Abstract—Automatic generation of control programs that satisfy complex task specifications given in high-level specification languages such as temporal logics has been studied extensively. However, optimality of such control programs, for instance with respect to a cost function, has received relatively little attention. In this paper, we study the problem of optimal trajectory synthesis for a large class of specifications, given in the form of deterministic $\mu$-calculus. We propose a sampling-based algorithm, based on the Rapidly-exploring Random Graphs (RRGs), that solves this problem with probabilistic completeness and asymptotic optimality guarantees. We evaluate our algorithm in a simulation studies involving a curvature constrained car. Our simulation results show that in this scenario the algorithm quickly discovers a trajectory that satisfies the specification, and improves this trajectory towards an optimal one if allowed more computation time. We also point out connections to (optimal) memoryless winning strategies in infinite parity games, which may be of independent interest.

I. INTRODUCTION

Automatic design of control programs from their temporal and logic specifications have received significant attention in the recent years. Following their applications in formally and naturally describing computer hardware and software [1], temporal logics such as the well-known Linear Temporal Logic (LTL) has been used for specifying desired properties of control systems, and several algorithmic approaches for designing control systems from their temporal logic specifications have been proposed [2]–[6].

It is known that even finding a feasible solution that fulfills a given LTL specification even for a discrete-state system is computationally challenging [7]. Thus, most of the algorithmic approaches available in the control theory literature focus solely on finding a feasible solution, often completely disregarding the quality of the solution returned.

Very recently, the study of the quality of the solution returned by algorithms tailored for control design has attracted significant attention in two different communities. On one hand, computer scientists have studied infinite games with both qualitative and quantitative objectives. For instance, infinite games involving, for example, a mean-payoff cost structure have been studied extensively [8], and games in which a optimal strategies can be memoryless (i.e., independent of the history) have been characterized [9]. On the other hand, roboticists have proposed motion planning algorithms that allow efficiently computing optimal dynamically-feasible trajectories for complex robotic systems [10]. In particular, algorithms such as the PRM$^*$ and the RRT$^*$ has been proposed as asymptotically-optimal and computationally-efficient counterparts of the Probabilistic RoadMap (PRM) and the Rapidly-exploring Random Tree (RRT), which lacked either asymptotic optimality or computational efficiency [10]. However, so far the computer science literature has focused solely on discrete-state systems, while the robotics literature has not yet taken into account complex quantitative objectives such as those that can be described by temporal logics.

This paper fills this gap, hence bringing together these two communities, by proposing an asymptotically-optimal incremental sampling-based motion planning algorithm that can deal with complex tasks specifications given in the form of deterministic $\mu$-calculus. It is worth noting at this point that our work in this paper is closely related to our previous work in [11], where we propose a sampling-based algorithm that extends the Rapidly-exploring Random Tree (RRT) algorithm to handle complex task specifications given in the form of deterministic $\mu$-calculus, however with no guarantees on optimality.

The contributions of this paper can be listed as follows. First, we definite the problem of optimal motion planning with deterministic $\mu$-calculus specifications, making connections to the theory of infinite parity games. Second, based on this theory, we provide a novel incremental (local) model checking algorithm for the deterministic fragment of $\mu$-calculus that is based on the incremental construction of a sequence of infinite parity games. Third, we propose an incremental sampling based algorithm with the asymptotic optimality guarantee, i.e., almost-sure convergence to optimal solutions. We show that in general the (per-iteration) expected amortized asymptotic computational complexity of the algorithm is order $\log^2 n$, that is a $\log n$ factor of the RRT$^*$, an asymptotically-optimal algorithm for the classical motion planning problem of “reaching the goal while avoiding the obstacle.” However, in the case of simple specifications of this form, a more thorough analysis shows that the proposed algorithm recovers the computational complexity of the RRT$^*$.

The paper is organized as follows. In Section II, dynamical systems, Kripke structures, deterministic $\mu$-calculus, and infinite parity games are introduced. Section III is devoted to the problem definition. The algorithms are described in Section IV, and their analysis in terms of completeness, optimality, and computational complexity is presented in Section V. A computational example is presented in Section VI. The paper is concluded with remarks in Section VII.

The authors with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.
II. PRELIMINARIES

A. Dynamical Systems and Kripke Structure Models

Let $X \subseteq \mathbb{R}^d$ and $U \subseteq \mathbb{R}^m$ be compact sets, and consider

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = z_{\text{init}},$$

where $x : \mathbb{R}_{\geq 0} \to X$ is the state trajectory, $u : \mathbb{R}_{\geq 0} \to U$ is the input signal, and $z_{\text{init}}$ is the initial state. We tacitly assume throughout the paper that $f$ is Lipschitz continuous in both of its variables and $u$ is Lebesgue measurable.

Let $\Pi$ be a finite set of atomic propositions, and let $\lambda : X \to 2^\Pi$ be a mapping that associates each state of the dynamical system with a set of atomic propositions that it satisfies. Given a state trajectory $x$, define $T_x := \{ t \in \mathbb{R}_{\geq 0} : \lambda(x(t)) \neq \lambda(x(t)) \}$, i.e., the set of all time instances at which $\lambda(x(t))$ changes its value. An infinite sequence $w_x = (w_0, w_1, \ldots)$ is said to be a word generated by $x$, if there exists a sequence $(t_1, t_2, \ldots)$, where $t_0 = 0$ and $t_i < t_{i+1}$ for all $i \in \mathbb{N}$, such that (i) $\lambda(x(t_i)) = w_i$ for all $i \in \mathbb{N}$, (ii) $\lambda(x(\tau)) = \lambda(x(t_i))$ for all $\tau$ in an open neighborhood of $t_i$ for all $i \in \mathbb{N} \setminus \{0\}$, and (iii) there exists at most one transition instance between two consecutive time instances, $t_i$ and $t_{i+1}$, i.e., $|T_x \cap (t_i, t_{i+1})| \leq 1$ for all $i \in \mathbb{N}$.

A widely-used (discrete) model of computer software in the context of model checking is the following.

**Definition 1 (Kripke Structure)** A Kripke structure $K$ defined on a set $\Pi$ of atomic propositions is a tuple $K = (S, s_{\text{init}}, R, L)$, where $S$ is a finite set of states, $s_{\text{init}} \in S$ is the initial state, $R \subseteq S \times S$ is a transition relation, and $L : S \to 2^\Pi$ is a labeling function.

A finite sequence $\sigma = (s_0, s_1, \ldots, s_k)$ is said to be a finite trace of $K$, if $s_0 = s_{\text{init}}$ and $(s_{i-1}, s_i) \in R$ for all $i \in \{1, 2, \ldots, k\}$. An infinite sequence $\sigma = (s_0, s_1, \ldots)$ is said to be an $\omega$-trace of $K$, if every finite prefix of $\sigma$ is a finite trace of $K$. Any $\omega$-trace $\sigma = (s_0, s_1, \ldots)$ induces a unique word, defined by $(L(s_0), L(s_1), \ldots)$, and denoted by $L(\sigma)$ with a slight abuse of notation.

The following definition provides a finite model for a dynamical system in terms of a Kripke structure.

**Definition 2 (\omega-trace-inclusive Model)** A Kripke structure $K = (S, s_{\text{init}}, R, L)$ is said to be an $\omega$-trace-inclusive model of a system of the form (1), if (i) $S \subseteq X$; (ii) $s_{\text{init}} = z_{\text{init}}$; (iii) for all $(s_1, s_2) \in R$, there exists a trajectory $x : [0, T] \to X$ satisfying Equation (1) such that $x(0) = s_1$, $x(T) = s_2$, and $\lambda(x(t))$ changes value at most once as $t$ varies from 0 to $T$, i.e., Card$\{t \in [0, T] : \lim_{t \to T} \lambda(x(t)) \neq \lambda(x(t))\} \leq 1$; and (iv) $L(s) = \lambda(s)$ for all $s \in S$.

The following proposition, asserting an important property of the previous definition, follows easily from the definitions.

**Proposition 3** Let $K$ be an $\omega$-trace-inclusive model of the system given by Equation (1). Then, for any $\omega$-trace $\sigma$ of $K$, there exists a trajectory $x : [0, \infty) \to X$ of the dynamical system and a word $w_x$ generated by $x$ such that $L(\sigma) = w_x$.

B. Deterministic $\mu$-calculus

The semantics of the $\mu$-calculus is given below.

**Definition 4 ($\mu$-calculus)** Let $\text{Var}$ be a finite set of variables and $\Pi$ be a finite set of atomic propositions. Then, the syntax of $\mu$-calculus is given in the Backus-Naur Form (BNF) as

$$\phi ::= p | \neg p | x \land \phi | x \lor \phi | \lozenge \phi | \square \phi | \mu x. \phi | \nu x. \phi,$$

where $p \in \text{Var}$ and $\phi \in \Pi$.

The set of all $\mu$-calculus formulas is denoted by $L_{\mu}$ for short. Let $K = (S, s_{\text{init}}, R, L)$ be a Kripke structure defined on a set $\Pi$ of atomic propositions. Let $[\phi]_K$ denote the set of all states (in $S$) that satisfy the $\mu$-calculus formula $\phi$. Given a subset $Q \subseteq S$ of states, and a variable $x \in \text{Var}$, define $K^Q_{\mu} = (S, s_{\text{init}}, R, L^\prime)$ defined on an augmented set $\Pi \cup \{x\}$ of atomic propositions, where

$$L^\prime(s) = \begin{cases} L(s) \cup \{x\} & \text{for all } s \in Q; \\ L(s) & \text{for all } s \notin Q. \end{cases}$$

Then, the semantics of the $\mu$-calculus is defined as follows.

**Definition 5 (Semantics of $\mu$-calculus)** Given $p \in \Pi$, $x \in \text{Var}$, $\phi, \psi \in L_{\mu}$, and a Kripke structure $K = (S, s_{\text{init}}, R, L)$, the set $[\phi]_K$ is recursively defined as follows:

- $[p]_K = \{ s \in S | p \in L(s) \}$, for all $p \in \Pi$,
- $[\neg p]_K = \{ s \in S | p \notin L(s) \}$, for all $p \in \Pi$,
- $[p \land \psi]_K = [p]_K \cap [\psi]_K$,
- $[p \lor \psi]_K = [p]_K \cup [\psi]_K$,
- $[\square \phi]_K = \{ s \in S : \text{there exists } s' \in S \text{ such that } (s, s') \in R \text{ and } s' \in [\phi]_K \}$,
- $[\diamond \phi]_K = \{ s \in S : \text{for all } s' \in S \text{ such that } (s, s') \in R \text{ implies } s' \in [\phi]_K \}$,
- $[\mu x. \phi]_K$ is the least set $Q$ such that $Q = [\phi]_{[\mu x. \phi]_K}$, i.e., $[\mu x. \phi]_K$ is such that $[\mu x. \phi]_K = [\phi]_{[\mu x. \phi]_K}$ and $\forall Q' \subseteq S. \{ Q' = [\phi]_{[\mu x. \phi]_K} \Rightarrow [\mu x. \phi]_K \subseteq Q' \}$,
- $[\nu x. \phi]_K$ is the greatest set $Q$ such that $Q = [\phi]_{[\nu x. \phi]_K}$, i.e., $[\nu x. \phi]_K$ is such that $[\nu x. \phi]_K = [\phi]_{[\nu x. \phi]_K}$ and $\forall Q' \subseteq S. \{ Q' = [\phi]_{[\nu x. \phi]_K} \Rightarrow Q' \subseteq [\nu x. \phi]_K \}$.

Deterministic $\mu$-calculus, also denoted by $L_1$, is a subset of the $\mu$-calculus. The syntax of $L_1$ is given below; its semantics follows from Definition 5.

**Definition 6 (Deterministic $\mu$-calculus)** Let $\text{Var}$ be a finite set of variables and $\Pi$ be a finite set of atomic propositions. The syntax of deterministic $\mu$-calculus is given in BNF as

$$\phi ::= p | \neg p | x \land \phi | x \lor \phi | \lozenge \phi | \square \phi | \mu x. \phi | \nu x. \phi,$$

where $p \in \Pi$ and $x \in \text{Var}$.

The deterministic fragment of $\mu$-calculus is a rich language in terms of its expressive power. For instance, the widely-used Linear Temporal Logic (LTL) is a strict subset of the deterministic $\mu$-calculus. In fact, it is equivalent to all $\omega$-regular properties. Some example specifications given of deterministic $\mu$-calculus specifications can be found in [11].
C. Infinite Parity Games

Let $V$ be a finite set of vertices and $V_1, V_2$ be a partition of $V$. An arena is a tuple $A = (V_1, V_2, E)$, where $V_1$ and $V_2$ are the vertices of Players 1 and 2, respectively, and $E \subseteq V \times V$ is the edge relation. A coloring function $\chi : V \rightarrow \mathbb{N}$ is a mapping that associates each vertex in $V$ with a natural number. In an infinite game played on a finite graph, the players start by placing a token on one of the vertices. The game proceeds in stages. At each stage, if the token is on a vertex in $V_1$ (respectively $V_2$), then Player 1 (respectively Player 2) gets to push it along an edge of the graph to another vertex. The game is repeated for infinitely many stages or until the token ends up in a vertex with no outgoing edges. A finite play of an infinite game is a finite sequence $\pi = (v_0, v_1, \ldots, v_k)$, where $v_i$ is the location of the token at stage $i$ and $v_k$ is a vertex with no outgoing edges, i.e., $\{v \in V : (v_k, v) \in E\} = \emptyset$. A infinite play of the game is an $\omega$-sequence $\pi = (v_0, v_1, \ldots) \in V^\omega$. Clearly, an infinite play is an $\omega$-path on $(V, E)$. The winning condition is described by the maximum color that appears on the play infinitely often. More precisely, Player 1 (respectively Player 2) is said to win the play if either one of the following holds: (i) the play $\sigma$ is an infinite play in which the maximum color that is repeated infinitely often, i.e., $\max\{\chi(s) : s \in \text{Inf}(\sigma)\}$, is even (respectively odd), where $\text{Inf}(\sigma)$ denotes the set of all states that appear infinitely often in $\sigma$, or (ii) the play $\sigma$ is a finite play in which Player 2 (respectively Player 1) gets stuck, i.e., cannot move the token any further.

A memoryless strategy for player $k \in \{1, 2\}$ is a function $h_k : V_k \rightarrow V$ such that $(v, h_k(v)) \in E$ for all $v \in V_k$. If Player 1 plays according to a strategy $h_k$, then she chooses the move to vertex $h_k(v)$ whenever the token is in vertex $v \in V_k$. A play $\pi = (v_0, v_1, \ldots)$ is said to be consistent with strategy $h_k$, if $v_0 \in V_k$ implies that $v_{i+1} = h_k(v_i)$ for all $i \in \omega$. A strategy $h_k$ is said to be a winning strategy for Player $k$, if she wins all the plays consistent with $h_k$. An important result in the theory of infinite games states that parity games are memorylessly determined, i.e., exactly one of the players has a memoryless winning strategy [12].

D. Infinite $L_\mu$ Games

Let $K = (\mathcal{S}, s_0, \mathcal{R}, \mathcal{L})$ be a Kripke structure and $\phi$ be an $L_1$ formula. A formula $\psi$ is said to be a sub-formula in $\psi$ if it satisfies the $L_1$ syntax $\phi$ contains $\psi$ as a sub-string. This induces a natural partial ordering of all the $L_1$ formulas. If $\psi$ is a sub-formula in $\phi$, then we write $\psi \leq_{\text{SF}} \phi$ to denote this ordering. The set of all sub-formulas in $\phi$ is denoted by $\text{SF}_\phi$, i.e., $\text{SF}_\phi = \{\psi \in L_1 : \psi \leq_{\text{SF}} \phi\}$. The set of free variables in an $L_1$ formula, denoted by $\text{FV}(\phi)$, is defined recursively as follows: (i) $\text{FV}(p) = \text{FV}(\neg p) = \emptyset$ for all $p \in \Pi$, (ii) $\text{FV}(x) = \{x\}$ for all $x \in \text{Var}$, (iii) $\text{FV}(\psi_1 \land \psi_2) = \text{FV}(\psi_1 \lor \psi_2) = \text{FV}(\psi_1) \cup \text{FV}(\psi_2)$, (iv) $\text{FV}(\bigotimes(x)) = \text{FV}(\bigotimes(x)) = \text{FV}(\psi)$, and (v) $\text{FV}(\mu x. \psi) = \text{FV}(\nu x. \psi) = \text{FV}(\psi) \setminus \{x\}$. A variable $x$ is said to be bound variable in $\phi$ if $x$ is a sub-formula of $\phi$ but it is not a free variable in $\phi$. A formula $\phi$ is said to be in normal form if every bound variable is bound only once, i.e., for all $x \in \text{Var}$ with $x \leq_{\text{SF}} \phi$ there exists a unique sub-formula $\psi \leq_{\text{SF}} \phi$ of the form either $\psi = \mu x. \varphi$ or $\psi = \nu x. \varphi$. This unique sub-formula of $\phi$ is called the binding formula of $x$ in $\phi$, denoted by $\text{BF}_\phi(x)$. Each $L_1$ formula $\phi$ is associated with an alternation depth, denoted by $\text{AD}(\phi)$, defined recursively as follows: (i) $\text{AD}(p) = \text{AD}(\neg p) = 0$, (ii) $\text{AD}(\psi_1 \land \psi_2) = \max\{\text{AD}(\psi_1), \text{AD}(\psi_2)\}$, (iii) $\text{AD}(\bigotimes(x)) = \text{AD}(\psi)$, (iv) $\text{AD}(\mu x. \psi) = \max\{1, \text{AD}(\psi)\} \cup \{\text{AD}(\mu x. \psi') + 1 : \psi' \leq_{\text{SF}} \psi, x \in \text{FV}(\psi')\}$, and (v) $\text{AD}(\nu x. \psi) = \max\{1, \text{AD}(\psi)\} \cup \{\text{AD}(\nu x. \psi') + 1 : \psi' \leq_{\text{SF}} \psi, x \in \text{FV}(\psi')\}$. Essentially, the alternation depth of a $\mu$-calculus formula is the number of alternations of different fixed-point operators.

An (infinite) $L_\mu$ game of a Kripke structure $K = (\mathcal{S}, s_0, \mathcal{R}, \mathcal{L})$ and an $L_\mu$ formula $\phi$ is a particular instance of an infinite parity game with vertex set $V = S \times \text{SF}_\phi$.

**Definition 7** ($L_\mu$ Game [12]) Let $K = (\mathcal{S}, s_0, \mathcal{R}, \mathcal{L})$ be a Kripke structure and $\phi$ be an $L_\mu$ formula. The $L_\mu$ game of $K$ and $\phi$ is defined as the infinite parity game with arena $(V_1, V_2, E)$, set $C = \{0, 1, \ldots, \text{AD}(\phi)\}$ of colors, and coloring function $\chi : V \rightarrow C$ as follows. For all $s \in S$,

1. $(s, p) \in V_1$ for all $p \in \Pi$ with $p \notin \mathcal{L}(s)$,
2. $(s, \neg p) \in V_1$ for all $p \in \Pi$ with $p \notin \mathcal{L}(s)$,
3. $(s, x) \in V_1$ and $((s, x), (s, \text{BF}_\phi(x))) \in E$, for all $x \in \text{Var}$,
4. $(s, \psi \land \psi') \in V_1$ and $((s, \psi \land \psi'), (s, \psi')) \in E$,
5. $(s, \bigotimes(x)) \in V_1$ and $((s, \bigotimes(x)), (s', \psi)) \in E$ for all $s' \in S$ with $(s, s') \in \mathcal{R}$,
6. $(s, \mu x. \psi) \in V_1$ and $((s, \mu x. \psi), (s', \psi)) \in E$ for all $s' \in S$ with $(s, \psi) \in \mathcal{R}$,
7. $(s, \nu x. \psi) \in V_1$ and $((s, \nu x. \psi), (s', \psi)) \in E$ for all $s' \in S$ with $(s, \psi) \in \mathcal{R}$,
8. $(s, p) \in V_2$ for all $p \in \Pi$ with $p \notin \mathcal{L}(s)$,
9. $(s, \neg p) \in V_2$ for all $p \in \Pi$ with $p \notin \mathcal{L}(s)$,
10. $(s, \psi \land \psi') \in V_2$ and $((s, \psi \land \psi'), (s, \psi')) \in E$ for all $p \in \Pi$,
11. $(s, \bigotimes(x)) \in V_2$ and $((s, \bigotimes(x)), (s', \psi)) \in E$ for all $s' \in S$.

The coloring function satisfies: for all $s \in S$ and $\psi \leq_{\text{SF}} \phi$,

- if $\psi$ is of the form $\psi = \mu x. \psi'$, then $\chi(\psi)$ is equal to the smallest odd number greater than or equal to $\text{AD}(\psi)$,
- if $\psi$ is of the form $\psi = \nu x. \psi'$, then $\chi(\psi)$ is equal to the smallest even number greater than or equal to $\text{AD}(\psi)$,
- otherwise, $\chi(\psi) = 0$.

Clearly, all $L_1$ games are memorylessly determined, since all infinite parity games are. Moreover, an important result states that the existence of a winning (memoryless) strategy for Player 1 in the $L_\mu$ game of $K$ and $\phi$ implies the satisfaction of the $\phi$ on the initial state of $K$.

**Theorem 8** Let $K = (\mathcal{S}, s_0, \mathcal{R}, \mathcal{E})$ be a Kripke structure and $\phi$ be an $L_\mu$ formula. Then, $\phi$ is satisfied at $s_0$ of $K$, i.e., $s_0 \models [0]_{\phi,K}$, if and only if Player 1 has a winning strategy in the $L_\mu$ game of $K$ and $\phi$.

Due to lack of space, we omit the proof, which can be deduced from the results presented in [12].
E. Infinite $L_1$ Games

The $L_1$ game of $K$ and an $L_1$ formula $\phi$ is defined as in Definition 7, and Theorem 8 holds also for the $L_1$ game. Note that in the $L_1$ game, Player 2 does not have vertices of the form $(s, i\psi)$, since the $\Box$ operator is not included in $L_1$. Moreover, Player 2 vertices of the form $(s, \psi \land \psi')$ are such that either $\psi = p$ or $\psi = \neg p$ for some $p \in \Pi$, again due to the semantics of $L_1$.

$L_1$ games are important for the purposes of this paper for the following two reasons. Firstly, $L_1$ games allow a mathematically-rigorous definition of a large class of motion planning problems involving complex task specifications given in $L_1$, which is discussed in the next section. Second, the memoryless determinacy of $L_1$ games (together with the simple structure of the vertex set of Player 2) lead to computationally-efficient incremental model-checking algorithms, which will be introduced in Section IV.

III. PROBLEM DEFINITION

In this section, the optimal motion planning problem with deterministic $\mu$-calculus specifications is introduced. First, let us note the following definitions. Let $K = (S, s_{init}, \mathcal{R}, L)$ be an $\omega$-trace-inclusive Kripke structure model of the form (1). Given $(s_1, s_2) \in \mathcal{R}$, let Trajectory$(s_1, s_2)$ denote the pair $(x, T)$, where $x : [0, T] \to X$ is the trajectory satisfying the properties given in Definition 2.

Definition 9 (Witness sequence and witness trajectory)

Let $K = (S, s_{init}, \mathcal{R}, L)$ be an $\omega$-trace-inclusive Kripke structure model of a dynamical system of the form (1) and $\phi$ be an $L_1$ formula such that $s_{init} \in \llbracket \phi \rrbracket_K$. Consider the $L_1$ game of $K$ and $\phi$ with arena $(V_1, V_2, E)$. Let $h : V_1 \to V_1 \cup V_2$ be a winning strategy for Player 1.

Then, the witness sequence for $K$ and $h$ is an $\omega$-sequence $\{s_0, \psi_0\}, \{s_1, \psi_1\}, \ldots$ such that $\psi_0 = \phi$ and for all $i \in \omega$,

- if $(s_i, \psi_i) \in V_1$, then $(s_{i+1}, \psi_{i+1}) = h((s_i, \psi_i))$,
- if $(s_i, \psi_i) \in V_2$ and $\psi_i \in \{\psi \land p, \psi \land \neg p\}$ for some $p \in \Pi$, then $(s_{i+1}, \psi_{i+1}) = (s_i, \psi')$,
- if $(s_i, \psi_i) \in V_2$ and $\psi_i \in \{\neg p\}$ for some $p \in \Pi$, then $(s_{i+1}, \psi_{i+1}) = (s_i, \psi_i)$.

The witness trajectory for $K$ and $h$ is denoted by $\bar{x} : [0, \infty) \to X$ and computed as follows: for all $i \in \mathbb{N}$,

- if $s_{i+1} = s_i$, then $t_{i+1} = t_i$,
- if $s_{i+1} \neq s_i$, then $t_{i+1} = t_i + T$ and $\bar{x}(\tau) = x(\tau - t_i)$ for all $\tau \in [0, T]$, where $(x, T) = \text{Trajectory}_K(s_i, s_{i+1})$.

Let $c$ be a cost function that maps each trajectory $x : [0, \infty) \to X$ of the dynamical system described by Equation (1) to a non-negative cost denoted by $c(x)$.

Problem 10 Given a dynamical system of the form given in Equation (1), an $L_1$ formula $\phi_{\text{spec}}$, and a cost function $c$, find an $\omega$-trace-inclusive Kripke structure model $K = (S, s_{init}, \mathcal{R}, L)$ of (1) and (a memoryless) winning strategy $h$ for Player 1 in the $L_1$ game of $K$ and $\phi_{\text{spec}}$ such that $s_{init} \in \llbracket \phi \rrbracket_K$ and the cost $c(x)$ of the witness trajectory $x$ for $K$ and $h$ is minimized.

In the rest of the paper, restrict our attention to cost functions of the form $c(x) = \int_0^{T_{fin}} g(x_{inf}(t)) dt + \int_0^{T_{fin}} f(x_{inf}(t)) dt$, where $x_{inf} : [0, T_{fin}] \to X$ is the segment of $x$ that is repeated infinitely often, $x_{inf} : [0, T_{fin}] \to X$ is the prefix of $x$ that starts before the first time $x_{inf}$ is observed, and $g : X \to \mathbb{R}_{\geq 0}$ is a continuous function. The algorithms presented in this paper apply to a much wider class of cost functions, including for example the discounted cost, although we consider the one described above for simplicity.

IV. ALGORITHMS

This section describes an incremental model checking algorithm for $L_1$ and a sampling-based algorithm that solves Problem 10 with probabilistic guarantees.

A. Incremental Local Model Checking for $L_1$

The incremental model checking algorithm answers the local model checking query, i.e., whether $s_{init} \in \llbracket \phi_{\text{spec}} \rrbracket_K$, for a Kripke structure $K = (S, s_{init}, \mathcal{R}, L)$, which is constructed and fed to the algorithm incrementally. A similar incremental model checking algorithm was presented in [11]. The version presented in this paper makes more concrete connections with infinite parity games. Indeed, the algorithm maintains a subset of the arena of the $L_1$ game of $K$ and $\phi_{\text{spec}}$. Loosely speaking, this subset contains the set of all those vertices that Player 1 can move the token to without losing the game by getting stuck.

The algorithm also maintains four global variables, denoted by $RV, Pr, Cost, nV$, and $1V$. Their roles in the algorithm can be described as follows. The variable $RV$, defined for all vertices $(s, \psi) \in V$, is a set of vertices, i.e., $RV((s, \psi)) \subset V$, such that for all $(s', \psi') \in RV((s, \psi))$ Player 1 can move the token from $(s', \psi')$ to $(s, \psi)$ without Player 2 being able to push the token to a state in which Player 1 loses the game by getting stuck. The variable $Pr$, defined for all $(s', \psi'), (s, \psi) \in V$ with $(s', \psi') \in RV(s, \psi)$, denotes the parent vertex $(s', \psi'' \land p)$ to $(s, \psi)$ such that the path for the token starting from $(s', \psi')$ passes through $(s', \psi'')$ right before reaching $(s, \psi)$. This variable will be used to quickly recover the route of the token in a play induced by the (optimal) winning strategy of Player 1. The variable Cost is used to maintain the total cost of the trajectory of the token in that case. The variable $nV$ is the set of all vertices $(s, \psi)$ for which $\psi$ is of the form $\nu x. \psi'$ for some $\psi' \in L_1$. In the sequel, such vertices are also called $\nu$-vertices. Finally, the variable $1V$ stores the set of all literal vertices in the arena, i.e., vertices of the form $(s, \psi)$ where $\psi = p$ or $\psi = \neg p$.

The local model checking algorithm provides two functions: $\text{AddState}$ and $\text{AddTransition}$. The former is invoked to add a new state into the Kripke structure while the latter is called to add a new transition between two existing states in the Kripke structure. Both functions return the new Kripke structure and the new arena pair, denoted by $(K, G)$.

The $\text{AddState}$ function is presented in Algorithm 1. The function simply adds the new state to the set $S$ of states of the Kripke structure $K$ while keeping the arena $G$ unchanged.
Algorithm 1: AddState((K,G),s)
1 S ← S ∪ {s};
2 return (K,G);

The AddTransition function is given in Algorithm 2.

First, the function augments the transition relation $R$ of the Kripke structure by adding the new transition. Subsequently, it calls UpdateArena and UpdateVertexSets procedures to update the arena $G$ as well as the global data structures.

Algorithm 2: AddTransition((K,G),(s₁,s₂))
1 $R ← R ∪ \{(s₁,s₂)\}$;
2 for all $\psi ∈ SF_{δ_{\text{true}}}$ do
3 \[ A ← \text{UpdateArena}(G,(s₁,\psi),(s₂,\psi)); \]
4 \[ \text{UpdateVertexSets}(G,(s₁,\psi),(s₂,\psi)); \]
5 return $(K,G)$

The UpdateArena procedure is presented in Algorithm 3. The procedure takes two vertices, $(s₁,\psi₁),(s₂,\psi₂) ∈ S × SF_{δ_{\text{true}}}$ and $(K,G)$. The procedure does not modify $K$, but it modifies the arena $G$ as follows. First, it adds the second vertex $(s₂,\psi₂)$ to the arena as an edge (Lines 17). Finally, it recursively calls itself to expand new vertices and edges that are connected to the vertex $(s₂,\psi₂)$ (Lines 18-32).

The algorithm does not expand arena to those vertices where Player 1 is doomed to lose by getting stuck (see Lines 2-8). The algorithm also updates the set $\nu$-vertices as it encounters new $\nu$-vertices that can be added to the arena, whenever the AcceptVertex($G,(s₂,\psi₂)$) procedure returns true (Lines 9-10). Different implementations of the AcceptVertex procedure lead to different algorithmic properties, which will be discussed in Section V. The set of literal vertices, i.e., vertices of the form $(s,p)$ and $(s,¬p)$ are also updated as they are encountered (Lines 11-12).

The UpdateVertexSets procedure is given in Algorithm 4, where $\chi((s,\psi))$ is the color assigned to vertex $(s,\psi)$ in the $L₁$ game (see Definition 7). The procedure transfers all those vertices in $RV(s₁,\psi₁)$ to the set $RV(s₂,\psi₂)$ if they have a color higher than that of $(s₂,\psi₂)$ (Lines 2-7). If a transfer occurs, then the $Pr$ data structure is updated accordingly. The procedure also puts $(s₁,\psi₁)$ itself into the list $RV((s₂,\psi₂))$ if $(s₁,\psi₁) ∈ nV$ and the same conditions on the colors hold (Lines 8-13). If the list $RV((s₂,\psi₂))$ is updated during the process, then the algorithm calls itself recursively on all neighbors of $(s₂,\psi₂)$ in the arena (Lines 15-16). In the algorithm, $c(s₁,s₂)$ is used to denote the cost of the trajectory connecting $s₁$ and $s₂$ in the $\omega$-trace-inclusive Kripke structure model, if $s₁ ≠ s₂$; otherwise, $c(s₁,s₂) = 0$.

B. The RRG and the RRT* Algorithms on an L₁-Game Arena

In this section, we combine the RRG and the RRT* (see [10]) to solve Problem 10 in a computationally-efficient manner with asymptotic optimality guarantees. First, let us present a set of preliminary procedures that the algorithm heavily relies on.
1) **Sampling:** The Sample procedure returns independent and uniformly distributed samples from the state space \(X\).

2) **Nearest and Near-by States:** Given a finite set \(S \subseteq X\) of states, and a state \(z \in X\), the \(\text{Nearest}(S, z)\) procedure returns the vertex (among those in \(S\)) that is closest to \(z\), i.e., \(\text{Nearest}(S, z) = \arg \min_{s \in S} \|z - s\|\), where \(\|\cdot\|\) is the usual Euclidean norm. The \(\text{Near}(S, z)\) procedure returns the states that are closest to \(z\) in the following sense:

\[
\text{Near}(S, z) := \left\{ s \in S : \|s - z\| \leq \gamma \left( \frac{\log(\text{Card}(S))}{\text{Card}(S)} \right)^{1/d} \right\},
\]

where \(\gamma\) is a constant independent of \(\text{Card}(S)\) and \(d\) is the dimensionality of the state space.

3) **Steering:** Given two states \(z, z'\), the \(\text{Steer}(z, z')\) returns the pair \((x, T)\), where \(x : [0, T] \rightarrow X\) is a trajectory that (i) starts from \(z\), (ii) reaches \(z'\), and (iii) minimizes \(\int_0^T g(x(t))dt\) whenever \(\lambda(x(t))\) changes value at most one while \(t\) varies from 0 to \(T\), i.e., \(\text{Card}\{t \in [0, T] : \lim_{t \uparrow T} \lambda(x(t)) \neq \lambda(x(t))\} \leq 1\).

The algorithm is presented in Algorithm 5. Initially, the algorithm starts with a Kripke structure with one state \(z_{\text{init}}\) and no transitions, and a (partial) arena that contains a single vertex, namely \((z_{\text{init}}, \phi_{\text{spec}})\) and no edges. The global variables are initialized as follows. The sets \(N, V\) contain no elements. However, the set \(RV(V_{\text{init}}, \phi_{\text{spec}})\) of vertices reaching \((V_{\text{init}}, \phi_{\text{spec}})\) contain a single dummy vertex denoted by \(v_0\). The parent \(\text{PR}(v_0, (V_{\text{init}}, \phi_{\text{spec}}))\) is set to NULL, whereas the cost \(\text{Cost}(v_0, (V_{\text{init}}, \phi_{\text{spec}}))\) is set to zero. The dummy vertex \(v_0\) is assumed to have color \(\chi(v_0) = 0\), so that the vertex is included in all reaching vertex sets \(RV(v)\) for all \(v \in V\) as the incremental model checking procedures are invoked. This vertex is used solely for keeping track of the cost to reach from the root to any vertex in the arena.

The algorithm proceeds in iterations (Lines 6-37). In each iteration, the algorithm first samples a new state, denoted by \(z_{\text{new}}\) (Line 7), computes the nearest state, denoted by \(z_{\text{nearest}}\) (Line 8), and steers \(z_{\text{nearest}}\) towards \(z_{\text{new}}\) (Line 9). If the steering procedure is successful, then the algorithm makes connections from and to the near-by states as in the RRG algorithm (Lines 11-20). Moreover, the algorithm implements the steps of the RRT* (Lines 14-36). In particular, the algorithm first finds the best parent, among those in the set of near vertices, for all vertices of the form \((s_{\text{new}}, \phi)\) where \(\phi\) is a sub-formula of \(\phi_{\text{spec}}\) (Lines 21-28). At this point, it is worth noting that \(((s, \phi), (s_{\text{new}}, \phi) \in E)\) is the only form of edge that satisfies \((s, s') \in E\). Moreover, the rewiring procedure of the RRT* is also executed for all vertices of the form \((z_{\text{new}}, \phi)\).

Informally speaking, the algorithm is designed to keep track of the minimum-cost path that starts from any \(v\)-vertex in \(V\) and reaches some vertex \((s, \phi)\) with \(v \in RV(v)\). For this purpose, the algorithm executes an RRT* algorithm for each vertex in \(V\) in Lines 14-36. At the same time, the algorithm also ends up maintaining the set of optimal paths that start from \((z_{\text{init}}, \phi_{\text{spec}})\) and reach a particular vertex in the arena simply by executing the same steps for the dummy vertex \(v_0\) created when initializing the algorithm.

4) **Termination and the Winning Strategy:** The Kripke structure \(K\) is a feasible solution to Problem 10 when either one of the following is satisfied: (i) there exists \((s, \psi) \in E\), or (ii) there exists \((s, \psi) \in V\) such that \((s, \psi) \in RV(v)\). These conditions can be checked during runtime, e.g., after the UpdateVertexSets procedure is invoked. Loosely speaking, this condition guarantees either one of the following. Either Player 1 can move the token to a vertex \((s, \psi) \in E\) in which Player 2 can not make any more moves, or she can move the token along a path starting from a vertex \((s, \psi) \in V\) and ending in \((s, \psi)\) while ensuring that the highest priority vertex encountered along the path is \((s, \psi')\) itself, which has even priority. This guarantees the existence of a winning strategy for Player 1. Then, the witness sequence (Section II-D), can be extracted easily through the parent vertices stored in the \(Pr\) data structure.
V. Analysis

In this section, the theoretical analysis of the algorithm is carried out. In particular, it is noted that the algorithm is probabilistically complete and asymptotically optimal. Moreover, the computational complexity of the algorithm is analyzed. It is shown that the algorithm carries out each iteration efficiently. Due to lack of space, the proofs are either omitted or only sketched.

A. Probabilistic Completeness

First, let us note that probabilistic completeness property of the algorithm.

Theorem 11 Suppose that the set \( \{ z \in X : \lambda(z) = P \} \) has non-empty interior for any \( P \subseteq \Pi \). Suppose that the system dynamics in Equation (1) is of the form \( f(z, v) = v \). Then, the probability that Algorithm 5 returns a feasible solution to Problem 10 approaches one as the number of iterations approaches infinity.

This theorem follows from the correctness of the incremental model checking algorithm (see [11]) and the probabilistic completeness of the RRG algorithm (see [10]).

B. Asymptotic Optimality

Asymptotic optimality of the algorithm is stated below. For any vertex \( (s, \psi) \), recall that \( \text{Cost}(v_0, (s, \psi)) \) stores the cost to reach \((s, \psi)\) starting from the initial vertex \((s_{\text{init}}, \phi_{\text{spec}})\). Recall also that the variable \( \text{Cost}((s, \psi), (s, \psi)) \), defined for all \((s, \psi) \in V \) that satisfies \((s, \psi) \in R((s, \psi))\), is the cost of starting from \((s, \psi)\) and reaching \((s, \psi)\) through the arena. Then, define the random variables \( Y'_n, Y''_n, Y'''_n \) as follows:

\[
Y'_n := \min\{\text{Cost}(v_0, (s, \psi)) : (s, \psi) \in 1V\}, \\
Y''_n := \min\{\text{Cost}(v_0, (s, \psi)) : (s, \psi) \in R((s, \psi))\}, \\
Y'''_n := \min\{Y'_n, Y''_n\}.
\]

In the case when \( 1V \) is empty and there is no \((s, \psi) \in V\) with \((s, \psi) \in R((s, \psi))\), then we set \( Y'''_n = 0 \).

Theorem 12 Suppose that the conditions of Theorem 11 hold. Suppose that the AcceptVertex procedure (see Algorithm 3) returns true infinitely often, and the cost of the best solution returned by the algorithm converges to the cost of the optimal solution for Problem 10 almost surely, i.e.,

\[
P\left(\lim_{n \to \infty} Y_n = c^*\right) = 1.
\]

The proof of this theorem follows from the fact that the RRT* algorithm keeps track of the shortest paths starting from each \( \nu \)-vertex in \( V \in nV \) and reaching back to the same vertex \( v \), and also the shortest path that starts from the initial vertex and reaches \( v \). These paths, when concatenated to form an infinite path, is guaranteed to converge to the optimal witness trajectory that passes through \( v \) by the asymptotic optimality of the RRT* algorithm. By ensuring that the AcceptVertex procedure returns true infinitely often, we guarantee generating \( \nu \)-vertices \( v = (s, \psi) \in nV \) such that their state components, namely \( s \), densely samples the state space. Thus, the witness trajectories returned by the algorithm converges to the optimal witness trajectory.

C. Computational Complexity

Finally, we draw the readers attention to interesting computational complexity properties of the algorithm. The following result summarizes the (per-iteration) computational complexity of the algorithm. Let \( M_n \) denote the number of vertices in the set \( nV \) of \( \nu \)-vertices by the end of iteration \( n \).

Theorem 13 The expected amortized (with respect to the number of iterations) computational complexity of the algorithm is \( O(M_n \log n) \), where \( n \) is the number of iterations.

This result can be deduced by a careful analysis of the incremental model checking procedure and also the sampling-based algorithm provided in Algorithm 5. Firstly, it can be shown that the amortized computational complexity of the incremental model checking algorithm is order \( \log \) \( M_n \log n \), since, loosely speaking, the algorithm executes \( M_n \) algorithms each of which take around \( \log \) \( M_n \log n \) time to execute, which is asymptotically negligible when compared to the computational complexity of its RRT* part. Finally noting that the near-by connections involve an expected number of \( \log \) \( n \) vertices, thus the expected value of \( \text{Card}(E) \) is order \( \log n \), and that \( \text{Card}(S) \leq n \), yields the result.

D. Implementation of the AcceptVertex procedure

The AcceptVertex procedure is implemented such that

\[
\text{AcceptVertex}(A = (V_1, V_2, E), (s_1, \psi_1)) \text{ returns true whenever} \quad \text{rand}(0, 1] \leq \frac{1}{\text{Card}(V_1 \cup V_2)},
\]

where \( \text{rand}(0, 1] \) returns a real-valued random variable between 0 and 1 according to the uniform distribution. It can be shown easily that this procedure results in the AcceptVertex procedure returning true infinitely often with probability one, thus satisfying the conditions of Theorem 12. Moreover, it can also be shown that the expected number of times the procedure returns true in \( n \) iterations is \( O(\log n) \), thus rendering the asymptotic expected amortized computational complexity of the algorithm as \( O(\log^2 n) \).
where calculus with no RRT
In fact, the algorithm can be shown to reduce to a simple
algorithm (for the classical motion planning problem) [10].

O reduces to
the following set of differential equations:

\[
\frac{dx}{dt} = f(x(t), u(t)) = [v \cos(x_3(t)), v \sin(x_3(t)), u(t)],
\]

where \( x = [x_1, x_2, x_3]^T \), \( |u(t)| \leq 1 \) for all \( t \). The states \([x_1, x_2]^T\) denotes the vehicle’s position in the plane and \( x_3 \) denotes its orientation. For simplicity, we set \( v = 1 \).

The specification requires the car to visit two distinct subsets \( R_1 \) and \( R_2 \) of the state space infinitely often, while avoiding another region \( R_3 \). Let \( p_1, p_2 \), and \( p_3 \) be the atomic propositions, which are satisfied by only those states that are in regions \( R_1, R_2, \) and \( R_3 \), respectively. More precisely, the dynamical system is labeled such that \( p_i \in \lambda(s) \) if \( s \in R_i \) for all \( i = 1, 2, 3 \). This specification can be given in the deterministic \( \mu \)-calculus as

\[
\mu \omega.((\neg p_3 \land \Box \omega) \lor \nu z.\{(p_2 \land \nu x.([\neg p_3 \land ((p_1 \land z) \lor \Box x)] \lor (p_1 \land \mu y.[\neg p_3 \land ((p_2 \land z) \lor \Box y)])\}).
\]

The algorithm described in the paper was run for this example until 1500 iterations. The cost function, in this case, is to minimize the path length for the part of the trajectory that is repeated infinitely often. The results of the computational experiment are shown in Figures 1 and 2. Figure 1 shows a depiction of the Kripke structure maintained by the algorithm. Figure 2, on the other hand, shows the best path in various stages of the algorithm. The improvements on the quality of the solution can be observed in the figure.

VI. SIMULATIONS

This section is devoted to simulation experiments evaluating the experiments. Due to lack of space, we defer a thorough experimental study to a full-length paper.

Consider the curvature constrained car governed by the following set of differential equations:

\[
\dot{x}(t) = f(x(t), u(t)) = [v \cos(x_3(t)), v \sin(x_3(t)), u(t)],
\]

where \( x = [x_1, x_2, x_3]^T \), \( |u(t)| \leq 1 \) for all \( t \).

Fig. 1. Graph maintained by the algorithm shown in different stages. The plot on the right is at the time when the first solution is found. The figure in the center shows the best path after 1500 iterations.

Fig. 2. The part of the solution that is repeated infinitely often is shown in different stages of the algorithm. The first solution found by the algorithm is shown in the figure on the left. The figure in the center shows the best path after 300 iterations, and the figure on the right shows the best path after 1500 iterations.

VII. CONCLUSIONS

This paper focused on the optimal motion planning problem to satisfy a large class of complex task specifications given in the form of deterministic \( \mu \)-calculus. An algorithm based on the RRG and the RRT* algorithms is described. It is shown that the algorithm is shown to be asymptotically optimal and incurs only order \( \log^2 n \) asymptotic expected amortized computational complexity per iteration, where \( n \) is the number of iterations. It was also shown that the algorithm naturally reduces to the RRT* algorithm, when the specification is to “reach the goal region while avoiding collision with obstacles,” in which case its computational complexity is order \( \log n \), the same as that of the RRT* algorithm. Thus, the algorithm described in this paper extends the RRT* algorithm to deal with complex task specifications.

Future work will involve experimental investigation of the algorithm, including benchmarking through simulation studies in more challenging scenarios involving a more complex dynamical systems, multiple agents, and more complex tasks.

REFERENCES