UNIVERSITÄT ZU LÜBECK

Automated Planning and Acting – Temporal Models

Institute of Information Systems

Mattis Hartwig

Content

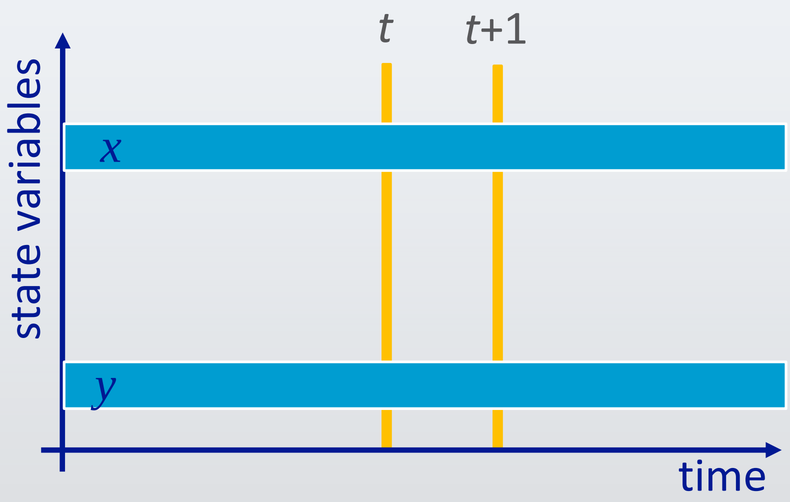
1. Planning and Acting with **Deterministic** Models
   Conventional AI planning
2. Planning and Acting with **Refinement** Methods
   Abstract activities → collections of less-abstract activities
3. Planning and Acting with **Temporal** Models
   Reasoning about time constraints
4. Planning and Acting with **Nondeterministic** Models
   Actions with multiple possible outcomes
5. **Standard** Decision Making
   Utility theory
   Markov decision process (MDP)
6. Planning and Acting with **Probabilistic** Models
   Actions with multiple possible outcomes, with probabilities
7. **Advanced** Decision Making
   Hidden goals
   Partially observable MDP (POMDP)
   Decentralised POMDP
8. **Human-aware** Planning
   Planning with a human in the loop
9. **Causal** Planning
   Causality & Intervention
   Implications for Causal Planning

Temporal Models

- Durations of actions
- Delayed effects and preconditions
  - E.g., resources borrowed or consumed during an action
- Time constraints on goals
  - Relative or absolute
- Exogenous events expected to occur in the future
  - When?
- Maintenance actions:
  - Maintain a property (≠ changing a value)
  - E.g., track a moving target, keep a spring latch in position
- Concurrent actions
  - Interacting effects, joint effects
- Delayed commitment
  - Instantiation at acting time

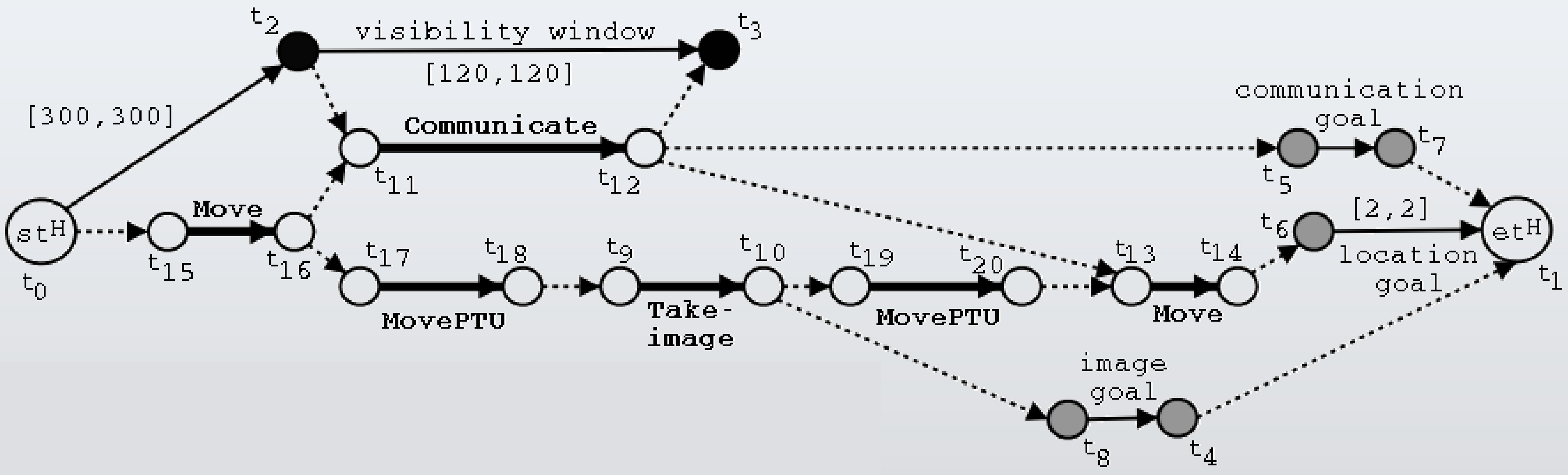
Timelines

- Up to now, "state-oriented view"
  - Time is a sequence of states s_0, s_1, s_2
  - Instantaneous actions transform each state into the next one
  - No overlapping actions
- Switch to a "time-oriented view"
  - Sequence of integer time points
    - $t = 1, 2, 3, \ldots$
  - For each state variable x, a timeline
    - Values during different time intervals
  - State at time $t = \{\text{state}-\text{variable values at time } t\}$



Timelines

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



Representation

UNIVERSITÄT ZU LÜBECK

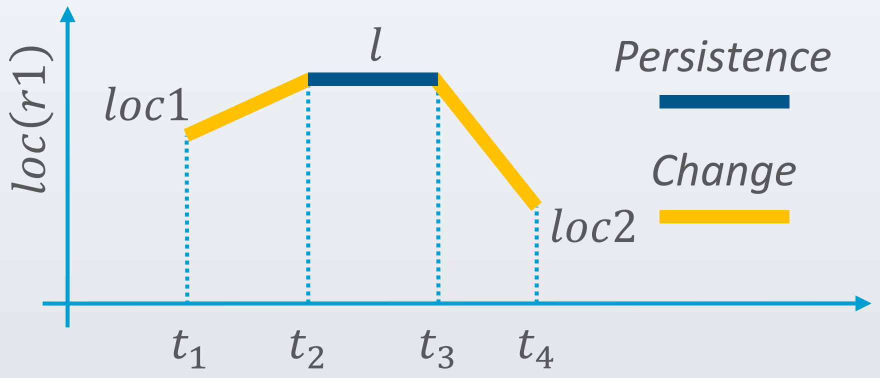
- Quantitative model of time
  - Discrete: time points are integers
- Expressions:
  - time-point variables
    - $t,\ t',\ t_2,\ t_j,\ \ldots$
  - simple constraints
    - $t < t'$
    - $d \leq t' - t \leq d'$,
  - x(t) refers to the value of variable x at time t
- Temporal assertion:
  - Value of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

Quiz

What is the right assertion that says robot r1 changes the location from loc2 to loc3 in the interval [t5, t6]?

Timeline

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



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

Quiz

The timeline ({[t1, t2]loc(r)=loc1, [t3, t4]loc(r1)=l}, {t1 < t2; t3 < t4}) is consistent but not secure. What is a potential conflict?

What needs to change so it becomes secure?

Actions

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

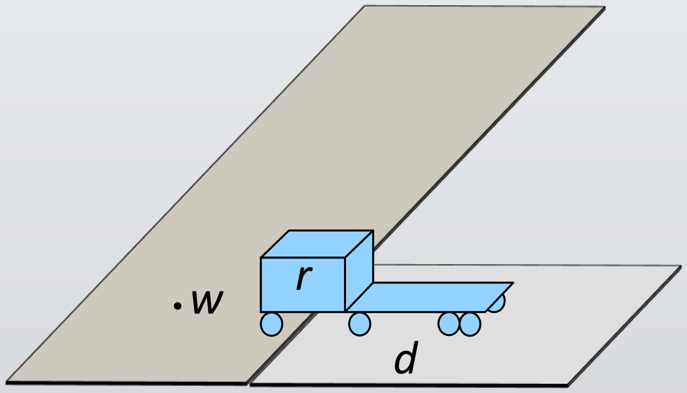
Actions

- $leave(r, d, w)$
  - Robot r leaves dock d,
    goes to adjacent waypoint w

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

- $loc(r)$ changes to w with delay $\le \delta_1$
- Dock d becomes empty

- Two additional parameters
  - Starting time t_s
  - Ending time t_e
- No separate preconditions and effects
  - Preconditions ⇔ need for causal support



Quiz

Specify the enter action template.

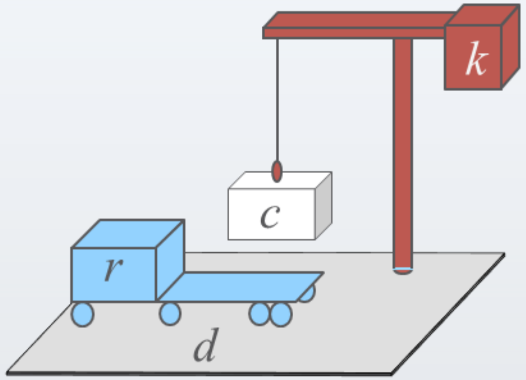
$$
\begin{aligned}
\mathsf{loc}(r) &\in \mathit{Docks} \cup \mathit{Waypoints} && \text{for } r \in \mathit{Robots}\\
\mathsf{freight}(r) &\in \mathit{Containers} \cup \{\mathsf{empty}\} && \text{for } r \in \mathit{Robots}\\
\mathsf{grip}(k) &\in \mathit{Containers} \cup \{\mathsf{empty}\} && \text{for } k \in \mathit{Cranes}\\
\mathsf{pos}(c) &\in \mathit{Robots} \cup \mathit{Cranes} \cup \mathit{Piles} && \text{for } c \in \mathit{Containers}\\
\mathsf{stacked\text{-}on}(c) &\in \mathit{Containers} \cup \{\mathsf{empty}\} && \text{for } c \in \mathit{Containers}\\
\mathsf{top}(p) &\in \mathit{Containers} \cup \{\mathsf{empty}\} && \text{for } p \in \mathit{Piles}\\
\mathsf{occupant}(d) &\in \mathit{Robots} \cup \{\mathsf{empty}\} && \text{for } d \in \mathit{Docks}.
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{attached} \subseteq (\mathit{Cranes} \cup \mathit{Piles}) \times \mathit{Docks}\\
&\mathsf{adjacent} \subseteq \mathit{Docks} \times \mathit{Waypoints}\\
&\mathsf{connected} \subseteq \mathsf{Waypoints} \times \mathsf{Waypoints}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{leave}(r, d, w) &: \text{robot } r \text{ leaves dock } d \text{ to an adjacent waypoint } w,\\
\mathsf{enter}(r, d, w) &: r \text{ enters } d \text{ from an adjacent wyapoint } w,\\
\mathsf{navigate}(r, w, w') &: r \text{ navigates from waypoint } w \text{ to a connected one } w',\\
\mathsf{stack}(k, c, p) &: \text{crane } k \text{ holding container } c \text{ stacks it on top of pile } p,\\
\mathsf{unstack}(k, c, p) &: \text{crane } k \text{ unstacks a container } c \text{ from the top of pile } p,\\
\mathsf{put}(k, c, r) &: \text{crane } k \text{ holding a container } c \text{ and puts it onto } r,\\
\mathsf{take}(k, c, r) &: \text{crane } k \text{ takes container } c \text{ from robot } r.
\end{aligned}
$$

Actions

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



book omits *d*

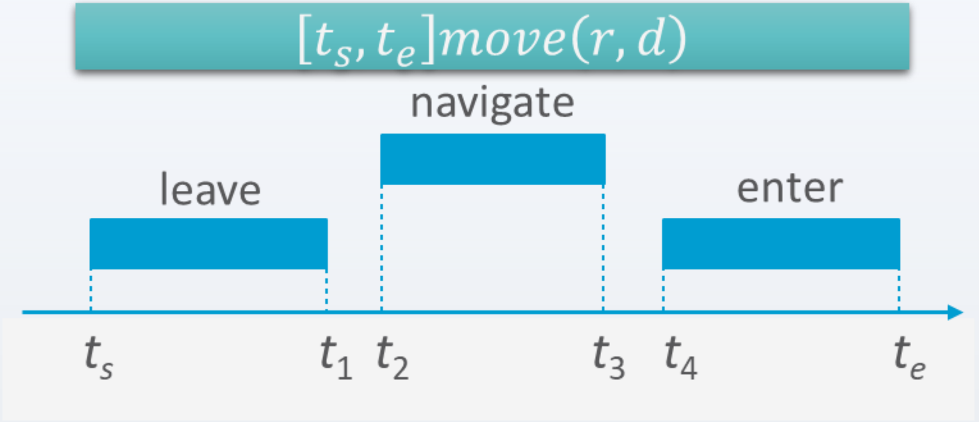
- Two additional parameters
  - Starting time t_s
  - Ending time t_e
- No separate preconditions and effects
  - Preconditions ⇔ need for causal support

```
take(k,c,r,d)
    assertions:
        [ts,te] pos(c): (r, k)           // where container c is
        [ts,te] grip(k): (empty, c)      // what crane k's gripper is holding
        [ts,te] freight(r): (c,empty)    // what r is carrying
        [ts,te] loc(r) = d               // where r is
    constraints:
        attached(k,d)
```

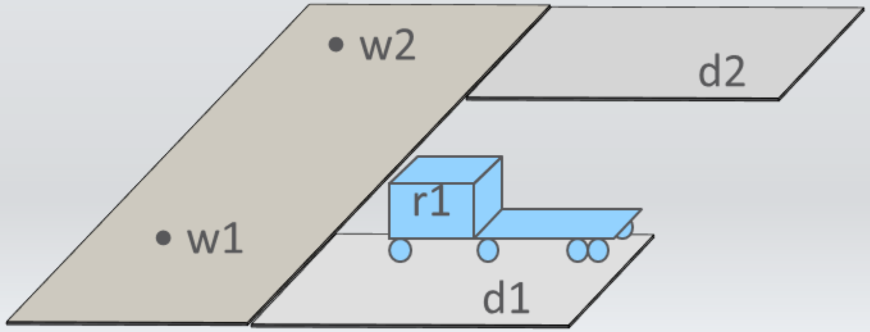
Tasks and Methods

- 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:
                [ts,t1] leave(r,d′,w′)
                [t2,t3] navigate(r,w′,w)
                [t4,te] enter(r,d,w)
    assertions:
                [ts,ts+1] loc(r) = d′
    constraints:
                adj(d,w),
                adj(d′,w′), d ≠ d′,
                connected(w,w′),
                t1 ≤ t2, t3 ≤ t4
```



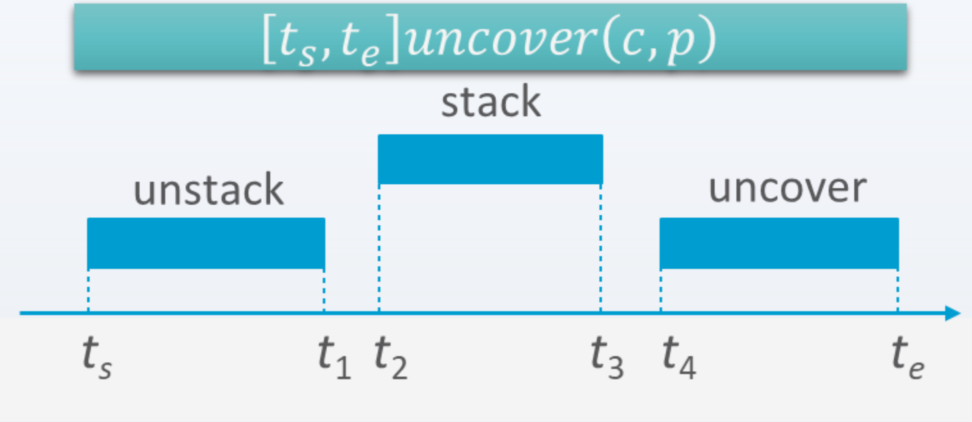
- d' becomes empty during $[t_s, t_1]$
  - another robot may enter it after t_1
- d doesn't need to be empty until t_4
  - when r starts entering it

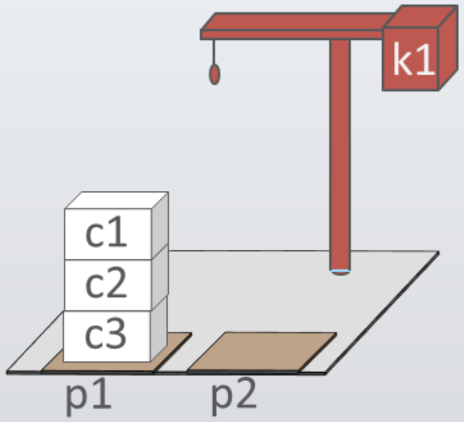


Tasks and Methods

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

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





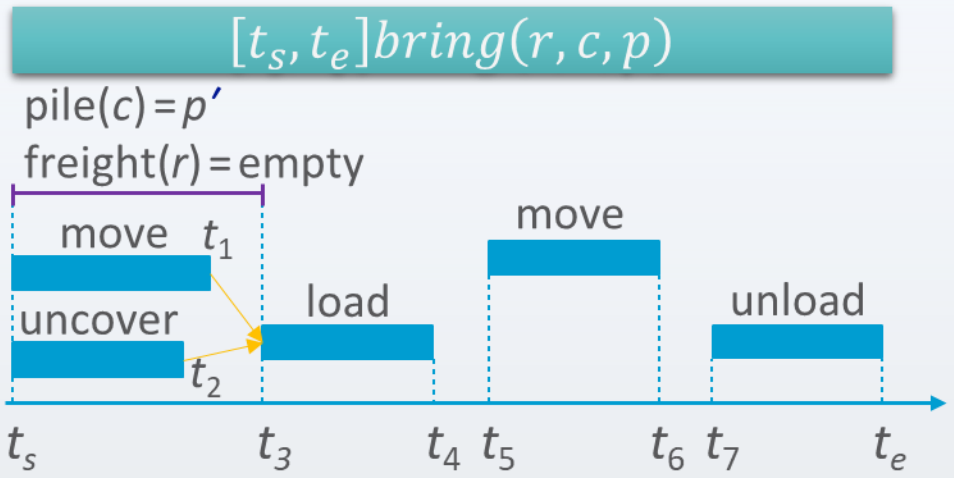
Tasks and Methods

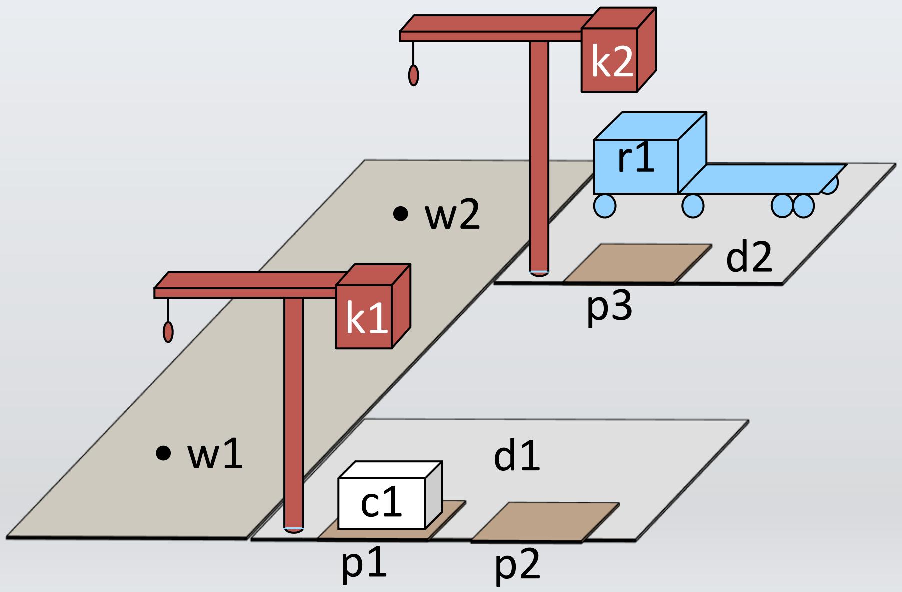
- Task: robot r brings container c to pile p
  - $[t_s, t_e] bring(r, c, p)$
- Method:

m-bring(r,c,p,p',d,d')
task: bring(r,c,p)
refinement: $[t_s,t_1]$ move(r,d')
$[t_s,t_2]$ uncover(c,p')
$[t_3,t_4]$ load(k',r,c,p')
$[t_5,t_6]$ move(r,d)
$[t_7,t_e]$ unload(k,r,c,p)
assertions: $[t_s,t_3]$ pile(c) = p'
$[t_s,t_3]$ freight(r) = empty
constraints: attached(p',d'), attached(p,d), $d \neq d'$
attached(k',d'), attached(k,d), $k \neq k'$
$t_1 \leq t_3$, $t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$

Refine into *unstack* and *put* primitives

Refine into *take* and *stack* primitives





Chronicles: Unions of Timelines

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

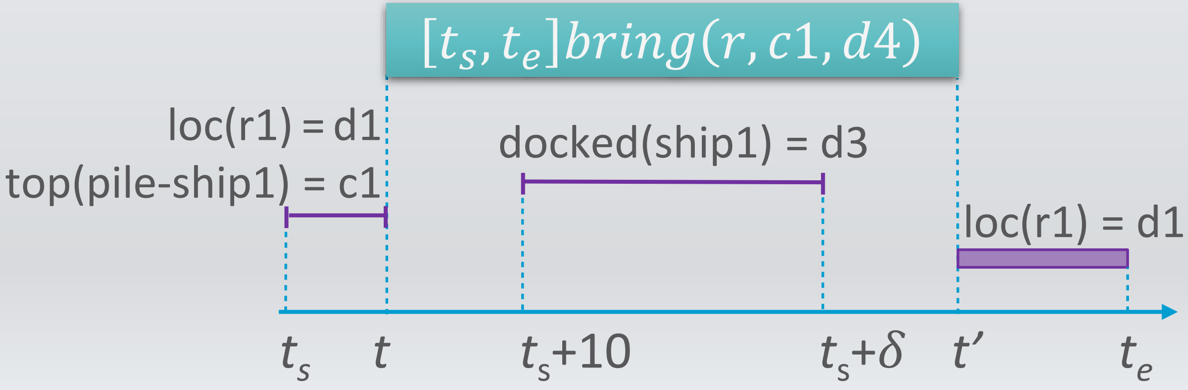
ϕ_0:

tasks: $[t,t']$ bring(r,c1,d4)

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

assertions: $[t_e]$ loc(r1)=d1
$[t_e]$ loc(r2)=d2

constraints: $t_s = 0 < t < t' < t_e$, $20 \leq \delta \leq 30$



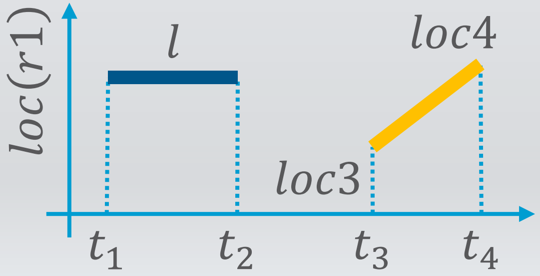
Intermediate Summary

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

Planning

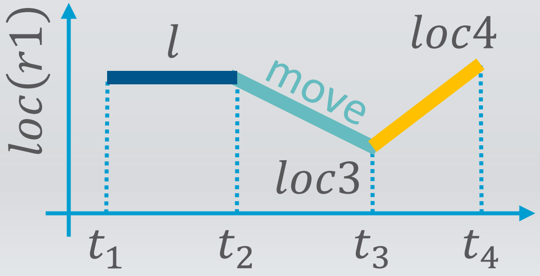
- Planning problem:
  - Chronicle ϕ_0 that has some flaws
    - Analogous to flaws in PSP

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



- Add new assertions, constraints, actions to resolve the flaws

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



Solution to Temporal Planning Problems

- Temporal Planning Problem
  - Σ : Planning Domain with objects, rigid relations, state variables, actions (primitives) and methods
  - ϕ_0 (A,S,T,C): Initial chronicle

- Planning is a refinement of tasks and generative search for goals

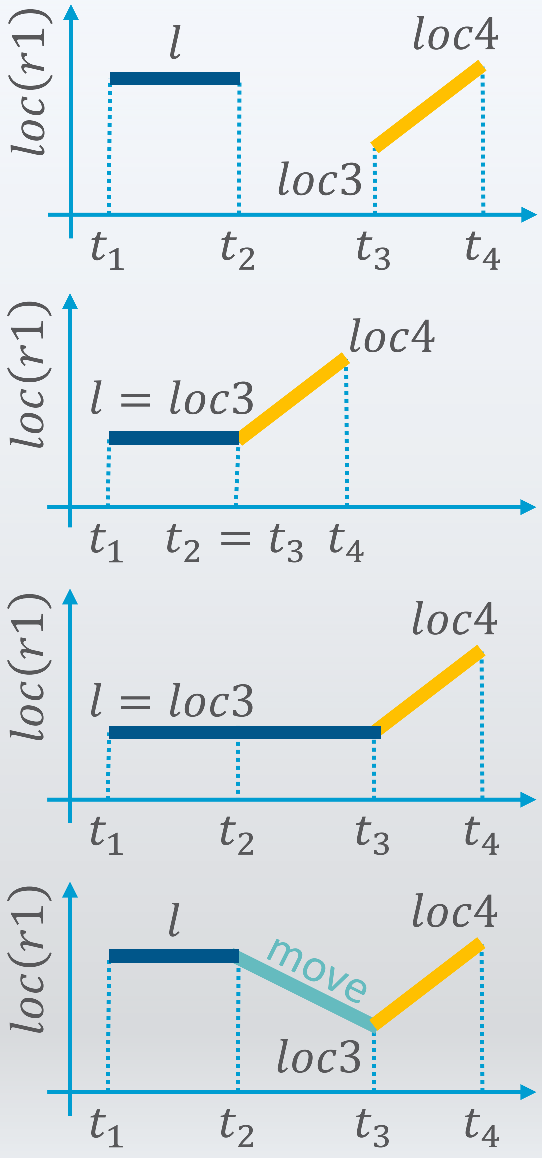
- A chronicle ϕ is a valid solution plan for the temporal planning problem if:
  - ϕ does not contain nonrefined tasks
  - All assertions in ϕ are causally supported, either by S in ϕ_0 or by assertions from methods and primitives in the plan
  - The chronicle ϕ is secure

Flaws (1)

1. **Temporal assertion α that is not *causally supported***
   - What causes $r1$ to be at $loc3$ at time t_3?

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

Like an open goal in PSP

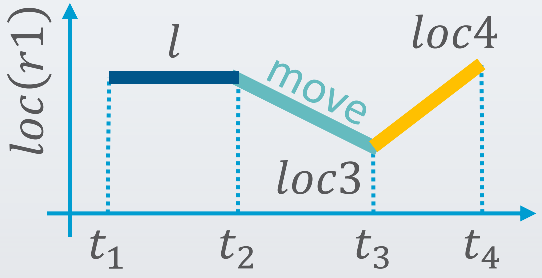


Flaws (2)

Like a task in SeRPE

2. Non-refined task

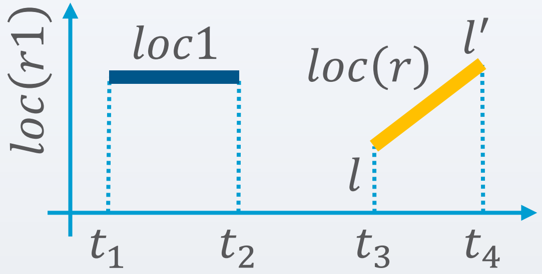
- *Resolver*: refinement method m
  - Applicable if it matches the task and its constraints are consistent with ϕ's
- Applying the resolver:
  - Modify ϕ by replacing the task with m
- Example: $[t_2, t_3]move(r1, loc3)$
  - Refinement will replace it with something like
    - $[t_2, t_5]leave(r1, l,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

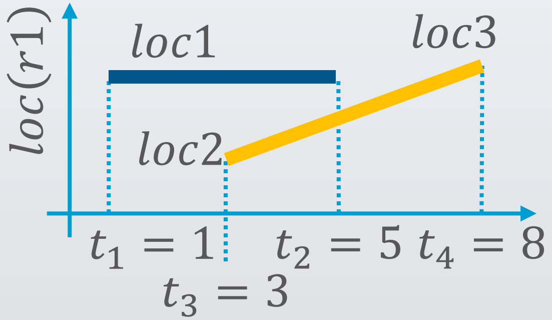


Flaws (3)

3. A pair of possibly-conflicting temporal assertions

- Temporal assertions α and β possibly conflict if they can have inconsistent instances
- Example
  - $[t_1, t_2]loc(r1) = loc1,\ [t_3, t_4]loc(r) : (l, l')$
  - ↓ ↓ ↓ ↓ ↓ ↓ ↘
  - $[1, 5]loc(r1) = loc1,\quad [3, 8]loc(r1) : (loc2, loc3)$
- *Resolvers*: separation constraints
  - $r \neq r1$
  - $t_2 < t_3$
  - $t_4 < t_1$
  - $t_2 = t_3, r = r1, l = loc1$
    - Also provides causal support for $[t_3, t_4]loc(r) :\ (l, l')$
  - $t_4 = t_1, r = r1, l' = loc1$
    - Also provides causal support for $[t_1, t_2]loc(r1) = loc1$





Planning Algorithm

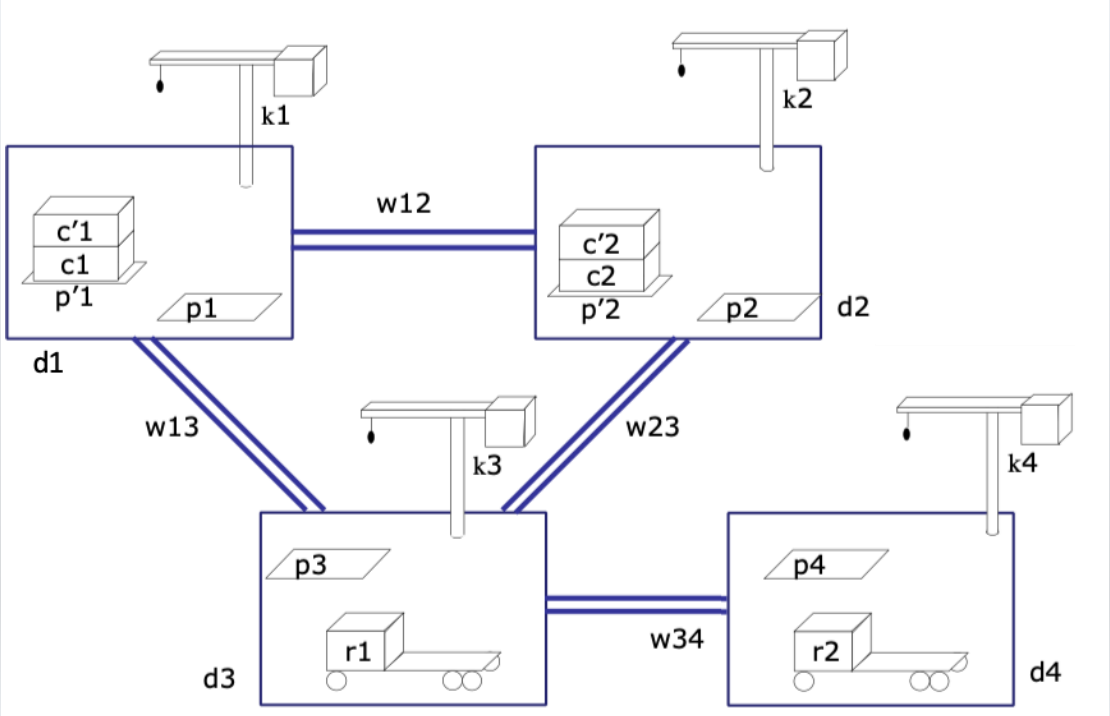
- Like PSP
  - Repeatedly selects flaws and chooses resolvers
- If resolving all flaws possible, at least one nondeterministic execution trace will do so
- In a deterministic implementation
  - Selecting a resolver ρ is a backtracking point
  - Selecting a flaw is not
  - (As in PSP)

```
TemPlan(ϕ)          // recursive version (book)
    Flaws ← set of flaws of ϕ
    if Flaws = ∅ then
        return ϕ
    arbitrarily select f ∈ Flaws
    Resolvers ← set of resolvers of f
    if Resolvers = ∅ then
        return failure
    nondeterministically choose ρ ∈ Resolvers
    ϕ ← Transform(ϕ, ρ)
    TemPlan(ϕ,Σ)
```

```
TemPlan(ϕ)          // iterative version
    loop
        Flaws ← set of flaws of ϕ
        if Flaws = ∅ then
            return ϕ
        arbitrarily select f ∈ Flaws
        Resolvers ← set of resolvers of f
        if Resolvers = ∅ then
            return failure
        nondeterministically choose ρ ∈ Resolvers
        ϕ ← Transform(ϕ, ρ)
```

Example

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



ϕ_0: tasks: bring(r,c1,p3)
bring(r',c2,p4)

supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p′1
[0] pile(c′1)=p′1
[0] pos(c1)=pallet
[0] pos(c′1)=c1
. . .

assertions: *(none)*

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

Example

- Flaws: two unrefined tasks
  - bring(r,c1,p3), bring(r',c2,p4)
- Refinement for both:

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

task: bring(r,c,p)

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

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

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

ϕ_0: tasks: bring(r,c1,p3)
bring(r',c2,p4)

supported:
[0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p′1
[0] pile(c′1)=p′1
[0] pos(c1)=pallet
[0] pos(c′1)=c1
. . .

assertions: *(none)*

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

Method Instance

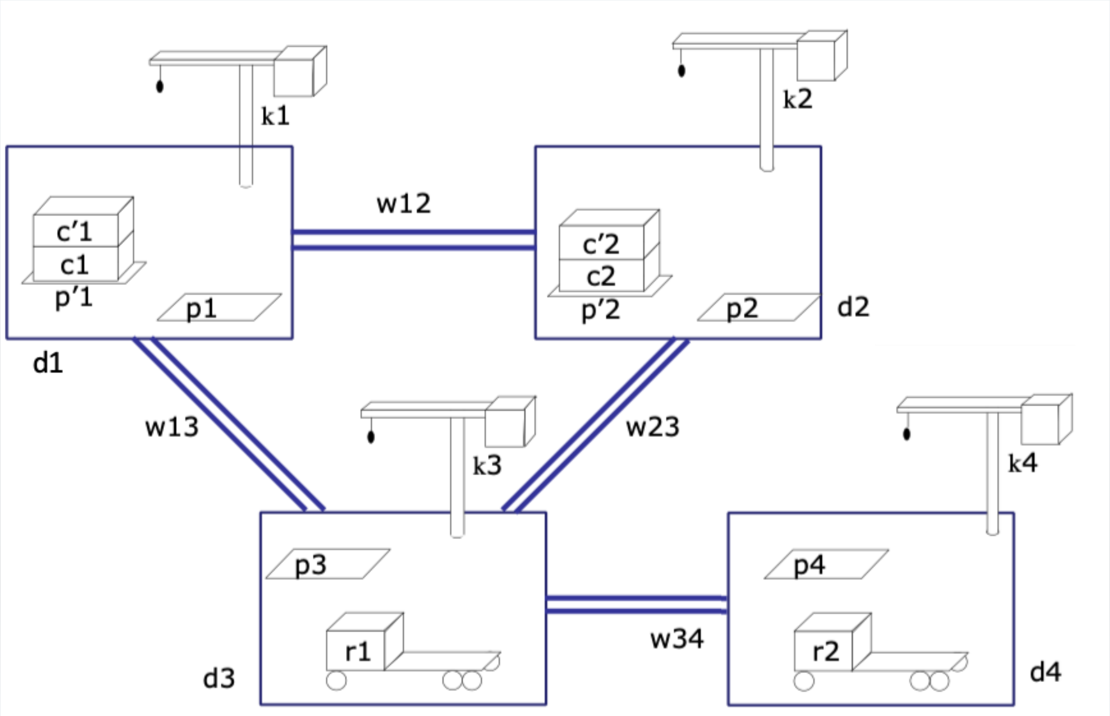
- Instantiate $c = c1$ and $p = p3$ to match $bring(r, c1, p3)$
  - p', d, d', k, k' instantiated to match book
  - Needed later to satisfy action preconditions

m-bring(*r*,c1,p3,p′1,d3,d1,k3,k1)
task: bring(*r*,c1,p3)
refinement: $[t_s,t_1]$ move(*r*,d1)
$[t_s,t_2]$ uncover(c1,p′1)
$[t_3,t_4]$ load(k1,*r*,c1,p′1)
$[t_5,t_6]$ move(*r*, d3)
$[t_7,t_e]$ unload(k3,*r*,c1,p3)
assertions: $[t_s,t_3]$ pile(c1) = p′1
$[t_s,t_3]$ freight(*r*) = empty
constraints: attached(p′1,d1),
attached(p3,d3), d3 ≠ d1
attached(k1,d1),
attached(k3,d3), k3 ≠ k1
$t_1 \leq t_3$, $t_2 \leq t_3$, $t_4 \leq t_5$, $t_6 \leq t_7$

ϕ_0: tasks: bring(*r*,c1,p3)
bring(*r*′,c2,p4)
supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p′1
[0] pile(c′1)=p′1
[0] pos(c1)=pallet
[0] pos(c′1)=c1
. . .
assertions: *(none)*
constraints:
adj(d1,w12)
adj(d1,w13)
. . .

Modified Chronicle

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



ϕ_1: tasks: $[t_s, t_1]$ move(r,d1)
$[t_s, t_2]$ uncover(c1,p'1)
$[t_3, t_4]$ load(k1,r,c1,p'1)
$[t_5, t_6]$ move(r,d3)
$[t_7, t_e]$ unload(k3,r,c1,p3)
bring(r',c2,p4)

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

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

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

Method Instance

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

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

ϕ_1: tasks: $[t_s,t_1]$ move(r,d1)
$[t_s,t_2]$ uncover(c1,p'1)
$[t_3,t_4]$ load(k1,r,c1,p'1)
$[t_5,t_6]$ move(r,d3)
$[t_7,t_e]$ unload(k3,r,c1,p3)
bring(r',c2,p4)
supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p'1
[0] pile(c'1)=p'1
[0] pos(c1)=pallet
[0] pos(c'1)=c1
. . .
assertions: $[t_s,t_3]$ pile(c1) = p'1
$[t_s,t_3]$ freight(r) = empty
constraints: $t_s<t_1\leq t_3$, $t_s<t_2\leq t_3$, $t_4\leq t_5$, $t_6\leq t_7$,
adj(d1,w12), adj(d1,w13),
. . .

Modified chronicle

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

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

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

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

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

Supporting the Assertions

- 3 ways to support
- $[t_s, t_3] pile(c1) = p'1$
  - Constrain $t_s = 0$,
    use $[0] pile(c1) = p'1$
  - Add persistence $[0, t_s] pile(c1) = p'1$
  - Add new action $[t_8, t_s] stack(k1, c1, p'1)$

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

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

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

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

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

Supporting the Assertions

- 3 ways to support
- $[t_s, t_3]pile(c1) = p'1$
  - $\boxed{\text{Constrain } t_s = 0,}$ use $[0]pile(c1) = p'1$
- To support
- $[0, t_3]freight(r) = empty$
  - $\boxed{\text{Constrain } r = r1,}$ use $[0]freight(r1) = empty$

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

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

assertions: $[\boxed{0}, t_3]$ pile(c1) = p'1
$[\boxed{0}, t_3]$ freight(r) = empty
$[t'_s,t'_3]$ pile(c2) = p'2
$[t'_s,t'_1]$ freight(r') = empty

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

Supporting the Assertions

- 3 ways to support
- $[t_s, t_3]pile(c1) = p'1$
  - $\boxed{\text{Constrain } t_s = 0,}$
    use $[0]pile(c1) = p'1$
- To support
- $[0, t_3]freight(r) = empty$
  - $\boxed{\text{Constrain } r = r1,}$
    use $[0]freight(r1) = empty$

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

supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p'1
. . .
$[0,t_3]$ pile(c1) = p'1
$[0,t_3]$ freight($\boxed{\text{r1}}$) = empty

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

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

Supporting the Assertions

- To support
- $[t'_s, t'_3]pile(c2) = p'2$
  - Add persistence condition $\boxed{[0, t'_s]pile(c2) = p'2}$
  - Constrain $t'_s = 0$
  - Add new action $stack(k2, c2, p'2)$

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

supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p'1
. . .
$[0,t_3]$ pile(c1) = p'1
$[0,t_3]$ freight(r1) = empty

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

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

Supporting the Assertions

- To support
- $[t'_s, t'_3] pile(c2) = p'2$
  - Add persistence condition $\boxed{[0, t'_s] pile(c2) = p'2}$
- To support
- $[t'_s, t'_1] freight(r') = empty$
  - $\boxed{\text{Constrain } r' = r2,}$ add persistence condition $\boxed{[0, t'_s] freight(r2) = empty}$

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

supported: [0] loc(r1)=d3
[0] freight(r1)=empty
[0] pile(c1)=p'1 . . .
$[0,t_3]$ pile(c1) = p'1
$[0,t_3]$ freight(r1) = empty
$\boxed{[0,t'_s] \text{ pile(c2)=p'2}}$
$[t'_s,t'_3]$ pile(c2) = p'2

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

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

Supporting the Assertions

- To support
- $[t'_s, t'_3]pile(c2) = p'2$
  - Add persistence condition $\boxed{[0, t'_s]pile(c2) = p'2}$
- To support
- $[t'_s, t'_1]freight(r') = empty$
  - $\boxed{\text{Constrain } r' = r2,}$ add persistence condition $\boxed{[0, t'_s]freight(r2) = empty}$

- All assertions currently supported
- Remaining flaws: unrefined tasks

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

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

assertions: *(none)*

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

Example of Conflicts

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

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

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

assertions: *(none)*

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

Heuristics for Guiding TemPlan

- Flaw selection, resolver selection heuristics similar to those in PSP
  - Select the flaw with the smallest number of resolvers
  - Choose the resolver that rules out the fewest resolvers for the other flaws
- There is also a problem with constraint management
  - We ignored it when discussing PSP
  - We discuss it next

```
TemPlan(ϕ)
    Flaws ← set of flaws of ϕ
    if Flaws = ∅ then
        return ϕ
    arbitrarily select f ∈ Flaws
    Resolvers ← set of resolvers of f
    if Resolvers = ∅ then
        return failure
    nondeterministically choose ρ ∈ Resolvers
    ϕ ← Transform(ϕ, ρ)
    TemPlan(ϕ)
```

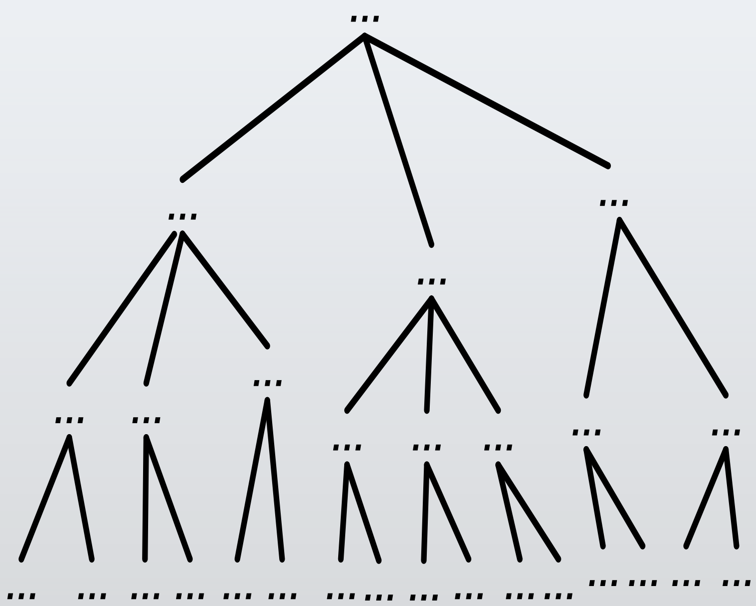
```
PSP(Σ,π)
    loop
        if Flaws(π) = ∅ then
            return π
        arbitrarily select f ∈ Flaws(π)
        R ←{all feasible resolvers for f}
        if R = ∅ then
            return failure
        nondeterministically choose ρ ∈ R
        π ← ρ(π)
    return π
```

Intermediate Summary

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

Constraint Management

- Each time TemPlan applies a resolver, it modifies $(\mathcal{T}, \mathcal{C})$
  - Some resolvers will make $(\mathcal{T}, \mathcal{C})$ inconsistent
    - No solution in this part of the search space
    - Detect inconsistency $\rightarrow$ prune this part of the search space
    - Do not detect it $\rightarrow$ waste time looking for a solution
- Analogy: PSP checks simple cases of inconsistency
  - E.g., cannot create a constraint $a \prec b$
    if there is already a constraint $b \prec a$
  - Ignores more complicated cases
  - Example:
    - $c_1, c_2, c_3 \in Containers = \{c1, c2\}$
    - Threats involving c_1, c_2, c_3
    - For resolvers, suppose PSP chooses
      - $c_1 \neq c_2, c_2 \neq c_3, c_1 \neq c_3$
    - No solutions in this part of the search space,
      but PSP searches it anyway



Constraint Management in TemPlan

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

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

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

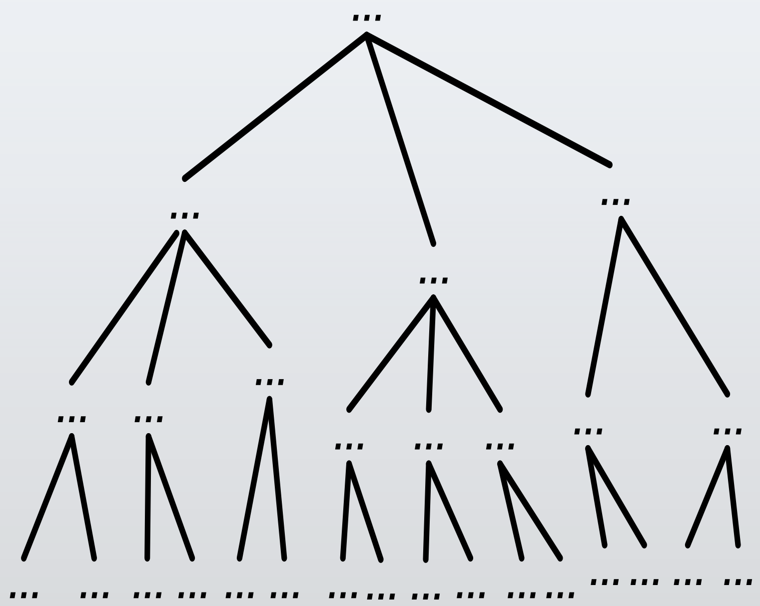
Consistency of $\mathcal{C}$

- $\mathcal{C}$ contains two kinds of constraints
  - Object constraints
    - $loc(r) \neq l_2, \quad l \in \{loc3, loc4\}, \quad r = r1, \; o \neq o'$
  - Temporal constraints
    - $t_1 < t_3, \quad a < t, \quad t < t', \quad a \leq t' - t \leq b$
  - Assume object constraints are independent of temporal constraints and vice versa
    - Exclude things like $t < f(l, r)$ with some function f

- Then two separate subproblems:
    - Check consistency of object constraints
    - Check consistency of temporal constraints
  - $\mathcal{C}$ is consistent iff both are consistent

Object Constraints

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

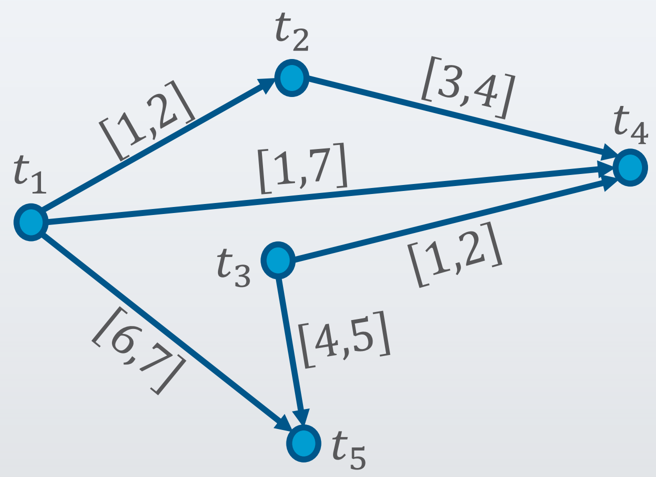


Time Constraints: Representation

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

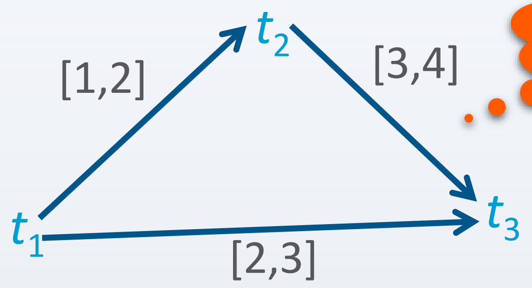
- Synthesise an STN incrementally starting from ϕ_0
  - TemPlan can check time constraints in time $O(n^3)$

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

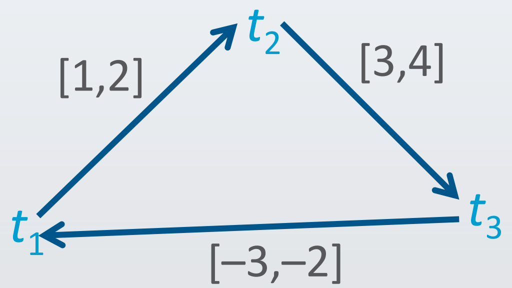


Simple Temporal Networks

- STN: a pair $(\mathcal{V}, \mathcal{E})$, where
  - $\mathcal{V} = \{$a set of temporal variables $t_1, \ldots, t_n\}$
  - $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges
- Each edge (t_i, t_j) is labelled with an interval $[a, b]$
  - Shorthand: represents constraint $a \leq t_j - t_i \leq b$
  - Equivalently, $-b \leq t_i - t_j \leq -a$
- Representing unary constraints
  - Dummy variable $t_0 = 0$
  - Edge (t_0, t_i) labelled with $[a, b]$ represents $a \leq t_i - 0 \leq b$
- Solution to an STN
  - Integer value for each t_i
  - All constraints satisfied
- Consistent STN
  - Has a solution



Is this network consistent?

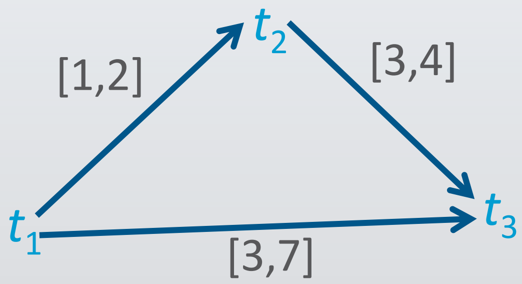


Book says:
- Solution
  - Integer value for each t_i
- Consistent:
  - Has a solution
  - *All constraints satisfied*

!

Time Constraints

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

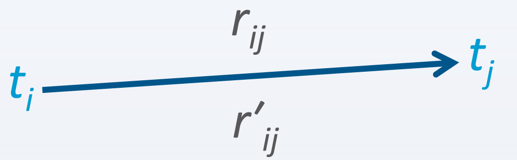


Operations on STNs

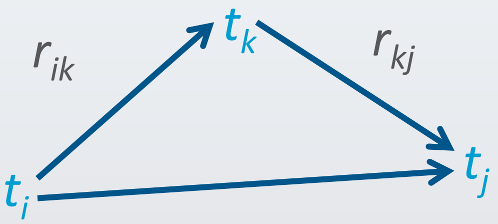
- Intersection, $\cap$
  - $t_j - t_i \in r_{ij} = [a_{ij}, b_{ij}]$
  - $t_j - t_i \in r'_{ij} = [a'_{ij}, b'_{ij}]$
  - Infer

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

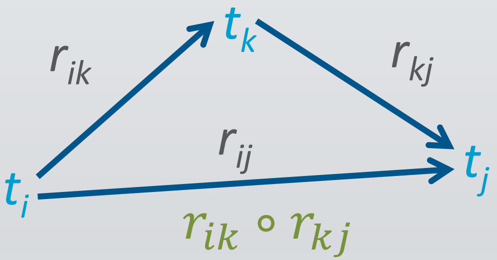
    $$t_j - t_i \in r_{ik} \circ r_{kj} = [a_{ik} + a_{kj}, b_{ik} + b_{kj}]$$
    - Reasoning: add up shortest and longest times
- Consistency checking
  - Three constraints r_{ik}, r_{kj}, r_{ij} are consistent only if $r_{ij} \cap (r_{ik} \circ r_{kj}) \neq \emptyset$ (*empty interval*)



$r_{ij} \cap r'_{ij}$

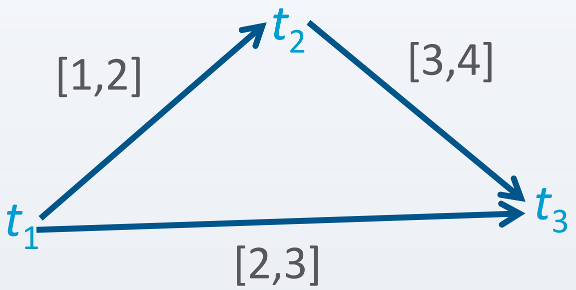


$r_{ik} \circ r_{kj}$

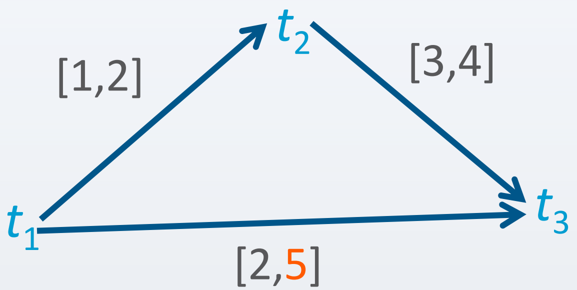


$r_{ij} \cap (r_{ik} \circ r_{kj})$

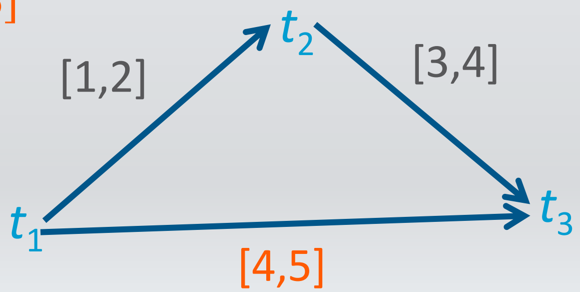
Two Examples



- STN $(\mathcal{V}, \mathcal{E})$, where
  - $\mathcal{V} = \{t_1, t_2, t_3\}$
  - $\mathcal{E} = \{r_{12} = [1,2], r_{23} = [3,4], r_{13} = [2,3]\}$
- Composition
  - $r'_{13} = r_{12} \circ r_{23} = [1,2] \circ [3,4] = [4,6]$
- Cannot satisfy both r_{13} and r'_{13}
  - $r_{13} \cap r'_{13} = [2,3] \cap [4,6] = \emptyset$
- $(\mathcal{V}, \mathcal{E})$ is inconsistent



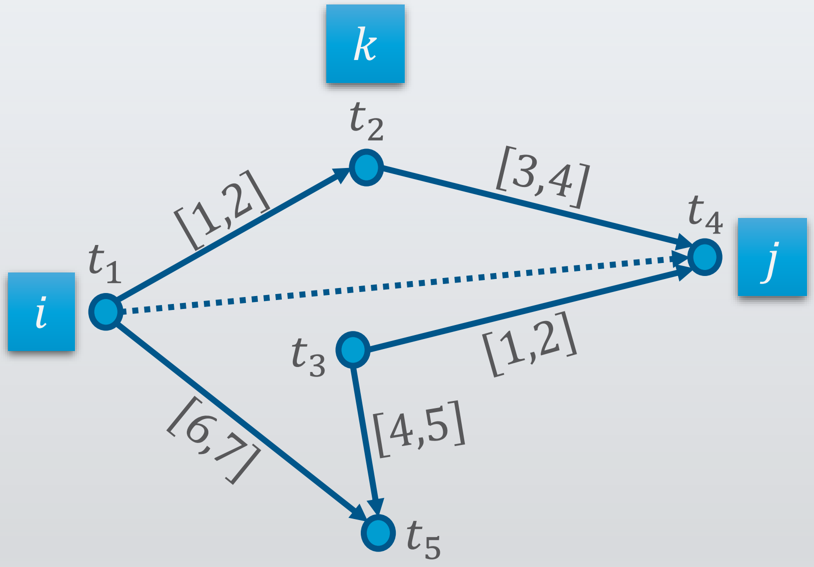
- STN $(\mathcal{V}, \mathcal{E})$, where
  - $\mathcal{V} = \{t_1, t_2, t_3\}$
  - $\mathcal{E} = \{r_{12} = [1,2], r_{23} = [3,4], r_{13} = [2,5]\}$
- Composition (as before)
  - $r'_{13} = r_{12} \circ r_{23} = [4,6]$
- $(\mathcal{V}, \mathcal{E})$ is consistent
  - $r_{13} \cap r'_{13} = [2,5] \cap [4,6] = [4,5]$
- Minimal network
  - $r_{13} = [4,5]$



Operations on STNs

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

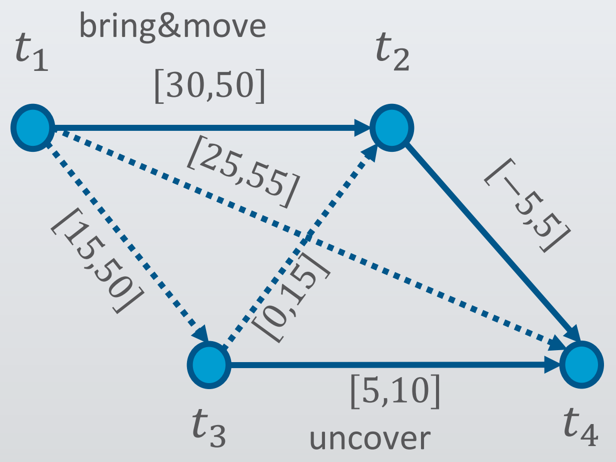
```
PC(𝒱,ℰ)
    for 1 ≤ k ≤ n do
        for 1 ≤ i < j ≤ n, i ≠ j, j ≠ k do
            r_ij ← r_ij ∩ [r_ik ∘ r_kj]
            if r_ij = ∅ then
                return inconsistent
    return consistent
```



Operations on STNs

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

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



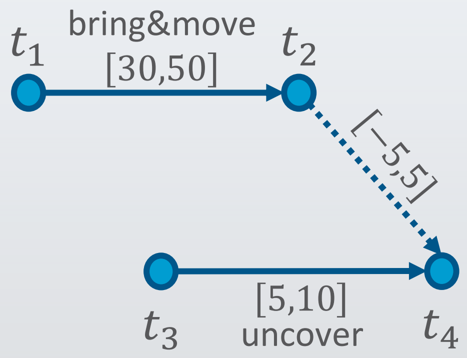
Pruning TemPlan's search space

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

Controllability

UNIVERSITÄT ZU LÜBECK

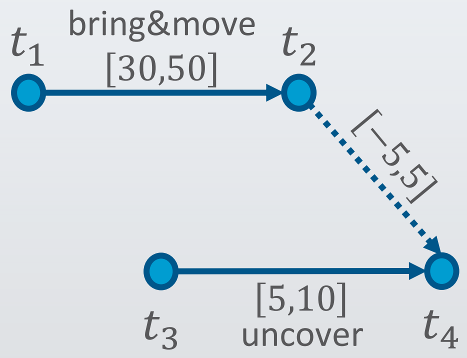
- Suppose TemPlan gives you a chronicle and you want to execute it
  - Constraints on time points
  - Need to reason about these to decide when to start each action



Controllability

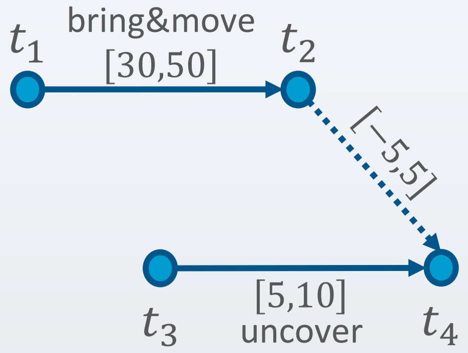
- Solid lines: duration constraints
  - Robot will do bring&move, will take 30 to 50 time units
  - Crane will do uncover, will take 5 to 10 time units
- Dashed line: synchronization constraint
  - Do not want either the crane or robot to wait long
  - At most 5 seconds between the two ending times

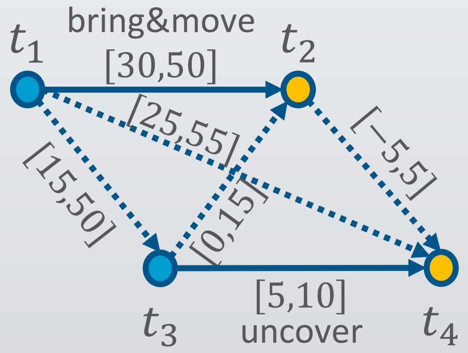
- Objective
  - Choose time points that will satisfy all the constraints



Controllability

- Suppose we run PC
- PC returns a minimal and consistent network
  - There *exist* time points that satisfy all the constraints
- Would work if we could choose all four time points
  - But we cannot choose t_2 and t_4
- t_1 and t_3 are controllable
  - Actor can control when each action starts
- t_2 and t_4 are contingent
  - Cannot control how long the actions take
  - Random variables that are known to satisfy the duration constraints
    - $t_2 \in [t_1 + 30, t_1 + 50]$
    - $t_4 \in [t_3 + 5, t_3 + 10]$





STNUs

- STNU (Simple Temporal Network with Uncertainty):
  - A 4-tuple $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$
    - $\mathcal{V}$ ={controllable time points}
      - E.g., starting times of actions
    - $\tilde{\mathcal{V}}$ ={contingent time points}
      - E.g., ending times of actions
    - $\mathcal{E}$ ={controllable constraints}
    - $\tilde{\mathcal{E}}$ ={contingent constraints}
- Controllable and contingent constraints:
  - Synchronization between two starting times: *controllable*
  - Duration of an action: *contingent*
  - Synchronization between ending points of two actions: *contingent*
  - Synchronization between end of one action, start of another:
    - *Controllable* if the new action starts after the old one ends
    - *Contingent* if the new action starts before the old one ends
- Want a way for the actor to choose time points in $\mathcal{V}$ (starting times) that guarantee that constraints are satisfied

Three kinds of controllability

- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is strongly controllable if the actor can choose values for $\mathcal{V}$ such that success will occur for all values of $\tilde{\mathcal{V}}$ that satisfy $\tilde{\mathcal{E}}$
  - Actor can choose the values for $\mathcal{V}$ offline
  - The right choice will work regardless of $\tilde{\mathcal{V}}$
- $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is weakly controllable if the actor can choose values for $\mathcal{V}$ such that success will occur for *at least one* combination of values for $\tilde{\mathcal{V}}$
  - Actor can choose the values for $\mathcal{V}$ only if the actor knows in advance what the values of $\tilde{\mathcal{V}}$ will be
- Dynamic controllability:
  - Game-theoretic model: actor vs. environment
  - A player's strategy: a function σ telling what to do in every situation
    - Choices may differ depending on what has happened so far
  - $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is dynamically controllable if $\exists$ strategy for an actor that will guarantee success regardless of the environment's strategy

Dynamic Execution

UNIVERSITÄT ZU LÜBECK

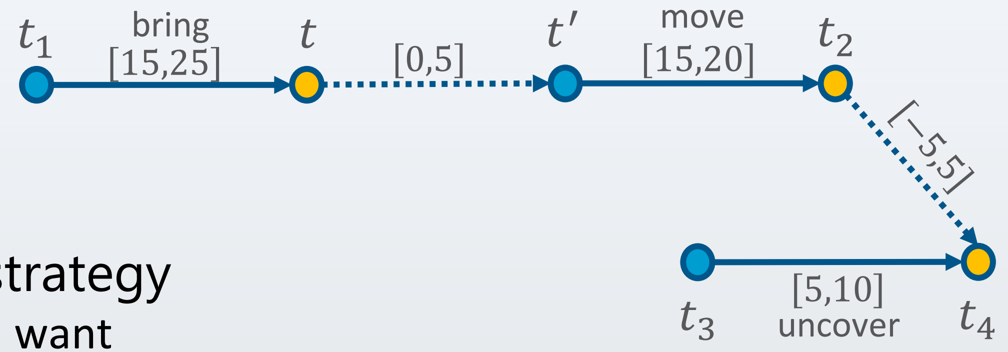
- For $t = 0, 1, 2, \ldots$
  - Actor chooses an unassigned set of variables $\mathcal{V}_t \subseteq \mathcal{V}$ that all can be assigned the value t without violating any constraints in $\mathcal{E}$
  - ≈ actions the actor chooses to start at time t
  - Simultaneously, environment chooses an unassigned set of variables $\tilde{\mathcal{V}}_t \subseteq \tilde{\mathcal{V}}$ that all can be assigned the value t without violating any constraints in $\tilde{\mathcal{E}}$
  - ≈ actions that finish at time t
  - Each chosen time point v is assigned $v \leftarrow t$
  - Failure if any of the constraints in $\mathcal{E} \cup \tilde{\mathcal{E}}$ are violated
  - There might be violations that neither $\mathcal{V}_t$ nor $\tilde{\mathcal{V}}_t$ caused individually
  - Success if all variables in $\mathcal{V} \cup \tilde{\mathcal{V}}$ have values and no constraints are violated

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

- Dynamic execution strategies 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 $\tilde{\mathcal{V}}$ to trigger at time t, given h_{t-1}}
    - $h_t = h_{t-1} \cdot \left(\sigma_A(h_{t-1}) \cup \sigma_E(h_{t-1})\right)$
- $\left(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}}\right)$ is dynamically controllable if $\exists\, \sigma_A$ that will guarantee success $\forall\, \sigma_E$

Example

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



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

Dynamic Controllability Checking

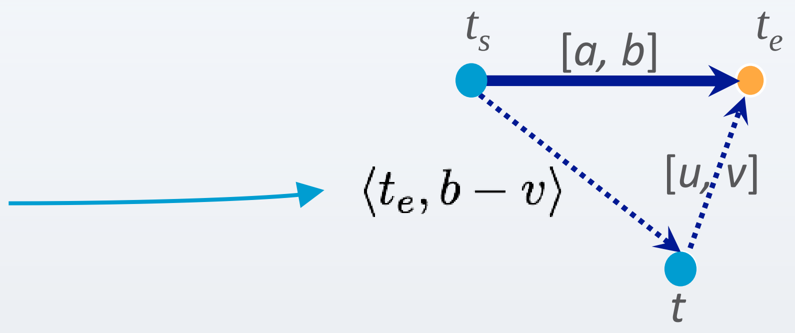
- For a chronicle $\phi = (\mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{C})$
  - Temporal constraints in $\mathcal{C}$ correspond to an STNU
  - Adapt TemPlan to test not only consistency but also dynamic controllability (*) of the STNU
  - If we detect cases where it is not dynamically controllable, then backtrack

*Use PC as well

- If PC($\mathcal{V} \cup \tilde{\mathcal{V}}, \mathcal{E} \cup \tilde{\mathcal{E}}$) reduces a contingent constraint, then $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is not dynamically controllable
  $\Rightarrow$ Can prune this branch
- If it *does not* reduce any contingent constraints, we do not know whether $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is dynamically controllable
  - Only necessary, not sufficient condition
- Two options
  - Either continue down this branch and backtrack later if necessary, or
  - Extend PC to detect more cases where $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ is not dynamically controllable
    - Additional constraint propagation rules

Additional Constraint Propagation Rules

- Case 1: $u \geq 0$
  - t must come before t_e
- Add a composition constraint $[a', b']$
  - Find $[a', b']$ such that $[a', b'] \circ [u, v] = [a, b]$
    - $[a' + u, b' + v] = [a, b]$
    - $a' = a - u, b' = b - v$

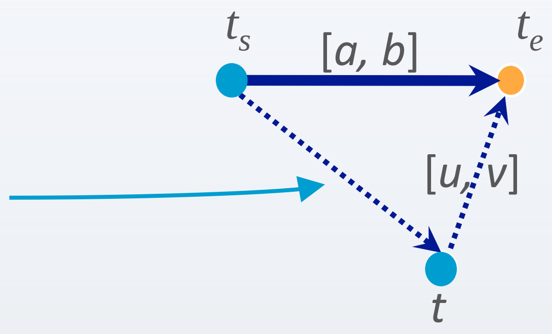


Conditions	Propagated constraint
$t_s \xRightarrow{[a,b]} t_e\ ,\ t \xrightarrow{[u,v]} t_e\ ,\ u \geq 0$	$t_s \xrightarrow{[b',a']} t$
$t_s \xRightarrow{[a,b]} t_e\ ,\ t \xrightarrow{[u,v]} t_e\ ,\ u < 0\ ,\ v \geq 0$	$t_s \xrightarrow{\langle t_e,b' \rangle} t$
$t_s \xRightarrow{[a,b]} t_e\ ,\ t_s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

$\Longrightarrow$ contingent $\longrightarrow$ controllable $\quad a' = a - u, b' = b - v$

Additional Constraint Propagation Rules

- Case 2: $u < 0$ and $v \geq 0$
  - t may be before or after t_e
- Add a wait constraint $\langle t_e, \alpha \rangle$
  - α defined w.r.t. some controllable time point t_s
  - Wait until either t_e occurs or current time is $t_s + \alpha$, whichever comes first



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

$\Longrightarrow$ contingent $\longrightarrow$ controllable $\quad a' = a - u, b' = b - v$

Extended Version of PC

- We want a fast algorithm that TemPlan can run at each node, to decide whether to backtrack
- There is an extended version of PC that runs in polynomial time, but it has high overhead
- Possible compromise: use ordinary PC most of the time
  - Run extended version occasionally, or at end of search before returning plan

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

$\Longrightarrow$ contingent $\longrightarrow$ controllable $\qquad a' = a - u, b' = b - v$

Intermediate Summary

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

Acting with Temporal Models

Atemporal Refinement of Primitive Actions

- TemPlan's action templates may correspond to compound tasks
  - In RAE, refine into commands with refinement methods
  - TemPlan's action template (descriptive model)

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

  - RAE's refinement method (operational model)

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

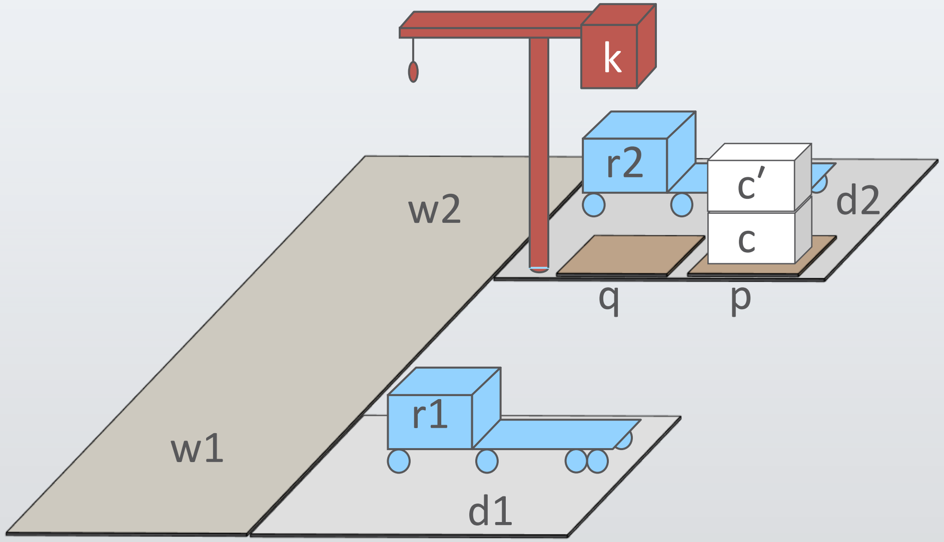
Discussion

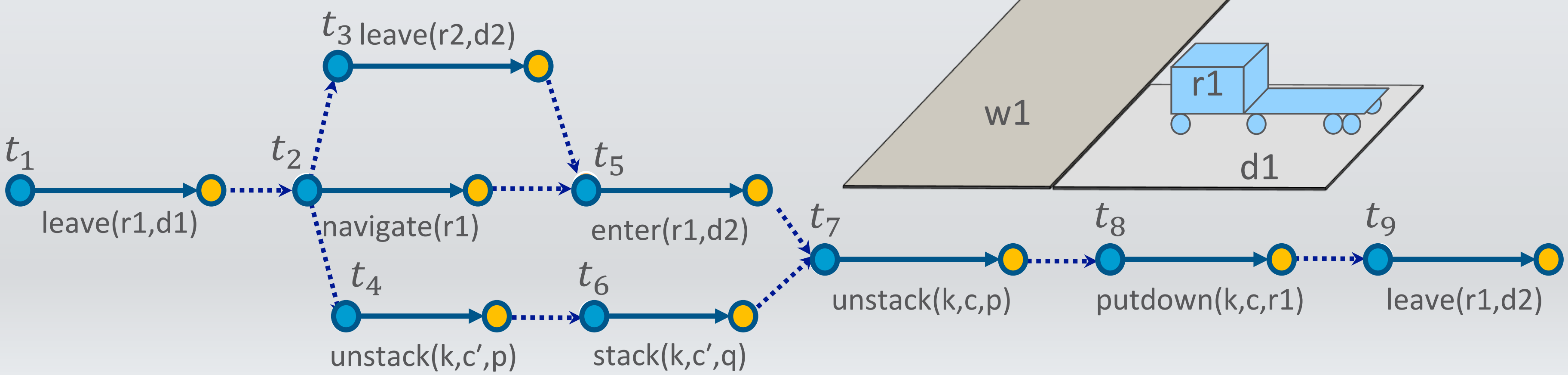
- Pros
  - Simple online refinement with RAE
  - Avoids breaking down uncertainty of contingent duration
  - Can be augmented with temporal monitoring functions in RAE
    - E.g., watchdogs, methods with duration preferences
- Cons
  - Does not handle temporal requirements at the command level,
    -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

- Can augment RAE to include temporal reasoning
  - Call it eRAE
  - One essential component: a dispatching function

Acting With Temporal Models

- Dispatching procedure: a dynamic execution strategy
  - Controls when to start each action
  - Given a dynamically controllable plan with executable primitives, it triggers corresponding commands from online observations
- Example
  - robot $r2$ needs to leave dock $d2$ before robot $r1$ can enter $d2$
  - crane k needs to uncover c then put c onto $r1$

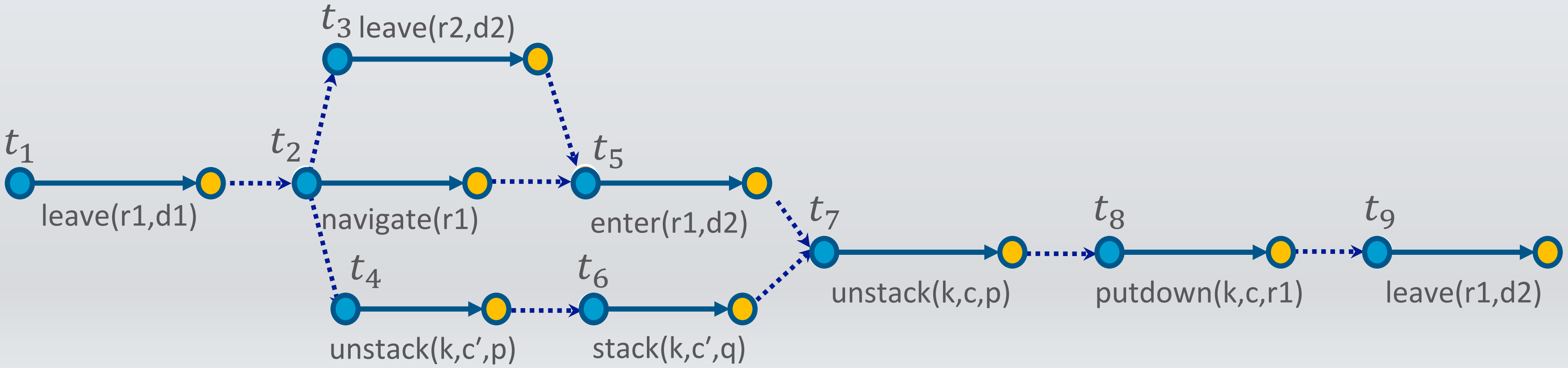




Example

- Trigger t_1, observe leave finish
- Enable and trigger t_2, enables t_3, t_4
- Trigger t_3 soon enough to allow $enter(r1, d2)$ at time t_5
- Trigger t_4 soon enough to allow $stack(k, c')$ at time t_6
- Rest of plan is linear:
  - Choose each t_i after the previous action ends

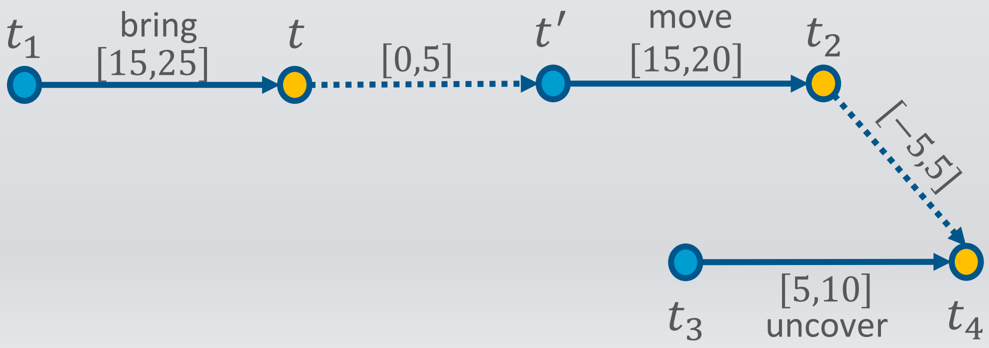
Dispatch($\mathcal{V}$, $\tilde{V}$, $\mathcal{E}$, $\tilde{E}$)
initialise the network
while there are time points in $\mathcal{V}$ that have not been triggered **do**
update *now*
update the time points in $\tilde{V}$ that have been newly observed
update *enabled*
trigger every $t \in$ *enabled* s.t.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



Previous Example

- Trigger t_1 at time 0
- Wait and observe t; this enables t'
- Trigger t' at any time from t to $t + 5$
- Trigger t_3 at time $t' + 10$
  - $t_2 \in [t' + 15, t' + 20]$
  - $t_4 \in [t_3 + 5, t_3 + 10] = [t' + 15, t' + 20]$
  - so $t_4 - t_2 \in [-5, 5]$

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



Dispatching

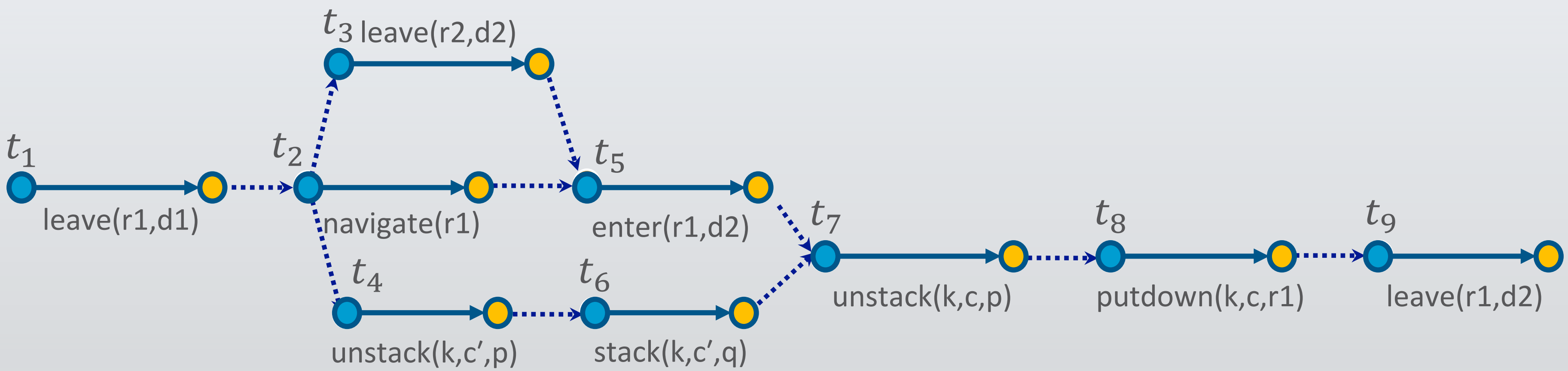
- Let $(\mathcal{V}, \tilde{\mathcal{V}}, \mathcal{E}, \tilde{\mathcal{E}})$ be a controllable STNU that is grounded
  - Different from a grounded expression in logic
  - At least one time point t^* is instantiated
    - Bounds each time point t within an interval $[l_t, u_t]$

- Controllable time point t in the future:
  - t is alive if current time $now \in [l_t, u_t]$
  - t is enabled if
    - It is alive
    - For every precedence constraint $t' < t$, t' has occurred
    - For every wait constraint $\langle t_e, \alpha \rangle$, t_e has occurred or α has expired
      - α has expired if t_s has occurred and $t_s + \alpha \leq now$

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

Deadline Failures

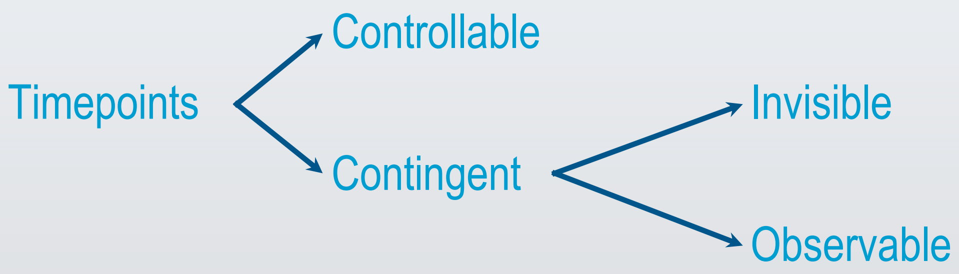
- Suppose something makes it impossible to start an action on time
- Do one of the following:
  - Stop the delayed action, and look for new plan
  - Let the delayed action finish, try to repair the plan by resolving violated constraints at the STNU propagation level
    - E.g., accommodate a delay in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



Partial Observability

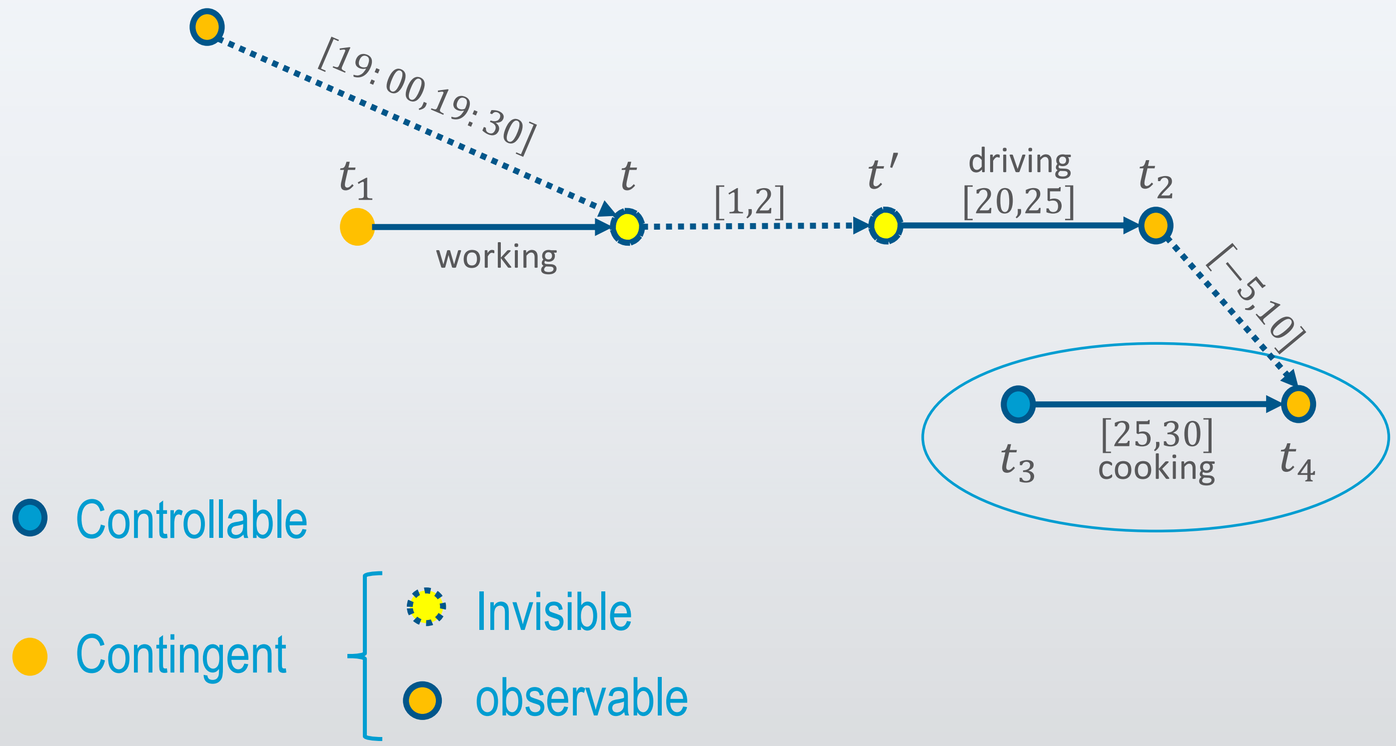
UNIVERSITÄT ZU LÜBECK

- Tacit assumption: All occurrences of contingent events are observable
  - Observation needed for dynamic controllability
- In general, not all events are observable
- POSTNU (Partially Observable STNU)
  - STNU where the contingent time points are given by a set of invisible and a set of observable timepoints
    - POSTNU = STNU if Invisible = ∅
  - Dynamically controllable?



Observation Actions

Example

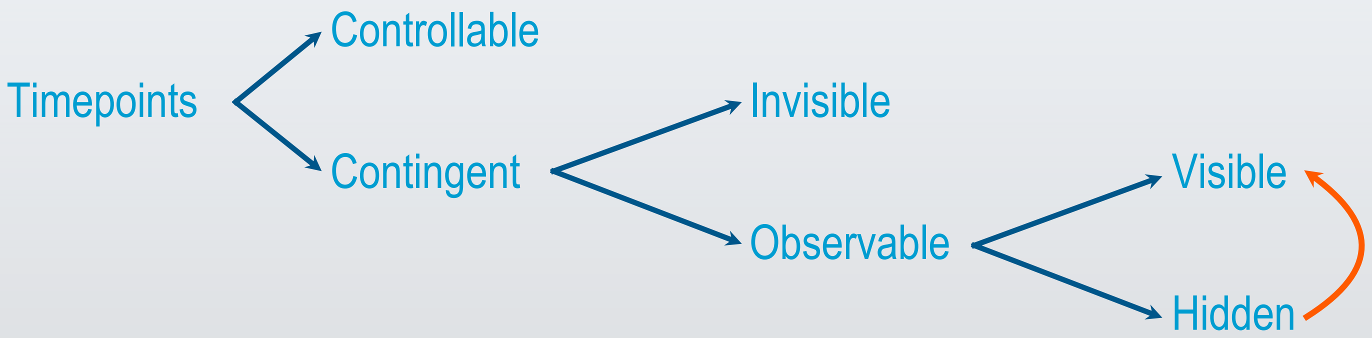


Dynamic Controllability

- A POSTNU is dynamically controllable if
  - there exists an execution strategy that chooses future controllable points to meet all the constraints, given the observation of past *visible* points
- Check dynamic controllability
  - Map an POSTNU to an STNU by deleting invisible time points and adding corresponding constraints on controllable and observable time points
  - Check dynamic controllability of the mapped STNU
    - E.g., using the extended PC algorithm
  - More details in the paper

Dynamic Controllability

- A POSTNU is dynamically controllable if
  - there exists an execution strategy that chooses future controllable points to meet all the constraints, given the observation of past *visible* points
- Observable ≠ visible
  - Observable means it will be known when observed
    - It can be temporarily hidden



- Aim: Find out which time points need to be observed for the plan to be dynamically controllable (details in paper)

Intermediate Summary

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

Content

1. Planning and Acting with **Deterministic** Models
   Conventional AI planning
2. Planning and Acting with **Refinement** Methods
   Abstract activities → collections of less-abstract activities
3. Planning and Acting with **Temporal** Models
   Reasoning about time constraints
4. Planning and Acting with **Nondeterministic** Models
   Actions with multiple possible outcomes
5. **Standard** Decision Making
   Utility theory
   Markov decision process (MDP)
6. Planning and Acting with **Probabilistic** Models
   Actions with multiple possible outcomes, with probabilities
7. **Advanced** Decision Making
   Hidden goals
   Partially observable MDP (POMDP)
   Decentralised POMDP
8. **Human-aware** Planning
   Planning with a human in the loop
9. **Causal** Planning
   Causality & Intervention
   Implications for Causal Planning