Centrum Wiskunde & Informatica

REPORT*RAPPORT*

MAS

Modelling, Analysis and Simulation

Modelling, Analysis and Simulation

Computable CTL for discrete-time and continuous-space dynamic systems

P.J. Collins, I.S. Zapreev

REPORT MAS-E0903 FEBRUARY 2009

Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Science Park 123, 1098 XG Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-3703

Computable CTL for discrete-time and continuous-space dynamic systems

ABSTRACT
Dynamic systems are widely applied for modelling and analysis in physiology, biology, chemistry and engineering. The high-profile and safety-critical nature of such applications has resulted in a large amount of work on formal methods for dynamic systems: mathematical logics, computational methods, formal verification, and etc. In our work, we focus on the verification approach called model checking, and its computability aspects. In this approach, a desired system property, specified using some logical formalism, is verified against the dynamic-system model via an exhaustive state-space exploration. This process typically involves computation of reachable and/or chain-reachable sets that in certain cases can not be obtained due to the continuity of state-space domain. Therefore, in this paper, we use topological approach along with the computability results of Type Two theory of Effectivity in order to construct a computable CTL semantics for discrete-time and continuous-space dynamic systems.

2000 Mathematics Subject Classification: 68Q60 68Q17
1998 ACM Computing Classification System: D.2.4
Keywords and Phrases: Computability; Model Checking; CTL; Dynamic Systems
Note: This work was carried out under the project MAS2.2 - "Topological Methods for Systems and Control"

Computable CTL for Discrete-Time and Continuous-Space Dynamic Systems

Pieter Collins and Ivan S. Zapreev
Pieter.Collins@cwi.nl, I.Zapreev@cwl.nl

February 3, 2009

Abstract

Dynamic systems are widely applied for modelling and analysis in physiology, biology, chemistry and engineering. The high-profile and safety-critical nature of such applications has resulted in a large amount of work on formal methods for dynamic systems: mathematical logics, computational methods, formal verification, and etc. In our work, we focus on the verification approach called model checking, and its computability aspects. In this approach, a desired system property, specified using some logical formalism, is verified against the dynamic-system model via an exhaustive state-space exploration. This process typically involves computation of reachable and/or chain-reachable sets that in certain cases can not be obtained due to the continuity of state-space domain. Therefore, in this paper, we use topological approach along with the computability results of Type Two theory of Effectivity in order to construct a computable CTL semantics for discrete-time and continuous-space dynamic systems.

Keywords: Computability, Model Checking, CTL, Dynamic Systems

1 Introduction

A dynamic system describes the system-state evolution over time. Sometimes the system has inputs and/or outputs and then it is called a control system. The inputs may represent either uncontrollable disturbances or control signals, and the outputs may represent quantities that either need to be measured or controlled. Dynamic systems are often used in modelling and analysis of processes in physiology, biology, chemistry and engineering. The commonly known application fields are air traffic control, automated manufacturing, chemical process control, and etc. Verification of performance requirements for dynamic systems is essential for designing systems, such as power plants and robots, and analysing natural phenomena, such as chemical reactions and biological processes.

Given a formal model $\mathcal{M}$ of a system design, along with a specification formula ϕ that represents a desired system property, formal verification answers the question: "Does the model $\mathcal{M}$ satisfy the property ϕ?". This question is typically put as a formula $\mathcal{M} \models \phi$, that uses a satisfiability relation $\models$.

The system model $\mathcal{M}$, representing the dynamic system, can be described using various formalisms, such as time-automaton, hybrid automaton, differential equation, differential inclusion and others. The system property ϕ is typically described using a logical formalism, some sort of temporal or modal logic, such as LTL [13], CTL [4], CTL^* [9], propositional modal/temporal μ-calculus [10]. In general, modal and temporal logics are second-order logics for reasoning about and operating on sets of states. This is distinct from the first-order logics in which one reasons about elements in the interpretation domain, functions, and predicates of elements. There are two main methods for the verification of modal or temporal logic properties, namely: model-checking algorithms, and deductive proof systems. The latter ones are Hilbert-style (axiomatic), tableaux, or Gentzen-style proof systems, whereas the former ones are algorithms for exhaustive state-space exploration. The focus of this paper will be on model checking.

As assumed by its name, except for the system property, model-checking required a formal model of the system. In case of dynamic systems, the model includes an evolution function, that defines how the system progresses over time, the domain on which the system operates (the state-space), and boundary conditions that provide the system's initial states. Typical system properties, that need to be verified, are *reachability* – "Does the system reach the certain set of goal states?" – and *repeated reachability* – "Does the system return to the set of goal states infinitely often?". The model-checking algorithms, for verifying such properties, require computation of system's reachable states (images or pre-images of sets of states under the system's evolution function), as well as computation of union, intersection of sets, and testing them for inclusions.

Unfortunately, the state-spaces of dynamic systems are typically not only infinite, but also continuous (e. g. $\mathbb{R}^n$) or hybrid (e. g. a product of $\mathbb{R}^n$ and $\mathbb{Z}$). At the same time, for model-checking algorithms to be sound, they have to be in some sense computable, i. e. effectively implementable on digital computers. These requirements, respectively, rise two problems. The first is that the ordinary computability and complexity theory is not powerful enough to express the computability of real-valued functions and therefore sets of any continuous or hybrid domain. This is because a computable partial word function f operates on a countable domain, i. e. $f : \Sigma^* \to \Sigma^*$ with a finite alphabet Σ. Clearly, Σ^* is not sufficient for encoding continuous domains. The second is that digital computers do not allow for exact computations on continuous domains (e. g. a subset of $\mathbb{R}$). This is why there is a need for approximate algorithms that can guarantee arbitrarily high levels of accuracy. A natural solutions to both of these problems are offered by Type-2 Theory of Effectivity (TTE) [15], which defines computability based on Turing machines with finite and infinite input/output sequences. TTE has been already applied to analysis of computability of reachable sets of control systems in [6].

The aim of this paper is to provide a "computable" logic for model-checking of discrete-time and continuous-space dynamic systems (DTCSDSs). This should be seen as a first step towards a computable logic for general hybrid systems. Our results are based on TTE and the way we approach the problem is divided into the following steps. First, we overview several modal logics for hybrid systems and choose CTL as the logic to work with. Second, since TTE hardly relies on topologies, we analyse the original CTL semantics and alter it to become computable (topological) for the case of DTCSDS models. For simplicity,

we work with discrete time domain.

The rest of the paper is organised as follows. The preliminary material is provided in Section 2. Since TTE hardly relies on topological spaces, we begin our discussion by recalling some of the important aspects of topological theory. Then, we talk about DTCSDS models and, because they are typically expressed by multivalued-maps, we discuss continuity aspects thereof and provide some of their properties. Further, we talk about TTE and computability of various sets/functions. After that, we conclude the section by an overview of various available logics for hybrid systems. In Section 3 we construct a computable semantics of CTL on DTCSDS models. This is done by considering the standard CTL semantics and then altering it in such a way that its model-checking becomes computable. The latter sketches the approximate model-checking algorithms of CTL on DTCSDSs that can be implemented with the use of digital computers. Section 4 concludes.

2 Preliminaries

2.1 Topological Spaces

Let us recall several important notions and results from the topological theory. A *topological space* is a pair $T = (X, \tau)$ where X is an arbitrary set and $\tau \subseteq 2^X$ is such that: $\emptyset, X \in \tau$, $\forall U_1, U_2 \in \tau \Rightarrow U_1 \cap U_2 \in \tau$, and $\forall \mathbb{U} \subseteq \tau \Rightarrow \bigcup_{U \in \mathbb{U}} U \in \tau$. For a topological space T, elements of τ are called *open* and their complements in X are called *closed*. Let $x \in X$ and $B \subseteq X$ then B is a *neighbourhood* of point x if there exists an open set $U \in \tau$ such that $x \in U \subseteq B$. Let $B \subseteq X$ and $\mathbb{U} \subseteq \tau$ then $\mathbb{U}$ is an *open cover* of B if $B \subseteq \bigcup_{U \in \mathbb{U}} U$. Let $S \subseteq X$, then the set $\mathrm{Int}\,(S) = \cup\{U | U \subseteq S \wedge U \in \tau\}$ is called the *interior* of S and $\mathrm{Cl}\,(S) = \cap\{A | S \subseteq A \wedge A \text{ is closed}\}$ is called the *closure* of S. A set $C \subseteq X$ is *compact* iff every open cover of C has a finite sub cover. A subset of X is *pre-compact* iff it's closure is compact. For a topological space we have $\mathcal{O}$ – a set of open, $\mathcal{A}$ – a set of closed, and $\mathcal{K}$ – a set of compact sets.

Let $T = (X, \tau)$ be a topological space. Then $\beta \subseteq \tau$ is a *base* of the topology τ if every element of τ can be represented as a union of elements from β. A topological space is called *second countable* if its topology has a countable base. A *Hausdorff space (T_2 space)* is a topological space such that $\forall x, y \in X$ where $x \neq y$ there exist $U_x, U_y \in \tau$ such that $x \in U_x$, $y \in U_y$ and $U_x \cap U_y = \emptyset$. A Hausdorff space is called *locally compact* if every $x \in X$ has a compact neighbourhood.

2.2 Dynamic Systems

Roughly speaking, for a given state space, a dynamic system describes state evolution over time. When such a system has inputs and outputs it is called a control system. The inputs then may represent either uncontrollable disturbances or control signals, and the outputs may represent quantities that either need to be measured or controlled.

In order to understand with what type of dynamic systems we will work, let us briefly discuss their classifications. Based on the state space type, dynamic systems are divided into: *Discrete* – The state values are from a countable of

finite domain; *Continuous* – The state takes values in Euclidean space $\mathbb{R}^n$, with $n \geq 1$; *Hybrid* – A part of the state takes values in $\mathbb{R}^n$ and another part takes values in a finite set.

Considering the the set of times over which the state evolves, dynamic systems can be also divided into: *Discrete time* – The time domain is a subset of integers ($\mathbb{Z}$). Here, the system-state evolution is described by a difference equation; *Continuous time* – The time domain is a sub interval of $\mathbb{R}$. In this case, the evolution of system state is typically described by an ordinary differential equation; *Hybrid time* – The system evolves over a continuous time domain, but the system dynamic changes at specific discrete time points.

In our work we consider discrete-time continuous-space dynamic systems, which means that the state space of the system is continuous and the time domain is discrete (the system state changes at discrete time points). In system theory, dynamic systems are given by functions $f : X \times U \to X$, where X is the state space, and U can either represent control or system noise. These functions are typically converted into multivalued maps $F : X \rightrightarrows X$ such that $F(x) = f(x,\ U)$.

2.3 DTCSDSs and Multivalued Maps

A *multivalued map* $F : X \rightrightarrows Y$, also known as multivalued function or multifunction, is a total relation on $X \times Y$. If we define $F(S) = \{F(x) \,|x \in S\}$ for $S \subseteq X$ then F can be seen as a function $F : X \to 2^Y$. This last definition is more convenient when we want to talk about function composition. For example, for two multivalued maps $F : X \rightrightarrows Y$, $G : Y \rightrightarrows Z$ and their composition $G \circ F$ we have $G \circ F : X \rightrightarrows Z$ and thus for any $x \in X$ we can simply write $G \circ F(x) = G(F(x))$. A *weak preimage* of F on $B \subseteq Y$ is $F^{-1}(B) = \{x \in X \ :\ F(x) \cap B \neq \emptyset\}$ and a *strong preimage* is $F^{\Leftarrow}(B) = \{x \in X \ :\ F(x) \subseteq B\}$. The notion of *continuity* for multivalued maps is an extension of continuity for the regular functions. Let us only note that, for a continuous multivalued map F and an open set $U \subseteq Y$ the preimages $F^{-1}(U)$ and $F^{\Leftarrow}(U)$ are open sets. The space of continuous multivalued maps $F : X \to 2^Y$ will be denoted as $\mathrm{C}(X, Y)$.

2.4 Type-2 Theory of Effectivity (TTE)

TTE [15], as well are regular computability theory, is based on Turing machines. The difference is that TTE (type-2) machines allow for infinite computations. In particular they can accept infinite inputs and produce infinite outputs. The computability is first defined on type-2 machines and then is extended to arbitrary functions, sets and their elements by means of notations and representations.

Let M be a type-2 machine with a fixed finite alphabet Σ, $k \geq 0$ input tapes, one output tape and $Y_i \in \{\Sigma^*, \Sigma^\omega\}$ where $i \in 0, \ldots, k$. Then, a (partial) string function $f_M : Y_1 \times \ldots \times Y_k \to Y_0$ is *computable* iff it is realised by a type-2 machine M. The latter means that for $y_i \in Y_i$ we have $f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^*$ iff M halts on input $(y_1, \ldots, y_k)$ with y_0 on the output tape and $f_M(y_1, \ldots, y_k) = y_0 \in \Sigma^\omega$ iff M computes forever on input $(y_1, \ldots, y_k)$ and writes y_0 to the output.

The important property of a type-2 machine is that it can not re-read from input tapes and re-write onto output tape. This implies that if f_M is a computable string function, then M must always eventually write something on the

output tape. This means that every finite part of the output must be defined by some finite part of the input. Since the output tape can not be re-written this can be interpreted, in case of $y_0 \in \Sigma^\omega$, as follows: a function is computable if increasing the accuracy of its arguments (read more symbols from the input tapes), we always eventually increase the accuracy of the function's result (write more symbols on the output tape).

By definition, every elements of Σ^* is computable, and $u \in \Sigma^\omega$ is computable iff the constant function $g : \emptyset \to \Sigma^\omega$, such that $g() = u$, is computable. The computability on Σ^* and Σ^ω is generalised by means of notations and representations. A *notation* of set X is a partial surjective function $\nu : \Sigma^* \to X$ and a *representation* is a partial surjective function $\delta : \Sigma^\omega \to X$. These functions encode elements of the domain X into strings and sequences. For example, for a T_0 space $T = (X, \tau)$ with a countable base and a condition that allows to compute intersection of the base elements, any closed subset $A \subseteq X$ can be identified (encoded) by the list of names of all $U \in \tau$ such that $U \cap A \neq \emptyset$. Further, we provide other important facts from TTE that we use in the Section 3.1.

A computable Hausdorff space is a tuple (X, τ, β, ν) such that (X, τ) is a second-countable locally-compact Hausdorff (T_2) space; β is a countable base of τ consisting of pre-compact open sets; ν is a notation of β; we take effectivity properties in [3] (Lemma 2.3) as axioms; and assume that $\mathrm{Cl} : \beta \to \mathcal{K}$ is computable. Let us have a computable Hausdorff space and the Sierpinski space $\mathcal{S}$. Then, the elements of $\mathcal{O}$, $\mathcal{A}$, and $\mathcal{K}$ have so called *canonical* and equivalent to them (*admissible*) representations that are needed for the computability results below. Also, let us assume that $F^{-1}(U)$ and $F^{\Leftarrow}(U)$ are computable for $U \in \tau$ and $F \in \mathrm{C}(X, X)$. If all of the above conditions are met then the following operations are *computable* (continuous): countable union as $\mathcal{O} \times \mathcal{O} \to \mathcal{O}$, complement as $\mathcal{O} \to \mathcal{A}$, subset operation as $\mathcal{K} \times \mathcal{O} \to \mathcal{S}$, the $F^{-1}(.)$ and $F^{\Leftarrow}(.)$ as $\mathcal{O} \to \mathcal{O}$. The following operations are known to be *uncomputable*: closure as $\mathcal{O} \to \mathcal{A}$, interior as $\mathcal{A} \to \mathcal{O}$.

2.5 Modal Logics

In this section, we are going to provide an overview of a selected set of logics for hybrid systems. We consider hybrid systems, since they are a more general class of models than DTCSDSs and our ultimate, but far distant goal, is to provide a computable logic for hybrid systems. This overview is in no way complete, and a better one can be found, e. g., in [8]. Our purpose here is to discuss the most used and convenient logical formalisms. Also, we want to take into account topological aspects of hybrid systems. Remember that, the latter is important for the computability analysis because it strongly depends on the topology of the underlying state space. E. g., sets computable in one topology can be uncomputable in another.

A hybrid system is a heterogeneous dynamical system that is characterised by interacting continuous and discrete dynamics. In a simple hybrid system the state can evolve continuously, for some period of time, according to one set of differential equations, then can be abruptly reset by some discrete event to a new value and continue evolution according to another set of differential equations. Typically, the continuous dynamic of the system corresponds to a network of plants and the discrete events, guiding the plant's behaviour, are generated by a discrete automaton, also known as a control automaton. These

automata can trigger only finitely many different control events (signals).

In [8], one can find an exhaustive discussion on various modal and temporal logics. In addition, this paper talks about temporal logics for hybrid systems, see Section IV.G. There, the authors focus on extensions of temporal logics that address issues arising when working with real time. The discussion is split into three parts, based on various time semantics: *(i)* branching , *(ii)* linear , and *(iii)* interval temporal logics.

With branching-time logics one can reason about every possible future behaviour of the system, starting from the given state. In other words, branching time refers to the fact that at each moment of time there may be several different possible futures. This way, logical formulae are evaluated on a tree of states, where each path represents one possible future sequence of computations. The semantics for linear temporal logics is different. There, one has the power to express properties of possible computation sequences. I. e., each formula is evaluated on every possible computation sequence, rather then on the tree of all possible futures. The idea behind interval temporal logics is somewhat different and for the sake of brevity we leave them out of scope.

The most commonly used branching and linear temporal logics are CTL [4] and LTL [13]. These logics are incomparable, see [5], and can be united to form a more powerful logic known as CTL^*. All of these logics are typically interpreted in terms of Kripke structures. According to [8], temporal extensions of these logics are often used to represent properties of hybrid systems. For CTL, this includes Timed CTL ($TCTL$), Integrator CTL ($ICTL$) and Timed μ-calculus ($TC\mu$). An extension of LTL is, e. g., Metric Temporal Logic (MTL).

Topological aspects play an essential role in model checking of hybrid systems. One of the important motivations for this is that the control automaton has to interpret the plant sensor readings and to transform them into the control events. Since the state space of a plant is continuous and the control automaton has only finitely many control events, it is natural to expect that close values, obtained from the sensor, trigger the same control signal. This implies that the preimage of every control symbol is an open set in the topology of the continuous state space of the plant. Moreover, only the finite sub topology generated by this finite set of open sets is used by the control automaton for making decisions. Therefore, having a topological interpretation of logical formulae is very important. Logics that provide topological semantics are called *topological logics* and deal with topological spaces rather than with Kripke structures.

One of the topological logics is a branching-time logic called Dynamical Topological Logic (DTL) [11]. A distinguishing feature of this logic is that it contains one topological modality, namely the interior operator. Another topological logic is discussed in [1]. Since the authors claim that modal logics are unable to deal with continuous or mixed (hybrid) dynamical systems, they provide a topological interpretation of a propositional logic with universal and existential quantifiers. Here, atomic propositions are open sets of states and each formulae must result in an open set. This, unlike in [11], requires an altered interpretation of negations, i. e. they results in the interior for the set complement.

Since we work with discrete time and we are interested in topological aspects, as well as usability of temporal logics, further we provide details on: DTL, LTL, CTL, and CTL^*. Then, we choose one of them to work with.

Dynamic Topological Logic (DTL) [11] is a logic for Dynamic Topological

Models (DTM) that are defined as tuples $\langle X, , \rangle$. Here, X is a topological space, V is a labelling function that assigns labels to some subsets of X, and $f : X \to X$ is a continuous function which, thought of in temporal terms, moves points of X from one moment of time to another. DTL is a three modal logic with one topological modality: $\circ$ – the set *interior* operator, and two temporal modalities: $\mathcal{X}$ – the *next* operator, and G – the *henceforth* operator. Here, for a set $A \subseteq X$ the interpretation of these operators is as follows: $A^\circ = \mathrm{Int}(A)$ – the interior of the set A; $\mathcal{X}\ A = f^{-1}(A)$ – the preimage of A; and $\mathrm{G}A = \bigcap_{i \in \mathbb{N}^+} \mathcal{X}^i\ A$. DTL also supports propositional operators: conjunction, disjunction, negation, and implication. These are interpreted in the common sense of the set theory.

CTL, LTL, and CTL^* are typically interpreted over Kripke structures. A Kripke structure M is a tuple $(S,\ I,\ R,\ L)$ where M is a countable set of states; $I \subseteq S$ is a set of initial states; $R \subseteq S \times S$ is a transition relation such that $\forall s \in S \exists s' \in S : (s, s') \in R$; AP is a finite set of atomic propositions; and $L : S \to 2^{AP}$ is an interpretation (labelling) function on S. A path in a Kripke structure M is an infinite sequence of states $s_0 s_1 s_2 \ldots$ such that $\forall i \geq 0 : (s_i, s_{i+1}) \in R$. A set of paths starting in state s is denotes as $Paths(s)$. For the same state s there is an infinite computation tree, obtained by unfolding the Kripke structure, with the root s, such that (s', s'') is an arc in the tree if $(s', s'') \in R$.

Computational Tree Logic (CTL) [4] has a syntax that is divided in to *state formulae*: $\Phi ::= p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \forall\phi \mid \exists\phi$ and *path formulae*: $\phi ::= \mathcal{X}\ \Phi \mid \Phi\ \mathcal{U}\ \Phi \mid \Phi\ \mathcal{R}\ \Phi$ The state formulae have the following semantics: $s \models p$ iff $p \in L(s)$; $s \models \neg\Phi$ iff $\neg(s \models \Phi)$; $s \models \Phi \wedge \Psi$ iff $(s \models \Phi) \wedge (s \models \Psi)$; $s \models \exists\phi$ iff $\exists\sigma \in Paths(s) : \sigma \models \phi$; $s \models \forall\phi$ iff $\forall\sigma \in Paths(s) : \sigma \models \phi$. The semantics of path formulae is as follows: $\sigma \models \mathcal{X}\ \Phi$ iff $\sigma[1] \models \Phi$; $\sigma \models \Phi\ \mathcal{U}\ \Psi$ iff $\exists j \geq 0 : (\sigma[j] \models \Psi \wedge \forall 0 \leq i < j : \sigma[i] \models \Phi)$; $\sigma \models \Psi\ \mathcal{R}\ \Phi$ iff $(\forall i \geq 0 : \sigma[i] \models \Phi) \vee (\exists j \geq 0 : (\sigma[j] \models \Psi \wedge \forall 0 \leq i \leq j : \sigma[i] \models \Phi))$. It is agreed that path formulae can be only used as sub formulae of state formulae.

Linear Temporal Logic (LTL) [13] consists of *state formulae*: $\Phi ::= \forall\phi$ and *path formulae*: $\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{X}\ \phi \mid \phi\ \mathcal{U}\ \phi \mid \psi\ \mathcal{R}\ \phi$ LTL reasons about computation sequences and therefore allows for a recursive use of path formulae. For example the semantics of until operator is: $\sigma \models \phi\ \mathcal{U}\ \psi$ iff $\exists j \geq 0 : (\sigma_j \models \psi \wedge \forall 0 \leq i < j : \sigma_i \models \phi)$; The state formulae have the following semantics: $s \models \forall\phi$ iff $\forall\sigma \in Paths(s) : \sigma \models \phi$. For the path $\sigma \in Paths(s)$, where $\sigma = s_0 s_1 s_2 \ldots$, for any $j \geq 0$ we have $\sigma_j = s_j s_{j+1} s_{j+2} \ldots$, and $\sigma[j] = s_j$, the semantics of path formulae is as follows: $\sigma \models p$ iff $p \in L(\sigma[0])$; $\sigma \models \neg\phi$ iff $\neg(\sigma \models \phi)$; $\sigma \models \phi \wedge \psi$ iff $(\sigma \models \phi) \wedge (\sigma \models \psi)$; $\sigma \models \mathcal{X}\ \phi$ iff $\sigma_1 \models \phi$; $\sigma \models \phi\ \mathcal{U}\ \psi$ iff $\exists j \geq 0 : (\sigma_j \models \psi \wedge \forall 0 \leq i < j : \sigma_i \models \phi)$; $\sigma \models \psi\ \mathcal{R}\ \phi$ iff $(\forall i \geq 0 : \sigma_i \models \phi) \vee (\exists j \geq 0 : (\sigma_j \models \psi \wedge \forall 0 \leq i \leq j : \sigma_i \models \phi))$.

Branching Temporal Logic (CTL^*) [9] is a combination of LTL and CTL, it's syntax is defined by *state formulae*: $\Phi ::= p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \forall\phi \mid \exists\phi$ and *path formulae*: $\phi ::= \Phi \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{X}\ \phi \mid \phi\ \mathcal{U}\ \phi \mid \psi\ \mathcal{R}\ \phi$ The corresponding semantics is naturally induced by CTL and LTL which allows for expressing more general properties.

Similar to [1], we will require formulae to result in open sets on states. Thus, the DTL semantics is not particularly suitable for us. There are two reasons for that (we assume f-continuous and A-open). First, the henceforth operator does not result in an open set, because the countable intersection of open sets is not necessarily open. The authors remedy this situation by using Alexandrov spaces, but this limitation rules out all cases of particular interest.

Second, the negation operator does not necessarily result in an open set, since a complement of an open set is closed (clopen sets are exceptions). Also the interior operator is not of a major concern for us, since we restrict our interest to atomic propositions given by open sets. For the rest, DTL is a subset of CTL. LTL and CTL^* generally require evaluation of a formula on each system path and thus assume reasoning about sets of paths. Since paths are infinite, this requires us to use product topology and to reason about open sets of paths, rather than states. This complicates matters a little, and therefore for this paper we decide to choose CTL that is easier to provide computable semantics for (avoiding reasoning about paths).

3 Computable CTL model checking

As it was mentioned in Section 2.5, CTL is the logic we choose to provide a computable semantics for. Our choice is motivated by a relative simplicity of it's original semantics (there are no nested path formulae), and a reasonably reach variety of operators (CTL and LTL have a non-trivial common fragment [12]). Below, we consider CTL formulae and argue about which of them are computable and which are not. We also state some desired properties of the set of initial states. Along the way, we construct the computable semantics of CTL that implies corresponding model-checking algorithms. In the following we strongly rely on results provided in Section 2.4.

Let us extend the definition of DTCSDS (given in Section 2.3) to make it suitable for a computable CTL model checking.

Definition 1 *A discrete-time continuous-space control systems (DTCSDSs) is a tuple* $M = (T,\ F,\ L)$ *where:* $T = (X,\ \tau,\ \beta,\ \nu)$ *is a computable Hausdorff space;* $F \in \mathrm{C}(X, X)$ *is a multivalued map which defines the system's evolution; and* $L : X \to 2^{\mathrm{AP}}$ *is a labelling function where* AP *is a finite subset of* τ*. For any* $x \in X$ *we have* $L(x) = \{U \in \mathrm{AP} | x \in U\}$.

The labelling function $L(.)$ is an equivalent of the interpretation function for Kripke structures. Each of the open sets in $L(x)$ can be seen as an *atomic proposition* that labels x and corresponds to some system property. The motivation behind choosing atomic propositions to be open sets is the same as in [1].

Let us consider a DTCSDS model $M = (T,\ F,\ L)$ with a set of initial states $I \subseteq X$. To derive a computable semantics for CTL we should define a computable meaning for $M, I \models \Phi$ (when obvious, we will omit M), i. e. the fact that the model M satisfies $\Phi \in CTL$ for all initial states I. From now on we assume that the system's evolution F is such that taking $F^{-1}(U)$ and $F^{\Leftarrow}(U)$ for $U \in \tau$ are computable operations.

Below, in Section 3.1, we first discuss computability of CTL formulae in the original semantics and devise a computable one. Further, Section 3.2 discusses how the modified semantics influences CTL model checking.

3.1 Computable semantics for CTL

For a CTL formula Φ, let $Sat(\Phi)$ be a set of states satisfying Φ on some model M. If we can compute $Sat(\Phi)$, then for a given set of initial states I, verification of $M, I \models \Phi$ requires verifying $I \subseteq Sat(\Phi)$. Due to this, we will often identify a

CTL formula with a set of states satisfying it. Thus, we can write things like: $M, I \models U$ for $U \in \tau$ or $\Phi \in \tau$.

Now, let us consider verifying validity of an atomic proposition $p \in AP$. By definition $I \models p \Leftrightarrow I \subseteq Sat(p)$. Since $Sat(p)$ is open, from Section 2.4 we know that, we can only verify $I \subseteq Sat(p)$ in a computable way, if I is a compact. Thus, to make things uniform, we must assume that in model-checking we can only consider sets I that are compact. Then, we also need to require that any Φ results in an open set of states, since otherwise $I \models \Phi$ is not computable.

In the following we will see that then we can alter CTL semantics in such a way that any state formula results in an open set of states. This will be done implicitly by induction on the length of the formula. I. e. for any CTL formula we will assume that its sub-formulae result in open sets of states.

Semantics: $I \models \neg\Phi$. By definition $Sat(\neg\Phi) = X \setminus Sat(\Phi)$ which is a closed set and thus $I \subseteq Sat(\neg\Phi)$ is uncomputable. Therefore, we follow the topological semantics given in [1] and define $Sat(\neg\Phi) = \mathrm{Int}(X \setminus Sat(\Phi))$. The latter provides as the following computable semantics:

$$I \subseteq \mathrm{Int}(X \setminus \Phi) \Rightarrow I \models \neg\Phi.$$

Here $X \setminus Sat(\Phi)$ is closed and computable but $\mathrm{Int}(X \setminus Sat(\Phi))$ is uncomputable due to the interior operator. Of course, we could assume that the representation of $Sat(\Phi)$ is such that we can compute $\mathrm{Int}(X \setminus Sat(\Phi))$, but for an arbitrary Φ this assumption is not very realistic.

It is known that (see e. g. [14]), any CTL formula can be transformed into an equivalent CTL formula in the negation normal form (NNF). In NNF, negation only prefix atomic propositions and thus one can make sure that the representations of those are good enough to make their negations computable. The transformation into NNF can be done using the following equivalences:

$$\neg\neg\Phi = \Phi, \qquad \neg(\Phi \wedge \Phi) = \neg\Phi \vee \neg\Phi, \qquad \neg\mathcal{X}\ \Phi = \mathcal{X}\ \neg\Phi,$$

$$\neg\neg\phi = \phi, \qquad \neg(\forall\phi) = \exists\neg\phi, \qquad \neg(\Phi\ \mathcal{U}\ \Psi) = \neg\Psi\ \mathcal{R}\ \neg\Phi$$

Let us consider the remaining CTL operators. We will omit the case of $\Phi \wedge \Psi$, since it is trivially computable, assuming that $Sat(\Phi)$ and $Sat(\Psi)$ are open sets.

Semantics: $I \models \forall(\mathcal{X}\ \Phi)$. For an open set $U_\Phi = Sat(\Phi)$, $F^{\Leftarrow}(U_\Phi)$ is the set of states from which we always go into U_Φ states in one step. Clearly, $F^{\Leftarrow}(U_\Phi)$ is open and computable and thus $I \subseteq F^{\Leftarrow}(U)$ is also computable. Therefore, we define the computable semantics as:

$$I \models \forall(\mathcal{X}\ U) \Leftrightarrow I \subseteq F^{\Leftarrow}(U).$$

Semantics: $I \models \exists(\mathcal{X}\ \Phi)$. Similar to the previous case, since $F^{-1}(U_\Phi)$ is open and computable, we define the computable semantics as:

$$I \models \exists(\mathcal{X}\ U_\Phi) \Leftrightarrow I \subseteq F^{-1}(U_\Phi).$$

We use weak preimage to account for the existential quantifier.

Semantics: $I \models \forall(\Phi\ \mathcal{U}\ \Psi)$. As before, we assume that Φ and Ψ corresponds to open sets U_Φ and U_Ψ. According to the original semantics of CTL:

$$\begin{aligned} I \models \forall(U_\Phi\ \mathcal{U}\ U_\Psi) \Leftrightarrow \forall x \in I : \forall\sigma \in Paths(x) : \exists j \geq 0 : \\ (\sigma[j] \in U_\Psi \wedge \forall 0 \leq i < j : \sigma[i] \in U_\Phi) \end{aligned}$$

In other words, initial set of states I satisfies the formula if, for every state $x \in I$, every path starting in x consists of a finite prefix of states from U_Φ that is then followed by at least one state from U_Ψ. Therefore, the set $S_0 = U_\Psi$ gives states in which the formula $\forall \Phi \, \mathcal{U} \, \Psi$ is satisfied right away. States $S_1 = F^{\Leftarrow}(U_\Psi) \cap U_\Phi$ satisfy the formula, since any path $\sigma \in Paths(S_1)$ is such that $\sigma[0] \in U_\Phi$ and $\sigma[0] \in U_\Psi$ (because $\sigma[1] \in F(\sigma[0]) \subseteq U_\Psi$). By induction, for any $n \geq 1$ we have that $S_n = F^{\Leftarrow}\left(\bigcup_{i=0}^{n-1} S_i\right) \cap U_\Phi$ is an open set that consists of states satisfying the formula. Clearly, the union $\bigcup_{n=0}^{\infty} S_n$ gives us all system states where the formula is satisfied and thus we can define: $I \models \forall (U_\Phi \, \mathcal{U} \, U_\Psi) \Leftrightarrow I \subseteq \bigcup_{n=0}^{\infty} S_n$. Notice that, this semantics is computable since every S_i is open and computable and a countable union of S_i is also open and computable.

Semantics: $I \models \exists (\Phi \, \mathcal{U} \, \Psi)$. Since $S'_0 = U_\Psi$ and $S'_n = F^{-1}\left(\bigcup_{i=0}^{n-1} S_i\right) \cap U_\Phi$ are open and computable, we define the computable semantics as:

$$I \models \exists (U_\Phi \, \mathcal{U} \, U_\Psi) \Leftrightarrow I \subseteq \bigcup_{n=0}^{\infty} S'_n.$$

We use weak preimage to account for the existential quantifier.

Semantics: $I \models \forall (\mathrm{G}\Phi)$. Let us consider $\mathrm{G}\Phi$ (*henceforth operator*) – a common abbreviation of (*false* $\mathcal{R}$ Φ). First, we provide a computable semantics for this formula and then discuss a more complex case of the release operator. Intuitively, the CTL semantics for this formula can be expressed as: $I \models \forall (\mathrm{G}U) \Leftrightarrow I \subseteq \bigcap_{n=0}^{\infty} S_n$ with $S_0 = U$ and for all $n \geq 1 : S_n = F^{\Leftarrow}(S_{n-1})$. It is easy to see that $\bigcap_{n=0}^{\infty} S_n$ is a set of states from which every path goes only through states in U. Unfortunately, even though every S_n is open, countable intersection of open sets is not necessarily open (see Section 2.1) and thus the semantics above is not computable. Notice that, a set equivalent to $\bigcap_{n=0}^{\infty} S_n$ is given by: $P ::= \bigcup \{S \subseteq X | S \subseteq U \wedge S \subseteq F^{\Leftarrow}(S)\}$. This set is neither open nor computable. Let us define $P' ::= \bigcup \{O \in \tau | O \subseteq U \wedge O \subseteq F^{\Leftarrow}(O)\}$. Clearly, $P' \subseteq P$ and the set P' is "almost computable". The latter is because P' is a union of open sets but since U and $F^{\Leftarrow}(O)$ are open the checks $O \subseteq U$ and $O \subseteq F^{\Leftarrow}(O)$ are not computable. Also, getting all open subsets from τ is not a computable operation. To remedy these problems we define the computable semantics as:

$$I \subseteq \bigcup \{B_r \in \tau | \mathrm{Cl}(B_r) \subseteq U \wedge \mathrm{Cl}(B_r) \subseteq F^{\Leftarrow}(B_r)\} \Rightarrow I \models \forall (\mathrm{G}U)$$

Here, each $B_r \in \tau$ is a finite union of base elements (e. g. open rational boxes in X) and from the definition of the computable Hausdorff space $\mathrm{Cl}(B_r)$ is computable and compact. Since the union of sets on the left-hand side of the implication is a subset of $P' \subseteq P = \bigcap_{n=0}^{\infty} S_n$, we have that if the formula satisfied in the computable semantics, then it is satisfied in the original semantics.

Semantics: $I \models \exists (\mathrm{G}\Phi)$. Similar to the previous case, we define

$$I \subseteq \bigcup \left\{B_r \in \tau | \mathrm{Cl}(B_r) \subseteq U \wedge \mathrm{Cl}(B_r) \subseteq F^{-1}(B_r)\right\} \Rightarrow I \models \exists (\mathrm{G}U)$$

We use weak preimage $F^{-1}(B_r)$ to account for the existential quantifier.

Semantics: $I \models \forall (\Psi \, \mathcal{R} \, \Phi)$. According to the original semantics of CTL:

$$\begin{aligned} I \models \forall (U_\Psi \, \mathcal{R} \, U_\Phi) \Leftrightarrow \forall x \in I : \forall \sigma \in Paths(x) : (\forall i \geq 0 : \sigma[i] \in U_\Phi) \vee \\ (\exists j \geq 0 : (\sigma[j] \in U_\Psi \wedge \forall 0 \leq i \leq j : \sigma[i] \in U_\Phi)) \end{aligned}$$

Assuming an abbreviation for $\mathrm{G}U_\Phi$, we get:

$$I \models \forall\,(U_\Psi\ \mathcal{R}\ U_\Phi) \Leftrightarrow \forall x \in I : \forall \sigma \in \mathit{Paths}\,(x) : (\sigma \models \mathrm{G}U_\Phi) \vee (\sigma \models U_\Phi\ \mathcal{U}\ (U_\Phi \cap U_\Psi))\,.$$

Clearly, a path σ satisfies $(U_\Psi\ \mathcal{R}\ U_\Phi)$ iff either every state of the path belongs to U_Φ or there is a finite prefix of states from U_Φ where the last state of the prefix is also in $U_\Phi \cap U_\Psi$. We already know how to treat universal until and henceforth operators, here we should simply combine the approaches:

$$I \subseteq \bigcup \{B_r \in \tau | \mathrm{Cl}\,(B_r) \subseteq U_\Phi \wedge (\mathrm{Cl}\,(B_r) \subseteq U_\Psi \vee \mathrm{Cl}\,(B_r) \subseteq F^{\Leftarrow}\,(B_r \cup (U_\Psi \cap U_\Phi)))\} \\ \Rightarrow \qquad I \models \forall\,(U_\Psi\ \mathcal{R}\ U_\Phi)$$

Note that, since we require $\mathrm{Cl}\,(B_r) \subseteq U_\Phi$, the second condition on the set B_r serves two purposes: *(i)* to provide fixed point characterisations for the paths satisfying $\mathrm{G}U_\Phi$; *(ii)* to allow paths with prefixes in U_Φ and the last states of these prefixes being in $U_\Psi \cap U_\Phi$. The latter results in paths satisfying $(U_\Phi\,\mathcal{U}\ (U_\Phi \wedge U_\Psi))$. Since $B_r \cup (U_\Psi \cap U_\Phi)$ is an open set, the resulting semantics is computable.

Semantics: $I \models \exists\,(\Psi\ \mathcal{R}\ \Phi)$. Similar to the previous case, we define:

$$I \subseteq \bigcup \left\{B_r \in \tau | \mathrm{Cl}\,(B_r) \subseteq U_\Phi \wedge \left(\mathrm{Cl}\,(B_r) \subseteq U_\Psi \vee \mathrm{Cl}\,(B_r) \subseteq F^{-1}\,(B_r \cup (U_\Psi \cap U_\Phi))\right)\right\} \\ \Rightarrow \qquad I \models \exists\,(U_\Psi\ \mathcal{R}\ U_\Phi)$$

Once again, we use weak preimage to account for existential quantifier.

3.2 Changes in the semantics

A consequence of the (necessary) choice of semantics of the negation and release (henceforth) operators is that there can not be a proof by contradiction (the Law of Excluded Middle does not hold). In other words, if Φ contains at least one of the above mentioned operator, the fact that $M, I \models \Phi$ does not hold does not imply that $M, I \models \neg\Phi$ holds. Moreover, such a Φ can be true (on M, I) but not computably verifiable.

There is nothing we can do about the Law of Excluded Middle for the release (henceforth) operator. Moreover, in [7] it was shown that our semantics for the henceforth operator is optimal. For the negation operator, let us notice that in the standard semantics $\neg\neg\Phi$ and Φ are equivalent whereas in the given topological semantics we generally have $I \models \neg\neg\Phi \not\Leftrightarrow I \models \Phi$. Thus, converting Φ into NNF, prior to model checking, does not only allow to make negation computable, but also reduces the effects of the topological semantics.

4 Concluding remarks

In this work we focused on model checking of discrete-time continuous-space control systems and in particular on computability aspects thereof. Due to continuity of the state space, regular computability and complexity theory of Turing is not applicable and therefore a more powerful approach is required. Our choice is the Type Two Effectivity theory (TTE). One of our main goals was to provide a logic that would allow to express system properties that are effectively verifiable (computable) in the sense of TTE. After considering several popular logics for hybrid systems, including topological logics, we decided to

provide a computable semantics for the branching time logic called CTL. We have analysed this logic with respect to its computability aspects. It turned out that logical operators such as *negation* and *release until* require a significant change in their interpretation, to make sets of states satisfying formulae using them computable.

As it was discussed in Section 3.1 the computable model checking of CTL on DTCSDSs assumes a compact set of initial states I, a CTL state formula Φ, and a system model $M = (T,\ F,\ L)$, where $T = (X, \tau, \beta, \nu)$ is a computable Hausdorff space and a map $F \in \mathrm{C}(X, X)$ for which $F^{-1}(U)$ and $F^{\Leftarrow}(U)$ are computable for $U \in \tau$. If Φ contains negations, then it must be converted into NNF and the open sets, corresponding to atomic propositions that are then prefixed with negations, must have representations that allow for computing interiors of their complements. In computable semantics, if Φ contains negation or release (henceforth) operators and $I, M \models \Phi$ does not hold then it does not imply that $I, M \models \neg\Phi$ holds. If the formula holds in the computable semantics, then it also holds in the original one.

Currently, we work on extending the approach presented in this paper towards computable model checking of LTL and CTL^*. Our long-standing goal is to provide a computable logic for the most general class of hybrid systems. Another plan is to implement the computable model checking algorithms in a framework for reachability analysis of hybrid systems called Ariadne [2].

References

[1] S. N. Artemov, J.M. Davoren, and A. Nerode. Logic, Topological Semantics and Hybrid Systems. In *International Conference on Decision and Control, CDC'97*, volume 1, pages 698–701, CA, USA, 1997. IEEE Press.

[2] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Symposium on Mathematical Theory of Networks and Systems (MTNS 2006)*, Kyoto, Japan, July 2006. To appear.

[3] Vasco Brattka and Gero Presser. Computability on subsets of metric spaces. *Theoretical Computer Science*, 305(1-3):43–76, 2003.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *AMC Transactions On Programming Languages And Systems*, 8(2):244–263, 1986.

[5] Edmund M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop*, pages 428–437, London, UK, 1989. Springer-Verlag.

[6] Pieter Collins. Continuity and computability of reachable sets. *Theoretical Computer Science*, 341(1):162–195, 2005.

[7] Pieter Collins. Optimal Semicomputable Approximations to Reachable and Invariant Sets. *Theory of Computing Systems*, 41(1):33–48, 2007.

[8] J. M. Davoren and Anil Nerode. Logics for hybrid systems. In *Proceedings of the IEEE*, volume 88, pages 985–1010. Springer-Verlag, 2000.

[9] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the Association for Computing Machinery (ACM)*, 33(1):151–178, 1986.

[10] D. Kozen. Results on the propositional μ-calculus. Research Report RC 10133 (44981), IBM Research Division, August 1983. 42 Pages.

[11] Philip Kremer and Grigori Mints. Dynamic topological logic. *Annals of Pure and Applied Logic*, 131(1–3):133–158, 2005.

[12] Monika Maidl. The common fragment of ctl and ltl. In *Annual Symposium on Foundations of Computer Science, FOCS'00*, pages 643–652. IEEE Computer Society, 2000.

[13] Amir Pnueli. The Temporal Semantics of Concurrent Programs. In *Semantics of Concurrent Computation*, pages 1–20. Springer, 1979.

[14] Klaus Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. TTCSS. Springer-Verlag, Berlin, Heidelberg, 2004.

[15] Klaus Weihrauch. *Computable Analysis: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2000.