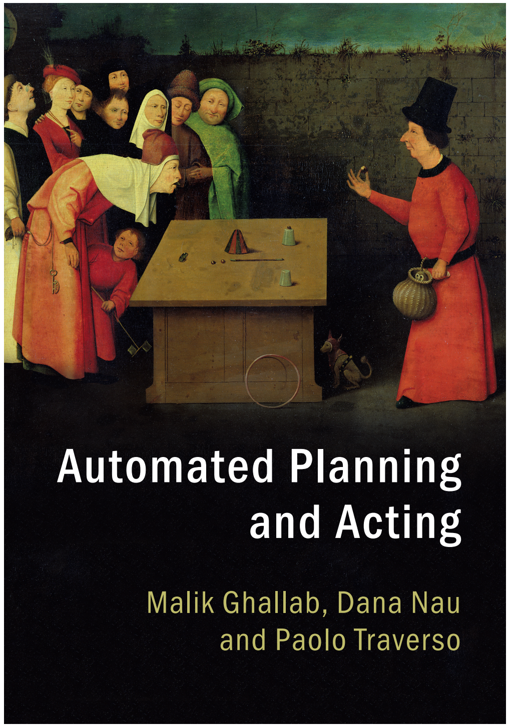
Last update: June 22, 2017

Deliberation in Planning and Acting

Part 3: Temporal Models



http://www.laas.fr/planning

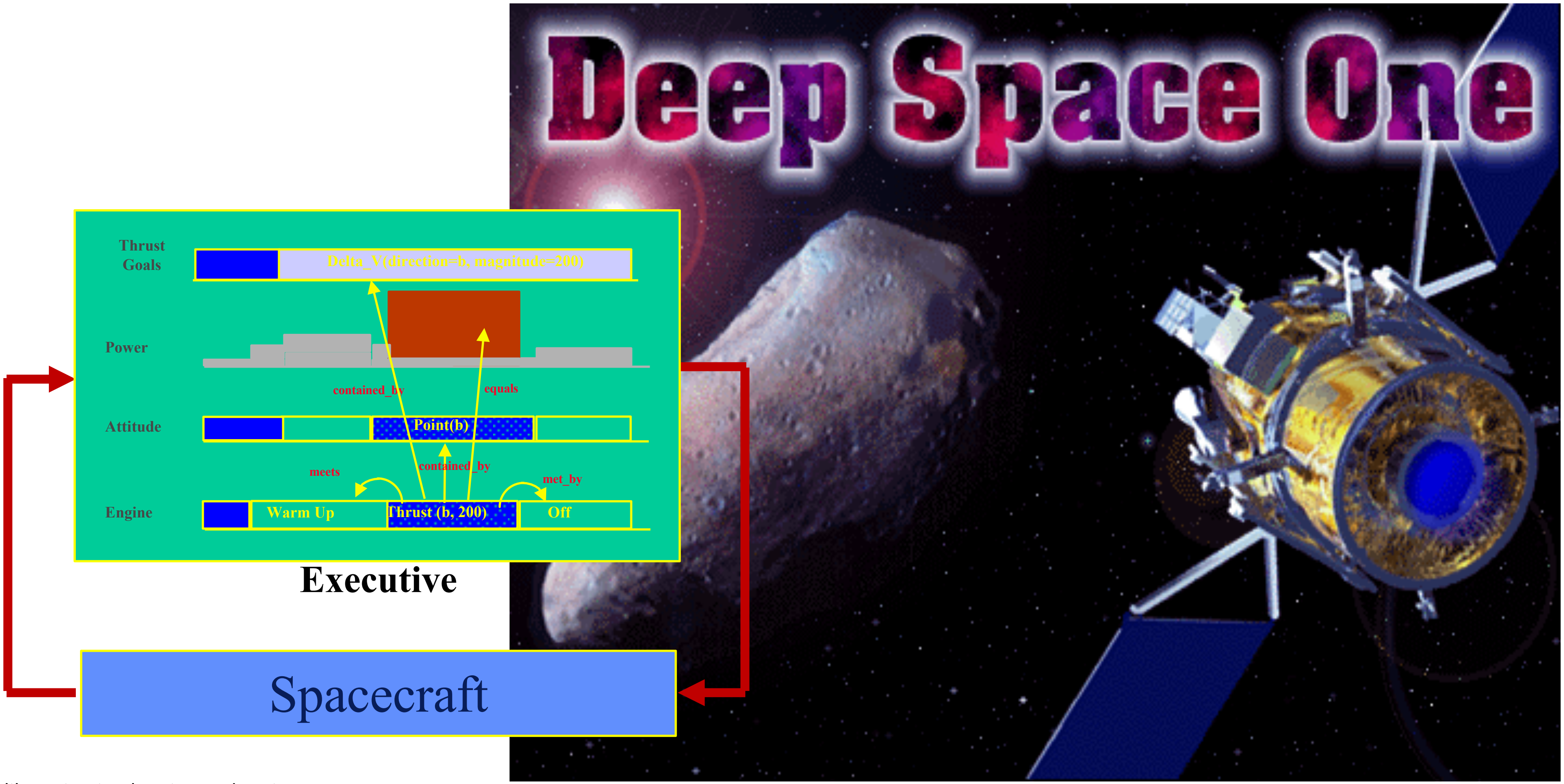
Malik Ghallab	LAAS/CNRS, University of Toulouse
Dana Nau	University of Maryland
Paolo Traverso	FBK ICT IRST, Trento, Italy

Motivation

- Some success stories …

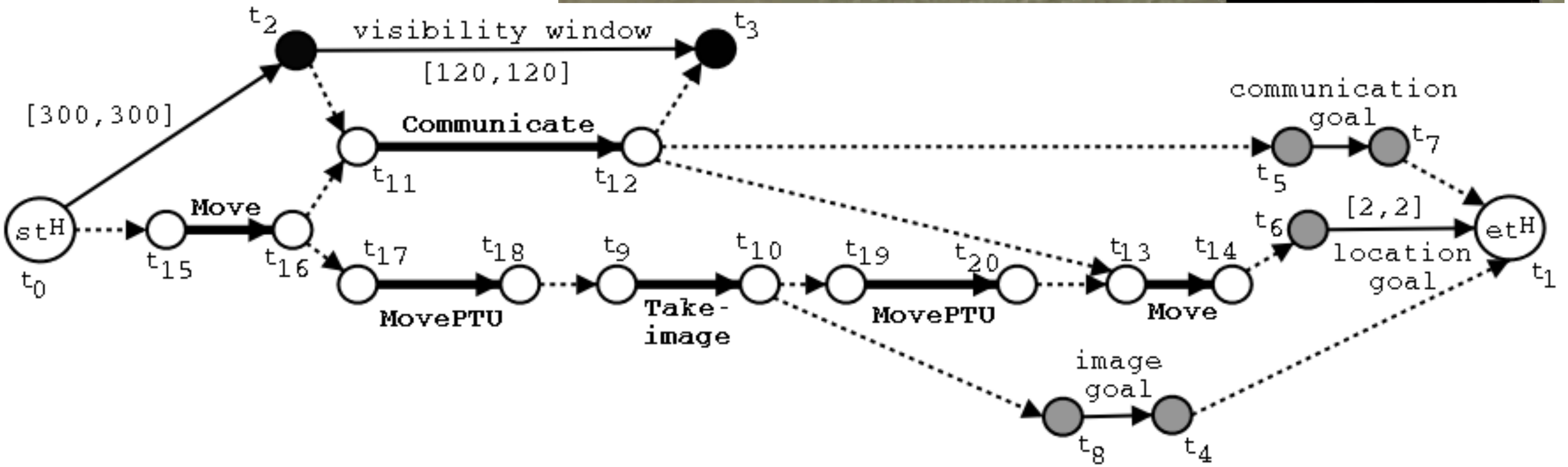
Remote Agent Experiment (RAX)

- First AI control system to control a spacecraft without human supervision
  - Deep Space One, 1999
- Key component: RAX-PS planner/scheduler



IxTeT

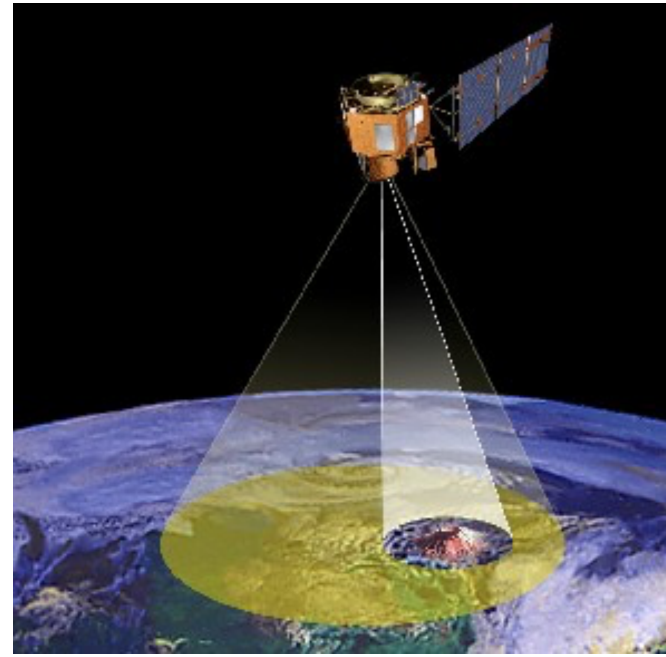
- LAAS/CNRS, Toulouse, France
- early 1990s to early 2000s



T-ReX

- MBARI, around 2005-2010

Casper (NASA JPL)



- NASA JPL, ongoing

- Common characteristic:
  explicit representation of time

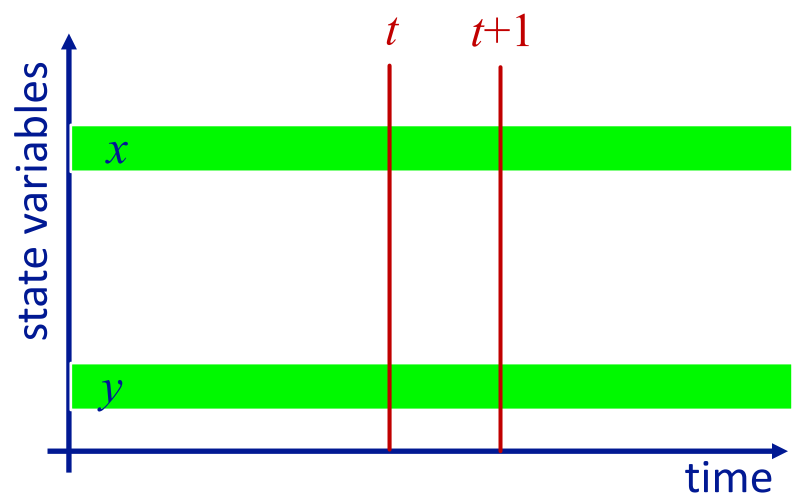
Temporal Models

- Durations of actions
- Delayed effects and preconditions
  - e.g., resources borrowed or consumed during an action
- Time constraints on goals
  - relative or absolute
- Exogenous events expected to occur in the future
  - when?
- Maintenance actions:
  - maintain a property (more general than just changing a value)
  - e.g., track a moving target, keep a spring latch in position
- Concurrent actions
  - interacting effects, joint effects
- Delayed commitment
  - instantiation at acting time

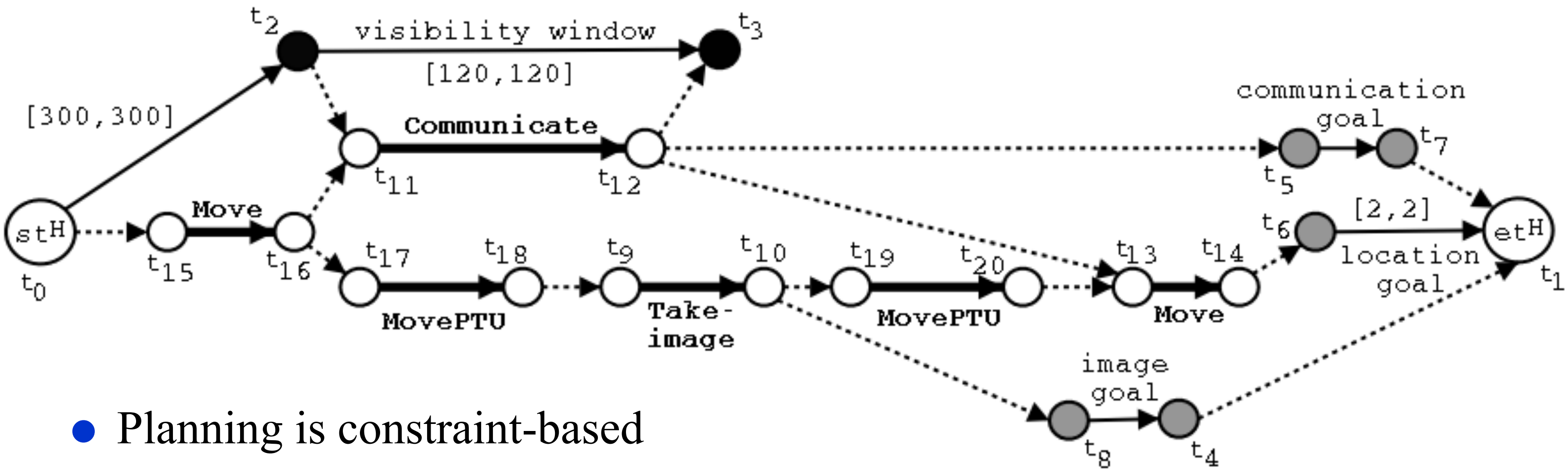
Timelines

- Up to now, we've used a "state-oriented view"
  - Sequence of states s_0, s_1, s_2
  - Instantaneous actions transform each state into the next one
  - No overlapping actions

- Switch to a "time-oriented view"
  - ➢ Sequence of integer *time points*
    - $t = 1, 2, 3, \ldots$
  - ➢ Scale is arbitrary
    - seconds, milliseconds, …
  - ➢ For each state variable, a *timeline*
    - Values during different time intervals
    - Reflect actions and events to occur at those times
  - ➢ State at time t = {values of the state-variable at time t}



Where We're Going



- Planning is constraint-based
  - ➢ Constrain values of state variables
    - at time points, over time intervals
- Somewhat like *plan-space planning* (partial-order causal link planning)
  - ➢ But with time durations, task refinement
- Can have a complete plan in which time points are constrained, but not completely fixed
  - ➢ Provide flexibility at acting time

Outline

- ✓ Introduction
- Representation
  - Timelines, separation constraints, causal support, actions, tasks, chronicles
- Temporal planning
- Speeding up TemPlan
- Controllability
- Acting with executable primitives
- Summary

Timeline

- A pair $(\mathcal{T},\mathcal{C})$
  - partially specifies evolution of a state variable
- $\mathcal{T}$: *temporal assertions*
  - *change*:

    $[t_1, t_2]$ loc(r1) : (loc1, l)

    $[t_3, t_4]$ loc(r1) : (l, loc2)
  - *persistence*:

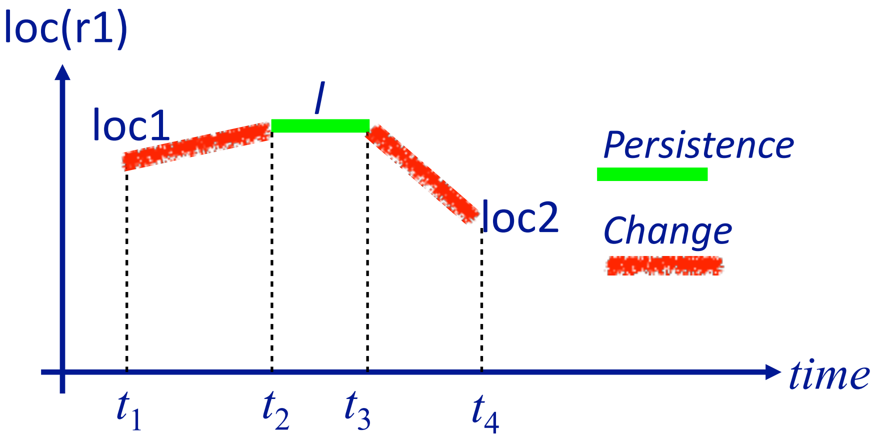
    $[t_2, t_3]$ loc(r1) $= l$
- $\mathcal{C}$: constraints
  - *time* constraints:

    $t_1 < t_2 < t_3 < t_4$

    $0 < t_4 - t_3 \leq c + 2$

    where c is a constant
  - *object* constraints:

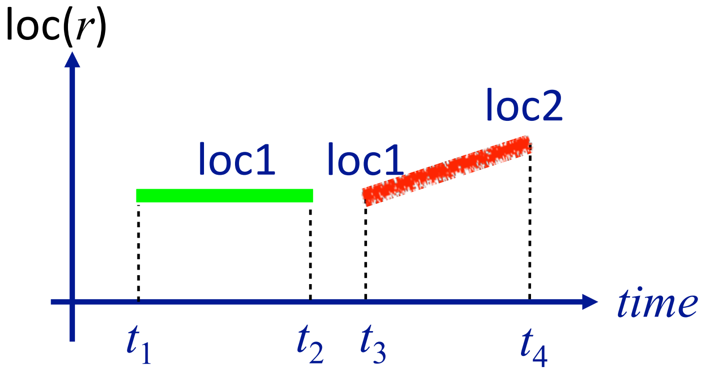
    $l \neq$ loc1, $l \neq$ loc2



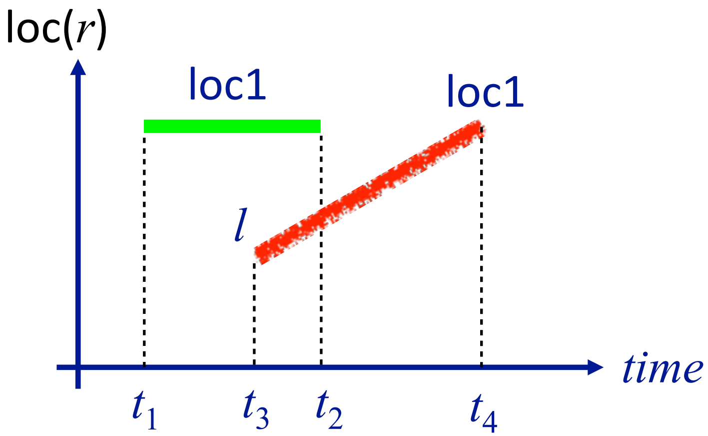
- Union of several timelines
  - $(\mathcal{T},\mathcal{C}) = (\mathcal{T}_1,\mathcal{C}_1) \cup \ldots \cup (\mathcal{T}_n,\mathcal{C}_n)$
    - $\mathcal{T} = \mathcal{T}_1 \cup \ldots \cup \mathcal{T}_n$
    - $\mathcal{C} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_n$

Consistency

- Let $(\mathcal{T}, \mathcal{C})$ be a timeline or union of several timelines
- 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 every state variable has at most one value at a time
- $(\mathcal{T}, \mathcal{C})$ is *consistent* if it has at least one consistent ground instance



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

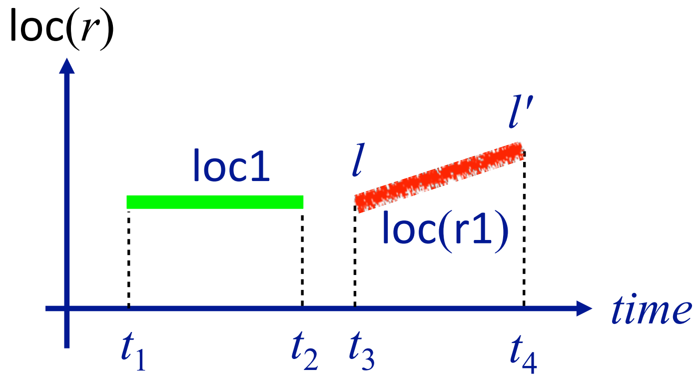


- Inconsistent:
  - $\mathcal{T} = \{[t_1,t_2]\ \text{loc}(r)\text{=loc1},$
    $[t_3,t_4]\ \text{loc}(r)\text{:}(l,\text{loc1})\}$
  - $\mathcal{C} = \{t_1 < t_3 < t_2,\ l \neq \text{loc1}\}$

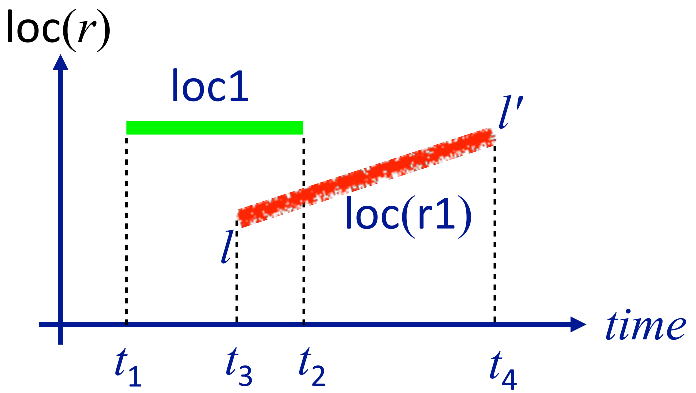
Security

- $(\mathcal{T}, \mathcal{C})$ – timeline or union of a set of timelines
  - $(\mathcal{T}, \mathcal{C})$ is *secure* if
    - it's consistent (i.e., at least one consistent ground instance)
    - every ground instance $(\mathcal{T}', \mathcal{C}')$ that satisfies $\mathcal{C}'$ is consistent

- Secure:
  - $\mathcal{T} = \{[t_1,t_2]$ loc(r1)=loc1,
    $[t_3,t_4]$ loc(r):$(l, l')\}$
  - $\mathcal{C} = \{t_1 < t_2 < t_3 < t_4\}$

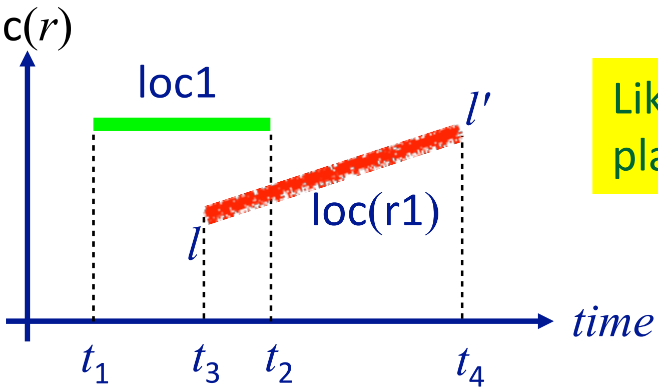


- Consistent but not secure:
  - $\mathcal{T} = \{[t_1,t_2]$ loc(r)=loc1,
    $[t_3,t_4]$ loc(r1):$(l,l')\}$
  - $\mathcal{C} = \{t_1 < t_2, t_3 < t_4\}$



Security

- Consistent but not secure:
  - $T = \{[t_1,t_2]$ loc(r)=loc1,
    $[t_3,t_4]$ loc(r1):$(l,l')\}$
  - $C = \{t_1 < t_2,\ t_3 < t_4\}$

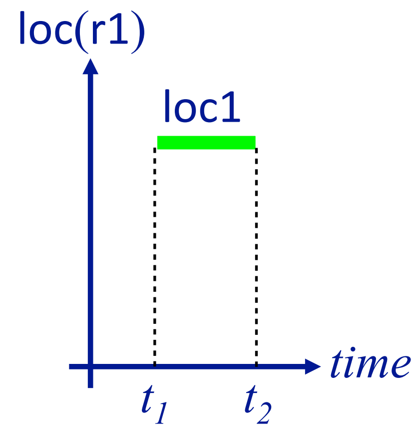


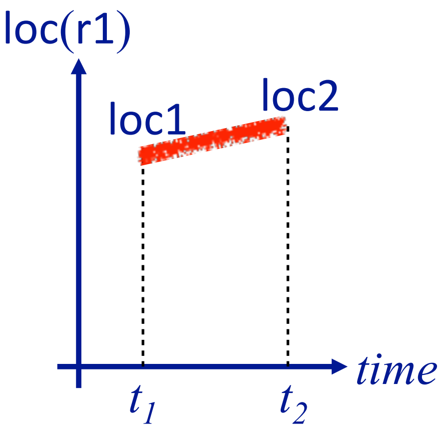
Like a *threat* in plan-space planning

- Can make it secure by adding a *separation constraint*
  - $r \neq$ r1
  - $t_2 < t_3$
  - $t_4 < t_1$
  - $t_2 = t_3,\ l =$ loc1
  - $t_4 = t_1,\ l' =$ loc1

Like a *resolver* in plan-space planning

Causal support





- Let α be one of these:
  - $[t_1,t_2]$ loc(r1)= loc1
  - $[t_1,t_2]$ loc(r1):(loc1, loc2)
- α says that at time t_1, r1 is at location loc1
  - How did it get there?

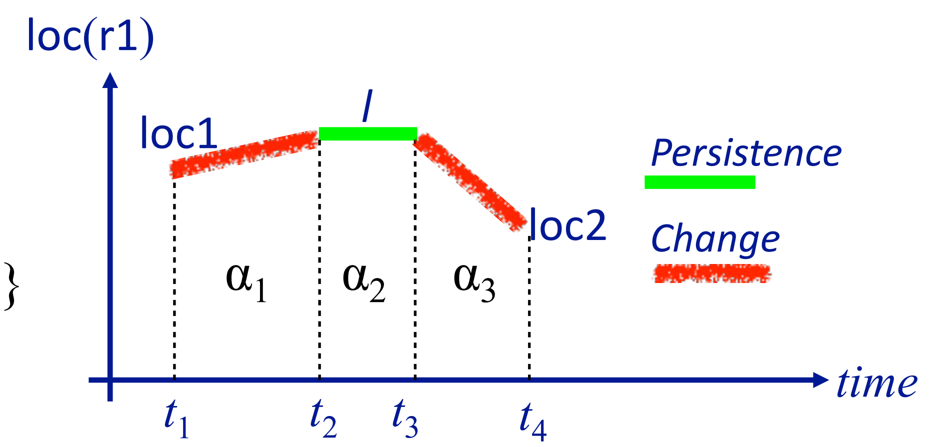
Like an *open goal* in plan-space planning

- *Causal support* for α
  - something that establishes loc(r1)= loc1 at time t_1
    - Another temporal assertion
      - $[t_0,t_1]$ loc(r1)= loc1
      - $[t_0,t_1]$ loc(r1):(l, loc1)
    - Or information telling us α is supported *a priori*

Like a *resolver* in plan-space planning

Causal support

- Timeline $(\mathcal{T}, \mathcal{C})$
  - $\mathcal{T} = \{\alpha_1, \alpha_2, \alpha_3\}$
    - $\alpha_1 = [t_1, t_2]$ loc(r1):(loc1,l)
    - $\alpha_2 = [t_2, t_3]$ loc(r1)=l
    - $\alpha_3 = [t_3, t_4]$ loc(r1):(l,loc2) }
  - $\mathcal{C} = \{ t_1 < t_2 < t_3 < t_4,$
    $l \neq$ loc1, $l \neq$ loc2$\}$



- α_3 is causally supported by α_2
- α_2 is causally supported by α_1
- No causal support for α_1

- $(\mathcal{T}, \mathcal{C})$ is *causally supported* if every assertion has a causal support
  - Three ways to add causal support for an assertion

In plan-space planning, like adding a resolver to a partial plan

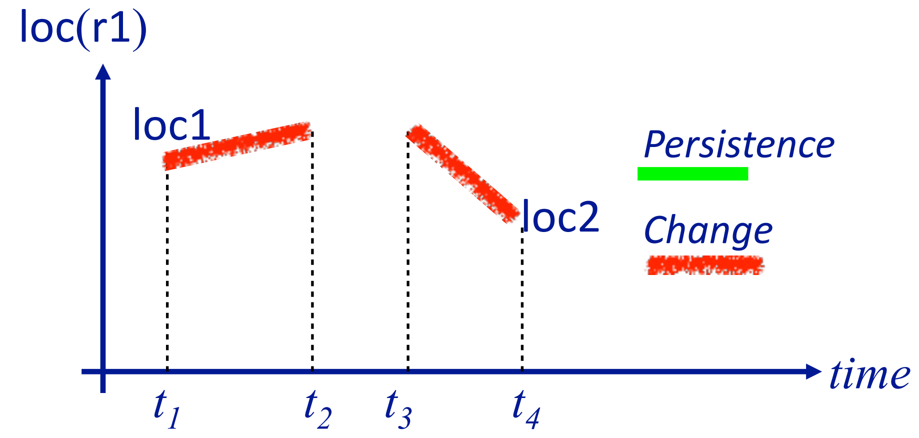
Establishing causal support

(1) Add a persistence assertion

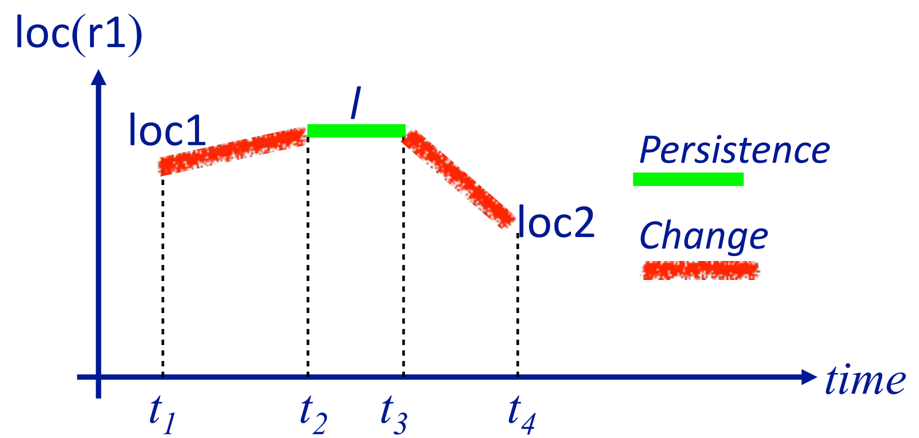
Timeline $(\mathcal{T},\mathcal{C})$

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

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



- Add $[t_3,t_4]$ loc(r1) = loc2
  - Supported by the first temporal assertion
  - Supports the second one



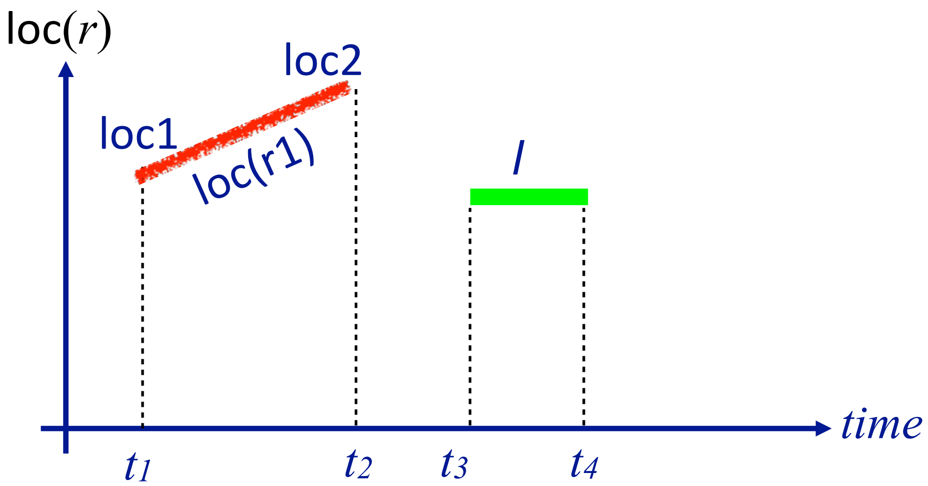
Establishing causal support

(2) Add constraints

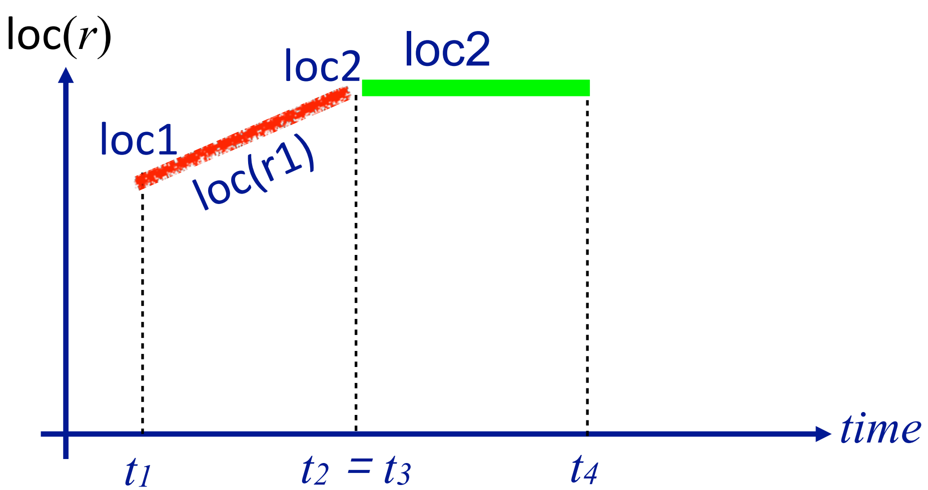
Timeline $(\mathcal{T}, \mathcal{C})$

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

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



- Add $t_2 = t_3$, $l = \text{loc2}$



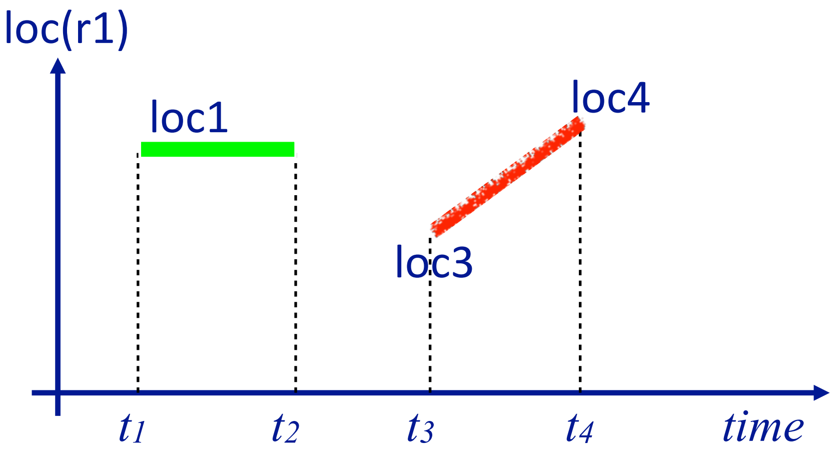
Establishing causal support

(3) Add a change assertion

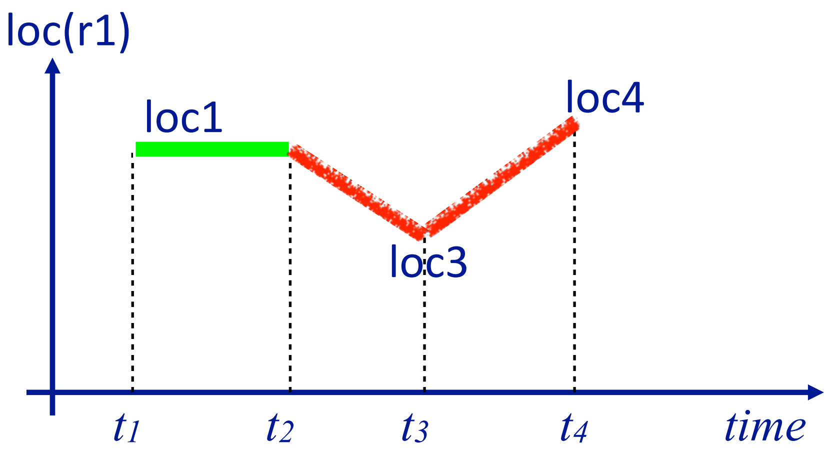
Timeline $(\mathcal{T}, \mathcal{C})$

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

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



- Add $[t_2,t_3]$ loc(r1):(loc1,loc3)
- *Caveat*:
  - Can't just do this directly
  - It needs to be part of an *action*



Actions

- *Action* or *primitive task*:
  - a triple $(head, \mathcal{T}, \mathcal{C})$
    - $head = name(arg_1, arg_2, \ldots, arg_n)$
    - $(\mathcal{T}, \mathcal{C})$ is the union of one or more timelines

- An action will always have two additional arguments
  - starting time t_s
  - ending time t_e

Actions

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

enter(r,d,w)

assertions:

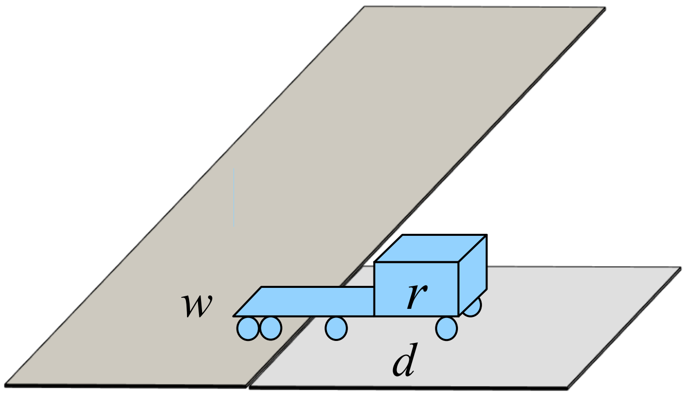
$[t_s,t_e]$ loc(r): (w,d)
$[t_s,t_e]$ occupant(d): (empty,r)

constraints:

$t_e \geq t_s+3$
adjacent(d,w)

- loc(r) changes to d
- dock d becomes occupied by r

- Things that need to be supported
- At time t_s
  - r at waypoint w
  - d empty



Actions

- leave(r,d,w)
  - robot r goes from dock d to adjacent waypoint w

leave(r,d,w)

assertions:

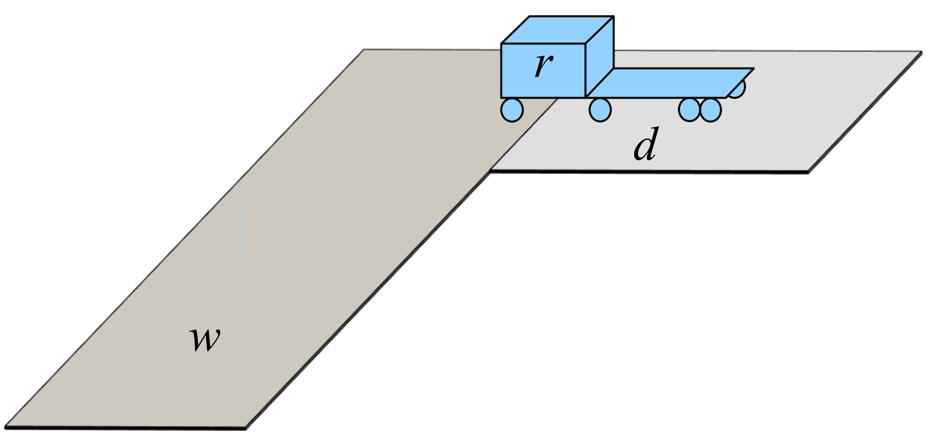
$[t_s,t_e]$ loc(r): (d,w)
$[t_s,t_e]$ occupant(d): $(r,$empty$)$

constraints:

$t_e \geq t_s+2$
adjacent(d,w)

- loc(r) changes to w
- dock d becomes empty

- Need causal support at time t_s
  - r at dock d
  - d occupied by r



Actions

- navigate(r,w,x)
  - robot r goes from waypoint w to connected waypoint x
- Need causal support at time t_s
  - r at waypoint w

navigate(r,w,x)

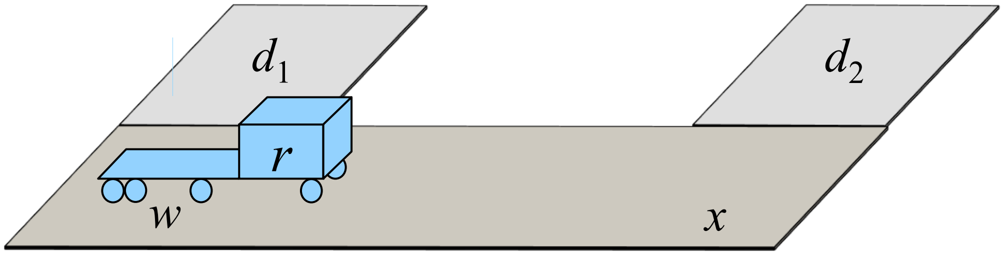
assertions:

[t_s,t_e] loc(r): (w,x)

constraints:

$t_s < t_e$
connected(d,w)

- loc(r) changes to x



Task and Method

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

- Refinement method:

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

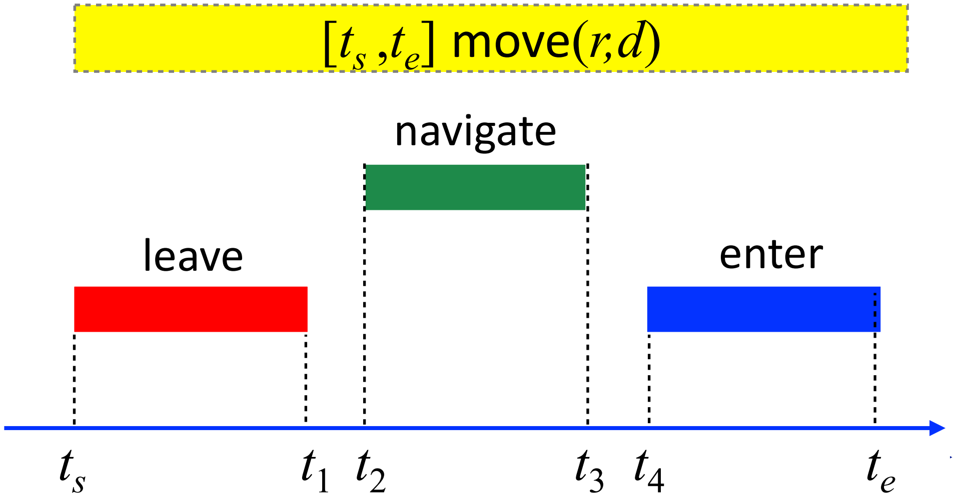
  task: move(r,d)

  refinement:

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

  constraints:

  $t_s < t_1,\ t_1 \leq t_2,\ t_2 < t_3,\ t_3 \leq t_4,\ t_4 < t_e$
  adjacent(d,w), adjacent(d',w')
  connected(w,w')



Chronicles

- Chronicle $\phi = (\mathcal{A}, S_{\mathcal{T}}, \mathcal{T}, \mathcal{C})$
  - $\mathcal{A}$: temporally qualified tasks
    - includes primitive tasks (actions)
  - $S_{\mathcal{T}}$: *a priori* supported assertions
  - $\mathcal{T}$: temporally qualified assertions
  - $\mathcal{C}$: constraints
- ϕ can include
  - Current state and future predicted events
  - Tasks to be performed
  - Assertions and constraints to be satisfied
- Can represent
  - a planning problem
  - partial or full solution

tasks:

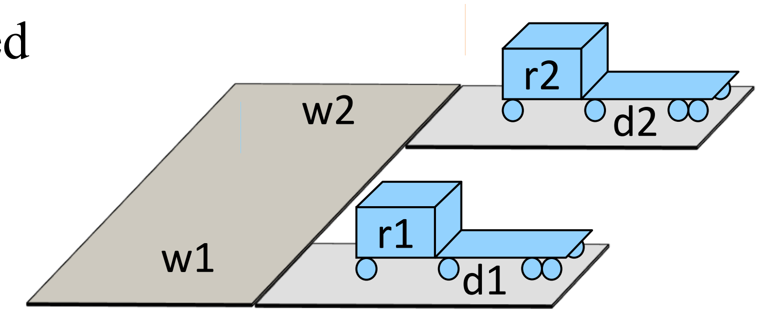
$[t_s, t_e]$ move(r1,d2)
$[t_s, t_e]$ move(r2,d1)

supported:

$[t_s]$ loc(r1)=d1
$[t_s]$ loc(r2)=d2

constraints:

$t_s < t_e$
adjacent(d1,w1),
adjacent(d2,w2),
connected(w1,w2)



Outline

- ✓ Introduction
- ✓ Representation
- Temporal planning
  - Algorithm, example, heuristics
- Speeding up TemPlan
- Controllability
- Acting with executable primitives
- Summary

Planning Algorithm

- Input is a chronicle
- Repeatedly
  - select a *flaw*
    - arbitrary choice
  - choose a *resolver*
    - nondeterministic choice
- Three kinds of flaws

  1. non-refined task

     Like a task in SeRPE

     - Resolver: apply refinement method or action definition

  2. unsupported assertion

     Like an open goal in plan-space planning

     - Resolver: add causal support

  3. possibly-conflicting assertions

     Like a threat in plan-space planning

     - Resolver: separation constraint

TemPlan(ϕ, Σ)
- $\mathit{Flaws} \leftarrow$ set of flaws of ϕ
- if $\mathit{Flaws}{=}\varnothing$ then return ϕ
- arbitrarily select $f \in \mathit{Flaws}$
- $\mathit{Resolvers} \leftarrow$ 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)

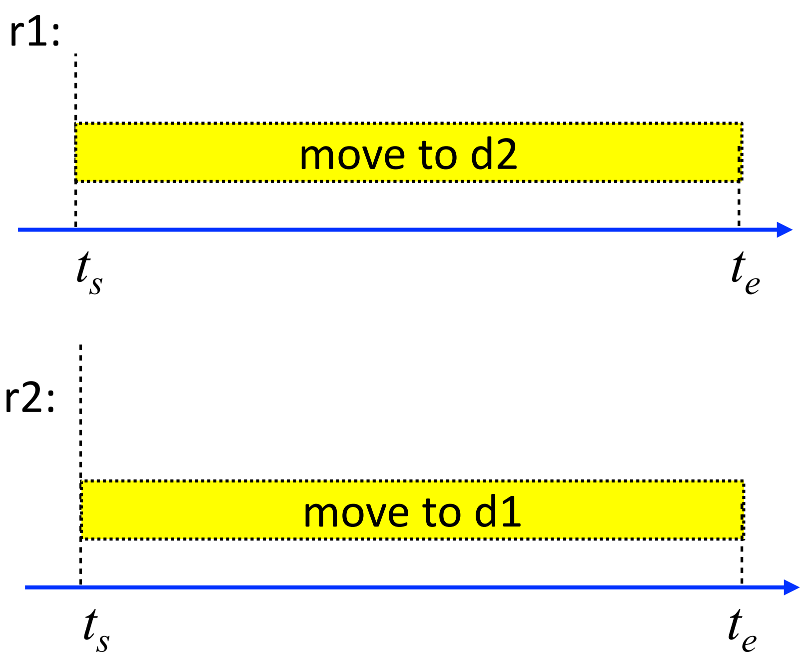
Initial Chronicle for a Planning Problem

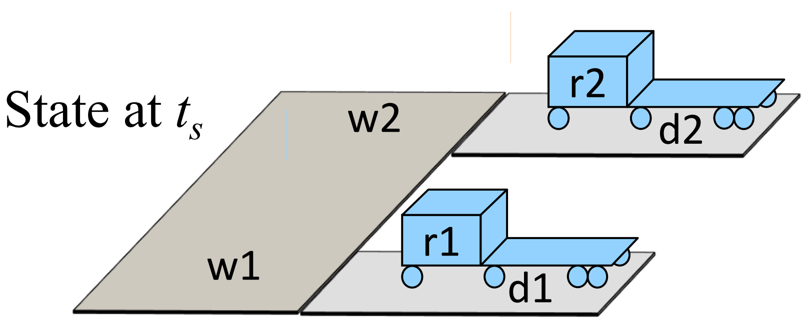
- Flaws:
  - two non-refined tasks
- Next, refine them
  - apply the move method

ϕ_0: tasks: $[t_s,t_e]$ move(r1,d2)
$[t_s,t_e]$ move(r2,d1)

supported: $[t_s]$ loc(r1)=d1
$[t_s]$ loc(r2)=d2

constraints: $t_s < t_e$
adjacent(d1,w1),
adjacent(d2,w2),
connected(w1,w2)





The *move* method

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

- Refinement method:

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

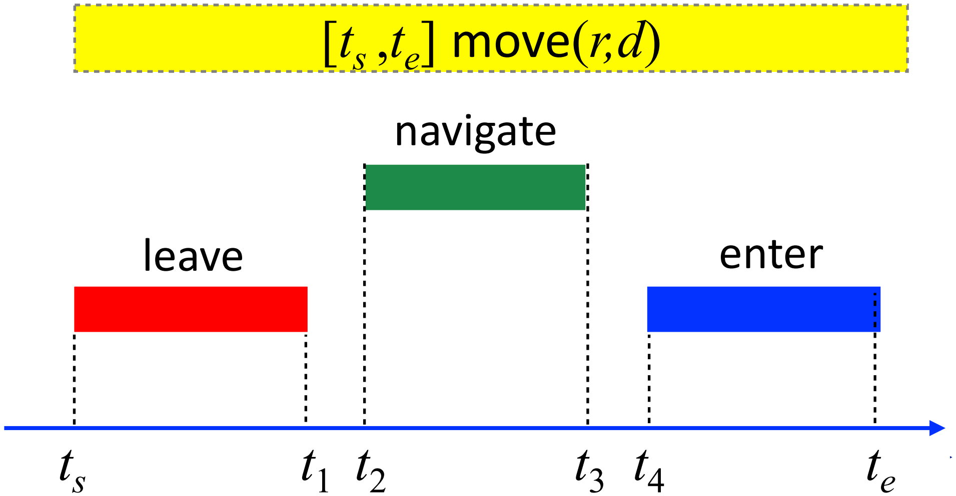
  task: move(r,d)

  refinement:

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

  constraints:

  $t_s < t_1,\ t_1 \le t_2,\ t_2 < t_3,\ t_3 \le t_4,\ t_4 < t_e$
  adjacent(d,w), adjacent(d',w')
  connected(w,w')



After refining *move*

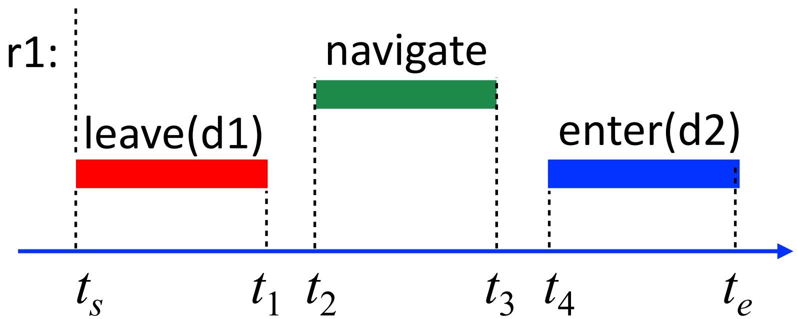
- Flaws:
  - ➢ 6 non-refined tasks (actions)
- Next, refine the red ones
  - apply action definition

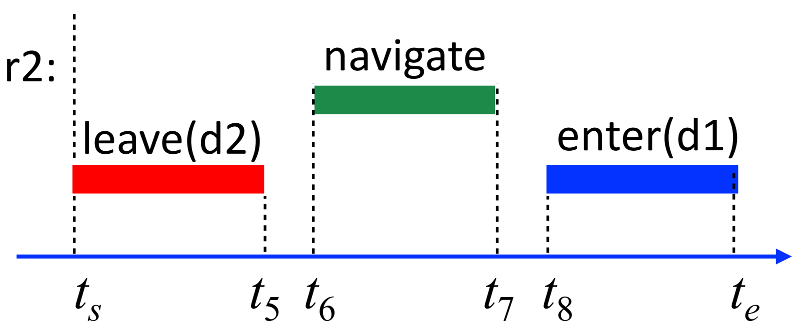
ϕ_1: tasks:
$[t_s,t_1]$ leave(r1,d1,w1)
$[t_2,t_3]$ navigate(r1,w1,w2)
$[t_4,t_e]$ enter(r1,d2,w2)

$[t_s,t_5]$ leave(r2,d2,w2)
$[t_6,t_7]$ navigate(r2,w2,w1)
$[t_8, t_e]$ enter(r2,d1,w1)

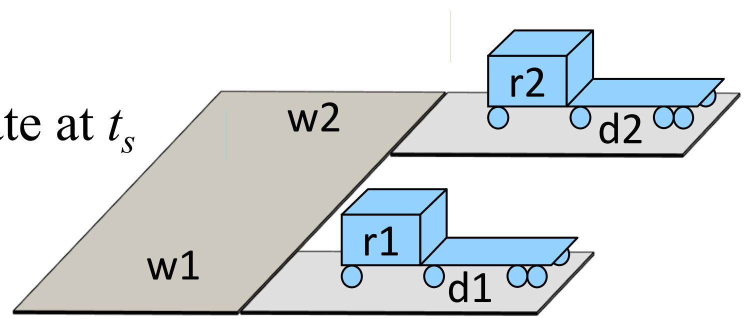
supported:
$[t_s]$ loc(r1)=d1
$[t_s]$ loc(r2)=d2

constraints:
$t_s < t_1 \leq t_2 < t_3 \leq t_4 < t_e$
$t_s < t_5 \leq t_6 < t_7 \leq t_8 < t_e$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)



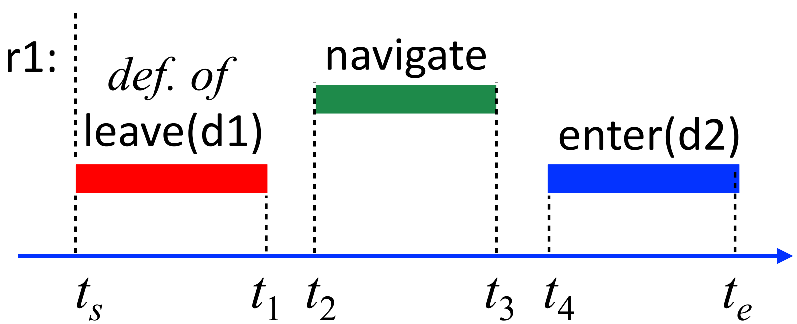


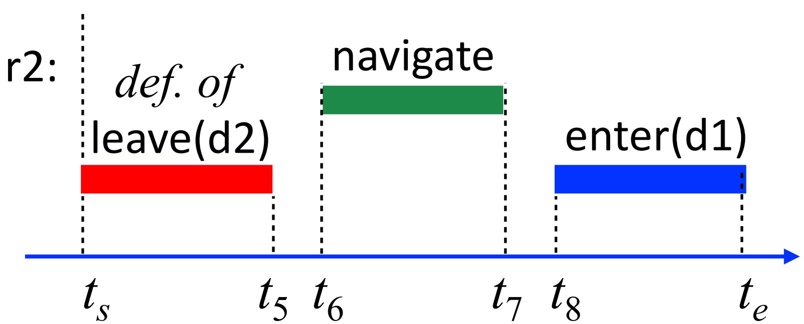
State at t_s



After refining *leave*

- Flaws:
  - 4 unrefined tasks
  - 4 unsupported assertions
- Next, refine the green and blue ones
  - apply action definitions





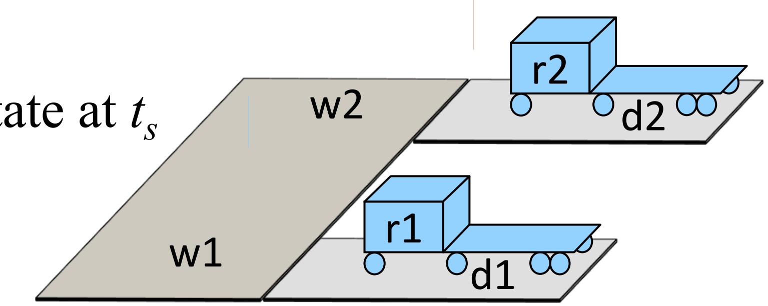
ϕ_2: tasks: $[t_2,t_3]$ navigate(r1,w1,w2)
$[t_4,t_e]$ enter(r1,d2,w2)
$[t_6,t_7]$ navigate(r2,w2,w1)
$[t_8, t_e]$ enter(r2,d1,w1)

assertions: $[t_s,t_1]$ loc(r1): (d1,w1)
$[t_s,t_1]$ occupant(d1): (r1,empty)
$[t_s,t_5]$ loc(r2): (d2,w2)
$[t_s,t_5]$ occupant(d2): (r2,empty)

supported: $[t_s]$ loc(r1)=d1
$[t_s]$ loc(r2)=d2

constraints: $t_s < t_1 \le t_2 < t_3 \le t_4 < t_e$
$t_s < t_5 \le t_6 < t_7 \le t_8 < t_e$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)

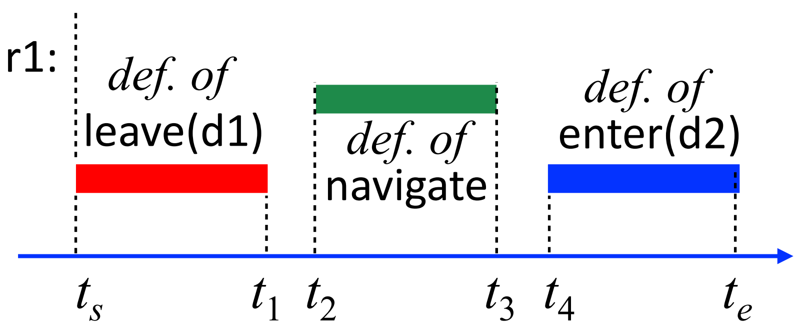
State at t_s

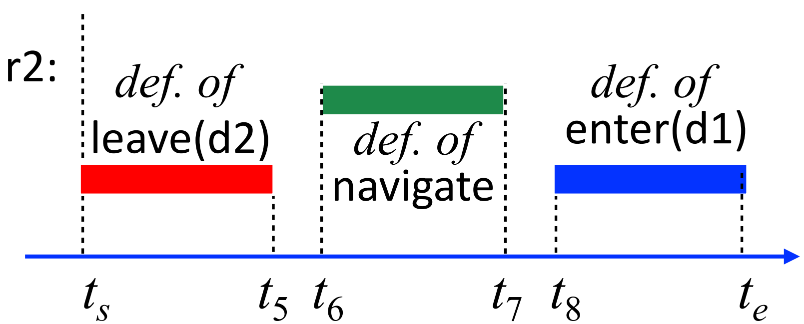


After refining *navigate* and *enter*

- Flaws:
  - 10 unsupported assertions
  - 2 possible conflicts

- Next, use the black ones to support the red ones



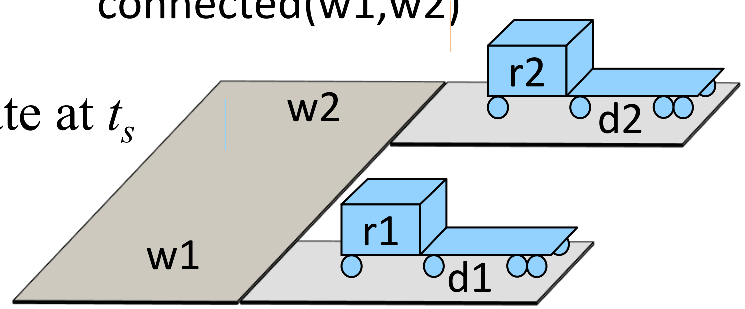


ϕ_3: assertions:
[t_s,t_1] loc(r1): (d1,w1)
[t_s,t_1] occupant(d1): (r1,empty)
[t_2,t_3] loc(r1): (w1,w2)
[t_4,t_e] loc(r1): (w2,d2)
[t_4,t_e] occupant(d2): (empty,r1)
[t_s,t_5] loc(r2): (d2,w2)
[t_s,t_5] occupant(d2): (r2,empty)
[t_6,t_7] loc(r2): (w2,w1)
[t_8,t_e] loc(r2): (w1,d1)
[t_8,t_e] occupant(d1): (empty,r1)

supported:
[t_s] loc(r1)=d1
[t_s] loc(r2)=d2

constraints:
$t_s+2 \le t_1 \le t_2 < t_3 \le t_4 \le t_e - 3$
$t_s+2 \le t_5 \le t_6 < t_7 \le t_8 \le t_e - 3$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)

State at t_s



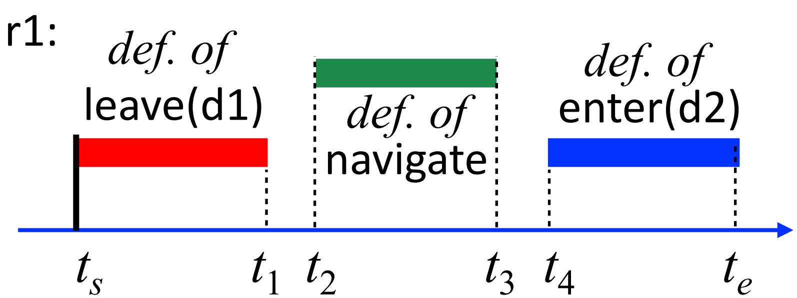
After supporting *leave*

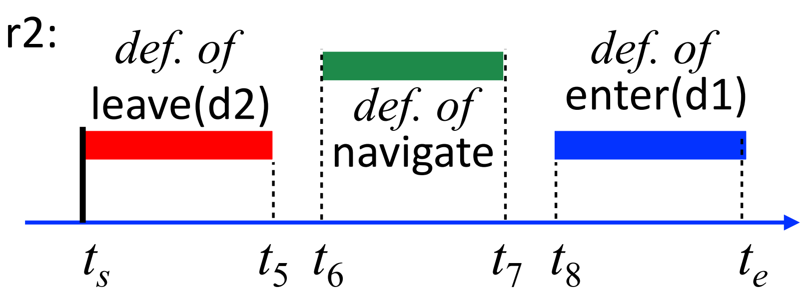
- Flaws:
  - 6 unsupported assertions
  - 2 possible conflicts
- Next, use the red ones to support the green ones
  - constrain $t_1 = t_2$ and $t_5 = t_6$

ϕ_4: assertions: $[t_2,t_3]$ loc(r1): (w1,w2)
$[t_4,t_e]$ loc(r1): (w2,d2)
$[t_4,t_e]$ occupant(d2): (empty,r1)
$[t_6,t_7]$ loc(r2): (w2,w1)
$[t_8,t_e]$ loc(r2): (w1,d1)
$[t_8,t_e]$ occupant(d1): (empty,r1)

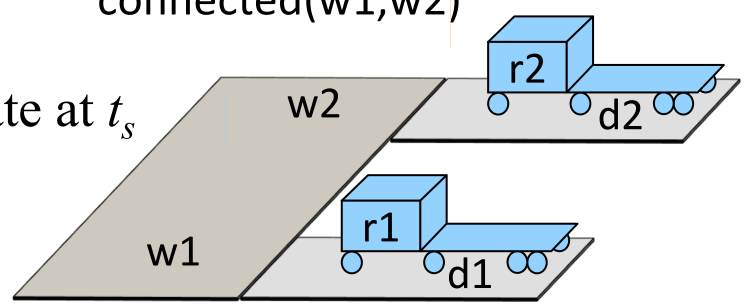
supported: $[t_s]$ loc(r1)=d1
$[t_s,t_1]$ loc(r1): (d1,w1)
$[t_s,t_1]$ occupant(d1): (r1,empty)
$[t_s]$ loc(r2)=d2
$[t_s,t_5]$ loc(r2): (d2,w2)
$[t_s,t_5]$ occupant(d2): (r2,empty)

constraints: $t_s+2 \le t_1 \le t_2 < t_3 \le t_4 \le t_e - 3$
$t_s+2 \le t_5 \le t_6 < t_7 \le t_8 \le t_e - 3$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)



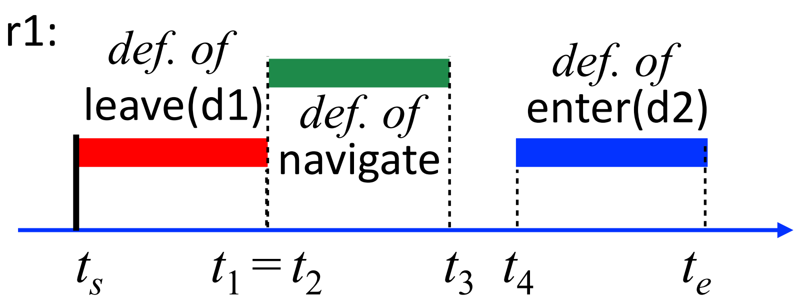


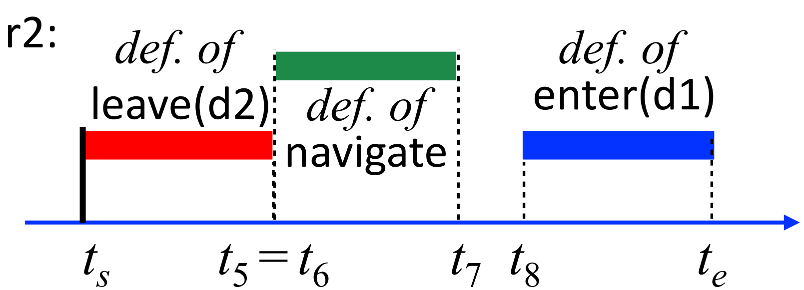
State at t_s



After supporting *navigate*

- Flaws:
  - 4 unsupported assertions
  - 2 possible conflicts
- Next, use the green ones to support the blue ones
  - constrain $t_3 = t_4$ and $t_7 = t_8$



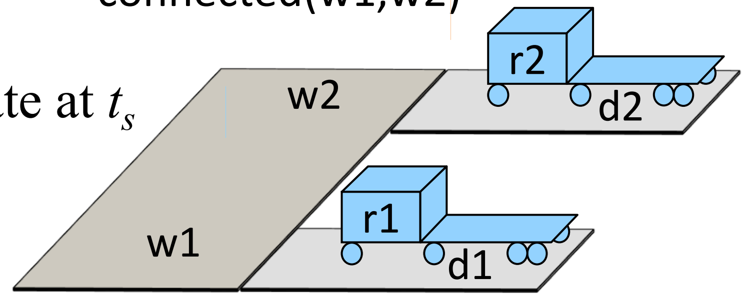


ϕ_5: assertions:
[t_4,t_e] loc(r1): (w2,d2)
[t_4,t_e] occupant(d2): (empty,r1)
[t_8,t_e] loc(r2): (w1,d1)
[t_8,t_e] occupant(d1): (empty,r1)

supported:
[t_s] loc(r1)=d1
[t_s,t_1] loc(r1): (d1,w1)
[t_s,t_1] occupant(d1): (r1,empty)
[t_2,t_3] loc(r1): (w1,w2)
[t_s] loc(r2)=d2
[t_s,t_5] loc(r2): (d2,w2)
[t_s,t_5] occupant(d2): (r2,empty)
[t_6,t_7] loc(r2): (w2,w1)

constraints:
$t_s+2 \le t_1 = t_2 < t_3 \le t_4 \le t_e - 3$
$t_s+2 \le t_5 = t_6 < t_7 \le t_8 \le t_e - 3$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)

State at t_s



After supporting *enter*

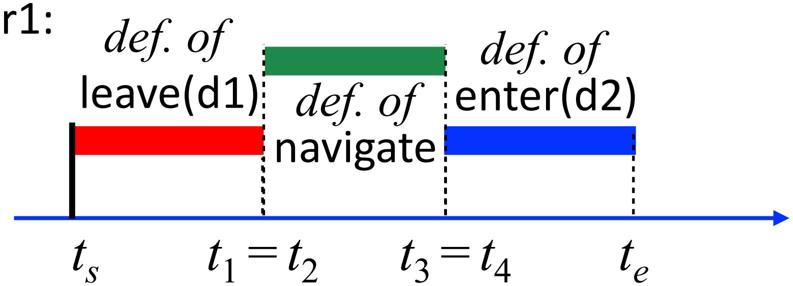
- Flaws: 2 possible conflicts

  if $t_3 < t_5$, r1 enters d2 before r2 has left
  - occupied(d2)=r1,r2

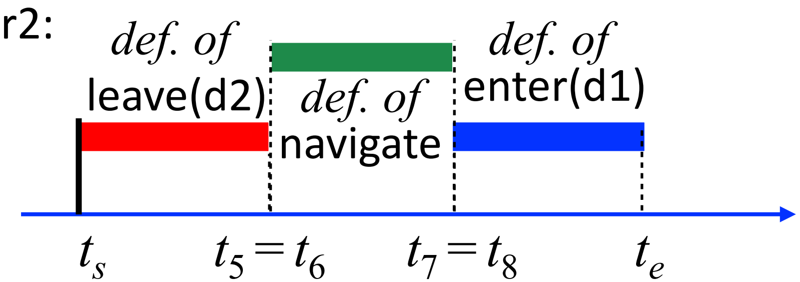
  if $t_7 < t_1$, r2 enters d1 before r1 has left
  - occupied(d2)=r1,r2

- Next, add separation constraints
  - $t_1 < t_7$ and $t_5 < t_3$

r1:



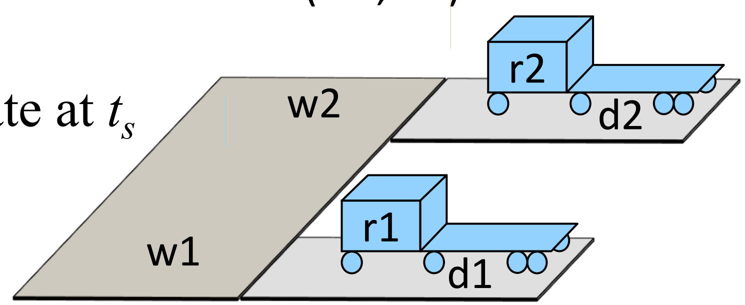
r2:



ϕ_6: supported:
[t_s] loc(r1)=d1
[t_s,t_1] loc(r1): (d1,w1)
[t_s,t_1] occupant(d1): (r1,empty)
[t_1,t_3] loc(r1): (w1,w2)
[t_4,t_e] loc(r1): (w2,d2)
[t_4,t_e] occupant(d2): (empty,r1)
[t_s] loc(r2)=d2
[t_s,t_5] loc(r2): (d2,w2)
[t_s,t_5] occupant(d2): (r2,empty)
[t_5,t_7] loc(r2): (w2,w1)
[t_8,t_e] loc(r2): (w1,d1)
[t_8,t_e] occupant(d1): (empty,r1)

constraints: $t_s+2 \le t_1 = t_2 < t_3 = t_4 \le t_e - 3$
$t_s+2 \le t_5 = t_6 < t_7 = t_8 \le t_e - 3$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)

State at t_s

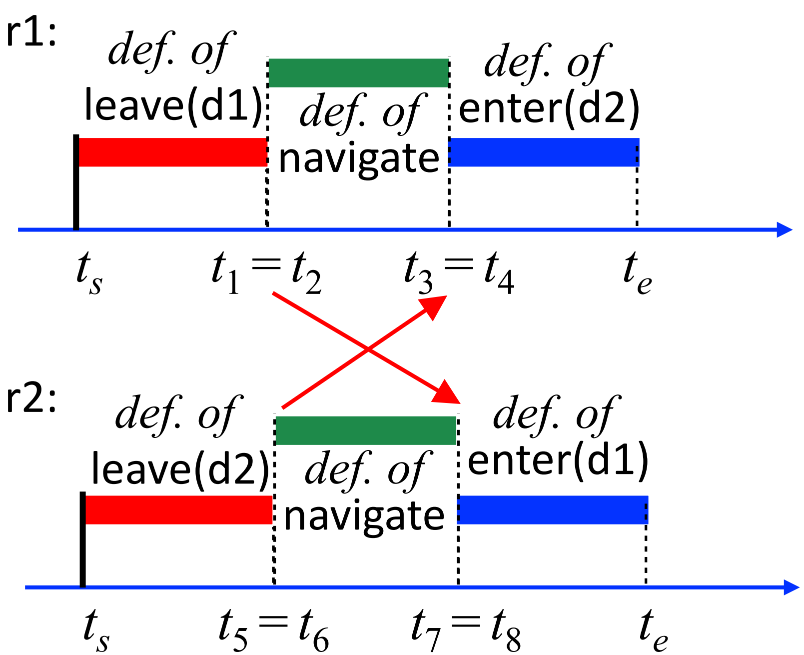


After adding separation constraints

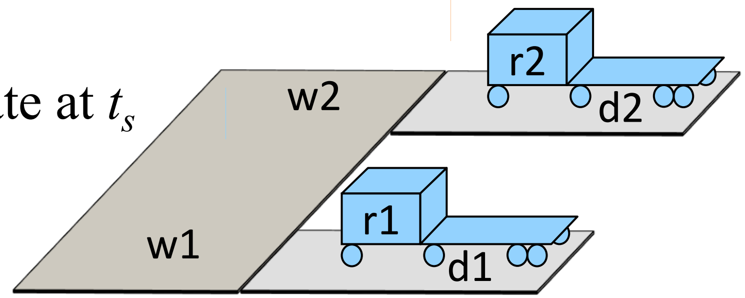
- Done!

ϕ_7: supported:
$[t_s]$ loc(r1)=d1
$[t_s,t_1]$ loc(r1): (d1,w1)
$[t_s,t_1]$ occupant(d1): (r1,empty)
$[t_1,t_3]$ loc(r1): (w1,w2)
$[t_3,t_e]$ loc(r1): (w2,d2)
$[t_3,t_e]$ occupant(d2): (empty,r1)
$[t_s]$ loc(r2)=d2
$[t_s,t_5]$ loc(r2): (d2,w2)
$[t_s,t_5]$ occupant(d2): (r2,empty)
$[t_5,t_7]$ loc(r2): (w2,w1)
$[t_7,t_e]$ loc(r2): (w1,d1)
$[t_7,t_e]$ occupant(d1): (empty,r1)

constraints: $t_s+2 \le t_1 = t_2 < t_3 = t_4 \le t_e - 3,\ t_1 < t_7$
$t_s+2 \le t_5 = t_6 < t_7 = t_8 \le t_e - 3,\ t_5 < t_3$
adjacent(d1,w1), adjacent(d2,w2),
connected(w1,w2)



State at t_s



Outline

- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- Speeding up TemPlan
  - Node selection heuristics, detection of constraint violations
- Controllability
- Acting with executable primitives
- Summary

Node Selection Heuristics

- Ideas similar to those in constraint-satisfaction algorithms

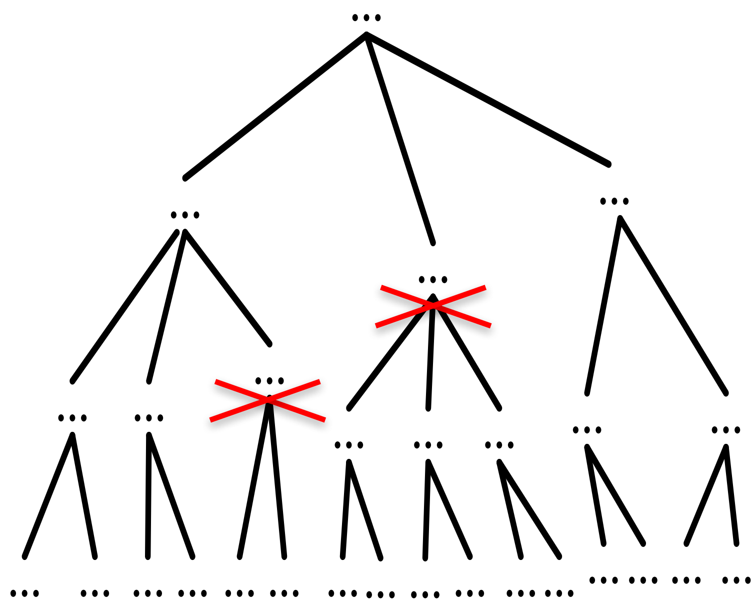
- Flaw selection, resolver selection
  - Select the flaw with the smallest number of resolvers
  - Choose the resolver that rules out the fewest resolvers for the other flaws

- More advanced heuristics
  - EUROPA2 [Bernardini & Smith, 2008]
  - FAPE [Bit-Monnot, 2016]

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)

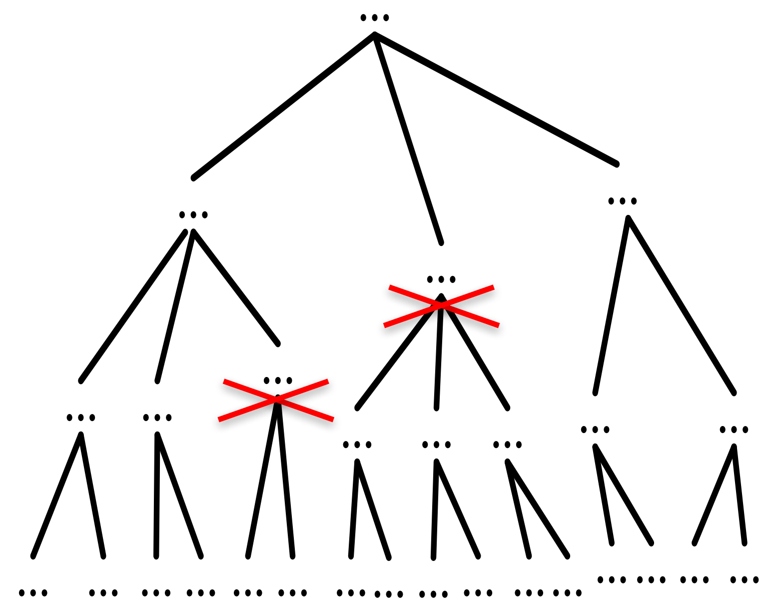
Detecting Constraint Violations

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



Detecting Constraint Violations

- When TemPlan changes C, check consistency
  - If C is inconsistent, then
    - No solutions below this node
    - Prune it



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

if ϕ is inconsistent then return failure

Consistency of C

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

- Assume object constraints are independent of temporal constraints and vice versa
  - exclude things like $t <$ speed(r1)

- Two separate subproblems

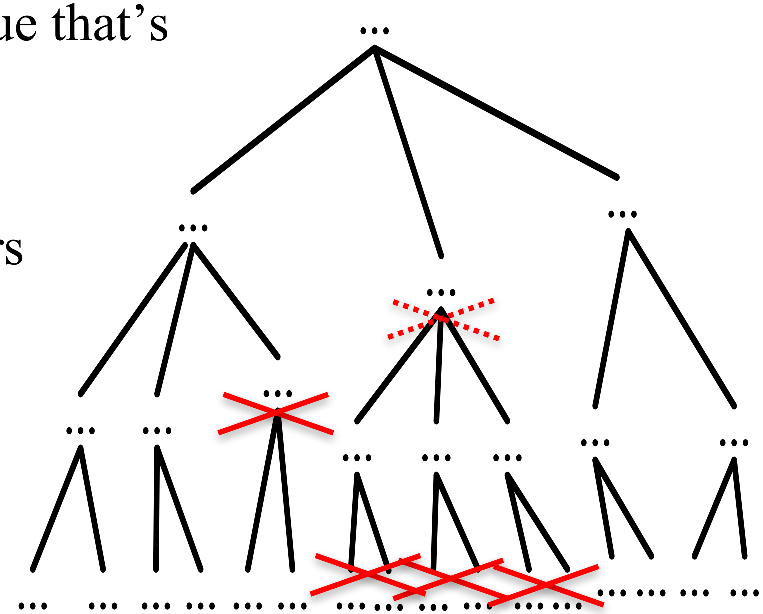
  (1) are the object constraints consistent?

  (2) are the temporal constraints consistent?
  - C is consistent iff both are consistent

Object Constraints

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

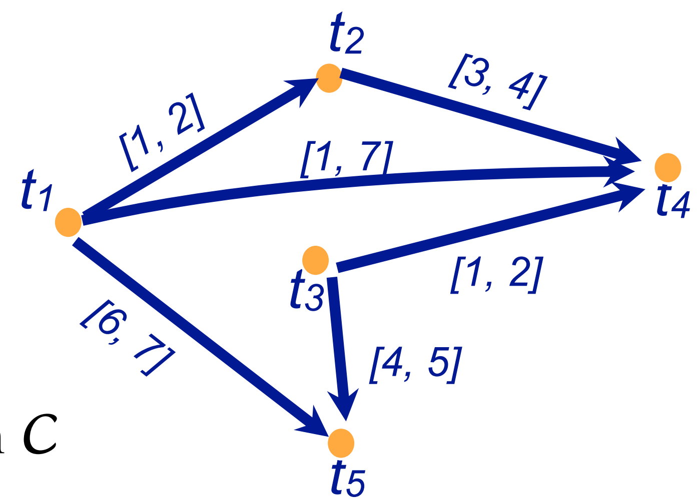
- Instead, use a constraint-satisfaction technique that's incomplete but takes *polynomial* time
  - arc consistency, path consistency
  - ➢ Detect some inconsistencies but not others
- Consequence
  - ➢ Don't prune as much of the search space
  - ➢ Affects efficiency but not correctness



Time Constraints

To represent time constraints:

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

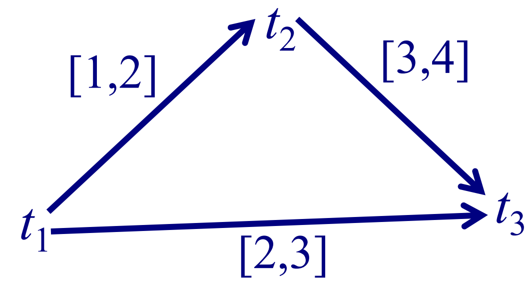


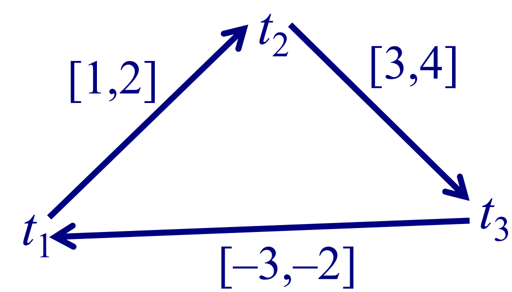
- Can modify TemPlan to
  - Create initial network from the constraints in C
  - Check consistency in polynomial time
    - $O(n^3)$
  - Every time C changes
    - update the network
    - check consistency

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

update temporal network
if it's inconsistent then
return failure

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 $a \le t_j - t_i \le b$
  - Equivalently, $-b \le t_i - t_j \le -a$
- Representing unary constraints: dummy variable $t_0 = 0$
  - ➢ Arc $r_{0i} = (t_0,t_i)$ labeled with $[a,b]$ represents $a \le t_i - 0 \le b$

- *Solution* to an STN: integer value for each t_i, all constraints satisfied
- *Consistent* STN: has a solution
- *Minimal* STN: for every arc (t_i,t_j) with label $[a,b]$, for every $t \in [a,b]$
  - ➢ there's at least one solution such that $t_j - t_i = t$

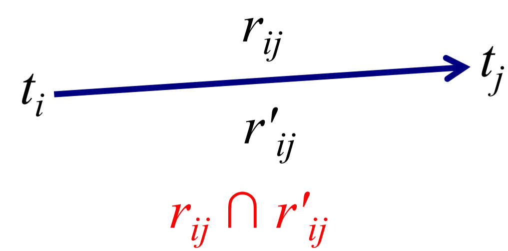
Operations on STNs

- Intersection:

$$t_j - t_i \in r_{ij} = [a_{ij}, b_{ij}]$$

$$t_j - t_i \in r'_{ij} = [a'_{ij}, b'_{ij}]$$

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



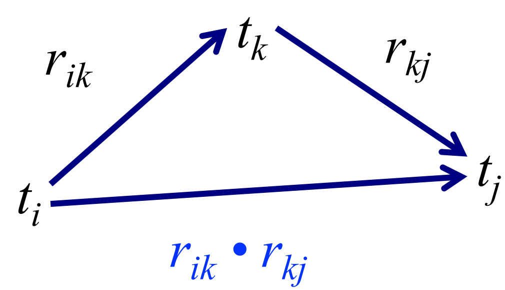
- Composition:

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

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

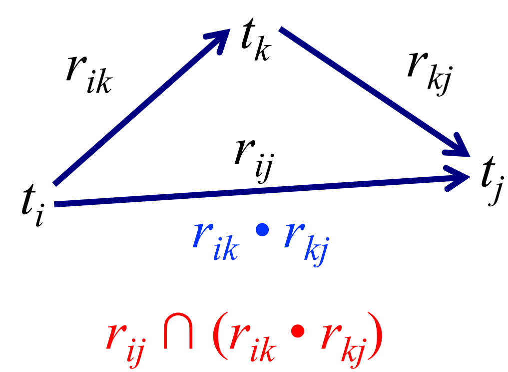
Infer $t_j - t_i \in r_{ik} \bullet r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}]$

Reason: shortest and longest times for the two intervals

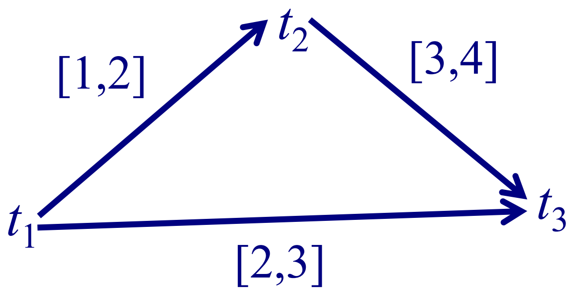


- Consistency checking:

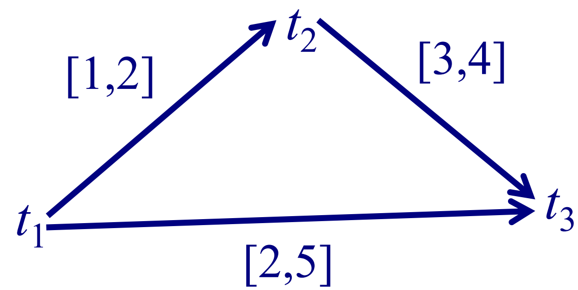
r_{ik}, r_{kj}, r_{ij} are consistent if $r_{ij} \cap (r_{ik} \bullet r_{kj}) \neq \varnothing$



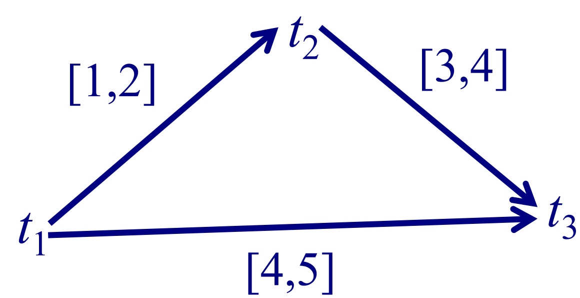
Two Examples



- 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}
  - $r_{13} \cap r'_{13} = [2,3] \cap [4,6] = \varnothing$
- $(\mathcal{V},\mathcal{E})$ is inconsistent



- STN $(\mathcal{V},\mathcal{E})$, where
  - $\mathcal{V} = \{t_1, t_2, t_3\}$
  - $\mathcal{E} = \{r_{12}=[1,2],\ r_{23}=[3,4],\ r_{13}=[2,5]\}$
- As before, $r'_{13} = [4,6]$
- This time, $(\mathcal{V},\mathcal{E})$ is consistent
  - $r_{13} \cap r'_{13} = [4,5]$
- To get minimal network, change $r_{13} \leftarrow [4,5]$



Operations on STNs

- PC (*Path Consistency*) algorithm:
  - Consistency checking on all triples
  - n constraints => n^3 triples
    => time $O(n^3)$
- Detects inconsistent networks
  - $r_{ij} = [a_{ij}, b_{ij}]$ empty => inconsistent
- Makes STN minimal
  - Shrinks each r_{ij} to exclude values that aren't in any solution

```
PC(V,E):
  for 1 ≤ k ≤ n do
    for 1 ≤ i < j ≤ n, i ≠ k, j ≠ k do
      r_ij ← r_ij ∩ [r_ik • r_kj]
      if r_ij = ∅ then
        return inconsistent
```

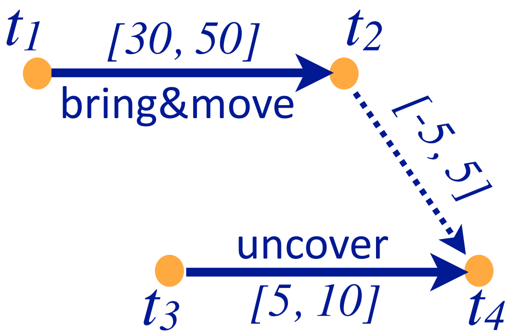
- Can modify it to make it *incremental*
  - Input: a consistent, minimal STN, and a new constraint r'_{ij}
  - Incorporate r'_{ij} in time $O(n^2)$
- Whenever the network becomes inconsistent, prune this part of the search space

Outline

- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- ✓ Speeding up TemPlan
- Controllability
- Acting with executable primitives
- Summary

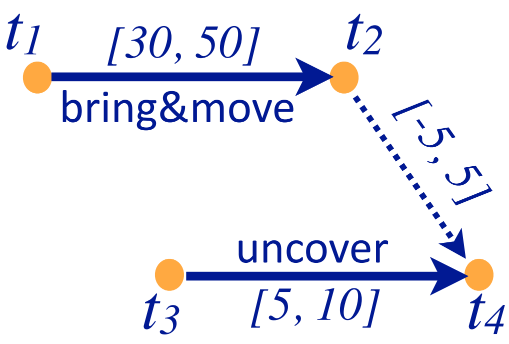
Controllability

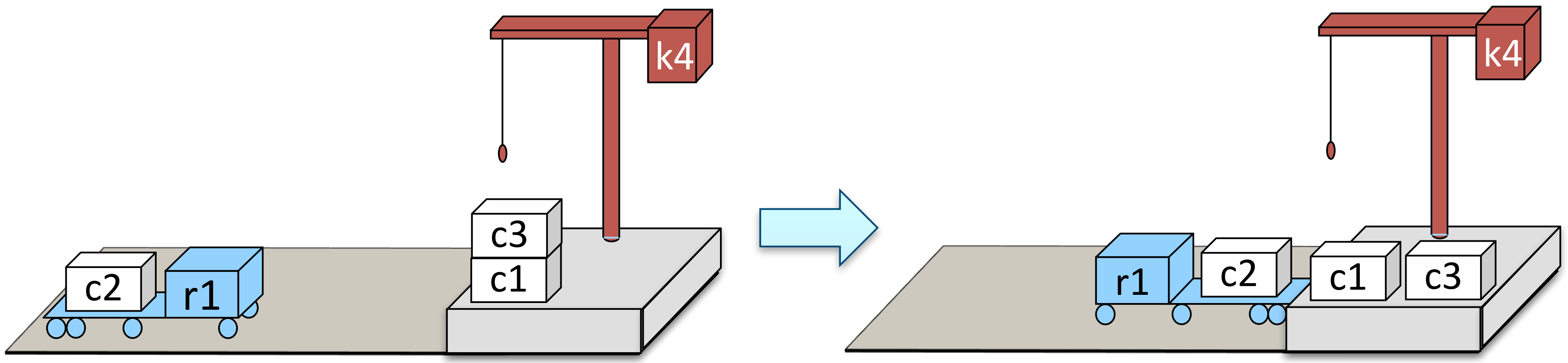
- Suppose TemPlan gives you a temporal network and you want to perform it
  - Constraints on time points
  - Need to reason about these in order to decide when to start each action



Controllability

- Solid lines: duration constraints
  - Robot will do bring&move, will take 30 to 50 time units
  - Crane will do uncover, will take 5 to 10 time units
- Dashed line: synchronization constraint
  - Don’t want either the crane or robot to wait long
  - At most 5 seconds between the two ending times
- Objective
  - Choose time points that will satisfy all the constraints

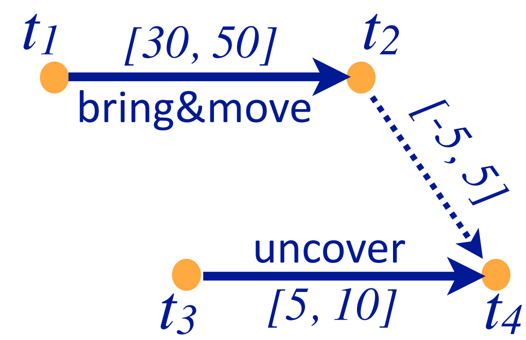




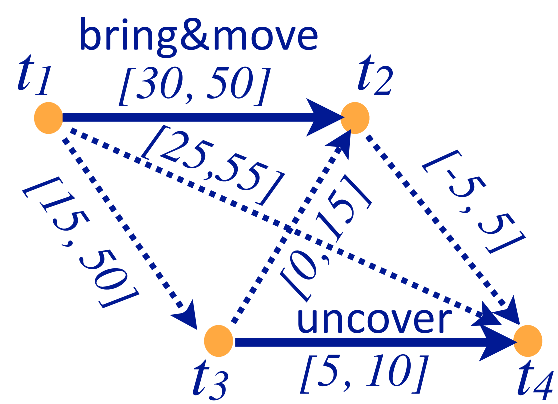
Controllability

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 $r_{ij} = \varnothing$ then
    return inconsistent

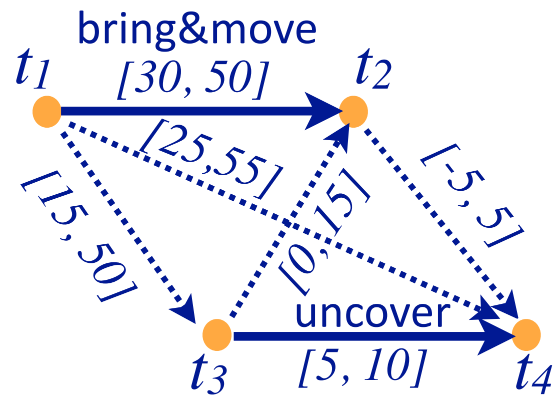
- Suppose we use PC
  - get a minimal and consistent network
- There *exist* time points that satisfy all the constraints
- Would work if we could choose all four time points
  - But we can't choose t_2 and t_4



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



Controllability



- Can't guarantee that all of the constraints will be satisfied

- Start bring&move at time $t_1 = 0$
- Suppose the durations are
  - bring&move 30, uncover 10

  ➢ $t_2 = 0 + 30 = 30$

  ➢ $t_4 = t_3 + 10$

  ➢ $t_4 - t_2 = t_3 - 20$
- Constraint $-5 \le t_4 - t_2 \le 5$

  $\rightarrow \quad -5 \le t_3 - 20 \le 5$
- Need to start uncover at $t_3 \le 25$

  ➢ If $t_3 > 25$ then $t_4 - t_2 > 5$

- But if we start uncover at $t_3 \le 25$, neither action has finished yet

  ➢ We don't yet know how long they'll take
- Might instead get this:
  - bring&move 50, uncover 5

  ➢ $t_2 = 0 + 50 = 50$

  ➢ $t_4 = t_3 + 5 \le 25 + 5 = 30$

  ➢ $t_4 - t_2 \le 30 - 50 = -20$

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} = {starting times of actions}
    - $\tilde{\mathcal{V}}$ = {*contingent* time points} = {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

Dynamic Execution

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

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

- Want *dynamic controllability*
  - Choose values for $\mathcal{V}$ online by observing what has happened so far
  - Need a strategy for how to choose the values

Dynamic Execution

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

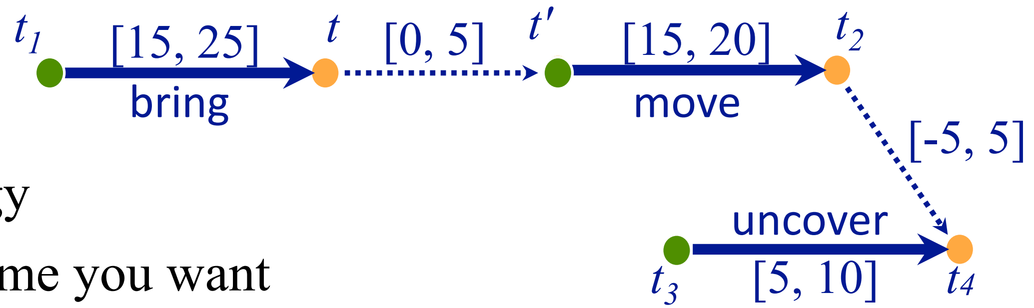
1. Actor chooses time points $\mathcal{V}_t \subseteq \mathcal{V}$ that can be triggered at time t without violating any *synchronization* constraints
   - actions that the actor chooses to start
2. Simultaneously, environment chooses time points $\tilde{\mathcal{V}}_t \subseteq \tilde{\mathcal{V}}$ that can be triggered at time t without violating any *duration* constraints
   - actions that the environment chooses to finish
3. They trigger the time points they've chosen, and remove them from $\mathcal{V}$ and $\tilde{\mathcal{V}}$
   - history h = record of all that has happened = $\{\mathcal{V}_t, \tilde{\mathcal{V}}_t\}$ for $i = 1, \ldots, t$
4. Failure if any of the constraints are violated
   - $r_{ij} = [l,u]$ is *violated* if t_i and t_j have values (step 3) and $t_j - t_i \notin [l,u]$
5. Success if no constraints violated, and $\mathcal{V} = \tilde{\mathcal{V}} = \varnothing$

- *Dynamic execution strategy* $\sigma_A(h)$ for actor, $\sigma_E(h)$ for environment
  - What to choose next, given h

- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is *dynamically controllable* if there *exists* a σ_A that will guarantee success for *every* σ_E

Example

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



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

Dynamic Controllability Checking

- How to check whether an STNU is dynamically controllable
  - Extension of consistency checking
- For a chronicle $\phi = (\mathcal{A}, \mathcal{S}_\mathcal{T}, \mathcal{T}, \mathcal{C})$
  - Temporal constraints in $\mathcal{C}$ correspond to an STNU
- TemPlan can keep the STNU dynamically controllable
  - use the incremental version of PC

- If PC reduces the size of a contingent constraint r_{ij}
  - Then the STNU isn't dynamically controllable
    - → prune this path in the search space
  - Otherwise, further test of dynamic controllability
    - extension of Path Consistency, additional constraint propagation rules

Outline

- ✓ Introduction
- ✓ Representation
- ✓ Temporal planning
- ✓ Speeding up TemPlan
- ✓ Controllability
- Acting with executable primitives
  - Acting with atemporal refinement
  - Dispatching
  - Observation actions
- Summary

Atemporal Refinement of Primitive Actions

- TemPlan's actions may correspond to tasks for Rae to refine using refinement methods not in TemPlan

- TemPlan action (descriptive model)

$$
\begin{aligned}
&\mathsf{leave}(r,d,w)\\
&\quad \text{assertions: } [t_s,t_e]\mathsf{loc}(r){:}(d,w)\\
&\qquad\qquad\qquad\;\; [t_s,t_e]\mathsf{occupant}(d){:}(r,\mathsf{empty})\\
&\quad \text{constraints: } t_e \le t_s + \delta_1\\
&\qquad\qquad\qquad\;\; \mathsf{adjacent}(d,w)
\end{aligned}
$$

- Rae refinement method (operational model)

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

Atemporal Refinement of Primitive Actions

- TemPlan's actions may correspond to tasks for Rae to refine using refinement methods not in TemPlan

- TemPlan action (descriptive model)

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

- Rae refinement method (operational model)

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)

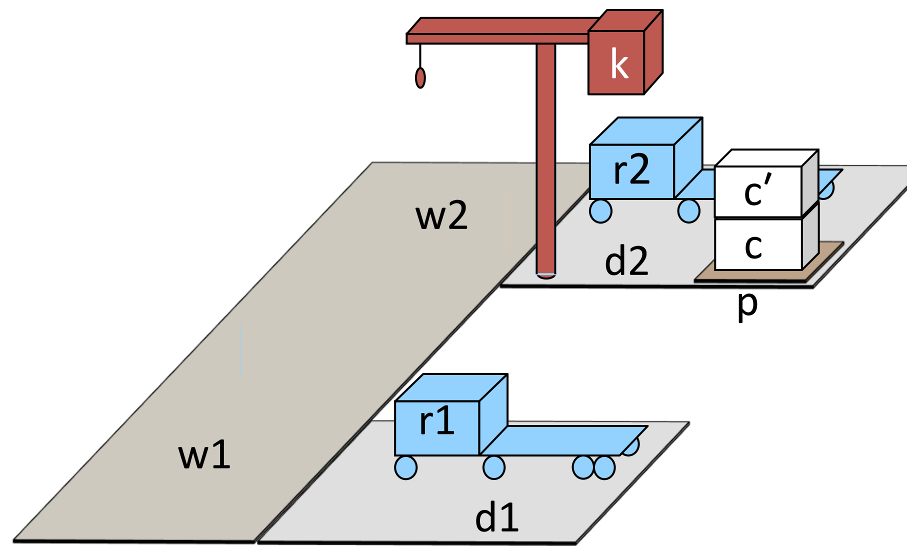
Discussion

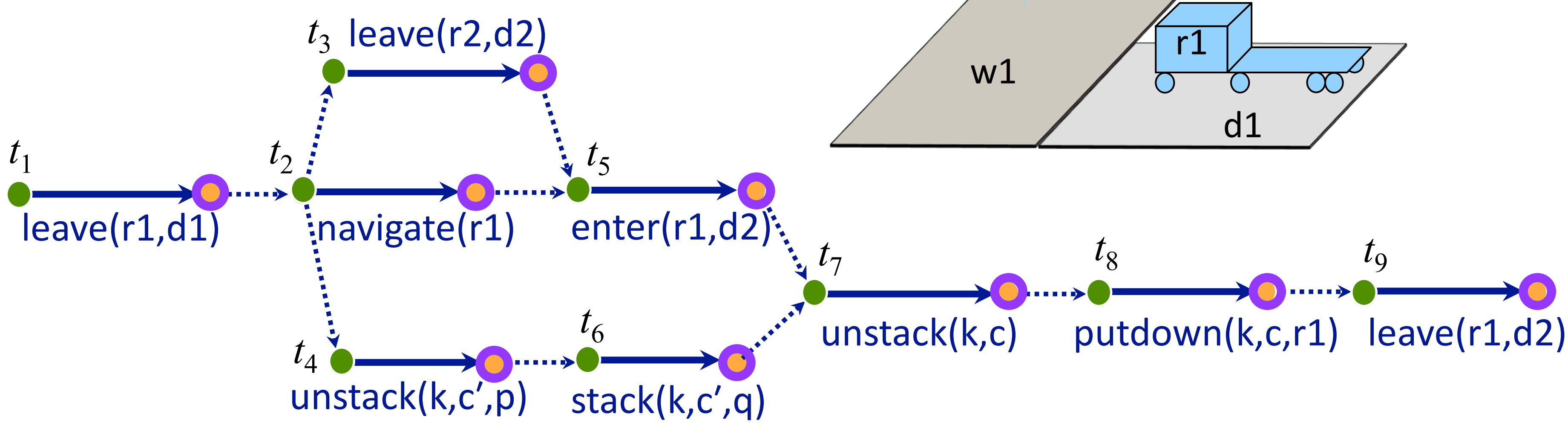
- Pros
  - Simple online refinement with Rae
  - Avoids breaking down uncertainty of contingent duration
  - Can be augmented with temporal monitoring functions in Rae
  - E.g., watchdogs, methods with duration preferences
- Cons
  - Does not handle temporal requirements at the command level, e.g., concurrency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

- 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 it 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 t is instantiated
- This bounds each time point t within an interval $[l_t,u_t]$

Controllable time point t in the future:

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

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

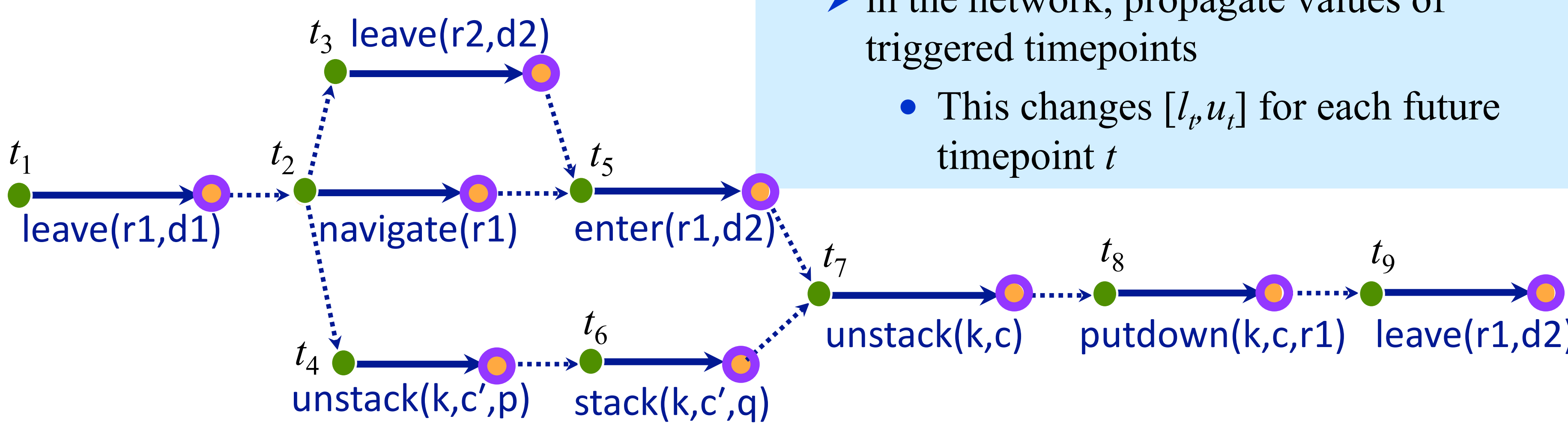
- initialize the network
- while there are time points in $\mathcal{V}$ that haven't yet been triggered, do
  - update *now*
  - update the time points in $\tilde{\mathcal{V}}$ that were triggered since the last iteration
  - *enabled* $\leftarrow \{t \in \mathcal{V} \mid t$ hasn't yet been triggered, and $l_t \leq now \leq u_t\}$
  - for every $t \in$ *enabled* such that $now = u_t$
    - trigger t
  - arbitrarily choose other time points in *enabled*, and trigger them
  - in the network, propagate values of triggered timepoints
    - This changes $[l_t,u_t]$ for each future timepoint t

Example

- trigger t_1, observe leave finish
- enable and trigger t_2, this enables t_3, t_4
- trigger t_3 (start leave(r2,d2)) soon enough to allow enter(r1,d2) at time t_5
- trigger t_4 (start unstack(k,c′)) soon enough to allow stack(k,c′) at time t_6
- rest of plan is linear: choose each t_i after the previous action ends

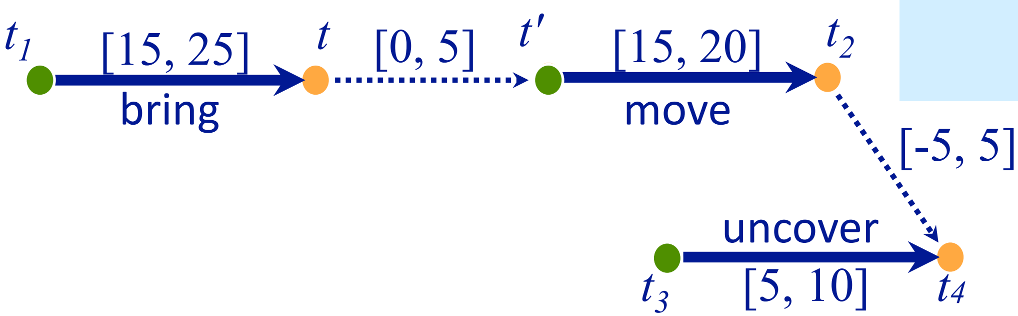
Dispatch($\mathcal{V}$,$\tilde{\mathcal{V}}$,$\mathcal{E}$,$\tilde{\mathcal{E}}$)

- initialize the network
- while there are time points in $\mathcal{V}$ that haven't yet been triggered, do
  - update *now*
  - update the time points in $\tilde{\mathcal{V}}$ that were triggered since the last iteration
  - *enabled* $\leftarrow \{t \in \mathcal{V} \mid t$ hasn't yet been triggered, and $l_t \le now \le u_t\}$
  - for every $t \in$ *enabled* such that $now = u_t$
    - trigger t
  - arbitrarily choose other time points in *enabled*, and trigger them
  - in the network, propagate values of triggered timepoints
    - This changes $[l_t, u_t]$ for each future timepoint t



Example

- trigger t_1 at time 0
- wait and observe t
- trigger t' at any time from t to t+5
- trigger t_3 at time $t' + 10$
  - $t_2 \in [t' + 15, t' + 20]$
  - $t_4 \in [t_3 + 5, t_3 + 10]$
    $= [t' + 15, t' + 20]$
    - so $t_4 - t_3 \in [-5, 5]$
- So all the constraints are satisfied

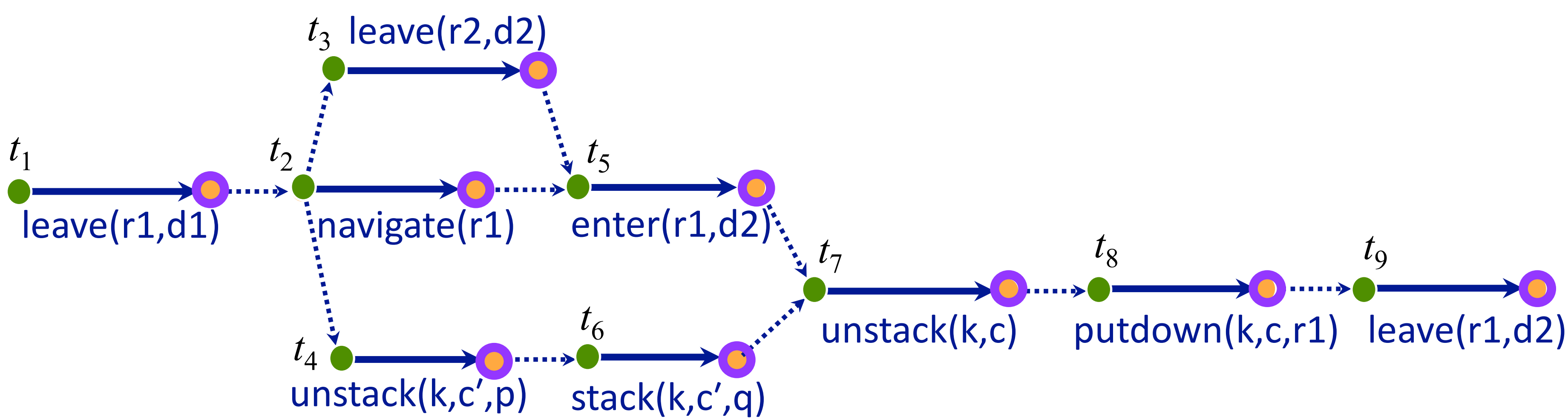


Dispatch($\mathcal{V},\tilde{\mathcal{V}},\mathcal{E},\tilde{\mathcal{E}}$)

- initialize the network
- while there are time points in $\mathcal{V}$ that haven't yet been triggered, do
  - update *now*
  - update the time points in $\tilde{\mathcal{V}}$ that were triggered since the last iteration
  - $enabled \leftarrow \{t \in \mathcal{V} \mid t$ hasn't yet been triggered, and $l_t \leq now \leq u_t\}$
  - for every $t \in enabled$ such that $now = u_t$
    - trigger t
  - arbitrarily choose other time points in *enabled*, and trigger them
  - in the network, propagate values of triggered timepoints
    - This changes $[l_t, u_t]$ for each future timepoint t

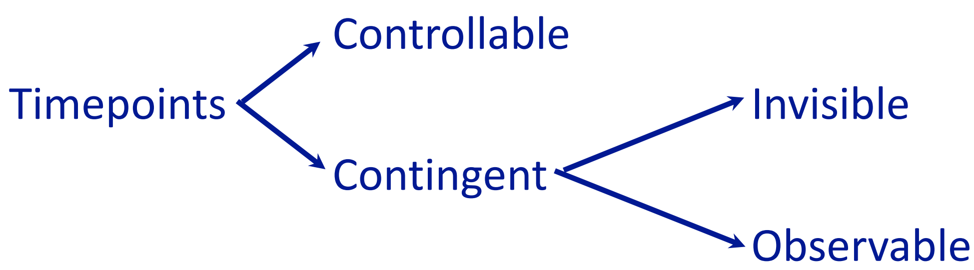
Deadline Failures

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



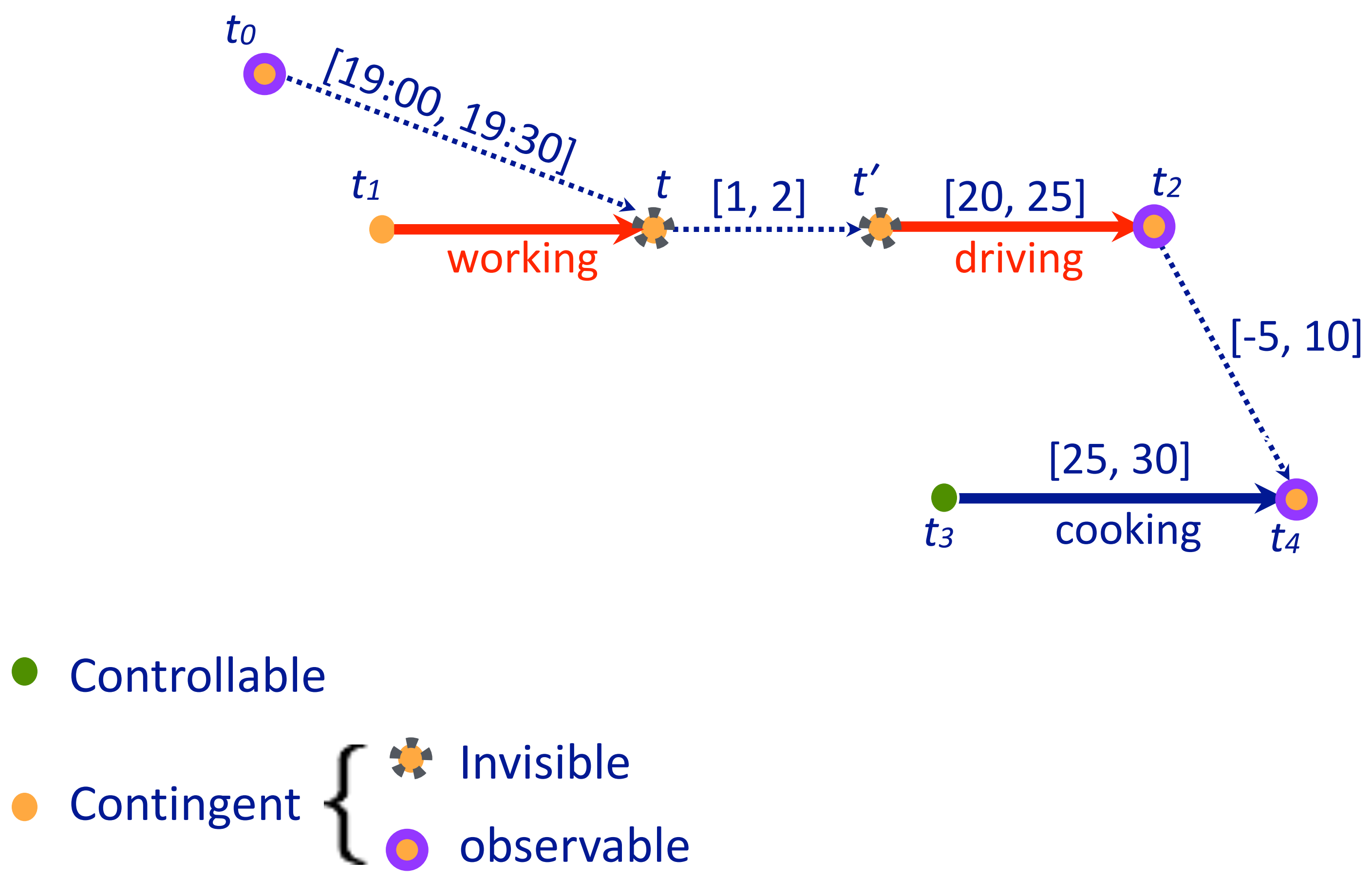
Partial Observability

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



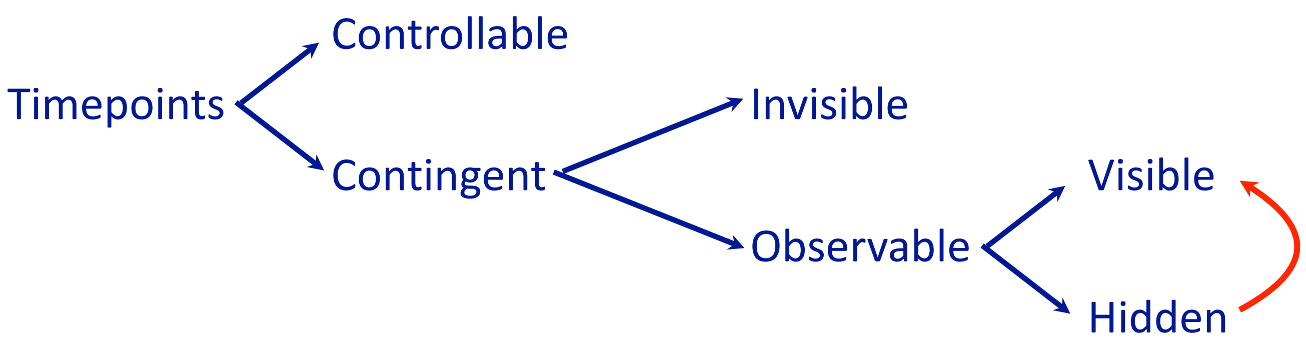
- Dynamically controllable?

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
- ✓ Speeding up TemPlan
- ✓ Controllability
- ✓ Acting with executable primitives
- Summary

Summary

- Timelines
  - Temporal assertions (change, persistence), constraints
  - Conflicts, consistency, security, causal support
  - Consistency, security, causal support
- Chronicle: timelines + supported/unsupported info + tasks
- 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
  - Managing constraints: like CSPs
    - Temporal constraints: STNs, PC algorithm (path consistency)
- Acting: dynamic controllability, STNUs, RAE and eRAE, dispatching

Relation to the Book

Any questions?

- Ghallab, Nau, and Traverso (2016). *Automated Planning and Acting.* Cambridge University Press
- Free downloads:
  - Lecture slides, final manuscript
  - http://www.laas.fr/planning
- Table of Contents
  1. Introduction
  2. Deterministic Models
  3. Refinement Methods
  4. Temporal Models
  5. Nondeterministic Models
  6. Probabilistic Models
  7. Other Deliberation Functions