THE CHURCH THESIS, ITS PROOF, AND THE NOTION OF STABILITY AND STABILIZATION FOR ANALOG ALGORITHMS

ZVI RETCHKIMAN KÖNIGSBERG[1] AND NACHUM DERSHOWITZ[2]
[1]INSTITUTO POLITÉCNICO NACIONAL, CIC
CIUDAD DE MEXICO, MEXICO
MZVI@CIC.IPN.MX
[2]TEL AVIV UNIVERSITY, SCHOOL OF COMPUTER SCIENCE
69978 RAMAT AVIV, ISRAEL
NACHUM@TAU.AC.IL

Abstract. This work presents Church's thesis, its proof, and the notion of stability and stabilization for analog algorithms. The Church thesis for discrete algorithms motivates us to consider the analogous thesis for the case when we are dealing with analog algorithms. Its presentation and proof follow a similar construction to the one for discrete algorithms. The notions of analog algorithm and dynamical system are postulated to be equivalent. The stability and stabilization concepts for analog algorithms are defined.
Keywords: Analog Algorithms, Dynamical Systems, Church Thesis, Stability, Stabilization, Lyapunov Functions.
Mathematics Subject Classification: 03D99, 03B05,68Q01, 93D05

1. Introduction

This work presents the Church thesis, its proof, and the notion of stability and stabilization for analog algorithms. The Church thesis for discrete algorithms motivates us to consider the Church thesis for the case when we are dealing with analog algorithms. Gurevich [1] has shown that any algorithm that satisfies three natural postulates can be step-by-step emulated by an abstract state machine (ASM). Adding two more postulates, Dershowitz and Gurevich [2] proceeded to prove that all notions of algorithms for common discrete-time models of computation in computer science are covered by this formalization. This includes Turing machines, Minsky

Date: December 3, 2018.

counter machines, Post machines, random access memory (RAM) and so on. Our presentation and proof follows a similar construction to the one given for discrete algorithms. Bournez, Dershowitz and Néron [3] have formalized a generic notion of analog algorithm; their proposed framework is an extension of [1, 2]. They provide postulates defining analog algorithms in the spirit of those given for discrete algorithms, and continue with some completeness results. The notions of analog algorithm and dynamical system are postulated to be equivalent.

In this paper, the stability and stabilization concepts for analog algorithms are defined.

This paper is organized as follows. First, in Section 2, the work of Dershowitz and Gurevich related to discrete algorithms is reviewed. In Section 3, Bournez, Dershowitz and Néron's generic notion of analog algorithm is rewritten and re-interpreted; the Church thesis is addressed and proved; stability and stabilization concepts for analog algorithms are defined. Section 4 presents an application example, and, finally, Section 5 discusses the stability concept for analog algorithms in terms of Lyapunov energy functions.

2. Discrete algorithms

The main goal of this section consists in giving a summary of the work done by Dershowitz and Gurevich [1, 2].

The basic characteristic of a computable function is that there must be a finite procedure (an "algorithm") telling how to compute the function. The following models of computation give different interpretations of what a procedure is and how it is used (informal algorithms): Turing machines, flowchart computable, λ calculus, Post machines, Herbrand-Gödel computable, etc. The fact that these models give equivalent classes of computable functions stems from the fact that each model is capable of reading and mimicking a procedure for any of the other models, much as a compiler is able to read instructions in one computer language and emit instructions in another language.

Enderton [4] gives the following characteristics of a procedure for computing a computable function; similar characterizations have been given by Turing [11], Rogers [8], and others.

1) "There must be exact instructions (i.e., a program), finite in length, for the procedure." (Thus every computable function must have a finite program that completely describes how the function

is to be computed. It is possible to compute the function by just following the instructions; no guessing or special insight is required.)
2) "If the procedure is given a k-tuple x in the domain of f, then after a finite number of discrete steps the procedure must terminate and produce $f(x)$." (Intuitively, the procedure proceeds step by step, with a specific rule to cover what to do at each step of the calculation. Only finitely many steps can be carried out before the value of the function is returned.)
3) "If the procedure is given a k-tuple x which is not in its domain of f, then the procedure might go on forever, never halting. Or it might get stuck at some point (i.e., one of its instructions cannot be executed), but it must not pretend to produce a value for f at x." (Thus if a value for $f(x)$ is ever found, it must be the correct value. It is not necessary for the computing agent to distinguish correct outcomes from incorrect ones because the procedure is always correct when it produces an outcome.)

To summarize, based on this view, a function is computable if: (a) given an input from its domain, possibly relying on unbounded storage space, it can give the corresponding output by following a procedure (program) that is formed by a finite number of exact unambiguous instructions (Postulate II and III; see below); (b) it returns such output (halts) in a finite number of steps; and (c) if given an input that is not in its domain it either never halts or it gets stuck.

Gurevich [1] has proposed a model which incorporates all these properties in the notion of a formal algorithm free of interpretations and which is outlined next.

Definition 1. A vocabulary $\mathcal{V}$ is a finite collection of fixed-arity (possibly nullary) function symbols. We assume that $\mathcal{V}$ contains the scalar (nullary) function TRUE. A first-order structure X of vocabulary $\mathcal{V}$ is a non-empty set S, the base set (domain) of X, together with interpretations of the function symbols in $\mathcal{V}$ over S, denoted $[\![f]\!]_X$. Similarly, the interpretation of a term $f(t_1, \ldots, t_n)$ in X is recursively defined by $[\![f(t_1, \ldots, t_n)]\!]_X = [\![f]\!]_X([\![t_1]\!]_X, \ldots, [\![t_n]\!]_X)$. Let X and Y be structures of the same vocabulary $\mathcal{V}$. An isomorphism from X onto Y is a one-to-one function ζ from the base set of X onto the base set of Y such that $f(\zeta t_1, \ldots, \zeta t_n) = \zeta(t_0)$ in Y whenever $f(t_1, \ldots, t_n) = t_0$ in X.

Definition 2. A *state-transition system* consists of a set of states S, a subset I of initial states, and transition functions on states that determines the next-state relation. (States with no "next" state will be *terminal* states.)

Postulate I (Sequential time)**.** An algorithm is a state-transition system. Its transitions are partial functions.

Postulate II (Abstract state)**.** States are first order structures with equality, sharing the same fixed, finite vocabulary. States and initial states are closed under isomorphism. Transitions preserve the domain, and transitions and isomorphisms commute.

Postulate III (Bounded exploration)**.** Transitions are determined by a fixed finite glossary of critical terms. That is, there exists some finite set of variable free terms over the vocabulary of the states, such that states that agree on the values of these glossary terms, also agree on all next-step state changes

An abstract state machine, or *ASM*, is a state-transition system in which algebraic states (no predicate symbols) store the values of functions of the current state. Transitions are programmed using a convenient language based on guarded commands for updating individual states. *ASMs* capture the notion that each step of an algorithm performs a bounded amount of work, whatever domain it operates over, so are central to the development.

Definition 3. An *abstract state machine (ASM)* is given by: a set (or proper class) S of algebraic states (no predicate symbols), closed under isomorphism, sharing a vocabulary $\mathcal{V}$; a set I of initial states, closed under isomorphism; and a program P, consisting of finitely many commands, each taking the form of a guarded assignment

$$\text{if} \quad q \quad \text{then} \quad t := u$$

for terms t and u over the vocabulary, and q is a conjunction of equalities and inequalities between terms, i.e., given a state α which belongs to S, program P defines and therefore computes the following subset of the set of updates $\bigtriangleup^+$:

$$\{f([\![\bar{s}]\!]_\alpha) := [\![u]\!]_\alpha : (\text{if } q \text{ then } f(\bar{s}) := u) \in P \text{ and } [\![q]\!]_\alpha = \textsc{true}\}.$$

Gurevich [1] continues by proving the following important result.

Theorem 4 (Representation theorem)**.** *For every process (algorithm) satisfying the postulates, there is an abstract state machine (ASM) in the same vocabulary (and with the same sets of states and initial states) that emulates it.*

Postulate IV (Arithmetical state)**.** Initial states are arithmetical and blank. Up to isomorphism, all initial states share the same static operations, and there is exactly one initial state for any given input values.

Now, employing this last postulate [2], arithmetical *ASM* machines are defined. With all this information, the Church thesis is proved.

Theorem 5 (Church thesis)**.** *A numeric function f is partial recursive iff f is computed by a state-transition system satisfying the four postulates.*

Remark 6. We have restricted this presentation to algorithms which work in the natural number domain. However, it is possible to extend it to other possible domains (which may include strings, lists, graphs, etc), by introducing an encoding notion and the concept of arithmetized algorithm (as is done in [2]). Finally, it is proved that no matter what other model of computation is chosen, its power of computation will not be increased more than that given by the partial recursive functions. Theorem 4 plays a fundamental role in the proof (see [2] for more details). As a corollary Turing's thesis is obtained.

We are interested in extrapolating all what was above discussed to analog algorithms following the lines given by Bournez, Dershowitz and Néron.

3. Analog Algorithms, the Church-Turing, stable algorithms and stabilization of unstable analog algorithms

In this section, the work done by Bournez, Dershowitz and Néron [3] is rewritten, re-interpreted, and extended.

Definition 7. A *dynamical system* has four components, T, X, A, ϕ_t, where T is called the *time* set, X is a state space (a metric space with metric d), A is the set of initial states $A \subseteq X$,

and $\phi_t : X \to X$ is a family of evolution operators parameterized by $t \in T$ satisfying the following properties: $\phi_0(x) = x$ for $x \in X$ and $\phi_{t+s} = \phi_t \circ \phi_s$.

Dynamical systems are classified based on the properties of T, X and ϕ_t. Is the time set T continuous or discrete? Is the state space X finite or infinite? Is it continuous or discrete? Is it finite-dimensional or infinite-dimensional? Is the map ϕ_t deterministic or stochastic? Autonomous or time-dependent? Invertible or not? Etc. Some examples of dynamical systems are: Turing machines, finite state automata, continuous systems, discrete systems, discrete event systems and hybrid systems, to mention but a few.

When $T = \mathcal{R} = (-\infty, \infty)$, we speak of a *continuous-time* dynamical system, and when $T = \mathcal{N} = \{0, 1, 2, \ldots\}$ we speak of a *discrete-time* dynamical system. When discussing hybrid dynamical, T must be generalized (for details see [10]).

A dynamical system is generally described by one or more differential or difference equations. There are other important classes of dynamical systems as those described by continuous differential equations, functional differential equations, semi-groups, to mention some.

Remark 8. When dealing with continuous dynamical systems determined by ordinary differential equations on $\mathcal{R}^n$, we define the euclidean metric d as:

$$d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}, \quad \forall x, y \in \mathcal{R}^n.$$

For discrete dynamical systems determined by difference equations, X equipped with the above euclidean metric defines a metric space.

Definition 9. A dynamical system is said to be *computable* if it is family of evolution operators (also called its trajectories) are obtained as solutions of its mathematical model.

Postulate A. An analog algorithm is a dynamical system.

Definition 10 (Abstract transition system)**.** An *abstract transition system* is a state transition system whose function symbols f are interpreted as functions, and that satisfies Postulate II, where the transition function on states is ϕ_t.

Postulate B. An analog algorithm is an abstract transition system.

Now, proceeding as in the discrete algorithm case, *ASMs* are defined adding to its program some new rules for which some prior definitions are needed.

Definition 11 (Locations)**.** If f is a j-ary function symbol in vocabulary $\mathcal{V}$ and a is a j-tuple of elements of the base set of X, then the pair (f, a) (also denoted $f(a)$) is called a *location*. We denote by $[\![f(a)]\!]_X$ its interpretation in X. If (f, a) is a location of X then $(f, a, [\![f(a)]\!]_X)$ is an update of X. When Y and X are structures over the same domain and vocabulary, $Y \setminus X$ denotes the set of updates $\bigtriangleup^+ = \{(f, a, [\![f(a)]\!]_Y) : [\![f(a)]\!]_Y \neq [\![f(a)]\!]_X\}$.

Definition 12 (Infinitesimal generators)**.** An *infinitesimal generator* is a function $\bigtriangleup$ that maps the state space X to a set $\bigtriangleup(X)$ of updates, and preserves isomorphisms, i.e., if ζ is an isomorphism of states X, Y, then for all updates $(f, a, [\![f(a)]\!]_X) \in \bigtriangleup(X)$, we have an isomorphic update $(f, \zeta a, \zeta[\![f(a)]\!]_X) \in \bigtriangleup(Y)$.

Definition 13 (Semantics)**.** A *semantics* ψ over a class C of sets S is a partial function mapping initial evolutions over some $S \in C$ to an element of S.

Definition 14. The infinitesimal generator associated with a semantics ψ maps the state space X, such that $\psi[X, f, a] = \psi([\![f(a)]\!]_{\phi_t(X)})$ is defined for all locations (f, a), to the set of updates $\bigtriangleup_\psi(X) = \{(f, a, \psi[X, f, a]) : (f, a) \in X\}$.

Remark 15. When $T = \mathcal{R}$, an example of semantics over the class of sets S containing T is the derivative ψ_{der}, when it exists. When $T = \mathcal{N}$, an example of semantics over the class of all sets would be the function $\psi_{\mathcal{N}}$ mapping f to $\psi_{\mathcal{N}}(f_n) = f_{n+1}$, $n \in \mathcal{N}$.

Remark 16. From now on, we assume that some semantics ψ is fixed to deal with different types of dynamical systems, it could be ψ_{der}, but it could also be another one. However, it is assumed that the class of dynamical systems is restricted to those that guarantee the existence of the respective semantics, and as a result its associated set of updates is well defined. Therefore, not all possible dynamical systems are allowed.

The following corresponds to the Bounded Exploration Postulate III but now for analog algorithms.

Postulate C. For any analog algorithm, there exists a finite set T of variable free terms over the vocabulary $\mathcal{V}$, such that for all states X and Y that coincide for T, $\bigtriangleup_\psi(X) = \bigtriangleup_\psi(Y)$.

Rules of the *ASM* program:

In addition to the rule of the *ASM* program (see Definition 3), we introduce the following rules:

Definition 17 (Parallel rule)**.** If $R_1, R_2, \ldots, R_k$ are rules of the *ASM*, i.e.,

$$\text{if} \quad q \quad \text{then} \quad t := u$$

then,

$$\text{par} \quad R_1, R_2, \ldots, R_k \quad \text{endpar}$$

is a rule that executes them in parallel, with $\bigtriangleup_\psi(X)$ equal to the union of the same sub-set of updates given in Definition 3 for each $R_1, R_2, \ldots, R_k$.

Definition 18. A rule of the form $\mathit{Dynamic}(f(t_1, t_2, \ldots, t_j), t_0)$ where f is a symbol of arity- j and, $t_0, t_1, t_2, \ldots, t_j$ are variable free terms, then the rule is defined by $\psi[X, f, (t_1, t_2, \ldots, t_j)] = \psi(f(t_1, t_2, \ldots, t_j)) := t_0$, where $\{\psi[X, f, (t_1, t_2, \ldots, t_j)]\}$ is an element of the set of updates $\bigtriangleup_\psi(X)$. In addition, if $R_1, R_2, \ldots, R_k$ are rules of the form *Dynamic* then,

$$\text{par} \quad R_1, R_2, \ldots, R_k \quad \text{endpar}$$

is also a rule, with $\bigtriangleup_\psi(X)$ being equal to the union of $\{(f_i, a_i, \psi[X, f_i, a_i]) : (f_i, a_i) \in X\}$ for $i = 1, \ldots, k$.

Definition 19 (Conditional rule)**.** If ϕ is a boolean term and R_1 and R_2 are rules then

$$\text{if } \phi \text{ then } R_1 \text{ else } R_2$$

is a rule.

The following result plays a fundamental role in the proof of the Church thesis for analog algorithms.

Theorem 20. *For every algorithm of vocabulary $\mathcal{V}$, there is a ψ-ASM program that, for all states, has the identical set of updates.*

Example 21. Let us consider a simple pendulum whose dynamics is described by the following second order differential equation $\theta'' + \frac{g}{l}\theta = 0$. Its evolution is described by its associated set of updates (with ψ_{der}) of the following program rule

$$\text{par} \quad Dynamic(\theta, \theta_1), \quad Dynamic(\theta_1, -\frac{g}{l}\theta) \quad \text{endpar}$$

.

Example 22 ([6])**.** A discrete event system, is a dynamical system whose state evolves in time by the occurrence of events at possibly irregular time intervals. Place-transitions Petri nets (commonly called Petri nets) are a graphical and mathematical modeling tool applicable to discrete event systems in order to represent its states evolution, whose mathematical model is given in terms of difference equations. The matrix difference equation describing the dynamical behavior of a Petri net with m places and t transitions is represented as [5]:

$$M_{n+1} = M_n + A^T u_n,\ n \in \mathcal{N}, M_n \in \mathcal{N}^m \text{and } u_n \in \{0,1\}^t \tag{1}$$

This evolution is described by its associated set of updates (with $\psi_{\mathcal{N}}$) of the following program rule
(2)

$$\text{par} \quad Dynamic(M_n(p_1), M_n(p_1) + \sum_{j=1}^{t} a_{j1}u_n(j)), \cdots, Dynamic(M_n(p_m), M_n(p_m) + \sum_{j=1}^{t} a_{jn}u_n(j)) \quad \text{end}$$

Notice that if M' can be reached from some other marking $M = M_n$ for some $n \in \mathcal{N}$ through a firing sequence $\{u_0, u_1, ..., u_{d-1}\}$ writing equation (1) for each one of the elements of the firing sequence, and summing up, we obtain that

$$M' = M + A^T u,\ u = \sum_{k=0}^{d-1} u_k. \tag{3}$$

Equation (3) would result in an *ASM* program, where the program rule (2) appears d times.

Definition 23. A ψ-*ASM* comprises the following: an *ASM* program, a set S of first-order structures with equality over some finite vocabulary $\mathcal{V}$ closed under isomorphisms with a subset S_0 of S

closed under isomorphisms, and a well defined update set of computations $\bigtriangleup_\psi$ associated with ψ.

Definition 24. An analog algorithm is a ψ-*ASM* which satisfies Postulates A, B and C.

We are assuming for that, for each dynamical system, the trajectories can be computed from the description of its dynamical system. (As for example, in the case of nonlinear differential equation, the Lipschitz conditions are satisfied, etc.) In other words, not all dynamical systems are contemplated, just those which guarantee their existence.

Bournez, Dershowitz and Néron finish their presentation giving their main result (similar to Theorem 4).

Definition 25. A semantics ψ is unambiguous if for all sets S of first-order structures over some finite vocabulary $\mathcal{V}$ closed under isomorphisms, and for all subsets $S_0 \in S$ closed under isomorphisms, whenever there exists some ϕ and a ψ-*ASM*, then ϕ is unique.

Theorem 26. *Assuming ψ is unambiguous, for every process (algorithm) satisfying Postulates A, B and C, there is an equivalent ψ-ASM.*

Theorem 27 (Church thesis for analog algorithms)**.** *The dynamical system is computable if and only if the ψ-ASM computes them.*

Proof. If the dynamical system is computable (recall Definition 9), there exists a procedure (algorithm) that computes its trajectories from its mathematical model description and, therefore, the ψ-*ASM* program will be able to emulate and compute these trajectories by a proper definition of its rules (see 20, 23). For the other direction of the implication, given a ψ-*ASM* program which first interprets the fixed dynamical system and then computes its trajectories, we define a numerical procedure which mimics it and therefore computes the dynamical system's trajectories. In fact, its trajectories define an exact mathematical model of themselves. □

3.1. **Stability and stabilization of analog algorithms.** In this section, we focus our attention on the class of continuous and discrete-time dynamical systems described by differential or difference equations, leaving other types for future work. We begin

by recalling some basic definitions in stability theory for this class [6, 11].

Definition 28 (Stability)**.**

- Let us consider a dynamical system represented by the following differential equation

(4)
$$dx/dt = f(t,x) : x(0) = x_0 \in \mathcal{R}^n, x \in \mathcal{R}^n, f : \mathcal{R}^+ \times \mathcal{R}^n \to \mathcal{R}^n \text{continuous},$$

  We say that state $x = 0$ of system (4) is stable if $\forall t_0 \in \mathcal{R}^+$ and $\forall \varepsilon > 0 \ \exists \ \delta = \delta(t_0, \varepsilon) > 0$ such that if $\|x_0\| < \delta \Rightarrow \|x(t, t_0, x_0)\| < \varepsilon \ \forall t \in (t_0, \infty)$.
- Consider a dynamical system represented by the following difference equation

(5)
$$x(n+1) = f[n, x(n)] : x(n_o) = x_0 \in \mathcal{R}^n, n \in \mathcal{N}_{n_0}, x(n) \in \mathcal{R}^n, f : N_{n_0} \times R^n \to R^n \text{continuous}$$

  We say that state $x = 0$ of system (5) is stable if and only if, $\forall n_0 \in \mathcal{N}$ and $\forall \varepsilon > 0 \ \exists \ \delta = \delta(n_0, \varepsilon) > 0$ such that if $\|x_0\| < \delta \Rightarrow \|x(n, n_0, x_0)\| < \varepsilon \ \forall n \in N_{n_0}^+$.

Now, let us divide the set of structures, i.e., the set of states of the dynamical system, in unstable and stable sets $X = \{X_{un}, X_s\}$.

Definition 29. An analog algorithm is said to be *stable* if and only if the dynamical system is stable if and only if the unstable structures are empty or they are not attained as the program of the ψ-*ASM* executes.

A clear example of an unstable analog algorithm is the one defined for chaotic dynamical systems.

Let us suppose that it is possible to pass from unstable structures to stable structures by properly defining the rules of the ψ-*ASM* program, then we will obtain a stable analog algorithm, i.e., we have managed to stabilize the unstable algorithm, i.e., the dynamical system is stabilizable.

Definition 30. An analog algorithm is said to be *stabilizable* if it is possible to avoid the unstable structures by properly defining the rules of the ψ-*ASM* program.

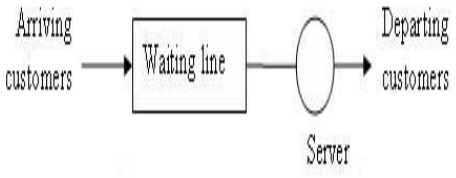

FIGURE 1. One server queuing system.

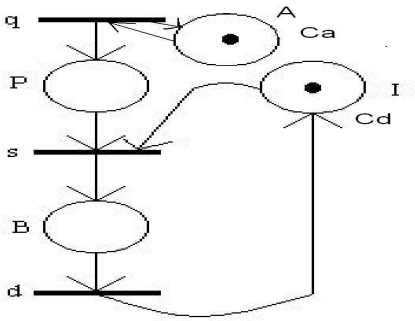

FIGURE 2. Timed Petri net model.

4. Discrete event dynamical systems: a case study [6, 7]

A discrete event system is a dynamical system whose state evolves in time by the occurrence of events at possibly irregular time intervals. Place-transitions Petri nets (commonly called just "Petri nets") are a graphical and mathematical modeling tool applicable to discrete event systems in order to represent its state evolution. Timed Petri nets are an extension of Petri nets that model discrete event systems where now the timing at which the state changes is taken into consideration. One of the most important performance issues to be considered in a discrete event dynamical system is its stability. By proving stability one is allowed to preassign the bound on the discrete event systems dynamics performance. (The reader not familiar with these concepts is encouraged to see [5, 6, 7] and the references quoted therein.)

Consider the simple one-server queuing system of Fig. 1, whose timed Petri net (TPN) model is depicted in Fig. 2. The events (transitions) that drive the system are: q – customers arrive to the queue; s – service starts; d – customer departs. The places (that represent the states of the queue) are: A – customers arriving; P – customers are waiting for service in the queue; B – customer is being served; I – server is idle. The holding times associated to the places A and I are Ca and Cd, respectively (with $Ca > Cd$). The Petri net (PN $[TPN]$) is unbounded since by the repeated firing of q, the marking in P grows indefinitely. However, employing Lyapunov and max-plus algebra techniques, it has been shown that by taking $u = [Ca, Ca, Ca]$, the PN is stabilizable, which implies that the TPN is stable [7]. This means that for the TPN to be stable and work properly the speed at which the one server queuing system works has to be equal to Ca, or being more precise, that all the transitions must fire at the same speed as the customers arrive. In other words, the customers have to be served as soon as they arrive in the queue, which is attained by setting $u = [Ca, Ca, Ca]$.

We have already discussed a ψ-ASM whose program describes the dynamical behavior of a Petri net (see 22, equation (3)), therefore setting in our program $m = 4$, $t = 3$, and $u = [Ca, Ca, Ca]$, we are able to bound the marking in P or, equivalently, avoid the unstable states of the queuing system, i.e., the set X_{un} of our analog algorithm. We conclude that by choosing properly the rules of the program ψ-ASM the analog algorithm for the one server queuing system is stabilizable.

5. Stability of analog algorithms in terms of Lyapunov energy functions

In this section, we consider the stability concept for analog algorithms in terms of Lyapunov energy functions. The results in this section are a generalization of what was discussed in Subsection 3.1 and Section 4, and include them as particular cases. We will deal with analog algorithms whose states are structures of vocabulary $\mathcal{V}$, where now the base set S is a metric space (S, d) with metric d.

Definition 31. Let us consider an analog algorithm. We will say that state X, with $a \in S$ and time-indexed location $f_{t,t_0}(a)$, where t and t_0 belong to T, is *stable* if $\forall t_0 \in T$ and $\forall \varepsilon > 0$, $\exists \delta = \delta(t_0, \varepsilon) > 0$ such that if given $a' \in S$, with $d(a', a) < \delta$, then $d([\![f_{t,t_0}(a')]\!]_X, [\![f_{t,t_0}(a)]\!]_X) < \varepsilon$, $\forall t \in T$.

Definition 32. A continuous function $\alpha : [0, \infty) \to [0, \infty)$ is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$.

Postulate D. The Lyapunov energy function associated to the analog algorithm at its starting time point $t_0 \in T$ bounds the whole Lyapunov energy function, transferred or transformed of the whole analog algorithm, as the Lyapunov energy function evolves in time.

Theorem 33. *Let us consider an analog algorithm. Assume there exists a Lyapunov function $V : S \times T \to \mathcal{R}^+$ and two functions α_1, α_2, which belong to $\mathcal{K}$, such that*

$$\alpha_1(d(\llbracket f_{t,t_0}(a') \rrbracket_X, \llbracket f_{t,t_0}(a) \rrbracket_X)) \leq V(\llbracket f_{t,t_0}(a') \rrbracket_X, t) \leq \alpha_2(d(\llbracket f_{t,t_0}(a') \rrbracket_X, \llbracket f_{t,t_0}(a) \rrbracket_X))$$

for all $a, a' \in S$, $t, t_0 \in T$. Assume Postulate D, and that $\llbracket f_{t_0,t_0}(a') \rrbracket_X = a'$ holds , then the analog algorithm is stable.

Proof. We want to show that there exists a $\delta = \delta(t_0, \varepsilon) > 0$ such that given a' with $d(a', a) < \delta \Rightarrow d(\llbracket f_{t,t_0}(a') \rrbracket_X, \llbracket f_{t,t_0}(a) \rrbracket_X) < \varepsilon$, $\forall t \in T$. Claim $\delta = \alpha_2^{-1}\alpha_1(\varepsilon)$ does the job. $d(\llbracket f_{t,t_0}(a') \rrbracket_X, \llbracket f_{t,t_0}(a) \rrbracket_X) \leq \alpha_1^{-1}(V(\llbracket f_{t,t_0}(a') \rrbracket_X, t)) \leq \alpha_1^{-1}(V(\llbracket f_{t_0,t_0}(a') \rrbracket_X, t_0)) = \alpha_1^{-1}(V(a', t_0)) \leq \alpha_1^{-1}\alpha_2(d(a', a)) < \varepsilon$, where Postulate D has been used in the second inequality and the equation $\llbracket f_{t_0,t_0}(a') \rrbracket_X = a'$ in the first equality . □

References

[1] Y. Gurevich, Sequential abstract-state machines capture sequential algorithms, *ACM Trans. Comput. Log.*, 1 (2000).

[2] N. Dershowitz, Y. Gurevich, *A natural axiomatization of computability and proof of Church's Thesis*, The Bulletin of Symbolic Logic, Vol. 14, 2008.

[3] O. Bournez, N. Dershowitz, P. Néron, *Axiomatizing analog algorithms*, Computability in Europe 2016: Pursuit of the Universal (CiE), Paris, France, Lecture Notes in Computer Science, vol. 9709, Springer-Verlag, Switzerland, pp. 215–224. Full version in ArXiv e-prints 1604.04295, `http://arxiv.org/abs/1604.04295`, 2016.

[4] H.B. Enderton, Elements of recursion theor, in *Handbook of Mathematical Logic*, J. Barwise, ed., North Holland, 1982, pp. 527–566.

[5] T. Murata, *Petri nets: Properties, analysis, and applications*, Proc. IEEE, Vol. 77, 1989.

[6] Z. Retchkiman, *Stability theory for a class of dynamical systems modeled with Petri nets*, International Journal of Hybrid Systems, Vol. 4, No. 1, 2005.

[7] Z. Retchkiman, *Timed Petri Net Modeling and Lyapunov/Max-Plus-Algebra Stability Analysis for a type of Queuing Systems*, International Journal of Pure and Applied Mathematics, Vol. 86 No. 2, 2013.

[8] H. Rogers, Jr., *Theory of recursive functions and effective computability,* McGraw-Hill, New York, 1967.

[9] A.M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2, vol. 42, parts 3 and 4, 1936.

[10] A.N. Michel, B. Hu, *Towards a stability theory of general hybrid dynamical systems*, Automatica, Vol. 35, 1999.

[11] V. Lakshmikantham, V.M. Matrosov, S. Sivasundaram, *Vector Lyapunov Functions and Stability Analysis of Nonlinear Systems,* Kluwer Academic Publ., Dordrecht, 1991.