Temporal Goal Networks: Work in Progress

Dana S. Nau

University of Maryland

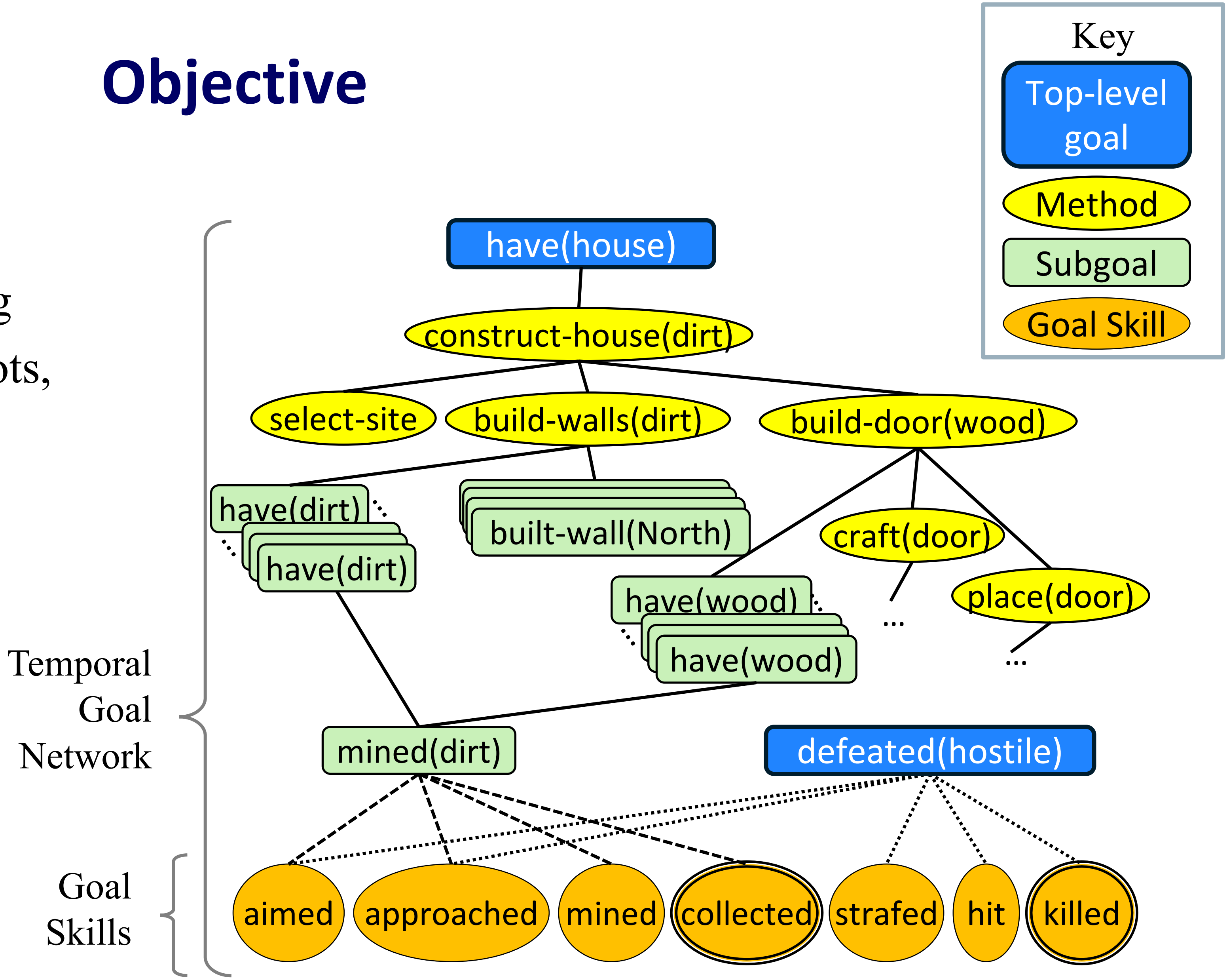
Collaborators:

Mak Roberts (NRL) – co-PI
Sunandita Patra – postdoctoral researcher
Ruoxi Li, Onur Kulaksizoglu, Mark Cavolowsky, Alex Mendelsohn – PhD students

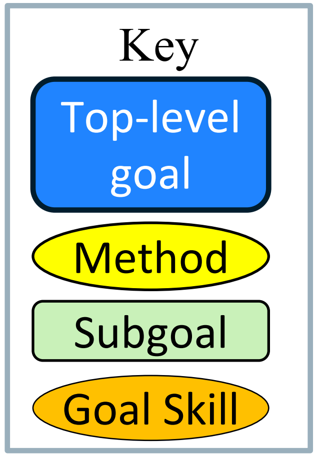
Research supported in part by ONR grant N000142012257 and NRL grant N0017320P0399

Objective

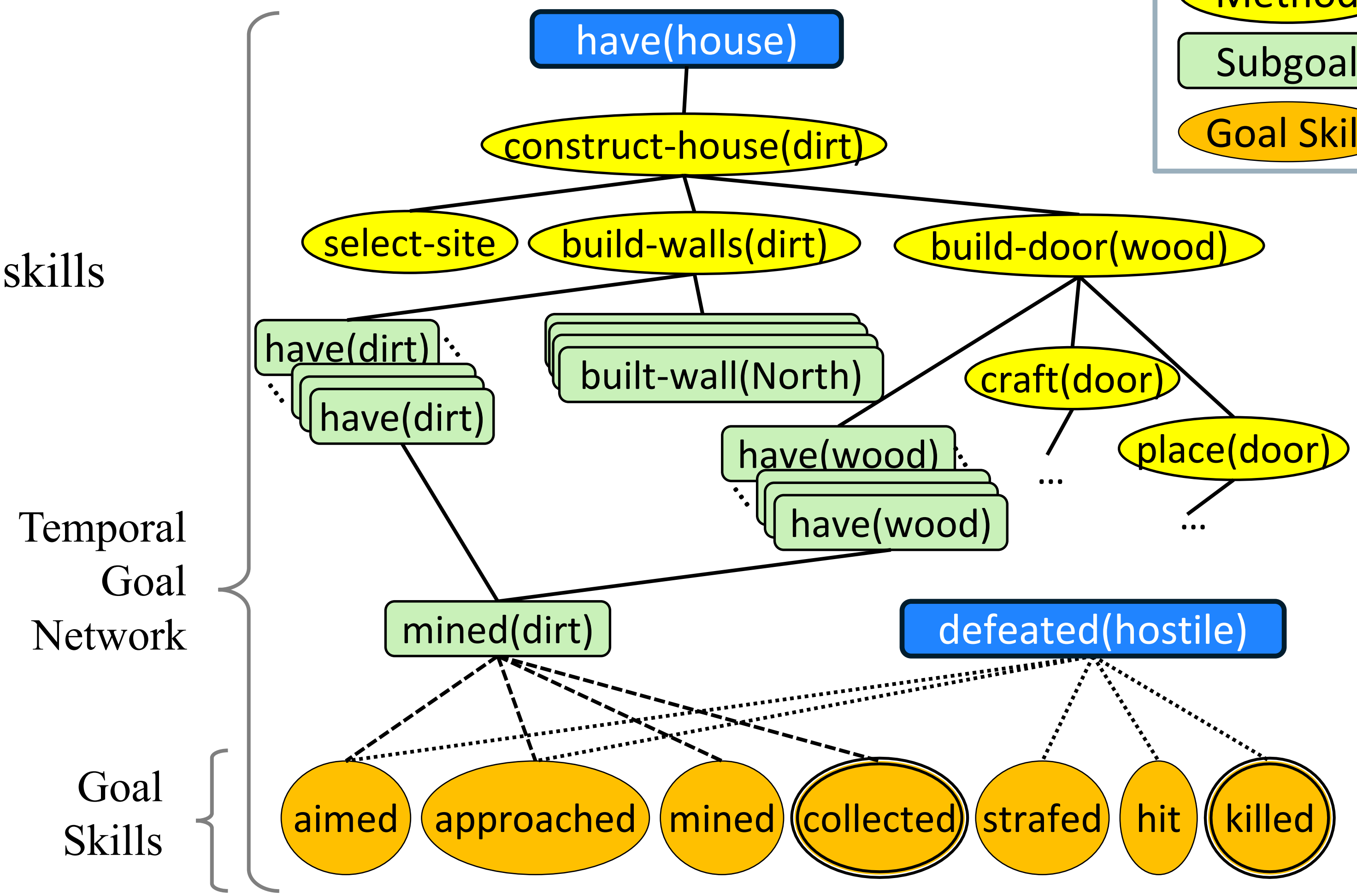
- Learn and utilize hierarchies of goals and skills
- Use for integrated acting and planning
- Scale to mixed teams of humans, robots, & software



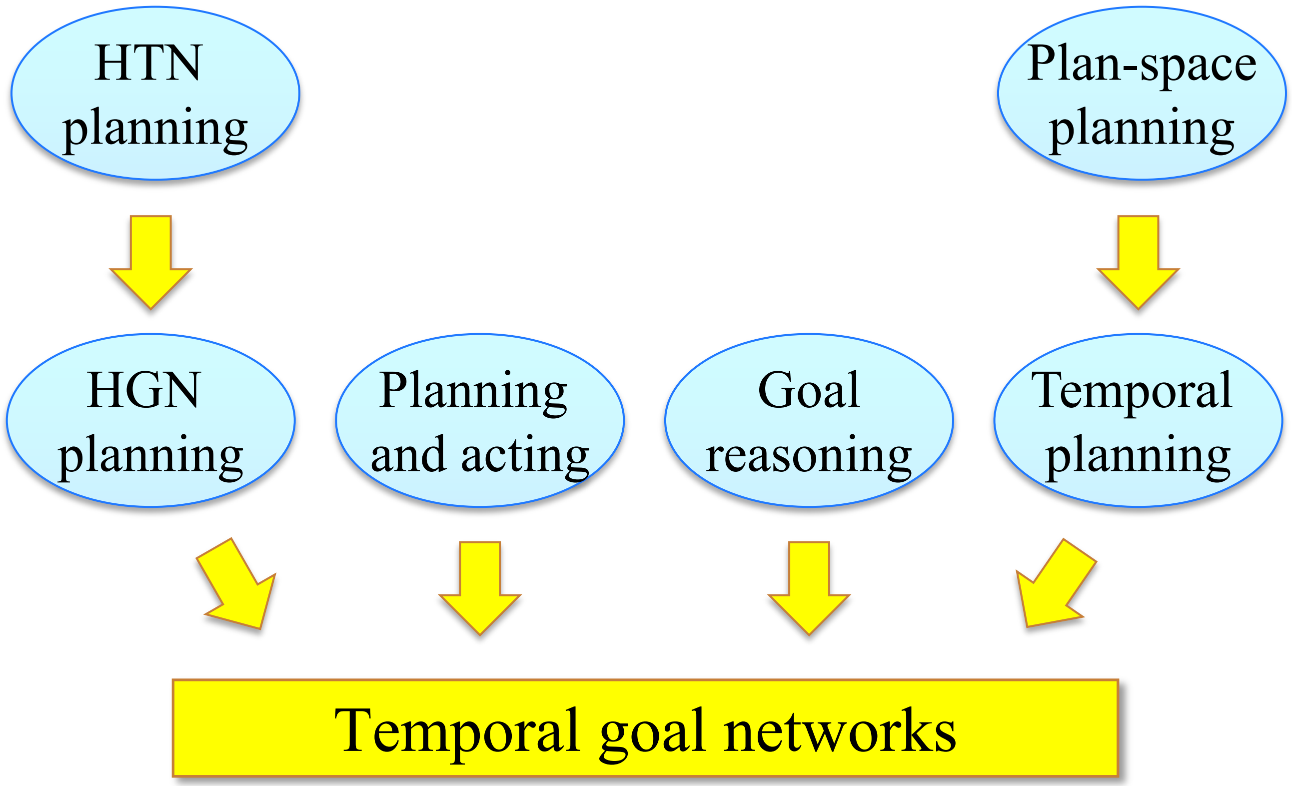
Approach



- Generalization of goal networks to temporal planning
  - Work in progress
  - Today's presentation
- Link goal networks to abstracted RL skills
  - Work in progress
  - I'm not prepared to talk about it
- Algorithms to automatically learn temporal-goal-network hierarchies
  - Not there yet



Background


HTN planning
Plan-space planning
HGN planning
Planning and acting
Goal reasoning
Temporal planning
Temporal goal networks

Background

HTN Planning

- For some planning problems, we may already have ideas for how to look for solutions
- Example: travel to a destination that's far away:
  - Brute-force search:
    - many combinations of vehicles and routes
  - Experienced human: small number of "recipes" that decompose tasks into smaller subtasks

    e.g., flying:

    1. buy ticket from local airport to remote airport
    2. travel to local airport
    3. fly to remote airport
    4. travel to final destination

- How can a planner make use of such information?

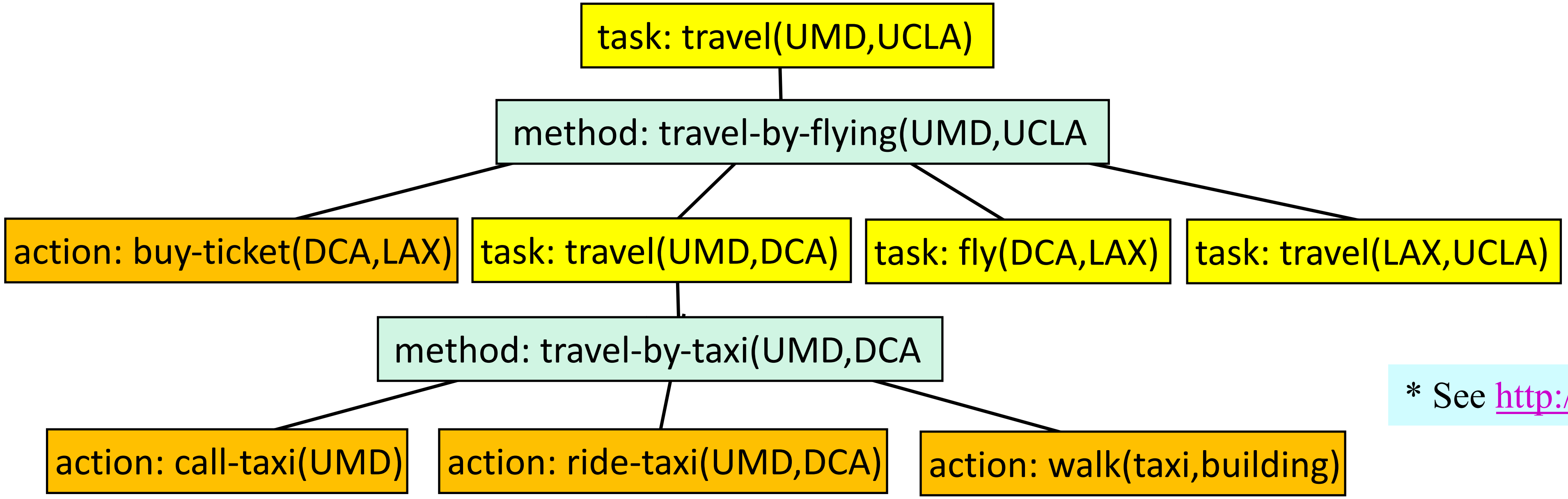
Background

HTN Planning

- Recursively use *methods* to decompose *tasks* into *subtasks*
- SHOP, SHOP2, Pyhop* do HTN planning *left-to-right*
  - decompose tasks in the same order that they'll be accomplished
  - always know current state
  - increases the scope of applicability
    - state can be arbitrary data structure
    - preconditions and effects can be computations

method:

travel-by-flying(x,y)
task: travel(x,y)
pre: at(x),
long-distance(x,y),
airport-near(x,u),
airport-near(y,v),
subtasks:
buy-ticket(u,v),
travel(x,u),
fly(u,v),
travel(v,y)

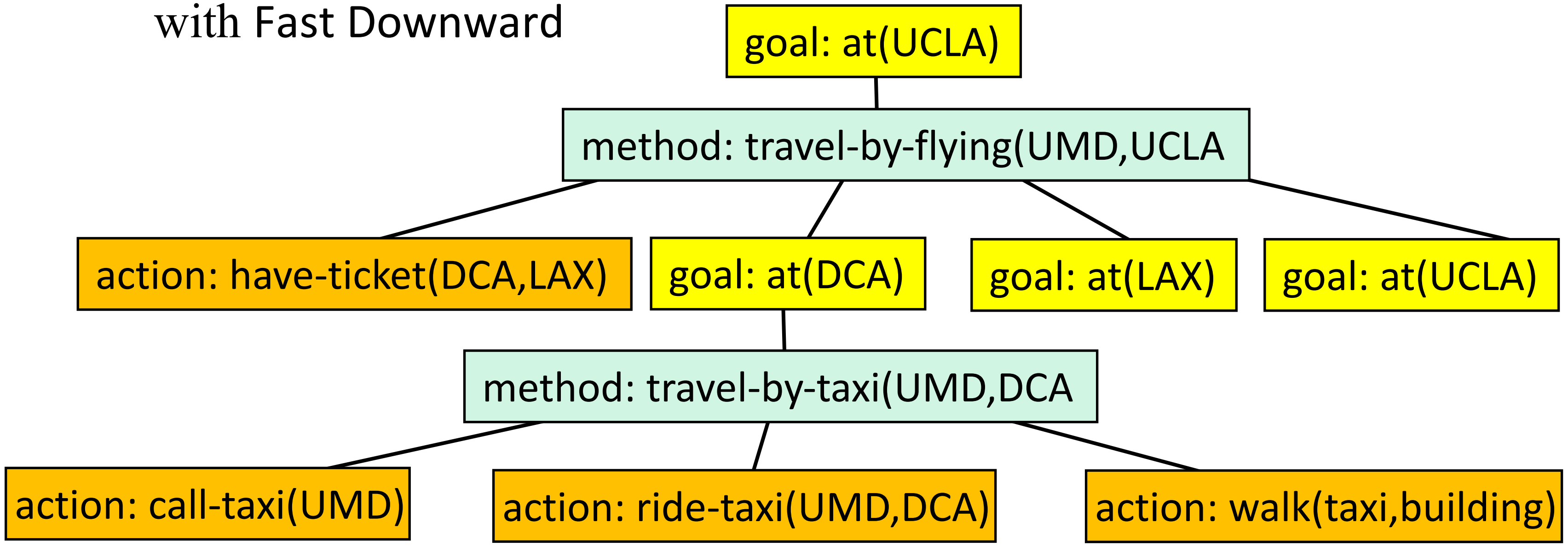


* See http://www.cs.umd.edu/projects/shop

Background

HGN Planning

- Recursively use *methods* to decompose *goals* into *subgoals*
- GDP, GoDel* decompose goals left-to-right
  - Like SHOP, SHOP2, Pyhop, always know current state
  - Also: can reason about goals
    - e.g., Godel's interaction with Fast Downward



method:

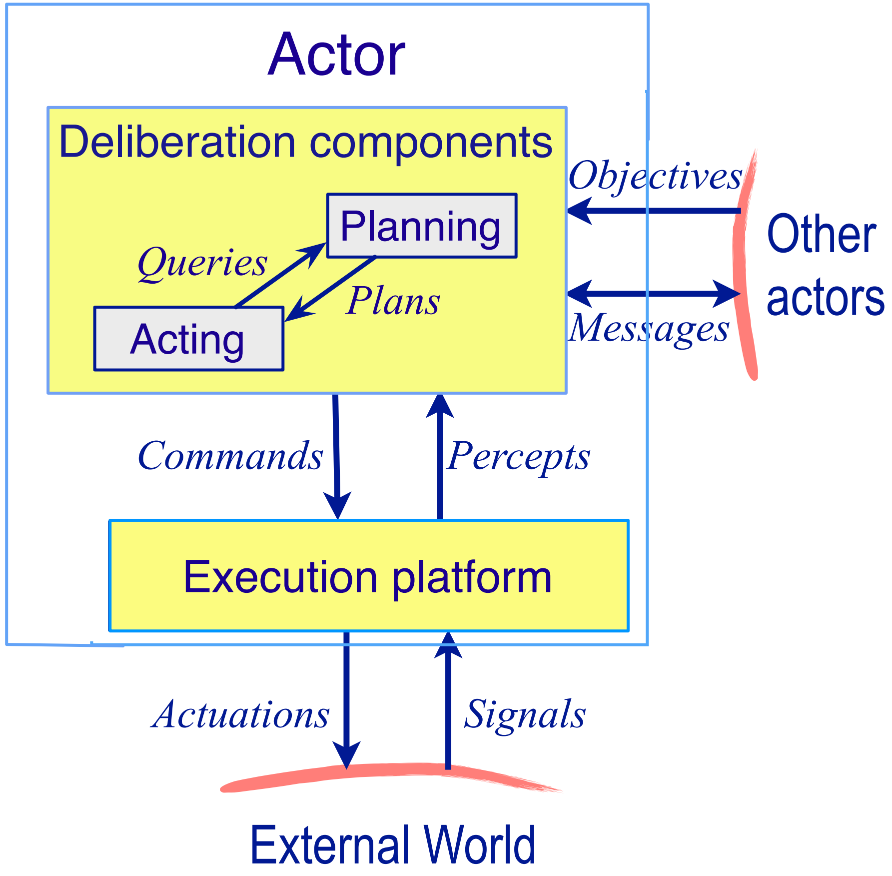
travel-by-flying(x,y)
pre: at(x),
long-distance(x,y),
airport-near(x,u),
airport-near(y,v),
subgoals:
have-ticket(u,v),
at(u),
at(v),
at(y)

* Shivashankar, Alford, Kuter, and Nau. The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *IJCAI*, pp. 2380–2386, 2013

Background

Planning and Acting*

- Planning: *prediction* + *search*
  - Search over predicted states, ways to organize tasks and actions
  - Has traditionally used *descriptive* models (e.g., PDDL)
    - predict *what* the actions will do
- Acting: *performing* the actions
  - Dynamic, unpredictable, partially observable environment
  - *Operational* models: tell *how* to perform the actions in the current context, react to events



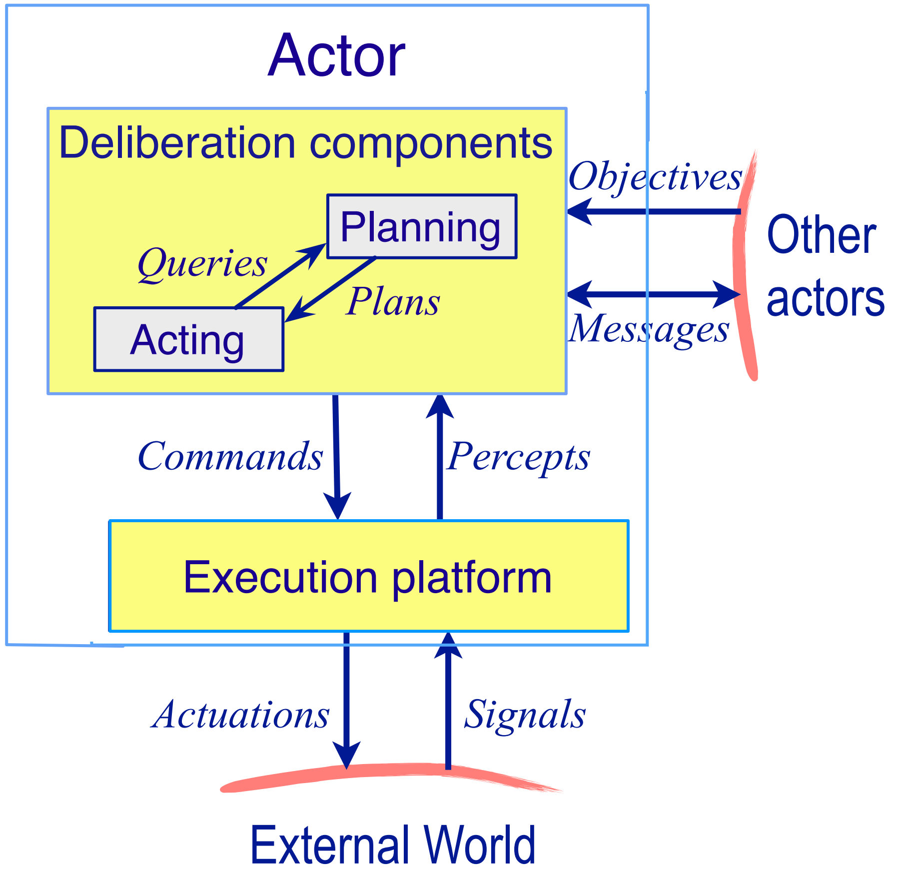
* Chapter 4 of Ghallab, Nau, and Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

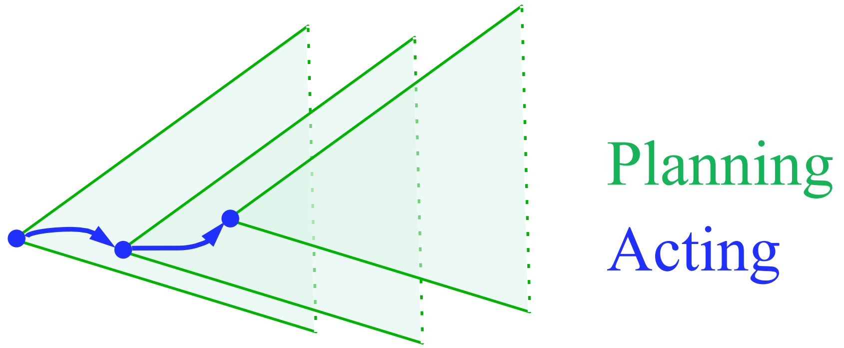
Background

Planning and Acting*

- Planning: *prediction* + *search*
  - Search over predicted states, ways to organize tasks and actions
  - Has traditionally used *descriptive* models (e.g., PDDL)
    - predict *what* the actions will do
- Acting: *performing* the actions
  - Dynamic, unpredictable, partially observable environment
  - *Operational* models: tell *how* to perform the actions in the current context, react to events
- Planning is online, recurs continually as the world changes

  (next slide)



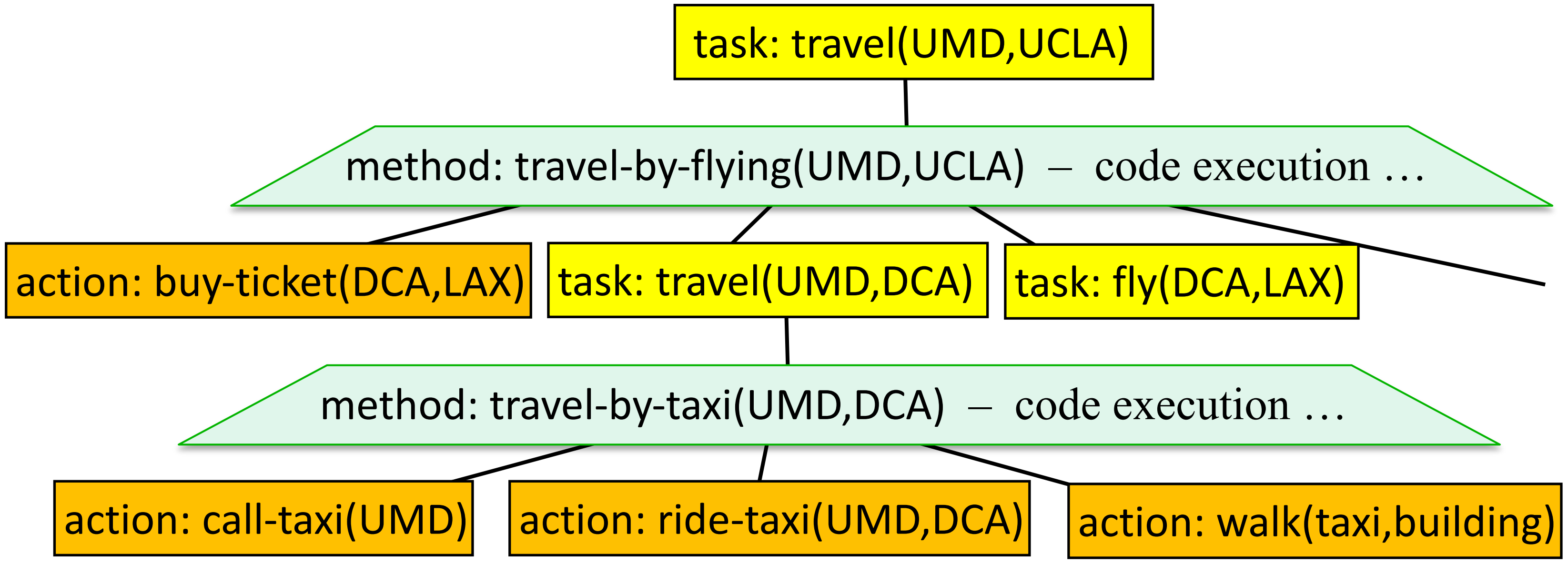


* Chapter 4 of Ghallab, Nau, and Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

Background

Planning and Acting

- RAE+UPOM acting-and-planning system*
  - Uses actor's operational models for both acting *and online planning*
  - Planning: receding-horizon search
    - Repeated Monte Carlo rollouts in a simulated environment
    - Statistical sample of possible outcomes
    - Choose the method with greatest expected utility



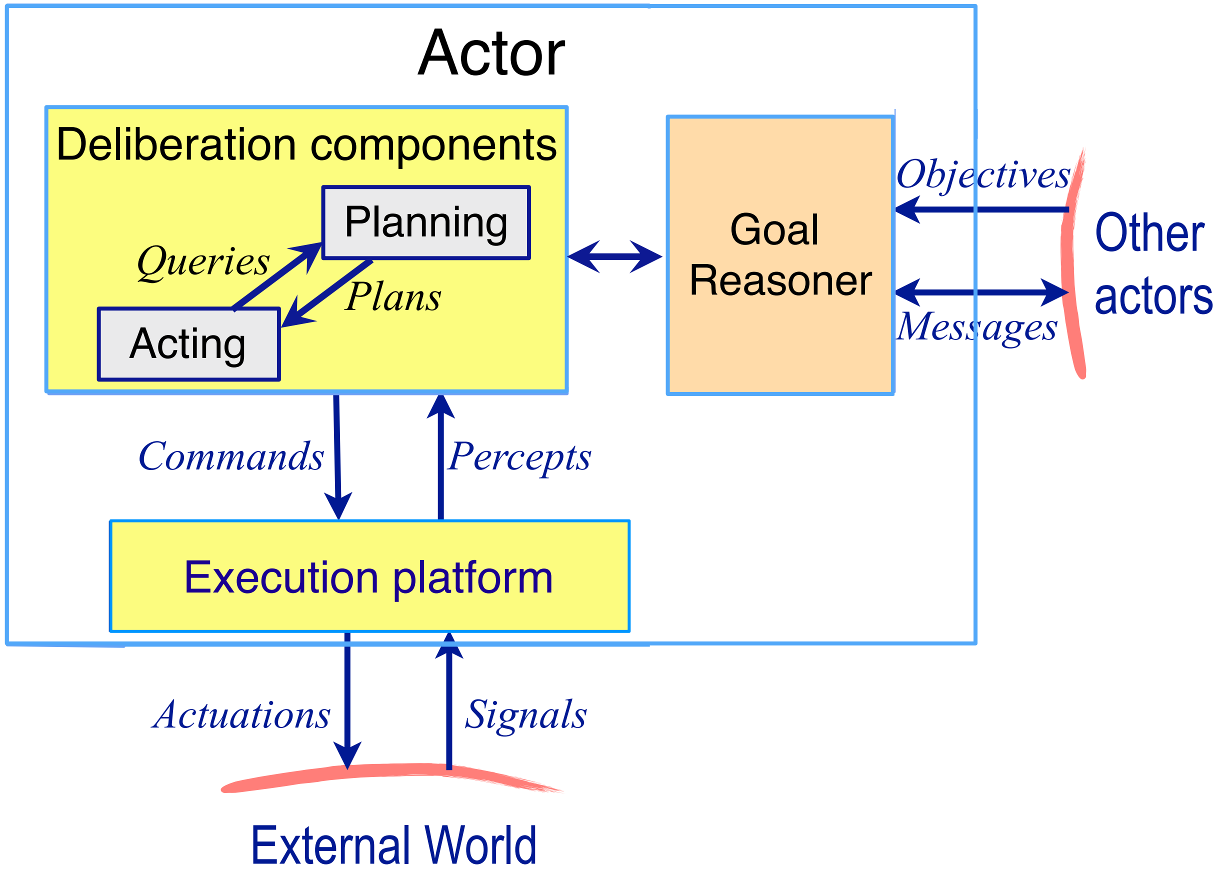
method-name(arg_1, …, arg_k)
  task: *task-identifier*
  pre: *test*
  body:
    computer program that may contain
    - *other tasks*
    - *commands*

* Patra, Mason, Kumar, Ghallab, Traverso, and Nau. Integrating acting, planning, and learning in hierarchical operational models. *ICAPS*, pp. 478–487, Oct. 2020. Best student paper honorable mention award.

Background

Goal Reasoning*

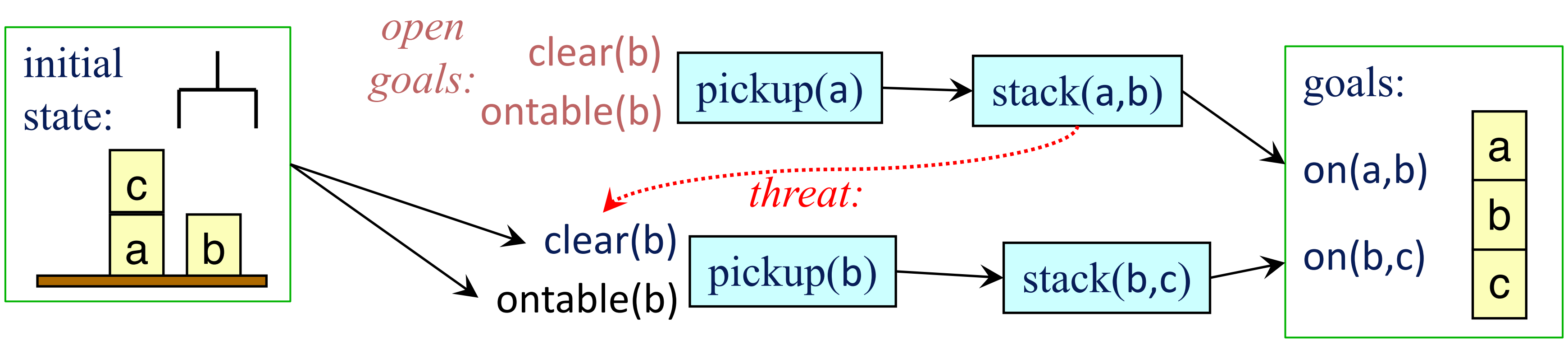
- Planning and acting, taken a step further
  - Actor may change its goals as the world changes
  - Overlaps with automated planning and acting, cognitive architectures, BDI systems
  - Early works used the Goal-Driven Autonomy model
    - Deliberation extended to include discrepancy detection, goal management and explanation
  - Goal networks support goal reasoning via the *goal lifecycle**



* D.W. Aha. Goal Reasoning: Foundations, Emerging Applications, and Prospects. *AI Magazine*, Vol. 29, No. 2, pp. 3-24.

Background

Plan-Space Planning



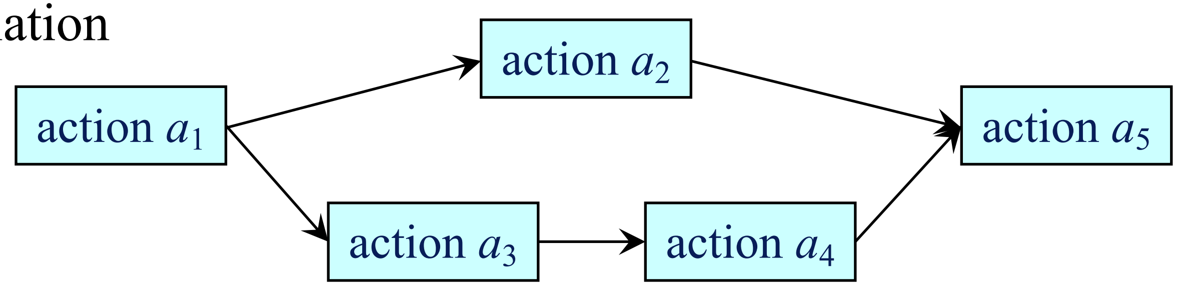
- Two kinds of *flaws*:
  - *Open goal*: unachieved precondition
    - To resolve: find or add an action that achieves it
  - *Threat*: potential interference with an achieved precondition
    - To resolve: add constraints that remove the interference

Plan-space planning:
  start with a dummy plan (initial state, goal)
  loop
    if no flaws, exit with success
    choose a flaw *f*
    if *f* is unresolvable, exit with failure
    nondeterministically choose a way to resolve *f*

Background

Plan-Space Planning

- Disadvantages:
  - large branching factor – huge search space on large problems
  - can't do receding horizon, planning as simulation
- Advantage: flexibility of solution

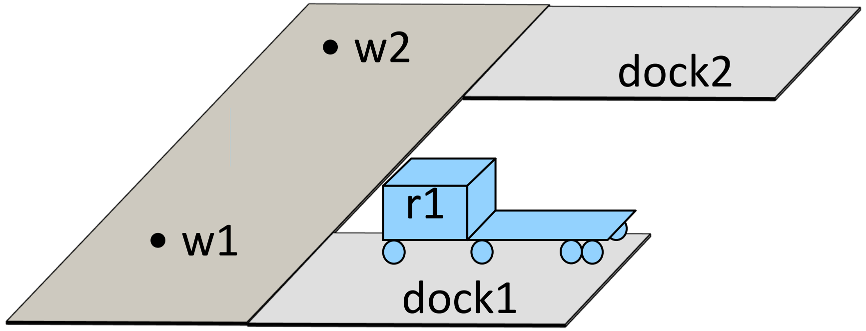


- Planning involves making *commitments*
  - e.g., use action a_1 to satisfy preconditions of action a_2
- *Least commitment principle:*
  - Look for a solution plan that makes as few commitments as possible
  - Solution plan may be partially ordered, partially instantiated

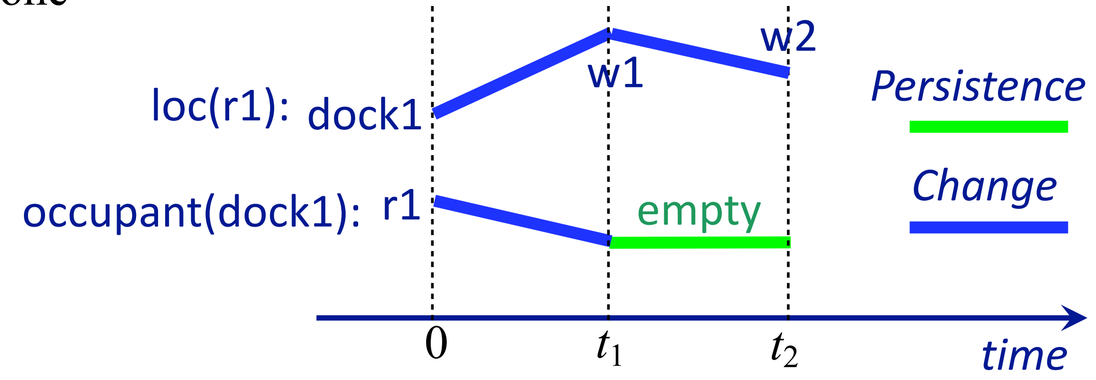
- In temporal plans, this can provide flexibility for
  - resource scheduling
  - responding to unexpected events
- Temporal plan-space planning has been used in many NASA projects

Background

Temporal Planning



- "Classical" AI planning algorithms use a *state-oriented view*
  - Time is a sequence of states s_0, s_1, s_2
  - Actions instantaneously transform each state into the next one
- *Time-oriented view*:
  - For each state variable x, a *timeline*
    - values of x over time
  - State at time t = {values of state-variable at time t}
- *Chronicle representation:*
  - notation for a collection of timelines and constraints
- TPS planning algorithm*
  - temporal version of plan-space planning



* Chapter 14 of Ghallab, Nau, and Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.

ϕ:

temporal assertions:

$[0,t_1]$ loc(r1):(dock1,w1), $[t_1,t_2]$ loc(r1):(w1,w2),

$[0,t_1]$ occupant(dock1):(r1,empty),

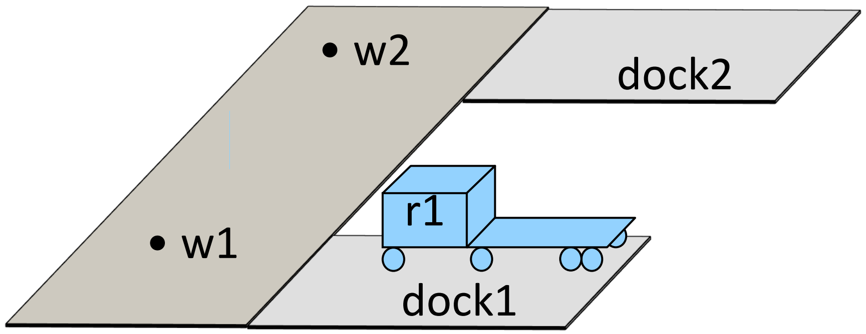
$[t_1,t_2]$ occupant(dock1) = empty

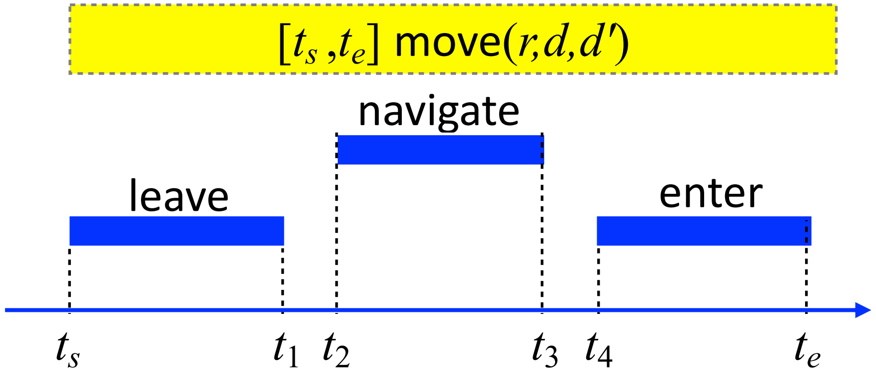
constraints:

adjacent(dock1,w1), connected(w1,w2), $t_0 < t_1 < t_2$

Background

Hierarchical Temporal Planning





- Chronicle representation of methods
  - May have multiple methods for the same goal
- TemPlan algorithm* ≈ TPS plus task refinement

* Chapter 4 of Ghallab, Nau, and Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

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

task: move(r,d) *// go to another loading dock*

refinement:

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

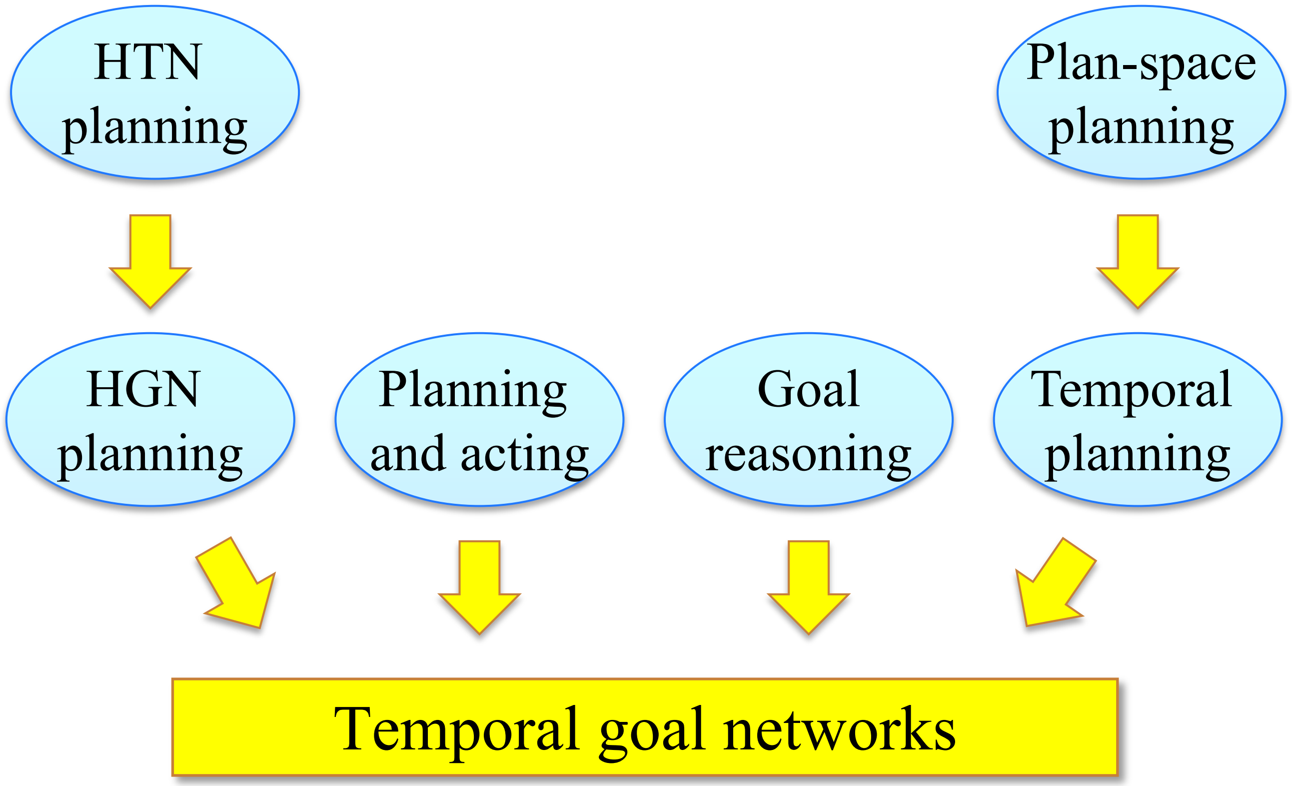
assertions:

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

constraints:

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

Temporal Goal Networks

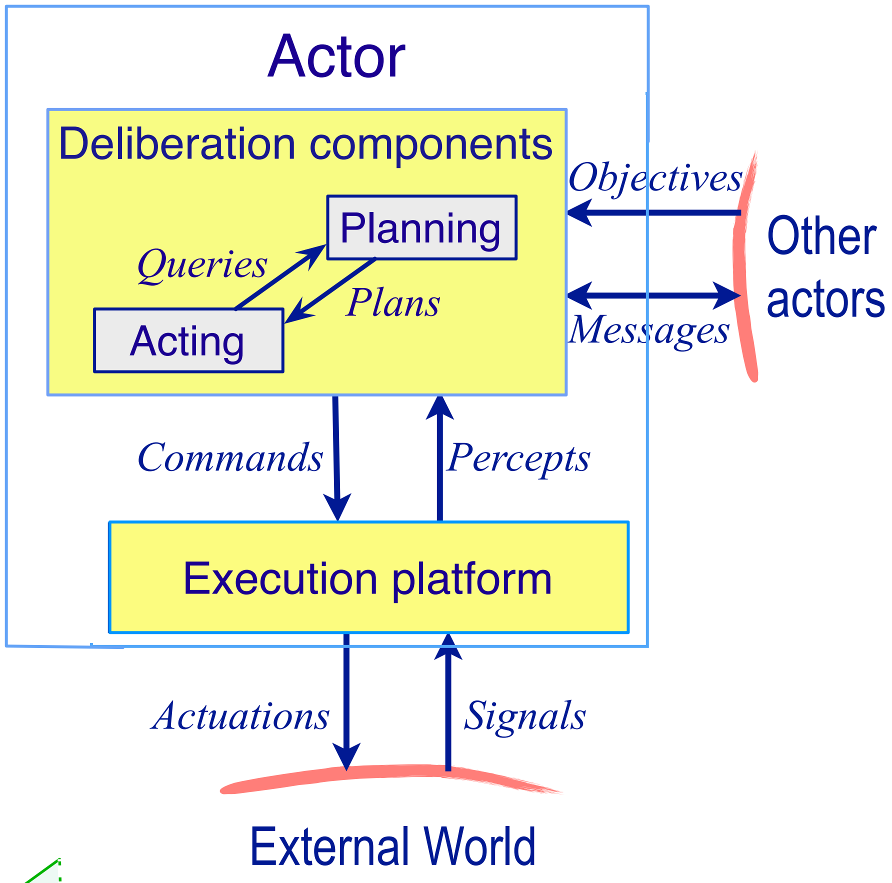


- Representation similar to what TemPlan uses*
  - But no tasks
  - Methods for goals, not tasks
- Planning algorithm similar to TemPlan
  - But doesn't use plan-space planning
  - It steps a "current time" left-to-right
    - resolves flaws in the order that it comes to them

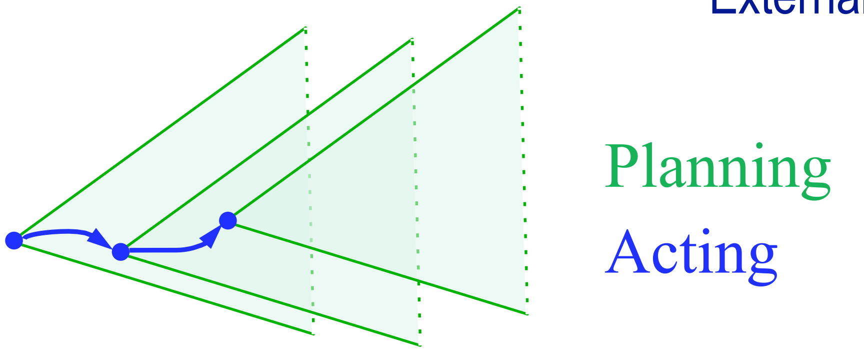
* Chapter 4 of Ghallab, Nau, and Traverso, *Automated Planning and Acting*, Cambridge University Press, 2016.

Motivation

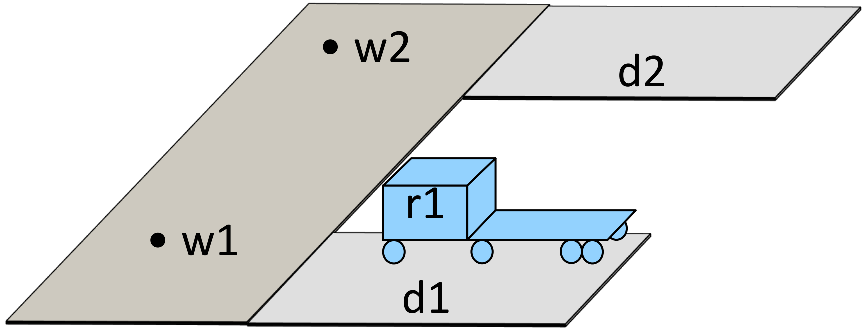
- Recall:
  - HTN planning: SHOP, SHOP2, SHOP3, Pyhop
  - HGN planning: GDP, Godel
  - Integrated acting and planning: RAE+UPOM
- Decompose tasks/goals left-to-right
  - Always know current state
  - State can be an arbitrary data structure
  - Preconditions & effects can be arbitrary computations
- Broadens scope of applicability
- Enables planning with real-time constraints



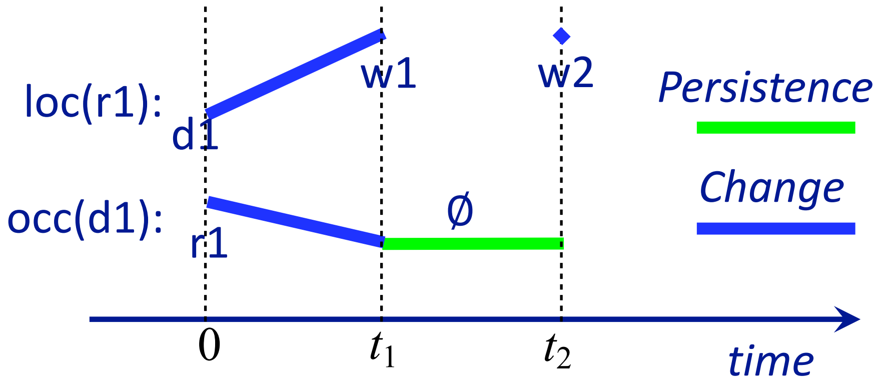
- Want to do the same for temporal planning



Chronicles



- A chronicle includes
  - *temporal assertions*
    - *change*, e.g., $[0,t_1]$ loc(r1):(dock1,w1)
    - *persistence*, e.g., $[t_1,t_2]$ occupant(dock1) = ∅
      - if $t_1 = t_2$ then the assertion is *instantaneous*
  - *constraints*
    - on *objects*, e.g., $r \neq$ r1
    - on *time points*, e.g., $t_0 < t_1 < t_2$
- Let C = {all the constraints}
- Divide the temporal assertions into two sets
  - $\mathcal{T}$ = {the *unsupported* assertions}
    - how did r1 get to w2 ?
  - S = {the *supported* assertions}
    - we know how occupant(d1) became ∅
    - we're told that initially loc(r1) = dock1, occupant(d1) = r1



ϕ:

S: $[t_0,t_1]$ loc(r1):(d1,w1),
$[t_0,t_1]$ occ(d1):(r1,∅),
$[t_1,t_2]$ occ(d1) = ∅

$\mathcal{T}$: $[t_2]$ loc(r1) = w2

C: adj(d1,w1), adj(d2,w2),
conn (w1,w2), conn (w2,w1),
$t_0 < t_1 < t_2$

<u>Abbreviations</u>:

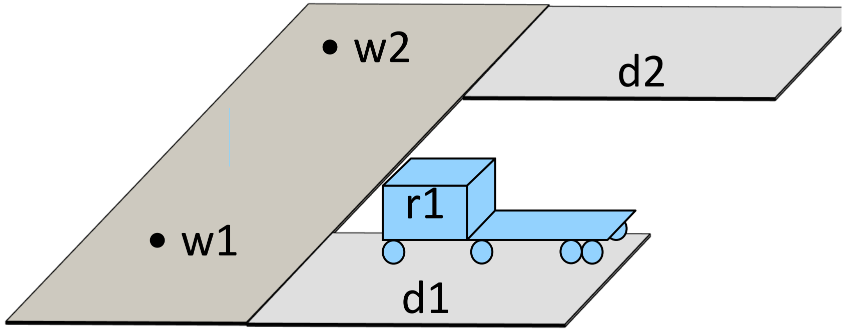
adjacent:	adj
connected:	conn
dock1, dock2:	d1, d2
empty:	∅
occupant:	occ

Actions

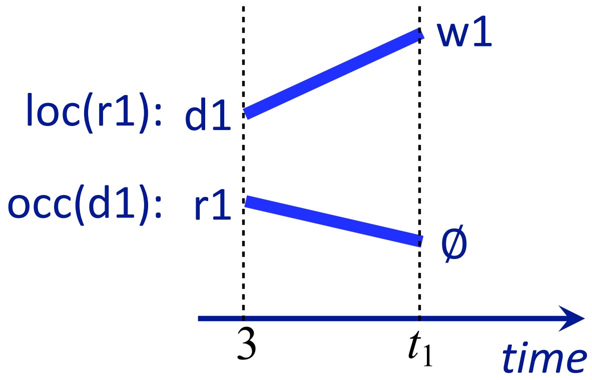
- *Action template*:
  - head (name and parameter list)
  - starting and ending times t_s, t_e
  - set $\mathcal{T}$ of unsupported assertions
  - set C of constraints
- Action: an *instance* of an action
  - substitute values for variables
- No supported assertions
  - if you insert an action into a chronicle, you need to figure out how to support it

$[t_s,t_e]$ leave(r,d,w):
 $\mathcal{T}$: $[t_s,t_e]$ loc(r):(d,w),
  $[t_s,t_e]$ occ(d):(r,∅)
 C: adj(d,w)

$[3,t_1]$ leave(r1,d1,w1):
 $\mathcal{T}$: $[3,t_1]$ loc(r1):(d1,w1),
  $[3,t_1]$ occ(d1):(r1,∅)
 C: adj(d1,w1)



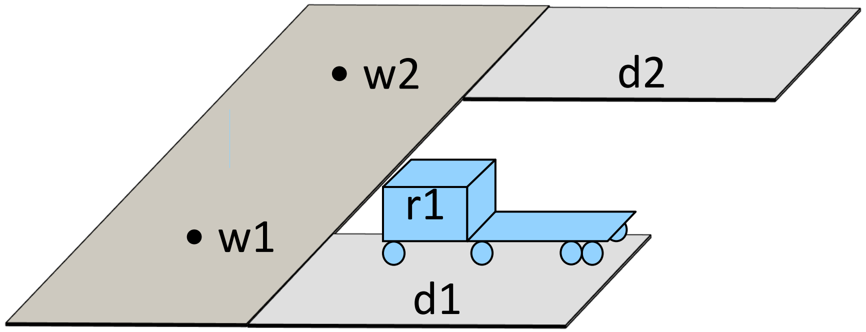
$[3,t_1]$ leave(r1,d1,w1)



Methods

- *Method*:
  - head (name and parameter list)
  - starting and ending times t_s, t_e
  - set $\mathcal{T}$ of *subgoals*: unsupported persistence assertions
    - No change assertions
    - Can't make a change happen, can only create subgoals
- *Method instance*: substitute values for variables

In Ghallab et al (2016), the body of a method contained *no* temporal assertions – just tasks and method instances



$[t_s,t_e]$ m-move(r,d,d',w,w')

$\mathcal{T}$: $[t_s]$ loc$(r) = d'$
$[t_1]$ loc$(r) = w'$
$[t_2]$ loc$(r) = w$
$[t_e]$ loc$(r) = d$

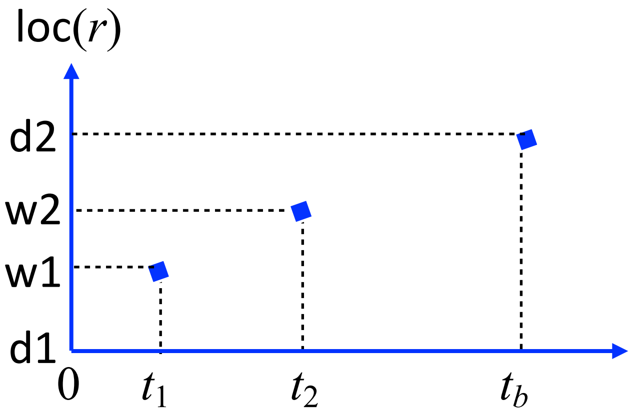
C: adj(d,w), conn(w,w'),
adj(d',w'),
$t_s < t_1 < t_2 < t_e$

$[0,t_b]$ m-move(r1,d1,d2,w1,w2)

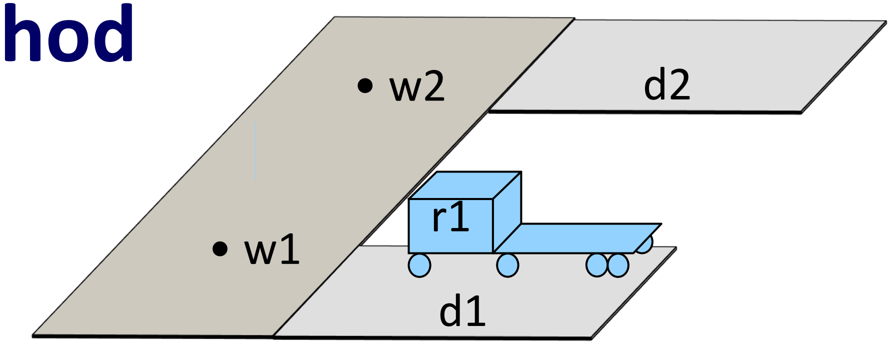
$\mathcal{T}$: $[0]$ loc(r1) = d1
$[t_1]$ loc(r1) = w1
$[t_2]$ loc(r1) = w2
$[t_b]$ loc(r1) = d2

C: adj(d1,w1), conn(w1,w2),
adj(d2,w2),
$0 < t_1 < t_2 < t_b$

$[0,t_b]$ m-move(r1,d1,d2,w1,w2)



Applicable action or method



- Let $\phi = (\mathcal{S},\mathcal{T},\mathcal{C})$ be a chronicle
  - a = an action or method
- Let:
  - Let A = {all temporal assertions in a whose starting time is the same as a's starting time}
  - a is *applicable* in ϕ *at time* t if
    - (1) a's starting time is t
    - (2) ϕ causally supports A
    - (3) a causally supports a temporal assertion $\alpha \in \mathcal{T}$
- Note: (3) prevents action chaining
  - Analogous to a restriction in GDP
  - We could omit (3), but then we would need to figure out how to control the search

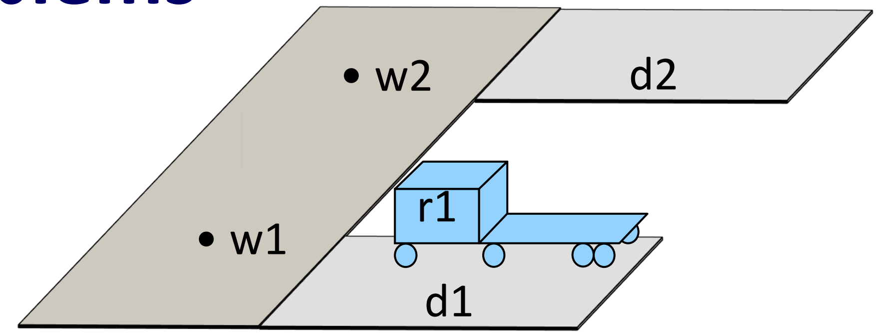
A = {[0,t_1] loc(r1):(d1,w1),
[0,t_1] occ(d1):(r1,Ø)}

α = {[t_1] loc(r1) = w1}

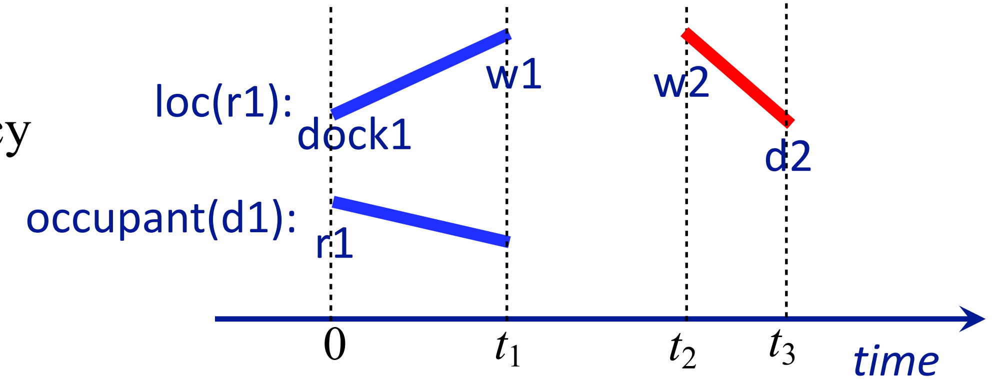
[0,t_1] leave(r1,d1,w1):
$\mathcal{T}$: [0,t_1] loc(r1):(d1,w1),
[0,t_1] occ(d1):(r1,Ø)
$\mathcal{C}$: adj(d1,w1)

ϕ_0:
$\mathcal{S}$: [0] loc(r1) = d1,
[0] occ(d1) = r1
$\mathcal{T}$: [t_1] loc(r1) = w1
$\mathcal{C}$: adj(d1,w1), adj(d2,w2), conn(w1,w2), conn(w2,w1), $0 < t_b$

Chronicles as Planning Problems



- As in plan-space planning, need to resolve all *flaws*
- In TPS and TemPlan
  - *Unsupported* temporal assertion
    - e.g., $[t_2,t_3]$ loc(r1):(w2,d2)
    - Goal: cause loc(r1)=w2 at time t_2
  - Resolver: a supported action
  - *Threats*: things that may interfere with chronicle's consistency
    - e.g., $[t_0,t_1]$ loc(r1):(d1,w1), $[t_2,t_3]$ loc(r1):(w2,d2)
    - if $0 < t_2 \le t_1 \le t_3$, r1 is at two places at the same time
  - Resolver: a new constraint $t_1 < t_2$



- To get something more like GDP and GoDel:
  - Need a current time *now* that we step left-to-right
  - Resolve flaws that can be resolved at time *now*

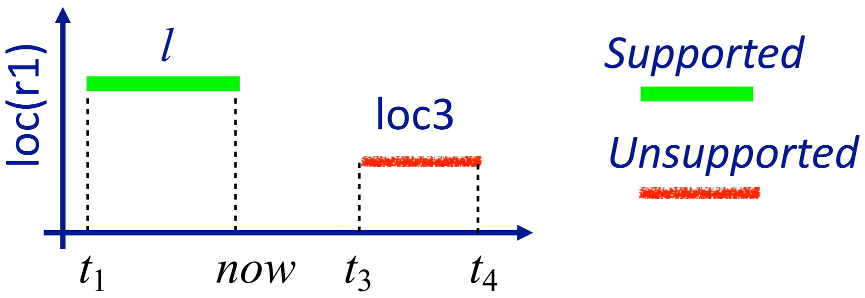
ϕ:

$\mathcal{S}$: $[0,t_1]$ loc(r1):(d1,w1), $[0,t_1]$ occupant(d1):(r1,Ø)

$\mathcal{T}$: $[t_2,t_3]$ loc(r1):(w2,d2)

$\mathcal{C}$: adj(d1,w1), adj(d2,w2), conn(w1,w2), conn(w2,w1), $t_0 < t_1$, $t_2 < t_3$

Flaws

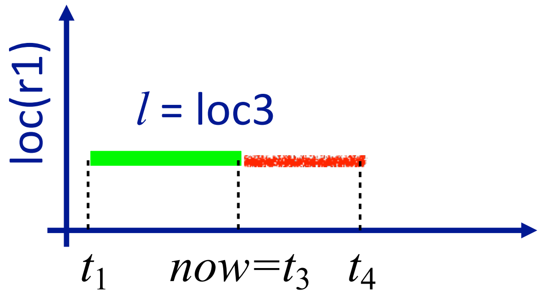


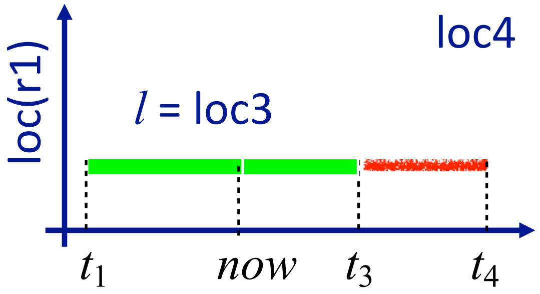
- Flaws are the same as in TPS and TemPlan
  - But resolvers must be usable at time *now*
  - Can’t change the past

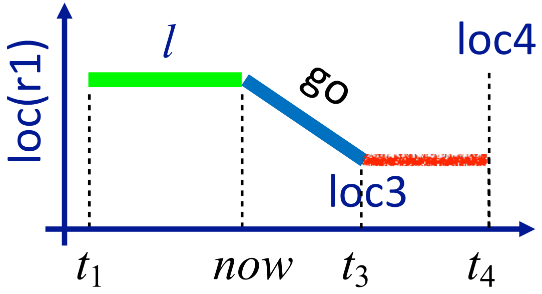
Flaw type 1: unsupported temporal assertion α

- e.g., what causes r1 to be at loc3 at time t_3?

- Resolvers that are usable at time *now*
  - Add constraints to support α from an assertion $\beta \in \mathcal{T}$ such that end(β) = *now*
    - e.g., l = loc3, $t_3 = now$
  - Add a supported persistence assertion β that starts at time *now* and supports α
    - e.g., l = loc3, [t_2, t_3] loc(r1) = loc3
  - Apply an action *or method instance* that’s applicable in ϕ at time *now* and supports α
    - e.g., a = [t_2,t_3] go(r1,l,loc3)





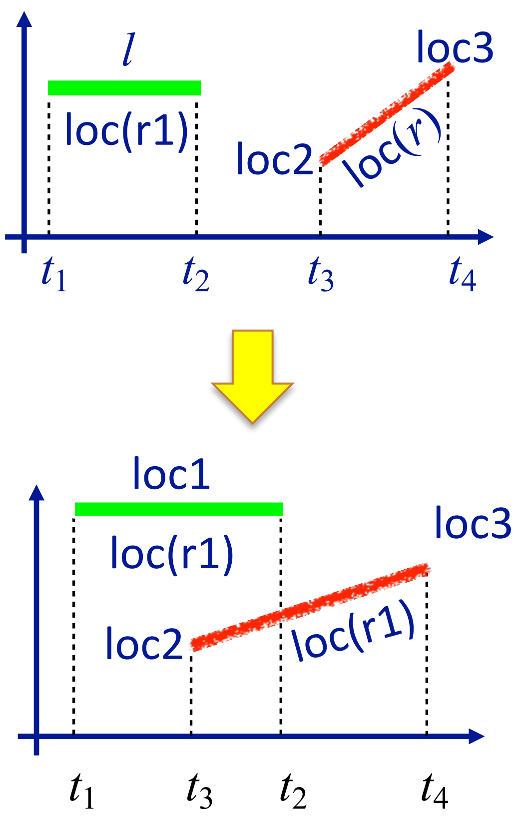


Flaws (2)

Flaw type 2: a pair of temporal assertions $\{\alpha,\beta\}$ that *possibly conflict*

- i.e., they can have inconsistent instances

  e.g., if $t_3 < t_2$, `r1` is in two places at once



- As in `TPS` and `TemPlan`, but resolvers must be usable at time *now*
  - Can't change the past
- *Resolvers* applicable at time *now*:
  - Various ways of adding constraints to resolve the inconsistencies
  - I'll skip them
    - Lots of special cases
    - I'm not sure I have all of them right

Planning Algorithm

- Basic idea:
  - Variable *now* representing current time
  - Step *now* through the time points in ϕ, in an order that satisfies the constraints in C
    - As we go, add time constraints to enforce the order we're creating
  - For each value of *now*,
    - resolve some of the flaws that can be resolved at time *now*
      - *i.e.,* flaws having resolvers that are applicable at time *now*
    - Choose what time point(s) to use for the next value of *now*

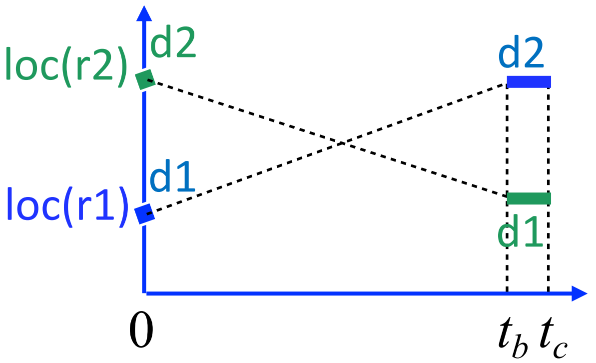
TGN-Forward-Plan(ϕ, Σ)
    now = ϕ's starting time; *plan* = $\emptyset$
    loop:
        if ϕ contains no flaws then return (ϕ, *plan*)
        if ϕ contains an unresolvable flaw then return failure
        nondeterministically choose
            $F \subseteq$ {flaws that can be resolved at time *now*}
        if $F \neq \emptyset$ then for every $f \in F$
            nondeterministically choose a resolver ρ for f
                that can be used at time *now*
            ϕ = Transform(ϕ, ρ)
            add ρ to *plan*
        Next = {time points in ϕ that may come next}
        nondeterministically choose $Next^* \subseteq Next$
        if $Next^* \neq \emptyset$ then
            $next \leftarrow$ any $t \in Next$
            $C \leftarrow C \cup \{now < next\} \cup \{t = next \mid t \in Next^*\}$
            $now \leftarrow next$

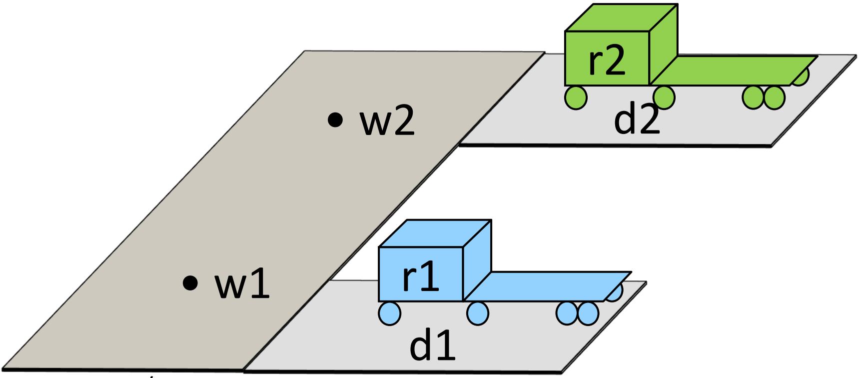
ϕ_0:

S: [0] loc(r1) = d1, [0] loc(r2) = d2,
[0] occ(d1) = r1, [0] occ(d2) = r2

$\mathcal{T}$: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$0 < t_b$





Example

- d1, d2 are loading docks
- only big enough to hold one vehicle at a time

Variables:

$r \in Robots$ = {r1,r2}

$d, d' \in Docks$ = {d1,d2}

$w, w' \in Waypoints$ = {w1,w2}

$t, t', t_* \in Timepoints$

Method:

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

$\mathcal{T}$: $[t_s]$ loc$(r) = d$
$[t_1]$ loc$(r) = w$
$[t_2]$ loc$(r) = w'$
$[t_e]$ loc$(r) = d'$

C: adj(d,w), conn(w,w'),
adj(d',w'),
$t_s < t_1 < t_2 < t_e$

Action templates:

leave(r,d,w):

$\mathcal{T}$: $[t_s, t_e]$ loc(r):(d,w),
$[t_s, t_e]$ occ(d):$(r,\emptyset)$

C: adj(d,w), $t_s+2 \le t_e$

enter(r,d,w):

$\mathcal{T}$: $[t_s, t_e]$ loc(r):(w,d),
$[t_s, t_e]$ occ(d):$(\emptyset,r)$

C: adj(w,d)

navigate(r,w,w'):

$\mathcal{T}$: $[t_s, t_e]$ loc(r):(w,w')

C: conn(w,w')

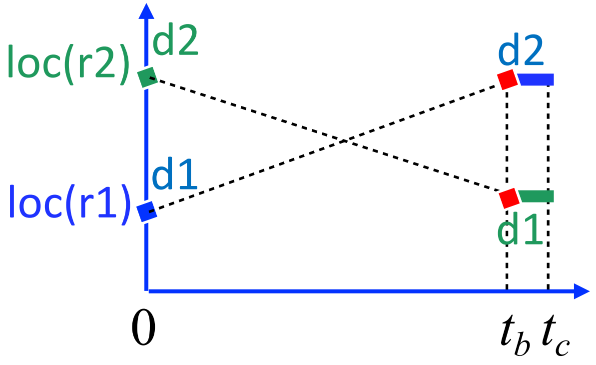
Example

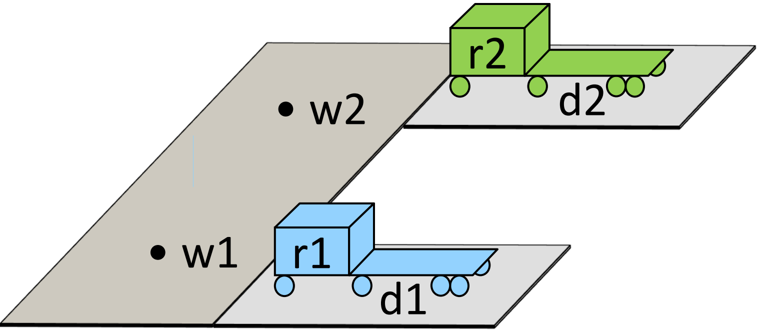
ϕ_0:

$\mathcal{S}$: [0] loc(r1) = d1, [0] loc(r2) = d2,
[0] occ(d1) = r1, [0] occ(d2) = r2,

$\mathcal{T}$: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1

$\mathcal{C}$: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$0 \le t_b$





$now = 0$

Open goals that can be resolved:
$[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1

- Resolve both

$[0,t_b]$ m-move(r1,d1,d2,w1,w2)

$\mathcal{T}$: [0] loc(r1) = d1
$[t_1]$ loc(r1) = w1
$[t_2]$ loc(r1) = w2
$[t_b]$ loc(r1) = d2

$\mathcal{C}$: adj(d1,w1), conn(w1,w2),
adj(d2,w2), $0 < t_1 < t_2 < t_b$

$[0,t_b]$ m-move(r2,d2,d1,w2,w1)

$\mathcal{T}$: [0] loc(r2) = d2
$[t'_2]$ loc(r2) = w2
$[t'_2]$ loc(r2) = w1
$[t_b]$ loc(r2) = d1

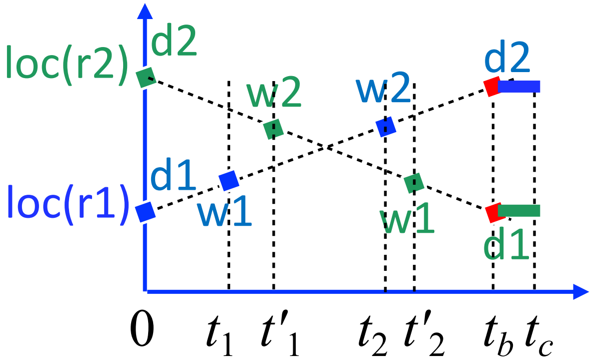
$\mathcal{C}$: adj(d2,w2), conn(w2,w1),
adj(d1,w1), $0 < t'_1 < t'_2 < t_b$

ϕ_1:

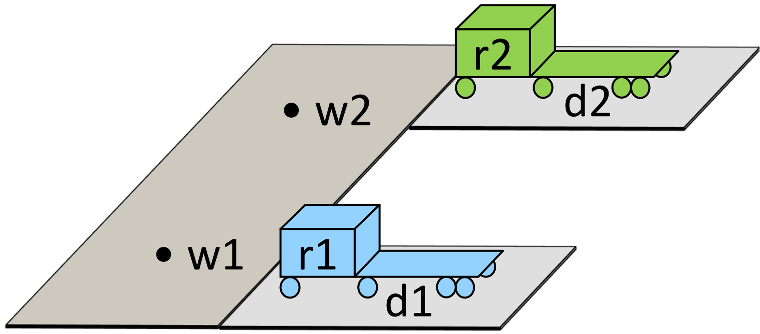
$\mathcal{S}$: [0] loc(r1) = d1, [0] loc(r2) = d2,
[0] occ(d1) = r1, [0] occ(d2) = r2,
$[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1

$\mathcal{T}$: $[t_1]$ loc(r1) = w1, $[t'_1]$ loc(r2) = w2,
$[t_2]$ loc(r1) = w2, $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

$\mathcal{C}$: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$0 < t_1 < t_2 < t_b$, $0 < t'_1 < t'_2 < t_b$



$Next = \{t_1,t'_1\}$
$Next^* \leftarrow \emptyset$
now doesn't change



ϕ_1:

$\mathcal{S}$: [0] loc(r1) = d1, [0] loc(r2) = d2,
[0] occ(d1) = r1, [0] occ(d2) = r2,
$[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1

$\mathcal{T}$: $[t_1]$ loc(r1) = w1, $[t'_1]$ loc(r2) = w2,
$[t_2]$ loc(r1) = w2, $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

$\mathcal{C}$: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$0 < t_1 < t_2 < t_b\,,\ 0 < t'_1 < t'_2 < t_b$

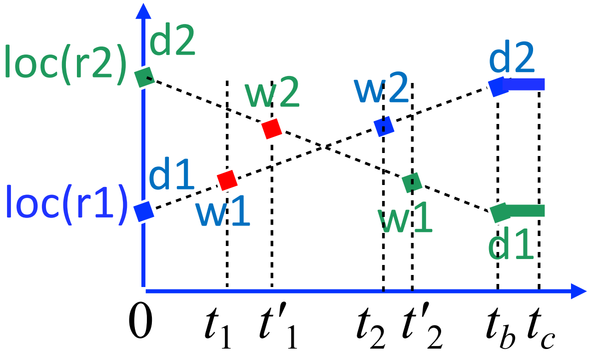
Example

$now = 0$

Open goals that can be resolved:

$[t_1]$ loc(r1) = w1,
$[t'_1]$ loc(r2) = w2

- Resolve both



$[0,t_1]$ leave(r1,d1,w1)

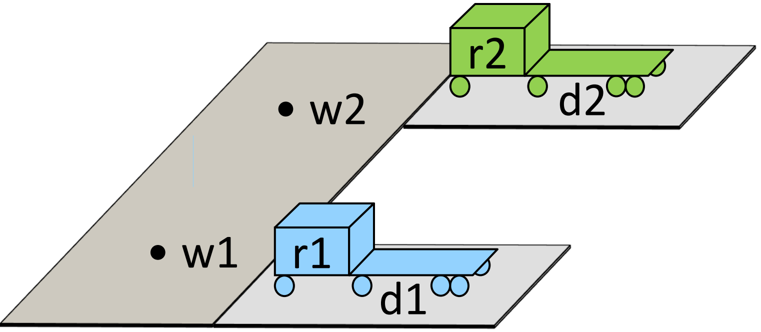
$\mathcal{T}$: $[0,t_1]$ loc(r1):(d1,w1)
$[0,t_1]$ occ(d1):(r1,Ø)

$\mathcal{C}$: adj(d1,w1), $2 \le t_1$

$[0,t'_1]$ leave(r2,d2,w2)

$\mathcal{T}$: $[0,t'_1]$ loc(r2):(d2,w2)
$[0,t'_1]$ occ(d2):(r2,Ø)

$\mathcal{C}$: adj(d1,w1), $2 \le t'_1$

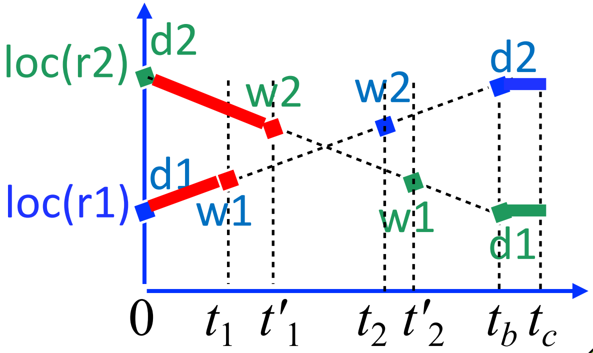


ϕ_2:

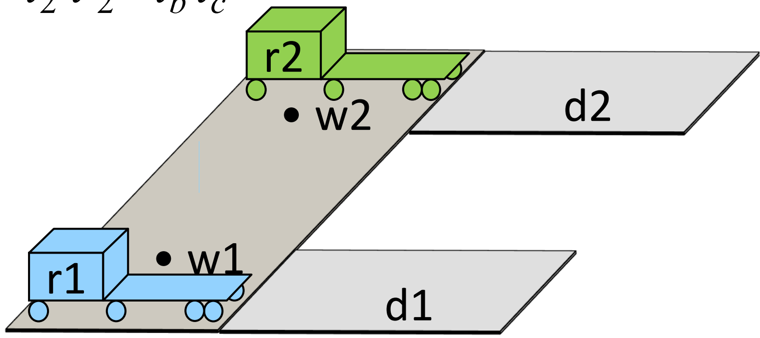
$\mathcal{S}$: [0] loc(r1) = d1, [0] loc(r2) = d2,
[0] occ(d1) = r1, [0] occ(d2) = r2,
$[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[t_1]$ loc(r1) = w1, $[t'_1]$ loc(r2) = w2,
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,Ø), $[0,t'_1]$ occ(d2):(r2,Ø),

$\mathcal{T}$: $[t_2]$ loc(r1) = w2, $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

$\mathcal{C}$: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \le t_1 < t_2 < t_b\,,\ 2 \le t'_1 < t'_2 < t_b$



$Next = \{t_1, t'_1\}$
$Next^* \leftarrow \{t_1\}$
$now \leftarrow t_1$



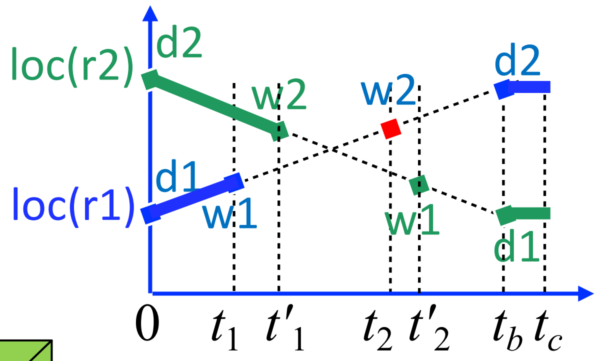
Example

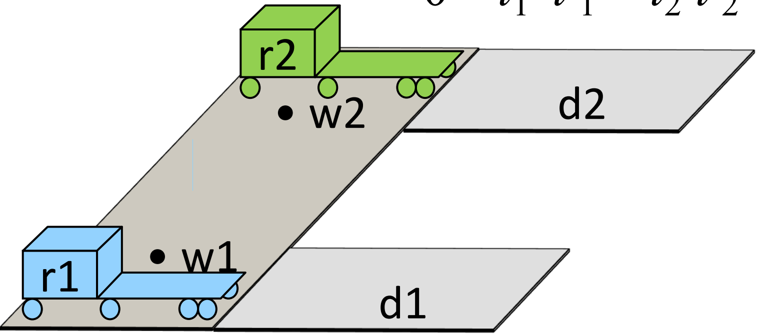
ϕ_2:

S: ~~[0] loc(r1) = d1, [0] loc(r2) = d2,~~
~~[0] occ(d1) = r1, [0] occ(d2) = r2,~~
$[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
~~$[t_1]$ loc(r1) = w1, $[t'_1]$ loc(r2) = w2,~~
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅)

$\mathcal{T}$: $[t_2]$ loc(r1) = w2, $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$





$now = t_1$

Open goal that can be resolved:

$[t_2]$ loc(r1) = w2

- Resolve it

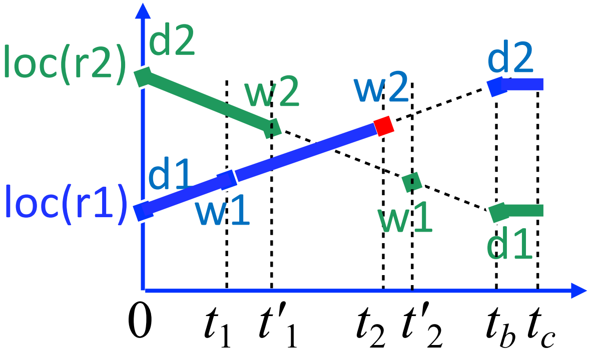
$[t_1,t_2]$ navigate(r1,w1,w2):
$\mathcal{T}$: $[t_1,t_2]$ loc(r1):(w1,w2),
C: conn(w1,w2), $t_1 < t_2$

ϕ_3:

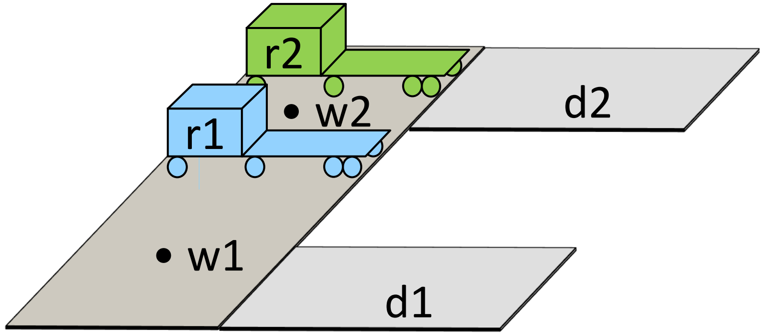
S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2),
$[t_2]$ loc(r1) = w2

$\mathcal{T}$: $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$



$Next = \{t'_1, t_2\}$
$Next^* \leftarrow \{t'_1\}$
$now \leftarrow t'_1$



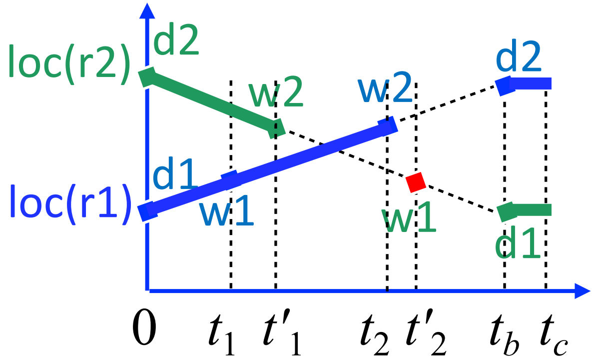
Example

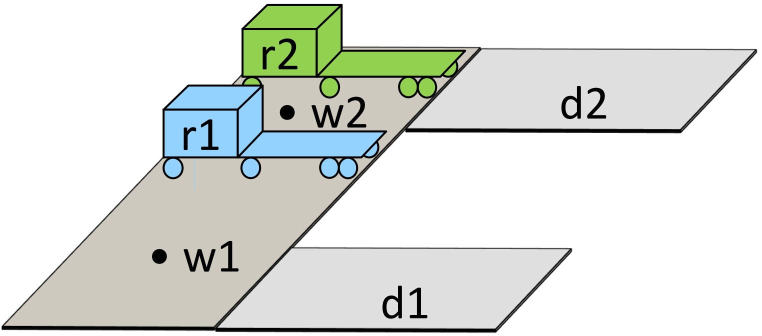
ϕ_3:

S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2),
~~$[t_2]$ loc(r1) = w2~~

$\mathcal{T}$: $[t'_2]$ loc(r2) = w1,
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$





$now = t'_1$

Open goal that can be resolved:

$[t'_2]$ loc(r2) = w1

- Resolve it

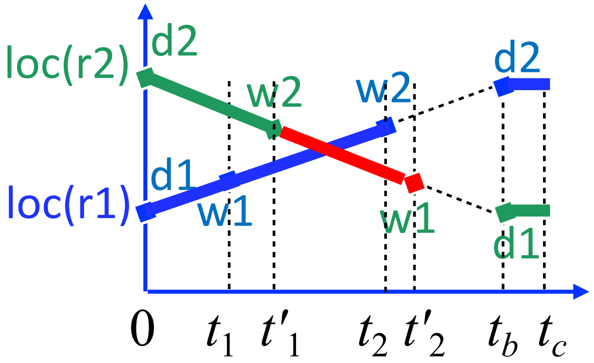
$[t'_1,t'_2]$ navigate(r2,w2,w1):
$\mathcal{T}$: $[t'_1,t'_2]$ loc(r2):(w2,w1),
C: conn(w2,w1), $t'_1 < t'_2$

ϕ_4:

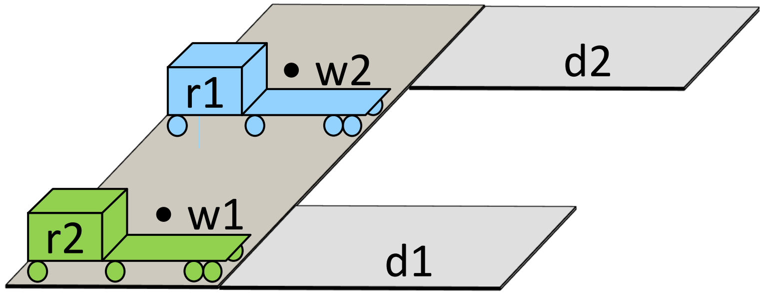
S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
$[t'_2]$ loc(r2) = w1

$\mathcal{T}$: $[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$



$Next = \{t_2, t'_2\}$
$Next^* \leftarrow \{t_2, t'_2\}$
$now \leftarrow t_2 = t'_2$



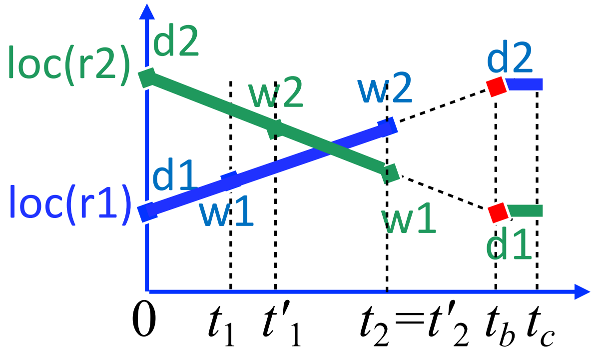
Example

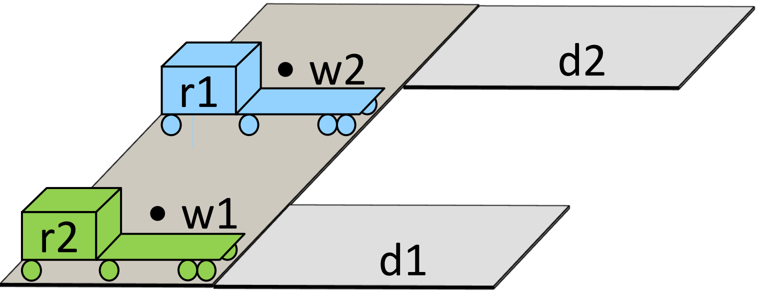
ϕ_4:

S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
~~$[t'_2]$ loc(r2) = w1~~

$\mathcal{T}$: $[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \le t_1 < t_2 < t_b\,,\ 2 \le t'_1 < t'_2 < t_b$





now = $t_2 = t'_2$

Open goals that can be resolved:

$[t_b,t_c]$ loc(r1) = d2
$[t_b,t_c]$ loc(r2) = d1

- Resolve both

$[t_2,t_b]$ enter(r1,d2,w2)
$\mathcal{T}$: $[t_2,t_b]$ loc(r1):(w2,d2)
$[t_2,t_b]$ occ(d2):(∅,r1)
C: adj(d2,w2), $t_2 < t_b$

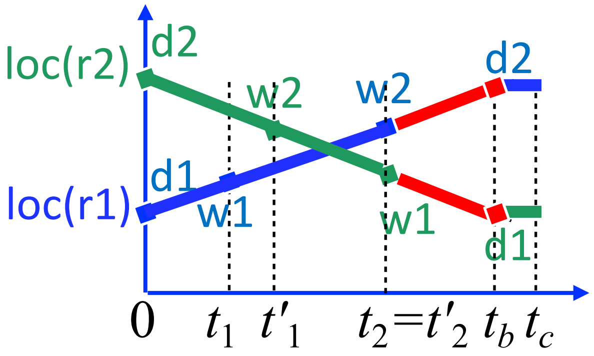
$[t'_2,t_b]$ enter(r2,d1,w1)
$\mathcal{T}$: $[t'_2,t_b]$ loc(r2):(w1,d1)
$[t'_2,t_b]$ occ(d1):(∅,r2)
C: adj(d1,w1), $t'_2 < t_b$

ϕ_5:

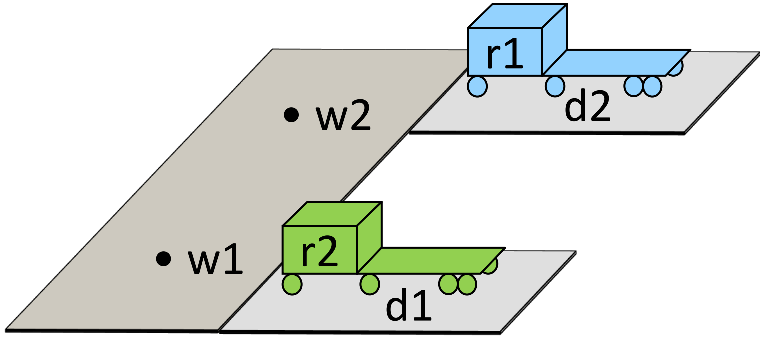
S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
$[t_2,t_b]$ loc(r1):(w2,d2), $[t'_2,t_b]$ loc(r2):(w1,d1),
$[t_2,t_b]$ occ(d2):(∅,r1), $[t'_2,t_b]$ occ(d1):(∅,r2),
$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

$\mathcal{T}$:

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \le t_1 < t_2 < t_b\,,\ 2 \le t'_1 < t'_2 < t_b\,,\ t_2 = t'_2$



$Next = \{t_b\}$
$Next^* \leftarrow \{t_b\}$
$now \leftarrow t_b$



Example

ϕ_5:

S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
$[t_2,t_b]$ loc(r1):(w2,d2), $[t'_2,t_b]$ loc(r2):(w1,d1),
$[t_2,t_b]$ occ(d2):(∅,r1), $[t'_2,t_b]$ occ(d1):(∅,r2),
~~$[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1~~

$\mathcal{T}$:

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$, $t_2 = t'_2$

$now = t_b$

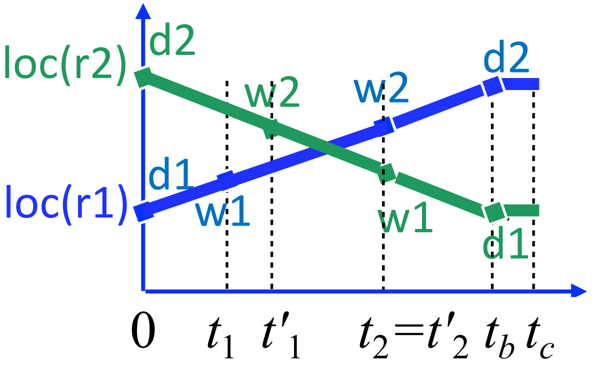
No flaws

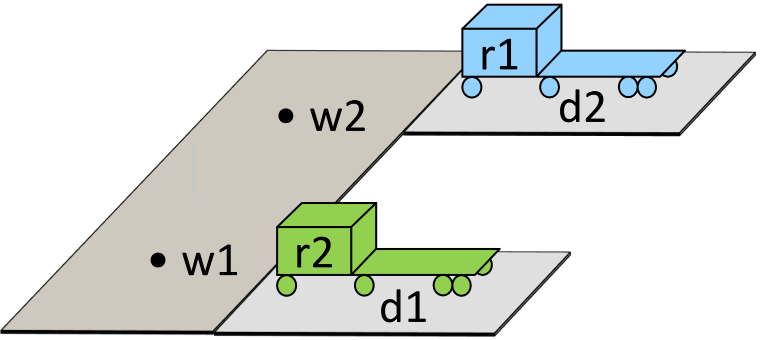
ϕ_6:

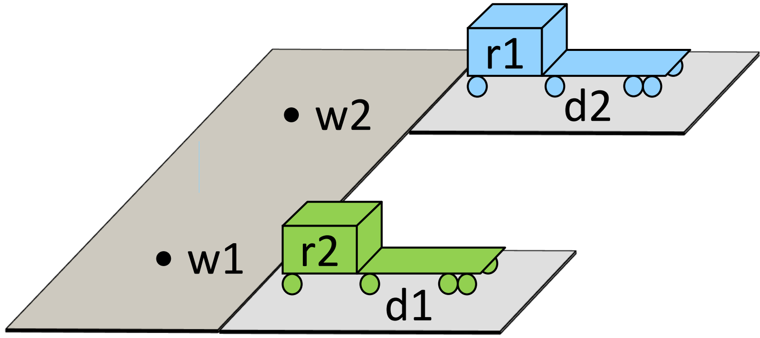
S: $[t_b,t_c]$ loc(r1) = d2, $[t_b,t_c]$ loc(r2) = d1
$[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
$[t_2,t_b]$ loc(r1):(w2,d2), $[t'_2,t_b]$ loc(r2):(w1,d1),
$[t_2,t_b]$ occ(d2):(∅,r1), $[t'_2,t_b]$ occ(d1):(∅,r2),

$\mathcal{T}$:

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \leq t_1 < t_2 < t_b$, $2 \leq t'_1 < t'_2 < t_b$, $t_2 = t'_2$







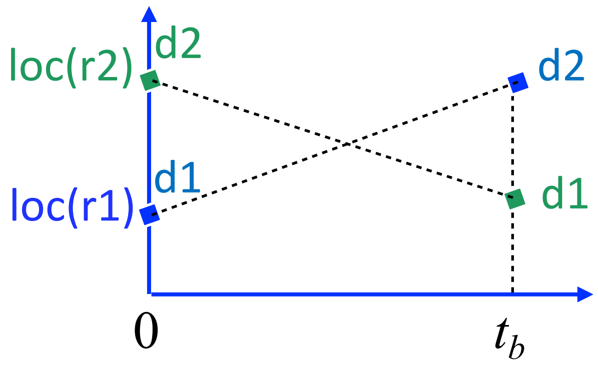
Example

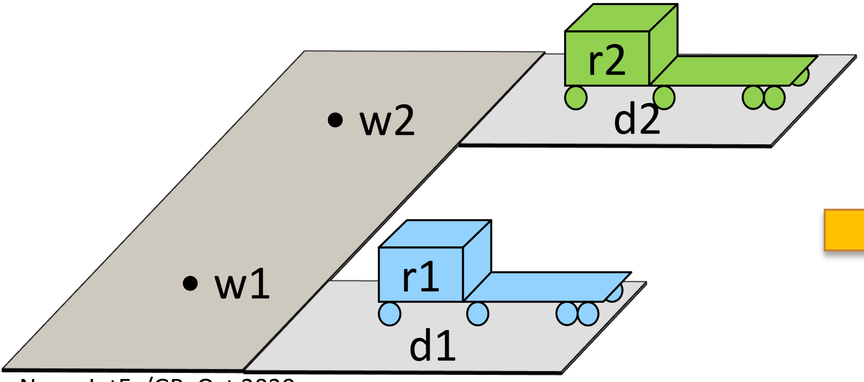
ϕ_0:

S: [0] loc(r1) = d1, [0] loc(r2) = d2

$\mathcal{T}$: $[t_b]$ loc(r1) = d2, $[t_b]$ loc(r2) = d1

C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$0 < t_b$





Temporal plan:

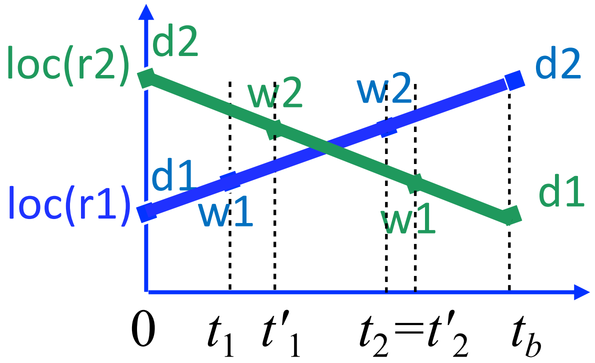
{$[0,t_1]$ leave(r1,d1,w1),
$[0,t'_1]$ leave(r2,d2,w2),
$[t_1,t_2]$ navigate(r1,w1,w2),
$[t'_1,t'_2]$ navigate(r2,w2,w1),
$[t_2,t_b]$ enter(r1,d2,w2),
$[t'_2,t_b]$ enter(r2,d1,w1)}

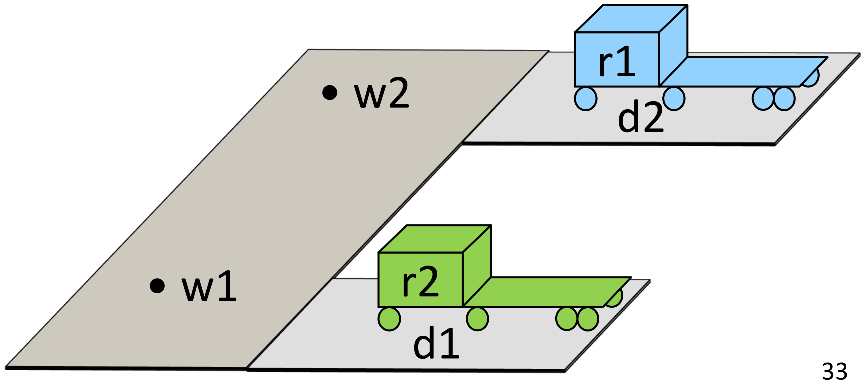
ϕ_6:

S: $[0,t_1]$ loc(r1):(d1,w1), $[0,t'_1]$ loc(r2):(d2,w2),
$[0,t_1]$ occ(d1):(r1,∅), $[0,t'_1]$ occ(d2):(r2,∅),
$[t_1,t_2]$ loc(r1):(w1,w2), $[t'_1,t'_2]$ loc(r2):(w2,w1),
$[t_2,t_b]$ loc(r1):(w2,d2), $[t'_2,t_b]$ loc(r2):(w1,d1),
$[t_2,t_b]$ occ(d2):(∅,r1), $[t'_2,t_b]$ occ(d1):(∅,r2),

$\mathcal{T}$:

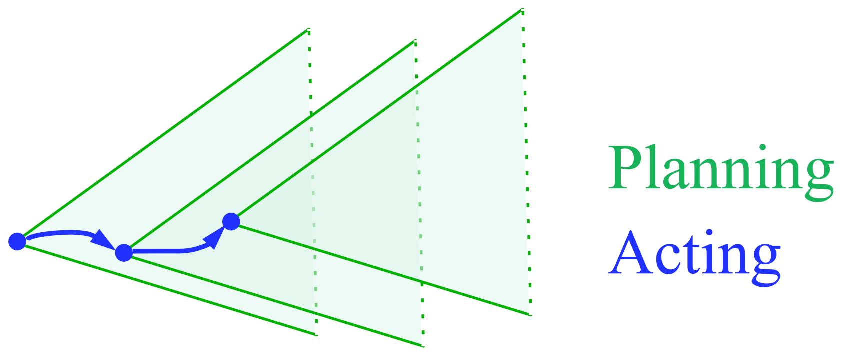
C: adj(d1,w1), adj(d2,w2),
conn(w1,w2), conn(w2,w1),
$2 \le t_1 < t_2 < t_b$, $2 \le t'_1 < t'_2 < t_b$, $t_2 = t'_2$





Contributions

- Temporal network formalism is mostly the same as in Ghallab, Nau, & Traverso
  - Two differences:
    - No tasks; temporal methods achieve goals
      - Will facilitate goal reasoning
  - Left-to-right planning algorithm (like GDP and GoDeL, but temporal)
    - Lower branching factor than plan-space planning
    - Always knows current state
    - Will facilitate
      - Online planning (integration of planning and acting)
      - Simulation-based planning (like RAE+UPOM)
      - Reasoning about uncertainty



Questions

- What's missing?
  - The actor
  - How to reason about uncertainty?
    - action outcomes; action durations; exogenous events
    - Generalize the state and action definitions
    - Incorporate Monte Carlo rollouts analogous to those in RAE+UPOM
  - Theoretical results: correctness, completeness, complexity, expressivity, …
  - Implementation, testing
  - How to learn actions and methods?
  - Goal reasoning?

- Is TGN-Forward-Plan the right algorithm?
- Is it even the right approach?
  - Perhaps use something like Linear Temporal Logic

- *Anything else?*