Neurosymbolic Planner Synthesis Incorporating Specification Closeness

Pranav

August 12, 2026

Table of Contents

STL and Robustness I

Definition (Signal)

A signal

$$s_{t_i}^{t_T} = (x_i, t_i),\ (x_{i+1}, t_{i+1}), \ldots, (x_T, t_T)$$

is an ordered ($t_{j-1} < t_j$) finite sequence of states $x_j \in \mathbb{R}^n$ and their associated times $t_j \in \mathbb{R}$ starting from time t_i and ending at time t_T.

Definition (Subsignal)

Given a signal $s_{t_0}^{t_T}$, a subsignal of $s_{t_0}^{t_T}$ is a signal $s_{t_i}^{t_T}$ where $i \geq 0$ and $t_i \geq t_0$.

- *qualitative* (or Boolean) semantics, giving the classical notion of satisfaction of a property over a trajectory, i.e. $s(\varphi, \xi, t) = 1$ if the trajectory ξ at time t satisfies the STL formula φ

STL and Robustness II

- *quantitative* semantics, denoted by $\rho(\varphi, \xi, t)$. The latter, also called *robustness*, is a measure of how robust is the satisfaction of φ w.r.t. perturbations of the signals. Robustness is recursively defined as:
  1. $\rho(s_t, \mu_c) = c - \mu(x_t)$ where μ_c denotes the predicate $\mu(x) < c$
  2. $\rho(s_t, \neg\phi) = -\rho(s_t, \phi)$
  3. $\rho(s_t, \phi \wedge \psi) = min(\rho(s_t, \phi), \rho(s_t, \psi))$
  4. $\rho(s_t, \Diamond_{[a,b]}\phi) = \max_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi)$
  5. $\rho(s_t, \Box_{[a,b]}\phi) = \min_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi)$
  6. $\rho(s_t, \phi \mathcal{U}_{[a,b]}\psi) =$
     $\max_{t' \in [t+a, t+b]}(min(\rho(s'_t, \psi), \min_{t'' \in [0, t']} \rho(s''_t, \phi))$
- Robustness is compatible with satisfaction via the following *soundness* property: if $\rho(\varphi, \xi, t) > 0$ then $s(\varphi, \xi, t) = 1$ and if $\rho(\varphi, \xi, t) < 0$ then $s(\varphi, \xi, t) = 0$. When $\rho(\varphi, \xi, t) = 0$ arbitrary small perturbations of the signal might lead to changes in satisfaction value.

Problem Framing and Approaches I

- **Input**:
  - The **task specification** φ is given as input. This specification describes the desired behaviour of the robot
  - Demonstrations or trajectories are also given, as part of the nuPlan dataset
- **Objective**:
  - Synthesize a **planning policy** that:
    - Produces trajectories close to those generated by φ
    - Satisfiable by the underlying controller

An example of a task specification φ can be understood by considering Figure 1.

Problem Framing and Approaches II

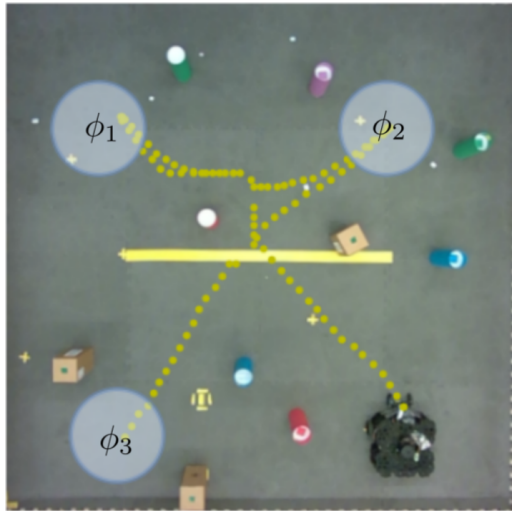


Figure: Task Specification Map: Three Blue Regions for Sequential Waypoint Visitation

As shown in the figure, the specification $\varphi_1 = \bigwedge_{i=1}^{N} \mathbf{F}_{[t_i, t_{i+1}]} \phi_i$ where ϕ_i denotes reaching each blue region is an example of a task specification given by the user.

Formal Example

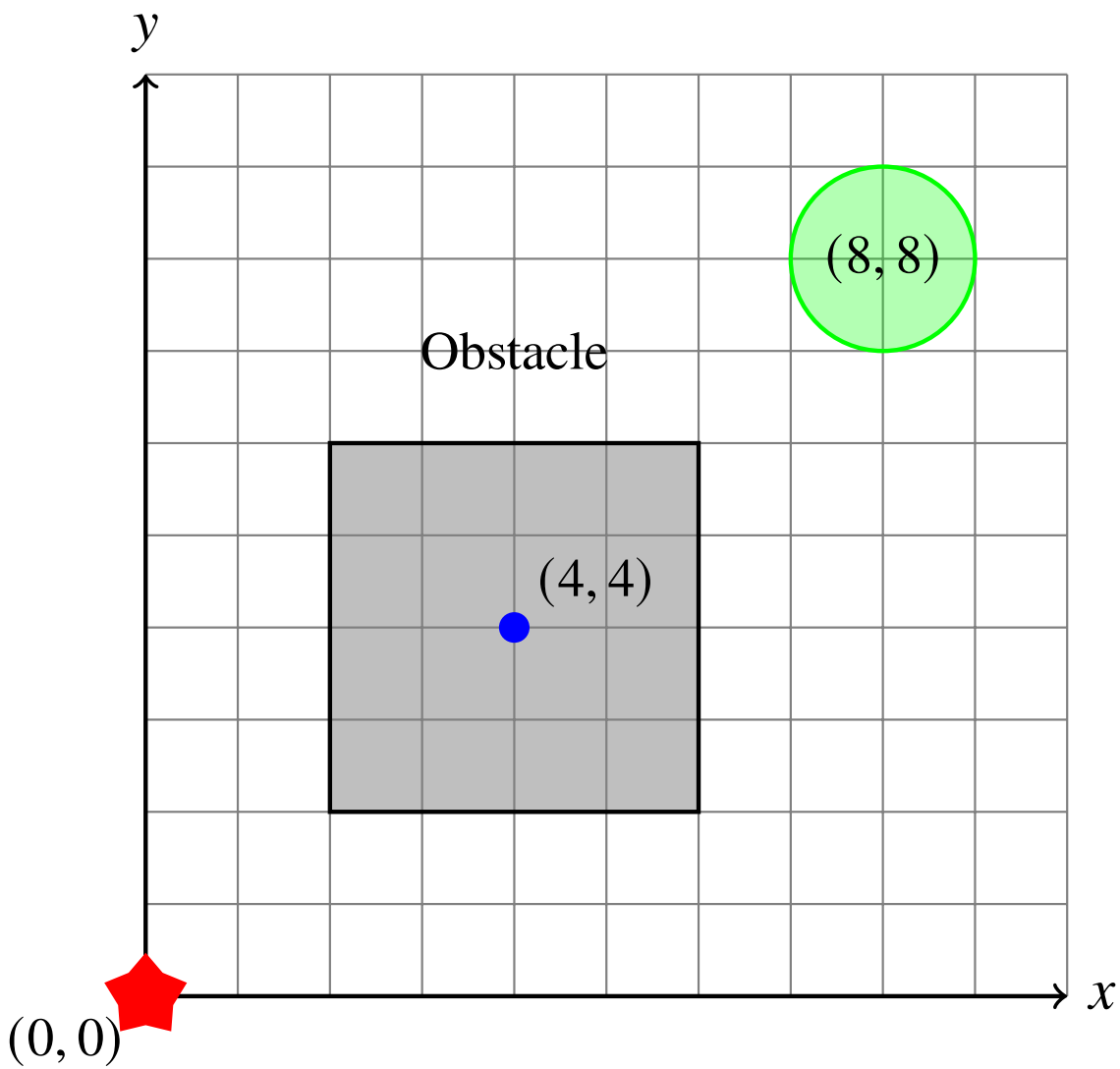
- **Starting Coordinates** : $(0, 0)$. **Goal Coordinates** : $(x_G, y_G) = (8, 8)$.
- **Predicates** : $AtGoal^{\theta}(x, y, v_x, v_y)$, $MaxSpeed^{V}(x, y, v_x, v_y)$, $AvoidObstacles(x, y.v_x, v_y)$.
- $AtGoal^{\theta}(x, y, v_x, v_y) =$

$$\begin{cases} \top & ||(x - x_G)^2 + (y - y_G)^2||_2 \leq \theta \\ \bot & otherwise \end{cases}$$

- $MaxSpeed^{V}(x, y, v_x, v_y) =$

$$\begin{cases} \top & (v_x \leq V) \cap (v_y \leq V) \\ \bot & otherwise \end{cases}$$

Formal Example



Formal Example

- $AvoidObstacles(x, y, v_x, v_y) =$

$$\begin{cases} \top & SDF(x, y, M) > 0 \\ \bot & otherwise \end{cases}$$

where $SDF(x, y, M) = min_{p \in M} ||\vec{r} - p||_2 \cdot sign(\cdot)$. Here, p denotes the extent of the square obstacle, and M is the obstacle mask which comprises of the gray shaded region. The $sign(\cdot)$ is positive if the point is outside the obstacle, and negative if it is inside.

Formal Example

- **Task Specification:**
  $\mathbf{F}_{[0,10]} AtGoal(\cdot) \wedge \mathbf{G}_{[0,T]} AvoidObstacles(\cdot)$.
- **Controller Specification:** $\mathbf{G}_{[0,T]} MaxSpeed^{V}(\cdot)$
- If we use $V = 0.5$, then the controller ensures that the ego vehicle can move only 5 units in 10 timesteps, which is impossibility.
- A more realistic example could be using a new predicate $NearObstacle^{\epsilon}(x, y, v_x, v_y) =$

$$\begin{cases} \top & SDF(x, y, M) < \epsilon \\ \bot & otherwise \end{cases}$$

- This characterizes the ego vehicle being close enough to the obstacle.
- Consider the controller specification $\mathbf{G}_{[0,T]}(NearObstacle^{\epsilon}(\cdot) \rightarrow MaxSpeed^{V}(\cdot))$. This specifies that the speed of the ego should be capped when it is near the obstacle, which makes sense in real-life. ϵ and V can be adjusted to get an infeasible specification.

Demarcation between specifications

- Some parts of the specification can be violated, but some cannot be
- Consider the overall specification of the example given above. It can be split into three parts each of which are in conjunction
  1. $\phi_1 = \mathbf{F}_{[0,10]} AtGoal(\cdot)$
  2. $\phi_2 = \mathbf{G}_{[0,T]} AvoidObstacles(\cdot)$
  3. $\phi_3 = \mathbf{G}_{[0,T]} (NearObstacle^{\epsilon}(\cdot) \rightarrow MaxSpeed^{V}(\cdot))$
- From these, ϕ_1 can be relaxed, but not ϕ_2 or ϕ_3, since the latter two are physical constraints of the environment or the controller (as learned by FLoRA).
- Any system we build needs to use this demarcation.
- The overall specification can be split into **User Specification** (relaxable) and **Environment Specification** (non-relaxable).

Alternate Specification

- The user may also specify parts of the task which are inviolable, which can be clubbed with the non-relaxable component of the specification.
- Ideally, we would need to relax the time constraint of ϕ_1
- $\phi_1 = \mathbf{F}_{[0,100]} AtGoal(\cdot)$ will be a suitable alternate specification (since the goal is reachable given adequate time).
- But just adjusting time may not be sufficient in all situations

Formal Example - II

- Consider the NuPlan dataset, and the Nevada map in particular.

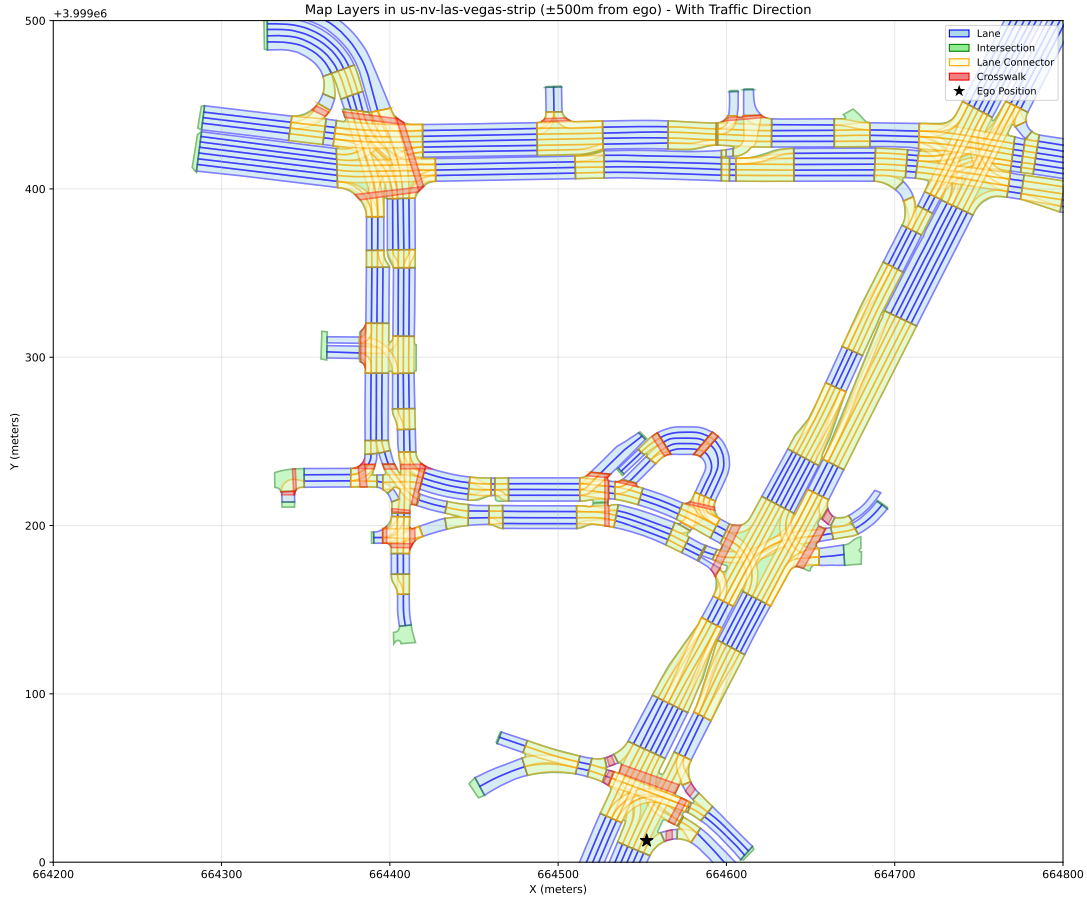


Figure: A section of the Nevada Map

Formal Example - II

- We use the predicate $AtObject(f_o, x, y, \vec{v})$ to denote the fact that the vehicle is at a particular object which is defined by a polygon. We use $f_o(x, y)$ to check if a point is inside the object or not. $f_o(x, y) > 0$ means that the point is inside the polygon.
- There are multiple kinds of objects -
  1. Lanes - These are normal straight lanes
  2. Lane Connectors - These connect lanes across an intersection, or help in turning
  3. Intersections - Here, two roads meet and all possible vehicle movements and turns can occur
- $AtObject(f_o, x, y, \vec{v}) =$

$$\begin{cases} \top & f_o(x, y) > 0 \\ \bot & otherwise \end{cases}$$

Formal Example - II

- We define the following sequential specification using the $AtObject(\cdot)$ predicate.
  $\mathbf{F}_{[0,t_1]} AtObject(f_2, x, y, \vec{v}) \wedge \mathbf{F}_{[t_1,t_2]} AtObject(f_1, x, y, \vec{v})$

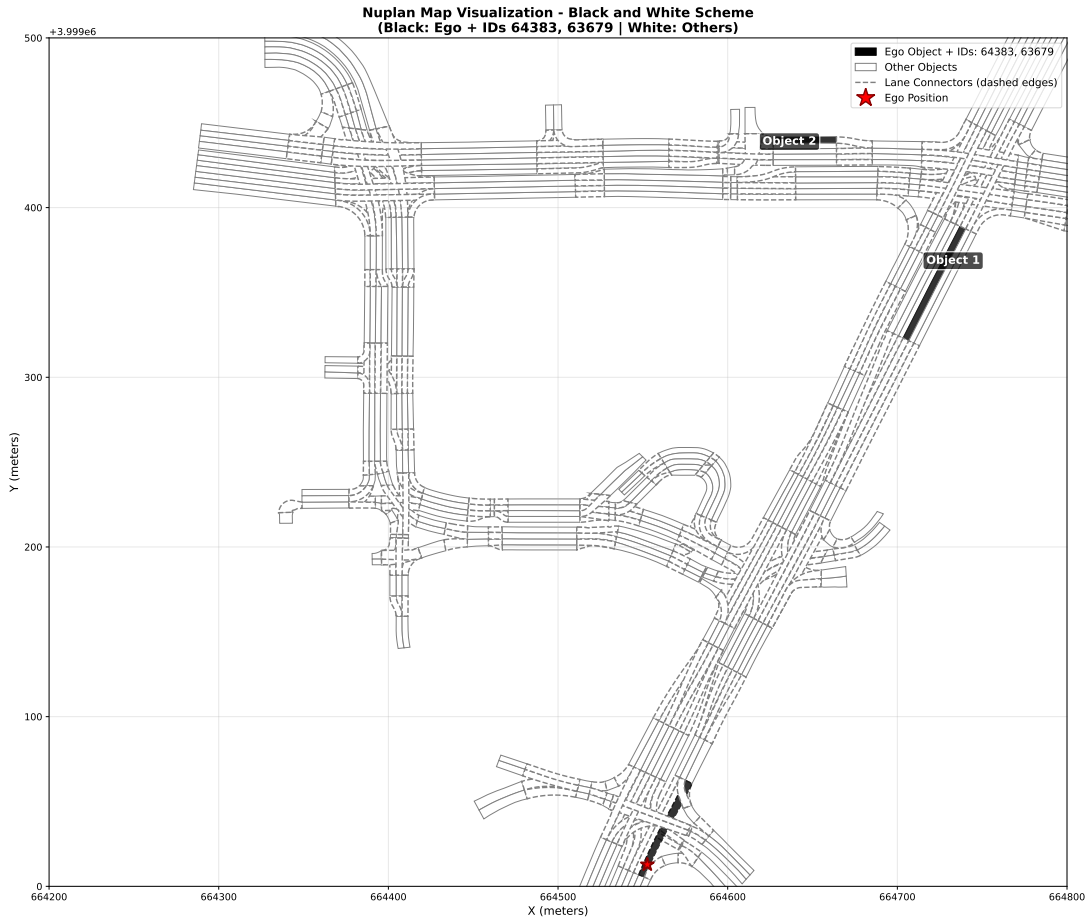


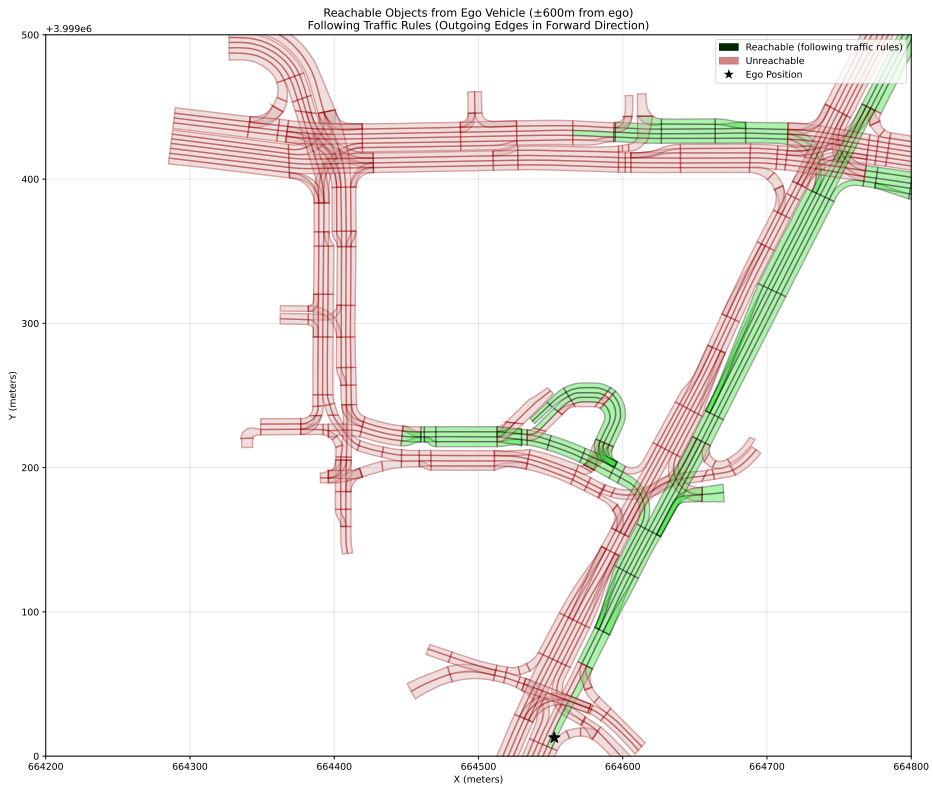
Figure: Task Specification Map

Formal Example - II

- We look at the task specification again - $\mathbf{F}_{[0,t_1]} AtObject(f_2, x, y, \vec{v}) \wedge \mathbf{F}_{[t_1,t_2]} AtObject(f_1, x, y, \vec{v})$.
- From the diagram it is clear that Object 1 is on the way between initial position of ego vehicle and Object 2.
- But sequential task spec gives the reverse order - first object 2 then object 1.
- Sometimes may not even be feasible to reach object 1 after object 2 (U-turns may not be available, etc.)

Formal Example - II

- We can do a simple BFS-based reachability analysis bounded by a time cost to understand the extent to which the ego vehicle can travel in that time.
- With respect to the above spec, consider $t_1 = 200$ and $t_2 = 300$. We look at the set of lane objects reachable in 200 steps.

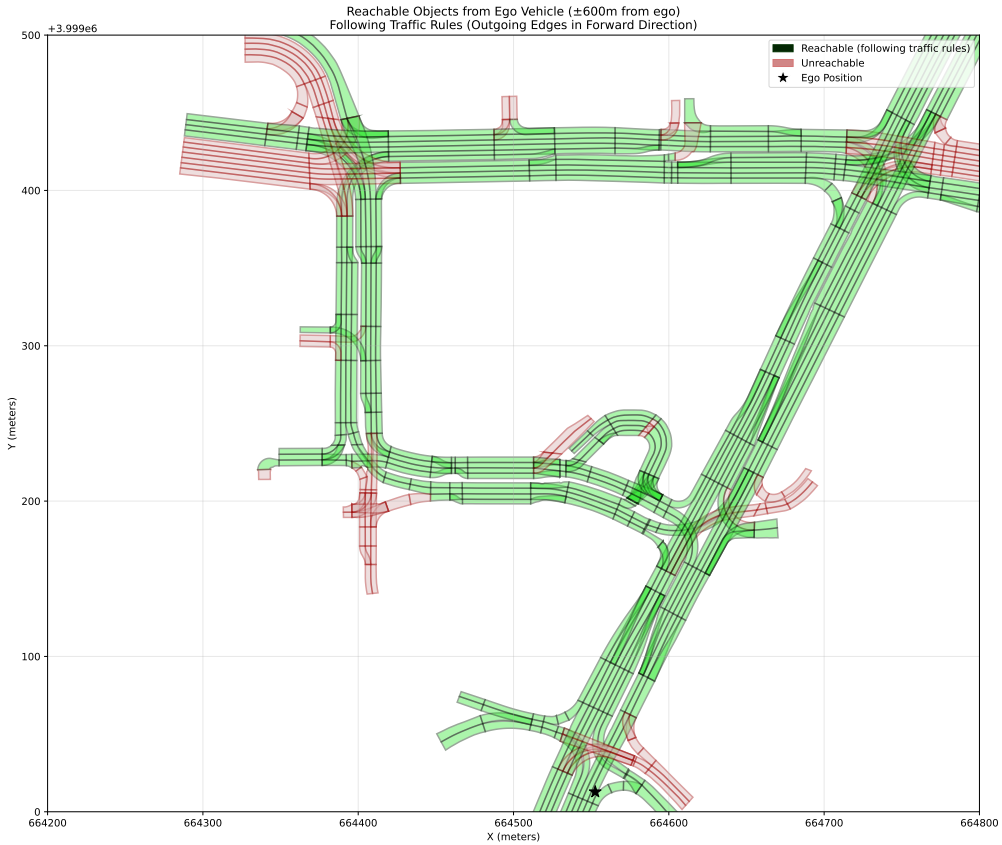


Formal Example - II

- Clearly, if we want to reach object 1 first and then object , then the specification is satisfiable.
- This leads us to a possible relaxation - converting **Sequential Tasks to Cover Tasks**.
- The corresponding cover task would look like $\mathbf{F}_{[0,T]} AtObject(f_2, x, y, \vec{v}) \wedge \mathbf{F}_{[0,T]} AtObject(f_1, x, y, \vec{v})$ where $T = 300$ in this case.
- An alternative would be to relax the time intervals. We can make $t_2 = 2000$ instead of 200

Formal Example - II

- Then, the ego vehicle can possibly go around the entire circular path after reaching object 2, and come back to object 1. This is possible in 2000 time steps (consider the below reachability diagram)



Notes on the FLoRA Architecture : Condition-Action Pair

The FLoRA paper's contribution is learning interpretable scoring rules from driving demonstrations, represented in temporal logic. The learnable STL formula is represented as
$\bigodot_{i=1}^{m}(\bigwedge_{j=1}^{n} condition_j \rightarrow action_i)$
Three kinds of layers -

- Temporal -

```
class TemporalLayer(torch.nn.Module):
```

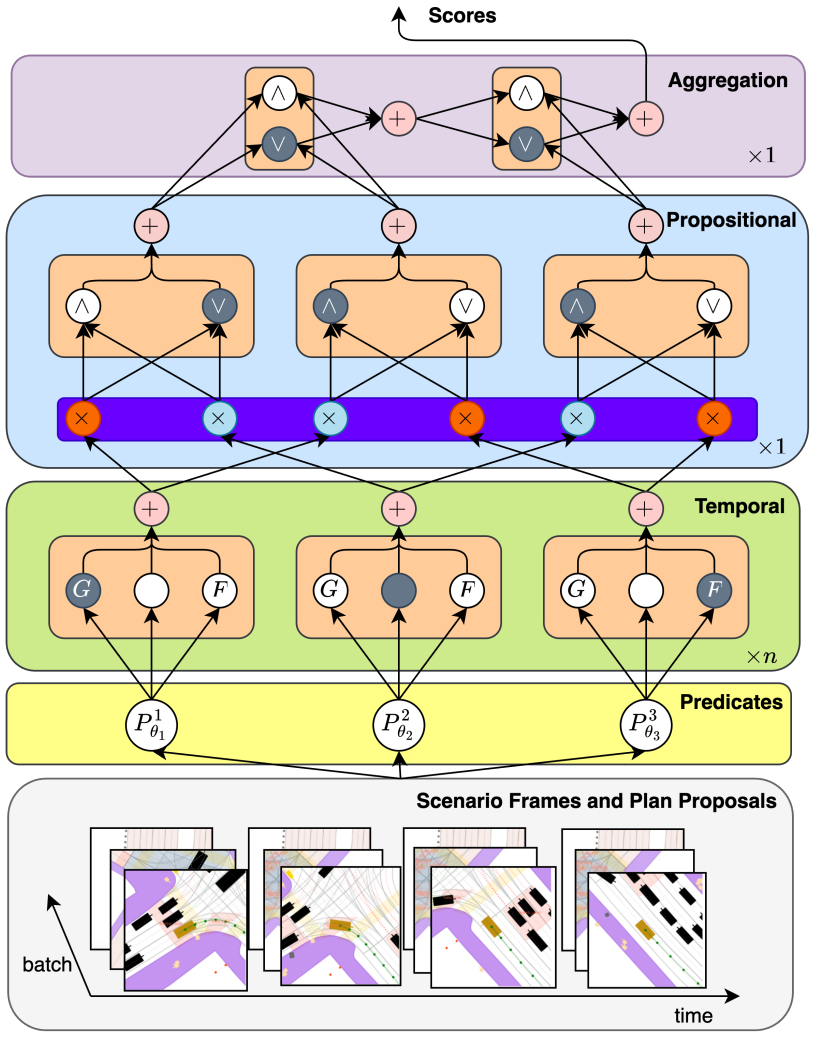
- Propositional, and
- Aggregation - together represented in

```
class LogicLayer(torch.nn.Module):
```

Layers

- Given a sequence of frames $F = \{f_1, f_2, \cdots f_n\}$, and N predicates $P_1, P_2, \cdots P_N$, we compute $x_t^i = P_i(f_t)$ at each time instance.
- Collect the outputs over time, $X^i = [x_1^i, x_2^i, \cdots x_n^i]$, and subsequently $\mathcal{X} = [X^1, X^2, \cdots X^N]$
- Pass these through the **temporal layer** which stack temporal operators (such as $\mathbf{F}(\mathbf{G}(P_i))$) on predicates
- Then, pass through the **propositional layer**, which takes either the temporal formula or its negation, and combines two formulas together ($\binom{N}{2}$ possible combinations) with either the $\wedge$ or $\vee$ operator.
- The **aggregation layer** combines the outputs of the propositional layer with either the $\wedge$ or $\vee$ operators.

Diagram Depicting Layers

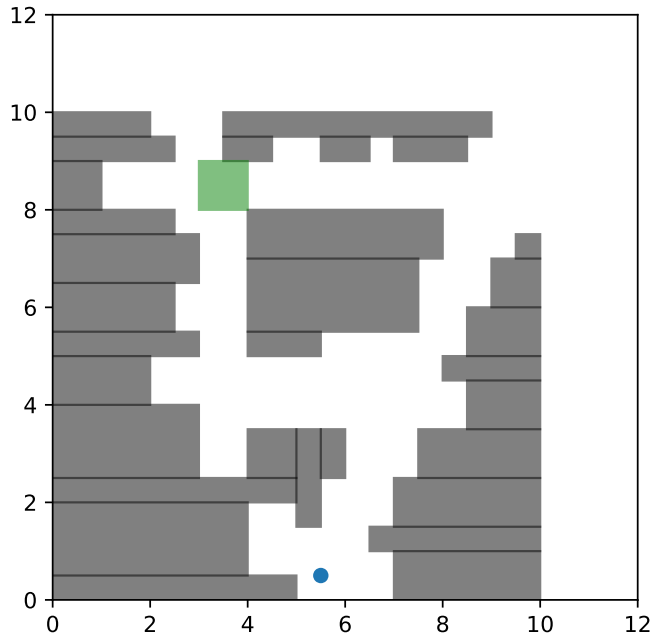


Composition of two solutions in STLPy

STLPy offers implementations of Mixed Integer Linear Programming (MILP)-based solvers to solve the *trajectory synthesis problem*, which is an NP-hard problem.

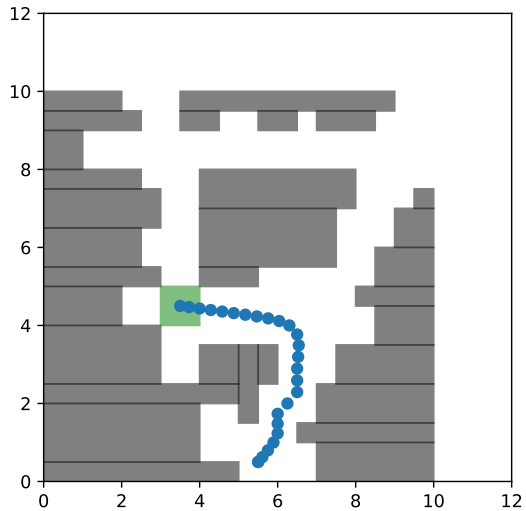
Consider the following problem - $\mathbf{F}_{[0,50]}\mathsf{InsideRectangle(3,4,8,9)} \wedge \mathbf{G}_{[0,50]}(\bigwedge_{i=1}^{31} \mathit{OutsideRectangle}(x_1^i, x_2^i, y_1^i, y_2^i)) \wedge x_0 = (5.5, 0.5, 0, 0)$.

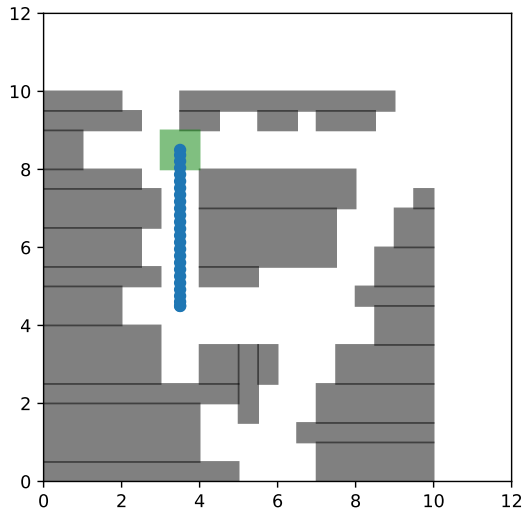
This is not solved by Gurobi even after 10 minutes.



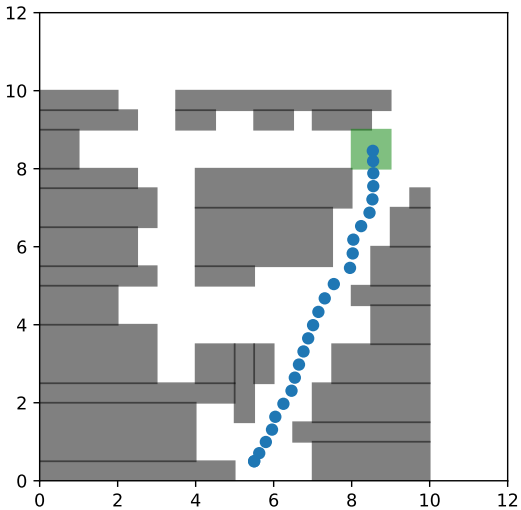
Composition of two solutions

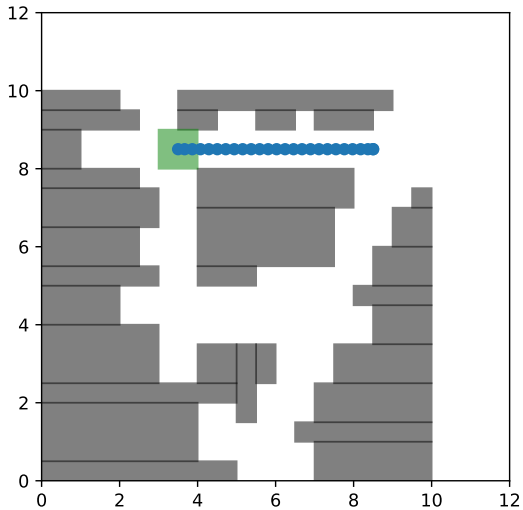
- However, if we use **landmarks** and split the problem into two - $\mathbf{F}_{[0,25]}$InsideRectangle(3,4,4,5) $\wedge$ $\mathbf{G}_{[0,25]}(\bigwedge_{i=1}^{31} OutsideRectangle(x_1^i, x_2^i, y_1^i, y_2^i)) \wedge x_0 =$ $(5.5, 0.5, 0, 0)$ followed by $\mathbf{F}_{[0,25]}$InsideRectangle(3,4,8,9) $\wedge$ $\mathbf{G}_{[0,25]}(\bigwedge_{i=1}^{31} OutsideRectangle(x_1^i, x_2^i, y_1^i, y_2^i)) \wedge x_0 =$ $(3.5, 4.5, 0, 0)$ Gurobi solves it in around 30 seconds (combined)





Composition of two solutions





- Each of the landmark-based solutions are achieved in 30 seconds
- Useful if we want to change the graph dynamically

Composition of two solutions

- How do we come up with suitable landmarks so that the planning problem is simplified?
- Landmarks also help us dynamically change the graph, and thus the STL planning environment after each iteration of planning
- Not tractable beyond small graphs.

LTL on Graphs I

An introduction to Linear Temporal Logic (LTL)

Definition

Definition: An LTL formula ϕ can be written in the form

$$\phi ::= \top | \bot | p | \neg\phi | \phi \wedge \psi | \mathbf{G}\phi | \mathbf{F}\phi$$

Models are transition systems. They can be written as $\mathcal{M} = (S, \rightarrow, L)$. Here, S is a set of states, $\rightarrow \subseteq S \times S$ denotes a transition from s_1 to s_2, and $L : S \rightarrow \mathcal{P}(Atoms)$ denotes the set of predicates which hold at each state.

LTL on Graphs II

An introduction to Linear Temporal Logic (LTL)

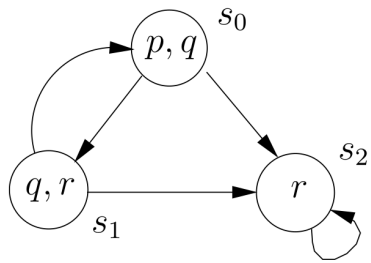


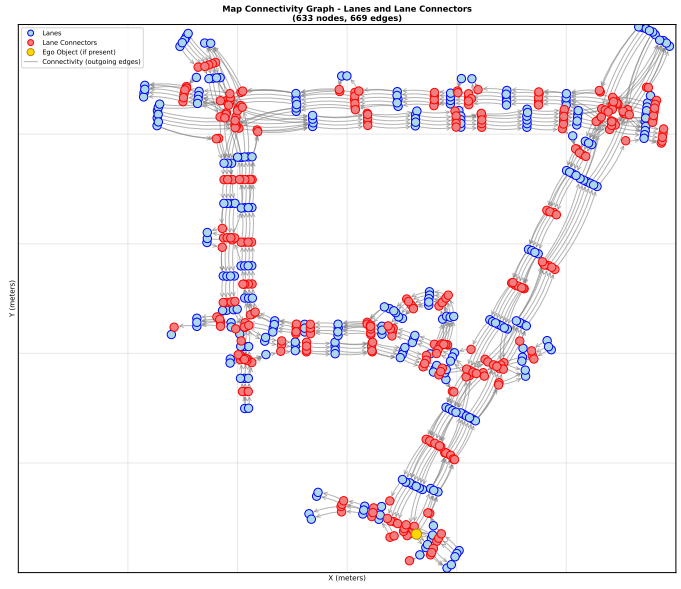
Figure 3.3. A concise representation of a transition system $\mathcal{M} = (S, \rightarrow, L)$ as a directed graph. We label state s with l iff $l \in L(s)$.

Figure: A sample model $\mathcal{M}$ represented as a transition system

- Some textbooks view a path π within this transition system as the primary unit to check satisfiability.
- A path in a model $\mathcal{M}$ is an infinite set of states $s_1 s_2 s_3 \cdots$ such that for each i, $(s_i, s_{i+1}) \in \rightarrow$.
- This seems similar to the directed graph of lane objects we see in NuPlan!

LTL on Graphs III

An introduction to Linear Temporal Logic (LTL)



- Each node can have predicates such as $AtObject_i$, $IsLeftLane()$, $MaxSpeedUnder10()$ to be true and the others to be false.

LTL on Graphs IV

An introduction to Linear Temporal Logic (LTL)

Definition 3.6 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model and $\pi = s_1 \rightarrow \ldots$ be a path in $\mathcal{M}$. Whether π satisfies an LTL formula is defined by the satisfaction relation $\models$ as follows:

1. $\pi \models \top$
2. $\pi \not\models \bot$
3. $\pi \models p$ iff $p \in L(s_1)$
4. $\pi \models \neg\phi$ iff $\pi \not\models \phi$
5. $\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
6. $\pi \models \phi_1 \vee \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$
7. $\pi \models \phi_1 \rightarrow \phi_2$ iff $\pi \models \phi_2$ whenever $\pi \models \phi_1$
8. $\pi \models \mathrm{X}\,\phi$ iff $\pi^2 \models \phi$
9. $\pi \models \mathrm{G}\,\phi$ iff, for all $i \geq 1$, $\pi^i \models \phi$

Figure: Semantics of LTL Formulae

LTL on Graphs V

- Now, for the model $\mathcal{M} = (S, \rightarrow, L)$ to satisfy a formula, i.e. for $\mathcal{M} \vDash \phi$, we need every infinite trace $\pi = s_1 s_2 s_3 \cdots$ in $\mathcal{M}$ to satisfy ϕ as per the aforementioned semantics.
- In order to perform this model checking efficiently, we use the help of Buchi automata, which are similar to Deterministic Finite Automata (DFA), but support infinite traces (i.e. subsets of 2^ω) as input.
- A *Buchi Automaton* is of the form

$$M = (Q, \Sigma, q_0, \Delta, F)$$

  with a finite set of states Q, input alphabet Σ, initial state $q_0 \in Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$ and a set of final states F.
- A *run* of M on α is a sequence of states $\rho = \rho(0)\rho(1)\cdots$ with $\rho(0) = q_0$ and $(\rho(i), \rho(i+1)) \in \Delta$ for $i \geq 0$.
- *M accepts α* $\Leftrightarrow$ there exists a run ρ of M on α, and a number i such that $\rho(i)$ occurs infinitely often, and $\rho(i) \in F$.

LTL on Graphs VI

These traces constitute an ω-language. Four steps are needed to implement this idea. For an input model $\mathcal{M}$ and an initial state s, and a formula φ

1. Define the ω-language of all label sequences through $(\mathcal{M}, s)$ by a Buchi automaton $\mathcal{A}_{\mathcal{M},s}$
2. Define the ω-language of all label sequences, which do not satisfy φ by a Buchi automaton $\mathcal{A}_{\neg\varphi}$
3. Construct a Buchi automaton $\mathcal{B}$ which recognizes $L(\mathcal{A}_{\mathcal{M},s}) \cap L(\mathcal{A}_{\neg\varphi})$, i.e. accepts all label sequences through $(\mathcal{M}, s)$ which violate φ.
4. Check $\mathcal{B}$ for nonemptiness; if $L(\mathcal{B}) \neq \emptyset$ then answer "$(\mathcal{M}, s)$ does not satisfy φ", otherwise answer "$(\mathcal{M}, s)$ satisfies φ".

LTL on Graphs VII

The hardest step is to convert LTL formulae into Buchi automata, which was shown to be possible in exponential time in the size of the formula. The rest of the steps are all done in polynomial time. Thus, model checking in LTL is a polynomial time algorithm in the size of $\mathcal{M}$ and exponential in the size of φ.

LTL on Graphs VIII

Nonemptiness Checking

Theorem 1.18. (Nonemptiness Problem) *The nonemptiness Problem for Büchi automata (with state set Q and transition relation Δ) is solvable in time $O(|Q| + |\Delta|)$.*

Proof Let $\mathfrak{A} = (Q, \Sigma, q_0, \Delta, F)$ be a Büchi automaton. Define $E = \{(p, q) \in Q \times Q \mid \exists a \in \Sigma : (p, a, q) \in \Delta\}$ and call $G := (Q, E)$ the *transition graph* of $\mathfrak{A}$.

Therefore $L(\mathfrak{A}) \neq \emptyset$ iff in the transition graph there is a path from q_0 to a final state q, from which there is a path back to q.

This is the case iff in the transition graph of $\mathfrak{A}$ there is a strongly connected component (SCC) C such that C contains a final state and is reachable by a path from q_0.

Nonemptiness test

1. Apply depth-first search from q_0 in order to determine the set Q_0 of states reachable from q_0.
2. Apply Tarjan's SCC-algorithm to list all SCC's over Q_0, and check each SCC for the containment of a final state.
3. If such an SCC is encountered, answer $L(\mathfrak{A}) \neq \emptyset$, otherwise $L(\mathfrak{A}) = \emptyset$.

LTL on Graphs IX

Construction of Intersection Automata

Theorem 1.19. *The intersection $L_1 \cap L_2$ of two Büchi recognizable ω-languages L_1, L_2 is again Büchi recognizable.*

Proof Assume L_i is recognized by the Büchi automaton $\mathfrak{A}_i = (Q_i, \Sigma, q_{i0}, \Delta_i, F_i)$ for $i = 1, 2$.

First Idea: Form the product automaton

$$(Q_1 \times Q_2, \Sigma, (q_{10}, q_{20}), \Delta, F_1 \times F_2)$$

where $((p, q), a, (p', q')) \in \Delta$ iff $(p, a, p') \in \Delta_1$ and $(q, a, q') \in \Delta_2$.

Problem: We cannot assume that the final states in the two runs of $\mathfrak{A}_1, \mathfrak{A}_2$ are visited simultaneously

Solution: Repeatedly do the following steps

1. Wait for a final state $p \in F_1$ in the first component.
2. When a $p \in F_1$ is encountered, wait for a final state $q \in F_2$ in the second component.
3. When a $q \in F_2$ is encountered, signal "cycle completed" and go back to 1.

LTL on Graphs X

Construction of Intersection Automata

Hence work with the state space $Q_1 \times Q_2 \times \{1,2,3\}$.
Form the refined product automaton

$$\mathfrak{A} = (Q_1 \times Q_2 \times \{1,2,3\}, \Sigma, (q_{10}, q_{20}, 1), \Delta', Q_1 \times Q_2 \times 3)$$

with the following transitions in Δ', in each case assuming $(p,a,p') \in \Delta_1$ and $(q,a,q') \in \Delta_2$:

- $((p,q,1),a,(p',q',1))$ if $p' \notin F_1$
- $((p,q,1),a,(p',q',2))$ if $p' \in F_1$
- $((p,q,2),a,(p',q',2))$ if $q' \notin F_2$

- $((p,q,2),a,(p',q',3))$ if $q' \in F_2$
- $((p,q,3),a,(p',q',1))$

Then a run of $\mathfrak{A}$ simulates two runs ρ_1 of $\mathfrak{A}_1$ and ρ_2 of $\mathfrak{A}_2$: It has infinitely often 3 in the third component iff ρ_1 visits infinitely often F_1 and ρ_2 infinitely often F_2.

□

- We need to check if there exists a path in a kripke model $\mathcal{M}$ which satisfies φ.

LTL on Graphs XI

- For this, we need to construct $\mathcal{A}_\varphi$, and check if $L(\mathcal{A}_{\mathcal{M},s}) \cap L(\mathcal{A}_\varphi)$ for nonemptiness.
- If that language is non-empty, we get a trace in the automata $M' = (Q_1 \times Q_2 \times \{1,2,3\}, \Sigma, (q_{10}, q_{20}, 1), \Delta', Q_1 \times Q_2 \times 3)$
- by construction of the intersection automata, we can extract the corresponding trace in $M = (Q_1, \Sigma, q_{10}.\Delta_1, F_1)$ by just projecting out the components. This is an infinite path satisfying φ.
- This means that *trajectory synthesis* on LTL formulae is possible in polynomial time of the size of the graph, and exponential in the size of the formula.
- For STL, we will have edge weights as well. For simplicity, edge weights are proportional to distance between the subsequent lanes (check figure below). This approximates the time taken to traverse a particular lane.

The Kernel Trick Paper

Steps to convert STL formulas into kernels

- **Characterize formulae by semantics**:
  1. View STL formula φ as a map $\rho(\varphi, \cdot) : \mathcal{T} \to \mathbb{R} \cong \mathbb{R}^{\mathcal{T}}$ of trajectories to their robustness
  2. This is an *infinite-dimensional* vector of reals.
  3. Therefore, we need kernels to help with the computation
- **Define the Kernel**:
  1. $$k(\varphi, \psi) = \langle \rho(\varphi, \cdot), \rho(\psi, \cdot) \rangle = \int_{\xi \in \mathcal{T}} \rho(\varphi, \xi) \rho(\psi, \xi) \, d\xi$$
  2. Analogous to $\langle \vec{x}, \vec{y} \rangle = \sum_i x_i y_i$
- **Define a probability measure over the trajectories**:

$$k(\varphi, \psi) = \int_{\xi \in \mathcal{T}} \rho(\varphi, \xi) \rho(\psi, \xi) \, d\mu_0(\xi) \tag{1}$$

- **Normalize Robustness**: Preserve satisfiability equivalence while reducing outliers. $\pi(x_1, \ldots, x_n) = (f_\pi(x_1, \ldots, x_n) \geq 0) \Rightarrow \hat{\rho}(\pi, \xi, t) = \tanh\left(f_\pi(x_1, \ldots, x_n)\right)$

The Optimization Problem I

- Let $\vec{x_o}$ be the given task
- We wish to compute the following -

$$\underset{\vec{x} \in LogicLayer}{\arg\max} \quad \langle \vec{x}, \vec{x_o} \rangle + \rho(\mathsf{Repr}(\vec{x}) \wedge \phi_2 \wedge \phi_3, \mathsf{DoubleIntegralSoln}(\cdot)) + ||\vec{x}||_2$$

where $\mathsf{Repr}(\vec{x})$ is the formulaic representaion of $\vec{x}$ obtained from the $LogicLayer$ and $\mathsf{DoubleIntegralSoln}(\cdot)$ is the solution obtained for the formula $Repr(\vec{x}) \wedge \phi_2 \wedge \phi_3$. The last term $||\vec{x}||_2$ is a regularization term which can be defined appropriately.

Note #1

The $LogicLayer$ will be different from the one used in FLoRA. However, the semantics will be exactly the same, just with different predicates more in tune with a specification rather than a scoring rule.

The Optimization Problem II

Note #2

$$\langle \vec{x}, \vec{x_o} \rangle = k(\vec{x}, \vec{x_o})$$

which is evaluated using monte-carlo trials (as described earlier).

We now consider how we can derive the gradients of each of the terms.

Gradient of the inner product

We note that $\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{N} x_i y_i$. Therefore, if we take the gradient of this with respect to $\vec{x}$, it will simply be $(y_1, y_2, \cdots y_n)$ which is the vector $\vec{y}$.

The Optimization Problem III

Gradient of the Robustness

- Recall the definition of robustness.
  1. $\rho(s_t, \mu_c) = c - \mu(x_t)$ where μ_c denotes the predicate $\mu(x) < c$
  2. $\rho(s_t, \neg\phi) = -\rho(s_t, \phi)$
  3. $\rho(s_y, \phi \wedge \psi) = min(\rho(s_t, \phi), \rho(s_t, \psi))$
  4. $\rho(s_t, \Diamond_{[a,b]}\phi) = \max_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi)$
  5. $\rho(s_t, \Box_{[a,b]}\phi) = \min_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi)$
  6. $\rho(s_t, \phi \mathcal{U}_{[a,b]}\psi) =$
     $\max_{t' \in [t+a, t+b]}(min(\rho(s'_t, \psi), \min_{t'' \in [0, t']} \rho(s''_t, \phi))$

  Many of these formulae seem to be piecewise, and not that gradient friendly. `stlcg` is an approach to compute the quantitative semantics of Signal Temporal Logic (STL) formulas using computation graphs. Now, gradients can be taken!