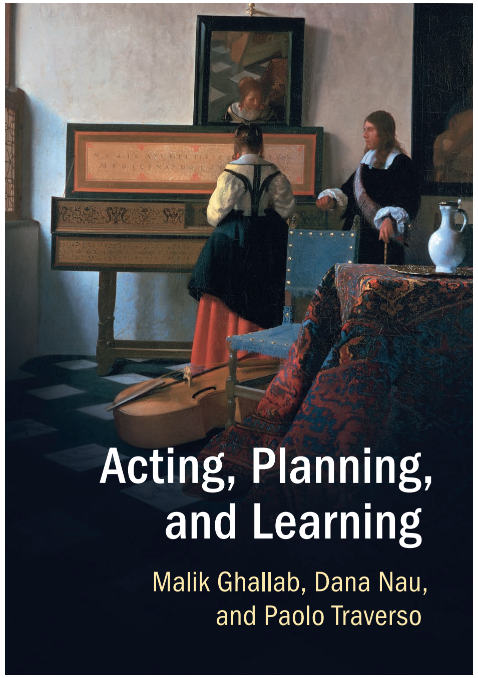
Last update: 11:10 PM, March 7, 2025



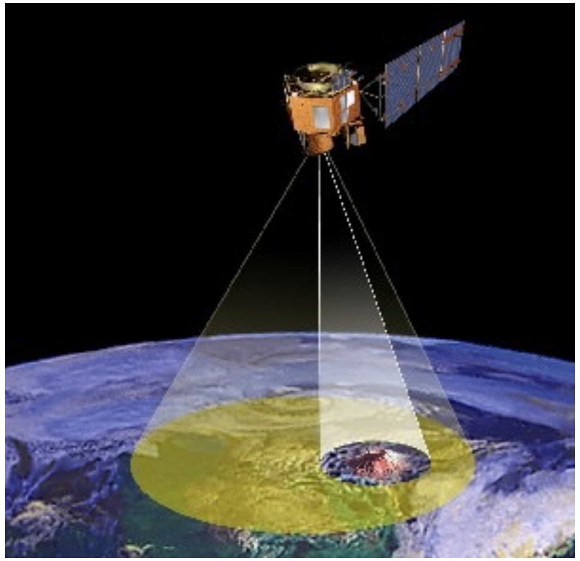
Chapters 17, 18
Temporal Represention, Acting, Planning

Dana S. Nau
University of Maryland

with contributions from
Mark "mak" Roberts

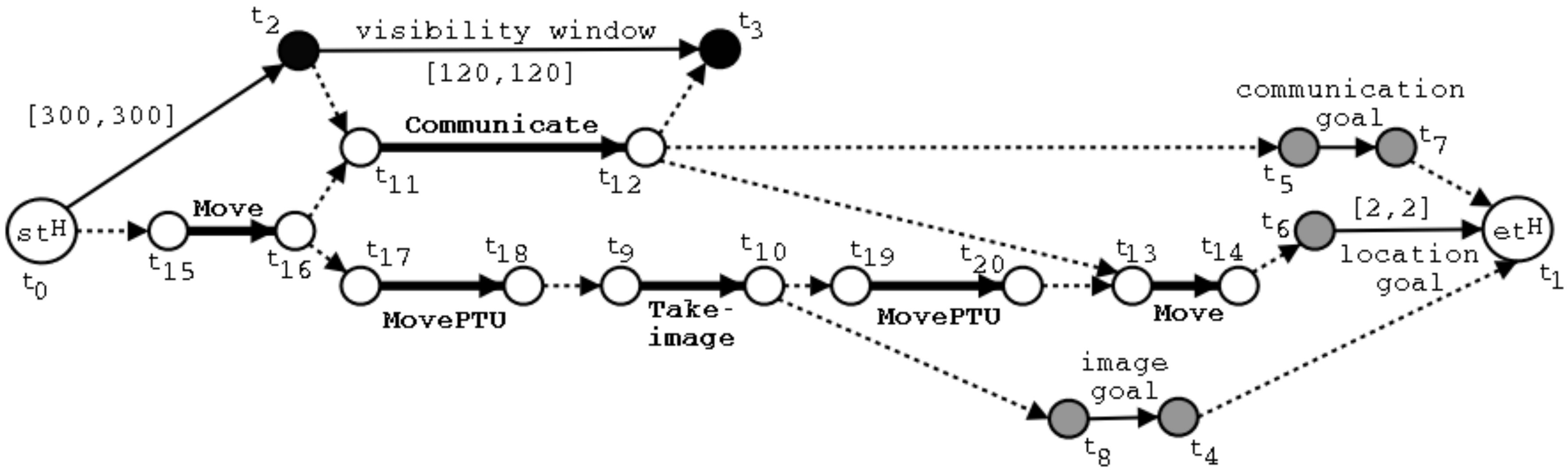
Some Example Applications

- RAX/PS
  - Planning/control of Deep Space One spacecraft
  - NASA Ames and JPL, 1999
- CASPER
  - Planning/control of spacecraft
  - NASA JPL, ≈ 1999–2017
- T-ReX
  - Planning/control of AUVs
  - Monterey Bay Aquarium Research Institute, ≈ 2005-2010



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



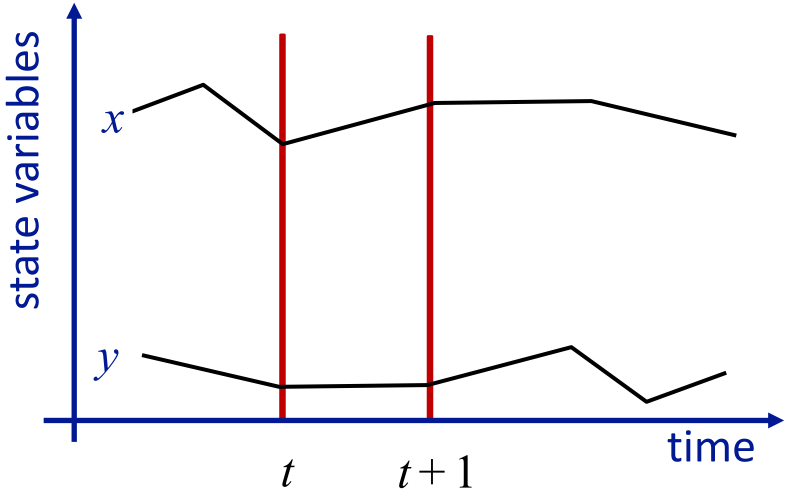
Outline

Topic	Section
• Introduction	17.1
• **Representation**	17.2
• Planning (briefly)	18.2
• Consistency and controllability	18.3
• Acting (Part 1: refinement)	17.3.1
• Acting (Part 2: dispatching)	17.3.1

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 = {values of all state variables at time t}

different positions on the y axis represent qualitative changes, not numeric values



Timeline

- A pair $(\mathcal{T}, \mathcal{C})$
  - $\mathcal{T}$ = {temporal assertions}; $\mathcal{C}$ = {constraints}
  - *partially* predicted evolution of one state variable
    - doesn't necessarily specify a value at every timepoint

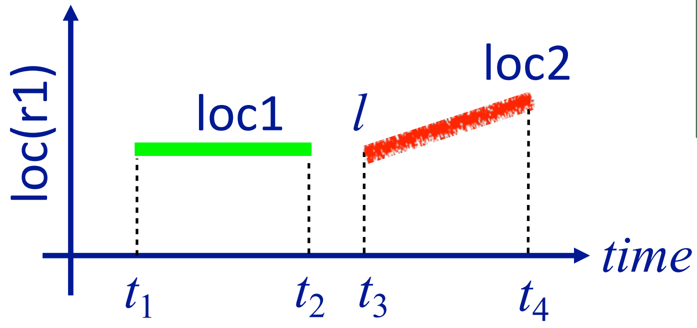
persistence
requires $t_1 \le t_2$

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

temporal constraints

object constraints

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



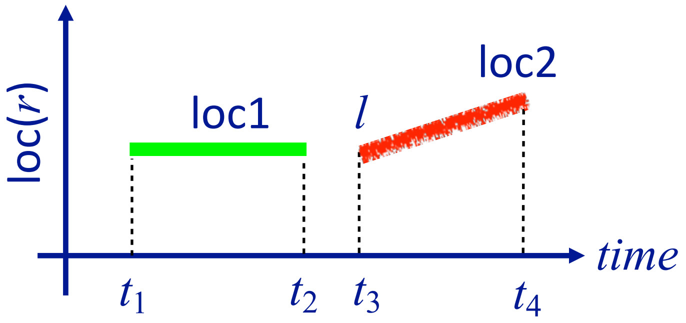
Reminder: qualitative changes, not numeric values

- If $\mathcal{T}$ contains $[t,t']\, x : (v,v')$ or $[t,t']\, x = v$ then $\mathcal{C}$ always contains $t \le t'$
  - To simplify the examples, we usually won't write $t \le t'$ 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 both
    - $\mathcal{T}'$ satisfies $\mathcal{C}'$
    - 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
- Two temporal assertions are *conflicting* if they have at least one inconsistent instance
  - May also have consistent instances, so "possibly conflicting" would be more accurate

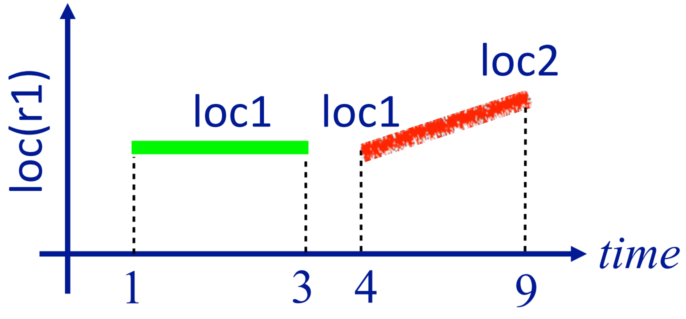
- Timeline:
  - $\mathcal{T}_1 = \{[t_1,t_2]\ \mathsf{loc}(r) = \mathsf{loc1},$
    $[t_3,t_4]\ \mathsf{loc}(r) : (l, \mathsf{loc2})\}$
  - $\mathcal{C}_1 = \{t_1 < t_2,\ t_3 < t_4,\ l \neq \mathsf{loc2}\}$



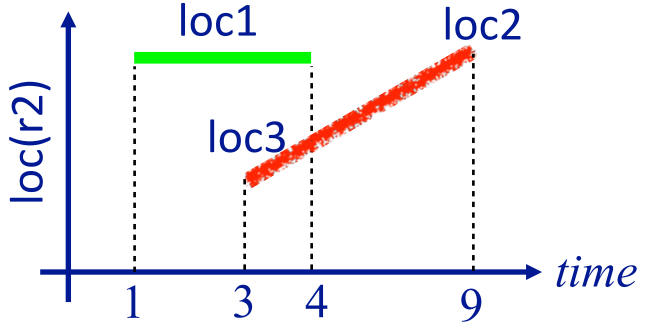
Poll 1: Is $(\mathcal{T}_1, \mathcal{C}_1)$ consistent?

Poll 2: are the two temporal assertions conflicting?

A. Yes
B. No
C. don't know



a consistent ground instance

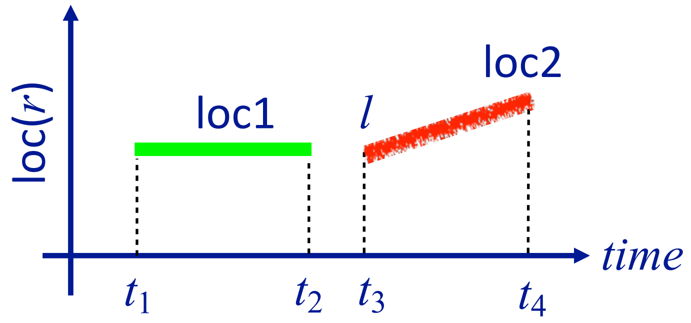


an inconsistent ground instance

Security

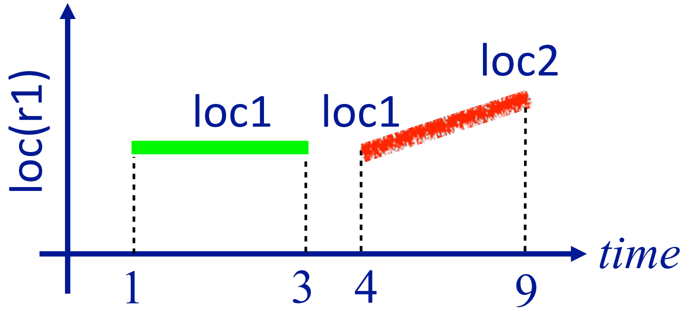
- $(\mathcal{T}, C)$ is *secure* if
  - it's consistent (at least one ground instance is consistent)
  - *every* ground instance that satisfies the constraints is consistent
- In PSP (Chapter 2), analogous to a partial plan that has no threats
- Can make a consistent timeline secure by adding *separation constraints* to C
  - additional temporal and object constraints
- Analogous to resolvers in PSP

- Not secure:
  - $\mathcal{T}_1 = \{[t_1,t_2]\ \mathrm{loc}(r) = \mathrm{loc1},\ [t_3,t_4]\ \mathrm{loc}(r):(l, \mathrm{loc2})\}$
  - $C_1 = \{t_1 < t_2,\ t_3 < t_4,\ l \neq \mathrm{loc2}\}$

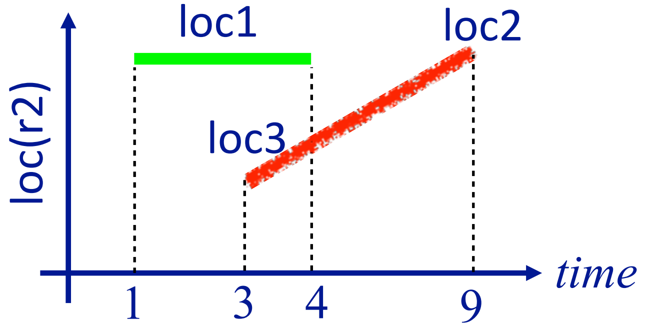


- Separation constraints:
  - $t_2 < t_3$

  or
  - $t_2 = t_3,\ l = \mathrm{loc1}$



a consistent ground instance

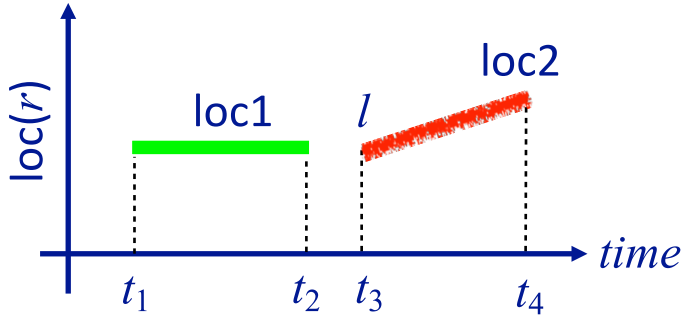


an inconsistent ground instance

Causal support

- Consider the assertion $[t_3,t_4]$ loc(r) : (l, loc2)
  - How did r1 get to location l?

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



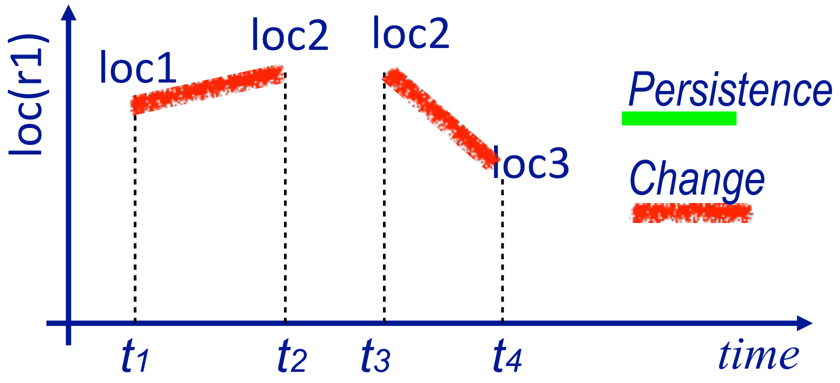
- Let α be a persistence $[t_1,t_2]\ x = v_1$ or change $[t_1,t_2]\ x : (v_1, v_2)$
- *Causal support* for α
  - Information saying α is supported *a priori*
  - Or another assertion that produces $x = v_1$ at time t_1
    - $[t_0,t_1]\ x = v_1$
    - $[t_0,t_1]\ x : (v_0, v_1)$

- A timeline $(\mathcal{T},\mathcal{C})$ 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

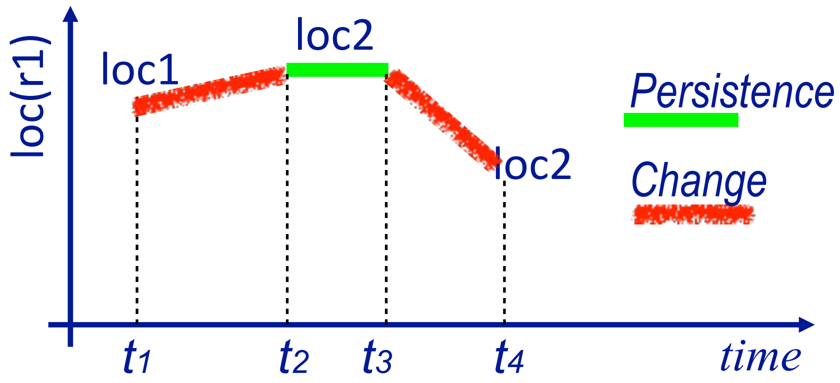
(1) Add a persistence assertion



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

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

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

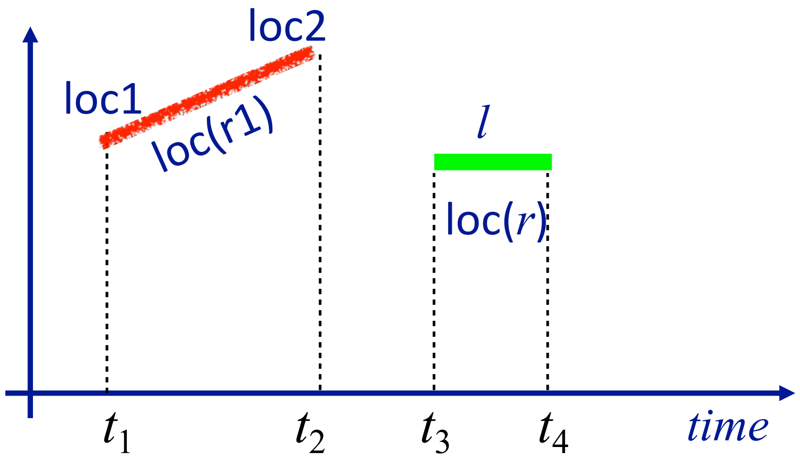


Establishing causal support

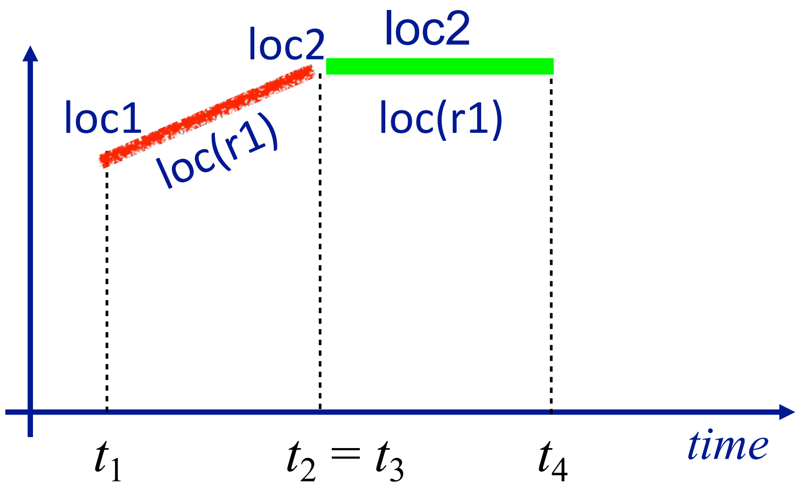
(2) Add constraints

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

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

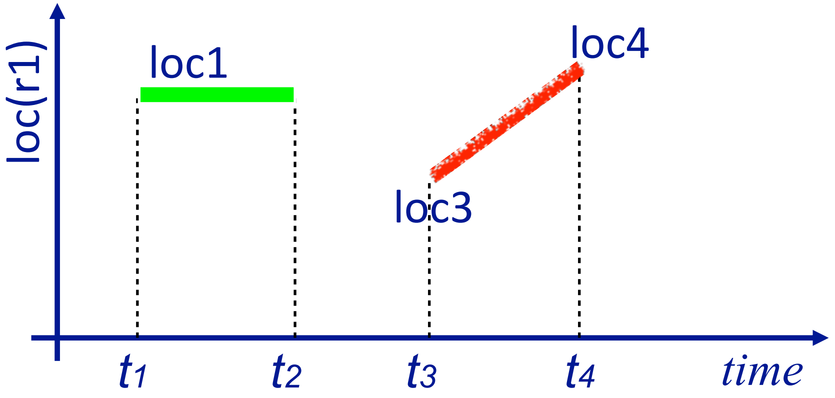


- Add $t_2 = t_3$, r = r1, l = loc2



Establishing causal support

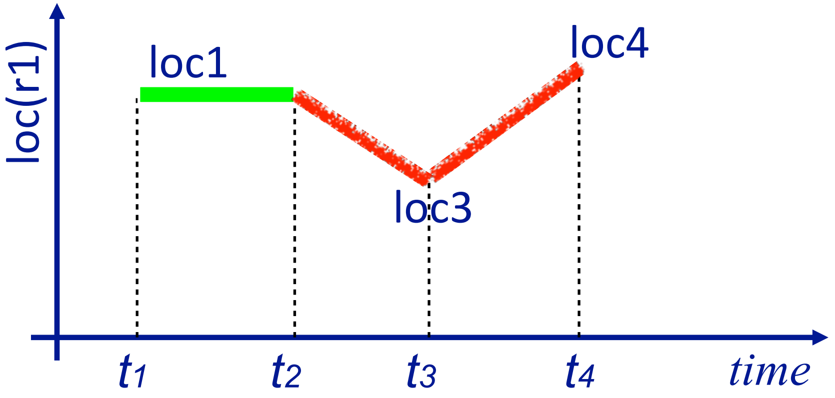
(3) Add an action



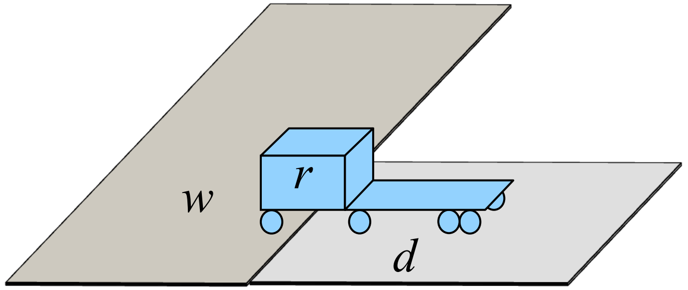
$\mathcal{T} = \{[t_1,t_2]$ loc(r1) = loc1,
$[t_3,t_4]$ loc(r1):(loc3,loc4)$\}$

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

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



Action Schemas



- *Action schema* (book also calls it a *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:

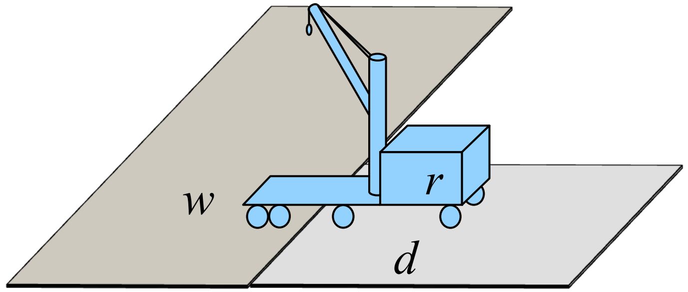
[t_s,t_e] loc(r): (d,w)
[t_s,t_e] occupant(d): (r,empty)

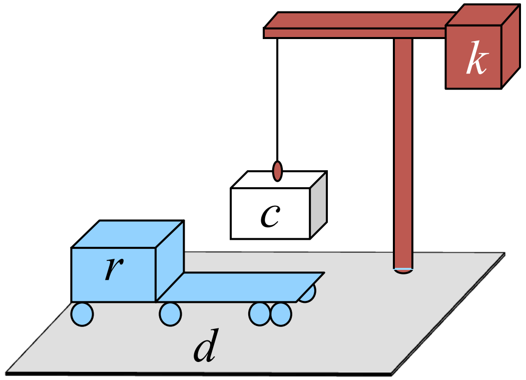
constraints:

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

- Action duration $t_e - t_s \leq \delta_1$
  - (I'm not sure why it's ≤)

Action Schemas





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:

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

- Action duration $t_e - t_s \le \delta_2$
- 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)

Action Schemas

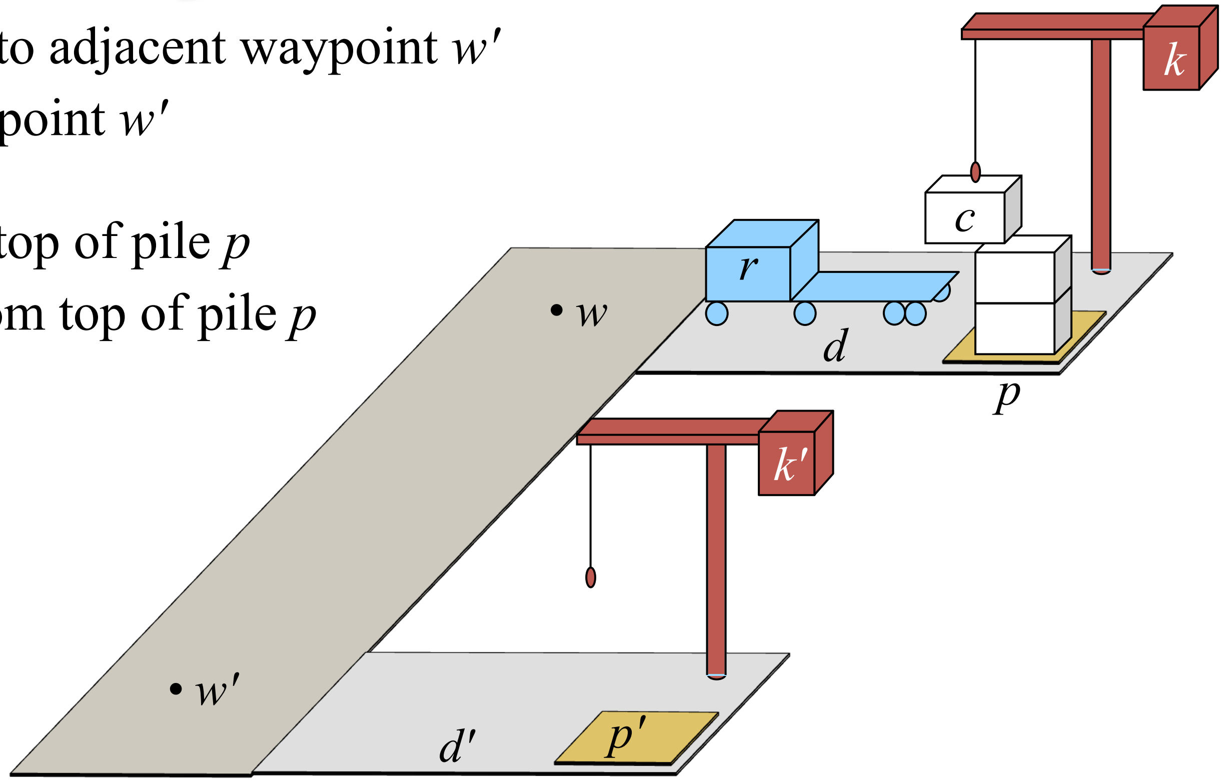
- leave(r,d,w) robot r leaves dock d to an adjacent waypoint w
- enter(r,d,w) r enters d from an adjacent waypoint w

- 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 adjacent waypoint w'
- connected(w,w') waypoint w is 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: ← *tasks and actions*

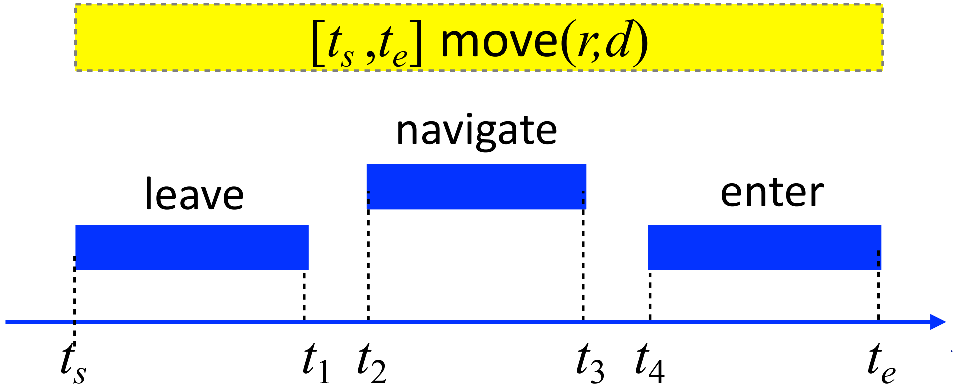
  $[t_s,t_1]$ leave(r,d',w')
  $[t_2,t_3]$ navigate(r,w',w)
  $[t_4,t_e]$ enter(r,d,w)

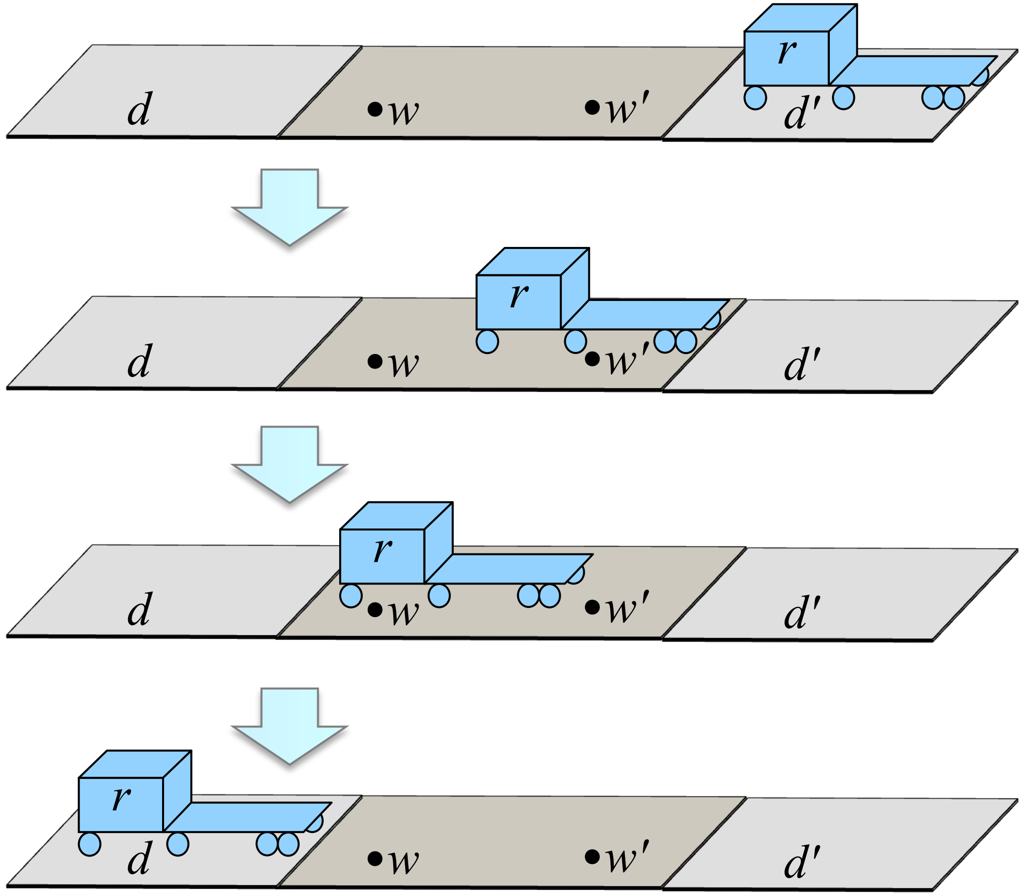
  assertions: ← *need causal establishment*

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

  constraints: ← *like C*

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





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 ← *like partial plans in PSP*
  - a planning problem
  - a plan or partial plan

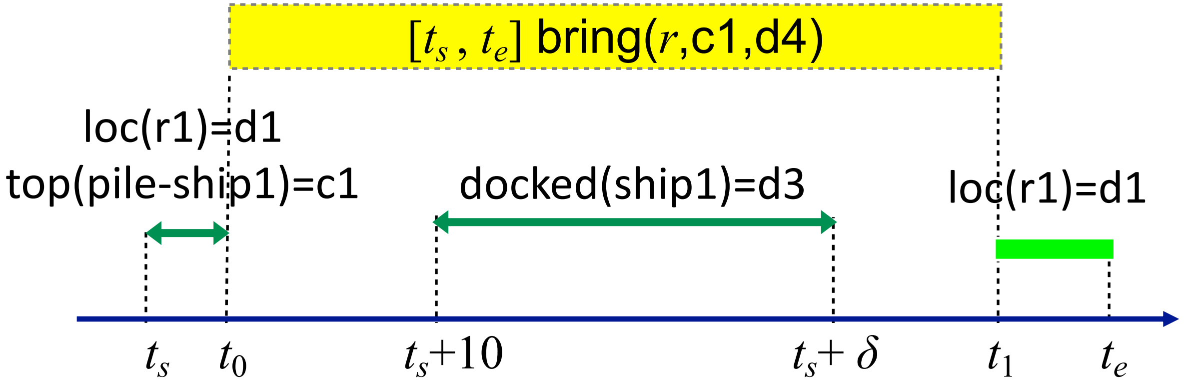
ϕ_0:

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

Topic	Section
• Introduction	17.1
• Representation	17.2
• **Planning (briefly)**	18.2
• Consistency and controllability	18.3
• Acting (Part 1: refinement)	17.3.1
• Acting (Part 2: dispatching)	17.3.1

Planning

- Planning problem: a chronicle ϕ_0 that has some *flaws*
  - Temporal assertions that aren't causally supported
    - like open goals in PSP
  - Temporal assertions that are (possibly) conflicting
    - like threats in PSP
  - Non-refined tasks
    - like tasks in HTN planning
- Resolvers
  - persistence assertions
  - constraints
  - actions
  - tasks
  - refinement methods

TemPLan(ϕ, Σ) :
while True **do**
  Flaws ← set of flaws of ϕ
  if *Flaws*=∅ **then** return ϕ
  arbitrarily select $f \in$ *Flaws*
  Resolvers ← set of resolvers of f
  if *Resolvers*=∅ **then** return failure
  nondeterministically choose $\rho \in$ *Resolvers*
  $\phi \leftarrow$ Transform(ϕ, ρ)

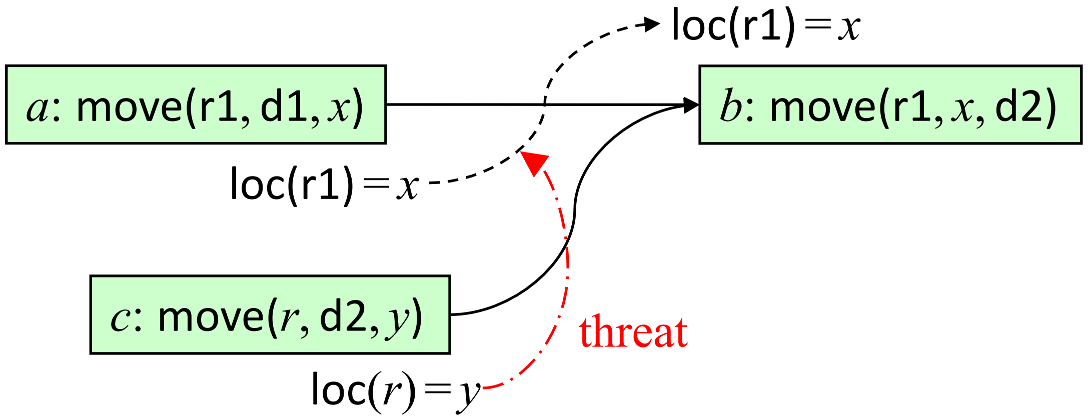
- If it's possible to resolve all flaws, then at least one of the nondeterministic execution traces will do so
- The details are intricate and tedious
  - If this interests you, I can point you to some good references

Outline

Topic	Section
• Introduction	17.1
• Representation	17.2
• Planning (briefly)	18.2
• **Consistency and controllability**	18.3
• Acting (Part 1: refinement)	17.3.1
• Acting (Part 2: dispatching)	17.3.1

Consistency of Constraints

- When TemPlan applies resolvers, it modifies $\phi = (\mathcal{A},\mathcal{S},\mathcal{T},\mathcal{C})$
  - Some resolvers will make ϕ inconsistent
    - No solution in this part of the search space
  - Would like to detect inconsistency, prune that part of the search space
    - Otherwise we'll waste time searching it



- Analogy: PSP checks simple cases of inconsistency
  - E.g., if there's already a constraint $c \prec b$,
    don't resolve a threat by adding a constraint $b \prec c$

- But PSP ignores more complicated cases
  - Suppose Range(c) = *Containers* = {c1, c2, c3}
  - To resolve three different threats, suppose PSP chooses
    $c \neq$ c1 , $c \neq$ c2 , $c \neq$ c3
    - No solutions in this part of the search
      space, but PSP searches it anyway

Consistency of Constraints

- $\phi = (\mathcal{A},\mathcal{S},\mathcal{T},\mathcal{C})$
- At various points, have TemPlan check whether C is consistent
  - If it isn't, then ϕ isn't either
  - Can prune this part of the search space

- Doesn't detect every possible inconsistency
  - If C is consistent, ϕ still may have other inconsistencies
- But if TemPlan can detect some of the inconsistencies, it may prune large parts of the search space

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

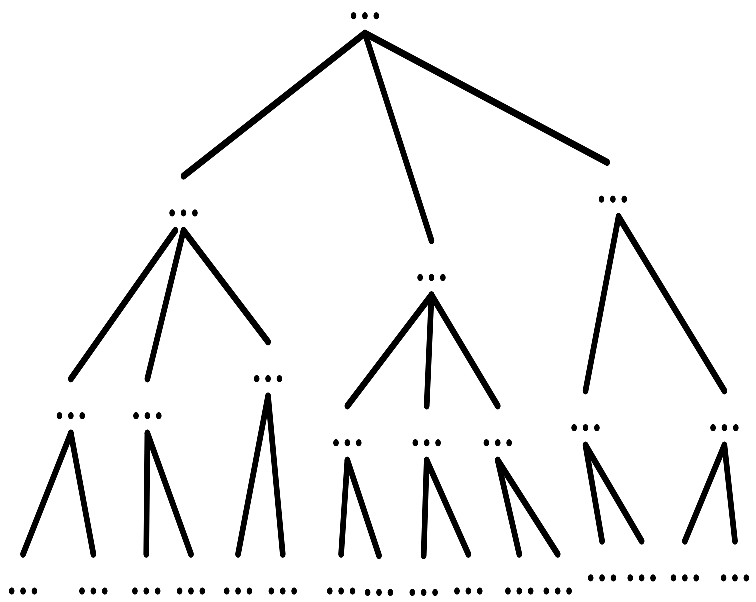
- Assume the two kinds of constraints are independent
  - exclude things like $t = \text{distance}(l,l') / \text{speed}(r)$

- Then two separate subproblems
  - (1) are the object constraints consistent?
  - (2) are the temporal constraints consistent?
- C is consistent iff both are consistent

(1) Object Constraints

- Constraint-satisfaction problem (CSP): NP-hard
- Can write a CSP algorithm that's *complete* but runs in exponential time
  - If there's an inconsistency, always finds it
  - Might enable a lot of pruning
  - But the calls to the CSP algorithm will take lots of time

- 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*

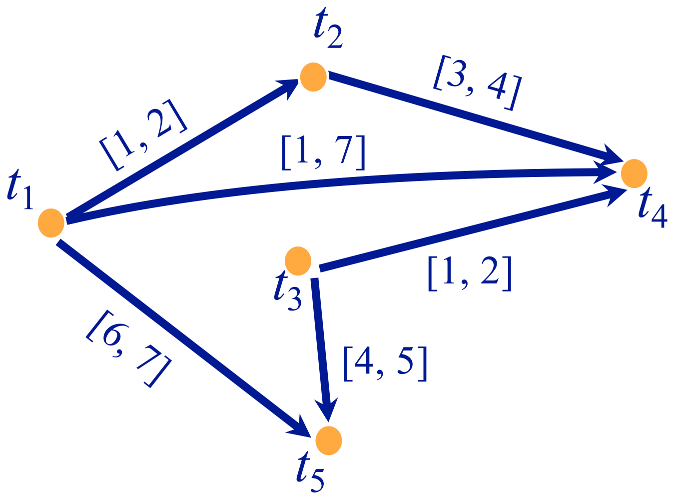
(2) Time Constraints

To represent time constraints:

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

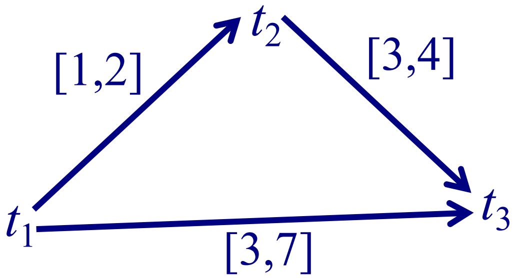
- Synthesize incrementally them starting from ϕ_0
  - can check time constraints in time $O(n^3)$

- Instantiate them incrementally during acting
- Keep them consistent throughout planning and acting

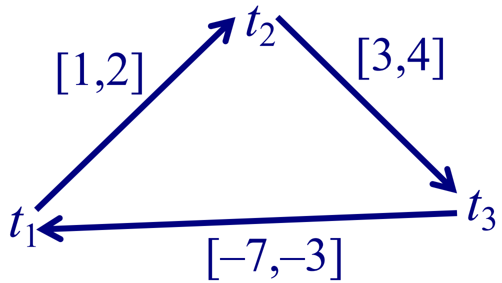


Time Constraints

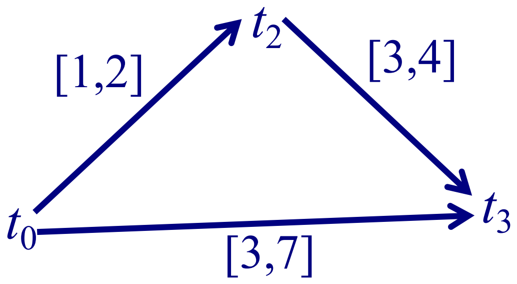
- *Simple Temporal Network* (STN):
- a pair $(\mathcal{V},\mathcal{E})$, where
  - $\mathcal{V}$ = {a set of temporal variables $\{t_1, \ldots, t_n\}$
  - $\mathcal{E} \subseteq \mathcal{V}^2$ is a set of arcs



- Each arc (t_i,t_j) is labeled with an interval $r_{ij} = [a,b]$
  - Represents constraint $a_{ij} \le t_j - t_i \le b_{ij}$
  - Sometimes written: $t_j - t_i \in [a, b]$
  - Or equivalently, $t_i - t_j \in [-b, -a]$



- To represent unary constraints:
  - Constraint of the form: $a \le t_j \le b$
  - Where a “dummy” variable $t_0 \equiv 0$
  - Then, arc $r_{0j} = [a,b]$ represents $t_j - 0 \in [a,b]$



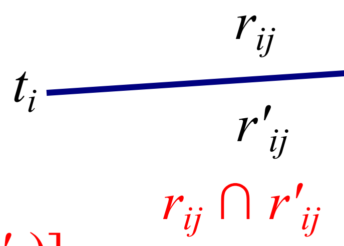
Operations on STNs

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

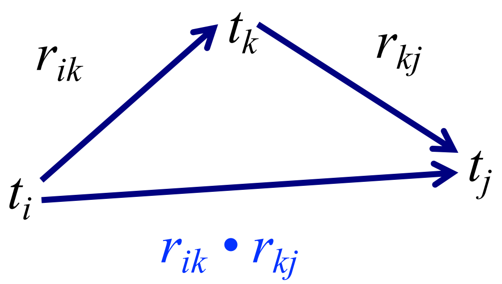


- Composition, $\bullet$

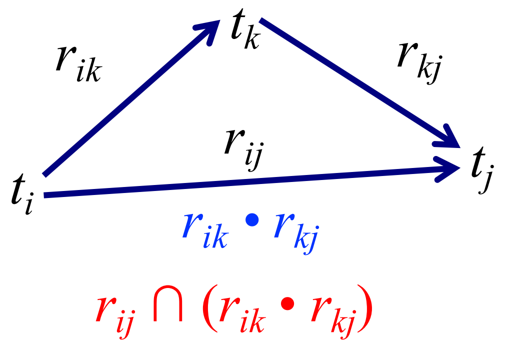
$$t_k - t_i \in r_{ik} = [a_{ik}, b_{ik}]$$

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

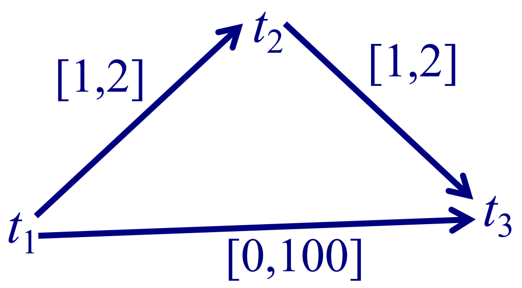
  - Infer $t_j - t_i \in r_{ik} \bullet r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}]$
  - Reason: shortest and longest times for the two intervals



- Consistency checking
  - r_{ik}, r_{kj}, r_{ij} are consistent iff $r_{ij} \cap (r_{ik} \bullet r_{kj}) \neq \emptyset$



Time Constraints



- *Solution* to an STN:
  - any assignment of integer values to the time points $\{ t_1, t_2, \ldots, t_n\}$ such that all the constraints are satisfied
- *Consistent* STN: has a solution

- Solutions:

$t_2 - t_1$	$t_3 - t_2$	$t_3 - t_1$
1	1	2
1	2	3
2	1	3
2	2	4

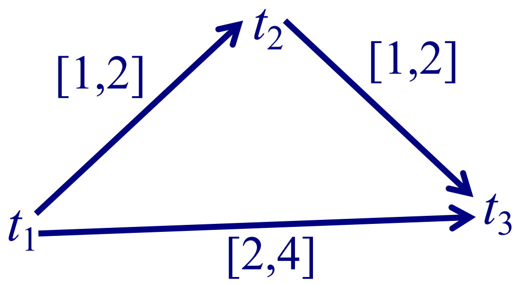
- *Minimal* STN:

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

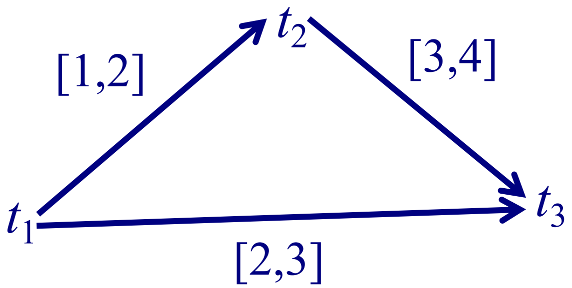
  for every $t \in [a,b]$,

  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

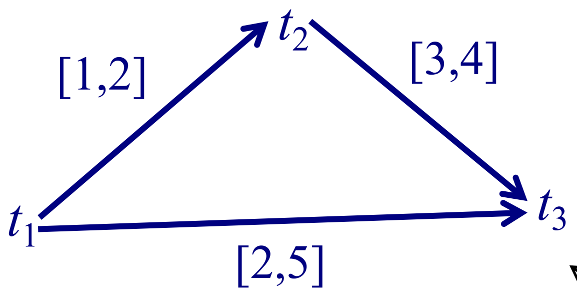


Two Examples



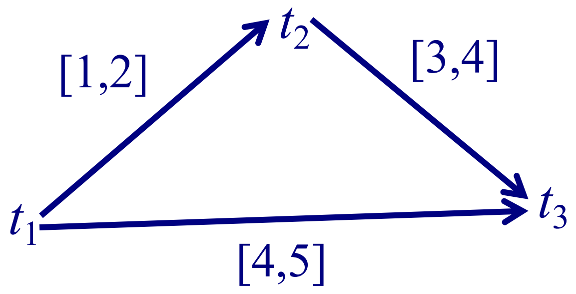
- $\mathcal{V} = \{t_1, t_2, t_3\}$
- $\mathcal{E} = \{r_{12}{=}[1,2],\ r_{23}{=}[3,4],\ r_{13}{=}[2,3]\}$

- Composition:
  - $r'_{13} = r_{12} \bullet r_{23} = [1{+}3, 2{+}4] = [4,6]$
- Thus
  - $r_{13} \cap r'_{13} = [2,3] \cap [4,6] = \emptyset$
- Can't satisfy both r_{13} and r'_{13}
- $(\mathcal{V},\mathcal{E})$ is inconsistent



- $\mathcal{V} = \{t_1, t_2, t_3\}$
- $\mathcal{E} = \{r_{12}{=}[1,2], r_{23}{=}[3,4], r_{13}{=}[2,5]\}$

- As before, $r'_{13} = r_{12} \bullet r_{23} = [4,6]$
  - $r_{13} \cap r'_{13} = [2,5] \cap [4,6] = [4,5]$
- $(\mathcal{V},\mathcal{E})$ is consistent
  - $r_{13} \leftarrow [4,5]$ will make it minimal



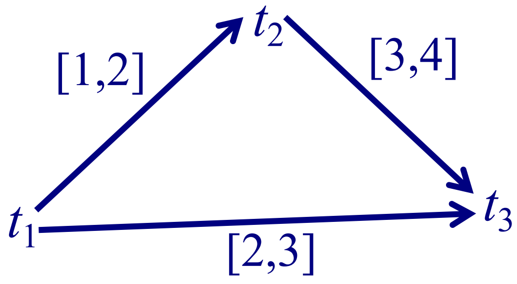
Same set of solutions:

$t_2 - t_1$	$t_3 - t_2$	$t_3 - t_1$
1	3	4
1	4	5
2	3	5
~~2~~	~~4~~	~~6~~

Time Constraints

- *Solution* to an STN:
  - any assignment of integer values to the time points $\{t_1, t_2, \ldots, t_n\}$ such that all the constraints are satisfied
- *Consistent* STN: has a solution

Poll: Is this network consistent?



- *Minimal* STN:

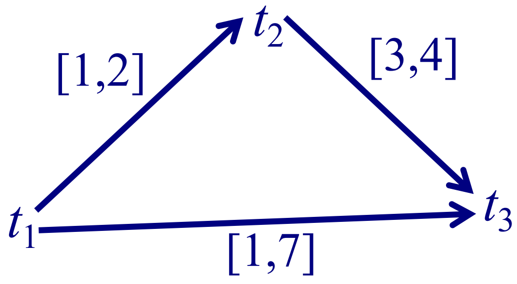
  for every arc (t_i, t_j) with label $[a,b]$,

  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

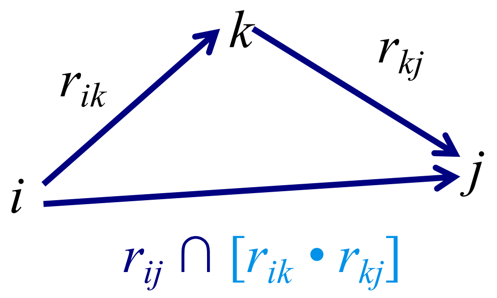
Poll: Is this network minimal?



Path Consistency

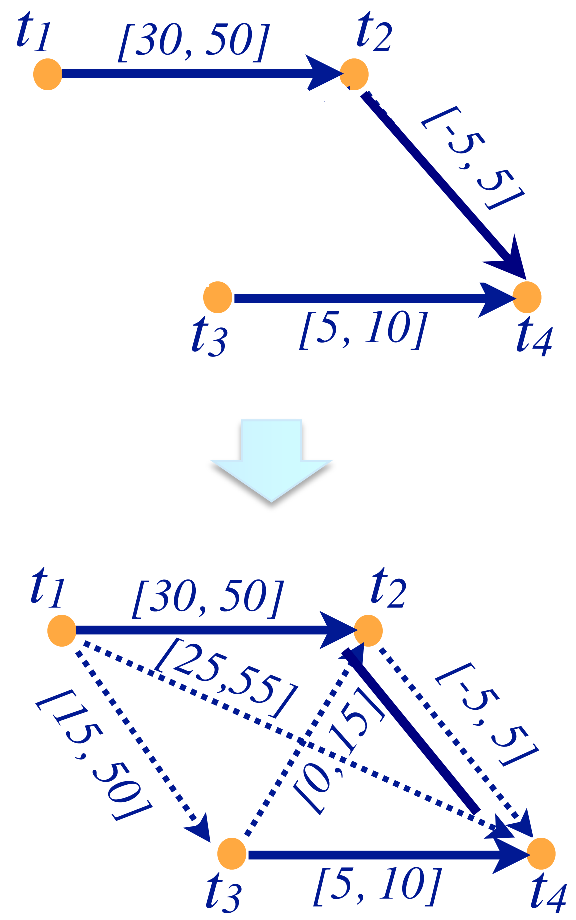
PC($\mathcal{V}$,$\mathcal{E}$):
  for $1 \le k \le n$ do
    for $1 \le i < j \le n,\ i \ne k,\ j \ne k$ do
      $r_{ij} \leftarrow r_{ij} \cap [r_{ik} \bullet r_{kj}]$
      if $r_{ij} = \emptyset$ then
        return inconsistent

- PC (*Path Consistency*) algorithm
- Iterate over each combination of *k, i, j*



- If an arc has no constraint, use $[-\infty, +\infty]$

- Makes network minimal
  - Reduce each r_{ij} to exclude values that aren't in any solution
- Detects inconsistent networks
  - inconsistent if r_{ij} shrinks to Ø
- *i, j, k* each go ≈ from 1 to *n*
  - $O(n^3)$ triples
  - total time $O(n^3)$



- Dashed lines: constraints shrunk from $[-\infty, \infty]$

Pruning TemPlan's search space

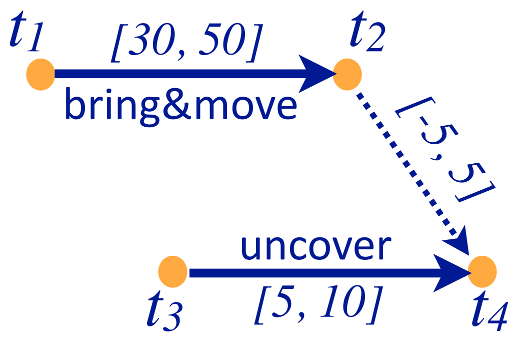
- Take the time constraints in C
  - Write them as an STN
  - Use Path Consistency to check whether STN is consistent
  - If it's inconsistent, TemPlan can backtrack

- TemPlan needs to add new constraints incrementally
  - Can modify PC to make it incremental
  - Given a consistent, minimal STN,
    incorporate a new constraint r'_{ij}
    - time $O(n^2)$

Controllability

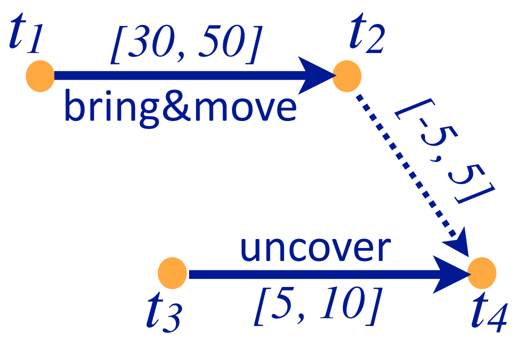
- Section 18.3.3 of the book
- Suppose TemPlan gives you a chronicle and you want to execute it
  - Constraints on time points
  - Need to reason about these in order to decide when to start each action

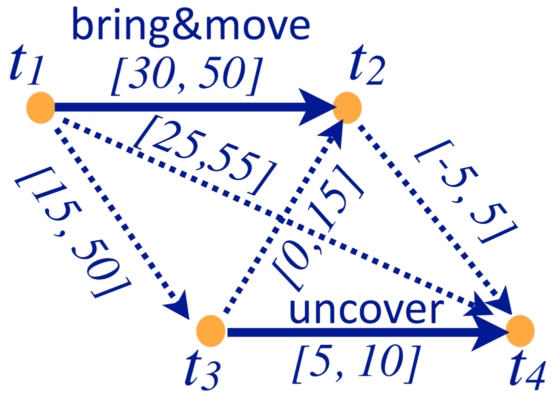
- Solid lines: duration constraints
  - Robot will do bring&move, will take 30 to 50 time units
  - Crane will do uncover, will take 5 to 10 time units
- Dashed line: synchronization constraint
  - At most 5 seconds between the two ending times



- Objective
  - Choose starting times that will satisfy the constraints

Controllability

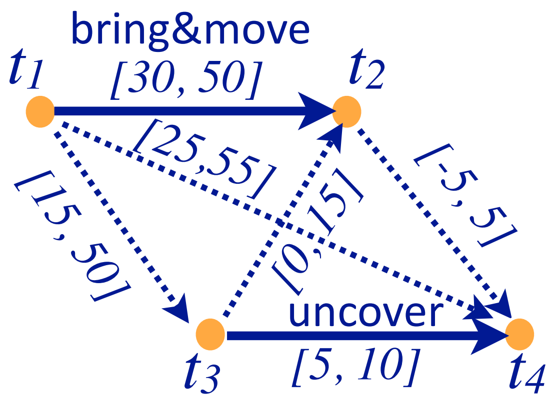




- Suppose we run PC
  - Returns a minimal and consistent network
- There *exist* time points that satisfy all the constraints
- Would work if we could choose all four time points
  - But we can't choose t_2 and t_4

- 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
    - $t_2 \in [t_1+30, t_1+50]$
    - $t_4 \in [t_3+5, t_3+10]$
- Want to choose t_1, t_3 that will work for every t_2, t_4

Controllability



- Start bring&move at time $t_1 = 0$
- Let d_b = duration of bring&move
  - Then $t_2 = d_b$
- Start uncover at time t_3
- Let d_u = duration of uncover
  - Then $t_4 = t_3 + d_u$
- r_{24}: $\quad -5 \leq t_4 - t_2 \leq 5$

$$-5 \leq t_3 + d_u - d_b \leq 5$$

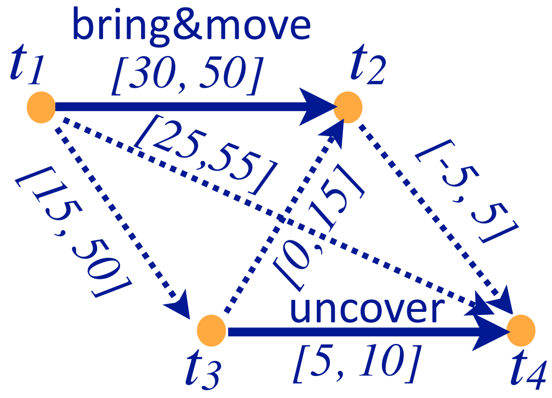
$$-5 + (d_b - d_u) \leq t_3 \leq 5 + (d_b - d_u)$$

- Suppose the durations are
  - bring&move 50
  - uncover 5
  - Then $d_b - d_u = 45$
    - $40 \leq t_3 \leq 50$
- Suppose the durations are
  - bring&move 30
  - uncover 10
  - Then $d_b - d_u = 20$
    - $15 \leq t_3 \leq 25$
- There's no t_3 that works in both cases

STNUs

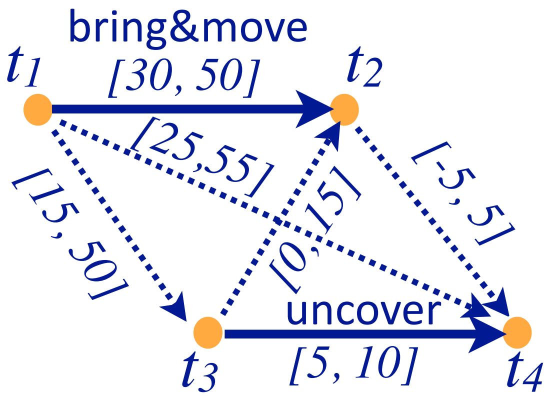
- *STNU (Simple Temporal Network with Uncertainty):*
  - A 4-tuple $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{E})$
    - $\mathcal{V}$ ={*controllable* time points}, e.g., starting times of actions
    - $\tilde{\mathcal{V}}$ ={*contingent* time points}, e.g., ending times of actions
    - $\mathcal{E}$ ={*controllable* constraints},
    - $\tilde{E}$ ={*contingent* constraints},
  - Synchronization between starting times of two actions: *controllable*
  - Synchronization between ending times of two actions: *contingent*
  - Synchronization between end of action a_1 and start of action a_2
    - If a_2 starts after a_1 ends, *controllable*
    - If a_2 starts before a_1 ends, *contingent*

- 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



- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is *strongly controllable* if the actor can choose values for $\mathcal{V}$ that satisfy $\mathcal{E}$, such that success occurs for all values of $\tilde{\mathcal{V}}$ that satisfy $\tilde{\mathcal{E}}$
  - Actor can choose the values for $\mathcal{V}$ offline
  - The right choice works 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 $\mathcal{V}$ that satisfy $\mathcal{E}$, such that success occurs for *at least one* combination of values for $\tilde{\mathcal{V}}$ that satisfy $\tilde{\mathcal{E}}$
  - To make the right choice, the actor needs to know in advance what the values of $\tilde{\mathcal{V}}$ will be
- *Dynamic execution strategy*: procedure the actor calls at each time point t, to assign the value t to zero or more unassigned variables in $\mathcal{V}$.
  - Input: t and a list of previous assignments to some variables in $\mathcal{V}$ and $\tilde{\mathcal{V}}$. Previous assignments will always be values in $[0, t\text{–}1]$ that satisfy $\mathcal{E}$ and $\tilde{\mathcal{E}}$.
- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is *dynamically controllable* if there exists a dynamic execution strategy for it that can guarantee that the constraints in $\mathcal{E}$ are satisfied.

Poll. Is the above STNU strongly controllable?

Poll. Is it weakly controllable?

Poll. Is it dynamically controllable?

Game-Theoretic Model

- Can model dynamic execution as a zero-sum game between actor and environment

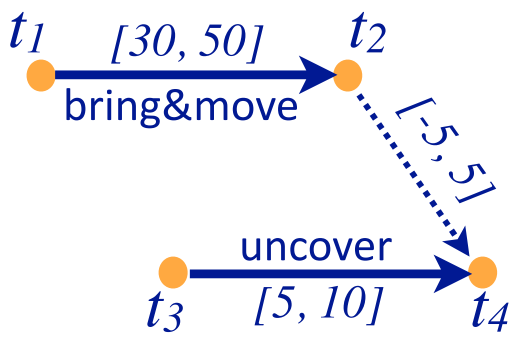
  For $t = 0, 1, 2, \ldots$

  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

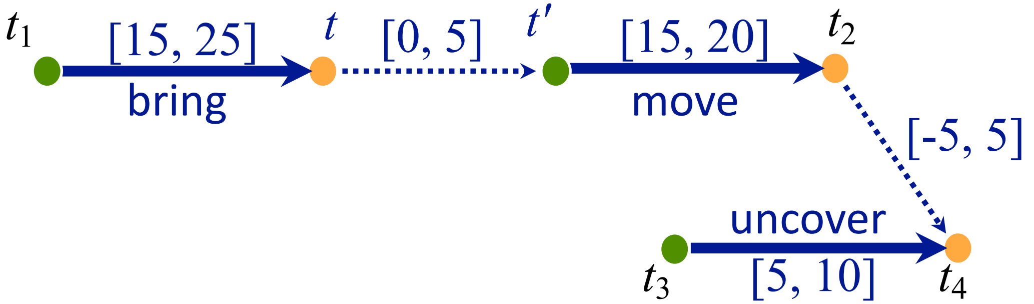
$r_{ij} = [l,u]$ is *violated* if t_i and t_j have values such that $t_j - t_i \notin [l,u]$

- *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 $t_3 = t' + 10$
  - $t_2 \in [t' + 15, t' + 20]$
  - $t_4 \in [t_3 + 5, t_3 + 10] = [t' + 15, t' + 20]$
    - $t_4 - t_2 \leq$ max value for t_4 – min value for t_2
      $= (t' + 20) - (t' + 15) = 5$
    - $t_4 - t_2 \geq$ min value for t_4 – max value for t_2
      $= (t' + 15) - (t' + 20) = -5$
    - so $t_4 - t_2 \in [-5, 5]$
  - The constraints are satisfied



$\mathcal{V} = \{t_1, t', t_3\}$

$\tilde{V} = \{t, t_2, t_4\}$

$\mathcal{E} = \{t' - t\}$

$\tilde{E} = \{t_4 - t_3\}$

Dynamic Controllability Checking

- For a chronicle $\phi = (\mathcal{A}, \mathcal{S}_\mathcal{T}, \mathcal{T}, \mathcal{C})$
  - 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
    - I'll skip the details

```
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
```

Outline

Topic	Section
• Introduction	17.1
• Representation	17.2
• Planning (briefly)	18.2
• Consistency and controllability	18.3
• **Acting (Part 1: refinement)**	**17.3.1**
• Acting (Part 2: dispatching)	17.3.1

Atemporal Refinement of Actions

- Templan's actions may correspond to compound tasks
  - In RAE, use refinement methods to refine them into commands

- Templan's action schema (descriptive model)

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

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

- RAE's refinement method (operational model)

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

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)

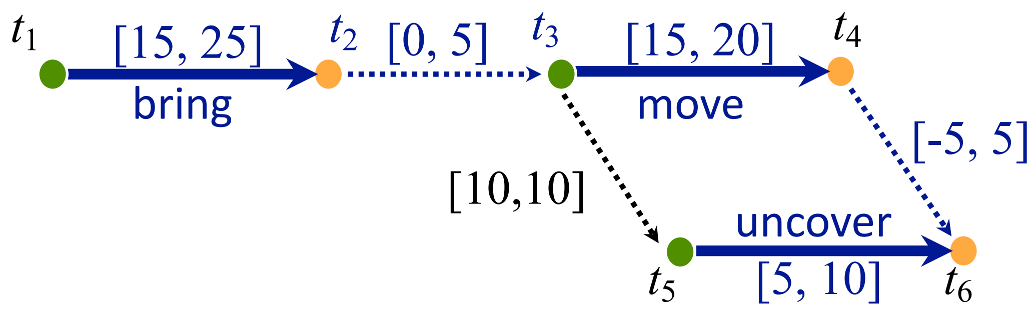
Discussion

- Atemporal Refinement of Actions
  - Advantages
    - Simple online refinement with RAE
    - Can be augmented to include some temporal monitoring functions in RAE
  - Disadvantages
    - Does not handle temporal requirements at the command level,
      - e.g., synchronize two robots that must act concurrently

Outline

Dispatching

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



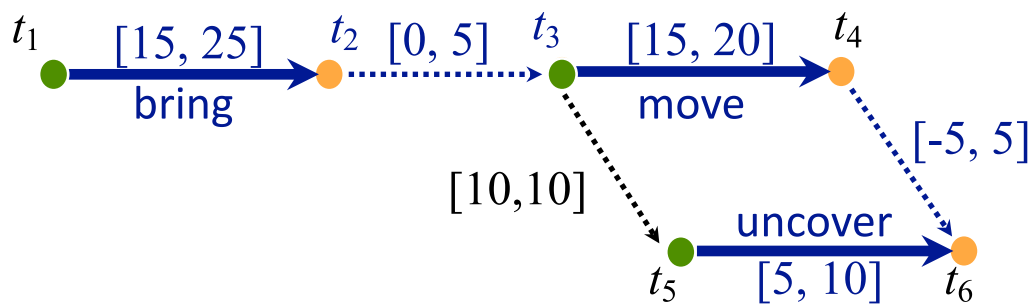
Dispatching

- Let $(\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}})$ be a *grounded* controllable STNU
- 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
  - every precedence constraint $t' < t_i$ has occurred
  - for every wait constraint $\langle t_e, \alpha \rangle$,
    - t_e has occurred or α has expired

- Let $t_1 = 0$. Then:
  - $t_2 \in [15,25]$
  - $t_3 \in [t_2, t_2+5]$
  - $t_4 \in [t_3+15, t_3+20]$
  - $t_5 \in [t_3+10, t_3+10]$
  - $t_6 \in [t_5+5, t_5+10] \cap [t_4-5, t_4+5]$
- Suppose bring finishes at t_2=20
  - t_3 is enabled during [20, 25]
- Suppose we start move at $t_3 = 22$
  - t_5 is enabled during [32,32]



Dispatching

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

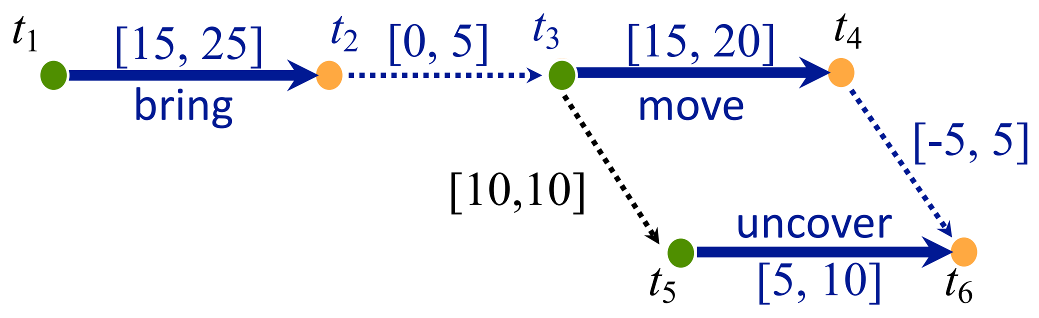
Controllable time point t in the future:

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

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
  1. update *now*
  2. update the time points in $\tilde{V}$ that were triggered since the last iteration
  3. update *enabled*
  4. trigger every $t_i \in$ *enabled* such that *now* $= u_i$
  5. arbitrarily choose other time points in *enabled*, and trigger them
  6. propagate values of triggered timepoints (change $[l_j,u_j]$ for each future timepoint t_j)

t_i is bounded by $[l_i , u_i]$



Example

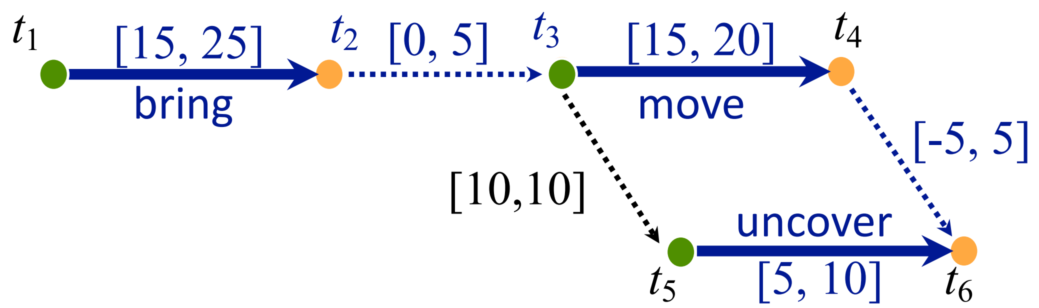
- Initially:

  $t_2 \in [t_1{+}15,\ t_1{+}25]$, $t_3 \in [t_2,\ t_2{+}5]$, $t_4 \in [t_3{+}15,\ t_3{+}20]$,

  $t_5 \in [t_3{+}10,\ t_3{+}10]$, $t_6 \in [t_5{+}5,\ t_5{+}10] \cap [t_4{-}5,\ t_4{+}5]$
- *now* = 0: trigger t_1
  - propagate $[l_j,u_j]$ values: $t_1 = 0$, $t_2 \in [15,25]$
- *now* = 20: bring finishes, update $t_2 \leftarrow 20$, add t_3 to *enabled*
  - propagate $[l_j,u_j]$ values:
    - $t_2 = 20$, $t_3 = [20, 25]$
- *now* = 22: trigger t_3, propagate $[l_j,u_j]$ values:
  - $t_3 = 22$, $t_4 \in [37, 42]$, $t_5 \in [32, 32]$
- *now* = 32: add t_5 to *enabled*; $now = u_5$ so we must trigger t_5
  - propagate values:
    - $t_5 = 32$, $t_6 \in [37, 42] \cap [t_4{-}5,\ t_4{+}5]$
- *now* = 37: move finishes, update $t_4 \leftarrow 37$
  - propagate values:
    - $t_4 = 37$, $t_6 \in [37, 42] \cap [32, 42]$
- *now* = 42: uncover finishes, update $t_6 \leftarrow 42$

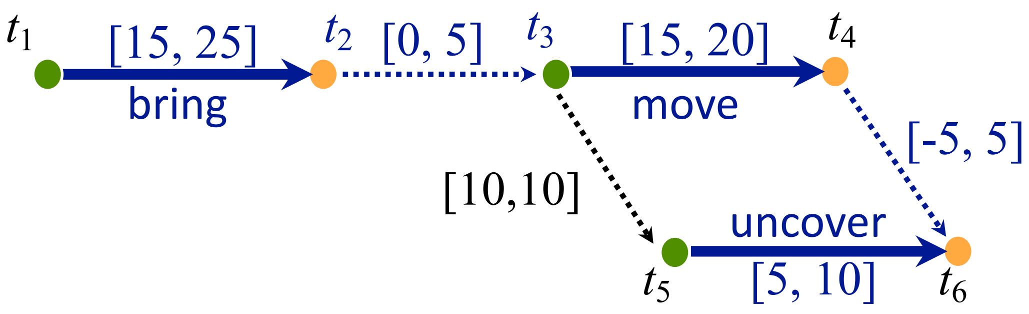
Dispatch($\mathcal{V},\tilde{V},\mathcal{E},\tilde{E}$)
- initialize the network
- while there are time points in $\mathcal{V}$ that haven't been triggered, do
  1. update *now*
  2. update the time points in $\tilde{V}$ that were triggered since the last iteration
  3. update *enabled*
  4. trigger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

t_i is bounded by $[l_i, u_i]$



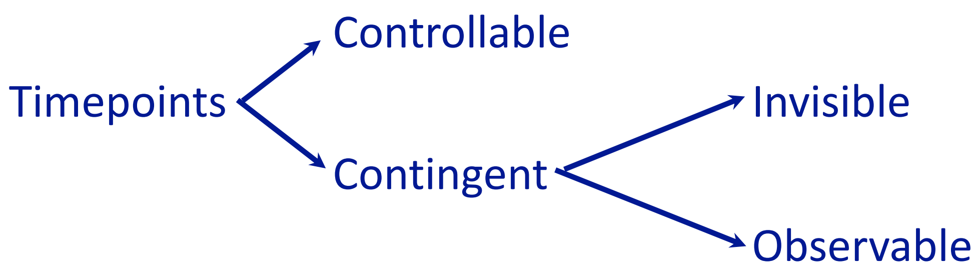
Deadline Failures

- 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 bring by delaying the whole plan
  - let the delayed action finish; try to repair the plan some other way



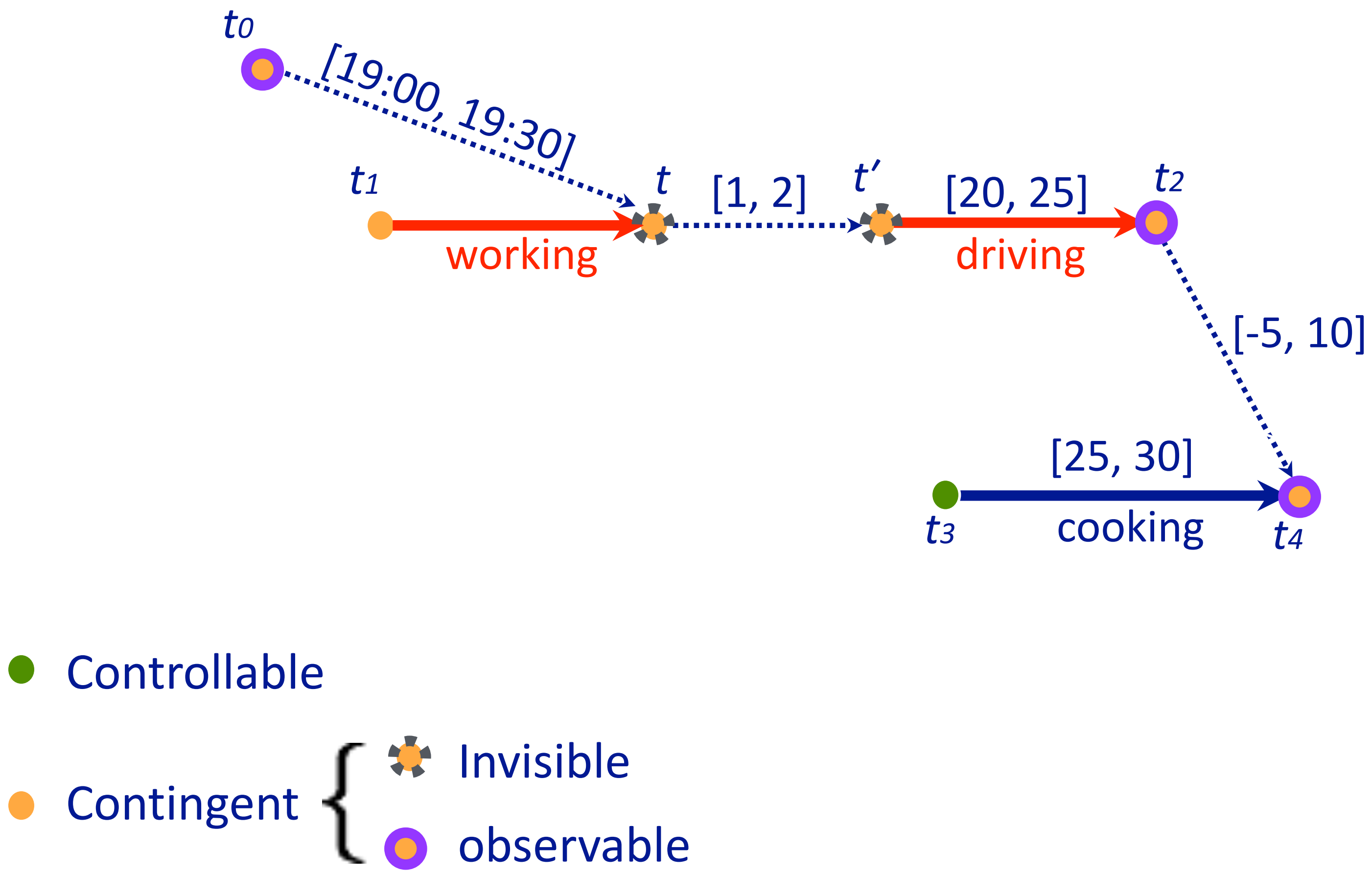
Partial Observability

- 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)



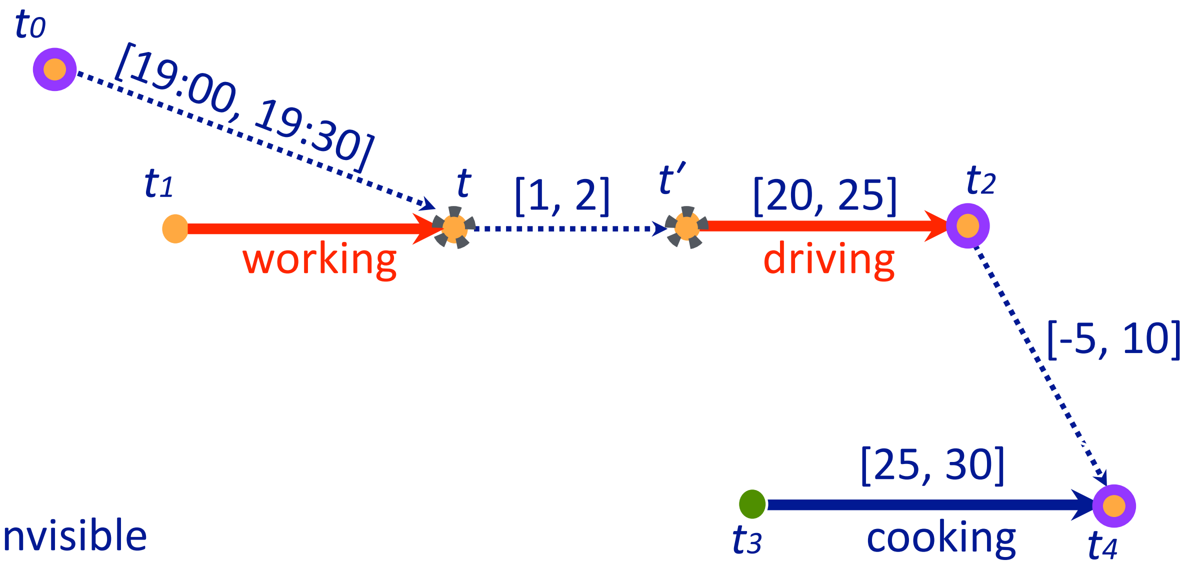
- Dynamically controllable?

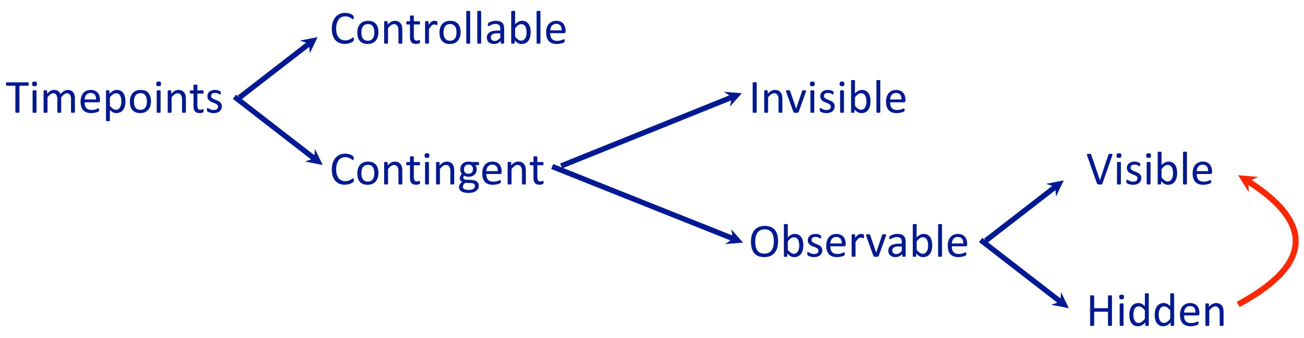
Observation Actions



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





Summary

- Representation
  - Time-oriented view
  - Timelines
    - Temporal assertions, object constraints, temporal constraints
  - Causal support
  - Action schemas, Methods
  - Chronicles
- Material from Chapter 18
  - Flaws, resolvers, TemPlan
  - Temporal constraints: STNs, PC algorithm (path consistency)
- Acting
  - Dynamic controllability
  - STNUs
  - RAE and eRAE
  - Dispatching