2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Sampling-based Algorithms for Optimal Motion Planning using Process Algebra Specifications

Valerio Varricchio[*] Pratik Chaudhari[†] Emilio Frazzoli[†]

Abstract— **This paper investigates motion-planning using formal language specifications for dynamical systems with differential constraints. In particular, we focus on process algebra as a language to specify complex task specifications motivated by autonomous electric vehicles operating in a mobility-on-demand scenario. We use ideas from sampling-based motion-planning algorithms to incrementally construct a finite abstraction of the dynamical system as a Kripke structure. Given a task specification expressed as a *process graph*, we use model checking techniques to construct a weighted product graph of the specification with the Kripke structure. We then devise an algorithm that provably converges to the optimal trajectory of the dynamical system that satisfies the task specification as the number of the states in the Kripke structure goes to infinity. The algorithm is demonstrated in simulation experiments, viz., charging the electric car at a busy charging station and scheduling pick-ups and drop-offs of passengers.**

I. Introduction

Formal languages are a popular way to express specifications for automated control synthesis and the general problem of finding controllers that satisfy temporal logics has been studied in a number of recent works [1], [2]. Languages such as Linear Temporal Logic (LTL), Computational Tree Logic (CTL) and deterministic μ-calculus are powerful ways to specify task specifications. With greater expressivity, however, comes greater complexity, e.g., checking LTL formulae is PSPACE-complete [3] while model checking deterministic μ-calculus belongs to NP $\cap$ co-NP [4]. In addition to this, one of the main challenges of these approaches is that the quality of controllers depends on finite abstractions of continuous dynamical systems. If the abstracted finite transition system is too coarse, there is no guarantee that the optimal controller will be found even if one exists [5]. Our approach is primarily motivated by these two problems.

Firstly, we note that algorithms from sampling-based motion planning literature like Probabilistic Road Maps (PRMs) and Rapidly-exploring Random Trees (RRTs) have been very successful in systematically generating feasible trajectories for dynamical systems with differential constraints [6]. Algorithms such as PRM* and RRT* [7] also guarantee asymptotic optimality while doing so, with minor overhead on computation cost. Although they have been primarily used for motion planning, we can also leverage them to generate finite abstractions of continuous-time dynamical systems. Model checking algorithms based on specification languages such as Finite LTL and μ-calculus can then be used to synthesize controllers that find a provably optimal trajectory of the continuous dynamical system that satisfies given task specifications [8], [9].

Secondly, in this work, we focus on languages which forgo expressive power but in turn allow for efficient model checking algorithms. As an example, note [10], which restricts specifications to reactive temporal logic to obtain polynomial complexity of translation into automata as opposed to doubly exponential complexity of the full LTL. We use process algebra specifications [11], [12] which, although not as expressive as LTL or μ-calculus, are still enough to specify a number of tasks of practical interest. In fact, expressing general specifications in these languages requires that the underlying Kripke structure is a graph, whereas in our work we can converge to optimality even with a tree structure.

Our work is similar to problems considered in [13], [14]; we however focus on continuous dynamical systems with differential constraints and show that our algorithm converges to the optimal solution in the limit. Another line of literature poses these problems using mixed-integer linear programs (e.g., [15], [16]). These methods are however restricted to linearizable dynamics and discrete time; also the number of integer constraints grows quickly with the number of obstacles. Sampling-based approaches to abstract the continuous dynamics help us alleviate these issues while still maintaining computational efficiency.

The contributions of this work are as follows: first, we use sampling based algorithms to incrementally generate a Kripke structure that is representative of the given dynamical system and provide a computationally efficient algorithm to synthesize control strategies based on model checking techniques. Secondly, this algorithm is shown to be probabilistically asymptotically optimal, i.e.,with probability one, the trajectory returned by the algorithm converges to the optimal trajectory of the dynamical system that satisfies the given task specification as the number of states in the Kripke structure goes to infinity. We also provide results of simulation experiments on examples motivated by autonomous vehicles operating in an urban mobility-on-demand scenario viz., charging at a busy charging station and picking up and dropping off passengers at various locations.

The paper is organized as follows: Sec. II introduces preliminary material on continuous-time dynamical systems and process algebra specifications. In Sec. III, we describe our approach and formalize the problem addressed in this paper. Sec. IV discusses the algorithm while Sec. V and

[*]The author is with Sant'Anna School of Advanced Studies and E.Piaggio Robotics Research Centre, University of Pisa, Italy.
Email: v.varricchio@gmail.com
[†]The authors are with the Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA.
Email: pratikac@mit.edu, frazzoli@mit.edu

Sec. VI describe results from simulation experiments. We conclude with directions for future work in Section VII.

II. Preliminaries

In this section, we introduce Kripke structures and adapt them for continuous dynamical systems. We also introduce Basic Process Algebra as a formal specification language.

A. Dynamical systems

Given a set of atomic propositions Π, for compact sets $X \subset \mathbb{R}^d$, $U \subset \mathbb{R}^m$, consider a dynamical system given by

$$\dot{x}(t) = f(x(t), u(t)); \quad x(0) = x_0 \tag{1}$$

where x_0 is the initial state. We denote state and control trajectories by $x : [0,T] \to X$ and $u : [0,T] \to U$, respectively. The function $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ is assumed to be Lipschitz continuous in both its arguments and u is Lebesgue measurable to guarantee existence and uniqueness of solutions. A labeling function $\mathcal{L}_c : X \to 2^{\Pi}$ maps each state to the atomic propositions that are true at that state. Also, let $D(x) = \{t \,|\, \lim_{s \to t^-} \mathcal{L}_c(x(s)) \neq \mathcal{L}_c(x(t)), t \in (0,T]\}$ denote the set of discontinuities of the labels of a trajectory x. We assume that $D(x)$ is finite for any trajectory x. Given a $D(x) = \{t_1, t_2, \ldots, t_n\}$, the finite word generated by a trajectory is the sequence $\rho(x) = \rho_0, \rho_1, \ldots, \rho_n$ such that, $\rho_0 = \mathcal{L}_c(x(0))$ and $\rho_i = \mathcal{L}_c(x(t_i))$ for all other $i \leq n$.

Definition 1 (Durational Kripke structure) *Given a set of atomic propositions Π, a durational Kripke structure is a tuple $K = (S, s_{init}, R, L, \Delta)$, such that,*

- *S is the finite set of states;*
- *$s_{init} \in S$ is the initial state;*
- *$R \subseteq S \times S$ is a deterministic transition relation;*
- *$L : S \to 2^{\Pi}$ is a state labeling function;*
- *$\Delta : R \to \mathbb{R}_{\geq 0}$ is a function assigning a time duration to every transition.*

A *trace* of K is a finite sequence of states $r = s_0, s_1, \ldots, s_n$ such that $s_0 = s_{init}$ and $(s_i, s_{i+1}) \in R \; \forall i : 0 \leq i < n$. It produces a finite *timed sequence* $(l_0, d_0), \ldots, (l_n, d_n)$ where $(l_i, d_i) = (L(s_i), \Delta(s_{i+1}, s_i))$ and $(l_n, d_n) = (L(s_n), 0)$. The corresponding *word* produced by this is $\rho(r) = \ell_0, \ell_1, \ldots, \ell_n$. Given such a $\rho(r)$, let $I = \{i_0, i_1, \ldots, i_k\}$ be the unique 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 $\ell_k = \ell_{k+1} = \ldots = l_n$ with the additional condition that $l_{i_j} \neq \emptyset$ for any $0 \leq j \leq k$. Define an operator *destutter* on a timed word as,

$$destutter(\rho(r)) = l_{i_0}, l_{i_1}, \ldots, l_{i_{k-1}}, l_{i_k},$$

i.e., the *destutter* operator removes repeated consecutive elements and elements of the trace which do not satisfy any atomic proposition from the set Π. The *destutter* operator helps in formalizing when a word of a dynamical system or Kripke structure satisfies a task specification.

In this work, we also require that Kripke structures for dynamical systems are *trace inclusive*, i.e., for any two states s_1, s_2 such that $(s_1, s_2) \in R$, there exists a trajectory $x : [0,T] \to X$ s.t. $x(0) = s_1$, $x(T) = s_2$ and $|D(x)| \leq 1$. The continuous trajectory changes its label at most once during every transition of a trace-inclusive Kripke structure.

B. Process Algebra

Process Algebra (PA) is a mathematical framework that was developed to formalize and understand the behavior of complex concurrent systems. *Behavior* is the result of *actions* that happen at specific time instants when their preconditions are true and represent the output of the system. In other words, behavior is simply the ordered set of actions that are observed in the system over a course of time. In this work, we consider atomic actions of the kind "get to region $\mathcal{A}$", "next, go to region $\mathcal{B}$", etc. where $\mathcal{A}, \mathcal{B}$ are regions in the robot's operating environment. Compositions of these actions are used to construct different behaviors.

Definition 2 (Basic Process Algebra) *Let A be a finite, non-empty set of atomic actions. The set $\mathbb{T}$ of Basic Process Algebra (BPA) terms over A is defined inductively as:*

- *For all actions $p \in A$, $p \in \mathbb{T}$;*
- *For $p, p' \in \mathbb{T}$, $p + p'$, $p \cdot p'$ and $p \,||\, p' \in \mathbb{T}$.*

Compositions of processes $p, p' \in \mathbb{T}'$, such as $p + p'$, $p \cdot p'$ and $p \,||\, p'$ are called alternative, sequential, and parallel compositions, respectively. The semantics of process algebra are as follows: a process $a \in \mathbb{T}$ executes a and terminates. The process $p + p' \in \mathbb{T}$ executes the behavior of either p or p'. Similarly, the $p \cdot p' \in \mathbb{T}$ first executes the term p and then executes the behavior of p' after p terminates. $p \,||\, p'$ executes the behaviors of both p and p' simultaneously and terminates after they both terminate. Parallel composition is widely used to study concurrent systems, our work however considers only a single agent and hence we do not use the parallel operator. As a convention, the sequential operator has priority over the alternative operator. A sequence of actions is also called as a *trace*. Given a process $p \in \mathbb{T}$, we define a *process graph* that maps the process to the set of set of traces that describe the behavior of the process.

Definition 3 (Process Graph) *A process graph for $p \in \mathbb{T}$, is a labeled transition system $G(p) = (Q, q_0, A, \pi, \delta, F)$ where Q is a set of states, $q_0 \in Q$ is the initial state, $\pi : Q \to \mathbb{T}$ associates each state $q \in Q$ with a term $t \in \mathbb{T}$ with $\pi(q_0) = p$, while $\delta \subseteq Q \times A \times Q$ is a transition relation and $F \subset Q$ is a set of accepting states.*

A process p is said to evolve to a process p' after actions $a_1, a_2, \ldots, a_n$ if there is a sequence of states $q_1, q_2, \ldots q_{n+1}$ in $G(p)$ such that $\pi(q_1) = p$ and $(q_i, a_i, q_{i+1}) \in \delta$ for all $1 \leq i \leq n$ with $\pi(q_{n+1}) = p'$. We can now define the *trace* of a process p as a sequence of atomic actions $\gamma = (a_1, a_2, \ldots, a_n)$ for which the corresponding sequence of states $q_1, q_2, \ldots, q_{n+1}$ exists such that $(q_i, a_i, q_{i+1}) \in \delta$ for all $1 \leq i \leq n$. The set of all traces of a term p is denoted by $\Gamma_{G(p)}$. Note that we have modified the conventional definition of the process graph to include the set of accepting

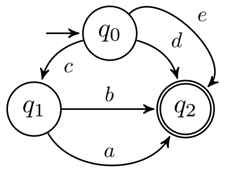


Fig. 1: An example process graph generated for the expression $c\cdot(a+b)+(d+e)$. The different traces are, $c, c.a, c.b, d, e$. The last four out of these are accepting traces.

states F, which contains nodes without any outgoing edges. The set of traces of p that end at a state inside F are called *accepting* traces. In this context, let us define a function $\texttt{cost}_G(q)$ to be the length of the shortest accepting trace of q. Note that $\texttt{cost}_G(q) = 0$ if $q \in F$. Given a BPA expression, the product graph G can be constructed using a simple recursive parser. In the sequel, we let the set of atomic propositions Π to be the same as the set of atomic actions A for the Basic Process Algebra.

III. Problem formulation and Approach

This section describes our approach and motivates the algorithm in Sec. IV. Roughly, to compute a trajectory that satisfies a specification $p \in \mathbb{T}$, we construct the product of a trace-inclusive Kripke structure K and the process graph $G(p)$. This product is constructed in such a way that accepting traces of the product graph can be mapped to accepting traces of the process graph. We first define the problem using the continuous-time dynamical system as shown below.

Problem 4 *Given a dynamical system modeled by Eqn. (1) and a process algebra term p, find a trajectory $x^* : [0,T] \to X$ such that, (i) x^* starts at the initial state, i.e., $x^*(0) = x_0$, (ii) x^* satisfies the task specification p, i.e., destutter$(\rho(x^*))$ is an accepting trace of $G(p)$, and (iii) x^* minimizes the cost function $c(x)$ among all trajectories that satisfy (i) and (ii).*

In this paper, we consider cost functions $c(x) = \int_0^T 1.dt$, i.e., time optimal cost functions. However, the problem formulation is general and can incorporate other kinds of costs such as minimum effort by changing the definition of the weighting function W' in Def. 5 below.

Definition 5 (Weighted Product graph) *Given a Kripke structure $K = (S, s_{init}, R, L, \Delta)$ and a process graph $G(p) = (Q, q_0, A, \pi, \delta, F)$ for a process p, the product graph is a labeled transition system defined as the tuple $P = (Q_P, q_{0,P}, A_P, \pi_P, \delta_P, F_P, W_P)$ where:*

- *$Q_P = S \times Q$ is the set of states;*
- *$q_{0,P} = (s_{init}, q_0)$ is the initial state;*
- *$A_P = S \times S \times A$ is the set of atomic actions;*
- *$\pi_P((s,q)) = \pi(q)$ for all $(s,q) \in Q_P$;*
- *$(q'_1, a', q'_2) \in \delta_P$ iff $(s_1, s_2) \in R$, and either $q_1 = q_2$ or $(q_1, a, q_2) \in \delta$ with $a \in L(s_2)$;*
- *$F_P = S \times F$ is the set of accepting states;*
- *$W_P(q'_1, q'_2) = \Delta(s_1, s_2)$ is a weighing function.*

where $q'_i = (s_i, q_i)$ and $a' = (s_1, s_2, a)$.

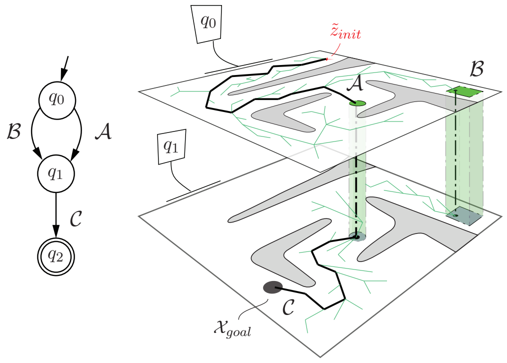


Fig. 2: Conceptually, the product graph is the Cartesian product of the process graph with the Kripke structure and can be visualized using layers. Given a term, e.g., $(\mathcal{A}+\mathcal{B})\cdot\mathcal{C}$, and the Kripke structure shown using green lines, the optimal trace in the product graph goes to either region $\mathcal{A}$ a or region $\mathcal{B}$ on the same layer and then travels downwards to reach region $\mathcal{C}$, whichever minimizes the cost.

We use $P = K \oplus G(p)$ to denote the product of a Kripke structure K and a process graph $G(p)$. Let us note how traces of the product graph are related to traces of the process graph. Given a $\gamma' = \{a'_1, a'_2, \ldots, a'_n\} \in \Gamma_P$, where $a'_i = (s_{i,1}, s_{i,2}, a_i)$, if $z_1, z_2, \ldots, z_{n+1}$ are the corresponding states in P, where $z_i = (s_i, q_i)$, the projection on the Kripke structure is $\alpha_K(\gamma') = s_{1,1}, s_{2,1}, \ldots, s_{n+1,1}$ and the projection on the process graph is $\alpha_G(\gamma') = q_1, q_2, \ldots, q_{n+1}$. The following lemma then follows easily.

Lemma 6 *An accepting trace of the product uniquely maps to an accepting trace of the process graph $G(p)$.*

IV. Algorithm

In this section, we describe the algorithm used to construct the product graph in Sec. III. We use techniques from sampling-based motion-planning algorithms to construct a trace-inclusive Kripke structure for a dynamical system. We then form the product graph as described in Def. 5 incrementally. An ordered tuple of the length of the shortest accepting trace from a node and the weighing function of the product graph is used to maintain a trace in the product graph that satisfies the task specification and projects onto a trace of the Kripke structure that minimizes a given cost function. In Sec. IV-B, we show that the solution of this algorithm converges to the solution of Prob. 4 in an appropriate sense.

A. Preliminary procedures

Let us introduce a few preliminary procedures. Nodes of the process graph are denoted by $z = (s,q)$ and $z' = (s', q')$.

a) Sampling: The $\texttt{sample}$ procedure draws a sample $s \in X$ independently of a uniform distribution over X.

b) Local steering: For a state s, for all $q \in Q$, and a node z', if $q = q'$ or there exists an $a \in A$ in the process graph G, s.t. $(q, a, q') \in \delta$ and $a \in L(s)$, the $\texttt{steer}$ function returns a real number T, a control trajectory $u : [0,T] \to U$ and a state trajectory $x : [0,T] \to X$ with $x(0) = s'$, $x(T) = s$, and such that x minimizes the cost function $c(z', z)$. The trajectory x is also required to be trace inclusive.

(s, q) is then added to the product graph and if $q \in F$, we also add (s, q) to the set of accepting states in the product graph.

c) Near vertices: For a state s, let $S_{near}(s)$ be a subset of the reachable space of the dynamical system consisting of the closest $k \log n$ ($k > 2$) states to s in the Kripke structure according to the cost metric used in the `steer` procedure. Note that S_{near} can be calculated efficiently using the Ball-Box Theorem (see [17] for more details). For the dynamics of a Dubins' car considered in this paper, the size of the boxes is computed in [17]. The `near` procedure returns

$$\mathtt{near}(s) = \{(s', q') \mid s' \in S_{near}(s), (s', q') \in Q_P\}.$$

d) Calculate Cost: Each node maintains a 2-tuple $\mathtt{cost}(z) = (\mathtt{cost}_G(q), \mathtt{cost}_K(s))$ which is an ordered tuple of the minimum number of steps that q takes to reach an accepting state in G and the cost of reaching a state s in the Kripke structure. Given z', z, the `calculate_cost` procedure returns the cost, $\mathtt{cost}(z)$, of reaching z through z'. $\mathtt{cost}_K(z', z)$ is the cost returned by $\mathtt{steer}(z', z)$, i.e., $W_P(z', z)$, and hence $\mathtt{cost}_K(z) = \mathtt{cost}_K(z') + \mathtt{cost}_K(z', z)$. Note that $\mathtt{cost}_G(z)$ can be calculated from the process graph. We use the lexicographical ordering to compare cost tuples.

e) Rewiring: The `rewire` procedure, as shown in Alg. 2, checks if any nodes connected to z in the product graph can improve their cost using the transition from z. If so, it reassigns their parents.

Algorithm 1: PARRT*
```
1  Input : $n$, $X$, $s_{init}$, $G(p)$;
2  $z_0 \leftarrow (s_{init}, q_0)$, $Q_P \leftarrow \{z_0\}$;
3  $\mathtt{cost}(z_0) \leftarrow (\mathtt{cost}_G(q_0), 0)$;
4  $i \leftarrow 0$;
5  for $i \leq n$ do
6      $s \leftarrow$ sample;
7      for $z' \in \mathtt{near}(s)$ do
8          for $q \in Q$ do
9              $z \leftarrow (s, q)$, $(s', q') \leftarrow z'$;
10             $x, u, T \leftarrow \mathtt{steer}(z', z)$;
11             $Q_P \leftarrow Q_P \cup \{z\}$, $\delta_P \leftarrow \delta_P \cup \{(z', z)\}$;
12             $W_P(z', z) \leftarrow T$;
13             if $q \in F$ then
14                 $F_P \leftarrow F_P \cup \{z\}$;
15             $c \leftarrow \mathtt{calculate\_cost}(z', z)$;
16             if $c < \mathtt{cost}(z)$ then
17                 $\mathtt{parent}(z) \leftarrow z'$, $\mathtt{cost}(z) \leftarrow c$;
18     rewire($s$);
19 $P_n \leftarrow (Q_P, q_{0,P}, \delta_P, F_P, W_P)$;
20 return $P_n$
```

Alg. 1 also maintains the state $z^* = (s^*, p^*)$ in the product with the least cost. Let the trace constructed by following $\mathtt{parent}(z^*)$ backwards be $z_0, z_1, z_2, \ldots, z_m$ where $z_0 = (s_{init}, q_0)$ and $z_m = z^*$ with γ^* being the corresponding trace in Γ_P. The projection $\alpha_K(\gamma^*) = s_0, s_1, s_2, \ldots, s_m$ gives the trace on the Kripke structure. Since K is trace-inclusive, we can construct a continuous trajectory of the dynamical system after n iterations of Alg. 1, say x_n by concatenating the outputs of the `steer` procedure between s_i, s_{i+1}. The following analysis shows that this trajectory converges to the optimal solution of Prob. 4.

Algorithm 2: rewire(s)
```
1  for $q \in Q : \{(s, q) \in Q_P\}$ do
2      $Z_{near} \leftarrow \{z' \mid (z', z) \in \delta_P\}$;
3      for $z' \in Z_{near}$ do
4          $c \leftarrow \mathtt{calculate\_cost}(z, z')$;
5          if $c < \mathtt{cost}(z')$ then
6              $\mathtt{parent}(z') \leftarrow z$, $\mathtt{cost}(z') \leftarrow c$;
```

B. Analysis

Theorem 7 *As the number of states in the Kripke structure tends to infinity, x_n converges to x^* in the bounded variation norm sense, almost surely, i.e.,* $\mathbb{P}\left(\{\lim_{n\to\infty} \|x_n - x^*\|_{BV} = 0\}\right) = 1$.

The proof of the above theorem is very similar to that of Thm. 16 in [8] and is hence omitted. Roughly, it can be shown that for a large enough n, the Kripke structure is such that it contains states in the neighborhood of the optimal trajectory x^*. This neighborhood is a function of the $k \log n$ nearest neighbors of the set S_{near} considered in the `near` procedure. Instead of covering the optimal trajectory x^* with an overlapping sequence of balls of radius $\gamma(\log n/n)^{1/d}$, we cover it with an overlapping sequence of scaled boxes from the Ball-Box Theorem which also have volume $\mathcal{O}(\log n/n)$. The result then follows by noticing that a transition in the Kripke structure exists for any two states lying in adjacent boxes, i.e., there exists a trace of the Kripke structure around the optimal trajectory x^*. It can be shown that the continuous trajectory constructed from this trace converges to the optimal trajectory almost surely [7].

C. Computational complexity

Let us now comment on the computational complexity of constructing the product graph using Alg. 1. Note that if n is the number of states in the Kripke structure, the size of $\mathtt{near}(z)$ is $\mathcal{O}(\log n)$ in expectation. For a process algebra expression p of length m, the size of process graph can be upper bounded by m. Therefore, at most $\mathcal{O}(m^2 \log n)$ states are added to the product graph in lines 11-14. The rewire procedure executes on $\mathcal{O}(m^2 \log n)$ neighbors and thus takes $\mathcal{O}(m^2 \log n)$ time. The total computational complexity in expectation can then be shown to be $\mathcal{O}(m^2 \log n)$ per iteration.

V. Experiments - Local Planner

Consider an autonomous electric vehicle that needs to charge at a charging station. In the event that the charging station is already occupied, the vehicle waits in an empty parking spot and proceeds to charge once the charging station becomes free. This behavior, which we call *local planner* is a direct application of Alg. 1 and is discussed in detail in this section. All examples in the following two sections were implemented in C++ using the Robot

Operating System (ROS) platform [18] on a computer with a 2.0 GHz processor and 4 GB of RAM.

A. Setup

We model the autonomous car as a Dubins vehicle with the following dynamics,

$$\dot{x}(t) = v\cos\theta(t) \quad \dot{y}(t) = v\sin\theta(t) \quad \dot{\theta}(t) = u,$$

where the state of the vehicle is $[x, y, \theta]^T$ while the control input is the turning rate, u with $|u(t)| \leq 1$ for all times $t > 0$. The velocity v is taken to be a constant. Time optimal trajectories for this dynamics can be easily calculated using Dubins curves [19]. In particular, a trajectory between two states that minimizes time is a concatenation of curves from the set $\{L, C, R\}$, where L, R denote turning left and right at maximum turning rate, respectively, while C denotes $u = 0$.

Fig. 4a shows the experimental scenario of a charging station inside a parking lot. The station is labelled s_1 while parking lots are labelled $w_1, w_2, \ldots, w_4$. The charging specification can then be written as,

$$\Phi_c = s_1 + (w_1 + w_2 + w_3 + w_4) \cdot \; s_1 \tag{2}$$

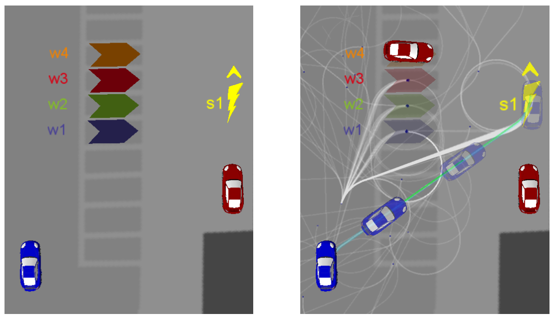
which express the task "either directly go to charging station" or "first go to one of the parking spots and then go to the charging station".

$$\begin{tikzcd}[column sep=large, row sep=large]
\arrow[r] & q_0 \arrow[d, bend right=60, "w_1"'] \arrow[d, bend right=20, "w_2"] \arrow[d, bend left=20, "w_3"'] \arrow[d, bend left=60, "w_4"] \arrow[r, bend left, "s_1"] & q_2 \\
& q_1 \arrow[ru, bend right, "s_1"'] &
\end{tikzcd}$$

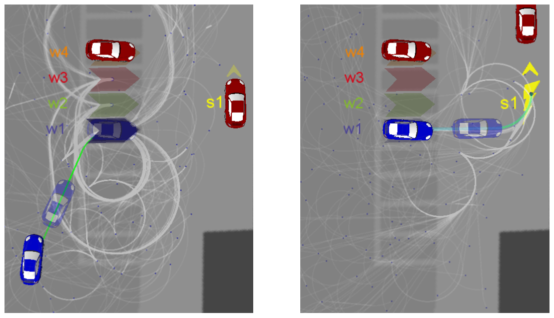
Fig. 3: Process graph for $s_1 + (w_1 + w_2 + w_3 + w_4) \cdot \; s_1$.

The problem domain X is split into disjoint sets X_{obs}, X_{free}. In case of multiple obstacles, X_{obs} is the union of all states $x \in X$ that lie inside obstacles while $X_{free} = X \backslash X_{obs}$. We define a region X_{a_i} for each atomic action a_i in the set $A = \{a_1, a_2 \ldots, a_n\}$. A state $x \in X_{a_i} \subset X$ if taking an action a_i from any other state x' results in x. Regions corresponding to atomic actions $\{w_1, w_2, w_3, w_4, s_1\}$ are shown in Fig. 4b. We modify the `steer` procedure in Sec. IV-A to return false if the trajectory passes through X_{obs}. Note that this does not affect the convergence properties of Alg. 1.

Fig. 4b shows a situation when the charging station is immediately available; the electric car therefore directly heads towards s_1 using the optimal trajectory shown in blue. Note that branches in Kripke structure also lead to the empty parking lots, but they are not preferred. On the other hand, when the station is occupied as shown in Fig. 4c, the Kripke structure only contains feasible trajectories that take the car into the parking lots. The cost tuple of these trajectories that end in the parking lot is $(1, *)$ since they are all one step away from reaching the accepting state s_1 in the process graph (see Fig. 3). As soon as the charging station becomes available, the algorithm obtains a trajectory that reaches the accepting state as shown in Fig. 4d. This example was run online and a first solution was computed in 0.8s.



(a) Charging domain (b) Station available immediately



(c) Station occupied (d) Station freed

Fig. 4: Charging specification

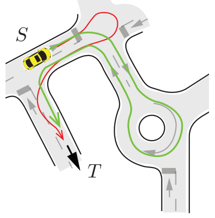
VI. Experiments - Route Planner

In this section, we provide another application, a *route planner*, which receives requests for pick-up and drop-off from a number of locations at the same time. The autonomous vehicle schedules these tasks using a process algebra specification which ensures that all requests are served optimally. We consider cases when these requests can be satisfied simultaneously, i.e., passengers share the autonomous car during transit. The autonomous vehicle can also include a charging maneuver in addition to these pick-ups and drop-offs. We assume that a passenger is not dropped off at a location that is not her destination.

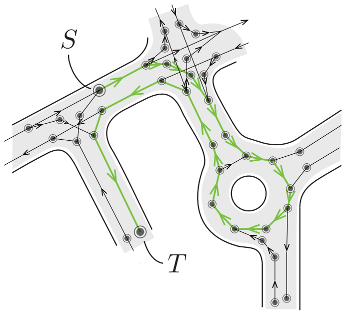
A. Generating specifications

We first discuss an algorithm to automatically construct process algebra specifications for n pick-ups $\{p_1, p_2, \ldots, p_n\}$ and drop-offs $\{d_1, d_2, \ldots, d_n\}$. Given a capacity κ, we require that (i) for any subsequence, the difference in the number of pick-ups and number of drop-offs is never larger than κ and, (ii) the index of p_i, which we denote by p_i itself, is always smaller than the index of d_i, i.e., $p_i < d_i$ for all $1 \leq i \leq n$. Valid traces for the n request problem are simply all permutations of the set $\{p_1, \ldots, p_n, d_1, \ldots, d_n\}$ that satisfy these two conditions.

Listing all permutations with the alternative operator will however give a PA term with a large number of sub-terms. The process graph for such an expression is not computationally efficient for Alg. 1 since it has a large branching factor, roughly $\mathcal{O}(e^{\mathcal{O}(n \log n)})$, with all nodes at a depth of one. We instead use a recursive algorithm that finds all permutations which satisfy constraints (i) and (ii) and also automatically



(a)



(b)

Fig. 5: Kripke structure for the route planner : Fig. 5a shows the optimal trajectory from S to T in red, while the green trajectory follows road rules. Fig. 5b shows how the generated Kripke structure incorporates these rules.

finds an efficient compression for the specification. For a sequence s, we denote the i^{th} element by s_i. Note that both (i) and (ii) can be efficiently checked by one traversal along the sequence.

The algorithm starts with $s = p_1.d_1.p_2.d_2 \ldots .p_n.d_n$. The procedure `capacity` calculates the capacity of the vehicle required for the current sequence s while the procedure `is_ordered` checks constraint (ii). Alg. 3 depicts a procedure to recursively compute a process algebra term for n pick-ups and drop-offs which can be obtained by executing `permute`$(s, 1)$.

Algorithm 3: `permute`(s, k)

```
1 if capacity($s$) $\leq \kappa$ and is_ordered($s$) then
2     $\Phi \leftarrow \varnothing$;
3     for $k \leq i \leq 2n$ do
4         swap($s_k, s_i$);
5         $\Phi \leftarrow \Phi + (s_k \cdot$ (permute($s, k+1$)));
6         swap($s_k, s_i$);
7     return $\Phi$;
```

B. Examples

The experiments in this section are performed on a pre-generated map of a part of the National University of Singapore's campus. We manually construct a Kripke structure on the map in such a way that road-lanes and directions of one-way roads are incorporated by including the appropriate directed edges in the Kripke structure. We have written a Flash based utility where the user can easily generate such a Kripke structure given a road map, as shown in Fig. 5. Note that traffic in Singapore drives on left side of the road.

1) Case 1: Fig. 6 shows the optimal plan with two pick-up and drop-off requests. The process algebra specification for this task with a capacity, $\kappa = 2$, is

$$\Phi_2 = p_1.(d_1.p_2.d_2 + p_2.(d_1.d_2 + d_2.d_1)) + \\ p_2.(d_2.p_1.d_1 + p_1.(d_1.d_2 + d_2.d_1)).$$

Based on the locations of p_1, p_2, d_1 and d_2, the algorithm returns a trajectory that satisfies the task $p_1.p_2.d_1.d_2$. It therefore outputs a trajectory where p_2 shares the car with p_1 while the vehicle drops off p_1 at her destination d_1.

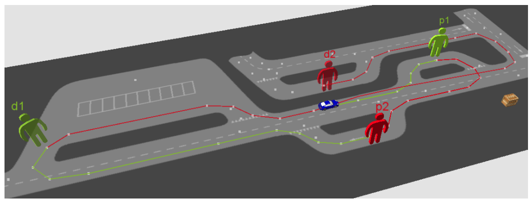


Fig. 6: Two requests : The autonomous vehicle first picks up p_1, shown using the green trajectory. p_2 shares the car with p_1 while the vehicle goes to destination d_1. The vehicle then proceeds to d_2 to drop-off p_2. Note that all trajectories satisfy rules of the road.

2) Case 2: In the second case, we incorporate charging into the specification for pick-ups and drop-offs. Consider m charging locations, $c_1, c_2, \ldots, c_m$, we can again find permutations of the extended set $\{p_1, \ldots, p_n, d_1, \ldots, d_n, c_1, \ldots, c_m\}$ such that, (i) and (ii) are true from Case 1, and, (iii) vehicle charges only when it is empty. The third condition can be checked by ensuring that before every c_i in the current string s, there are equal number of pick-ups and drop-offs, i.e., the vehicle is empty. This is checked by a procedure `charging_constraint` which is executed along with `is_ordered` on Line 1 of Alg. 3 and returns true if (iii) is satisfied.

Each node in the product graph stores the battery level $b(z)$ and the cost as a 3-tuple, i.e., $\texttt{cost}(z) = (\texttt{cost}_B(z), \texttt{cost}_P(z), \texttt{cost}_K(z))$ where $\texttt{cost}_P$ and $\texttt{cost}_K$ are the same as the `calculate_cost` procedure in Sec. IV. If $z = (s, q)$ is a charging station, $\texttt{cost}_B(z) = 0$ while it is $\max\{b_{min} - b(z), 0\}$ otherwise. b_{min} is some minimum battery threshold and $b(z)$ decreases linearly with the distance traveled since the last charging station.

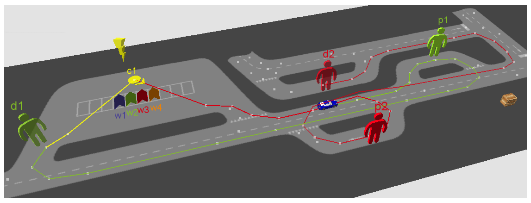
Fig. 7 shows an example with two pick-up and drop-off requests and one charging station. The specification for this problem, $\Phi_{2,c}$, is

$$p_1.(d_1.(\epsilon + \Phi_c).p_2.d_2 + p_2.(d_1.d_2 + d_2.d_1).(\epsilon + \Phi_c)) + \\ p_2.(d_2.(\epsilon + \Phi_c).p_1.d_1 + p_1.(d_1.d_2 + d_2.d_1).(\epsilon + \Phi_c)).$$

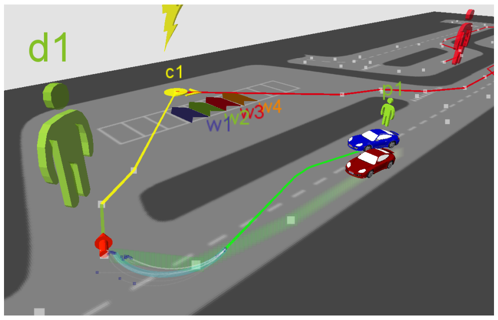
where we denote the empty atomic action by ϵ. Note that we have replaced c_1 by the charging specification Φ_c from Eqn. (2). This enables behaviors similar to those shown in Fig. 4 in case the charging station is occupied. We consider the same locations of p_1, p_2, d_1, d_2 as Case 1 along with a charging station. When the vehicle starts with a depleted battery, in contrast to the plan returned in Case 1, it picks up p_1 but directly drives to d_1 to drop her off. After swapping batteries at the charging station c_1, it proceeds to pick-up p_2 and then onward to d_2. The overall computation time for these experiments averages to 0.5s.

C. Implementation details

In this section, we briefly note some implementation details of the examples in Sec. V and Sec. VI. Both the examples use a local map around the current position of the vehicle for obstacle checking; this is represented as an occupancy grid and is constructed using two simulated laser sensors. We run Alg. 1 in a receeding horizon fashion, i.e., given a task specification, we only execute a pre-defined



(a)



(b)

Fig. 7: Two requests with charging : Fig. 7a shows the trajectory generated for $\Phi_{2,c}$. Fig. 7b shows randomly sampled states of the Kripke structure in order to avoid obstacles.

portion of the trajectory returned by Alg. 1. The present algorithm inherits "anytime" properties of the RRT* algorithm, and can incrementally improve a sub-optimal solution as the vehicle is executing this trajectory. We use a pure-pursuit tracking controller that tracks Dubins time-optimal paths for executing these trajectories [20].

The route-planner, i.e. experiments in Sec. VI, is executed on a large, sparse Kripke structure constructed manually as described in Fig. 5. After generating the specifications, say $p \in \mathbb{T}$, using Alg. 3, we calculate the optimal trajectory on the product that satisfies this specification using the well-known Dijkstra's algorithm. This trajectory is executed sequentially by passing PA sub-terms to the local planner. The manually constructed Kripke structure is also used to bias the `sample` procedure in Sec. IV. Note that this enables the vehicle to find feasible trajectories around obstacles as shown in Fig 7b.

VII. Conclusions

This paper discusses the problem of motion planning for dynamical systems with differential constraints using process algebra as a formal specification language to express complex missions for an autonomous electric vehicle in a mobility-on-demand scenario. Inspired by ideas from sampling-based motion-planning algorithms, we incrementally construct a Kripke structure as a concretization of a dynamical system. Coupled with ideas from model checking literature, we propose an algorithm whose solution converges to the optimal continuous-time trajectory that satisfies the task specification. We demonstrate the algorithm on two examples, one where an electric vehicle needs to charge itself at a busy charging station, and a second one where a mobility-on-demand vehicle picks up and drops off passengers at their respective destinations. Directions for future work consist of testing this algorithm on an experimental autonomous vehicle and generalizing the presented algorithms to properly plan for a fleet of vehicles.

VIII. Ackowledgements

This work was initiated when the authors were visiting researchers at the Singapore-MIT Alliance for Research and Technology (SMART), Singapore. It was funded by National Science Foundation, Singapore through the Future-Mobility SMART IRG.

References

[1] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.
[2] Stephen L Smith, Jana Tumova, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14):1695–1708, 2011.
[3] Philippe Schnoebelen. The Complexity of Temporal Logic Model Checking. *Advances in Modal Logic*, 4:393–436, 2002.
[4] E Allen Emerson. Model checking and the mu-calculus. *DIMACS series in discrete mathematics*, 31:185–214, 1997.
[5] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon control for temporal logic specifications. In *Proc. of ACM international conf. on Hybrid systems*, pages 101–110, 2010.
[6] James J Kuffner Jr and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. of IEEE Conf. on Robotics and Automation*, volume 2, pages 995–1001, 2000.
[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011.
[8] 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.
[9] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *Proc. of American Control Conference (ACC)*, pages 735–742, 2012.
[10] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Proc. of IEEE Conf. on Robotics and Automation*, pages 3116–3121, 2007.
[11] Jos CM Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131–146, 2005.
[12] Wan Fokkink. *Introduction to process algebra*. Springer, 2000.
[13] Sertac Karaman, Steven Rasmussen, Derek Kingston, and Emilio Frazzoli. Specification and planning of uav missions: a process algebra approach. In *Proc. of American Control Conference (ACC)*, pages 1442–1447, 2009.
[14] Xu Chu Ding, Marius Kloetzer, Yushan Chen, and Calin Belta. Automatic deployment of robotic teams. *IEEE Robotics & Automation Magazine*, 18(3):75–86, 2011.
[15] Matthew G Earl and Raffaello D'andrea. Iterative milp methods for vehicle-control problems. *IEEE Transactions on Robotics*, 21(6):1158–1167, 2005.
[16] Sertac Karaman and Emilio Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *Proc. of American Control Conference*, pages 2003–2009, 2008.
[17] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *Proc. of IEEE International Conference on Robotics and Automation (ICRA)*, pages 5041–5047, 2013.
[18] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: An open-source Robot Operating System. In *Workshop on Open-Source Software, ICRA*, 2009.
[19] 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.
[20] Yoshiaki Kuwata, Sertac Karaman, J Teo, E Frazzoli, JP How, and G Fiore. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.