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

Goal Skills

Goal Network

Temporal

Method

Subgoal

Goal Skill

Top-level goal

Key

have(house)

construct-house(dirt)

select-site

build-walls(dirt)

build-door(wood)

have(dirt)

have(dirt)

have(dirt)

have(wood)

have(wood)

defeated(hostile)

mined(dirt)

mined(dirt)

placedoor)

craft(door)

built-wall(North)

aimed

approached

mined

collected

strafed

hit

killed

have(house)

have(dirt)

have(dirt)

have(wood)

have(wood)
Background

Temporal goal networks

HTN planning
HGN planning
Planning and acting

Plan-space planning
Goal reasoning
Temporal planning

Temporal goal networks
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?
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(UMD,UCLA)

action: buy-ticket(DCA,LAX)

method: travel-by-taxi(UMD,DCA)

action: call-taxi(UMD)

method: travel-by-flying(UMD,UCLA)

action: purchase-ticket(DCA,LAX)

method: travel-by-taxi(UMD,DCA)

action: ride-taxi(UMD,DCA)

action: walk(taxi,building)
```

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(UMD,UCLA)
```

```
goal: at(UCLA)
```

```
action: have-ticket(DCA,LAX)
goal: at(DCA)
goal: at(LAX)
goal: at(UCLA)
```

```
method: travel-by-taxi(UMD,DCA)
```

```
action: call-taxi(UMD)
action: ride-taxi(UMD,DCA)
action: walk(taxi,building)
```

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 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)

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

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*

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 $f$
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
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 $t = \{\text{values of state-variable at time } t\}$

- *Chronicle representation:*
  - notation for a collection of timelines and constraints

- TPS planning algorithm*
  - temporal version of plan-space planning


Background
Hierarchical Temporal Planning

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

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

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
A chronicle includes

- **temporal assertions**
  - *change*, e.g., \([0, t_1]\) \(\text{loc}(r_1):(\text{dock1}, w_1)\)
  - *persistence*, e.g., \([t_1, t_2]\) \(\text{occupant}(\text{dock1}) = \emptyset\)
    - if \(t_1 = t_2\) then the assertion is *instantaneous*

- **constraints**
  - on *objects*, e.g., \(r \neq r_1\)
  - on *time points*, e.g., \(t_0 < t_1 < t_2\)

- Let \(C = \{\text{all the constraints}\}\)

- Divide the temporal assertions into two sets
  - \(T = \{\text{the unsupported assertions}\}\)
    - how did \(r_1\) get to \(w_2\)?
  - \(S = \{\text{the supported assertions}\}\)
    - we know how \(\text{occupant}(d_1)\) became \(\emptyset\)
    - we’re told that initially \(\text{loc}(r_1) = \text{dock1}, \text{occupant}(d_1) = r_1\)

\[
\begin{align*}
\phi : \quad & [t_0, t_1] \text{loc}(r_1):(d_1, w_1), \\
& [t_0, t_1] \text{occ}(d_1):(r_1, \emptyset), \\
& [t_1, t_2] \text{occ}(d_1) = \emptyset \\
S : \quad & [t_0, t_1] \text{loc}(r_1):(d_1, w_1), \\
& [t_0, t_1] \text{occ}(d_1):(r_1, \emptyset), \\
& [t_1, t_2] \text{occ}(d_1) = \emptyset \\
T : \quad & [t_2] \text{loc}(r_1) = w_2 \\
C : \quad & \text{adj}(d_1, w_1), \text{adj}(d_2, w_2), \\
& \text{conn}(w_1, w_2), \text{conn}(w_2, w_1), \\
& t_0 < t_1 < t_2
\end{align*}
\]

Abbreviations:
- adjacent: \(\text{adj}\)
- connected: \(\text{conn}\)
- dock1, dock2: \(d_1, d_2\)
- empty: \(\emptyset\)
- occupant: \(\text{occ}\)
- Action template:
  - head (name and parameter list)
  - starting and ending times $t_s$, $t_e$
  - set $T$ 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

**Actions**

\[
[t_s, t_e] \text{ leave}(r,d,w):
\]

\[
T: [t_s, t_e] \text{ loc}(r):(d,w),
\]

\[
[t_s, t_e] \text{ occ}(d):(r,\emptyset)
\]

\[
C: \text{ adj}(d,w)
\]

\[
[3, t_1] \text{ leave}(r_1,d_1,w_1):
\]

\[
T: [3, t_1] \text{ loc}(r_1):(d_1,w_1),
\]

\[
[3, t_1] \text{ occ}(d_1):(r_1,\emptyset)
\]

\[
C: \text{ adj}(d_1,w_1)
\]
• **Method:**
  ▶ head (name and parameter list)
  ▶ starting and ending times \( t_s, t_e \)
  ▶ set \( T \) of subgoals: unsupported persistence assertions
    - No change assertions
    - Can’t make a change happen, can only create subgoals
• **Method instance:** substitute values for variables

\[
[0,t_b] \text{ m-move}(r,d,d',w,w')
\]
\[
T: [t_s] \text{ loc}(r) = d' \\
[t_1] \text{ loc}(r) = w' \\
[t_2] \text{ loc}(r) = w \\
[t_e] \text{ loc}(r) = d
\]
\[
C: \text{ adj}(d,w), \text{ conn}(w,w'), \text{ adj}(d',w'), \\
\text{ } \text{ } t_s < t_1 < t_2 < t_e
\]

\[
[0,t_b] \text{ m-move}(r1,d1,d2,w1,w2)
\]
\[
T: [0] \text{ loc}(r1) = d1 \\
[t_1] \text{ loc}(r1) = w1 \\
[t_2] \text{ loc}(r1) = w2 \\
[t_b] \text{ loc}(r1) = d2
\]
\[
C: \text{ adj}(d1,w1), \text{ conn}(w1,w2), \text{ adj}(d2,w2), \\
\text{ } \text{ } 0 < t_1 < t_2 < t_b
\]

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

- Let $\phi = (S, T, C)$ be a chronicle
  - $a$ = an action or method
- Let:
  - Let $A = \{\text{all temporal assertions in } a \text{ whose starting time is the same as } a\text{’s starting time}\}$
    - $a$ is applicable in $\phi$ at time $t$ if
      1. $a$’s starting time is $t$
      2. $\phi$ causally supports $A$
      3. $a$ causally supports a temporal assertion $\alpha \in T$
- 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
Chronicles as Planning Problems

- As in plan-space planning, need to resolve all flaws.
- In TPS and TemPlan:
  - *Unsupported* temporal assertion
    - e.g., \([t_2, t_3]\) \(\text{loc}(r_1):(w_2, d_2)\)
    - Goal: cause \(\text{loc}(r_1) = w_2\) at time \(t_2\)
  - Resolver: a supported action
  - *Threats*: things that may interfere with chronicle’s consistency
    - e.g., \([t_0, t_1]\) \(\text{loc}(r_1):(d_1, w_1)\), \([t_2, t_3]\) \(\text{loc}(r_1):(w_2, d_2)\)
    - if \(0 < t_2 \leq t_1 \leq t_3\), \(r_1\) 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*
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 $\alpha$

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

- Resolvers that are usable at time *now*
  - Add constraints to support $\alpha$ from an assertion $\beta \in T$ such that $\text{end}(\beta) = \text{now}$
    - e.g., $l = \text{loc3}, t_3 = \text{now}$
  - Add a supported persistence assertion $\beta$ that starts at time *now* and supports $\alpha$
    - e.g., $l = \text{loc3}, [t_2, t_3] \text{loc}(r1) = \text{loc3}$
  - Apply an action *or method instance* that’s applicable in $\phi$ at time *now* and supports $\alpha$
    - e.g., $a = [t_2, t_3] \text{go}(r1, l, \text{loc3})$
Flaws (2)

Flaw type 2: a pair of temporal assertions \( \{a, \beta\} \) 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

TGN-Forward-Plan(\(\phi, \Sigma\))
\(\text{now} = \phi\)'s starting time; \(\text{plan} = \emptyset\)
loop:

- if \(\phi\) contains no flaws then return \((\phi, \text{plan})\)
- if \(\phi\) contains an unresolvable flaw then return failure
- nondeterministically choose \(F \subseteq \{\text{flaws that can be resolved at time } \text{now}\}\)
- if \(F \neq \emptyset\) then for every \(f \in F\)
- nondeterministically choose a resolver \(\rho\) for \(f\)
  that can be used at time \(\text{now}\)
- \(\phi = \text{Transform}(\phi, \rho)\)
- add \(\rho\) to \(\text{plan}\)

Next = \{time points in \(\phi\) that may come next\}
nonce\(deterministically choose \(\text{Next}^* \subseteq \text{Next}\)
- if \(\text{Next}^* \neq \emptyset\) then
  - \(next \leftarrow \text{any } t \in \text{Next}\)
  - \(C \leftarrow C \cup \{\text{now} < next\} \cup \{t=next \mid t \in \text{Next}^*\}\)
  - \(\text{now} \leftarrow next\)

- Basic idea:
  - Variable \(\text{now}\) representing current time
  - Step \(\text{now}\) through the time points in \(\phi\), 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 \(\text{now}\),
    • resolve some of the flaws that can be
      resolved at time \(\text{now}\)
      - \text{i.e.}, flaws having resolvers that are
        applicable at time \(\text{now}\)
    • Choose what time point(s) to use for
      the next value of \(\text{now}\)
Example

**Variables:**

- \( r \in \text{Robots} = \{r_1, r_2\} \)
- \( d, d' \in \text{Docks} = \{d_1, d_2\} \)
- \( w, w' \in \text{Waypoints} = \{w_1, w_2\} \)
- \( t, t', t^* \in \text{Timepoints} \)

**Action templates:**

- **leave** \((r; d, w)\): \[
\mathcal{T}: \ [t_s, t_e] \ \text{loc}(r):(d, w),
\ [t_s, t_e] \ \text{occ}(d):(r, \emptyset)
\]
  \(C: \ \text{adj}(d, w), \ t_s + 2 \leq t_e\)

- **enter** \((r; d, w)\): \[
\mathcal{T}: \ [t_s, t_e] \ \text{loc}(r):(w, d),
\ [t_s, t_e] \ \text{occ}(d): (\emptyset, r)
\]
  \(C: \ \text{adj}(w, d)\)

- **navigate** \((r; w, w')\): \[
\mathcal{T}: \ [t_s, t_e] \ \text{loc}(r):(w, w')
\]
  \(C: \ \text{conn}(w, w')\)

**Method:**

- \(m\text{-move}(r; d, d', w, w')\)
  \[
  \mathcal{T}: \ [t_s] \ \text{loc}(r):d
  \]
  \[
  [t_1] \ \text{loc}(r):w
  \]
  \[
  [t_2] \ \text{loc}(r):w'
  \]
  \[
  [t_e] \ \text{loc}(r):d'
  \]
  \(C: \ \text{adj}(d, w), \ \text{conn}(w, w'), \ \text{adj}(d', w'), t_s < t_1 < t_2 < t_e\)

- \(d_1, d_2\) are loading docks
- only big enough to hold one vehicle at a time

\(\phi_0:\)

- \([0] \ \text{loc}(r_1) = d_1, \ [0] \ \text{loc}(r_2) = d_2,\)
- \([0] \ \text{occ}(d_1) = r_1, \ [0] \ \text{occ}(d_2) = r_2\)

\(\mathcal{T}: \ [t_b, t_c] \ \text{loc}(r_1) = d_2, \ [t_b, t_c] \ \text{loc}(r_2) = d_1\)

\(C: \ \text{adj}(d_1, w_1), \ \text{adj}(d_2, w_2),\)
  \(\text{conn}(w_1, w_2), \ \text{conn}(w_2, w_1), \ 0 < t_b\)
Example

\(\phi_0:\)

\[S:\ [0] \text{loc}(r1) = d1, [0] \text{loc}(r2) = d2, [0] \text{occ}(d1) = r1, [0] \text{occ}(d2) = r2,\]
\[T:\ [t_b,t_c] \text{loc}(r1) = d2, [t_b,t_c] \text{loc}(r2) = d1\]
\[C:\ \text{adj}(d1,w1), \text{adj}(d2,w2), \text{conn}(w1,w2), \text{conn}(w2,w1), 0 \leq t_b\]

now = 0

Open goals that can be resolved:
\([t_b,t_c] \text{loc}(r1) = d2, [t_b,t_c] \text{loc}(r2) = d1\)
- Resolve both

\([0,t_b] \text{m-move}(r1,d1,d2,w1,w2)\]
\[T:\ [0] \text{loc}(r1) = d1\]
\[\quad \quad [t_1] \text{loc}(r1) = w1\]
\[\quad \quad [t_2] \text{loc}(r1) = w2\]
\[\quad \quad [t_b] \text{loc}(r1) = d2\]
\[C:\ \text{adj}(d1,w1), \text{conn}(w1,w2), \text{adj}(d2,w2), 0 < t_1 < t_2 < t_b\]

\([0,t_b] \text{m-move}(r2,d2,d1,w2,w1)\]
\[T:\ [0] \text{loc}(r2) = d2\]
\[\quad \quad [t'_1] \text{loc}(r2) = w1\]
\[\quad \quad [t'_2] \text{loc}(r2) = w2\]
\[\quad \quad [t_b] \text{loc}(r2) = d1\]
\[C:\ \text{adj}(d2,w2), \text{conn}(w2,w1), \text{adj}(d1,w1), 0 < t'_1 < t'_2 < t_b\]

Next = \{t_1,t'_1\}
Next* ← ∅
now doesn’t change
Example

\[ \text{now} = 0 \]

Open goals that can be resolved:

- \([t_1] \text{loc}(r_1) = w_1, \]
- \([t'_1] \text{loc}(r_2) = w_2 \]

- Resolve both

\[ \phi_1: \]

\[ S: \begin{cases} [0] \text{loc}(r_1) = d_1, [0] \text{loc}(r_2) = d_2, \\ [0] \text{occ}(d_1) = r_1, [0] \text{occ}(d_2) = r_2, \\ [t_b t_c] \text{loc}(r_1) = d_2, [t_b t_c] \text{loc}(r_2) = d_1 \end{cases} \]

\[ T: \begin{cases} [t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [t_2] \text{loc}(r_1) = w_2, [t'_2] \text{loc}(r_2) = w_1, \\ [t_b] \text{loc}(r_1) = d_2, [t_b] \text{loc}(r_2) = d_1 \end{cases} \]

C: adj(d_1, w_1), adj(d_2, w_2), conn(w_1, w_2), conn(w_2, w_1), 0 < t_1 < t_2 < t_b, 0 < t'_1 < t'_2 < t_b

\[ \phi_2: \]

\[ S: \begin{cases} [0] \text{loc}(r_1) = d_1, [0] \text{loc}(r_2) = d_2, \\ [0] \text{occ}(d_1) = r_1, [0] \text{occ}(d_2) = r_2, \\ [t_b t_c] \text{loc}(r_1) = d_2, [t_b t_c] \text{loc}(r_2) = d_1 \end{cases} \]

\[ T: \begin{cases} [t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \\ [0, t_1] \text{loc}(r_1) = w_1, [t'_1] \text{loc}(r_2) = w_2, \\ [0, t_1] \text{loc}(r_1):(d_1, w_1), [0, t'_1] \text{loc}(r_2):(d_2, w_2), \\ [0, t_1] \text{occ}(d_1):(r_1, d_1), [0, t'_1] \text{occ}(d_2):(r_2, d_2), \end{cases} \]

C: adj(d_1, w_1), adj(d_2, w_2), conn(w_1, w_2), conn(w_2, w_1), 0 < t_1 < t_2 < t_b, 0 < t'_1 < t'_2 < t_b

Next = \{t_1, t'_1\}

Next* ← \{t_1\}

now ← t_1
Example

\( \text{now} = t_1 \)

Open goal that can be resolved:

- \([t_2] \text{loc}(r_1) = w_2\)

  • Resolve it

\( \phi_3: \)

\( S: \)

\[ [0, t_b, t_c] \text{loc}(r_1) = d_1, [0, t_b, t_c] \text{loc}(r_2) = d_2, \]

\[ [0, t_b, t_c] \text{occ}(d_1) = r_1, [0, t_b, t_c] \text{occ}(d_2) = r_2, \]

\[ [t_b, t_c] \text{loc}(r_1) = d_2, [t_b, t_c] \text{loc}(r_2) = d_1 \]

\( T: \)

\[ [t_2] \text{loc}(r_1) = w_2, [t'_2] \text{loc}(r_2) = w_1, \]

\[ [t_b] \text{loc}(r_1) = d_2, [t_b] \text{loc}(r_2) = d_1 \]

\( C: \)

adj(d_1,w_1), adj(d_2,w_2),

conn(w_1,w_2), conn(w_2,w_1),

\(2 \leq t_1 < t_2 < t_b, 2 \leq t'_1 < t'_2 < t_b\)

Next = \(\{t'_1, t_2\}\)

Next* ← \(\{t'_1\}\)

now ← \(t'_1\)
Example

now = t'₁

Open goal that can be resolved:

- [t'₂] loc(r2) = w₁
  - Resolve it

[τ₁,τ₂] navigate(r2,w₂,w₁):
T: [τ₁,τ₂] loc(r2):(w₂,w₁),
C: conn(w₂,w₁), τ₁ < τ₂

Next = {t₂,t'₂}
Next* ← {t₂,t'₂}
now ← t₂ = t'₂

ϕ₃:
S: [tₐ,tₐ] loc(r₁) = d₂, [tₐ,tₐ] loc(r₂) = d₁
[0,t₁] loc(r₁):(d₁,w₁), [0,t'₁] loc(r₂):(d₂,w₂),
[0,t₁] occ(d₁):(r₁,∅), [0,t'₁] occ(d₂):(r₂,∅),
[t₁,t₂] loc(r₁):(w₁,w₂),
[t₂] loc(r₁) = w₂

T: [τ₂] loc(r₂) = w₁,
[tᵣ] loc(r₁) = d₂, [tᵣ] loc(r₂) = d₁
C: adj(d₁,w₁), adj(d₂,w₂),
    conn(w₁,w₂), conn(w₂,w₁),
    2 ≤ t₁ < t₂ < τᵣ, 2 ≤ t'₁ < t'₂ < τᵣ

ϕ₄:
S: [tₐ,tₐ] loc(r₁) = d₂, [tₐ,tₐ] loc(r₂) = d₁
[0,t₁] loc(r₁):(d₁,w₁), [0,t'₁] loc(r₂):(d₂,w₂),
[0,t₁] occ(d₁):(r₁,∅), [0,t'₁] occ(d₂):(r₂,∅),
[t₁,t₂] loc(r₁):(w₁,w₂), [t'₁,t'₂] loc(r₂):(w₂,w₁),
[t'₂] loc(r₂) = w₁

T: [tᵣ] loc(r₁) = d₂, [tᵣ] loc(r₂) = d₁
C: adj(d₁,w₁), adj(d₂,w₂),
    conn(w₁,w₂), conn(w₂,w₁),
    2 ≤ t₁ < t₂ < τᵣ, 2 ≤ t'₁ < t'₂ < τᵣ
Example

Now \( t_2 = t'_2 \)

Open goals that can be resolved:
- \([t_b,t_c]\) \text{loc}(r_1) = d_2
- \([t_b,t_c]\) \text{loc}(r_2) = d_1

Resolve both

\[ \phi_4: \]
- \( S: \ [t_b,t_c]\ \text{loc}(r_1) = d_2, [t_b,t_c]\ \text{loc}(r_2) = d_1 \)
- \( T: \ [t_b]\ \text{loc}(r_1) = d_2, [t_b]\ \text{loc}(r_2) = d_1 \)
- \( C: \ \text{adj}(d_1,w_1), \text{adj}(d_2,w_2), \text{conn}(w_1,w_2), \text{conn}(w_2,w_1), 2 \leq t_1 < t_2 < t_b, 2 \leq t'_1 < t'_2 < t_b \)

\[ \phi_5: \]
- \( S: \ [t_b,t_c]\ \text{loc}(r_1) = d_2, [t_b,t_c]\ \text{loc}(r_2) = d_1 \)
- \( T: \ [t_b,t_c]\ \text{loc}(r_1) = d_2, [t_b,t_c]\ \text{loc}(r_2) = d_1 \)
- \( C: \ \text{adj}(d_1,w_1), \text{adj}(d_2,w_2), \text{conn}(w_1,w_2), \text{conn}(w_2,w_1), 2 \leq t_1 < t_2 < t_b, 2 \leq t'_1 < t'_2 < t_b, t_2 = t'_2 \)

Next = \{t_b\}
Next* ← \{t_b\}
Now ← \( t_b \)
Example

$N_{\text{IntEx/GR}}, \text{Oct 2020}$

\[ \phi_5: \]

\[ S: \quad [t_b, t_c] \ \text{loc}(r_1) = d_2, \ [t_b, t_c] \ \text{loc}(r_2) = d_1 \]
\[ [0, t_1] \ \text{loc}(r_1):(d_1, w_1), \ [0, t_1'] \ \text{loc}(r_2):(d_2, w_2), \]
\[ [0, t_1] \ \text{occ}(d_1):(r_1, \emptyset), \ [0, t_1'] \ \text{occ}(d_2):(r_2, \emptyset), \]
\[ [t_1, t_2] \ \text{loc}(r_1):(w_1, w_2), \ [t_1', t_2'] \ \text{loc}(r_2):(w_2, w_1), \]
\[ [t_2, t_b] \ \text{loc}(r_1):(w_2, d_2), \ [t_2, t_b] \ \text{loc}(r_2):(w_1, d_1), \]
\[ [t_2, t_b] \ \text{occ}(d_2): (\emptyset, r_1), \ [t_2, t_b] \ \text{occ}(d_1): (\emptyset, r_2), \]
\[ [t_a] \ \text{loc}(r_1) = d_2, \ [t_a] \ \text{loc}(r_2) = d_1 \]

$T$:

$C$:\ adj(d_1, w_1), \ adj(d_2, w_2), \$
\conn(w_1, w_2), \ conn(w_2, w_1),$
\[ 2 \leq t_1 < t_2 < t_b,\ 2 \leq t_1' < t_2' < t_b,\ t_2 = t_2' \]

\[ \phi_6: \]

\[ S: \quad [t_b, t_c] \ \text{loc}(r_1) = d_2, \ [t_b, t_c] \ \text{loc}(r_2) = d_1 \]
\[ [0, t_1] \ \text{loc}(r_1):(d_1, w_1), \ [0, t_1'] \ \text{loc}(r_2):(d_2, w_2), \]
\[ [0, t_1] \ \text{occ}(d_1):(r_1, \emptyset), \ [0, t_1'] \ \text{occ}(d_2):(r_2, \emptyset), \]
\[ [t_1, t_2] \ \text{loc}(r_1):(w_1, w_2), \ [t_1', t_2'] \ \text{loc}(r_2):(w_2, w_1), \]
\[ [t_2, t_b] \ \text{loc}(r_1):(w_2, d_2), \ [t_2, t_b] \ \text{loc}(r_2):(w_1, d_1), \]
\[ [t_2, t_b] \ \text{occ}(d_2): (\emptyset, r_1), \ [t_2, t_b] \ \text{occ}(d_1): (\emptyset, r_2), \]

$T$:

$C$:\ adj(d_1, w_1), \ adj(d_2, w_2), \$
\conn(w_1, w_2), \ conn(w_2, w_1),$
\[ 2 \leq t_1 < t_2 < t_b,\ 2 \leq t_1' < t_2' < t_b,\ t_2 = t_2' \]
\( \phi_0: \)

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

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

\[ C: \text{adj}(d1,w1), \text{adj}(d2,w2), \]

\[ \text{conn}(w1,w2), \text{conn}(w2,w1), \]

\[ 0 < t_b \]

---

**Example**

**Temporal plan:**

\{[0,t_1] \text{leave}(r1,d1,w1), \]

\[ [0,t_1'] \text{leave}(r2,d2,w2), \]

\[ [t_1,t_2] \text{navigate}(r1,w1,w2), \]

\[ [t_1',t_2'] \text{navigate}(r2,w2,w1), \]

\[ [t_2,t_b] \text{enter}(r1,d2,w2), \]

\[ [t_1',t_b] \text{enter}(r2,d1,w1) \} \]

---

**\( \phi_6: \)**

\[ S: [0,t_1] \text{loc}(r1):(d1,w1), [0,t_1'] \text{loc}(r2):(d2,w2), \]

\[ [0,t_1] \text{occ}(d1):(r1,\emptyset), [0,t_1'] \text{occ}(d2):(r2,\emptyset), \]

\[ [t_1,t_2] \text{loc}(r1):(w1,w2), [t_1',t_2'] \text{loc}(r2):(w2,w1), \]

\[ [t_2,t_b] \text{loc}(r1):(w2,d2), [t_2,t_b] \text{loc}(r2):(w1,d1), \]

\[ [t_2,t_b] \text{occ}(d2): (\emptyset,r1), [t_2,t_b] \text{occ}(d1): (\emptyset,r2), \]

\[ T: \]

\[ C: \text{adj}(d1,w1), \text{adj}(d2,w2), \]

\[ \text{conn}(w1,w2), \text{conn}(w2,w1), \]

\[ 2 \leq t_1 < t_2 < t_b, 2 \leq t_1' < t_2' < t_b, t_2 = t_2' \]
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?