Towards an Axiomatization of Simple Analog Algorithms

Olivier Bournez[1], Nachum Dershowitz[2], and Evgenia Falkovich[2]

[1] LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, France
Olivier.Bournez@lix.polytechnique.fr
[2] School of Computer Science, Tel Aviv University, Ramat Aviv, 69978 Israel
nachum.dershowitz@cs.tau.ac.il, jenny.falkovich@gmail.com

Abstract. We propose a formalization of analog algorithms, extending the framework of abstract state machines to continuous-time models of computation.

The states of 'continuous' machinery . . . form a continuous manifold, and the behaviour of the machine is described by a curve on this manifold. All machinery can be regarded as continuous, but when it is possible to regard it as discrete it is usually best to do so.

The property of being 'discrete' is only an advantage for the theoretical investigator, and serves no evolutionary purpose, so we could not expect Nature to assist us by producing truly 'discrete' brains.

Alan M. Turing, *Intelligent Machinery*, 1948

1 Introduction

We would like to gain an understanding of the fundamentals of analog systems, that is, systems that operate in continuous (real) time and with real values. There have been several different approaches that have led to continuous-time models of computations. One approach is inspired by continuous-time analog machines, and has its roots in models of natural or artificial analog machinery. An alternate approach, one that can be referred to as inspired by continuous-time system theories, is broader in scope, and derives from research in systems theory done from a computational perspective. Hybrid systems and automata theory, for example, are two such sources of inspiration. See the survey in [7].

At the outset, continuous-time computation theory was mainly concerned with analog machines. Determining which systems can actually be considered to be computational models is an intriguing question and relates to philosophical discussions about what constitutes a programmable machine. All the same, there were some early examples of actual analog devices that are generally accepted to be programmable machines. These include Pascal's 1642 *Pascaline* [10], Hermann's 1814 *Planimeter*, Bush's landmark 1931 *Differential Analyzer* [6], as well as Bill Phillips' 1949 water-run *Financephalograph* [1]. Continuous-time computational models also include neural networks and systems that can be built using electronic analog devices. Such systems begin in some initial state and evolve over time in response to input signals. Results are read off from the evolving state and/or from a terminal state.

Another line of development of continuous-time models was motivated by hybrid systems, particularly by questions related to the hardness of their verification and control. Here, models are not seen as models of necessarily analog machines, but, rather, as abstractions of systems about which one would like to establish some properties or derive verification algorithms.

Our goal is to capture all these models within one uniform notion of computation and of algorithm. The most interesting case is the hybrid one, where the system dynamics change in response to changed conditions, so there are discrete transitions as well as continuous ones. To that end, we adopt some of the ideas embodied in Gurevich's abstract-state machine formalism for discrete algorithms [14].

Abstract state machines (ASMs) constitute a most general model of sequential digital computation, one that can operate on any level of abstraction of data structures and native operations. It has been shown [15] that any algorithm that satisfies three "Sequential Postulates" can be step-by-step emulated by an ASM. These postulates formalize the following intuitions: (I) one is dealing with discrete, deterministic state-transition systems; (II) the information in states suffices to determine future transitions and may be captured by logical structures that respect isomorphisms; and (III) transitions are governed by the values of a finite and input-independent set of (variable-free) terms. All notions of algorithms for classical discrete-time models of computation in computer science, like Turing machines, random-access memory (RAM) machines, as well as classical extensions of them, including oracle Turing machines, alternating Turing machines, and the like, fall under the purview of the Sequential Postulates. This provides a basis for deriving computability theory, or even complexity theory, upon these very basic axioms about what an algorithm really is. In particular, adding a fourth axiom about initial states, yields a way to derive a proof of the Church-Turing Thesis [4,12,5], as well as its extended version about relative complexity [11].

Capturing the notion of an algorithm and a computation for analog systems is a first step towards a better understanding of computability theory for continuous-time systems. Even this first step is a non-trivial task. Some work in this direction has been done for simple signals. See, for example, [8,9] for an approach within the abstract-state machine framework. An interesting approach to specifying some continuous-time evolutions, based on abstract state machines and using infinitesimals, is [18]. However, a comprehensive framework capturing general analog systems seems to be wanting. See [7] for a discussion of the diverse analog computability theories.

Here, we adapt and extend ideas from work on ASMs to the analog case, that is to say, from notions of algorithms for digital models to analogous notions for *analog systems*. We go beyond the easier issue of "continuous space", that is, discrete-time models or algorithms with real-valued operations, since these have already been made to fit comfortably within the ASM framework, for which, see [2]. Indeed, algorithms for discrete-time analog models, like algorithms for the Blum-Shub-Smale model of computation [3], can be covered in this setting. The geometric constructions in [17] are simple (loop-free) examples of continuous-space algorithms.

In the next section, we introduce dynamical transition systems, defining signals and transition systems. In Sect. 3, we introduce abstract dynamical systems. Then, in Sect. 4, we define what an algorithmic dynamical system is. Finally, in Sect. 5, we define analog programs and provide some examples.

2 Dynamical Transition Systems

Analog systems may be thought of as "states" that evolve over "time". The systems we deal with receive inputs, called "signals", but do not otherwise interact with their environment.

2.1 Signals

Typically, a signal is a function from an interval of time to a "domain" value, or to a tuple of atomic domain values. For simplicity, we will presume that signals are indexed by real-valued time $\mathbb{T} = \mathbb{R}$, are defined only for a finite initial (open or closed) segment of $\mathbb{T}$, and take values in some domain D. Usually, the domain is more complicated than simple real numbers; it could be something like a tuple of infinitesimal signals. Every signal $u : \mathbb{T} \rightharpoonup D$ has a *length*, denoted $|u|$, such that $u(j)$ is undefined beyond $|u|$. To be more precise, the length of signals that are defined on any of the intervals $(0, \ell), [0, \ell), (0, \ell], [0, \ell]$ is ℓ. In particular, the length of the (always undefined) *empty* signal, ε, is 0, as is the length of any point signal, defined only at moment 0.

The *concatenation* of signals is denoted by juxtaposition, and is defined as expected, except that concatenation of a right-closed signal with a left-closed one is only defined if they agree on the signal value at those closed ends. The empty signal ε is a neutral element of the concatenation operation.

Let $\mathcal{U}$ be the set of signals for some particular domain D. The *prefix* relation on signals, $u \leq v$, holds if there is a $w \in \mathcal{U}$ such that $v = u\,w$. As usual, we write $u < v$ for *proper* prefixes ($u \leq v$ but $u \neq v$). It follows that $\varepsilon \leq u \leq uw$ for all signals $u, w \in \mathcal{U}$. And, $u \leq v$ implies $|u| \leq |v|$, for all u, v.

2.2 Transition Systems

Definition 1 (Transition System). *A* transition system $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{U}, \mathcal{T} \rangle$ *consists of the following:*

- *A nonempty set (or class) $\mathcal{S}$ of* states *with a nonempty subset (or subclass) $\mathcal{S}_0 \subseteq \mathcal{S}$ of* initial *states.*
- *A set $\mathcal{U}$ of input signals over some domain D.*

– *A $\mathcal{U}$-indexed family $\mathcal{T} = \{\tau_u\}_{u\in\mathcal{U}}$ of* state transformations $\tau_u : \mathcal{S} \to \mathcal{S}$.

It will be convenient to abbreviate $\tau_u(X)$ as just X_u, the state of the system after receiving the signal u, having started in state X. We will also use $X_{\widetilde{u}}$ as an abbreviation for the *trajectory* $\{X_v\}_{v<u}$, describing the past evolution of the state.

For simplicity, we are assuming that the system is deterministic. Note that the ASM framework, that is to say, the classical ASM framework for digital algorithms, though initially defined for deterministic systems, has been extended to nondeterministic transitions in [16,13].

Should one want to model the possibility of *terminal* states, then the transformations would be partial functions $\tau_u : \mathcal{S} \rightharpoonup \mathcal{S}$. We gloss over this distinction in what follows.

Definition 2 (Dynamical System). *A* dynamical system $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{U}, \mathcal{T}\rangle$ *is a transition system, where the transformations satisfy*

$$\tau_{uv} = \tau_v \circ \tau_u, \tag{1}$$

for all $u, v \in \mathcal{U}$, and where τ_ε is the identity function on states.

This implies that $X_{uv} = (X_u)_v$.

Remark 1. It follows from this definition that $\tau_{(uv)w} = \tau_{u(vw)}$, since composition is associative. It also follows that instantaneous transitions are idempotent. That is, $\tau_a \circ \tau_a = \tau_a$, for point signal a, because then $aa = a$.

3 Abstract Dynamical Systems

3.1 Abstract States

A vocabulary $\mathcal{V}$ is a finite collection of fixed-arity function symbols, some of which may be tagged *relational*. A term whose outermost function name is relational is termed *Boolean*.

Definition 3 (Abstract Transition System). *An* abstract transition system *is a dynamical transition system whose states $\mathcal{S}$ are (first-order) structures over some finite vocabulary $\mathcal{V}$, such that the following hold:*

(a) *States are closed under isomorphism, so if $X \in \mathcal{S}$ is a state of the system, then any structure Y isomorphic to X is also a state in $\mathcal{S}$, and Y is an initial state if X is.*

(b) *Input signals are closed under isomorphism, so if $u \in \mathcal{U}$ is a signal of the system, then any signal v isomorphic to u (that is, maps to isomorphic values) is also a signal in $\mathcal{U}$.*

(c) *Transformations preserve the domain (base set); that is,* Dom $X_u =$ Dom X *for every state $X \in \mathcal{S}$ and signal $u \in \mathcal{U}$.*

(d) *Transformations respect isomorphisms, so, if $X \cong_\zeta Y$ is an isomorphism of states $X, Y \in \mathcal{S}$, and $u \cong_\zeta v$ is the corresponding isomorphism of input signals $u, v \in \mathcal{U}$, then $X_u \cong_\zeta Y_v$.*

In particular, system evolution is *causal* ("retrospective"): a state at any given moment is completely determined by past history and the current input signal. This is analogous to the Abstract State Postulate for discrete algorithms, as formulated in [15], except that subsequent states X_u depend on the whole signal u, not just the prior state X and current input.

To keep matters simple, we are assuming (unrealistically) that all operations are total. Instead, we simply model partiality by including some *undefined* element $\perp$ in domains. See, however, the development in [2].

Vocabularies. We will assume that the vocabularies of all states include the Boolean truth constants, the standard Boolean operations, equality, and function composition, and that these are always given their standard interpretations. We treat predicates as truth-valued functions, so states may be viewed as algebras.

There are idealized models of computation with reals, such as the BSS model [3], for which true equality of reals is available in all states. On the other hand, there are also models of computable reals, for which "numbers" are functions that approximate the idealized number to any desired degree of accuracy, and in which only partial equality is available. See [2] for how to extend the abstract-state-machine framework to deal faithfully with such cases.

3.2 Locations in States

Locations. Since a state X is a structure, it interprets function symbols in $\mathcal{V}$, assigning a value b from Dom X to the "location" $f(a_1, \ldots, a_k)$ in X for every k-ary symbol $f \in \mathcal{V}$ and values $a_1, \ldots, a_k$ taken from Dom X. In this way, state X assigns a value $[\![t]\!]_X \in \text{Dom } X$ to any ground term t over $\mathcal{V}$. Similarly, a state X assigns the appropriate function value $[\![f]\!]_X$ to each symbol $f \in \mathcal{V}$.

States. It is convenient to view each state as a collection of the graphs of its operations, given in the form of a set of location-value pairs, each written conventionally as $f(a_1, \ldots, a_k) \mapsto b$, for $a_1, \ldots, a_k, b \in \text{Dom } X$. This allows one to apply set operations to states.

3.3 Updates of States

We need to capture the changes to a state that are engendered by a system. For a given abstract transition system, define its *update function* Δ as follows:

$$\Delta(X) = \lambda u.\ X_u \setminus X$$

We write $\Delta_u(X)$ for $\Delta(X)(u)$. The trajectory of a system may be recovered from its update function, as follows:

$$X_u = (X \setminus \nabla_u(X)) \cup \Delta_u(X) \tag{2}$$

where

$$\nabla_u(X) := \{\ell \mapsto [\![\ell]\!]_X : \ell \mapsto b \in \Delta_u(X) \text{ for some } b\}$$

are the location-value pairs in X that are updated by Δ_u.

4 Algorithmic Dynamic Systems

We say that states X and Y *agree*, with respect to a set of terms T, if $[\![s]\!]_X = [\![s]\!]_Y$ for all $s \in T$. This will be abbreviated $X =_T Y$. We also say that states X and Y are *similar*, with respect to a set of terms T, if or all terms $s, t \in T$, we have $[\![s]\!]_X = [\![t]\!]_X$ iff $[\![s]\!]_Y = [\![t]\!]_Y$. This will be abbreviated $X \sim_T Y$.

4.1 Algorithmicity

The current state, "modulo" its critical terms, unambiguously determines future states.

Definition 4 (Algorithmic Transitions). *An abstract transition system with states $\mathcal{S}$ over vocabulary $\mathcal{V}$ is* algorithmic *if there is a fixed* finite *set T of* critical *terms over $\mathcal{V}$, such that $\Delta_u(X) = \Delta_u(Y)$ for any two of its states $X, Y \in \mathcal{S}$ and signal $u \in \mathcal{U}$, whenever X and Y agree on T. In symbols:*

$$X =_T Y \Rightarrow \Delta_u(X) = \Delta_u(Y)\,. \tag{3}$$

This implies

$$X_{\widetilde{u}} =_T Y_{\widetilde{u}} \Rightarrow \Delta_u(X) = \Delta_u(Y)\,. \tag{4}$$

Furthermore, similarity should be preserved:

$$X_{\widetilde{u}} \sim_T Y_{\widetilde{v}} \Rightarrow X_{ua} \sim_T Y_{va}\,, \tag{5}$$

where $a \in \mathcal{U}$ is any point signal ($|a| = 0$).

Following the reasoning in [15, Lemma 6.2], every new value assigned by $\Delta_u(X)$ to a location in state X is the value of some critical term. That is, if $\ell \mapsto b \in \Delta_u(X)$, then $b = [\![t]\!]_X$ for some critical $t \in T$.

Proposition 1. *Every new value assigned by $\Delta_u(X)$ to a location in state X is the value of some critical term. That is, if $\ell \mapsto b \in \Delta_u(X)$, then $b = [\![t]\!]_X$ for some critical $t \in T$.*

Proof. By contradiction, assume that some b is not critical. Let Y be the structure isomorphic to X that is obtained from X by replacing b with a fresh element b'. By the abstract-state postulate, Y is a state. Check that $[\![t]\!]_Y = [\![t]\!]_X$ for every critical term t. By the choice of T, $\Delta_u(Y)$ equals $\Delta_u(X)$ and therefore contains b in some update. But b does not occur in Y. By (the inalterable-base-set part of) the abstract-state postulate, b does not occur in Y_u either. Hence it cannot occur in $\Delta_u(Y) = U_u - Y$. This gives the desired contradiction.

Agreeability of states is preserved by algorithmic transitions:

Lemma 1. *For an algorithmic transition system with critical terms T, it is the case that*

$$X =_T Y \Rightarrow X_u =_T Y_u \tag{6}$$

for any states $X, Y \in \mathcal{S}$ and input signal $u \in \mathcal{U}$.

4.2 Flows and Jumps

A "jump" in a trajectory is a change in the dynamics of the system, in contrast with "flows", during which the dynamics are fixed. Formally, a jump corresponds to a change in the equivalences between critical terms, whereas, when the trajectory "flows", equivalences between critical terms are kept invariant. Accordingly, we will say that a trajectory $X_{\widetilde{u}}$ *flows* if all intermediate states X_w and X_v ($\epsilon < w < v < u$) are similar. It *jumps* at its end if there is no prefix $w < u$ such that all intermediate X_v, $w < v < u$, are similar to X_u. It *jumps* at its beginning if there is no prefix $w \leq u$ such that all intermediate X_v, $\epsilon < v < w$, are similar to X.

4.3 Analgorithms

Putting everything together, we have arrived at the following.

Definition 5 (Analog Algorithm). *An* analog algorithm *(or "analgorithm") is an algorithmic (abstract) transition system, such that no trajectory has more than a finite number of (prefixes that end in) jumps.*

In other words, an analog algorithm is a signal-indexed deterministic state-transition system (Definitions 1 and 2), whose states are algebras that respect isomorphisms (Definition 3), whose transitions are governed by the values of a fixed finite set of terms (Definition 4), and whose trajectories do not change dynamics infinitely often (Definition 5).

4.4 Properties

System evolution is *causal* ("retrospective"): a state at any given moment is completely determined by past history and the current input signal.

Theorem 1. *For any analog algorithm, the trajectory can be recovered from the immediate past (or updates from the past). That is, X_u, for right-closed signal u, can be obtained (up to isomorphism) as a function of $X_{\widetilde{u}}$ (that is, the X_v, for $v < u$) plus the final input u_*.*

In fact, X_u depends on arbitrarily small segments $X_{u(t,|u|)}$ $(t < |u|)$ of past history.

Proof. This is a direct consequence of Definition 3. □

4.5 Further Considerations

It might also make sense to disallow the value given to a location ℓ at some time t to depend on infinitely many prior changes. For example, one would not want the value of $f(t)$ to be set at every moment t to $2f(t/2)$. Rather, the value of every location ℓ at moment t should be determined by values provided by the signal at time t and by values of locations in the state that are "stable" at t. By *stable*, we mean that there is a non-empty interval of time up to t in which its value is constant. Furthermore, this temporal dependency of locations should be well-founded.

It might also happen that the system of equations that controls transitions has a critical non-unique solution for the given initial conditions. For example, the equation $y'(x)^2 = 4y(x)$, restricted to the initial condition $y(0) = 0$, has two distinct solutions, namely, $y \equiv 0$ and $y = x^2$. In this case, we would want to add some continuity constraint. We would want to require that a choice of the solution made in the initial state is not changed for the whole trajectory governed by that equation.

5 Programs

5.1 Definition

Definition 6. *An* analog program *P over a vocabulary $\mathcal{V}$ is a finite text, taking one of the following forms:*

– *A* constraint *statement $v_1, \ldots, v_n$* **such that** *C, where C is a Boolean condition over $\mathcal{V}$ and the v_i are terms over $\mathcal{V}$ (usually subterms of C) whose values may change in connection with execution of this statement.*
– *A* parallel *statement $[P_1 \parallel \cdots \parallel P_n]$ $(n \geq 0)$, where each of the P_i is an ASM program over $\mathcal{V}$. (If $n = 0$, this is "do nothing" or "skip".)*
– *A* conditional *statement* **if** *C* **then** *P, where C is a Boolean condition over $\mathcal{V}$, and P is an ASM program over $\mathcal{V}$.*

We can use an assignment statement $f(s_1, \ldots, s_n) := t$ as an abbreviation for $f(s_1, \ldots, s_n)$ **such that** $f(s_1, \ldots, s_n) = t$. But bear in mind that the result is instantaneous, so that $x := 2x$ is tantamount to $x := 0$, regardless of the prior value of x. Similarly, $x := x + 1$ is only possible if the domain includes an "infinite" value ∞ for which $\infty = \infty + 1$.

5.2 Semantics

In the simple case, where the changes in state at time t depend only on the current signal u and state X, we can envision the following sequence of events:

(a) All non-stable locations in X (see Sect. 4.5) have undefined values.
(b) The signal sets the value of location $\imath$, yielding X'.
(c) Critical terms are evaluated in X'. (Only relevant terms need be evaluated, per [2].) This may involve looking up the values of pre-defined "static" operations in the state, like multiplication or division.
(d) All conditionals are evaluated, yielding a set of enabled constraints.
(e) All enabled constraints are solved (deterministically, we are assuming). In the explicit case, this means that all enabled assignments are "executed" in parallel, yielding a resultant state X''.

5.3 Examples

To begin with, consider analog algorithms that are purely flow, that is to say without any jumps.

In simple continuous-time systems, the state evolves continually, governed by ordinary differential equations, say. Flow programs invoke a time parameter, which we assume is supplied by the input signal.

Example 1 (Pendulum). The motion of an idealized simple pendulum is governed by the second-order differential equation

$$\theta'' + \frac{g}{L}\theta = 0\,,$$

where θ is angular displacement, g is gravitational acceleration, and L is the length of the pendulum rod. Let the signal $u \in \mathcal{U}$ be just real time. States report the current angle $\theta \in \mathcal{V}$. All states are endowed with the same (or isomorphic) operations for real arithmetic, including sine and square root, interpreting standard symbols. Initial states contain values for g, L, and the initial angle θ_0 when the pendulum is released.

For small θ_0, the flow trajectory $\tau_t(X)$ can be specified simply by

$$\theta = \theta_0 \cdot \sin\left(\sqrt{\frac{g}{L}} \cdot \imath\right)\,,$$

where $\imath$ is the input port and nothing but θ changes from state to state. The update function is, accordingly,

$$\Delta_t(X) = \left\{\theta \mapsto \theta_0 \cdot \sin\left(\sqrt{\frac{g}{L}} \cdot \imath\right)\right\}\,.$$

Hence, the critical term is $\theta_0 \cdot \sin(\sqrt{g/L} \cdot \imath)$.

It can be described by program

$$\left[\theta \textbf{ such that } \theta = \theta_0 \cdot \sin\left(\sqrt{\frac{g}{L}} \cdot \imath\right)\right]\,.$$

□

Example 2 (GPAC). One of the most famous models of analog computations is the General Purpose Analog Computer (GPAC) of Claude Shannon [19].

Figure 1 depicts a (non-mimimal) GPAC that generates sine and cosine: in this picture, the $\int$ signs denote some integrator, and the -1 denote some constant block. If initial conditions are set up correctly, such a system will evolve according to the following initial value problem:

$$\begin{cases} x' = z & x(0) = 1 \\ y' = x & y(0) = 0 \\ z' = -y' & z(0) = 0\,. \end{cases}$$

It follows that $x(t) = \cos(t)$, $y(t) = \sin(t)$, $z = -\sin(t)$.

In other words, this simple GPAC that generates sine and cosine can be modeled implicitly as a system with initial state having $x = 1; y = 0; z = 0$ and by a program

$$[x, y, z \textbf{ such that } x' = z \wedge y' = x \wedge z' = -y']\,,$$

where x', y', z' denote the derivatives of the corresponding functions.

The proposed model can also adequately describe systems (like a bouncing ball) in which the dynamics change periodically.

Example 3. The physics of a bouncing ball are given by the explicit flow equations

$$\begin{aligned} v &= v_0 - g \cdot t \\ x &= v \cdot t\,, \end{aligned}$$

where g is the gravitational constant, v_0 is the velocity when last hitting the table, and t is the time signal—except that upon impact, each time $x = 0$, the velocity changes according to

$$v_0 = -k \cdot v\,,$$

where k is the coefficient of impact. The critical Boolean term is $x = 0$. In any finite time interval, this condition changes value only finitely many times.

This system can be described by a program like

$$\begin{aligned} &[\textbf{if } x \neq 0 \textbf{ then } x, v \textbf{ such that } v = v_0 - g \cdot t, x = (v_0 - g \cdot t) \cdot t \\ &\parallel \textbf{if } x = 0 \textbf{ then } v_0 := -k \cdot v]\,, \end{aligned}$$

where x stands for its height, and v, its speed. Every time the ball bounces, its speed is reduced by a factor k.

□

References

1. Wikipedia. MONIAC computer. `http://en.wikipedia.org/wiki/MONIAC_Computer`.
2. Andreas Blass, Nachum Dershowitz, and Yuri Gurevich. Exact Exploration and Hanging Algorithms. In: Proceedings of the 19th EACSL Annual Conferences on Computer Science Logic (Brno, Czech Republic). Lecture Notes in Computer Science, Vol. 6247., Berlin, Germany, Springer (2010) 140–154. Available at `http://nachum.org/papers/HangingAlgorithms.pdf` (viewed June 3, 2011); longer version, Exact Exploration, at `http://nachum.org/papers/ExactExploration.pdf` (viewed May 27, 2011).
3. Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc. (NS)*, 21:1–46, 1989.
4. Udi Boker and Nachum Dershowitz. The Church-Turing Thesis over Arbitrary Domains. Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday, Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, eds., Lecture Notes in Computer Science, vol. 4800, Springer-Verlag, Berlin, pp. 199–229, 2008. Available at `http://nachum.org/papers/ArbitraryDomains.pdf` (viewed Jan. 10, 2012).
5. Udi Boker and Nachum Dershowitz. Three Paths to Effectiveness. Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday, Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, eds., Lecture Notes in Computer Science, vol. 6300, Springer-Verlag, Berlin, 2010. Available at `http://nachum.org/papers/ThreePathsToEffectiveness.pdf` (viewed Jan. 10, 2012).
6. Vannevar Bush. The differential analyser. *Journal of the Franklin Institute*, 212(4):447–488, 1931.
7. Olivier Bournez and Manuel L. Campagnolo. A survey on continuous time computations. In *New Computational Paradigms. Changing Conceptions of What is Computable* (Cooper, S.B. and Löwe, B. and Sorbi, A., Eds.). New York, Springer-Verlag, pp. 383-423. 2008.
8. Joëlle Cohen and Anatol Slissenko. On implementations of instantaneous actions real-time ASM by ASM with delays. *Proc. of the 12th Intern. Workshop on Abstract State Machines (ASM '2005)*, Paris, France, pp. 387–396, 2005.
9. Joëlle Cohen and Anatol Slissenko. Implementation of Sturdy Real-Time Abstract State Machines by Machines with Delays. *Proc. of the 6th Intern. Conf. on Computer Science and Information Technology (CSIT'2007)*, September 2007, Yerevan, Armenia. y of Science of Armenia.
10. Doug Coward. Doug Coward's Analog Computer Museum, 2006. `http://www.cowardstereoview.com/analog/` (viewed Jan. 10, 2012).
11. Nachum Dershowitz and Evgenia Falkovich. A Formalization and Proof of the Extended Church-Turing Thesis (Extended Abstract), *Studia Logica Conference on Trends in Logic, IX: Church Thesis: Logic, Mind and Nature*, Krakow, Poland, June 2011. Available at `http://nachum.org/papers/ECTT.pdf` (viewed Jan. 10, 2012).
12. Nachum Dershowitz and Yuri Gurevich. A natural axiomatization of computability and proof of Church's Thesis. *The Bulletin of Symbolic Logic*, 14(3):299-350, 2008. Available at `http://nachum.org/papers/Church.pdf` (viewed Apr. 15, 2009).

13. Andreas Glausch and Wolfgang Reisig. A semantic characterization of unbounded-nondeterministic abstract state machines *Algebra and Coalgebra in Computer Science*, Lecture Notes in Computer Science, vol. 4624, Springer, Berlin, pp. 242–256, 2007.
14. Yuri Gurevich. Evolving algebras 1993: Lipari guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995. Available at `http://research.microsoft.com/~gurevich/opera/103.pdf` (viewed Apr. 15, 2009)
15. Yuri Gurevich. Sequential abstract-state machines capture sequential algorithms. *ACM Transactions on Computational Logic*, 1, 2000, pp. 77–111. Available at `http://research.microsoft.com/~gurevich/opera/141.pdf` (viewed Apr. 15, 2009).
16. Yuri Gurevich and Tatiana Yavorskaya. On bounded exploration and bounded nondeterminism. Technical Report MSR-TR-2006-07, Microsoft Research, Redmond, WA. January 2006. Available at `http://research.microsoft.com/~gurevich/opera/177.pdf` (viewed Jan. 10, 2012).
17. Wolfgang Reisig. On Gurevich's theorem on sequential algorithms. Acta Informatica, 39(5):273–305, 2003.
18. Heinrich Rust. Hybrid abstract state machines: Using the hyperreals for describing continuous changes in a discrete notation. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, eds., *International Workshop on Abstract State Machines (Monte Verita, Switzerland)*, TIK-Report 87, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, pp. 341–356, March 2000.
19. Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics*, 20:337–354, 1941.

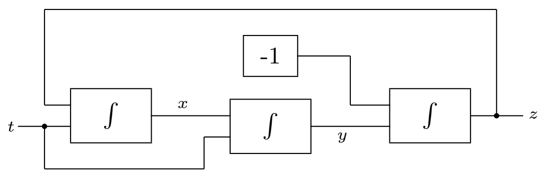

Fig. 1. A GPAC for sine and cosine.