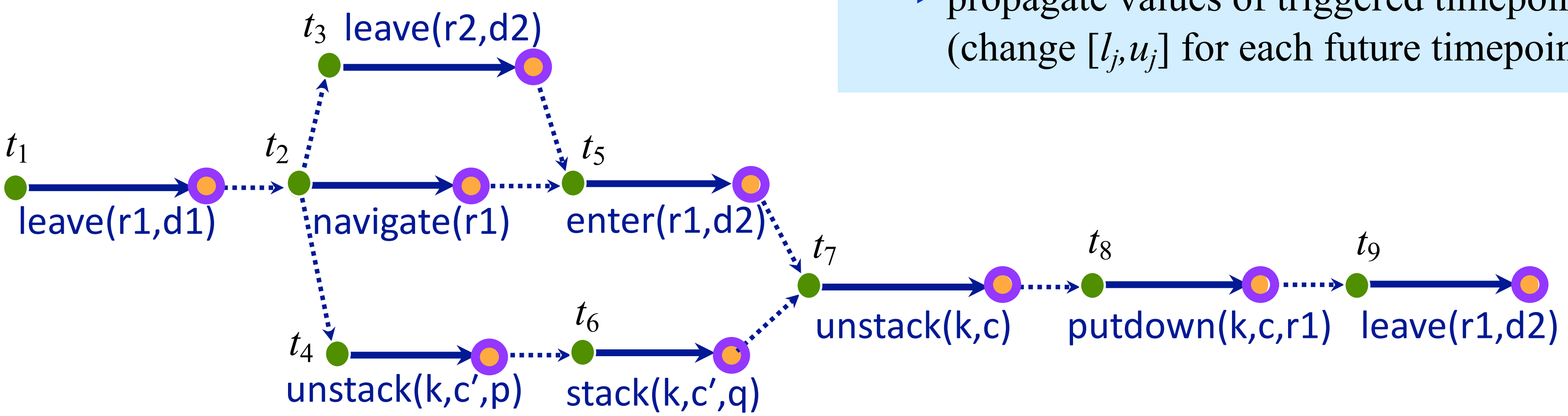
t_i is bounded by $[l_i\, , u_i]$

Abstract Example

- trigger t_1, propagate $[l_j,u_j]$ values
- observe leave finish, this enables t_2
- Trigger t_2, propagate $[l_j,u_j]$ values
- This enables t_3, t_4
- trigger t_3 soon enough to allow enter(r1,d2) at time t_5
  - propagate $[l_j,u_j]$ values
- trigger t_4 soon enough to allow stack(k,c′) at time t_6
  - propagate $[l_j,u_j]$ values
- rest of plan is linear: trigger each t_i after previous action ends

before u_3

before u_4

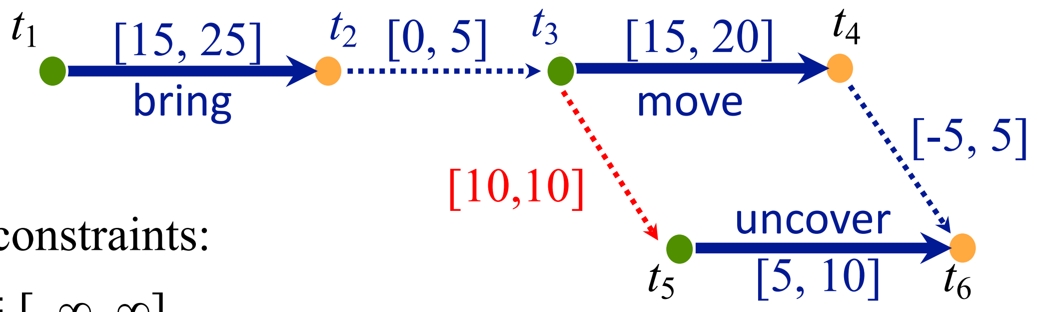


Dispatch($\mathcal{V},\tilde{V},\mathcal{E},\tilde{E}$)

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

t_i is bounded by $[l_i, u_i]$

Detailed Example



- Initial constraints:
  - $t_1 \in [-\infty, \infty]$
  - $t_2 \in [t_1+15, t_1+25]$
  - $t_3 \in [t_2, t_2+5]$
  - $t_4 \in [t_3+15, t_3+20]$
  - $t_5 \in [t_3+10, t_3+10]$
  - $t_6 \in [t_5+5, t_5+10] \cap [t_4-5, t_4+5]$
- Start at time *now*=0
  - $enabled = \{t_1\}$
- Trigger t_1 (bring) when *now*=0
- Propagate t_1=0:
  - $t_2 \in [15,25]$

- Suppose bring ends at *now*=20
  - $enabled = \{t_3\}$
- Propagate $t_2 = 20$
  - $t_3 \in [20, 25]$
- Trigger t_3 (move) when *now*=25
- Propagate t_3=25:
  - $t_4 \in [40, 45]$
  - $t_5 \in [35, 35]$
- At time *now*=35,
  - $enabled = \{t_5\}$
  - $now = u_5$, must trigger t_5

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

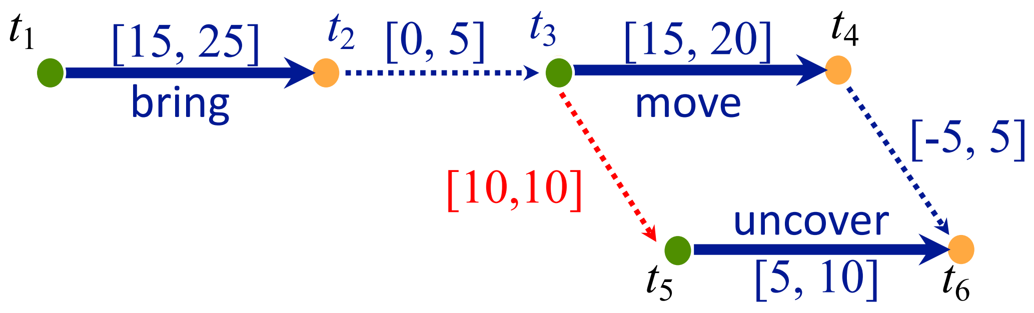
- initialize the network
- while there are time points in $\mathcal{V}$ that haven't been triggered, do
  - update *now*
  - update the time points in $\tilde{V}$ that were triggered since the last iteration
  - update *enabled*
  - trigger every $t_i \in enabled$ such that $now = u_i$
  - arbitrarily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

t_i is bounded by $[l_i , u_i]$

- Trigger t_5 (uncover) when $now = 35$
- Propagate t_5=35:
  - $t_6 \in [40, 45] \cap [t_4-5, t_4+5]$
- Suppose move takes 15 time units, ends at *now*=40
  - $enabled = \emptyset$
- Propagate t_4=40
  - $t_6 \in [40, 45] \cap [35, 45] = [40, 45]$

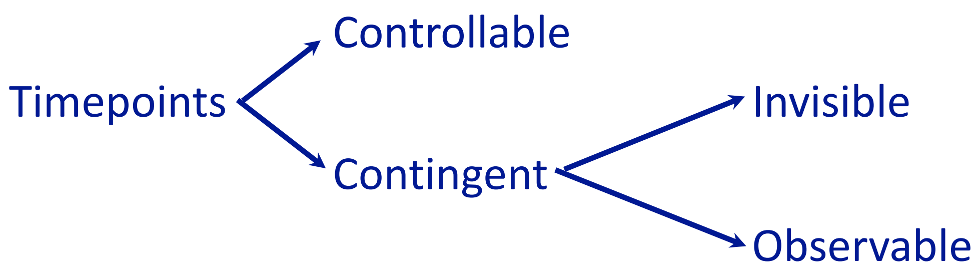
Deadline Failures

- 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



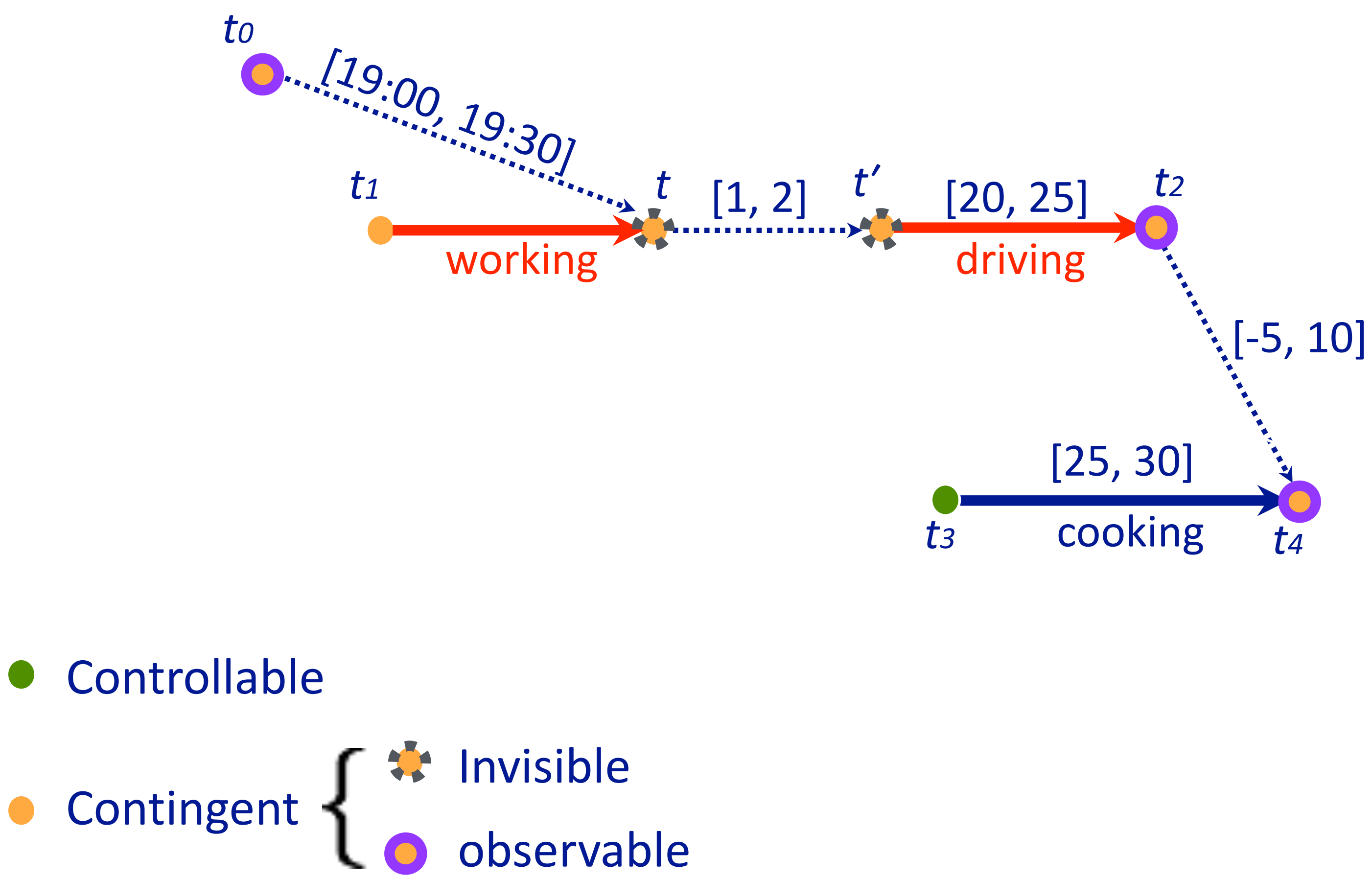
Partial Observability

- 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)



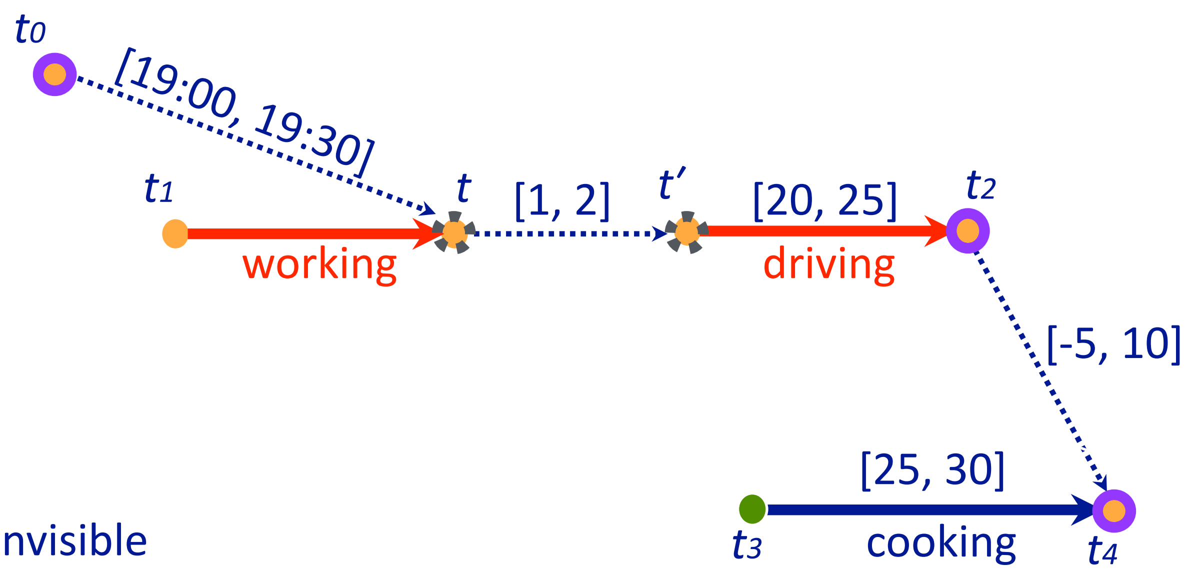
- Dynamically controllable?

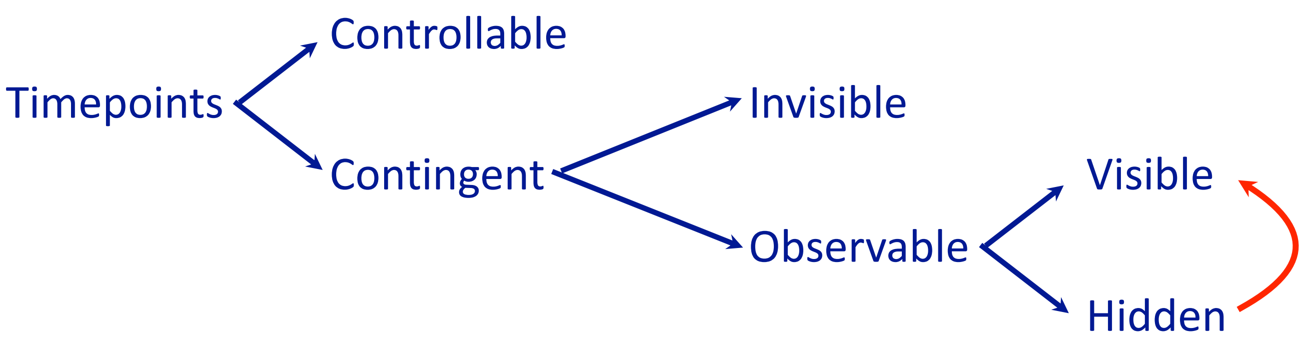
Observation Actions



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 ≠ 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 of Sections 4.4, 4.5

- Managing constraints in TemPlan: like CSPs
  - Temporal constraints: STNs, PC algorithm (path consistency)
- Acting
  - Dynamic controllability
  - STNUs
  - RAE and eRAE
  - Dispatching