MIT Open Access Articles

Sampling-based motion planning with deterministic u-calculus specifications

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation	S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In Proc. of IEEE Conference on Decision and Control, Dec. 2009.
As Published	http://dx.doi.org/10.1109/CDC.2009.5400278
Publisher	Institute of Electrical and Electronics Engineers
Version	Author's final manuscript
Citable link	http://hdl.handle.net/1721.1/57434
Terms of Use	Attribution-Noncommercial-Share Alike 3.0 Unported
Detailed Terms	http://creativecommons.org/licenses/by-nc-sa/3.0/

Massachusetts Institute of Technology

DSpace@MIT

Sampling-based Motion Planning with Deterministic μ-Calculus Specifications

Sertac Karaman Emilio Frazzoli

***Abstract*— In this paper, we propose algorithms for the on-line computation of control programs for dynamical systems that provably satisfy a class of temporal logic specifications. Such specifications have recently been proposed in the literature as a powerful tool to synthesize provably correct control programs, for example for embedded systems and robotic applications. The proposed algorithms, generalizing state-of-the-art algorithms for point-to-point motion planning, incrementally build finite transition systems representing a discrete subset of dynamically feasible trajectories. At each iteration, local μ-calculus model-checking methods are used to establish whether the current transition system satisfies the specifications. Efficient sampling strategies are presented, ensuring the probabilistic completeness of the algorithms. We demonstrate the effectiveness of the proposed approach on simulation examples.**

I. Introduction

The automatic generation of control programs that provably satisfy complex specifications on the system's behavior is a problem of great current interest, e.g., for the design and certification of high-confidence embedded and robotic systems, e.g., in automotive, aerospace, security, and medical applications. Various flavors of temporal logics, including, e.g., Linear Temporal Logic (LTL) [1], [2] and Metric Temporal Logic (MTL) [3], [4], have been shown to be not only powerful languages to express complex specifications, but also amenable to formal methods for control design.

A common approach to control design with temporal logic specifications is based on the construction of feedback controllers leading to an abstraction of the underlying physical system into a finite transition system (e.g., Kripke structures). For example, in [2], a partition of the state space is constructed, and control laws are designed that ensure direct transitions between neighboring cells in the partition. Model checking [5] is then performed on a negation of the specifications for such discrete transition system, thus synthesizing as a counter-example a control law governing transitions, ultimately satisfying the given specification.

The ability of these methods to synthesize a control program satisfying the specification depend heavily on the construction of the abstracted finite transition system. In other words, abstraction-based methods are, in general, not complete, since the choice of the abstraction constrains the achievable system's behaviors, and the synthesis procedure may not yield a control law satisfying the specifications even though one exists. Completeness may be achieved limiting the class of dynamical systems and/or specifications to highly-structured cases (e.g., $\omega-$regular properties can be checked for rectangular hybrid automata, which constitutes a maximal class of such systems [6], [7]). In addition, such methods rely on off-line computations to construct the finite transition system abstraction, and are not directly applicable, e.g., to dynamically changing environments.

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

In this paper, we propose a different approach, building on two main ideas. First, instead of relying on a fixed abstraction of the underlying dynamical system, we incrementally construct a finite transition system representing a discrete sample of the dynamically feasible trajectories for the system. This is done through a sampling procedure inspired by state-of-the-art methods originally designed to solve point-to-point motion planning problems in robotics [8]. Second, we propose incremental model checking methods, establishing in an efficient way whether the transition system at the current iteration is rich enough to satisfy the specification. In doing so, we concentrate on deterministic μ-calculus, a temporal logic that is known to (i) admit efficient model-checking algorithms, and (ii) be strictly more expressive than other linear time temporal logics used in the literature, including LTL.

The paper is structured as follows: In Section II, we formally introduce the syntax and semantics of the deterministic fragment of μ-calculus and provide the problem formulation. Section III contains the main contributions of the paper. More specifically, in III-A we present an efficient, probabilistically-complete incremental sampling-based algorithm to compute a control program for a dynamical system satisfying μ-calculus specifications. In III-B, we propose an incremental model checking algorithm, which complements the sampling-based algorithm. In Section IV, we discuss the effectiveness of the proposed algorithms through numerical experiments.

II. Background and problem formulation

Consider a discrete-time, time-invariant dynamical control system, described by the equations

$$z(i+1) = f(z(i), u(i)), \tag{1}$$

where $z : \mathbb{N} \to \mathbb{R}^n$ is the state trajectory, $u : \mathbb{N} \to [-1, 1]^m$, is the control law, and $f : \mathbb{R}^n \times [-1, 1]^m \to \mathbb{R}^n$ is Lipschitz. Let Π be a set, the elements of which are called the *atomic propositions*, usually denoted by $p_1, p_2, \ldots$, and let $L : \mathbb{R}^n \to 2^{\Pi}$ be a state-labeling function, which maps each state to the set of atomic propositions it satisfies.

Given an initial condition $z(0)$, it is desired to design a control law u such that the (infinite) state trajectory

z satisfies a given temporal logic specification Φ. In the following, we will provide some details of our choice of specification language, i.e., μ-calculus, and its semantics in terms of the system (1). In particular, we will define what we mean by saying that a control law u correctly implements a specification Φ on system (1).

A. Finite models of dynamical control systems

A common model used in computer science to check temporal properties of systems, such as reachability, safety, fairness, or liveness, is a class of finite transition systems called Kripke structures, which we formalize as follows.

Definition II.1 (Kripke Structure) *A Kripke structure $\mathcal{K}$, defined on a set Π of atomic propositions, is a tuple $\mathcal{K} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{L})$, where $\mathcal{S}$ is a finite set of states, $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, where for all $s \in S$ there exists $s' \in S$ such that $(s, s') \in \mathcal{R}$, is the transition relation, $\mathcal{L} : \mathcal{S} \to 2^{\Pi}$ is the labeling function.*

Even though the dynamical control system model (1), endowed with a state-labeling function, is richer than a Kripke structure, a subset of its possible behaviors can be modeled as Kripke structures. More precisely,

Definition II.2 (Finite models of dynamical systems) *A Kripke structure $K = (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L}^*)$ models the dynamical control system* (1)*, with initial condition $z(0)$, if (i) $\mathcal{S} \subseteq \mathbb{R}^n \cup \{\bowtie\}$; (ii) $s_0 = z(0) \in \mathcal{S}$; (iii) $(s, s') \in \mathcal{R}$ only if $s' = \bowtie$, or if $s \in \mathbb{R}^n$ and there exists $v \in \mathbb{R}^m$, $\|v\|_\infty \leq 1$, such that $s' = f(s, v)$; (iv) $\mathcal{L}(s) = L(s)$ for all $s \in \mathbb{R}^n$, and $\mathcal{L}(\bowtie) = \emptyset$.*

In other words, all transitions in the Kripke structure K can be mapped to trajectories for the system (1). (The converse is clearly not true.) The symbol $\bowtie$ represents a sink state, satisfying no atomic propositions, which allows to express finite-time trajectories using the Kripke structure formalism.

Given a model of the system as a Kripke structure, the model-checking question is to find those states/paths that satisfy a (temporal) logic formula on the set of atomic propositions, or prove that no such state/path exists. The logic formula is generally referred to as the *specification*, since the logic, in this case, constitutes a language to represent the specification, which itself encodes a desirable behavior of the system. Note that in our case, in which Kripke structures are finite models of dynamical control systems, if a state in a Kripke structure modeling the system satisfies a specification, so does the original system. On the other hand, if the state does not satisfy the specification, it may be the case that the particular Kripke structure is not a rich enough model of the system; a refinement of the Kripke structure may be required to prove that the specification can indeed be satisfied.

B. Temporal logics and the μ-calculus

Several specification logics have been proposed in the literature, including Computation Tree Logic (CTL), Linear Temporal Logic (LTL), and their superset CTL*. For this work, we concentrate on a form of temporal logic called deterministic μ-calculus. The reasons for our choice are that (i) μ-calculus admits very efficient model-checking algorithms, and (ii) it is very expressive, the full μ-calculus being a strict superset of other temporal logics such as those mentioned above. In fact, the deterministic μ-calculus is known to be able to express all ω-regular properties [9] (i.e., properties that can be stated on a Büchi automaton, see [10]). In the following, we briefly discuss a fragment of μ-calculus, which is particularly appropriate for robotics applications.

Let Var be a set of *variables*.We will commonly use the letters x and y to indicate variables. The syntax of the deterministic μ-calculus is defined as follows.

Definition II.3 (Deterministic μ-calculus) *Let Π and Var be two disjoint sets. The syntax of the deterministic μ-calculus is given in BNF form as follows:*

$$\phi := p \,|\, \neg p \,|\, x \,|\, p \wedge \phi \,|\, \neg p \wedge \phi \,|\, \phi \vee \phi \,|\, \Diamond\phi \,|\, \mu x.\phi \,|\, \nu x.\phi$$

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

Following [11], the set of all deterministic μ-calculus formulae will be denoted by L_1. The $\Diamond$ operator will be referred to as the *existential successor operator*, whereas the operators $\mu x.$ and $\nu x.$ will be called, respectively, the least and greatest *fixed-point operators*. The size of a μ-calculus formula Φ, denoted by $|\Phi|$, is defined as the total number of atomic propositions, variables, and operators in Φ.

The semantics of μ-calculus formulae is commonly defined on Kripke structures. Let $\mathcal{K} = (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L})$ be a Kripke structure defined of a set of atomic propositions Π. Given a formula $\phi \in L_1$, the subset of S for which ϕ holds will be denoted as $[\![\phi]\!]_{\mathcal{K}}$; a state $s \in [\![\phi]\!]_{\mathcal{K}}$ will be referred to as a ϕ-state. Moreover, let $\mathcal{K}_x^Q$, where $Q \subseteq \mathcal{S}$ and $x \in \mathrm{Var}$, be the Kripke structure $\mathcal{K}_x^Q = (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L}')$—defined on an augmented set of atomic propositions, namely $\Pi \cup \{x\}$—such that

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

Definition II.4 (Semantics of the deterministic μ-calculus) *Given a formula $\phi \in L_1$, the set $[\![\phi]\!]_{\mathcal{K}}$ is recursively defined as follows*[1]*:*

- $[\![p]\!]_{\mathcal{K}} = \{s \in \mathcal{S} \,|\, x \in \mathcal{L}(s)\}$, *for all* $p \in \Pi$,
- $[\![\neg p]\!]_{\mathcal{K}} = \{s \in \mathcal{S} \,|\, x \in \mathcal{L}(s)\}$ *for all* $p \in \Pi$,
- $[\![\phi \wedge \psi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}}$,
- $[\![\phi \vee \psi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}}$,
- $[\![\Diamond\phi]\!]_{\mathcal{K}} = \{s \in \mathcal{S} \,|\, \text{there exists } s' \in \mathcal{S} \text{ such that } (s, s') \in \mathcal{R} \text{ and } s' \in [\![\phi]\!]_{\mathcal{K}}\}$,
- $[\![\mu x.\phi]\!]_{\mathcal{K}}$ *is the least set Q such that $Q = [\![\phi]\!]_{\mathcal{K}_x^Q}$, i.e., $[\![\mu x.\phi]\!]_{\mathcal{K}}$ is such that $[\![\mu x.\phi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}_x^{[\![\mu x.\phi]\!]_{\mathcal{K}}}}$ and $\forall Q' \subseteq \mathcal{S}. \left[Q' = [\![\phi]\!]_{\mathcal{K}_x^{Q'}} \Rightarrow [\![\mu x.\phi]\!]_{\mathcal{K}} \subseteq Q' \right]$.*

[1]By convention, unary operators have precedence over binary operators.

- $[\![\nu x.\phi]\!]_{\mathcal{K}}$ = *is the greatest set* Q *such that* $Q = [\![\phi]\!]_{\mathcal{K}_x^Q}$, *i.e.,* $[\![\nu x.\phi]\!]_{\mathcal{K}}$ *is such that* $[\![\nu x.\phi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}_x^{[\![\nu x.\phi]\!]_{\mathcal{K}}}}$ *and* $\forall Q' \subseteq \mathcal{S}. \left[Q' = [\![\phi]\!]_{\mathcal{K}_x^{Q'}} \Rightarrow Q' \subseteq [\![\nu x.\phi]\!]_{\mathcal{K}} \right]$.

One of the main advantages of the μ-calculus is that its model-checking procedure is very simple and intuitive. Note that in the absence of the fixed-point and the existential successor operators the formulae can be evaluated at a given state, i.e., even without the knowledge of $\mathcal{R}$. Formulae with finite nesting of the existential successor operator can also be handled fairly easily by searching for a successor state that satisfies the subformula with the existential successor operator. The fixed-point operators, however, appear rather troublesome. Interestingly, they can also be model-checked easily. The following theorem suggest a natural global model-checking procedure for fixed-point formulae.

Theorem II.5 (Tarski-Knaster Theorem (see e.g., [12])) *Let $\mathcal{K}$ be a Kripke structure and ϕ be an L_1 formula. Moreover, let Q_i be defined recursively as follows:*

- $Q_0 = \emptyset$,
- $Q_i = [\![\phi]\!]_{\mathcal{K}_x^{Q_{i-1}}}$.

Then, (i) $Q_i \subseteq Q_{i-1}$, for all $i \in \mathbb{N}$, (ii) there exists a number $n \in \mathbb{N}$ such that $Q_n = Q_{n-1}$, and (iii) $Q_n = [\![\mu x.\phi]\!]_{\mathcal{K}}$. Furthermore, if $Q_i = \mathcal{S}$ in the above definition, then we have that (i) $Q_{i-1} \subseteq Q_i$, for all $i \in \mathbb{N}$, (ii) there exists a number $m \in \mathbb{N}$ such that $Q_m = Q_{m-1}$, and (iii) $Q_m = [\![\nu x.\phi]\!]_{\mathcal{K}}$.

This interpretation of the Tarski-Knaster fixed-point theorem follows from the fact that, for any μ-calculus formula ϕ, and any Kripke structure $\mathcal{K}$, the function $f(Q) := [\![\phi]\!]_{\mathcal{K}_x^Q}$, which maps $2^{\mathcal{S}}$ to itself, is a monotonic function, i.e., for any $P, Q \subseteq \mathcal{S}$, $P \subseteq Q$ implies that $f(P) \subseteq f(Q)$ (for the proofs and related discussion see, for example, [12]).

Deterministic μ-calculus is the fragment of modal μ-calculus, in which no branching property can be expressed. Rather than a limitation, this is desirable feature for motion planning problems, since the motion plan in the end is itself a "trajectory" respecting the linear flow of the time. Hence, by employing the deterministic μ-calculus we rule out all the branching-time specifications and focus only on those specifications for which a linear time trajectory can be generated. In terms of its expressive power, the deterministic μ-calculus is strictly more expressive than the Linear Temporal Logic [6], [9]. More precisely, L_1 is known to be equally expressive as the set of all $\omega-$regular properties. That is, any temporal property that can be expressed using, for instance, Büchi automata [10], can be expressed in L_1 (see for example [9] or [6] for constructive proofs). Hence, L_1 is indeed the most expressive regular language that can be used for the specification of linear time properties, which makes it the most expressive temporal logic for motion planning applications.

Despite its raw expressive power, the μ-calculus is not well accepted for direct use in practical applications due to its unnatural semantics. That is, unlike, for instance the temporal logics, long μ-calculus specifications are found to be quite hard to understand by inspection, and expressing temporal properties using μ-calculus, even though possible, is hard for humans. However, there are algorithms which convert a given temporal logic specification, e.g., in LTL, into a μ-calculus specification automatically (see for instance [9], [13]). To further introduce the μ-calculus, we present a few example formulae, which will be revisited in the next section after introducing sampling-based algorithms.

Examples

a) Reachability Specifications: Consider the μ-calculus formula $\Phi = \mu x.(p \vee \Diamond x)$. In words, Φ is satisfied by the smallest set of states, which, if labeled with x, would satisfy $p \vee \Diamond x$. Notice that such set is the set Q of all states, which either satisfy p or can reach a state that satisfies p. One way to see this is to carry out the iteration in the Tarski-Knaster theorem: Q_1 is the set of states that satisfy p, Q_2 is the set of states that either satisfy p (that are in Q_1) or that have an outgoing edge to a state that satisfies p, Q_3 is the set of states that are either in Q_2 or have a transition to a state in Q_2, etc. This iteration converges to the set of all states, from which there is a trajectory leading to a state that satisfies p. Another, perhaps more intuitive look at Φ is the following. First, note that such set of states is indeed a fixed point, i.e., if Q is labeled with x, then the set of states that satisfy $p \vee \Diamond x$ would be Q itself. That is, all the states in Q satisfy $p \vee \Diamond x$ and no other state outside Q satisfy $p \vee \Diamond x$. The former statement is true, since each state in Q either satisfy p or has a transition to a state that satisfies x. The latter one is also correct, since if there is any state s' that is not in Q but it satisfies $p \vee \Diamond x$, then it either has to satisfy p or it has to have a transition to a state which is labeled with x. In any case, s' would have a path that reaches Q and thus reaches a state that satisfy p. Hence, Φ essentially defines a *reachability property*, ensuring the reachability to a state that satisfies p.

b) Safety Specifications: Next, consider $\Phi = \nu x.(q \wedge \Diamond x)$. In words, this formula is satisfied by the largest set Q of states that both satisfy q and has a transition to a state with the same property. Hence, for any state in Q, there must be cycle of states, all of which satisfy q.

c) Safely Reaching a Region: The standard motion planning objective is, for instance, to avoid obstacles and reach a goal state. Let us label the goal states with p and the obstacles with q. Then, the specification $\Phi = \mu x.(\neg q \wedge (p \vee \Diamond x))$ is the smallest set of states for which there exists a trajectory reaching state that satisfies p (a goal state) and along the way never goes through a state that satisfies q (an obstacle).

d) Reaching a Safe Region: Another example is the one in which the specification is to eventually reach a point where a property p can be retained forever. Essentially, this can be done easily by merging the first two examples together as in $\Phi = \mu x.((\nu y.(p \wedge \Diamond y)) \vee \Diamond x)$.

e) Ordering Specifications: A common specification is, for instance, to ensure some property p until another

property q is attained. In this case, a corresponding μ-calculus specification is $\mu x.(p \vee (q \wedge \Diamond x))$, which intuitively states that either q is satisfied or there is a next state which satisfies p and the property $p \vee (q \wedge \Diamond x)$.

f) Liveness Specifications: Consider a final example, where it is desired to satisfy a property p infinitely often. That is, at all points along the path, p is satisfied in the future. One way to specify such a behavior is to use $\nu y.\mu x.\Diamond((p \wedge y) \vee x)$, which intuitively states that in the next states either the property p is satisfied, or there is a path to a state which satisfies p (stated via the disjunction and the $\mu x.$ operators). Moreover, this statement is true at all times (stated via the conjunction and the $\nu y.$ operators).

C. Problem Formulation

At this point, we can relate discrete-time dynamical systems to μ-calculus specifications as follows.

Definition II.6 *A dynamical control system of the form* (1)*, endowed with a state-labeling function L, is said to satisfy a μ-calculus specification Φ at some initial state x_0 if and only if there exists a Kripke structure $\mathcal{K}^* = (\mathcal{S}^*, \{s_0\}, \mathcal{R}^*, \mathcal{L}^*)$ modeling the system, and such that $s_0 \in [\![\Phi]\!]_{\mathcal{K}^*}$.*

Hence, by definition, a discrete-time dynamical system satisfies a given μ-calculus specification if one can construct a Kripke structure $\mathcal{K}^*$ from a finite subset of its state space, such that $\mathcal{K}^*$ respects the state transitions of the dynamical system described by Equation (1) and x_0 satisfies Φ in $\mathcal{K}$.

Given the above definition, the motion planning problem with μ-calculus specifications can be stated as follows:

Problem II.7 *Given a discrete-time dynamical control system* (1)*, and an L_1 formula Φ, determine whether or not the dynamical system satisfies Φ. If yes, return a control law u implementing the specification. If not, return failure.*

It is worth mentioning at this point that even though this problem definition is not directly related to motion planning, we will show in the next sections that satisfaction of a μ-calculus specification can be related to a "path", which in turn can be used as the motion plan with desired properties.

III. Planning Algorithms

In principle, Problem II.7 could be addressed using the following iterative procedure: (i) Choose a finite set of states (including the initial condition) from $\mathbb{R}^m$, e.g., by random sampling; (ii) Construct a Kripke structure modeling the system (1) from this set of states and the sink state $\bowtie$, i.e., by determining which state pairs are in $\mathcal{R}$, and defining the appropriate labeling function, as in Def. II.6; (iii) Check whether this model satisfies the specification Φ, using, e.g., the Tarski-Knaster iteration [12]. If not, repeat the procedure adding more states into the Kripke structure.

The soundness of the procedure above is a consequence of the following technical lemma, which will be proved in the appendix.

Lemma III.1 *Let $\mathcal{K} = (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L})$ and $\mathcal{K}' = (\mathcal{S}', \{s_0\}, \mathcal{R}', \mathcal{L}')$ be two Kripke structures such that (i) $\mathcal{S} \subseteq \mathcal{S}'$, (ii) $\mathcal{R} \subseteq \mathcal{R}'$, (iii) for all $s \in \mathcal{S}$, $\mathcal{L}(s) \cap \Pi = \mathcal{L}'(s) \cap \Pi$. Then, for any L_1 formula Φ, $[\![\Phi]\!]_{\mathcal{K}} \subseteq [\![\Phi]\!]_{\mathcal{K}'}$.*

Intuitively, the lemma states that any Kripke structure $\mathcal{K}'$ that can be constructed from $\mathcal{K}$ by adding extra states and transitions satisfies the same set of L_1 formulae as $\mathcal{K}$ at all the states that are common to both structures. Even though this seems intuitive, this property can easily be shown not to be true for, for instance, the full μ-calculus.

In order to make the iterative procedure effective, it is necessary to ensure that samples are chosen in such a way that the resulting Kripke structure models a rich set of trajectories of the dynamical system. Moreover, it is desirable that the complexity of checking whether a Kripke structure satisfies a specification at each iteration only depend on the number of states added at that iteration, not on the total number of states in the finite model. We will address these two points next.

A. Sampling-based Kripke Structures

Sampling-based algorithms have been recently proposed as a very efficient approach to robotic motion planning. Such algorithms, e.g., Probabilistic RoadMaps (PRM) [14], and Rapidly-exploring Random Trees (RRT) [15] effectively build a finite transition system modeling a dynamical system, and check whether such finite transition contains a trajectory from the initial state to a desired goal state. In PRMs, states are chosen randomly, independently and identically from a given distribution; moreover, PRMs are typically undirected graphs, and are not directly applicable to general dynamical systems. On the other hand, RRTs are constructed as directed trees, and new states/transitions are added in a way that efficiently probes the set of all feasible trajectories of a general dynamical system. Unfortunately, since RRTs are directed trees—and thus unable to express cyclic trajectories—they cannot serve as finite models of trajectories satisfying general ω-regular properties.

In order to combine efficient exploration with the ability to satisfy general ω-regular properties, we propose an extension of the RRT algorithm, to which we refer as Rapidly-exploring Random Graph (RRG). Before giving the full algorithm, let us introduce some necessary components.

Distance function: Let $\mathrm{Dist} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ be a continuous function, with $\mathrm{Dist}(u, v) = 0$ iff $v = f(u, 0)$.

Sample generation: Let $\mathrm{Sample} : \mathbb{N} \to \mathbb{R}^n$ be a function that generates independent, identically distributed samples from a distribution S supported on $\mathbb{R}^n$ (or a subset containing all states that can possibly satisfy the specification).

Local steering: Let $\mathrm{Steer} : \mathbb{R}^n \times \mathbb{R}^n \to [-1, 1]^m$ be a function that computes a control input moving the initial state "closer" to a desired state. More precisely, if $v = \mathrm{Steer}(s_1, s_2)$, with $\mathrm{Dist}(s_1, s_2) > 0$, then $\mathrm{Dist}(f(s_1, v), s_2) < \mathrm{Dist}(s_1, s_2)$. The function Steer^- is similarly defined, backwards in time. In other words, let $f^- : \mathbb{R}^n \times [-1, 1]^m \to \mathbb{R}^n$ be such that, if $s' = f(s, v)$,

then $s = f^-(s', v)$. Then, if $v = \text{Steer}^-(s_2, s_1)$, with $\text{Dist}(s_1, s_2) > 0$, then $\text{Dist}(f^-(s_2, v), s_2) < \text{Dist}(s_1, s_2)$. Finally, we assume that the local steering functions provide exact one-step steering when feasible. In other words, if s_2 is reachable in one step from s_1, then $f(s_1, \text{Steer}(s_1, s_2)) = s_2$, and $s_1 = f^-(s_2, \text{Steer}^-(s_2, s_1))$.

The main body of the RRG algorithm is given in Alg. 1. We incrementally build a Kripke structure modeling the system (1), initializing it to a trivial structure containing only the initial condition. At each step, we first check whether the current Kripke structure satisfies the specification Φ (an efficient algorithm to perform this computation will be given in the next section). If not, a sample is chosen, based on which new states/transitions are added to the Kripke structure, until the specification is satisfied. Note that the algorithm may not terminate, unless a trajectory of (1) exists that satisfies Φ, and can be expressed by the Kripke structure at some iteration.

Algorithm 1: $\text{RRG}(\Phi, z(0))$

$s_0 \leftarrow z(0)$, $\mathcal{S} \leftarrow \{s_0, \bowtie\}$, $\mathcal{R} \leftarrow \{(s_0, \bowtie)\}$, $i \leftarrow 0$;
while $s_0 \notin [\![\Phi]\!]_{\mathcal{K}}$ **do**
  $\mathcal{K} \leftarrow (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L})$;
  $q \leftarrow \text{Sample}(i)$, $i \leftarrow i + 1$;
  $(\mathcal{S}^+, \mathcal{R}^+) \leftarrow \text{Expand}(\mathcal{K}, q)$;
  $(\mathcal{S}^-, \mathcal{R}^-) \leftarrow \text{Expand}^-(\mathcal{K}, q)$;
  $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}^+ \cup \mathcal{S}^-$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}^+ \cup \mathcal{R}^-$;
Return $\mathcal{K}$;

New states/transitions are added to the Kripke structure using the Expand procedure outlined in Alg. 2. Essentially, for each sample q, we find the nearest neighbor s^* already in the Kripke structure, and attempt to reach q from this state. Then we repeat the same procedure, considering only states in the Kripke structure, and in the intersection of half-spaces containing q but not previously-considered nearest neighbors, until there are no more such states. A similar function, Expand^-, can be defined working backwards in time (i.e., using the Steer^-function); in addition, in Expand^- we do not create any transitions to states that are not reachable from the initial state.

Algorithm 2: $\text{Expand}(\mathcal{K} = (\mathcal{S}, s_0, \mathcal{R}, \mathcal{L}), q)$

$\mathcal{C} \leftarrow \mathbb{R}^n$, $\mathcal{S}' \leftarrow \emptyset$, $\mathcal{R}' \leftarrow \emptyset$;
while $\mathcal{S} \cap \mathcal{C} \neq \emptyset$ **do**
  $s^* = \arg\min_{s \in \mathcal{S} \cap \mathcal{C}} \text{Dist}(s, q)$;
  $s' \leftarrow \text{Steer}(s^*, q)$;
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{s'\}$, $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{(s^*, s'), (s', \bowtie)\}$;
  Remove from $\mathcal{C}$ a halfspace $H(s^*, q)$ containing s^* but not q, i.e., $\mathcal{C} \leftarrow \mathcal{C} \setminus H(S^*, q)$;
Return the sets $\mathcal{S}'$ and $\mathcal{R}'$;

The RRG algorithm yields a Kripke structure that contains an RRT-like tree, with the addition of edges generating cycles. The algorithm inherits the probabilistic completeness of RRTs, i.e., if there exists a Kripke structure $\mathcal{K}^*$ as given in Definition II.6, under mild technical conditions, the algorithm finds it with high probability as the number of samples increases. More formally,

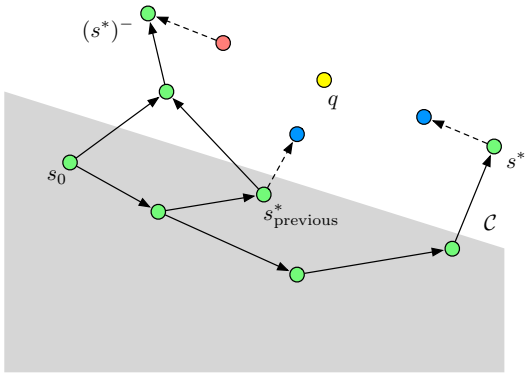

Fig. 1. Illustration of the RRG algorithm. Blue and red circles represent new states in $\mathcal{S}^+$ and $\mathcal{S}^-$, respectively.

Theorem III.2 *Consider a time-invariant, discrete-time dynamical system* (1)*, satisfying a* L_1 *specification* Φ*. Assume that (i) there exists a Kripke structure* $\mathcal{K}^* = (\mathcal{S}^*, \{s_0\}, \mathcal{R}^*, \mathcal{L}^*)$*, which models an overconstrained version of* (1)*, in which the control must lie in* $[-\eta, \eta]^m$*,* $\eta \in (0, 1)$*, (ii)* $z(0) \in [\![\Phi]\!]_{\mathcal{K}^*}$*, (iii) there exists* $\epsilon > 0$ *such that for all* $s \in \mathcal{S}$*, and* $z \in \mathbb{R}^n : \|z - s\| \leq \epsilon$*,* $L(s) = L(z)$*. Then the RRG algorithm terminates with an output* $\mathcal{K} = (\mathcal{S}, \{s_0\}, \mathcal{R}, \mathcal{L})$*, such that* $z(0) \in [\![\Phi]\!]_{\mathcal{K}}$*, with probability approaching one, as the number of samples increases.*

Proof: (Sketch) It is a well-known fact that (infinite) trajectories satisfying ω-regular properties (such as L_1 specifications) can be decomposed into a finite-length prefix and a (possibly trivial) finite-length loop that is repeated infinitely often [12]. Hence, a finite number of (ordered) states are sufficient to completely characterize a solution. Let us call this set of states $\bar{\mathcal{S}}$. Based on the assumptions on the theorem, and the continuity of the function f in the dynamical system, one can define a finite sequence (respectively, loop) of neighborhoods around the states in $\bar{\mathcal{S}}$ such that all points in a neighborhood can reach in one step all points in the next neighborhood (modulo the loop length if appropriate). Reasoning by induction, one can show that (i) there is a finite probability that at the i-th iteration, the sample q will be generated in the neighborhood of the first state in $\bar{\mathcal{S}}$ after the initial condition $z(0)$, thus adding a state in that state's neighborhood; (ii) assuming that the Kripke structure at the i-th iteration already includes transitions up to the neighborhood of the k-th state in the solution, there is a finite probability that the sample q will be generated in the neighborhood of the $(k+1)$-th state, thus adding a new state there. Hence, as the number of samples goes to infinity, the probability that the set of states in the incrementally-

built Kripke structure $\mathcal{K}$ does not contain all the states in $\bar{\mathcal{K}}$ vanishes. ■

The above theorem does not state bounds on, e.g., the rate at which the probability of success converges to one. However, we refer interested readers to the literature on RRTs and incremental sampling methods (e.g., [15]) for some additional insights on the matter.

B. Incremental Model Checking

In this section, we first present a simple local model checking procedure for deterministic μ-calculus. Then, we extend this algorithm to an incremental model checker, where new states or transitions can be added to the Kripke structure without necessarily running the whole model checking procedure all over. This type of model checking procedure well suits the sampling based algorithm presented in the previous section.

A local model checking procedure for the deterministic fragment of the μ-calculus is presented in Algorithm 3, which checks whether or not the initial state of a given Kripke structure $\mathcal{K}$ satisfies a given L_1 formula Φ. The algorithm also assumes a global data structure, called `Stack`, which is essentially a set that stores state and L_1 formula pairs, i.e., $\texttt{Stack} \subseteq \mathcal{S} \times L_1$. Moreover, the algorithm uses the function $\texttt{BndFormula}(x)$ which maps the set Var of variables to the subformula of the form $\sigma x.\phi$, i.e., the subformula that x is bound by in Φ.

Lemma III.3 *The Algorithm* $\texttt{ModelCheck}(s_0, \Phi)$ *returns* `True` *if and only if* $s_0 \in [\![\Phi]\!]_{\mathcal{K}}$ *holds.*

Algorithm 3 is very similar to the local model checking algorithm that appears in [12] and has many common grounds with the global algorithm provided in [11]. The proof of Lemma III.3 is very similar to the correctness proofs of the procedures given in those references, hence this proof will not be carried out here. Even though we do not provide full proofs here for the sake of brevity, Lemma III.3 can be used to prove the correctness of the incremental model checking algorithm, which will be outlined shortly.

Given an L_1 formula ϕ, let $\texttt{SF}(\phi)$ be the set of all subformulae of ϕ. For incremental model checking purposes, we maintain a graph $G = (V, E)$, where $V \subseteq \mathcal{S} \times \texttt{SF}(\phi)$ is called the set of nodes and $E \subseteq V \times V$ is the set of edges. Given two nodes, $v = (s, \psi)$ and $v' = (s, \psi')$ in V, there exists an edge (v, v') in E, if one of the following holds:

- $\psi = p \wedge \psi'$ and $p \in \mathcal{L}(s)$,
- $\psi = \neg p \wedge \psi'$ and $p \notin \mathcal{L}(s)$,
- $\psi = \psi' \vee \psi''$ (for some ψ''),
- $\psi = \mu x.\psi'$ or $\psi = \nu x.\psi'$,
- $\psi = x$ where $x \in \text{Var}$ and $\psi' = \mu x.\psi''$ or $\psi' = \nu x.\psi''$ for some ψ''.

Let $v' = (s', \psi)$, then there exists an edge (v, v') in E if the following holds:

- $\psi = \Diamond\psi'$ and $(s, s') \in \mathcal{R}$.

Algorithm 3: $\text{ModelCheck}(\mathcal{K}, \Phi, s, \phi)$

```
switch φ do
    case p where p ∈ Π
        return p ∈ L(s)
    case ¬p where p ∈ Π
        return p ∉ L(s)
    case p ∧ φ
        return p ∧ ModelCheck(s, φ)
    case ¬p ∧ φ
        return ¬p ∧ ModelCheck(s, φ)
    case φ ∨ ψ
        return
        ModelCheck(s, φ) ∨ ModelCheck(s, ψ)
    case ◇φ
        for ∀s' ∈ suc(s) do
            if ModelCheck(s', φ) then
                return True
        return False
    case σx.φ where σ ∈ {μ, ν}
        Stack := Stack ∪ {(s, φ)}
        Value := ModelCheck(s, φ)
        Stack := Stack \ {(s, φ)}
        return Value
    case x where x ∈ Var
        if (s, BndFormula(x)) ∈ Stack then
            switch BndFormula(x) do
                case μx.φ
                    return False
                case νx.φ
                    return True
        else
            return ModelCheck(s, BndFormula(x))
```

Notice that these conditions resemble the branching conditions in Algorithm 3. Intuitively, there is an edge between two nodes (s, ψ) and (s', ψ') in $G = (V, E)$, if $\texttt{ModelCheck}(\mathcal{K}, \Phi, s, \psi)$ calls $\texttt{ModelCheck}(\mathcal{K}, \Phi, s', \psi')$.

The incremental model checking algorithm also maintains a reachability relation $\texttt{Reaches} \subset V \times V$, where $(v, v') \in \texttt{Reaches}$ and $v = (s, \psi)$, implies that ψ is a $\nu-$formula, i.e., ψ is of the form $\psi = \nu x.\psi'$, and that there is a path from v to v' in E.

Given a Kripke structure $\mathcal{K}$, adding a state s in to $\mathcal{K}$ is done by updating $\mathcal{S}$ as $\mathcal{S} \cup \{s\}$ and adding (s, ψ) into V, for all (s, ψ'). Moreover, edges (v, v'), where $v = (s, \psi)$ and $v' = (s, \psi')$, are added into E according to the definition outlined above. Let $v = (s, \psi)$ and $v' = (s', \psi')$ be two nodes in V. When a transition (s, s') is added into $\mathcal{R}$, we add (v, v') into E if $\psi = \Diamond\psi'$. After adding each edge (v, v') to E, the reachability relation $\texttt{Reach}$ is updated with $(\bar{v}, v')$, if there exists a node $\bar{v} = (\bar{s}, \bar{\psi})$ in V such that $\bar{\psi}$ is a $\nu-$formula and there is a path on $G = (V, E)$, which does not cross a node (s'', ψ''), where ψ'' is a $\mu-$formula bigger

than $\bar{\psi}$.

A given L_1 specification is satisfied if and only if there exists two nodes $v = (s, \psi)$ and $v' = (s, \psi')$ in V such that $\psi' = x$ with $x \in \text{Var}$ and $\psi = \nu x.\varphi$ for some φ and that (i) there is a path from v to v' in $G = (V, E)$, (ii) this path does not cross any node $v'' = (s'', \psi'')$, where ψ'' is a $\mu-$formula that is larger than ψ, and (iii) this path is reachable from the node (s_0, ϕ), where s_0 is the initial state and ϕ is the specification.

Precisely speaking, one has to incrementally maintain the relation `Reach` as well the set of those nodes in V that can be reached from (s_0, ϕ). Let us note that maintenance of the reachability relation of a graph is a problem of particular interest in incremental computation (see, for instance, [16]). Although in the case of adding and deleting edges its incremental time complexity is known to be unbounded, if the case where only adding edges is considered it is one of the easiest problems in incremental computation (see [16] for further discussion and proofs).

IV. Simulation Results

In this section, we provide an illustrative example. We consider a linear discrete time dynamical system with $z(i+1) = Az(i) + Bu(i)$, where

$$A = \begin{bmatrix} 1.019 & -0.029 \\ 0.049 & 0.95 \end{bmatrix}, \quad B = \begin{bmatrix} 0.101 & -0.0015 \\ 0.0025 & 0.098 \end{bmatrix}.$$

The initial condition is $z_0 = [0, 0]$.

The specification requires the system to visit two distinct subsets R_1 and R_2 of the state-space infinitely often while avoiding a large 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 $p_i \in L(s)$ if $s \in R_i$ for all $i = 1, 2, 3$. This specification can be given in the deterministic $\mu-$calculus as

$$\mu w.((\neg p_3 \wedge \Diamond w) \vee \nu z.\{(p_2 \wedge \mu x.[\neg p_3 \wedge ((p_1 \wedge z) \vee \Diamond x)]) \vee (p_1 \wedge \mu y.[\neg p_3 \wedge ((p_2 \wedge z) \vee \Diamond y)])\})$$

The algorithm described in this paper takes about 3.5 seconds to solve this example sampling slightly more than 1000 states and exploring close to 6000 nodes. The solution trajectory as well as the parts of the state-space that were explored are shown in Figure 2. The graph produced by the RRG algorithm is depicted in Figure 3, while searching for this solution.

In Figure 4, an example run on a system with linear dynamics $z(i+1) = Az(i) + Bu(i)$, where A and B are identity matrices, is considered. The layout of the regions in the state-space is the same as the previous example. The algorithm took less than 0.1 seconds to find the answer exploring about 350 nodes.

We have also run some limited experiments on similar problems in higher-dimensional spaces (up to 12), obtaining computation time of the order of a few seconds. Further investigation of the performance of the algorithm in high-dimensional spaces will be the objective of future work.

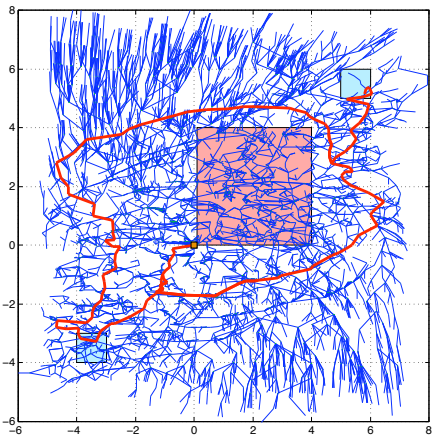

Fig. 2. The part of the state space explored and the trajectory that satisfies the specification. Regions R_1 and R_2 are shown in red in the upper right and lower left corners. Region R_3 is the red rectangular region in the middle.

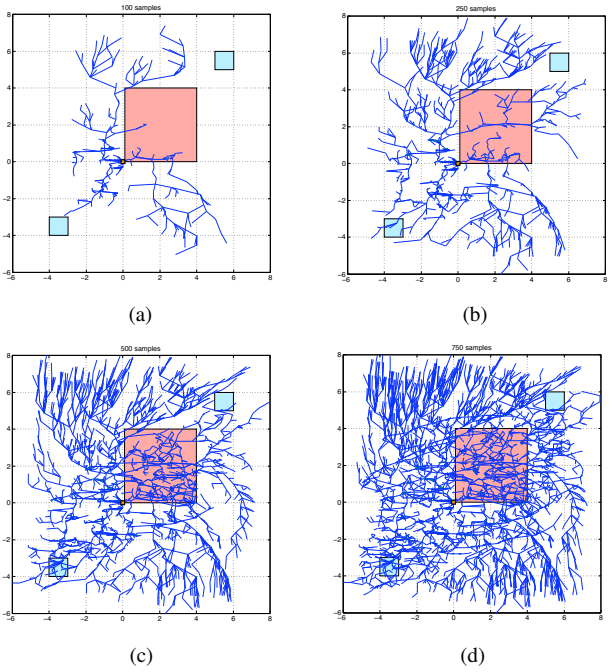

Fig. 3. Demonstration of the RRG algorithm on the example.

V. Conclusions

In this paper, we propose an incremental, sampling-based methodology for the generation of motion plans for dynamical systems that provably satisfy temporal logic specifications. In particular, we concentrate on specifications expressed in the deterministic μ-calculus, which is a superset of other well-known linear temporal logic formulas which have been extensively used, e.g., for robotics applications. Our approach is based on two steps: (i) a sampling-based generation of finite transition systems modeling a subset of the possible system trajectories, and (ii) an incremental model-checking algorithm that can establish whether the current model of the system is rich enough to express

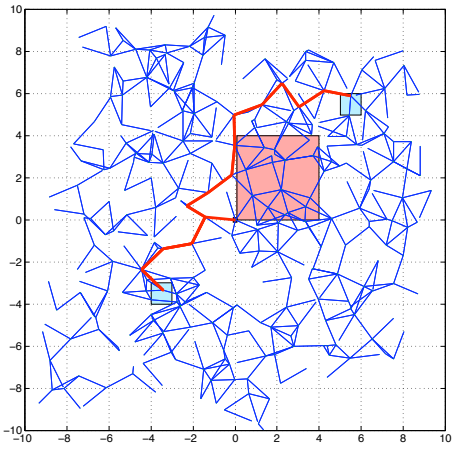

Fig. 4. Simulation with simple dynamics.

behaviors satisfying the specification. Numerical experiments suggest that the proposed approach is fast enough for on-line implementation in robotics and embedded systems, even in high-dimensional problems. Future work will include extensions to address feedback control policies, reactive planning, efficient sampling methods, and trajectory optimization.

ACKNOWLEDGMENTS

The authors are grateful to Dr. Amit Bhatia for several inspiring discussions about motion planning algorithms and their possible extensions to handle complex specifications. This research was done with support from the Michigan/AFRL Collaborative Center on Control Sciences, AFOSR grant no. FA 8650-07-2-3744. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the supporting organizations.

REFERENCES

[1] P. Tabuada and G.J. Pappas. Linear temporal logic control of discrete-time linear systems. *IEEE Trans. Automatic Control*, 14(1):61–70, 2006.

[2] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. Automatic Control*, 53(1):287–297, 2008.

[3] S. Karaman and E. Frazzoli. Vehicle routing problem with metric temporal logic specifications. In *IEEE Conference on Decision and Control*, 2008.

[4] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[5] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. Springer, 1999.

[6] T.A. Henzinger, R. Majumdar, and J. Raskin. A classification of symbolic transition systems. *ACM Transactions on Computational Logic*, 6(1):1–32, 2005.

[7] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What is decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[8] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control and Dynamics*, 25(1):116–129, 2002.

[9] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model checking for the mu-calculus and its fragments. *Theoretical Computer Science*, 258:491–522, 2001.

[10] W. Thomas. *Handbook of Theoretical Computer Science*, chapter Automata on Infinite Objects. Elsevier Science, 1990.

[11] E. Emerson, C. Jutla, and A. Sistla. On model-checking for the fragments of mu-calculus. In *CAV 93: Computer-aided Verification*, 1993.

[12] K. Schneider. *Verification of Reactive Systems*. Springer, 2004.

[13] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[14] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[15] S. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[16] G. Ramalingam. *Bounded Incremental Computation*. Number 1089 in Lecture Notes in Computer Science. Springer, 1996.

APPENDIX

Proof: [Lemma III.1] The proof is an induction on the size of the formula Φ. For $|\Phi| = 1$, we have that Φ must be of the form $\Phi = p$, where $p \in \Pi$. In this case, for all $s \in \mathcal{S}$, we have $s \in [\![p]\!]_{\mathcal{K}} \Leftrightarrow p \in \mathcal{L}(s)$, which is itself then equivalent to $p \in \mathcal{L}'(s)$ (by Condition (iii.a) of the lemma), which in turn is equivalent to $s \in [\![\Phi]\!]_{\mathcal{K}'}$ by the semantics.

Assume that the hypothesis holds for all L_1 formulae of size $n-1$. For the induction step, let us consider all the different possible cases. Let Φ be of the form $\neg\phi$, which can only happen if $\Phi = \neg p$, where $p \in \Pi$, following the syntax of L_1. This case is very similar to the base case, i.e., $s \in [\![\neg p]\!]_{\mathcal{K}} \Leftrightarrow p \notin \mathcal{L}(s) \Leftrightarrow p \notin \mathcal{L}'(s) \Leftrightarrow s \in [\![\neg p]\!]_{\mathcal{K}'}$ for all $s \in \mathcal{S}$. Let Φ be of the form $\Phi = \phi \vee \psi$, then, by the induction hypothesis, there holds $[\![\phi]\!]_{\mathcal{K}} \subseteq [\![\phi]\!]_{\mathcal{K}'}$ and $[\![\psi]\!]_{\mathcal{K}} \subseteq [\![\psi]\!]_{\mathcal{K}'}$, using which we conclude $[\![\phi \vee \psi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}'} \subseteq [\![\phi]\!]_{\mathcal{K}'} \cup [\![\psi]\!]_{\mathcal{K}'} = [\![\phi \vee \psi]\!]_{\mathcal{K}'}$. For the case when $\Phi = \phi \wedge \psi$, the same fact can be used to deduce $[\![\phi \wedge \psi]\!]_{\mathcal{K}} = [\![\phi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}'} \subseteq [\![\phi]\!]_{\mathcal{K}'} \cap [\![\psi]\!]_{\mathcal{K}'} = [\![\phi \wedge \psi]\!]_{\mathcal{K}'}$. Consider the case $\Phi = \Diamond\phi$. Then, for all $s \in \mathcal{S}$, we have that $s \in [\![\Diamond\phi]\!]_{\mathcal{K}}$ is equivalent to $\exists \tilde{s} \in \mathcal{S}$ such that $(s, \tilde{s}) \in \mathcal{R}$ and $\tilde{s} \in [\![\psi]\!]_{\mathcal{K}}$. Note that since $\tilde{s} \in \mathcal{S}$ and $(s, \tilde{s}) \in \mathcal{R}$, we have that $\tilde{s} \in \mathcal{S}'$ and $(s, \tilde{s}) \in \mathcal{R}'$, by Conditions (i) and (ii) of the lemma. Hence, the last statement implies that there exists $\tilde{s} \in \mathcal{S}'$ such that $(s, \tilde{s}) \in \mathcal{R}'$. Moreover, by the induction hypothesis, we have $s' \in [\![\phi]\!]_{\mathcal{K}}$ implies that $\tilde{s} \in [\![\phi]\!]_{\mathcal{K}'}$. These statements together are equivalent to $s \in [\![\phi]\!]_{\mathcal{K}'}$, which finally establishes $[\![\phi]\!]_{\mathcal{K}} \subseteq [\![\phi]\!]_{\mathcal{K}'}$. Consider the case, in which ϕ is of the form $\Phi = \mu x.\phi$. To prove this case we show that the sets $Q_0 := \emptyset$, $Q_i := [\![\phi]\!]_{\mathcal{K}_x^{Q_{i-1}}}$ as well as $Q'_0 := \emptyset$, $Q'_i := [\![\phi]\!]_{\mathcal{K}'^{Q'_{i-1}}_x}$ satisfy $Q_i \subseteq Q'_i$ for all i. Noting that, by the Tarski-Knaster Theorem, Q_i and Q'_i converge to $[\![\mu x.\phi]\!]_{\mathcal{K}}$ and $[\![\mu x.\phi]\!]_{\mathcal{K}'}$, respectively, this result will imply that $[\![\mu x.\phi]\!]_{\mathcal{K}} \subseteq [\![\mu x.\phi]\!]_{\mathcal{K}'}$. Hence, it remains to show that $Q_i \subseteq Q'_i$ for all i. To show this property let us consider an inner induction on the number i. For the base case, $i = 0$, the statement holds trivially. In the induction step, noting that $Q_{i-1} \subset Q'_{i-1}$ holds by the induction hypothesis, we have that $\mathcal{K}_x^{Q_i}$ and $\mathcal{K}'^{Q'_{i-1}}_x$ satisfy all the conditions of the lemma. Hence, we have $Q_i = [\![\phi]\!]_{\mathcal{K}_x^{Q_i}} \subset [\![\phi]\!]_{\mathcal{K}'^{Q'_i}_x} = Q'_i$, where the set inclusion is by the induction hypothesis of the outer induction since ϕ is of size $n-1$. The case when $\Phi = \nu x.\phi$ is very similar to the previous case, hence we omit that part of the proof here for the sake of brevity. ■