Incremental Minimum-Violation Control Synthesis for Robots Interacting with External Agents

Pratik Chaudhari* Tichakorn Wongpiromsarn† Emilio Frazzoli*

Abstract— **We consider the problem of control strategy synthesis for robots that interact with external agents, together known as the *environment*. Both the robot and the environment are modeled as dynamical systems with differential constraints and take part in a nonzero-sum two-player differential game to fulfill their respective task specifications while satisfying a set of *safety rules*. They minimize a cost function that is representative of the *level of unsafety* with respect to these safety rules. Throughout, the problem is motivated by an autonomous car in an urban environment that interacts with other cars in situations such as navigating stop signs at road junctions and single-lane roads. Ideas behind sampling-based motion-planning algorithms are used to incrementally construct a finite Kripke structure abstraction of a continuous dynamical system. Model-checking techniques for safety rules expressed using Linear Temporal Logic (LTL) are then leveraged to propose an algorithm which synthesizes a control strategy for the two-player game. We analyze the algorithm to show that, with probability one, it converges to the Stackelberg equilibrium asymptotically. This algorithm is also demonstrated in a number of simulation experiments.**

I. Introduction

Autonomous cars have quickly transitioned from experimental projects such as the DARPA Grand Challenge [1] into urban mobility systems. As they increasingly share infrastructure with other drivers, it is essential to ensure that they interact with human-driven vehicles according to the rules of driving and safety on the road. Behaviors such as merging into busy lanes or handling stop-signs at cross-roads, which are typically enforced using right-of-way or even communication, are easy for human drivers but are arguably much harder for autonomous agents [2].

This paper addresses problems motivated from autonomous urban mobility where agents interact with other external agents. It draws from roughly three different areas, i.e., differential games, model checking and sampling-based motion-planning algorithms. Two players, the robot and the environment, take part in a differential game, while satisfying task specifications such as reaching a certain goal region subject to safety rules such as "always stay in the right lane" or "do not change lanes frequently". Such rules can be typically expressed as temporal logic formulas using formal languages such as Linear Temporal Logic (LTL) and deterministic μ-calculus.

* The authors are with the Massachusetts Insititute of Technology, Cambridge, MA, USA. Email: pratikac@mit.edu, frazzoli@mit.edu

† The author is with the Thailand Center of Excellence for Life Sciences, Thailand. Email: tichakorn@tcels.or.th

This work was initiated when the authors were visiting researchers at the Singapore-MIT Alliance for Research and Technology (SMART), Singapore. It is funded by NSF-Singapore through the SMART-FM IRG.

Differential games [3], [4] are popular models for problems such as multi-agent collision avoidance [5], and pursuit-evasion problems [6]. However, analytical solutions for differential games exist for only specific problems, e.g., the "lady in the lake" game, or Linear Quadratic Regulator (LQR) games [4]. For problems involving more complex dynamics or other kinds of cost functions, solutions are hard to characterize in closed form. Numerical techniques are based on converting the problem to a finite dimensional optimization problem [7] or solving the corresponding partial differential equations using shooting methods [8], level set methods [9] etc.

On the other hand, the problem of synthesizing control strategies that satisfy temporal logic specifications has been studied in a number of recent works [10], [11]. However, the main challenge here lies in finding good abstractions of continuous dynamical systems [12]. These methods are in general not complete, i.e., the algorithm may not be able to return a controller that satisfies task specifications even if one exists, depending upon the level of discretization.

Sampling-based motion planning algorithms, e.g., Probabilistic Road Maps (PRMs) and Rapidly-exploring Random Trees (RRTs), have been widely applied to solve the problem of finding an optimal trajectory for dynamical systems from an initial state to a goal state. Very recently, they have also seen application in solving pursuit-evasion games [13], and handling complex task specifications [14]. Algorithms such as PRM* and RRT* [15] which are computationally efficient counterparts of these algorithms have been instrumental in these applications, since they provide both probabilistic completeness and asymptotic optimality guarantees.

The main contribution of this work is a formulation to compute equilibria for two-player differential games where players try to accomplish a task specification while satisfying safety rules expressed using temporal logic. Building upon our previous work [16], we formulate the interaction between an autonomous agent and its environment as a non-zero sum differential game; both the robot and the environment minimize the *level of unsafety* of a trajectory with respect to safety rules expressed using LTL formulas. We abstract a continuous-time dynamical system with differential constraints into finite Kripke structures and employ model checking techniques to quantify the level of unsafety. We describe an algorithm to compute the open-loop Stackelberg equilibrium (OLS) of the differential game on these Kripke structures. It can be shown that the algorithm converges to the OLS as the number of samples in the Kripke structure goes to infinity. A number of examples motivated from autonomous urban mobility such as stop signs at junctions and single lane roads are discussed using simulation experiments.

This paper is organized as follows: After introducing preliminary concepts in Sec. II, we formally state the problem in Sec. III. Sec. IV details an algorithm to compute the Stackelberg equilibrium of the differential game, which in turn is analyzed in Sec. V. We demonstrate this algorithms using computational experiments in Sec. VI and conclude with directions for future work in Sec. VII.

II. Preliminaries

This section introduces material on Kripke structures for dynamical systems, finite automata, LTL and differential games used in this paper.

A. Durational Kripke Structures

Let Π be a set of atomic propositions. The cardinality and the set of all subsets of Π are denoted by $|\Pi|$ and 2^{Π}, respectively. Let $X_r \subset \mathbb{R}^{d_r}$ and $U_r \subset \mathbb{R}^{m_r}$ be compact sets. Consider a time-invariant dynamical system given by,

$$\dot{x}_r(t) = f_r(x_r(t), u_r(t)), \qquad x_r(0) = x_{r,0} \tag{1}$$

where $x_{r,0}$ is the initial state of the robot. Trajectories of the states and controls are denoted by $x_r : [0,T] \to X_r$ and $u_r : [0,T] \to U_r$, respectively, for some $T \in \mathbb{R}_{\geq 0}$. Similarly, let $X_e \subset \mathbb{R}^{d_e}$ and $U_e \subset \mathbb{R}^{m_e}$ be compact sets. The dynamical system for the environment is then given by,

$$\dot{x}_e(t) = f_e(x_e(t), u_e(t)), \qquad x_e(0) = x_{e,0} \tag{2}$$

The functions, $f_r : X_r \times U_r \to X_r$ and $f_e : X_e \times U_e \to X_e$ are assumed to be differentiable, measurable and Lipschitz continuous in both its arguments for existence and uniqueness of solutions. For the rest of this section X refers to both X_e and X_r while U refers to both U_e and U_r.

Let $\mathcal{L}_c : X \to 2^{\Pi}$ be a labeling function that maps each state to the atomic propositions that it satisfies. For a trajectory $x : [0,T] \to X$, define $\mathcal{D}(x) = \{t \,|\, \lim_{s \to t^-} \mathcal{L}_c(x(s)) \neq \mathcal{L}_c(x(t))\}$ to be the ordered set of discontinuities of $\mathcal{L}_c$. We assume that $\mathcal{D}(x)$ is finite for any x. Let $\mathcal{D}(x) = \{t_1, t_2, \ldots, t_n\}$. The finite timed word generated by x is defined to be, $\rho(x) = (\rho_0, \rho_1, \ldots, \rho_n)$, where $\rho_i = (l_i, d_i)$ such that, $l_i = \mathcal{L}_c(x(t_i))$ and $d_i = t_{i+1} - t_i$ for $0 \leq i < n$ with $t_0 = 0$, $l_n = \mathcal{L}_c(x(t_n))$ and $d_n = T - t_n$. The finite word $w(\rho)$ of x is then simply the projection $w(\rho) = (l_0, l_1, \ldots, l_n)$.

Definition 1 (Durational Kripke Structure [17]) *A durational Kripke structure is a tuple $K = (S, s_0, R, \Pi, \mathcal{L}, \Delta)$, where, (i) S is a finite set of states, (ii) $s_0 \in S$ is an initial state, (iii) $R \subseteq S \times S$ is a deterministic transition relation, (iv) Π is a set of atomic propositions, (v) $\mathcal{L} \colon S \to 2^{\Pi}$ is a state labeling function, and, (vi) $\Delta \colon R \to \mathbb{R}_{\geq 0}$ is a function assigning a time duration to each transition.*

A *trace* of K is a finite sequence of states $\xi = s_0, s_1, \ldots, s_n$ such that $s_0 = s_0$ and $(s_i, s_{i+1}) \in R$, for all $0 \leq i < n$. It produces a finite timed word $\rho = \rho_0 \rho_1 \ldots \rho_{n-1}$, with $\rho_i = (\mathcal{L}(s_i), \Delta(s_i, s_{i+1}))$ and $\rho_n = (\mathcal{L}(s_n), 0)$. For a word $w(\rho) = l_0 \ldots l_n$, with $l_i = \mathcal{L}(s_i)$, let $I = \{i_0, i_1, \ldots, i_k\}$ be a set of indices, such that $i_0 = 0$, $l_{i_j} = l_{i_j+1} = \ldots = l_{i_{j+1}-1} \neq l_{i_{j+1}}$, for all $0 \leq j \leq k-1$, and $l_k = l_{k+1} = \ldots = l_n$. Define $destutter(w(\rho)) = l_{i_0}, l_{i_1}, \ldots, l_{i_{k-1}}, l_{i_k}$. It is thus a word with all consecutive repeated symbols removed from $w(\rho)$.

We require Kripke structures to be *trace inclusive* with respect to the continuous dynamical system. A Kripke structure K is called trace inclusive if, $S \subset X$, $s_0 = x_0$, and there exists a trajectory $x : [0,T] \to X$ such that $x(0) = s_1$, $x(T) = s_2$ with $T = \Delta(s_1, s_2)$ and $|\mathcal{D}_x| \leq 1$ for all $(s_1, s_2) \in R$. It then follows that given any trace ξ of K, there exists a trajectory of the dynamical system such that $destutter(w(\rho)) = w(\rho(x))$ which helps us formalize when both ρ and x, satisfy certain specifications.

B. Linear Temporal Logic (LTL)

We briefly describe finite automata and LTL in this section. Please see [18], [19] for a more thorough exposition.

1) Finite Automata: A non-deterministic finite automaton (NFA), $A = (Q, q_0, \Sigma, \delta, F)$, is a tuple where, Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is an input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a non-deterministic transition relation, and, $F \subseteq Q$ is a set of accepting states. The semantics of NFAs are defined over finite words produced by durational Kripke structures (see Def. 1). In particular, in this work, the alphabet Σ is chosen to be 2-tuples of atomic propositions, i.e., $2^{\Pi} \times 2^{\Pi}$. Similar to [16], we assume, without loss of generality, that NFAs are non-blocking, i.e., $(q, (l_1, l_2), q') \in \delta$ for all q, l_1, l_2. As discussed in [19], for any NFA A, one can construct a non-blocking NFA that is equivalent to A by inserting a trapping state. The set of all words accepted by A, also known as its *language* is denoted by $L(A)$.

Given a non-blocking automaton A, let $\overline{A} = (Q, q_0, \Sigma, \overline{\delta}, F, W)$ be an automaton with priority $\varpi(A) \in \mathbb{N}$ such that, $\overline{\delta} = \delta \cup (q, (\sigma, \sigma'), q')$ for all $q, q' \in Q$ and $(\sigma, \sigma') \in \Sigma$. W is a weighing function such that $W(\tau) = 0$ if $\tau \in \delta$ and $W(\tau) = \varpi(A)$ otherwise for all $\tau \in \overline{\delta}$. $\overline{A}$ is referred to as the weighted automaton. Note that the language of $\overline{A}$ is simply Σ^*, i.e., it accepts all finite words, but weights words that do not belong to $L(A)$ using W.

Given a set of weighted NFAs, $\overline{A}_1, \ldots, \overline{A}_n$, we can construct their product $\overline{A}_\Psi = (Q_\Psi, q_{0,\Psi}, \Sigma, \delta_\Psi, F_\Psi, W_\Psi)$. The weight of a trace satisfying $W_\Psi(\tau) = (W_1(\tau_1), W_2(\tau_2), \ldots, W_n(\tau_n))$ where W_i is the weight function of automaton A_i and $\tau \in \delta_\Psi$ while $\tau_i \in \delta_i$ for all i. The rest of algorithm to construct the product automaton is standard [16]. Note that the language of the product automaton is the intersection of the languages of the component automata while the weight of each run is a tuple of the weights of runs on component automata.

2) Finite LTL: In this work, we use the class of LTL formulas without the next operator, which is denoted by $\mathrm{FLTL}_{-\mathsf{X}}$. An $\mathrm{FLTL}_{-\mathsf{X}}$ formula over a set Π of atomic propositions is inductively defined as follows:

- $(a, a') \in \Pi \times \Pi$ is a formula for all $a, a' \in \Pi \cup \{\mathsf{true}, \mathsf{false}\}$,

- For any two formulas ϕ_1, ϕ_2, $\phi_1 \vee \phi_2$, $\neg\phi_1$, $\phi_1 \mathsf{U} \phi_2$, $\mathsf{G}\,\phi_1$, and $\mathsf{F}\,\phi_1$ are also formulas.

Here, $\neg$ (negation) and $\vee$ (disjunction) are Boolean operators, and U, G, and F are temporal operators. In general, FLTL$_{-\mathsf{X}}$ formulas are interpreted on infinite traces [19]. In this work, we follow the semantics described in [20], which allows FLTL$_{-\mathsf{X}}$ formulas to be interpreted on finite traces. As shown in [20], FLTL$_{-\mathsf{X}}$ over finite traces can be automatically translated into equivalent finite automata.

C. Level of Unsafety

We now introduce the specific structure of cost functions considered in this work. The cost of a trace of the Kripke structure K is a lexicographical tuple where each element is the cost of violating a safety rule expressed in the form of a finite automaton, A. The priority $\varpi(A)$ is simply the index of the rule A in this tuple.

Definition 2 (Level of unsafety) *For a word $w = l_0, \ldots, l_n$ and subset $I = \{i_1, i_2, \ldots, i_k\} \subset \{0, 1, \ldots, n\}$, define $\text{vanish}(w, I) = l_0, \ldots, l_{i_1-1}, l_{i_1+1}, \ldots l_n$ to be the sub-sequence of w after erasing elements $l_{i_1}, \ldots, l_{i_k}$. The level of unsafety $\lambda(w, A)$ of w with respect to A is :*

$$\lambda(w, A) = \min_{I \subseteq \{0,\ldots,n\} \,|\, \text{vanish}(w,I) \in L(A)} |I| \ \varpi(A).$$

Similarly, $\lambda(\xi, A)$ for a trace $\xi = s_0, \ldots, s_{n+1}$ of the durational Kripke structure is :

$$\lambda(\xi, A) = \min_{I \subseteq \{0,\ldots,n\} | \text{vanish}(w(\rho),I) \in L(A)} \sum_{i \in I} \Delta(s_i, s_{i+1}) \varpi(A).$$

By convention, the level of unsafety of an empty trace or an empty word is zero.

For a set of rules $\Psi = (\Psi_1, \ldots, \Psi_n)$ with each rule $\psi_j \in \Psi_i$, for all $1 \le i \le n$ given in the form of a finite automaton A_{ij}, we define the level of unsafety $\lambda(w, \Psi)$ to be the ordered tuple, $(\lambda(w, \Psi_i), \ldots, \lambda(w, \Psi_n))$ where $\lambda(w, \Psi_i) = \sum_j \lambda(w, A_{ij})$. We use the standard lexicographical ordering on $\lambda(w, \Psi)$ to compare the level of unsafety of two words w_1, w_2. By picking the definition of the level of unsafety and the definition of the weighted automaton in Sec. II-B.1, we have ensured that the weight of the shortest accepting run of $\overline{A}$ over a word w is exactly equal to $\lambda(w, A)$.

Lemma 3 (Lem. 1 in [21]) *Any word $w \in \Sigma^*$ is accepted by $\overline{A}$ and the weight of the shortest accepting run is $\lambda(w, A)$.*

Similarly the role of the product $\overline{A}_\Psi$ is characterized as follows: its weights are chosen in such a way that the level of unsafety of any accepting run on it is equal to the level of unsafety of the word with respect to safety rules.

Definition 4 (Product Automaton P) *Given a trace-inclusive Kripke structure $K = (S, s_0, R, \Pi, \mathcal{L}, \Delta)$ and $\overline{A}_\Psi = (Q_\Psi, q_{0,\Psi}, \Sigma, \delta_\Psi, F_\Psi, W_\Psi)$, their product is $P = (Q_P, q_{0,P}, \delta_P, F_P, W_P)$ where*

- *$Q_P = S \times Q_\Psi$;*
- *$q_{0,P} = s_0 \times q_{0,\Psi}$;*
- *$(z, z') \in \delta_P$, $W_P(z, z') = (W_\Psi(\tau), \Delta(s, s'))$ if $(s, s') \in R$ and $(\tau) \in \delta_\Psi$ where $z = (s, q)$, $z' = (s', q')$ and the transition $\tau = (q, (\mathcal{L}(s), \mathcal{L}(s')), q')$ and,*
- *$F_P = S \times F_\Psi$.*

The product automaton can be used to compute $\lambda(\xi, \Psi)$ for any trace ξ of the Kripke structure by summing weights along its transitions. Note that the trace of the product with the smallest weight uniquely corresponds to a trace of the Kripke structure that minimizes the level of unsafety [16], [21].

D. Differential Games

For dynamics shown in Eqn. (1) and Eqn. (2), let $x = (x_r, x_e)^T$ be the combined state of the game such that,

$$\frac{dx}{dt} = f(x(t), u_r(t), u_e(t)) = \begin{bmatrix} f_r(x_r(t), u_r(t)) \\ f_r(x_e(t), u_e(t)) \end{bmatrix}, \quad (3)$$

for all $t \in \mathbb{R}_{\ge 0}$ with the initial state $x(0) = x_0 = (x_{r,0}, x_{e,0})^T$. The state of the game $x \in X = X_r \times X_e$. It is easily seen that conditions for existence and uniqueness of solutions of Eqn. (1) and Eqn. (2) given in Sec. II-A also ensure that Eqn. (3) has similar properties. Given a trajectory of the game $x : [0, T] \to X$, let the corresponding trajectories of $\mathbf{R}$ and $\mathbf{E}$ be x_r and x_e, respectively. Players $\mathbf{R}$, $\mathbf{E}$ optimize the cost functions $J_e(x_0, u_r, u_e)$, $J_r(x_0, u_r, u_e)$, respectively.

We consider an information structure wherein, $\mathbf{R}$ knows the cost function of $\mathbf{E}$, but the player $\mathbf{E}$ does not know the cost function of $\mathbf{R}$, i.e., it has no information of the intention of $\mathbf{R}$. $\mathbf{E}$ however knows the control strategy of player $\mathbf{R}$ and can take it into account while devising its strategy. Define BR to be the mapping from a trajectory $u_r : [0, T] \to U_r$ to $u_e : [0, T] \to U_e$ such that,

$$\text{BR}(u_r) = \arg\min_{u_e} J_e(x_0, u_r, u_e).$$

BR(u_r) is thus the best response that $\mathbf{E}$ can take given a control trajectory of $\mathbf{R}$. The player $\mathbf{R}$ then picks its best strategy u_r^*, such that,

$$J_r(x_0, u_r^*, u_e^*) \le J_r(x_0, u_r', \text{BR}(u_r'))$$

where $u_e^* = \text{BR}(u_r^*)$, for any u_r'. Such a strategy, i.e., (u_r^*, u_e^*) is called an open-loop Stackelberg (OLS) equilibrium for this differential game. Necessary conditions for existence of open-loop Stackelberg equilibria can be characterized as shown in [22].

Note that $\mathbf{E}$ solves an optimization problem given $\mathbf{R}$'s control u_r, however $\mathbf{R}$ can change its control if the game starts at some later time and if $\mathbf{R}$ can obtain a lower value for its cost function. For example, if an obstacle in the environment moves at some time $t_1 \in (0, T)$, the player $\mathbf{R}$ can change its control for times $t > t_1$ in order to obtain a lower cost. $\mathbf{E}$ then has no option but to devise a new strategy that minimizes its cost function with respect to $\mathbf{R}$'s new strategy. This property is known as *time inconsistency* and open-loop Stackelberg strategies are not time consistent. Feedback strategies, which can be shown to be time consistent are the subject of future work.

III. Problem formulation

This section formalizes the problem considered in this paper and models it as a nonzero-sum differential game between two players, the robot ($\mathbf{R}$) and the environment ($\mathbf{E}$). Both $\mathbf{R}$ and $\mathbf{E}$ minimize their respective cost functions while satisfying their task specifications. In the sequel, for clarity of presentation, we assume that the dynamics of both players is the same. Also, atomic propositions, Π, and safety rules Ψ, are same for both players. The formulation is however general and also applies to cases where players with different dynamics minimize cost for different sets of safety rules.

We consider the task specification defined as "traveling from an initial state to a final goal set without colliding with any obstacles or other agent". In this context, define compact sets $X_{r,obs}, X_{r,G} \subset X_r$ and $X_{e,obs}, X_{r,G} \subset X_e$. A trajectory of the game in Eqn. 3, x, can be projected to obtain trajectories x_r, x_e of players $\mathbf{R}, \mathbf{E}$ respectively. x_r is said to satisfy the task specification Φ_r if for some $T_r \in \mathbb{R}_{\geq 0}$; $x_r(0) = x_{r,0}$, $x_r(T_r) \in X_{r,G}$, $x_r(t) \notin X_{r,obs}$ and $\|x_r(t) - x_e(t)\|_2 > c$ for all $t \in [0, T_r]$ for a fixed constant c. Similarly, x_e is said to satisfy Φ_e if for some $T_e \in \mathbb{R}_{\geq 0}$; $x_e(0) = x_{e,0}$, $x_e(T_e) \in X_{e,G}$, $x_e(t) \notin X_{e,obs}$ and $\|x_r(t) - x_e(t)\|_2 > c$ for all $t \in [0, T_e]$.

A. Normalized Level of Unsafety

The level of unsafety for a set of safety rules in Def. 2 is a cost tuple and is compared using the lexicographical ordering. For convenience, we normalize it and convert it into a scalar cost function $\overline{\lambda}(w, \Psi)$ by setting $\varpi(A_{ij}) = 2^{-i}$. Specifically, define

$$\overline{\lambda}(w, A) = \begin{cases} 1 - \exp(-\lambda(w, A)) & \text{if } \lambda(w, A) < \infty \\ 1 & \text{otherwise;} \end{cases}$$

for any word w and $\overline{\lambda}(w, \Psi) = \sum_{i=0}^{n} \sum_j \overline{\lambda}(w, A_{ij})$ with $\varpi(A_{ij}) = 2^{-i}$. It is easy to see that given two words w_1, w_2 and a set of safety rules $\Psi = \{A_{ij}\}$ such that $\lambda(w_1, \Psi) \leq \lambda(w_2, \Psi)$, we also have $\overline{\lambda}(w_1, \Psi) \leq \overline{\lambda}(w_2, \Psi)$.

Let us now define a continuous version of the level of unsafety. For a continuous trajectory $x : [0, T] \to X$, if $\mathcal{D}(x) = \{t_1, t_2, \ldots, t_n\}$, then the minimizing index set in Def. 2 for $w(\rho(x))$, say $I^* = \{t_{i_1}, \ldots, t_{i_k}\}$, is a subset of $\mathcal{D}(x)$. The cost of violating a rule A is

$$\lambda_c(x, A) = \sum_{j=1}^{k} \int_{t_{i_j}}^{t_{i_j+1}} \varpi(A)\, dt \triangleq \int_0^T g_A(x(t)) dt, \quad (4)$$

where the cost function $g_A(x)$ is defined as,

$$g_A(x(t)) = \begin{cases} \varpi(A) & \text{if } t \in [t_{i_j}, t_{i_j+1}) \text{ for some } t_{i_j} \in I^* \\ 0 & \text{else.} \end{cases}$$

$g_A(x(\cdot))$ is differentiable everywhere except on I^*. We now construct the normalized continuous level of safety, call it $\overline{\lambda}_c(x, \Psi)$, in a similar fashion as $\overline{\lambda}(w, \Psi)$. Note that for a trace-inclusive Kripke structure, we have $\overline{\lambda}_c(x, \Psi) = \overline{\lambda}(\xi, \Psi)$ for any trace ξ and its corresponding trajectory x.

B. Problem statement

Sec. III-A motivates the form of cost functions for the robot and the environment. For task specifications Φ_r, Φ_e, and a set of safety rules, $\Psi = \{\Psi_1, \ldots, \Psi_n\}$, define the cost function of $\mathbf{R}$ as,

$$J_r(x_0, u_r, u_e) = \overline{\lambda}_c(x_r, \Psi) + 2^{-(n+1)}\ T_r \quad (5)$$

where $T_r = \inf\{t \in \mathbb{R}_{\geq 0} : x_r(t) \in X_{r,G}\}$ and $u_r : [0, T_r] \to U_r$. Similarly, let $T_e = \inf\{t \in \mathbb{R}_{\geq 0}\ : x_e(t) \in X_{e,G}\}$ and $u_e : [0, T_e] \to U_e$. Define the cost function of $\mathbf{E}$ to be:

$$J_e(x_0, u_r, u_e) = \overline{\lambda}_c(x_e, \Psi) + 2^{-(n+1)}\ T_e. \quad (6)$$

Note that J_r is really the normalized, scalar form of the lexicographic cost tuple $(\lambda_c(x_r, \Psi_1), \ldots, \lambda_c(x_r, \Psi_n), T_r)$, i.e., players first minimize the level of unsafety of the trajectory with respect to safety rules and then prioritize reaching the goal set as quickly as possible.

Problem 5 *Given task specifications Φ_r, Φ_e, a set of prioritized safety rules Ψ, for dynamics described by Eqn. (3), find a control strategy, $u_r^* : [0, T_r] \to U_r$ where $T_r = \inf\{t : x_r(t) \in X_{r,G}\}$, such that:*

1. *Trajectories x_r, x_e satisfy tasks Φ_r, Φ_e respectively, and*
2. *Among all trajectories x_r, x_e respectively, that satisfy 1, (u_r^*, u_e^*), where $u_e^* = BR(u_r^*)$, is the open-loop Stackelberg equilibrium of the differential game with cost functions J_r, J_e, respectively.*

Note that the necessary conditions for the OLS equilibrium are obtained by minimization of the Hamiltonian if the dynamics and running cost are differentiable [22]. In our case, even though we are guaranteed that $f \in C^1$, the running cost, shown in Eqn. (4), is discontinuous. We can however still characterize the OLS equilibrium using viscosity solutions to optimal control problems with discontinuous running cost.

Theorem 6 (Thm. 4.1 in [23]) *Consider a dynamical system given by $\dot{x}(t) = f(x(t), u(t))$ with $x(0) = x_0$ where $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ belongs to C^1 and $u(\cdot)$ belongs to the class of relaxed controls, i.e.,*

$$u \in \mathcal{U} = \{u(\cdot) : u : \mathbb{R}_{\geq 0} \to P(U) \text{ measurable}\}$$

where $P(U)$ is the set of Radon measures over the compact set $U \subset \mathbb{R}^m$. For cost functionals of the form, $J(x, t) = \int_0^t g(x(s))\, ds$ where $g(x(s))$ is bounded but possibility discontinuous on a set of measure zero, there exists a unique viscosity solution if the dynamics is small time locally controllable.

Thm. 6 can be used to show that the solution of Prob. 5 is unique if the dynamical system in Eqn. (3) is small-time locally controllable and if the initial conditions x_0 and the goal regions $X_{r,G}, X_{e,G}$ are such that optimal trajectories spend only a measure zero time at the discontinuities of the labeling function $\mathcal{L}_c$. Let us also note that under the above assumptions, cost functions in Eqn. (5) and Eqn. (6) are continuous in a neighborhood of the optimal trajectory.

IV. Algorithm

This section describes an algorithm to incrementally construct the product automaton (see Def. 4 in Sec. II-C). Ideas from sampling-based motion-planning algorithms are used to construct Kripke structures for $\mathbf{R}$ and $\mathbf{E}$, say K_r and K_e, resp. Safety rules with priorities, expressed as finite automata are used to first construct their weighted product, i.e, $\overline{A}_\Psi = (Q_\Psi, q_{0,\Psi}, \Sigma, \delta_\Psi, F_\Psi, W_\Psi)$ (see Sec. II-B.1). We then construct the weighted product automata, call them P_r, P_e, for players $\mathbf{R}, \mathbf{E}$, respectively. For convenience, let $\alpha \in \{r, e\}$. Conceptually, these product automata are used to compute traces corresponding to the Stackelberg equilibrium incrementally. The algorithm however maintains two product automata $P_{\alpha,f}$ and $P_{\alpha,b}$ for each player α, i.e., $P_\alpha = P_{\alpha,f} \cup P_{\alpha,b}$. This enables some computational benefits and helps us quickly compute best responses and corresponding costs.

Let $Q_{P_{\alpha,f}}$ and $Q_{P_{\alpha,b}}$ be the set of states of $P_{\alpha,f}$ and $P_{\alpha,b}$, resp. $Q_{P_{\alpha,f}}$ is initialized with $z_{0,\alpha} = (s_{0,\alpha}, q_{0,\Psi})$ while $Q_{P_{\alpha,b}}$ is initialized with $z_{g,\alpha} = (s_{g,\alpha}, q_{g,\alpha})$ for some $s_{g,\alpha} \in X_{\alpha,G}$ and $q_{g,\alpha} \in F_\Psi$. Each sampled vertex z_α is added to both $Q_{P_{\alpha,f}}$ and $Q_{P_{\alpha,b}}$; however, transitions are made towards z_α, i.e., $(*, z_\alpha)$ in $\delta_{P_{\alpha,f}}$ while they are made away from z_α, i.e., $(z_\alpha, *)$ in $\delta_{P_{\alpha,b}}$. Each vertex maintains two costs, $J^d_{\alpha,f}$ and $J^d_{\alpha,b}$, where $J^d_{\alpha,f}$ is the least weight of a trace reaching z_α from $z_{0,\alpha}$ while $J^d_{\alpha,b}$ is the least weight of a trace from z_α to $z_{g,\alpha}$ in $Q_{P_{\alpha,b}}$. Let $J^d_\alpha(z_\alpha) = J^d_{\alpha,f}(z_\alpha) + J^d_{\alpha,b}(z_\alpha)$. The preliminary procedures below are written for $P_{\alpha,f}$, they are analogous for $P_{\alpha,b}$.

a) Sampling: The sampling procedure $\texttt{sample}_\alpha : \mathbb{N} \to X_\alpha$ returns independent, identically distributed samples from a distribution supported over $X_\alpha \setminus X_{\alpha,obs}$.

b) Steer: Given two samples $z_1 = (s_1, q_1)$ and $z_2 = (s_2, q_2)$ in $P_{\alpha,f}$, the $\texttt{steer}_\alpha$ procedure returns the minimum cost of going from z_1 to z_2. It computes a time $T \in \mathbb{R}_{\geq 0}$ and trajectories $x_\alpha : [0, T] \to X_\alpha$, $u_\alpha : [0, T] \to U_\alpha$, such that, x_α, u_α satisfy the dynamics $\dot{x}_\alpha = f_\alpha(x_\alpha, u_\alpha)$ with $x(0) = s_1$, $x(T) = s_2$ and x_α is trace-inclusive and minimizes the cost function $J_\alpha(s_1, u_\alpha, \cdot)$. It returns $J^d_{\alpha,f}(z_1, z_2) = J_\alpha(s_1, u_\alpha, \cdot)$ and returns false if such a trajectory is infeasible or if $(q_1, q_2) \notin \delta_\Psi$. Examples of how the $\texttt{steer}_\alpha$ procedure can be constructed for certain dynamics can be found, for example, in [24], [25].

Algorithm 1: $\texttt{extend}_\alpha$

```
1  $s \leftarrow \texttt{sample}_\alpha$;
2  $Z, Z_{rewire} \leftarrow \emptyset$;
3  foreach $q \in Q_\Psi$ do
4      $z \leftarrow (s, q)$;
5      foreach $z' \in \texttt{near}_\alpha(s)$ do
6          if $\texttt{steer}_\alpha(z', z)$ then
7              $Z \leftarrow Z \cup z$;
8  foreach $z \in Z$ do
9      $Q_{P_{\alpha,f}} \leftarrow Q_{P_{\alpha,f}} \cup \{z\}$;
10     $Q_{P_{\alpha,b}} \leftarrow Q_{P_{\alpha,b}} \cup \{z\}$;
11     $\texttt{connect}_\alpha(z)$;
12     if $s \in X_{\alpha,G}$ and $q \in F_\Psi$ then
13         $F_{P_{\alpha,f}} \leftarrow F_{P_{\alpha,f}} \cup \{z\}$;
14     $\texttt{update}_\alpha(z)$;
15     $Z_{rewire} \leftarrow Z_{rewire} \cup \texttt{rewire}_\alpha(z)$;
16 return $Z, Z_{rewire}$
```

Algorithm 2: $\texttt{construct\_product}$

```
1  foreach $\alpha \in \{r, e\}$ do
2      $Q_{P_{\alpha,f}} \leftarrow \{z_{0,\alpha}\}$;
3      $Q_{P_{\alpha,b}} \leftarrow \{z_{g,\alpha}\}$;
4      $\delta_{P_\alpha} \leftarrow \emptyset$;
5  for $i \leq n$ do
6      $Z_r, Z_{r,rewire} \leftarrow \texttt{extend}_r$;
7      $Z_e, Z_{e,rewire} \leftarrow \texttt{extend}_e$;
8      $\texttt{update\_best\_response\_queue}(Z_{e,rewire})$;
9      $z_r^* \leftarrow \texttt{update\_best\_vertex}$;
10     $\texttt{delete\_collisions}(\texttt{trace}(z_r^*))$;
11 return $z_r^*, P_r, P_e$
```

c) Near vertices: For a state $s \in X_\alpha$, let $X_{\alpha,near}(s) \subset X_\alpha \cap K_\alpha$ consist of the closest $k \log n$ ($k > 2$) samples according to the cost metric J_α used in the $\texttt{steer}_\alpha$ procedure. It is thus a subset of the reachable space of the dynamical system and can be computed efficiently using the Ball-Box Theorem (see [26] for details). The $\texttt{near}_\alpha$ procedure returns:

$$\texttt{near}_\alpha(s) = \{(s', q) \ : \ s' \in X_{\alpha,near}(s), (s', q) \in Q_{P_{\alpha,f}}\}.$$

d) Connect: Given a vertex $z = (s, q)$, the $\texttt{connect}_\alpha$ procedure computes a vertex z_p such that

$$z_p = \arg \min_{z' \in \texttt{near}_\alpha(s)} J^d_{\alpha,f}(z') + \texttt{steer}_\alpha(z', z).$$

It updates the transition function as $\delta_{P_{\alpha,f}} = \delta_{P_{\alpha,f}} \setminus \{(*, z)\} \cup \{(z_p, z)\}$, i.e., it removes all transitions to z and adds the one that minimizes the cost of z. The weighing function is updated to be $W_{P_{\alpha,f}}(z_p, z) = \texttt{steer}_\alpha(z_p, z)$.

e) Update: Given $z \in Q_{P_{\alpha,f}}$, the $\texttt{update}_\alpha$ procedure updates the cost as, $J^d_{\alpha,f}(z) = J^d_{\alpha,f}(z_p) + \texttt{steer}_\alpha(z_p, z)$ for $(z_p, z) \in \delta_{P_{\alpha,f}}$.

f) Descendents: $\texttt{descendents}_\alpha(z_\alpha)$ are all vertices in $Q_{P_{\alpha,f}}$ that are reachable from z_α. $\texttt{ancestors}_\alpha(z_\alpha)$ in $Q_{P_{\alpha,b}}$ are defined similarly.

g) Rewiring: For a vertex $z = (s, q)$, if $J^d_{\alpha,f}(z') > J^d_{\alpha,f}(z) + \texttt{steer}_\alpha(z, z')$ for some z' in the set $\texttt{near}_\alpha(s)$, the $\texttt{rewire}_\alpha$ procedure executes the $\texttt{connect}_\alpha$ procedure on z'. For every such z', which requires rewiring, we call $\texttt{update}_\alpha$ for all $z'' \in \texttt{descendents}_\alpha(z')$. Let $Z_{\texttt{rewire}_\alpha}$ be the set of all the vertices which are rewired.

h) Compute trace: For $z_\alpha \in Q_{P_{\alpha,f}}$, the procedure $\texttt{trace}$ returns ξ_α which is the unique trace from $z_{0,\alpha}$ to z_α in $P_{\alpha,f}$ concatenated with the unique trace from z_α in $P_{\alpha,b}$ to $z_{g,\alpha}$. Since the Kripke structures are trace-inclusive, we can also construct the continuous trajectory $\texttt{traj}(\xi_\alpha)$ from ξ_α using the $\texttt{steer}_\alpha$ procedure.

i) Calculate best response: Given $z_r \in Q_{P_{r,f}}$ as input, the procedure $\texttt{best\_response}$ returns a vertex $\texttt{BR}(z_r) \in Q_{P_{e,f}}$ such that $\texttt{trace}(\texttt{BR}(z_r))$ is the best response of $\mathbf{E}$ if $\mathbf{R}$ executes $\texttt{trace}(z_r)$. We maintain a priority queue for the set $\{J^d_e(z_e) : \ z_e \in Q_{P_{e,f}}\}$, this enables us to search for the best response of any given $z_r \in Q_{P_{r,f}}$ efficiently.

j) Update best response: For every vertex z_e that is rewired, the cost, $J^d_{e,f}(z'_e)$ changes for all $z'_e \in \texttt{descendents}_e(z_e)$ (and $J^d_{e,b}$ changes for all $\texttt{ancestors}_e(z_e)$). The new costs, J^d_α are computed using the `update_best_response_queue` procedure.

k) Update best vertex: The algorithm incrementally maintains the best vertex z^*_r in $Q_{P_{r,f}}$ that minimizes the cost of $\mathbf{R}$. For a newly sampled vertex $z_r \in Q_{P_{r,f}}$, if $\mathbf{R}$ executes $\xi_r = \texttt{trace}(z_r)$ and $\mathbf{E}$ executes $\xi_e = \texttt{trace}(\texttt{BR}(z_r))$, this procedure evaluates the cost, i.e, $J_r(z_{0,r}, \xi_r, \xi_e)$ and updates the best vertex z^*_r if the trajectories do not collide.

l) Delete collisions: Given a trace ξ_r, this procedure removes all vertices z_e from $Q_{P_{e,f}}$ and their descendents whose trajectories collide with $\texttt{traj}(\xi_r)$ and then calls the `update_best_response_queue` procedure.

The tuple $(\xi^*_{P_r}, \xi^*_{P_e})$, where $\xi^*_{P_r} = \texttt{trace}(z^*_r)$ and $\xi^*_{P_e} = \texttt{trace}(\texttt{BR}(z^*_r))$ is therefore the open-loop Stackelberg equilibrium for the game played on discrete product automata. The projections of these traces onto individual Kripke structure are ξ^*_r, ξ^*_e respectively. Trajectories $x^*_r = \texttt{traj}(\xi^*_r)$ and $x^*_e = \texttt{traj}(\xi^*_e)$ then are the continuous-time trajectories returned by the algorithm. Alg. 2 presents the complete procedure for computing $(\xi^*_{P_r}, \xi^*_{P_e})$.

V. ANALYSIS

This section provides an analysis of the algorithm presented in Sec. IV. We first show how the two product automata differ. Roughly, the automaton $P_r = P_{r,f} \cup P_{r,b}$ of player $\mathbf{R}$ is such that $J^d_{\alpha,f}$ for every vertex is the best cost from $z_{0,r}$. On the other hand, the automaton $P_e = P_{e,f} \cup P_{e,b}$ of player $\mathbf{E}$ consists of best responses and hence the cost of a vertex $z_e \in P_e$ can be larger than its optimal cost, i.e., without considering the trajectory of $\mathbf{R}$. We omit most proofs in the interest of space. Technical arguments presented in this section are similar to those in [13], [15], [16].

A. Asymptotic optimality and Probabilistic completeness

Theorem 7 (Asymptotic optimality of P_r) *For any vertex $z_r = (s_r, q_r)$ of Q_{P_r}, let $x_r : [0,T] \to X_r$ be the optimal trajectory that minimizes J_r such that $x_r(0) = x_{0,r}$ and $x_r(T) = s_r$. Then the cost $J^d_{r,f}(z_r)$ in Alg. 2 converges to $J_r(x_r)$ in the limit almost surely, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} J^d_{r,f}(z_r) = J_r(x_r)\}\right) = 1.$$

The above theorem is an application of asymptotic optimality of the RRT* algorithm [15]. As a particular instance of the above theorem, the best vertex z^*_r returned by Alg. 2 also has the optimal cost. On the other hand, the continuity of the cost function J_r translates to $\mathbf{E}$'s Kripke structure as follows.

Lemma 8 *For $z_e = (s_e, q_e) \in Q_{P_e}$ if $x_e : [0,T] \to X_e$ is the optimal trajectory that minimizes J_e such that $x_e(0) = x_{0,e}$ and $x_e(T) = s_e$, then the cost $J^d_{e,f}(z_e)$ is at least as much as $J_e(x_e)$ almost surely, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} J^d_{e,f}(z_e) \geq J_e(x_e)\}\right) = 1.$$

The proof of this is an immediate consequence of the fact that the `delete_collisions` procedure removes transitions from P_e that collide with the current best trajectory of $\mathbf{R}$. It thus contains fewer transitions than the optimal RRT* tree.

Alg. 2 inherits probabilistic completeness from the RRT* algorithm (see Thm. 23 in [13]), i.e., it returns trajectories $x^*_{r,n}$ and $x^*_{e,n}$ such that they converge to the open-loop Stackelberg equilibrium of Prob. 5, with probability one, as the number of samples approaches infinity. In addition to this, we can also show the nature of convergence as follows.

Theorem 9 *The trajectories returned by Alg. 2 after n iterations, $x^*_{r,n}$ and $x^*_{e,n}$, converge to the solution of Prob. 5 in the bounded variation norm sense,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} \|x^*_{r,n} - x^*_r\|_{BV} = 0\}\right) = 1$$

and similarly for $x^*_{e,n}$.

Proof: The proof of this theorem is very similar to that of Thm. 16 in [16] and is hence omitted. Roughly, since the Stackelberg equilibrium exists and is unique from Thm. 6, we can show that there exists traces ξ_r of K_r and ξ_e of K_e, that lie in the neighborhood of x^*_r and x^*_e. The corresponding continuous trajectories then satisfy the claim. ■

Now, it easily follows that the cost of the trajectories returned by the algorithm converges to the optimal cost.

Corollary 10 (Asymptotic optimality) *The costs, $J_r(x^*_{r,n})$ and $J_e(x^*_{r,e})$ converge to the optimal costs in the limit with probability one, i.e.,*

$$\mathbb{P}\left(\{\lim_{n\to\infty} J_r(x^*_{r,n}) = J_r(x^*_r)\}\right) = 1,$$

and similarly for $x^*_{e,n}$.

B. Computational complexity

Let us briefly discuss the computational complexity of Alg. 2. If m is the number of states in $\overline{A}_\Psi$, $\texttt{near}_\alpha(s)$ returns $\mathcal{O}(m \log n)$ vertices in expectation. The complexity of running $\texttt{connect}_\alpha$ these vertices is $\mathcal{O}(m^2 \log^2 n)$. The $\texttt{rewire}_\alpha$ procedure considers an $\mathcal{O}(m \log n)$ neighbors and updates the cost of their descendants. It can be shown that the complexity of such an update is $\mathcal{O}(m^2 \log^2 n)$ in expectation. Similarly, we see that the `update_best_response_queue` procedure also updates $\mathcal{O}(m^2 \log^2 n)$ vertices in $Q_{P_{e,f}}$. The complexity of `update_best_vertex` is $\mathcal{O}(m \log n)$ if we maintain a priority queue of vertices in $Q_{P_{r,f}}$ with their best responses.

It can be shown that the `delete_collisions` procedure deletes descendants of nodes in an area that scales as $\mathcal{O}(\log n/n)$. Also, the height of a random node in the random Kripke structure concentrates around its expectation which is $\mathcal{O}((n^2/\log n)^{1/d})$ [27]. Thus, the `delete_collisions` procedure removes only a small number of descendants beyond this height; the number of nodes deleted per iteration in fact goes to zero in the limit as $n \to \infty$.

The total expected amortized complexity of Alg. 2 is therefore $\mathcal{O}(m^2 \log^2 n)$ per iteration.

VI. Computational Experiments

In this section, we present simulation experiments to demonstrate the proposed algorithm in a number of different scenarios motivated by autonomous urban driving.

A. Experimental setup

1) Dynamics: Consider a Dubins vehicle with control on acceleration as a model for the dynamics of both the robot and the environment with the dynamics given by, $\dot{x}(t) = v(t)\cos(\theta(t))$, $\dot{y}(t) = v(t)\sin(\theta(t))$, $\dot{v}(t) = u_1(t)$, $\dot{\theta}(t) = v(t)\,u_2(t)$, where the state is (x, y, v, θ) with bounds on acceleration, i.e., $|u_1(t)| \leq 1$ and turning rate, i.e., $|v(t)\,u_2(t)| \leq c$. Following a similar analysis as given in [24], we can see that time optimal paths between two states $(x_1, y_1, v_1, \theta_1)$ and $(x_2, y_2, v_2, \theta_2)$ for the dynamics given above can be split into two cases, (i) constrained by the shortest length Dubins curve, and, (ii) constrained by change in velocity. As an approximation, in this implementation, we only consider the first case, i.e., the $\mathtt{steer}_\alpha$ procedure returns an infinite cost if the increment in velocity is larger than what can be achieved along the shortest length Dubins curve. Note that this does not affect the completeness guarantees of Sec. V. Also note that Dubins curves for any pair (x_1, y_1, θ_1) and (x_2, y_2, θ_2) are composed of straight lines and maximum turning rate segments and can be computed efficiently [25].

2) Safety Rules: X_r, X_e are partitioned into compact non-empty subsets X_{sw}, X_{ll}, X_{rl} for the sidewalk, left-lane and right-lane respectively. Atomic propositions are given by $\Pi = \{sw, rl, ll, dir, slow\}$. For a state s_α, sw, rl, ll are true iff $s_\alpha \in X_{sw}, X_{ll}$ and X_{rl}, respectively. dir is true iff α is traveling in the right direction, i.e., forward in the right-lane (which is computed using geometry of the road). $slow$ is true iff s_α has a velocity smaller some fixed constant V_{nom}. We consider the following road-safety rules :

- Do not take a transition that leads to a sidewalk:
$$\psi_1 = \mathsf{G}\neg(\mathsf{true}, sw)$$
- Always travel in the correct direction:
$$\psi_{2,1} = \mathsf{G}(\mathsf{true}, dir)$$
- Do not change lanes:
$$\psi_{2,2} = \mathsf{G}\neg((rl, ll) \vee (ll, rl))$$
- Motivated by the fact that drivers on road would not like to slow down for each other, we include a rule which encourages drivers to maintain the speed above V_{nom}.
$$\psi_3 = \mathsf{G}\neg(\mathsf{true}, slow)$$

The priorities are set as $\varpi(\psi_1) = 1$, $\varpi(\psi_{2,1}) = \varpi(\psi_{2,2}) = 2$ and $\varpi(\psi_3) = 3$. We use 2-tuples of Π to denote transitions between states. The two components capture atomic propositions of the starting and ending states, respectively. The automaton shown in Fig. 1 is the same for all these rules.

B. Examples

For the following examples, the algorithm was implemented in C++ on a 2.2 GHz processor with 4 GB of memory

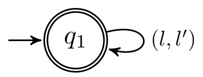
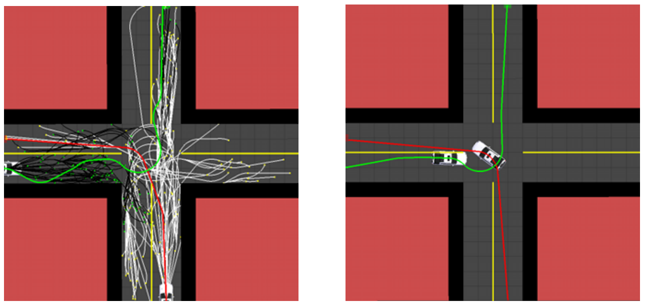


Fig. 1: Finite automaton for safety rules: E.g., $l, l' \in 2^\Pi$, $sw \notin l'$ gives ψ_1, $dir \in l'$ gives $\psi_{2,1}$, while $rl \notin l, ll \notin l'$ or $ll \notin l, rl \notin l'$ gives $\psi_{2,2}$ while $slow \notin l'$ gives ψ_3.

in the Linux environment. In Figs. 2 and 3, the Kripke structure maintained by $\mathbf{R}$ is shown in white while the Kripke structure maintained by $\mathbf{E}$, i.e., K_e is shown in black. The current trajectory for $\mathbf{R}$ returned by Alg. 2 is plotted in red while the best response to this trajectory is plotted using green. Stationary obstacles are shown in red, while X_{sw} is shown in black. Right and left lanes are demarcated using yellow lines. Goal regions for both players, $X_{r,G}, X_{e,G}$ are shown in red and green respectively. Both players start with the same initial velocity V_{nom}.

1) Case 1: We first consider a scenario with both $\mathbf{R}$ and $\mathbf{E}$ at a cross-road junction to demonstrate the incremental nature of the algorithm. Fig. 2a shows the Kripke structures and the solution after 400 samples. It returns paths with cost 2.4231 for $\mathbf{R}$ which is close to optimal, but in order to satisfy the task specification without colliding with $\mathbf{R}$, player $\mathbf{E}$ breaks the slow driving rule and incurs a cost of 6.819. Fig. 2b shows this path after 1000 samples.

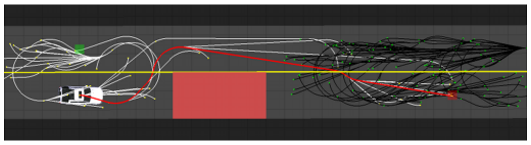


(a) (b)

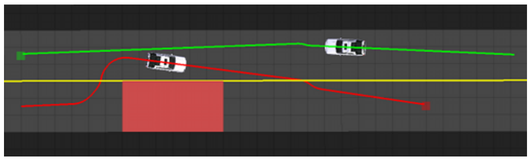
Fig. 2: Fig. 2a sub-optimal trajectories with sparse Kripke structures. As shown in Fig. 2b, with more samples, the algorithm converges to a trajectory for $\mathbf{E}$ that has a much smaller cost. Both cars are drawn at the same instant in time.

2) Case 2: In this example, we consider a single-lane road with an obstacle in the lane of $\mathbf{R}$. Fig. 3a shows the algorithm after 500 samples where the environment cannot find any trajectory that reaches its goal without colliding with the trajectory that $\mathbf{R}$ would like to execute. With additional computation time, as shown in Fig. 3b, $\mathbf{E}$ obtains a trajectory that has a cost of 3.65 without breaking any safety rules. On the other hand, $\mathbf{R}$ incurs a penalty for breaking the lane changing rule and obtains a best cost of 13.95.

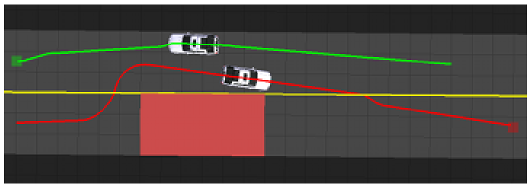
3) Case 3: Consider a slight modification of Case 2 where the environment starts closer to the obstacle than before. As shown in Fig. 3c, $\mathbf{R}$ now forces the Stackelberg equilibrium upon player $\mathbf{E}$ and by still choosing the same trajectory. In order to avoid a collision, i.e., generate a valid best response that still satisfies the task specification, $\mathbf{E}$ slows down and incurs a cost for breaking rule ψ_3 to let $\mathbf{R}$ cross across the obstacle.



(a)



(b)



(c)

Fig. 3: Fig. 3a shows the Kripke structures when player $\mathbf{E}$ which starts at the root of the black tree does not have any trajectory to reach the goal region. The algorithm converges to the trajectories shown in Fig. 3b. On the other hand, as shown in FIg. 3c, if $\mathbf{E}$ starts closer to the obstacle, it is forced to obey the Stackelberg equilibrium and has to slow down to let $\mathbf{R}$ cross the obstacle.

VII. Conclusions

This paper addresses the problem of computing open-loop equilibria for two-player, non-zero sum differential games where players accomplish a task specification while satisfying safety rules expressed using temporal logic. Ideas from sampling-based motion-planning algorithms and model checking literature are used to incrementally construct weighted product automata where the weight of a trace is representative of its level of unsafety with respect to the safety rules. We propose an algorithm to compute the Stackelberg equilibrium and provably converge to the continuous time equilibrium as the number of samples in the Kripke structures go to infinity. The algorithm is demonstrated on a number of examples motivated from autonomous urban mobility on demand systems.

Directions for future work consist of computing feedback equilibria and considering safety rules that depend on both players to enable a richer class of behaviors.

References

[1] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[2] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

[3] Rufus Isaacs. *Differential games: A mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Dover Publications, 1999.

[4] Tamer Basar and Geert Jan Olsder. *Dynamic noncooperative game theory*, volume 200. SIAM, 1995.

[5] Ian M Mitchell, Alexandre M Bayen, and Claire J Tomlin. A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Trans. on Automatic Control*, 50(7):947–957, 2005.

[6] Leonidas J Guibas, Jean-Claude Latombe, Steven M LaValle, David Lin, and Rajeev Motwani. A visibility-based pursuit-evasion problem. *Int. Journal of Computational Geometry & Applications*, 9(04n05):471–493, 1999.

[7] Tuomas Raivio. Capture set computation of an optimally guided missile. *Journal of Guidance, Control, and Dynamics*, 24(6):1167–1175, 2001.

[8] MH Breitner, HJ Pesch, and W Grimm. Complex differential games of pursuit-evasion type with state constraints, Part 2: Numerical computation of optimal open-loop strategies. *Journal of Optimization Theory and Applications*, 78(3):443–463, 1993.

[9] James Albert Sethian. *Level set methods and fast marching methods*, volume 3. Cambridge university press, 1999.

[10] Xu Chu Ding, Stephen L Smith, Calin Belta, and Daniela Rus. MDP optimal control under temporal logic constraints. In *Proc. of IEEE Conf. on Decision and Control and European Control Conference*, pages 532–538, 2011.

[11] Paulo Tabuada and George J Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[12] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon control for temporal logic specifications. In *Proc. of ACM Int. conf. on Hybrid Systems: Computation and Control*, pages 101–110, 2010.

[13] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for a class of pursuit-evasion games. In *Algorithmic Foundations of Robotics IX*, pages 71–87. Springer, 2011.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *American Control Conference*, pages 735–742, 2012.

[15] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.

[16] Luis I Reyes Castro, Pratik Chaudhari, Jana Tumova, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Incremental sampling-based algorithm for minimum-violation motion planning. In *Proc. of IEEE International Conference on Decision and Control (CDC)*, 2013.

[17] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. On Model Checking Durational Kripke Structures. In *Foundations of Software Science and Computation Structures*, pages 264–279. Springer, 2002.

[18] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, 3rd edition, 2012.

[19] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[20] Elsa L. Gunter and Doron Peled. Temporal debugging for concurrent systems. In *Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 431–444. Springer-Verlag, 2002.

[21] Jana Tumova, Gavin C Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Proc. of conference on Hybrid Systems: Computation and Control*, pages 1–10, 2013.

[22] CL Chen and J Cruz Jr. Stackelburg solution for two-person games with biased information patterns. *IEEE Transactions on Automatic Control*, 17(6):791–798, 1972.

[23] Pierpaolo Soravia. Optimal control with discontinuous running cost: eikonal equation and shape-from-shading. In *Proc. of IEEE Conf. on Decision and Control*, volume 1, pages 79–84, 2000.

[24] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In *Proc. of IEEE Conf. on Decision and Control*, pages 2379–2384, 2007.

[25] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[26] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Proc. of IEEE Conf. on Robotics and Automation*, 2013.

[27] Tobias Friedrich, Thomas Sauerwald, and Alexandre Stauffer. Diameter and broadcast time of random geometric graphs in arbitrary dimensions. In *Algorithms and Computation*, pages 190–199. Springer, 2011.