Lecture slides for

Automated Planning: Theory and Practice

Chapter 14
Temporal Planning

Dana S. Nau

University of Maryland

2:19 PM April 23, 2012

Temporal Planning

- Motivation: want to do planning in situations where actions
  - have nonzero duration
  - may overlap in time
- Need an explicit representation of time

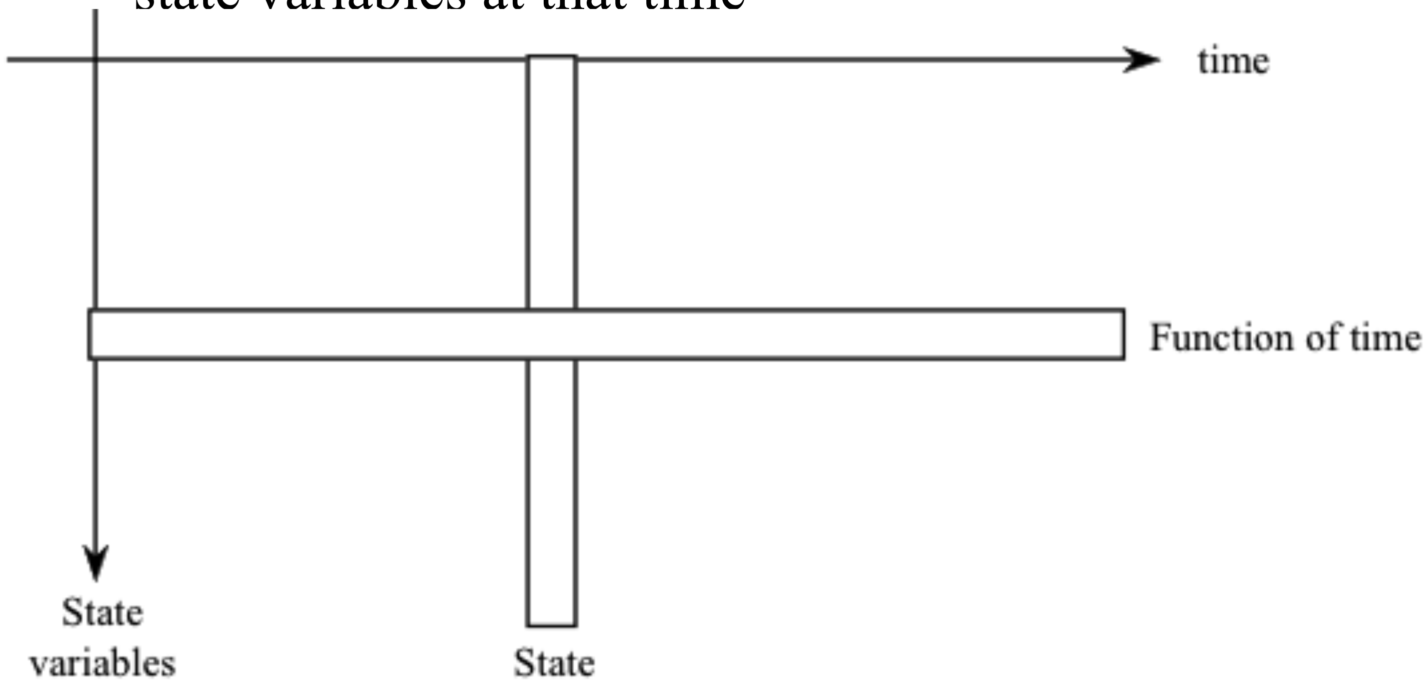
- In Chapter 10 we studied a "temporal" logic
  - Its notion of time is too simple: a sequence of discrete events
  - Many real-world applications require continuous time
  - How to get this?

Temporal Planning

- The book presents two equivalent approaches:
  1. Use logical atoms, and extend the usual planning operators to include temporal conditions on those atoms
     - » Chapter 14 calls this the "state-oriented view"
  2. Use state variables, and specify change and persistence constraints on the state variables
     - » Chapter 14 calls this the "time-oriented view"
- In each case, the chapter gives a planning algorithm that's like a temporal-planning version of PSP

The Time-Oriented View

- We'll concentrate on the "time-oriented view": Sections 14.3.1–14.3.3
  - It produces a simpler representation
  - State variables seem better suited for the task
- States not defined explicitly
  - Instead, can compute a state for any time point, from the values of the state variables at that time



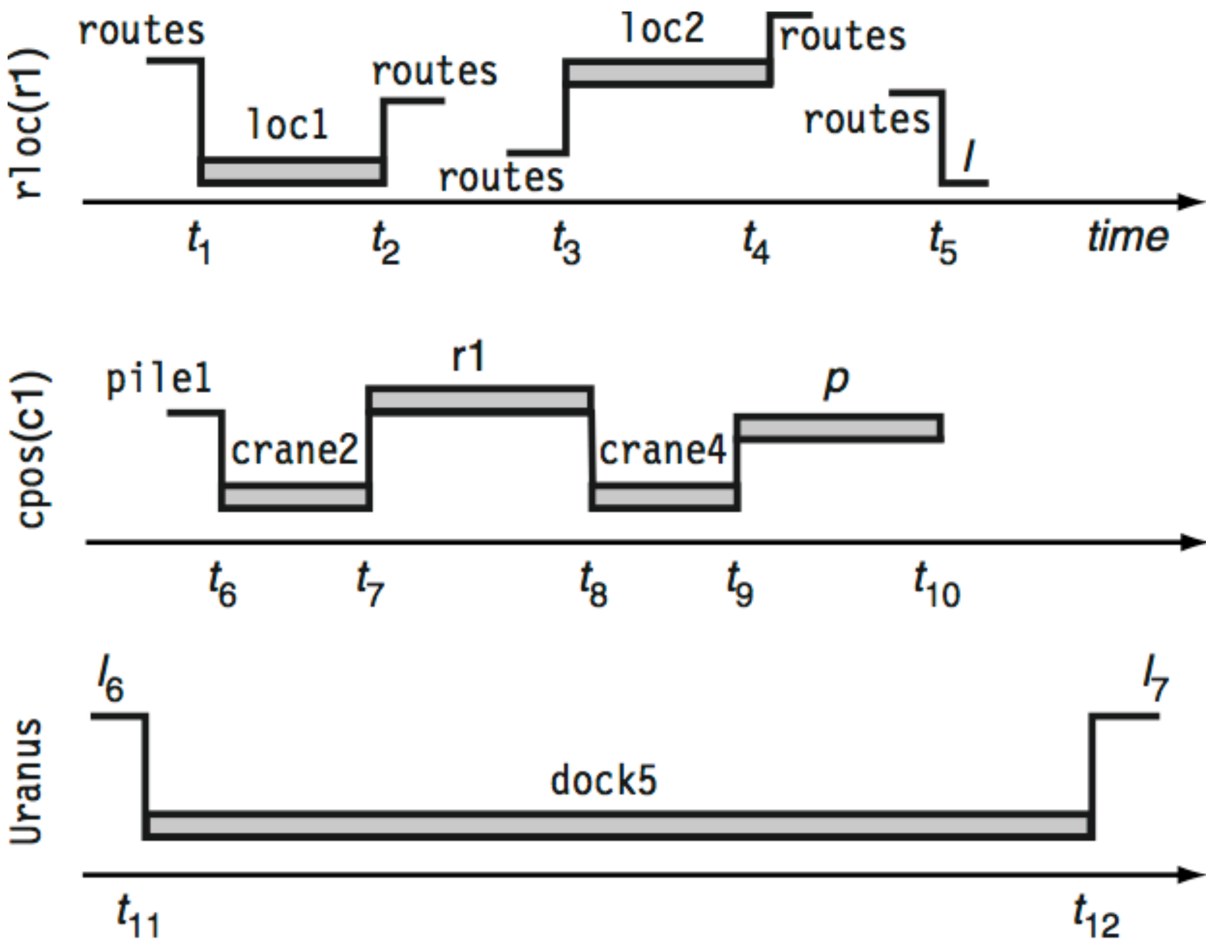
State Variables

- A **state variable** is a partially specified function telling what is true at some time *t*
  - cpos(c1) : time → containers U cranes U robots
    - » Tells what c1 is on at time *t*
  - rloc(r1) : time → locations
    - » Tells where r1 is at time *t*
- Might not ever specify the entire function

- cpos(*c*) refers to a collection of state variables
  - But we'll be sloppy and just call it a state variable

DWR Example

- robot r1
  - in loc1 at time t_1
  - leaves loc1 at time t_2
  - enters loc2 at time t_3
  - leaves loc2 at time t_4
  - enters l at time t_5
- container c1
  - in pile1 until time t_6
  - held by crane2 until t_7
  - sits on r1 until t_8
  - held by crane4 until t_9
  - sits on p until t_{10} (or later)
- ship Uranus
  - stays at dock5 from t_{11} to t_{12}



Temporal Assertions

- Temporal assertion:
  - *Event*: an expression of the form $x@t : (v_1,v_2)$
    - » At time t, x changes from v_1 to $v_2 \neq v_1$
  - *Persistence condition*: $x@[t_1,t_2) : v$
    - » $x = v$ throughout the interval $[t_1,t_2)$
  - where
    - » t, t_1, t_2 are constants or temporal variables
    - » v, v_1, v_2 are constants or object variables
- Note that the time intervals are semi-open
  - Why?

Temporal Assertions

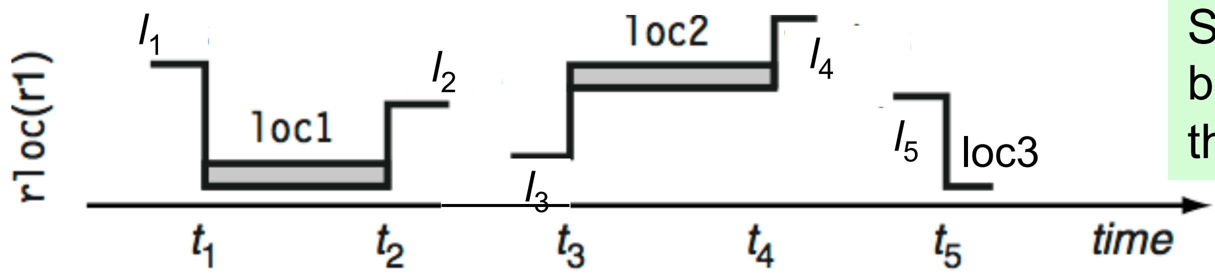
- Temporal assertion:
  - *Event*: an expression of the form $x@t : (v_1,v_2)$
    - » At time t, x changes from v_1 to $v_2 \neq v_1$
  - *Persistence condition*: $x@[t_1,t_2) : v$
    - » $x = v$ throughout the interval $[t_1,t_2)$
  - where
    - » t, t_1, t_2 are constants or temporal variables
    - » v, v_1, v_2 are constants or object variables
- Note that the time intervals are semi-open
  - Why?
  - To prevent potential confusion about x's value at the endpoints

Chronicles

- *Chronicle*: a pair $\Phi = (F,C)$
  - F is a finite set of temporal assertions
  - C is a finite set of constraints
    - temporal constraints and object constraints
  - C must be consistent
    - i.e., there must exist variable assignments that satisfy it
- *Timeline*: a chronicle for a single state variable

- The book writes F and C in a calligraphic font
  - Sometimes I will, more often I'll just use italics

Example



Similar to Figure 14.5, but changed to match the timeline

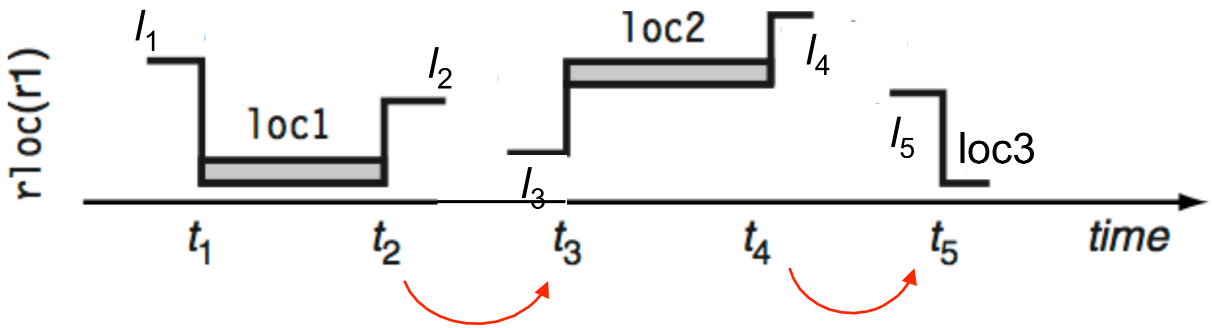
- Timeline for rloc(r1), from Example 14.9 of the book

$$
\begin{aligned}
(\{\quad & \mathsf{rloc(r1)}@t_1 : (l_1, \mathsf{loc1}),\\
& \mathsf{rloc(r1)}@[t_1, t_2) : \mathsf{loc1},\\
& \mathsf{rloc(r1)}@t_2 : (\mathsf{loc1}, l_2),\\
& \mathsf{rloc(r1)}@t_3 : (l_3, \mathsf{loc2}),\\
& \mathsf{rloc(r1)}@[t_3, t_4) : \mathsf{loc2},\\
& \mathsf{rloc(r1)}@t_4 : (\mathsf{loc2}, l_4),\\
& \mathsf{rloc(r1)}@t_5 : (l_5, \mathsf{loc3}) \quad \},\\
\{\quad & \mathsf{adjacent}(l_1, \mathsf{loc1}), \mathsf{adjacent}(\mathsf{loc1}, l_2),\\
& \mathsf{adjacent}(l_3, \mathsf{loc2}), \mathsf{adjacent}(\mathsf{loc2}, l_4), \mathsf{adjacent}(l_5, \mathsf{loc3}),\\
& t_1 < t_2 < t_3 < t_4 < t_5 \ \} \ ).
\end{aligned}
$$

C-consistency

- A timeline (F,C) is *c-consistent* (chronicle-consistent) if
  - C is consistent, and
  - Every pair of assertions in F are either disjoint or they refer to the same value and/or time points:
    - If F contains both $x@[t_1,t_2]:v_1$ and $x@[t_3,t_4]:v_2$, then C must entail $\{t_2 \leq t_3\}$, $\{t_4 \leq t_1\}$, or $\{v_1 = v_2\}$
    - If F contains both $x@t:(v_1,v_2)$ and $x@[t_1,t_2]:v$, then C must entail $\{t < t_1\}$, $\{t_2 < t\}$, $\{v = v_2,\ t_1 = t\}$, or $\{t_2 = t,\ v = v_1\}$
    - If F contains both $x@t:(v_1,v_2)$ and $x@t':(v'_1,v'_2)$, then C must entail $\{t \neq t'\}$ or $\{v_1 = v'_1,\ v_2 = v'_2\}$
- (F,C) is c-consistent iff every timeline in (F,C) is c-consistent
- The book calls this consistency, not c-consistency
  - But it's a stronger requirement than ordinary mathematical consistency
- Mathematical consistency: C doesn't contradict the separation constraints
- c-consistency: C must actually entail the separation constraints
  - It's sort of like saying that (F,C) contains no threats

Example

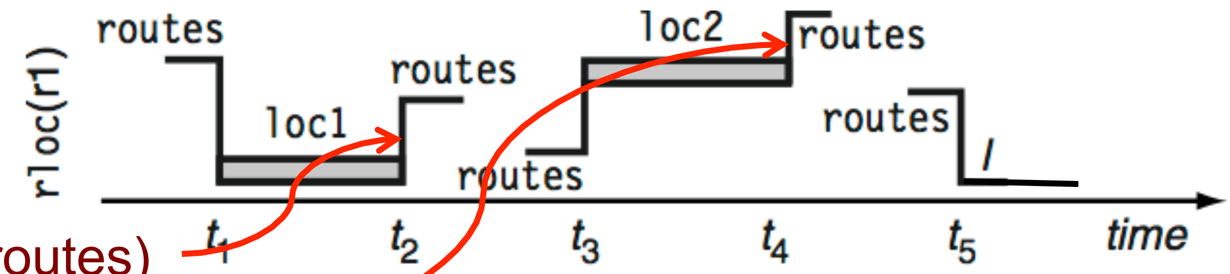


- Let (F,C) be the timeline given earlier for r1
- (F,C) is not c-consistent
  - To ensure that rloc(r1)@$[t_1,t_2)$:loc1 and rloc(r1)@t_3:$(l_3$,loc2) don't conflict, need $t_2 < t_3$ or $t_3 < t_1$
  - To ensure that rloc(r1)@$[t_1,t_2)$:loc1 and rloc(r1)@$[t_3,t_4)$:loc2 don't conflict, need $t_2 < t_3$ or $t_4 < t_1$
  - Etc.
- If we add some additional time constraints, (F,C) will be consistent:
  - e.g., $t_2 < t_3$ and $t_4 < t_5$

Support and Enablers

- Let α be either $x@t{:}(v,v')$ or $x@[t,t'){:}v$
  - Note that α requires $x = v$ either at t or just before t
- Intuitively, a chronicle $\Phi = (F,C)$ *supports* α if
  - F contains an assertion β that we can use to establish $x = v$ at some time $s < t$,
    - » β is called *the support for* α
  - and if it's consistent with Φ for v to persist over $[s,t)$ and for α be true
- Formally, $\Phi = (F,C)$ supports α if
  - F contains an assertion β of the form $\beta = x@s{:}(w',w)$ or $\beta = x@[s',s){:}w$, and
  - $\exists$ separation constraints C' such that the following chronicle is c-consistent:
    - » $(F \cup \{x@[s,t){:}v,\ \alpha\},\ C \cup C' \cup \{w{=}v,\ s < t\})$
  - C' can either be absent from Φ or already in Φ
- The chronicle $\delta = (\{x@[s,t){:}v,\ \alpha\},\ C' \cup \{w{=}v,\ s < t\})$ is an *enabler* for α
  - *Analogous to a causal link in PSP*
- Just as there could be more than one possible causal link in PSP, there can be more than one possible enabler

Example



β_1 = rloc(r1)@t_2 : (loc1, routes)

β_2 = rloc(r1)@t_4 : (loc2, routes)

α_1 = rloc(r1)@t : (routes, loc3)

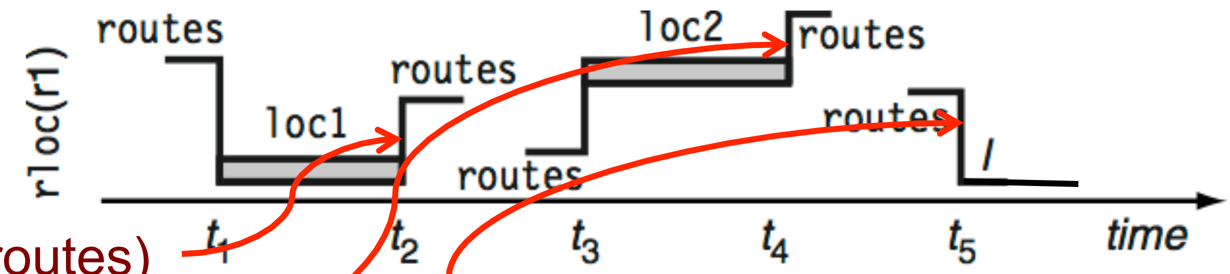
- Φ supports α_1 in two different ways:
  - β_1 establishes rloc(r1) = routes at time t_2
    - » this can support α_1 if we constrain $t_2 < t < t_3$
    - » enabler is $\delta_1 = (\{$rloc(r1)@$[t_2,t)$:routes, $\alpha_1\}, \{t_2 < t < t_3\})$
  - β_2 establishes rloc(r1) = routes at time t_4
    - » this can support α_1 if we constrain $t_4 < t < t_5$
    - » enabler is $\delta_2 = (\{$rloc(r1)@$[t_4,t)$:routes, $\alpha_1\}, \{t_4 < t < t_5\})$

Enabling Several Assertions at Once

- $\Phi = (F,C)$ *supports* a set of assertions $E = \{\alpha_1, \ldots, \alpha_k\}$ if both of the following are true
  - $F \cup E$ contains a support β_i for α_i other than α_i itself
  - There are enablers $\delta_1, \ldots, \delta_k$ for $\alpha_1, \ldots, \alpha_k$ such that the chronicle $\Phi \cup \delta_1 \cup \ldots \cup \delta_k$ is c-consistent

- Note that some of the assertions in E may support each other!
- $\phi = \{\delta_1, \ldots, \delta_k\}$ is an *enabler* for E

Example



β_1 = rloc(r1)@t_2 : (loc1, routes)

β_2 = rloc(r1)@t_4 : (loc2, routes)

β_3 = rloc(r1)@t_4:(loc2, routes)

α_1 = rloc(r1)@t : (routes, loc3)

α_2 = rloc(r1)@$[t',t'')$: loc3

$\delta_1 = (\{$rloc(r1)@$[t_2,t)$:routes, $\alpha_1\}, \{t_2 < t < t_3\}$

$\delta_2 = (\{$rloc(r1)@$[t_4,t)$:routes, $\alpha_1\}, \{t_4 < t < t_5\}$

- Φ supports $\{\alpha_1, \alpha_2\}$ in four different ways:
  - As before, for α_1 we can use either β_1 and δ_1 or β_2 and δ_2
  - We can support α_2 with β_3
    - Enabler is $\delta_3 = (\{$rloc(r1)@$[t_5,t')$:loc3, $\alpha_2\}, \{l =$ loc3, $t_5 < t'\})$
  - Or we can support α_2 with α_1
    - If used β_1 and δ_1 for α_1,
      - Then α_2's enabler is $\delta_4 = (\{$rloc(r1)@$[t,t')$:loc3, $\alpha_2\}, \{t < t' < t_3\})$
    - If we used β_1 and δ_2 for α_1, then replace t_3 with t_5 in δ_4

One Chronicle Supporting Another

- Let $\Phi' = (F', C')$ be a chronicle
- Suppose $\Phi = (F, C)$ supports F'.
- Let $\delta_1, \ldots, \delta_k$ be all the possible enablers of Φ'
  - For each δ_i, let $\delta'_i = \delta_1 \cup \ C'$
- If there is a δ'_i such that $\Phi \cup \ \delta'_i$ is c-consistent,
  - Then Φ *supports* Φ', and δ'_i is an *enabler* for Φ'
  - If $\delta'_i \subseteq \Phi$, then Φ *entails* Φ'
- The set of all enablers for Φ' is $\theta(\Phi/\Phi') = \{\delta'_i : \Phi \cup \delta'_i$ is c-consistent$\}$

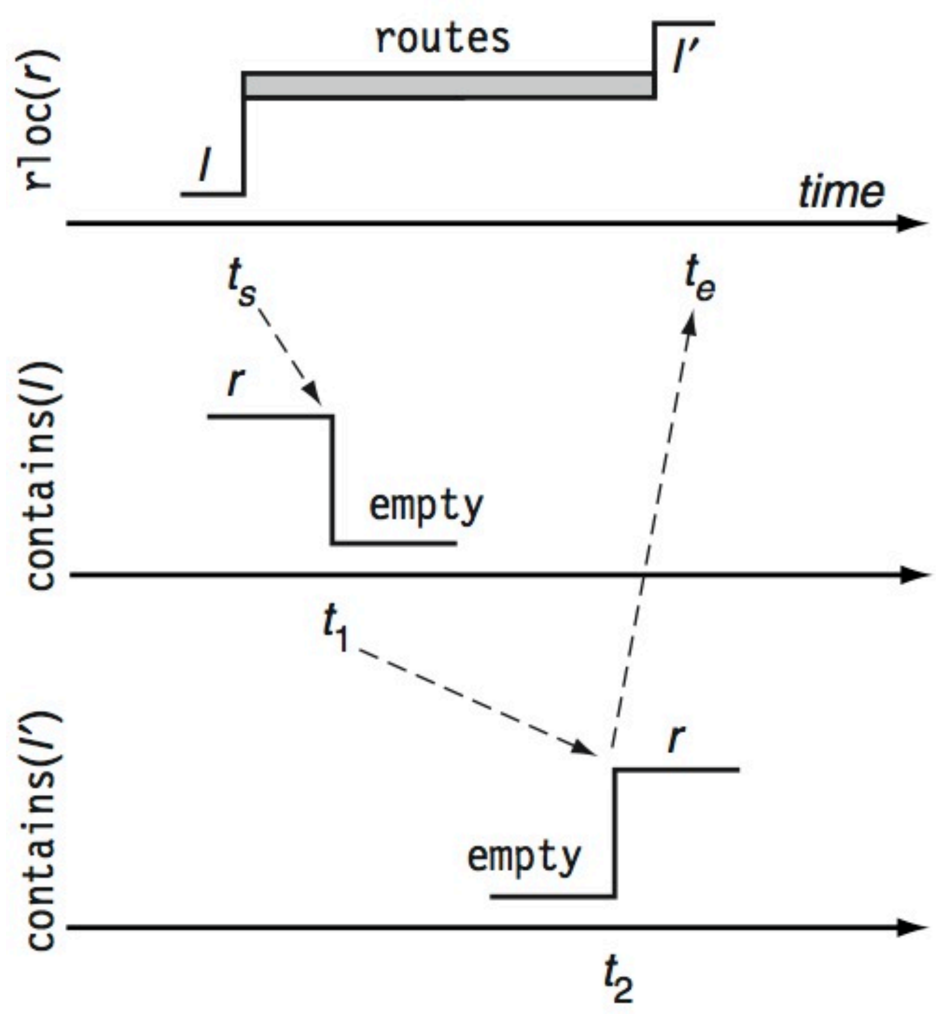
Chronicles as Planning Operators

- Chronicle planning operator: a pair $o = (\text{name}(o), (F(o), C(o)))$, where
  - name(o) is an expression of the form $o(t_s, t_e, \ldots, v_1, v_2, \ldots)$
    - » o is an operator symbol
    - » $t_s, t_e, \ldots, v_1, v_2, \ldots$ are all the temporal and object variables in o
  - $(F(o), C(o))$ is a chronicle

- Action: a (partially) instantiated operator, a
- If a chronicle Φ supports $(F(a),C(a))$, then a is *applicable* to Φ
  - a may be applicable in several ways, so the result is a set of chronicles
    - » $\gamma(\Phi,a) = \{\Phi \cup \phi \mid \phi \in \theta(a/\Phi)\}$

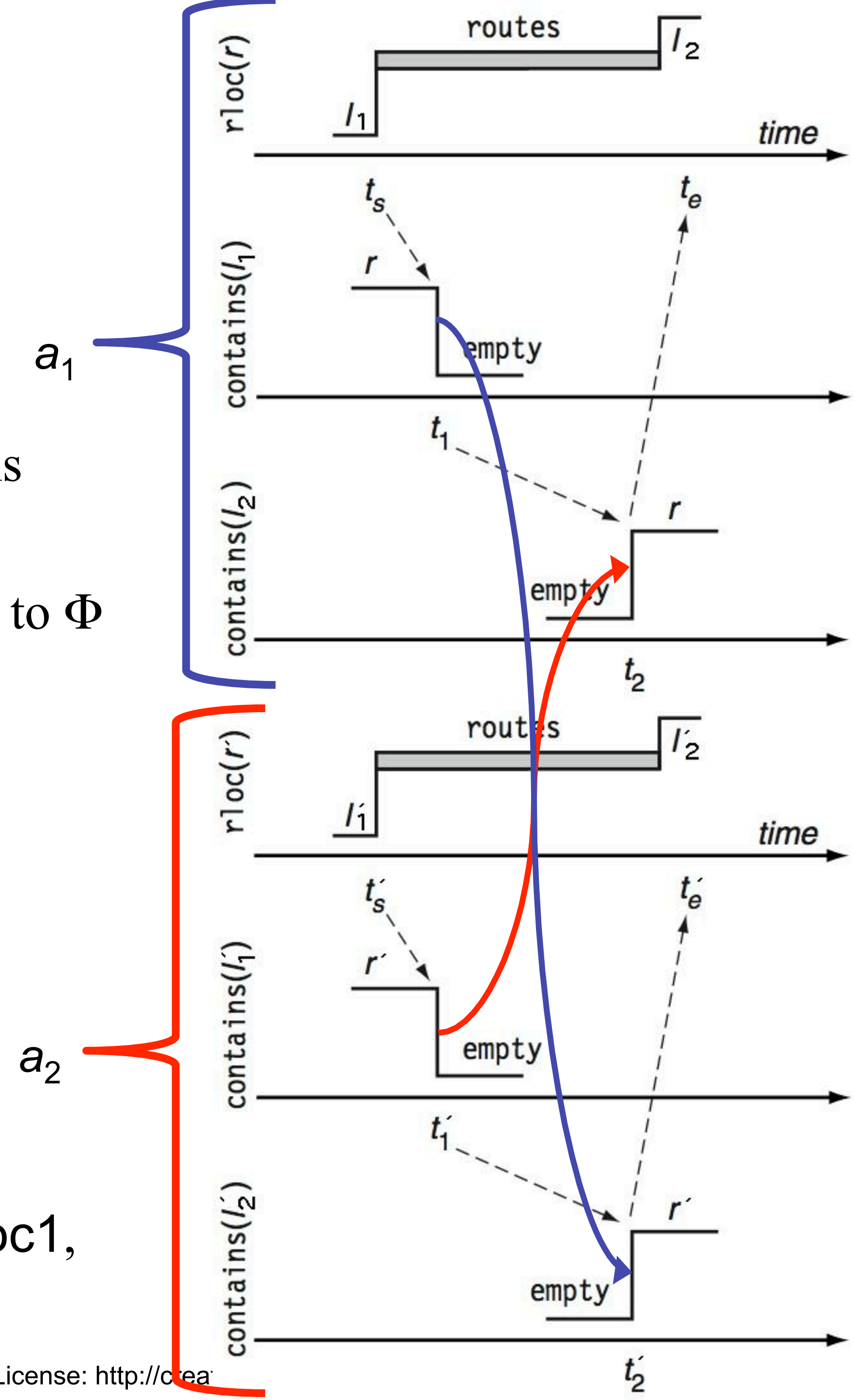
Example: Operator for Moving a Robot

$$
\begin{aligned}
&\text{move}(t_s, t_e, t_1, t_2, r, l, l') = \\
&\{ \quad \text{rloc}(r)@t_s && : \quad (l, \text{routes}), \\
&\quad\ \ \text{rloc}(r)@[t_s, t_e) && : \quad \text{routes}, \\
&\quad\ \ \text{rloc}(r)@t_e && : \quad (\text{routes}, l'), \\
&\quad\ \ \text{contains}(l)@t_1 && : \quad (r, \text{empty}), \\
&\quad\ \ \text{contains}(l')@t_2 && : \quad (\text{empty}, r), \\
&\quad\ \ t_s < t_1 < t_2 < t_e, \\
&\quad\ \ \text{adjacent}(l, l') \ \}
\end{aligned}
$$



Applying a Set of Actions

- Just like several temporal assertions can support each other, several actions can also support each other
  - Let $\pi = \{a_1, \ldots, a_k\}$ be a set of actions
  - Let $\Phi_\pi = \cup_i \, (F(a_i), C(a_i))$
  - If Φ supports Φ_π then π is applicable to Φ
  - Result is a ***set*** of chronicles $\gamma(\Phi,\pi) = \{\Phi \cup \phi \mid \phi \in \theta(\Phi_\pi/\Phi)\}$
- Example:
  - Suppose Φ asserts that at time t_0, robots r1 and r2 are at adjacent locations loc1 and loc2
  - Let a_1 and a_2 be as shown
  - Then Φ supports $\{a_1, a_2\}$ with $l_1 =$ loc1, $l_2 =$ loc2, $l'_1 =$ loc2, $l'_2 =$ loc1, $t_0 < t_s < t_1 < t'_2$, $t_0 < t'_s < t'_1 < t_2$



Domains and Problems

- Temporal planning *domain*:
  - A pair $\boldsymbol{D} = (\Lambda_\Phi, O)$
    - » O = {all chronicle planning operators in the domain}
    - » Λ_Φ = {all chronicles allowed in the domain}
- Temporal planning *problem* on $\boldsymbol{D}$:
  - A triple $\boldsymbol{P} = (\boldsymbol{D}, \Phi_0, \Phi_g)$
    - » $\boldsymbol{D}$ is the domain
    - » Φ_0 and Φ_g are initial chronicle and goal chronicle
    - » O is the set of chronicle planning operators
- Statement of the problem $\boldsymbol{P}$:
  - A triple $P = (O, \Phi_{0,} \Phi_g)$
    - » O is the set of chronicle planning operators
    - » Φ_0 and Φ_g are initial chronicle and goal chronicle
- *Solution plan*:
  - A set of actions $\pi = \{a_1, \ldots, a_n\}$ such that at least one chronicle in $\gamma(\Phi_0, \pi)$ entails Φ_g

set of open goals (*tqes*)

set of sets of enablers

$\mathsf{CP}(\Phi, G, \mathcal{K}, \pi)$

    if $G = \mathcal{K} = \emptyset$ then return(π)

    perform the two following steps in any order

        if $G \neq \emptyset$ then do

            select any $\alpha \in G$

            if $\theta(\alpha/\Phi) \neq \emptyset$ then return($\mathsf{CP}(\Phi, G - \{\alpha\}, \mathcal{K} \cup \{\theta(\alpha/\Phi)\}, \pi)$)

            else do

                relevant $\leftarrow \{a \mid a$ contains a support for $\alpha\}$

                if *relevant* $= \emptyset$ then return(failure)

                nondeterministically choose $a \in$ *relevant*

                return($\mathsf{CP}(\Phi \cup (\mathcal{F}(a), \mathcal{C}(a)), G \cup \mathcal{F}(a), \mathcal{K} \cup \{\theta(a/\Phi)\}, \pi \cup \{a\})$)

        if $\mathcal{K} \neq \emptyset$ then do

            select any $C \in \mathcal{K}$

            threat-resolvers $\leftarrow \{\phi \in C \mid \phi$ consistent with $\Phi\}$

            if *threat-resolvers* $= \emptyset$ then return(failure)

            nondeterministically choose $\phi \in$ *threat-resolvers*

            return($\mathsf{CP}(\Phi \cup \phi, G, \mathcal{K} - C, \pi)$)

end

- As in plan-space planning, there are two kinds of flaws:
  - Open goal: a *tqe* that isn't yet enabled
  - Threat: an enabler that hasn't yet been incorporated into Φ

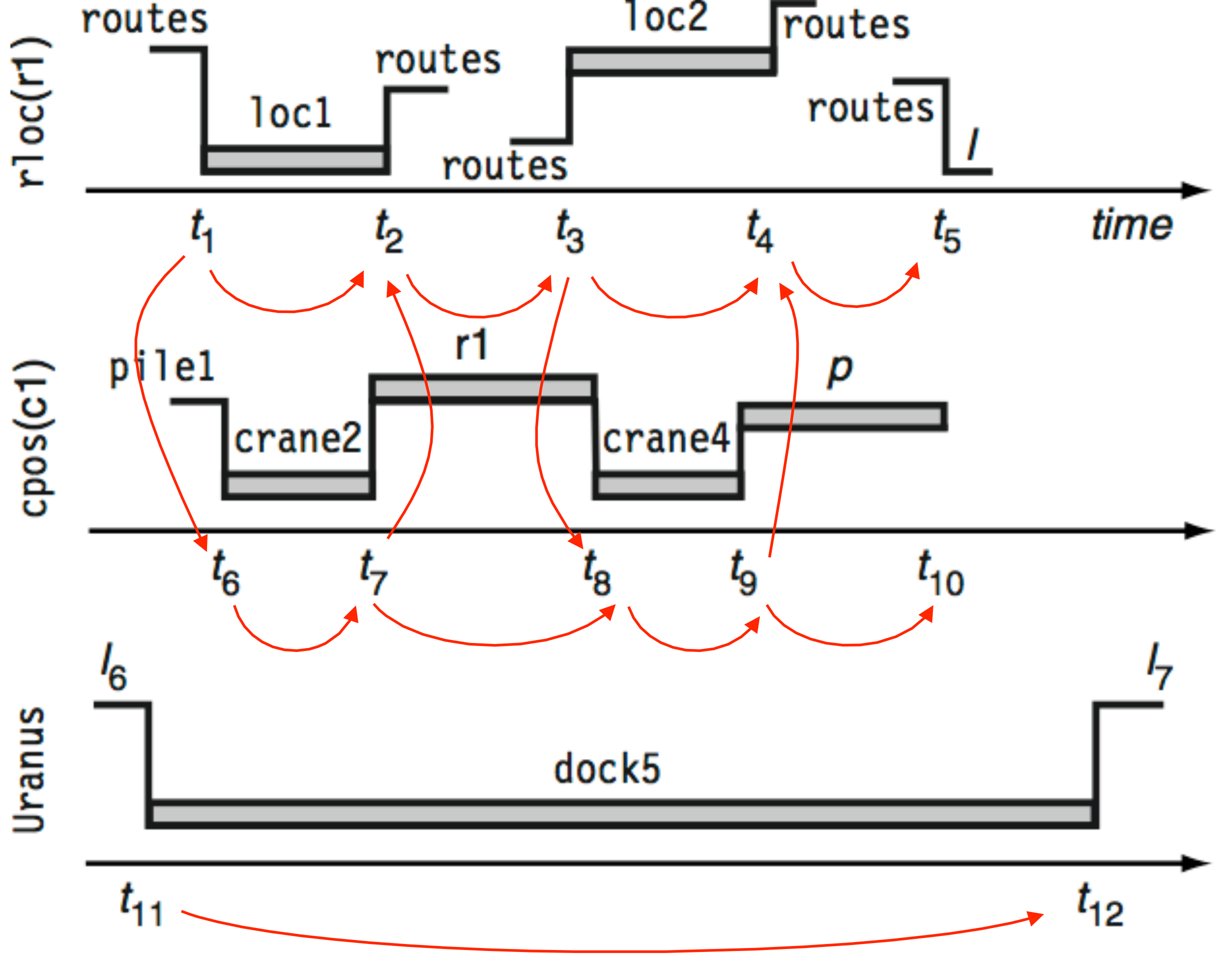
Resolving Open Goals

- Let $\alpha \in G$ be an open goal
- Case 1: Φ supports α
  - Resolver: any enabler for α that's consistent with Φ
  - Refinement:
    - $G \leftarrow G - \{\alpha\}$
    - $K \leftarrow K \cup \theta(\alpha/\Phi)$
- Case 2: Φ doesn't support α
  - Resolver: an action $a = (F(a), C(a))$ that supports α
    - We don't yet require a to be supported by Φ
  - Refinement:
    - $\pi \leftarrow \pi \cup \{a\}$
    - $\Phi \leftarrow \Phi \cup (F(a), C(a))$
    - $G \leftarrow G \cup F(a)$ Don't remove α yet: we haven't chosen an enabler for it
      - We'll choose one in a later call to CP, in Case 1 above
    - $K \leftarrow K \cup \theta(a/\Phi)$ put a's set of enablers into K

Resolving Threats

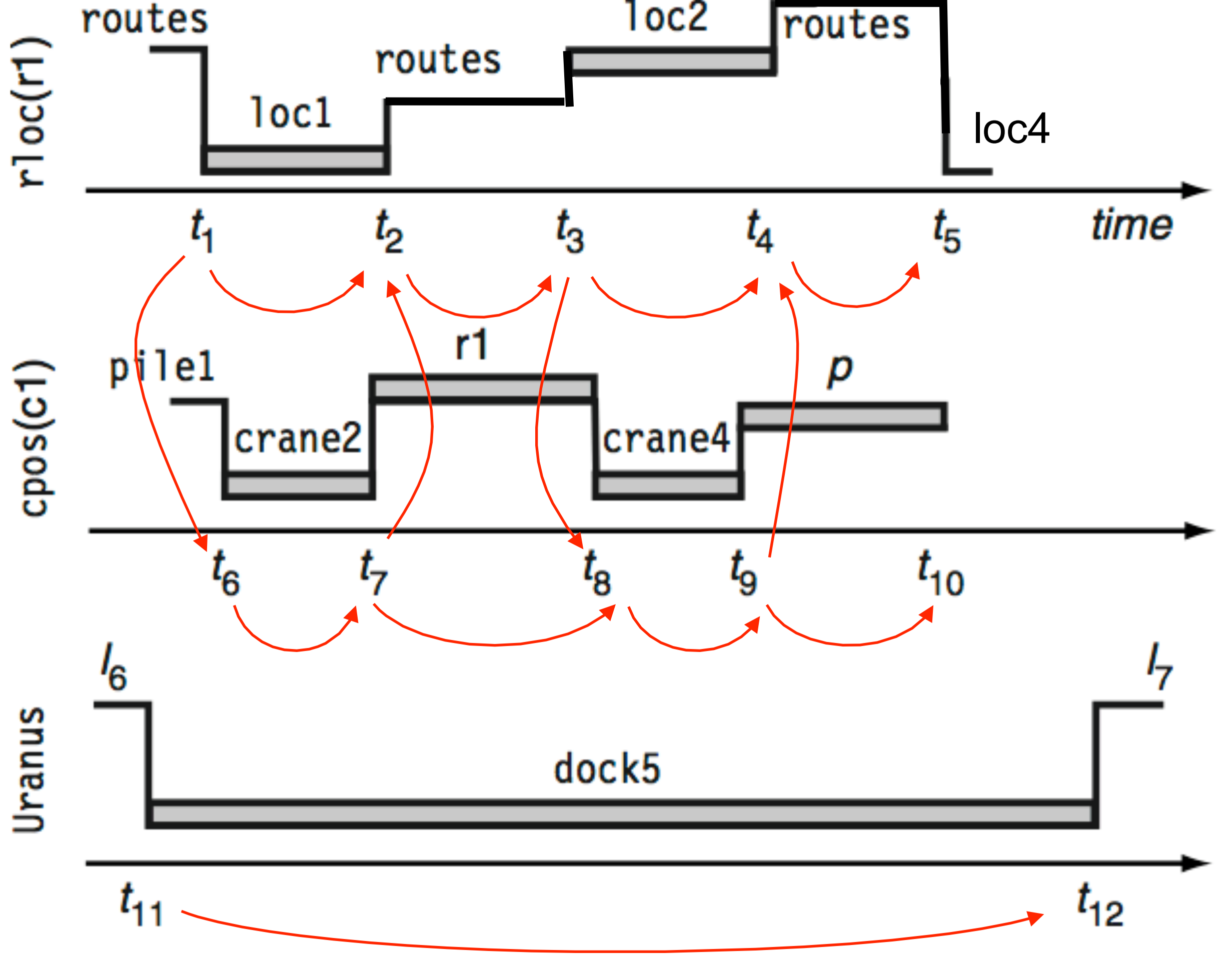
- *Threat*: each enabler in K that isn't yet entailed by Φ is threatened
  - For each C in K, we need only one of the enablers in C
    - They're alternative ways to achieve the same thing
  - "Threat" means something different here than in PSP, because we won't try to entail *all* of the enablers
    - Just the one we select
  - Resolver: any enabler ϕ in C that is consistent with Φ
  - Refinement:
    - $K \leftarrow K - C$
    - $\Phi \leftarrow \Phi \cup \phi$

Example



- Let Φ_0 be as shown, and $\Phi_g = \Phi_0 \cup (\{\alpha_1,\alpha_2\},\{\})$, where α_1 and α_2 are the same as before:
  - α_1 = rloc(r1)@t:(routes, loc3)
  - α_2 = rloc(r1)@$[t',t'')$:loc3
- As we saw earlier, we can support $\{\alpha_1,\alpha_2\}$ from Φ_0
  - Thus CP won't add any actions
  - It will return a modified version of Φ_0 that includes the enablers for $\{\alpha_1,\alpha_2\}$

Modified Example



- Let Φ_0 be as shown, and $\Phi_g = \Phi_0 \cup (\{\alpha_1,\alpha_2\},\{\})$, where α_1 and α_2 are the same as before:
  - α_1 = rloc(r1)@t:(routes, loc3)
  - α_2 = rloc(r1)@$[t',t'')$:loc3
  - This time, CP will need to insert an action move(t_s, t_e, t_1, t_2, r1, loc4, loc3)
    - with $t_5 < t_s < t_1 < t_2 < t_e$