2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Receding Horizon Planning with Rule Hierarchies for Autonomous Vehicles

Sushant Veer, Karen Leung, Ryan K. Cosner, Yuxiao Chen, Peter Karkus, and Marco Pavone

***Abstract*— Autonomous vehicles must often contend with conflicting planning requirements, e.g., safety and comfort could be at odds with each other if avoiding a collision calls for slamming the brakes. To resolve such conflicts, assigning importance ranking to rules (i.e., imposing a rule hierarchy) has been proposed, which, in turn, induces rankings on trajectories based on the importance of the rules they satisfy. On one hand, imposing rule hierarchies can enhance interpretability, but introduce combinatorial complexity to planning; while on the other hand, differentiable reward structures can be leveraged by modern gradient-based optimization tools, but are less interpretable and unintuitive to tune. In this paper, we present an approach to equivalently express rule hierarchies as differentiable reward structures amenable to modern gradient-based optimizers, thereby, achieving the best of both worlds. We achieve this by formulating *rank-preserving reward functions* that are monotonic in the rank of the trajectories induced by the rule hierarchy; i.e., higher ranked trajectories receive higher reward. Equipped with a rule hierarchy and its corresponding rank-preserving reward function, we develop a two-stage planner that can efficiently resolve conflicting planning requirements. We demonstrate that our approach can generate motion plans in ∼7-10 Hz for various challenging road navigation and intersection negotiation scenarios.**

I. Introduction

Autonomous Vehicles (AVs) must satisfy a plethora of rules pertaining to safety, traffic rules, passenger comfort, and progression towards the goal. These rules often, unfortunately, conflict with each other when unexpected events occur. For instance, avoiding collision with a stationary or dangerously slow non-ego vehicle on the highway might necessitate swerving on to the shoulder, violating the traffic rule to keep the shoulder clear. To resolve the juxtaposing requirements posed by these rules, ordering them according to their importance in a hierarchy was proposed in [1], which induces a ranking on trajectories. Trajectories that satisfy higher importance rules in the hierarchy are ranked higher than those trajectories which satisfy lower importance rules; see Fig. 1 for an illustration of rule hierarchies.

Rule hierarchies provide a systematic approach to plan motions that prioritize more important rules (e.g., safety) over the less important ones (e.g., comfort) in the event that all rules cannot be simultaneously satisfied. Furthermore, they offer greater transparency in planner design and are more amenable to introspection. However, rule hierarchies introduce significant combinatorial complexity to planning; in the worst-case, an N-level rule hierarchy could require solving 2^N optimization problems. An alternate to rule hierarchies is to plan with a "flat" differentiable reward function comprised of weighted contributions from all the rules. The weights can be tuned manually or using data [2], [3]. Although such flat differentiable reward structures are amenable to planning with standard optimization tools, they are less interpretable and diagnosable. AV developers are, therefore, faced with the choice between more interpretable but challenging to plan with rule heirarchies and more opaque but easy to plan with standard reward structures. In this paper, we show that rule hierarchies can, in fact, be unrolled in a principled manner into flat differentiable rewards by leveraging recent developments in differentiable Signal Temporal Logic (STL) representations [4] and the notion of rank-preserving reward functions that we introduce. Hence, the flat differentiable representation of hierarchies achieves the best of both worlds: the expressive power of hierarchies and the computational advantages of gradient-based optimization through modern optimization tools.

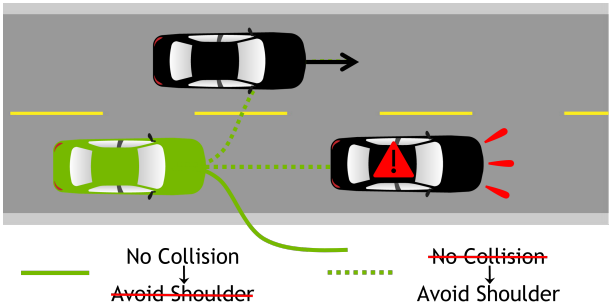


Fig. 1: Illustration of rule hierarchy. Ego vehicle is green and non-ego vehicles are black. The rule hierarchy has two rules ordered in decreasing importance: (i) No collision, (ii) Avoid shoulder. Non-ego vehicle with exclamation suddenly brakes. Ego trajectory that prevents collision but does not avoid the shoulder (solid green) has a higher rank than the trajectories that collide (dashed green).

Our approach for planning with rule hierarchies only solves *two* optimization problems (instead of the worst-case 2^N) regardless of the choice of N. The key to achieving this is the formulation of a *rank-preserving reward function* R that exhibits the following property: higher ranked trajectories that satisfy more important rules receive a higher reward compared to trajectories that only satisfy lower importance rules. Maximizing this reward directly allows our planner to choose trajectories that have a higher rule-priority ordering without checking all combinations of rules. However, this reward function is highly nonlinear and suffers from an abundance of local maxima. To overcome this challenge, we use a two-stage planning approach: the first stage searches through a finite set of primitive trajectories and selects the

Sushant Veer, Yuxiao Chen, and Peter Karkus are with NVIDIA Research {{sveer,yuxiaoc,pkarkus}@nvidia.com}. Karen Leung is with the University of Washington and NVIDIA Research {kymleung@uw.edu, kaleung@nvidia.com}. Ryan Cosner is with the California Institute of Technology {rkcosner@caltech.edu} (this work was conducted while Ryan was an intern at NVIDIA Research). Marco Pavone is with Stanford University and NVIDIA Research {pavone@stanford.edu, mpavone@nvidia.com}.

trajectory that satisfies the highest priority rules; the second stage warm starts a continuous optimization for maximizing the reward R with the trajectory supplied by the first-stage. To the best of our knowledge, this is the first work that plans for AVs in real-time with rule hierarchies. Prior work [5], [6] considered tracking an a priori given reference trajectory under rule hierarchies; in this paper, we do not assume the availability of an a priori reference trajectory.

Statement of Contributions. Our contributions are three-fold: (i) We introduce the notion of a rank-preserving reward function in Definition 1 and provide a systematic approach to constructing such a function for any given rule hierarchy in Theorem 1 and Remark 1. (ii) We present a two-stage receding-horizon planning approach that leverages the rank-preserving reward to efficiently plan with rule hierarchies and operates at a frequency of around $\sim$7-10 Hz. (iii) We demonstrate the ability of the rank-preserving reward based planner to rapidly adapt to challenging road navigation and intersection negotiation scenarios online without any scenario-specific hyperparameter tuning.

II. Related Works

Tuning Reward/Cost Functions. Inverse optimal control [2] and inverse reinforcement learning [3] offer approaches to tune the contribution of competing criteria in the total reward/cost function by harnessing data. Various recent works use data to learn such reward functions for autonomous vehicles by learning a trajectory scoring function [7], for planners that combine behavior generation and local motion planning [8], and by leveraging trajectory preferences of an expert [9], [10], among many others. Due to the challenges associated with learning reward functions and then using them in traditional trajectory optimizers, some approaches directly learn a data-driven policy via imitation learning [11] or reinforcement learning [12]. In this paper, we present a method that, given a rule hierarchy, analytically provides a "weighting" of the reward terms associated with each rule without requiring any additional data; see Theorem 1.

Hierarchical Specifications. The use of hierarchies for negotiating between various criteria in an optimization was proposed as early as 1967 [13] to avoid tuning the weights on individual cost functions in their aggregated weighted sum. Hierarchical optimizations have also appeared in the field of logic programming [14] to "optimally" negotiate between strict requirements and non-strict preferences. Recently, satisfiability modulo theories [15], [16] have been used to detect [17] and react to traffic-rule violations [18]. Notions of maximum satisfaction and minimum violation of rules [19], [20], [21] have also been studied in the Linear Temporal Logic (LTL) and Signal Temporal Logic (STL) literature. Philosophically our paper shares the approach of formulating a scalar-valued function that reflects the rank of rule satisfaction with [19] and [21]. However, [19] considers finite-state dynamics, while the conjunction aggregation in [21, Equation (7)] does not satisfy the *strict* rank-preserving property we enforce in Definition 2; i.e., if the rank of a trajectory is higher, then its reward should be strictly higher. The strictness of Definition 2 facilitates efficient planning by clearly distinguishing trajectories with different ranks. Unlike the papers discussed above, we formulate a reward function in Theorem 1 which satisfies the strict Definition 2 and then use it to plan motions for AVs in complex driving scenarios.

Rule Hierarchies for AVs. Recently, rulebooks—which are a pre-ordering of rules—for AVs were proposed in [1] as "blue-prints" for desirable driving preferences. The rulebook, being a pre-ordering, can be adapted to specific geographic and cultural driving norms into total-order rulebooks for planning and control while retaining the hierarchy structure prescribed by the pre-order. The consistency and completeness of rulebooks were investigated in [22] using formal methods, [23] used rulebooks for verification and validation of motion plans, and [24] used rulebooks to model human-driving behavior. A control strategy to track an a priori given reference trajectory while satisfying a total-order rulebook was presented in [5], [6]. We remark that none of the above mentioned papers generate motion plans for AVs that adhere to rule hierarchies *online*, unlike our paper.

III. Problem Formulation

Ego Dynamics. We refer to the AV as the ego vehicle. Let $x \in \mathcal{X} \subseteq \mathbb{R}^n$ be the ego's state and $u \in \mathcal{U} \subseteq \mathbb{R}^m$ be the control inputs. The ego exhibits discrete-time dynamics:

$$x_{t+1} = f(x_t, u_t), \tag{1}$$

where t represents the discrete-time and $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is continuously differentiable. Let $\mathcal{S}$ represent the space of trajectories generated by (1) starting from any admissible initial state $x_0 \in \mathcal{X}$ and evolving under the influence of any control sequence $u_{0:T}$ spanning a horizon of T time-steps.

Non-ego Agents and Map. We denote the state of non-ego agents by x_{ne} and the world map which contains information regarding lane lines, stop signs, etc. by x_{map}. For notational convenience, we augment non-ego agent state trajectories $x_{\mathrm{ne},0:T}$ and the world map x_{map} to create the world scene $w := (x_{\mathrm{ne},0:T}, x_{\mathrm{map}}) \in \mathcal{W}$. In practice, the planner will receive a predicted world scene $\hat{w}$ which is generated from maps and by the prediction [25], [26] modules in an AV stack. However, dealing with these modules is beyond the scope of this paper which focuses on planning with rule hierarchies. Therefore, in the interest of staying focused on the planner, we assume availability of the world scene w. We discuss possible extensions with predicted $\hat{w}$ in Section VII.

Rules and Rule Robustness. We define a rule $\phi : \mathcal{S} \times \mathcal{W} \to \{\texttt{True}, \texttt{False}\}$ as a boolean function that maps an ego trajectory $x_{0:T} \in \mathcal{S}$ and the world scene $w \in \mathcal{W}$ to `True` or `False`, depending on whether the trajectory satisfies the rule or not. The robustness $\hat{\rho}_i : \mathcal{S} \times \mathcal{W} \to \mathbb{R}$ of a rule ϕ_i is a metric that provides the degree of satisfaction for a rule. The robustness is a positive scalar if the rule is satisfied and negative otherwise; more positive values indicate greater satisfaction of the rule while more negative values indicate greater violation. We express the rules as STL formulae [27] using STLCG [4], which comes equipped with backpropagatable robustness metrics. Expressing rules as STL formulae allows us to easily encode complex spatio-temporal specifications, such as stop in front of a stop sign for at least 1 second before driving on.

Rule Hierarchy. A rule hierarchy φ is defined as a sequence of rules $\varphi := \{\phi_i\}_{i=1}^{N}$ where the highest priority rule is

indexed by 1 and lowest by N. The robustness of a rule hierarchy φ for a trajectory $x_{0:T} \in \mathcal{S}$ in a world scene $w \in \mathcal{W}$ is an N-dimensional vector-valued function whose elements comprise the robustness of the N-rules in φ, expressed formally as $\hat{\rho} : (x_{0:T}, w) \mapsto (\hat{\rho}_1(x_{0:T}, w), \cdots, \hat{\rho}_N(x_{0:T}, w))$.

Rank of a Trajectory. The rule hierarchy gives rise to a total order on trajectories. If a trajectory satisfies all the rules, then it has the highest rank in the order, while if it satisfies all rules but the lowest priority rule, then it has second rank, and so on; see Table I for further clarity on trajectory ranking via a 3-rule example. Given a trajectory $x_{0:T}$ and the world scene w, let the robustness vector for the trajectory be defined as $\rho := \hat{\rho}(x_{0:T}, w)$. Using the robustness vector we formally define the rank of a trajectory:

Rank	Satisfied Rules
1	ϕ_1, ϕ_2, ϕ_3
2	ϕ_1, ϕ_2
3	ϕ_1, ϕ_3
4	ϕ_1
5	ϕ_2, ϕ_3
6	ϕ_2
7	ϕ_3
8	$\emptyset$

TABLE I: Illustration of trajectory ranks for three rules.

Definition 1 (Rank of a Trajectory)**.** *Let φ be a rule hierarchy with N rules. Given a trajectory $x_{0:T}$ and the world scene w, let $\rho := (\rho_1, \rho_2, \cdots \rho_N)$ be the robustness vector of the trajectory as defined above. Let* $\text{step} : \mathbb{R} \to \{0, 1\}$ *map negative real numbers to* 0 *and all other real numbers to* 1*. Then the rank $r : \mathbb{R}^N \to \{1, 2, \cdots, 2^N\}$ is defined as:*

$$r(\rho) := 2^N - \sum_{i=1}^{N} 2^{N-i} \text{step}(\rho_i). \tag{2}$$

Problem. We want to solve the following optimization to obtain control inputs that result in a trajectory with the highest achievable rank in accordance with a rule hierarchy:

$$\begin{aligned} \min_{u_{0:T}} \quad & r \circ \hat{\rho}(x_{0:T}, w) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t), \text{ for } t = 1, \cdots, T. \end{aligned} \tag{3}$$

In particular, we want to solve this problem efficiently without checking all 2^N combinations of rule satisfaction. Note that the constraints for this optimization, such as control bounds, can be baked in the rule hierarchy φ.

IV. Rank-Preserving Reward Function

In this section, we present a differentiable rank-preserving reward function which enables us to circumvent the combinatorial challenges in solving (3) naïvely. We first define the notion of a rank-preserving reward function that assigns higher rewards for trajectories with higher rank and lower rewards for trajectories with lower rank.

Definition 2 (Rank-Preserving Reward Function)**.** *A rank-preserving reward function $R : \rho \mapsto R(\rho) \in \mathbb{R}$ satisfies:*

$$r(\rho) < r(\rho') \implies R(\rho) > R(\rho') \ . \tag{4}$$

Definition 2 does not impose any requirement on the reward if $r(\rho) = r(\hat{\rho})$, i.e., if two trajectories have the same rank. The choice of how the reward should serve as a tie-breaker in this event is left to the designer. In the next theorem we provide a candidate hierarchy-preserving reward and then rigorously show that it satisfies Definition 2.

Theorem 1 (Rank-Preserving Reward Function)**.** *Let $a > 2$ and let $\rho_i \in [-a/2, a/2]$ for all $i \in \{1, 2, \cdots, N\}$. Let the reward function R be defined as follows:*

$$R(\rho) := \sum_{i=1}^{N} \left(a^{N-i+1} \text{step}(\rho_i) + \frac{1}{N} \rho_i \right). \tag{5}$$

Then R satisfies Definition 2.

The proof of Theorem 1 is presented in the Appendix. The key idea behind the construction of (5) is to ensure that the reward contribution on satisfaction of rule i should exceed the sum of the reward contributions by all rules with lower priority. This is achieved by multiplying the step function with a constant that grows exponentially with the priority of a rule. To distinguish between trajectories that have the same rank, we use the average robustness for all N rules as a criterion; note that the ρ_i/N term in (5) sums up to the average robustness across all rules in the hierarchy.

Remark 1 (Differentiable Reward)**.** *The reward* (5) *is not differentiable since it involves step functions. To facilitate continuous optimization using this reward, we approximate the step functions by sigmoids as follows:*

$$R(\rho) := \sum_{i=1}^{N} \left(a^{N-i+1} \text{sigmoid}(c\rho_i) + \frac{1}{N} \rho_i \right), \tag{6}$$

where $c > 0$ is a scaling constants which is chosen to be large to mimic the step function.

To provide more intuition on the reward function, in Fig. 2, we plot the differentiable reward (6) of a 2-rule hierarchy (with $a = 2.01$, $c = 30$) by varying robustness ρ_1 and ρ_2 in $[-1, 1]$. The quadrant where $\rho_1, \rho_2 < 0$ (neither rule satisfied) has the lowest reward while the quadrant where $\rho_1, \rho_2 > 0$ (both rules satisfied) has the highest reward. Of the remaining two quadrants, the one with $\rho_1 > 0$ (more important rule satisfied) has a higher reward than the one with $\rho_2 > 0$ (less important rule satisfied). These observations align well with Definition 2.

V. Receding Horizon Planning With Rule Hierarchies

Equipped with a rule hierarchy and a method to cast it into a (nonlinear) differentiable reward function, we now present a two-stage algorithm to tractably solve (3). In the following sections, we describe the two stages: a coarse trajectory selection followed by a refinement process.

A. Stage 1: Planning with Motion Primitives

The objective of the first stage is to generate a coarse initial trajectory for warm-starting the continuous optimizer

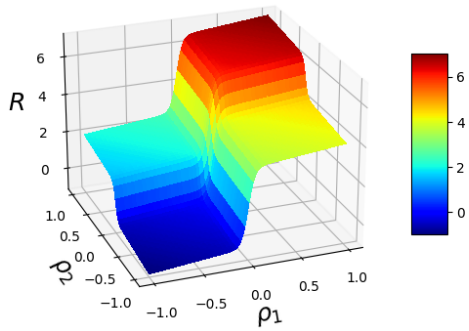


Fig. 2: Visualization of the reward for a 2-rule hierarchy.

in the second stage; we note that such multi-stage planning is common in the literature [28], [29]. We achieve this by selecting the trajectory with the largest rule-hierarchy reward R from a finite family of motion primitives. To generate the family of primitives $\mathcal{T}$, we use the approach presented in [25]. We first choose a set $\mathcal{M}$ of open-loop controls that coarsely cover a variety of actions that the AV can take; e.g., `(accelerate, turn right)`, `(decelerate, keep straight)`, etc. With the open-loop controls, the dynamics are propagated forward from the initial state for τ time-steps to generate $|\mathcal{M}|$ branches, where $|\mathcal{M}|$ is the cardinality of $\mathcal{M}$. At the terminal nodes of each of these $|\mathcal{M}|$ branches, all control actions are applied again for τ time-steps. This process is inductively repeated for the entire time horizon T to produce a tree with $|\mathcal{M}|^{\lceil T/\tau \rceil}$ branches; see Fig. 3 for visualizing the motion primitives. The tree can be created efficiently by parallelizing the branch generation.

B. Stage 2: Continuous Trajectory Optimization

The objective of the second stage is to refine the trajectory obtained from the first stage by solving the following optimization problem

$$\begin{aligned} \min_{u_{0:T}} \quad & -R \circ \hat{\rho}(x_{0:T}, w) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t), \text{ for } t = 1, \cdots, T \ . \end{aligned} \tag{7}$$

We reiterate that the reward function is analytically differentiable due to Remark 1 and the differentiability of the robustness of STL formulae afforded by STLCG [4]. Hence, we compute the gradients analytically and use the Adam optimizer [30] to solve (7). Owing to the lack of convexity of this optimization problem, we have no assurance of global optimality. Regardless, even convergence to a local optima in the vicinity of the initialization improves the trajectory's compliance to the rule hierarchy.

A psuedo-code for our two-stage motion planner is provided in Algorithm 1 below.

VI. Experimental Evaluation

In this section we demonstrate the ability of our rule-hierarchy based receding horizon planner to navigate various complex driving scenarios while maintaining a high planning frequency. The simulations are performed in Python using the `highway-gym` [31] and tested on a desktop computer with an `AMD Threadripper Pro 3975WX` CPU and an `NVIDIA RTX 3090` GPU. We use PyTorch [32] for motion planning to facilitate parallelization on GPU and analytical gradient computation via backpropagation and use STLCG [4] to encode STL formulae.

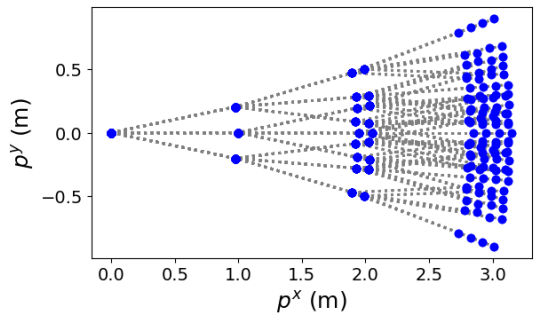


Fig. 3: Motion primitive tree for $|\mathcal{M}| = 6$, $T = 3$, and $\tau = 1$ at vehicle position $(0, 0)$ m, heading 0 rad, and speed 10 m/s.

Dynamics. We use the kinematic bicycle model [33] for the ego vehicle's dynamics. The state $x := (p^x, p^y, \psi, v)$ comprises of the position (p^x, p^y) of the vehicle's center in an inertial frame of reference, its orientation ψ measured with respect to the horizontal axis of the inertial frame, and its heading velocity in the vehicle's frame of reference. The control inputs $u := (\alpha, \delta)$ comprise of the acceleration α and the steering angle δ; see [33, Section II] for further details.

Hierarchy-preserving Reward Function. We use the reward function (6) described in Remark 1 with $a = 2.01$. The robustness values directly returned by STLCG [4] do not necessarily lie within $[-a/2, a/2]$, as is required by Theorem 1. To remedy this, we scale the robustness values using $\tanh$. Let ρ_{STLCG} be the robustness of an STL formula provided by STLCG. Then we use $\rho = \tanh(\rho_{\text{STLCG}}/s)$ as the robustness in our reward function (6), where $s > 0$ is a scaling constant chosen based on the range of values that ρ_{STLCG} takes. We remark that in our study, choosing s was straightforward and did not require much tuning.

Planning. For the first planning stage, the motion primitives are generated using the method described in Section V-A. We choose $\mathcal{M}$ as the mesh grid generated by the Cartesian product of the accelerations $\{-5, 5\}$ m/s^2 with the steering angles $\{-\pi/8, 0, \pi/8\}$ rad; consequently, $|\mathcal{M}| = 6$. The motion primitive tree is generated for $T = 10$ and $\tau = 2$, and therefore, has 6^5 branches. The tree generation and choosing the branch with the highest reward takes less than 0.04 s due to parallelization on the GPU. A sample motion primitive tree is plotted in Fig. 3 for visualization. As described in Section V-B, the second-stage of the planner performs continuous trajectory optimization using Adam [30] to solve (7). We set the learning rate at 0.01 and the maximum iterations K (see line 10 in Algorithm 1) as 10. For all forthcoming examples, we use a planning horizon of $T = 10$ and execute only the first action (i.e. $t_{\text{execute}} = 1$; see line 15 in Algorithm 1) from the plan before replanning.

Algorithm 1 Planning with a Rule Hierarchy

1: **Input:** Reward function R for the rule hierarchy φ
2: **Hyperparameters:** total planning time $\overline{T}$, planning horizon T, number of time-steps to execute t_{execute}
3: **Hyperparameters for Planning with Primitives:** set of open-loop controls for primitive tree generation $\mathcal{M}$, number of time-steps τ for which a control in $\mathcal{M}$ is executed
4: **Hyperparameters for Continuous Optimizer:** learning rate `lr`, maximum iterations K,
5: **while** $t < \overline{T}$ **do**
6: $x_t \leftarrow$ `updateEgoState()`
7: $w \leftarrow$ `updateWorldScene()`
8: $\mathcal{T} \leftarrow$ `generatePrimitiveTree`$(x_t, \mathcal{M}, \tau, T)$
9: $u_{t:t+T} \leftarrow \arg\max_{\mathcal{T}} R$
10: **for** k in $1 : K$ **do**
11: Compute $-\nabla_{u_{t:t+T}} R$
12: $u_{t:t+T} \leftarrow$ `Adam`$(u_{t:t+T}, -\nabla_{u_{t:t+T}} R,$ `lr`$)$
13: **end for**
14: `Execute`$(u_{t:t+t_{\text{execute}}})$
15: $t \leftarrow t + t_{\text{execute}}$
16: **end while**

Priority	Rule Description
1	No Collision
2	Do not cross solid lane line
3	Do not cross dashed lane line
4	Orient along the lane by the end of the planning horizon
5	Speed ≥ 2 m/s
6	Speed ≤ 15 m/s

TABLE II: Rule hierarchy for road navigation scenarios.

A. Road Navigation

We use six rules in the rule hierarchy, listed in Table II, for the scenarios in Fig. 4. The highest priority rule is collision avoidance which requires the ego's position to lie outside a $10\ \mathrm{m} \times 4\ \mathrm{m}$ rectangluar zone around both non-ego vehicles. The second and third rules incentivize the ego vehicle to not cross the solid-white lane line on its right and the dashed-white lane line on its left, respectively. Not crossing the solid-white lane line is prioritized over dashed-white lane line in accordance with common traffic laws. The fourth rule incentivizes the ego to be aligned with the lane by the end of the planning horizon to prevent oscillations about the lane center. Finally, the last two rules require the ego to maintain a lower and upper speed limit. The lower speed limit ensures that the ego vehicle only stops if necessary.

1) Overtake from lane: In this scenario, shown in Fig. 4(a), the yellow ego vehicle is moving at a sufficiently high speed that by the time it observes the stationary orange vehicle, it cannot stop in time to avoid a collision. Therefore, a lane change is the only option to avoid a collision with the orange car. The blue non-ego vehicle is significantly faster than the ego vehicle, resulting in a gap on the left lane. Hence, our planner relaxes rule number 3 in Table II.

2) Overtake from shoulder: This scenario, shown in Fig. 4(b), is similar to the scenario discussed above, with the salient difference being that the blue non-ego vehicle moves at a similar speed as the yellow ego vehicle. Hence, changing the lane is not viable as it would result in a collision with the blue vehicle. Therefore, our planner relaxes rule 2 in Table II to overtake the orange car from the shoulder.

3) Stop instead of overtake: In this scenario, shown in Fig. 4(c), the only difference from the scenario in Fig. 4(b) is that the ego vehicle is moving slow enough to be able to stop before colliding with the orange vehicle. In compliance with the rule hierarchy in Table II, violating the lower priority speed rules is preferable to changing lanes, hence, the planner relaxes rule 5 and brings the ego vehicle to a halt.

4) Double-parked vehicle: In this scenario, shown in Fig. 4(d), the stationary orange vehicle is not blocking the entire lane, but is, instead, double-parked. The ego vehicle is able to navigate around the orange vehicle without ever leaving its lane. Our planner is able to find a trajectory that complies with all the rules in the rule hierarchy.

B. Intersection Negotiation

The rule hierarchy in this example includes seven rules, six of which are identical to the rule hierarchy used in Section VI-A. The only new addition is a stop-sign rule that requires the ego vehicle to stop for 1 second in front of a stop sign (denoted by the transparent red square in Fig. 5) before moving on. The rule hierarchy is listed in Table III. It is worth noting that the stop-sign rule conflicts with the minimum-speed rule; nonetheless, when the ego vehicle approaches a stop sign, the planner will violate the minimum-speed rule in favor of the stop-sign rule owing to their priorities in the rule hierarchy.

Priority	Rule Description
1	No Collision
2	Do not cross solid lane line
3	Do not cross dashed lane line
4	Stop at stop sign for at least 1 second
5	Orient along the lane by the end of the planning horizon
6	Speed ≥ 2 m/s
7	Speed ≤ 15 m/s

TABLE III: Rule hierarchy for intersection negotiation scenarios.

Scenario	Mean$\pm$Std (s)	Median (s)	Max (s)	Min (s)
Overtake from lane	0.094 ± 0.002	0.095	0.097	0.086
Overtake from shoulder	0.092 ± 0.001	0.092	0.094	0.086
Stop instead of overtake	0.091 ± 0.003	0.092	0.095	0.078
Double-parked vehicle	0.093 ± 0.001	0.093	0.095	0.085
Intersection: wait	0.105 ± 0.017	0.091	0.127	0.085
Intersection: go	0.104 ± 0.019	0.090	0.133	0.082

TABLE IV: Time statistics for planning. The statistics are computed for all planning cycles within a single run of a particular scenario.

1) Wait: In this scenario, shown in Fig. 5(a), the ego vehicle in yellow must stop at a stop sign for 1 second before crossing an intersection. A blue non-ego vehicle is driving on the perpendicular lane which has no stop sign. By the end of the ego vehicle's 1-second stop, the blue vehicle is already in the intersection, so the ego vehicle waits for the blue vehicle before crossing the intersection.

2) Go: In this scenario, shown in Fig. 5(b), the blue vehicle is still very far from the intersection by the time the ego vehicle finishes its 1-second stop at the stop sign, unlike the previous scenario. Therefore, the ego vehicle proceeds to go through the intersection first.

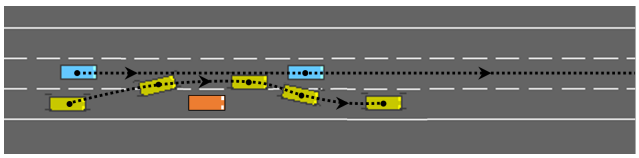
C. Discussion

We observe from Table IV that our planner, even in the worst-case, is able to generate motion plans within 0.1 s for road navigation scenarios and within 0.14 s for intersection negotiation scenarios. The key reason behind the real-time performance of our planner is the use of the rank-preserving reward function (6). Instead of going through 2^N rule-satisfaction combinations (in the worst case), the reward function (6) allows us to plan with just two optimizations per planning cycle, as described in Section V. Consequently, our planning times also exhibit low standard deviation; note that all four road navigation scenarios and both the intersection negotiation scenarios have comparable planning times.

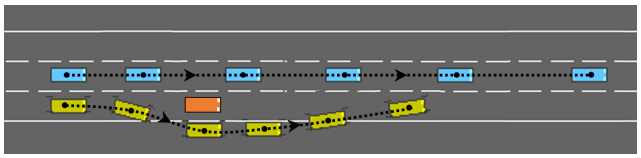
Another noteworthy point about our approach is the robustness of hyperparameters. We emphasize that all hyperparameters—robustness scaling constant, sigmoid sharpness, learning rate, etc.—were same across all scenarios in both settings, road and intersection navigation; besides the inclusion of a new stop-sign rule in the rule-hierarchy for intersection navigation. Finally, we emphasize again that planning with rule hierarchies facilitates complex decision making and seamless adaptation to a rich class of scenarios.

VII. Conclusions and Future Work

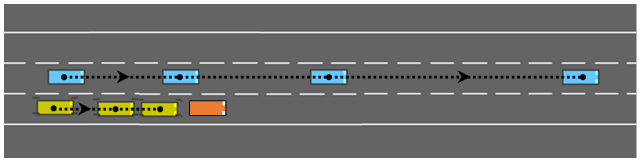
In this paper, we presented a framework for online motion planning of AVs with rule hierarchies. We achieved this



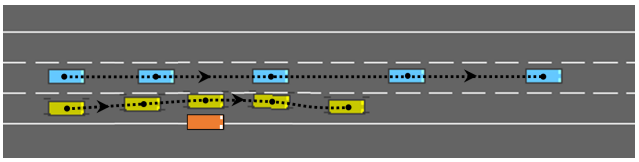
(a) Overtake from adjacent lane



(b) Overtake from shoulder

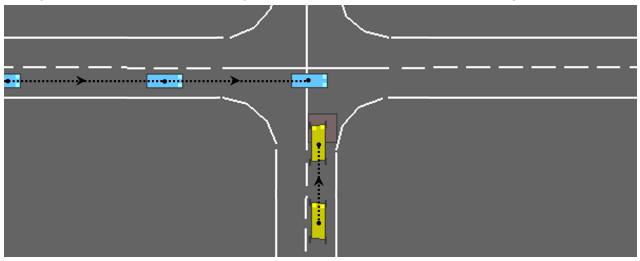


(c) Stop instead of overtake

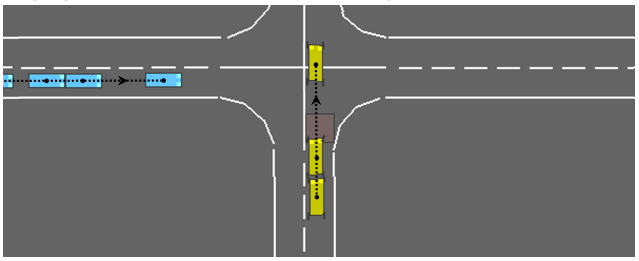


(d) Double-parked vehicle

Fig. 4: Snapshots of road navigation scenarios. The ego vehicle is in yellow, the moving non-ego vehicle is in blue, and the stationary non-ego vehicle is in orange. Dotted lines indicate trajectories of moving agents in the direction indicated by the arrows.



(a) Wait



(b) Go

Fig. 5: Snapshots of intersection navigation scenarios. The ego vehicle is in yellow and the non-ego vehicle is in blue. The transparent red-box is the stopping zone of the stop-sign. Dotted lines indicate trajectories of moving agents in the direction indicated by the arrows.

by formulating a hierarchy-preserving reward function that allowed us to plan without checking all combinations of rule satisfaction. The rule hierarchies were expressed as STL formulae which allowed us to incorporate temporal constraints, such as wait at the stop-sign before driving on. Finally, we demonstrated the ability of our planner to seamlessly adapt in road navigation and intersection negotiation scenarios.

This work opens up various exciting future directions. As an immediate future direction, we intend to perform closed-loop evaluation of our planner with data-driven trajectory predictors [34], [26] in state-of-the-art traffic simulators [35], [36]. Another interesting future direction is to imbue our planner with the ability to reason about estimation and prediction uncertainties by translating them to uncertainties in the rank induced by the rule-hierarchy. Finally, we note that rule hierarchies can model human driving behavior well, as suggested by [24]; hence, we hope to leverage rule hierarchies for estimating the degree of responsibility and responsiveness exhibited by other non-ego vehicles.

APPENDIX

Proof of Theorem 1. Let ρ and ρ' be the given robustness vectors for which $r(\rho) < r(\rho')$. Let the first index at which ρ satisfies a rule that is not satisfied by ρ' be

$$k := \min\{i \mid \text{step}(\rho_i) > \text{step}(\rho'_i),\ i \in \{1, 2, \cdots, N\}\}. \quad (8)$$

Now, decompose the reward function R as follows:

$$R(\rho) = \underbrace{\sum_{i=1}^{k-1} a^{N-i+1}\text{step}(\rho_i) + a^{N-k+1}}_{=:b} + \sum_{i=k+1}^{N} a^{N-i+1}\text{step}(\rho_i) + \frac{1}{N}\sum_{j=1}^{N}\rho_j.$$

The first term in the above, defined as a constant b, is the same for both ρ and ρ'. Since, $\text{step}(\rho_k)$ can only be 0 or 1, it follows from (8) that $\text{step}(\rho_k) = 1$ while $\text{step}(\rho'_k) = 0$. Hence, the reward for ρ' can be decomposed as follows:

$$R(\rho') = b + 0 + \sum_{i=k+1}^{N} a^{N-i+1}\text{step}(\rho'_i) + \frac{1}{N}\sum_{j=1}^{N}\rho'_j. \quad (9)$$

We make the following claim:

Claim 1: $\sum_{i=k+1}^{N} a^{N-i+1}\text{step}(\rho'_i) < a^{N-k+1} - a$.

Now we will prove this claim. Consider

$$\sum_{i=k+1}^{N} a^{N-i+1}\text{step}(\rho'_i) \le \sum_{i=k+1}^{N} a^{N-i+1} = \frac{a(a^{N-k}-1)}{a-1}.$$

Using $a > 2 \iff a - 1 > 1 \iff 1/(a-1) < 1$ above:

$$\sum_{i=k+1}^{N} a^{N-i+1}\text{step}(\rho'_i) \le \frac{a(a^{N-k}-1)}{a-1} < a(a^{N-k}-1), \quad (10)$$

completing the proof of Claim 1.

Now we make a second claim:

Claim 2: $\frac{1}{N}\sum_{j=1}^{N}\rho'_j - a \le \frac{1}{N}\sum_{j=1}^{N}\rho_j$.

The proof for this claim immediately follows by using $\rho_j, \rho'_j \in [-a/2, a/2]$ as follows:

$$\frac{1}{N}\sum_{j=1}^{N}\rho'_j - \frac{1}{N}\sum_{j=1}^{N}\rho_j \le \frac{1}{N}\left(\frac{aN}{2} + \frac{aN}{2}\right) = a \quad (11)$$

With both these claims established, use Claim 1 in (9) followed by Claim 2 to get

$$R(\rho') < b + a^{N-k+1} + \frac{1}{N}\sum_{j=1}^{N}\rho'_j - a \le R(\rho),$$

completing the proof of this theorem. □

ACKNOWLEDGMENT

We are grateful to Ryan Holben for helpful discussions on planning with rule hierarchies and to Wei Xiao for discussing his prior work on optimal control with rule hierarchies.

REFERENCES

[1] A. Censi, K. Slutsky, T. Wongpiromsarn, D. Yershov, S. Pendleton, J. Fu, and E. Frazzoli, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *Proc. IEEE Conf. on Robotics and Automation*, 2019, pp. 8536–8542.

[2] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," *arXiv preprint arXiv:1206.4617*, 2012.

[3] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Int. Conf. on Machine Learning*. Citeseer, 2000.

[4] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *arXiv preprint arXiv:2008.00097*, 2020.

[5] W. Xiao, N. Mehdipour, A. Collin, A. Y. Bin-Nun, E. Frazzoli, R. D. Tebbens, and C. Belta, "Rule-based optimal control for autonomous driving," in *Proc. ACM/IEEE Int. Conf. on Cyber-Physical Systems*, 2021, pp. 143–154.

[6] ——, "Rule-based evaluation and optimal control for autonomous driving," *arXiv preprint arXiv:2107.07460*, 2021.

[7] T. Phan-Minh, F. Howington, T.-S. Chu, S. U. Lee, M. S. Tomov, N. Li, C. Dicle, S. Findler, F. Suarez-Ruiz, R. Beaudoin, *et al.*, "Driving in real life with inverse reinforcement learning," *arXiv preprint arXiv:2206.03004*, 2022.

[8] S. Rosbach, V. James, S. Großjohann, S. Homoceanu, and S. Roth, "Driving with style: Inverse reinforcement learning in general-purpose planning for automated driving," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2019, pp. 2658–2665.

[9] D. Sadigh, A. Dragan, S. Sastry, and S. Seshia, "Active preference-based learning of reward functions," in *Proc. Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[10] S. M. Katz, A. Maleki, E. Bıyık, and M. J. Kochenderfer, "Preference-based learning of reward function features," *arXiv preprint arXiv:2103.02727*, 2021.

[11] M. Vitelli, Y. Chang, Y. Ye, A. Ferreira, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, and P. Ondruska, "Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies," in *Proc. IEEE Conf. on Robotics and Automation*, 2022, pp. 897–904.

[12] Z. Cao, E. Bıyık, W. Z. Wang, A. Raventos, A. Gaidon, G. Rosman, and D. Sadigh, "Reinforcement learning based control of imitative policies for near-accident driving," *arXiv preprint arXiv:2007.00178*, 2020.

[13] F. Waltz, "An engineering approach: hierarchical optimization criteria," *IEEE Transactions on Automatic Control*, vol. 12, no. 2, pp. 179–180, 1967.

[14] M. Wilson and A. Borning, "Hierarchical constraint logic programming," *The Journal of Logic Programming*, vol. 16, no. 3-4, pp. 277–318, 1993.

[15] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of model checking*. Springer, 2018, pp. 305–343.

[16] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *Proc. IEEE Conf. on Decision and Control*, 2017, pp. 1132–1137.

[17] Q. Zhang, D. K. Hong, Z. Zhang, Q. A. Chen, S. Mahlke, and Z. M. Mao, "A systematic framework to identify violations of scenario-dependent driving rules in autonomous vehicle software," *Proc. ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 2, pp. 1–25, 2021.

[18] Y. Lin and M. Althoff, "Rule-compliant trajectory repairing using satisfiability modulo theories," in *Proc. IEEE Intelligent Vehicles Symposium*, 2022, pp. 449–456.

[19] J. Tůmová, L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation ltl planning with conflicting specifications," in *Proc. American Control Conference*, 2013, pp. 200–205.

[20] R. Dimitrova, M. Ghasemi, and U. Topcu, "Maximum realizability for linear temporal logic specifications," in *Proc. Int. Symp. on Automated Technology for Verification and Analysis*. Springer, 2018, pp. 458–475.

[21] N. Mehdipour, C.-I. Vasile, and C. Belta, "Specifying user preferences using weighted signal temporal logic," *IEEE Control Systems Letters*, vol. 5, no. 6, pp. 2006–2011, 2020.

[22] T. Phan-Minh, K. X. Cai, and R. M. Murray, "Towards assume-guarantee profiles for autonomous vehicles," in *Proc. IEEE Conf. on Decision and Control*, 2019, pp. 2788–2795.

[23] A. Collin, A. Bilka, S. Pendleton, and R. D. Tebbens, "Safety of the intended driving behavior using rulebooks," in *Proc. IEEE Intelligent Vehicles Symposium*, 2020, pp. 136–143.

[24] B. Helou, A. Dusi, A. Collin, N. Mehdipour, Z. Chen, C. Lizarazo, C. Belta, T. Wongpiromsarn, R. D. Tebbens, and O. Beijbom, "The reasonable crowd: Towards evidence-based and interpretable models of driving behavior," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2021, pp. 6708–6715.

[25] E. Schmerling, K. Leung, W. Vollprecht, and M. Pavone, "Multimodal probabilistic model-based planning for human-robot interaction," in *Proc. IEEE Conf. on Robotics and Automation*, 2018, pp. 3399–3406.

[26] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data," in *Proc. European Conf. on Computer Vision*. Springer, 2020, pp. 683–700.

[27] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Springer Berlin Heidelberg, 2004, pp. 152–166.

[28] U. Rosolia, S. De Bruyne, and A. G. Alleyne, "Autonomous vehicle control: A nonconvex approach for obstacle avoidance," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 2, pp. 469–484, 2016.

[29] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[31] E. Leurent, "An environment for autonomous driving decision-making," https://github.com/eleurent/highway-env, 2018.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[33] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Proc. IEEE Intelligent Vehicles Symposium*, 2015, pp. 1094–1099.

[34] A. Kamenev, L. Wang, O. B. Bohan, I. Kulkarni, B. Kartal, A. Molchanov, S. Birchfield, D. Nistér, and N. Smolyanskiy, "Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation," in *Proc. IEEE Conf. on Robotics and Automation*, 2022, pp. 8936–8942.

[35] D. Xu, Y. Chen, B. Ivanovic, and M. Pavone, "BITS: Bi-level imitation for traffic simulation," *arXiv preprint arXiv:2208.12403*, 2022.

[36] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. Wolff, A. Lang, L. Fletcher, O. Beijbom, and S. Omari, "nuPlan: A closed-loop ml-based planning benchmark for autonomous vehicles," in *Proc. Conf. on Computer Vision and Pattern Recognition ADP3 workshop*, 2021.