From Finite Automata toward Hybrid Systems (Extended Abstract)

A. Rabinovich and B. A. Trakhtenbrot

Department of Computer Science, Tel Aviv University, Tel Aviv, Israel 69978

Abstract. We consider two orthogonal extensions of the basic finite automaton paradigm and clarify to what degree and in what form do they preserve fundamental facts from the theory of finite automata. Hopefully, this approach facilitates a lucid adaptation of Automata Theory to Hybrid Systems.

1 Introduction

Hybrid Systems (HS) incorporate both discrete and continuous dynamics; different models of HS, including clocked automata, are extensively investigated in the literature [3, 5]. The continuous aspects of these models may require incursions into Calculus (say into differential equations) which have little (if anything) in common with existing, well understood tools of Automata Theory and Logic. At the other side the adaptation of these tools may be (and sometimes happens to be) hampered and even obscured by the inherent complicacy of the model and also by adhoc definitions. In order to elucidate the underlying computational intuitions and to avoid reinvention of existing ideas we suggest to start with the separate consideration of two extensions of the basic finite automata paradigm. The first extension is relativization; it incorporates the idea of interaction with environment (oracle) and is easy to illustrate for discrete time. The second one focuses on the use of continuous time instead of discrete time and ignores interaction with the oracle. These two extension are conceptually orthogonal. We believe that they may facilitate a structured formalization of HS and a lucid adaptation (lifting) of basic automata theory to HS. This is in contrast with the situation today where there is a proliferation of models, adhoc terminology and notations.

Sections 2 and 3 survey some fundamentals of automata theory, which cover: (1) the notions of a retrospective operator (a sequential causal function from sequences to sequences) and its residual memory; (2) the Trinity [11] of basic specification formalisms. This trinity includes second order monadic logic (for which popular temporal logics are a sort of useful "syntactic sugar"), circuits or nets (i.e interconnected networks of retrospective operators) and labeled transition systems. The above survey is exposed in a manner which is invariant under the choice of time domain, so the primary semantic objects can be either sequences (time domain $\mathbf{N}$), signals or non-Zeno signals (time domain $\mathbf{R}$).

The main results of automata theory (recapitulated in Section 4) show that: (1) What is definable by the logic is what can be accepted by transition systems; (2) Boolean operators + delay form a basis with which one can construct circuits for any finite-memory retrospective operator; (3) One can determinize non-deterministic transition systems; (4) One can solve the Church synthesis problem for finite automata: given acceptor for a subset $R \subset X \times Y$ find an (uniformizing) automaton which computes $f : X \rightarrow Y$ such that $R(x, f(x))$ for every x in X and (5) Emptiness is decidable. All these are facts underlying the whole verification enterprise.

One way to extend the scope of automata theory (see Section 5) is to consider interconnecting the automaton with an "oracle" of some sort. This oracle can be a stack, a queue, a clock or a differential operator. Whatever the automaton can do while using this oracle is called its "relativization" wrt this oracle. In Section 5 the focus is on the case when the oracle itself is also a (may be infinite) discrete automaton. It turns out that some of the above stated results still hold when relativized. For example if a language is recognizable by an automaton using oracle F, then it is expressible using the same logic as before augmented with a symbol for F. If one adds F to the "classical" basis one can compute all F-relativized operators, etc. But other issues may hold or fail depending on specific properties of the oracle.

Another dimension of extension is to consider continuous time (see Section 6), so the primary semantic objects are signals (and in particular - non-Zeno signals) in time domain $\mathbf{R}$, retrospective operators on such signals etc.. Here an important distinction is to be made between speed-dependent and speed-independent operators. The later are just the continuous analog of discrete transducers and they are invariant under "stretching" of the time axis; the passage of time itself is not a cause for change in the output - they work on abstract time defined by input events. The delay operator is an example of a speed-dependent operator. For speed-independent operators it is shown that all classical results can be lifted from discrete to continuous time.

Finally, in Section 7, is suggested how the theory of real-time and hybrid systems can be developed around signal automata + oracles for continuous processes. We believe that this approach helps understanding real-time and hybrid systems by separating the issues of dense time and the physical oracles and by adapting the relevant heritage of Automata Theory.

2 Languages and Operators

2.1 Basic Concepts

Notations: $\mathbf{N}$ is the set of natural numbers; $\mathbf{R}$ is the set of real numbers, $\mathbf{R}^{\geq 0}$ is the set of non negative reals; **BOOL** is the set of booleans and Σ is a finite non-empty set.

A function from $\mathbf{N}$ to Σ is called an ω-string over Σ. A function from $\mathbf{R}^{\geq 0}$ to Σ is called a signal over Σ or Σ-signal. A signal x is non-Zeno if there exists an unbounded increasing sequence $\tau_0 = 0 < \tau_1 < \tau_2 \ldots < \tau_n < \ldots$ such that x is constant on every interval (τ_i, τ_{i+1}).

Note that for any non-Zeno signal x and any $\tau > 0$ there is $\tau' < \tau$ such that x is constant in the interval $(\tau',\ \tau)$; we denote the value of x in this interval by $llim(x)(\tau)$ or by $x(\tau - 0)$. We say that x is continuous at τ if there are $\tau_1,\ \tau_2$ such that $\tau_1 < \tau < \tau_2$ and x is constant in the interval $(\tau_1,\ \tau_2)$. Note that no signal is continuous at 0.

A set of ω-strings over Σ is called a ω-language over Σ. A function from ω-strings over Σ_1 to ω-strings over Σ_2 is called an ω-operator of type $\Sigma_1 \to \Sigma_2$. A function from signals over Σ_1 to signals over Σ_2 is called a signal operator of type $\Sigma_1 \to \Sigma_2$. A function from non-Zeno signals over Σ_1 to non-Zeno signals over Σ_2 is called a non-Zeno signal operator of type $\Sigma_1 \to \Sigma_2$.

An operator F is **retrospective** if Fx and Fy coincide in an interval $[0, t]$ whenever x and y coincide in $[0, t]$; an operator F is **strong retrospective** if Fx and Fy coincide in $[0, t]$ whenever x and y coincide in $[0, t)$.

An ω-operator G is the **residual** of an ω-operator F with respect to a string u of length k if $y = G(x)$ and $z = F(ux) \Rightarrow \forall \tau \geq 0.\ y(\tau) = z(\tau + k)$, where ux is the concatenation of u and x.

A signal operator G is the residual of a signal operator F with respect to a function $u:\ [0, a) \to \Sigma_1$ if $y = G(x)$ and $z = F(ux) \Rightarrow \forall \tau \geq 0.\ y(\tau) = z(\tau + a)$; here ux is the signal concatenation of u and x, i.e. $ux(\tau) = u(\tau)$ for $\tau < a$ and $ux(\tau) = x(\tau - a)$ for $\tau \geq a$.

G is a **residual** of F if there is u such that G is a residual of F with respect to u.

F has **finite memory** if the set of its residuals is finite.

A signal language L is **speed independent** if for every bijective increasing function $\rho:\ \mathbf{R}^{\geq 0} \to\ \mathbf{R}^{\geq 0}$ the following condition holds: $x \in L$ iff $x \circ \rho \in L$.

Note that if ρ is an increasing bijective function between non negative reals, then x is a non-Zeno signal iff $x \circ \rho \triangleq \lambda v.x(\rho(v))$ is a a non-Zeno signal. Hence the notion of speed independence is well defined for non-Zeno signal languages.

A signal operator F is speed independent if its graph is a speed independent language. It is easy to see that F is speed independent if $Fx = F(x \circ \rho) \circ \rho^{-1}$ for any monotonic bijection ρ on $[0, \infty)$.

2.2 Examples of Retrospective Operators

Example 1 (Finite memory operators).

Pointwise extensions: If f is a function from Σ_1 to Σ_2 then the pointwise extension P_f of f is defined as $P_f(x)(t) = f(x(t))$.
Unit Delays: $Delay_a$ is the ω-operator defined as $y = Delay_a(x)$ if $y(0) = a$ and $\forall \tau.\ y(\tau + 1) = x(\tau)$.

The next three operators are non-Zeno signals operators of type $\Sigma \to \Sigma$.

Left Limits: $y = leftlim_a(x)$ if $y(0) = a$ and $\forall \tau > 0.\ y(\tau) = x(\tau - 0)$.
Last jump value operator: $y = LJV_a(x)$ if

$$y(\tau) = \begin{cases} a & \text{if } \tau = 0\ . \\ x(\tau') & \text{if } x \text{ is constant in } (\tau',\ \tau) \text{ and } x \text{ is not continuous at } \tau'. \end{cases}$$

Last jump step operator: $y = LJS_a(x)$ if

$$y(\tau) = \begin{cases} a & \text{if } x \text{ is constant in } (0,\ \tau). \\ x(\tau' - 0) & \text{if there is } \tau' \text{ such that } 0 < \tau' < \tau \text{ and } x \text{ is constant in } (\tau',\ \tau) \\ & \text{and } x \text{ is not continuous at } \tau'. \end{cases}$$

Example 2 (Infinite memory). A strong retrospective operator $SqClock$ with infinite memory: $SqClock$ maps a binary string $a_0, \ldots a_n \ldots$ to a binary string $b_0, \ldots b_n \ldots$ as follows: $b(t) = 1$ if the number of steps elapsed since the last (before t) occurrence of 1 in $a_0, \ldots a_t$ is equal to the square of a positive natural number.

3 Three formalisms

We remind and shortly survey the essence of three specification formalisms: Monadic Logic, Nets and Labeled Transition Systems (LTS) which are fundamental in automata theory.

Warning. By abuse of notations we don't distinguish: (i) between a string over a structured alphabet $\Sigma_1 \times \Sigma_2$ and the pair of corresponding strings over Σ_1, Σ_2; (ii) between a subset of a set A and the corresponding characteristic predicate $A \to \{0,\ ,1\}$; (iii) between n-tuple of subsets of A and the corresponding function from A into alphabet $\{0,\ 1\}^n$. For example, the predicate $L(x_1,\ x_2)$ may be used for the corresponding set of sequences over $\Sigma_1 \times \Sigma_2$.

3.1 Monadic Second Order Theory of Order

Syntax The language $L_2^<$ of monadic second order theory of order has individual variables, monadic second order variables, a binary predicate $<$, the usual propositional connectives and first and second order quantifiers. We use $t,\ v$ for individual variables and $x,\ y$ for second order variables.

The atomic formulas of $L_2^<$ are formulas of the form: $t < v$ and $x(t)$. The formulas are constructed from atomic formulas by logical connectives and first and second order quantifiers.

We write $\psi(x, y, t, v)$ to indicate that the free variables of a formula ψ are among $x,\ y,\ t,\ v$.

Semantics A structure $K = \langle A,\ B, <_K\ \rangle$ for $L_2^<$ consists of a set A partially ordered by $<_K$ and a set B of monadic functions from A into $BOOL$.

We will be interested mainly in the following structures:

1. Structure $\omega = \langle \mathbf{N}, 2^{\mathbf{N}}, <_N\ \rangle$, where $2^{\mathbf{N}}$ is the set of all monadic functions from $\mathbf{N}$ into $BOOL$.
2. The structure $REAL = \langle \mathbf{R}, 2^{\mathbf{R}}, <_R\ \rangle$, where $2^{\mathbf{R}}$ is the set of all monadic functions from $\mathbf{R}$ into $BOOL$.
3. The signal structure Sig is defined as $Sig = \langle \mathbf{R}^{\geq 0},\ SIGNAL, <_R\ \rangle$, where $SIGNAL$ is the set of boolean non-Zeno signals.

The satisfiability relation $\tau_1, \dots \tau_m, \mathbf{x_1} \dots \mathbf{x_n} \models \psi(t_1, \dots t_m,\ x_1, \dots x_m)$ is defined in a standard way.

Definability Let $\phi(x)$ be an $L_2^<$ formula and $K = \langle A,\ B, <_K\ \rangle$ be a structure. We say that a set $C \subseteq B$ is definable by $\phi(x)$ if $\mathbf{x} \in C$ if and only if $\mathbf{x} \models_K \phi(x)$.

Example 3 (Interpretations of Formulas).

1. The formula $\forall t_1 \forall t_2.\ t_1 < t_2 \wedge (\neg \exists t_3.\ t_1 < t_3 < t_2) \rightarrow (x(t_1) \leftrightarrow \neg x(t_2))$ defines the ω-language $\{(01)^\omega,\ (10)^\omega\}$ in the structure ω and defines the set of all signals in the signal structure.
2. The formula $\exists y.\ \forall t.\ x(t) \rightarrow y(t) \wedge \forall t_1 \forall t_2 . t_1 < t_2 \wedge y(t_1) \wedge y(t_2) \rightarrow \exists t_3 . t_1 < t_3 < t_2 \wedge \neg y(t_3)$ defines in the structure ω the set of strings in which between any two occurrences of 1 there is an occurrence of 0. In the signal structure the above formula defines the set of signals that receive value 1 only at isolated points.
3. The formula $\forall t_1 \forall t_2 . t_1 < t_2 \wedge\ (x(t_1) \leftrightarrow x(t_2)) \rightarrow (\exists t_3.\ t_1 < t_3 < t_2 \wedge\ x(t_3) \leftrightarrow \neg x(t_2))$ defines the empty languages in the structure ω, the empty languages in the signal structure; In the structure REAL it defines the set of functions that receive value 1 and 0 in every non-empty open interval.

In the above examples, all formulas have one free second order variable and they define languages over alphabet $\{0,\ 1\}$. A formula $\psi(x_1, \dots x_n)$ with n free second order variables defines a language over alphabet $\{0,\ 1\}^n$.

3.2 Nets: Systems of Equations

Let $x_1, \dots, x_n$, and $y_1, \dots, y_m$ be distinct variable names and let F_i be $(n+m)$-ary functional symbols. A system of equation Sys is a set of equations $x_i = F_i(x_1, \dots, x_n,\ y_1, \dots, y_m)$ for $i = 1, \dots, n$.

Let D be a set and let $f_1, \dots, f_n :\ D^{n+m} \rightarrow D$ be interpretations of the symbols F_i.

1. A tuple $\langle a_1, \dots, a_n,\ b_1 \dots, b_m \rangle \in D^n \times D^m$ satisfies Sys under the interpretation $f_1, \dots, f_n$ if $\forall i.\ a_i = f_i(a_1, \dots, a_n,\ b_1, \dots, b_m)$.
2. Sys defines the sequence of functions $g_1, \dots, g_n\ : D^m \rightarrow D$ if

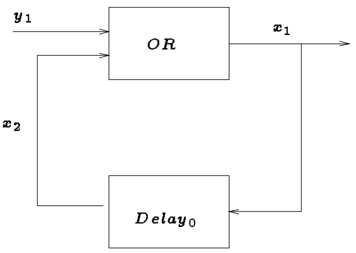
$$\langle a_1, \dots, a_n,\ b_1 \dots, b_m \rangle \text{ satisfies } Sys\ \Leftrightarrow a_i = g_i(b_1 \dots, b_m) \text{ for } i = 1, \dots, n.$$

   Also each of these g_i is said to be defined by Sys.

It is clear that if Sys defines functions then the set of tuples that satisfy Sys is the graph of a function from D^m to D^n.

Remark. It was assumed above that all functions F_i have the same arity. However, definition can be straightforwardly extended to the case where systems of equations are constructed over the functions of distinct arity. Also instead of one domain D many sorted functions are usually considered.

It is common to present systems of equations as nets. For example, in Fig 1 there is a net and the corresponding system of equations. The following functions g_1 and g_2 are defined by this net: $g_1(0^\omega) = g_2(0^\omega) = 0^\omega$, and $g_1(0^k 1w) = 0^k 1^\omega$ and $g_2(0^k 1w) = 0^{k+1} 1^\omega$.



$$Sys = \begin{cases} x_1 = OR(y_1,\ x_2) \\ x_2 = Delay_0(x_1) \end{cases}$$

Fig. 1. A net and its system of equations

Definition 1 (Basis). A set of function $\mathcal{F}$ is a basis for a set of function $\mathcal{G}$ if

1. $\mathcal{F} \subseteq \mathcal{G}$.
2. For every g in $\mathcal{G}$ there exists a system of equation Sys and $f_i, \ldots f_n \in \mathcal{F}$ such that g is definable by Sys under the interpretation $f_1, \ldots f_n$.

Example 4 (Classical Bases).

1. The function **not** and the function **and** form a basis for all boolean functions.
2. The pointwise extension of **not** and of **and** to ω-strings over $\{0,\ 1\}$ and the function $Delay_0$ from Example 1 form a basis for all finite memory retrooperators over ω-strings.

3.3 Labeled Transition Systems

Syntax A Labeled Transition System T is a triple $\langle Q,\ \Sigma,\ \rightarrow\rangle$ that consists of a set Q of states, a finite alphabet Σ of actions and a transition relation $\rightarrow$ which is a subset of $Q \times \Sigma \times Q$; we write $q \xrightarrow{a} q'$ if $\langle q,\ a,\ q'\rangle \in\rightarrow$; If Q is finite we say that the LTS is finite; We say that T is *deterministic* if for every q and $a \in \Sigma$ there exists at most one q' such that $q \xrightarrow{a} q'$; we say that T is *complete* if for every q and $a \in \Sigma$ there exists q such that $q \xrightarrow{a} q'$.

Sometimes the alphabet Σ of T will be the Cartesian product $\Sigma_1 \times \Sigma_2$ of other alphabets; in such a case we will write $q \xrightarrow{a,b} q'$ for the transition from q to q' labeled by the pair $(a,\ b)$.

Let T be an LTS over the alphabet $\Sigma_1 \times \Sigma_2$. We say that T is Σ_1-*deterministic* if for every q and $a \in \Sigma_1$ there exists at most one q' and $b \in \Sigma_2$ such that $q \xrightarrow{a,b} q'$. We say that T is Σ_1-*complete* if for every q and a there exist q' and b such that $q \xrightarrow{a,b} q'$.

An ω- **acceptor** $\mathcal{A}$ over Σ is a triple $\langle T,\ INIT(\mathcal{A}),\ FAIR(\mathcal{A})\rangle$, where $T = \langle Q,\ \Sigma,\ \rightarrow\rangle$ is an LTS over alphabet Σ; $INIT(\mathcal{A}) \subseteq Q$ - the initial states of $\mathcal{A}$. and $FAIR(\mathcal{A})$ - a collection of fairness conditions (subsets of Q).

A **transducer** of type $\Sigma_1 \rightarrow \Sigma_2$ is a pair $\langle T,\ q_0\rangle$, where T is an LTS over the alphabet $\Sigma_1 \times \Sigma_2$, which is Σ_1-deterministic and Σ_1-complete. Therefore, there are functions $nexstate:\ Q \times \Sigma_1 \rightarrow Q$ and $out:\ Q \times \Sigma_1 \rightarrow \Sigma_2$ such that $q \xrightarrow{a,b} q'$ iff $q' = nextstae(q,\ a)$ and $b = out(q,\ a)$.

Semantics Note that unlike the semantics of Monadic Logic and of nets, which are denotational, the semantics of LTS is operational. It relies on the consecutive execution of transitions which generates a run i.e. a finite or infinite sequence of steps. The acceptor semantics retains only those runs which meet the fairness/acceptance conditions.

A run of $\mathcal{A}$ is an ω-sequence $q_0a_0q_1a_1\dots$ such that $q_i \stackrel{a_i}{\to} q_{i+1}$ for all i. Such a run meets the initial conditions if $q_0 \in INIT(\mathcal{A})$. A run meets the fairness conditions if the set of states that occur in the run infinitely many times is a member of $FAIR(\mathcal{A})$.

An ω-string $a_0, a_1 \dots$ over Σ is accepted by $\mathcal{A}$ if there is a run $q_0a_0q_1a_1\dots$ that meets the initial and fairness conditions of $\mathcal{A}$. The ω-language *definable* (or accepted) by $\mathcal{A}$ is the set of all ω-strings acceptable by $\mathcal{A}$.

If $\mathcal{A}$ is a transducer of type $\Sigma_1 \to \Sigma_2$ then for every ω-sequence $a_0, a_1 \dots$ over Σ_1 there exist a unique ω-sequence $q_0, q_1 \dots$ of states and a unique ω-sequence $b_0, b_1 \dots$ over Σ_2 such that q_0 is the initial state of $\mathcal{A}$ and $q_i \stackrel{a_i,b_i}{\to} q_{i+1}$ for all i. Therefore, one can associate with $\mathcal{A}$ an ω-operator that maps $a_0, a_1 \dots$ to $b_0, b_1 \dots$ as above. This ω-operator is said to be *computable* by $\mathcal{A}$.

It is easy to see that the operator computable by a transducer is retrospective. If F is computable by a transducer $\langle T,\ q\rangle$ and $SAFE(\mathcal{A})$ is the set of all subsets of the states of T then the ω-language definable by $\langle T,\ q,\ SAFE(\mathcal{A})\rangle$ is the graph of F.

4 Finite Automata

The following two propositions are easy

Proposition 2. *For every retrospective ω-operator F of type $\Sigma_1 \to \Sigma_2$ there exists a transducer $Tran(F)$ that computes it.*

$Tran(F)$ can be defined as follows: (1) States - are all finite strings over Σ_1; (2) The initial state is the empty string ϵ; (3) $nextstate(a_1, \dots a_n,\ a) = (a_1, \dots a_n a)$ and (4) $b = out(a_1, \dots a_n,\ a)$ if b is the $n+1$-the letter of $F(a_1 \dots a_n a x)$ for some ω-string x (Note that b does not depend on x, due to retrospectivity of F). The following proposition is also easy:

Proposition 3 (Operators vs transducers). *An ω-operator is computable by a transducer iff it is a retrospective operator. An ω-operator is computable by a finite state transducer iff if it is a finite memory retrospective operator.*

Now we remind five issues from the theory of finite automata, one would like to extend for other computational models. The first two issues presume appropriate boolean encoding of the involved alphabets and display the convergence of the three fundamental specification formalisms: automata, logic, nets (see [11]).

Monadic Logic: An ω-language is definable by a finite state ω-acceptor iff it is definable by a monadic formula.

Nets: The set {pointwise extensions of **and**, $\neg$, unit delay} is a basis for ω-operators computable by finite state transducers (see Example 4).

Determinization: An ω-language is definable by an finite state ω-acceptor iff it is definable by a deterministic finite state ω-acceptor.

Uniformization: An ω-language L over alphabet $\Sigma_1 \times \Sigma_2$, which is definable by a finite state acceptor contains the graph of a retrooperator iff L contains a graph of retrooperator which is computable by a finite state transducer.

Emptiness Decidability: It is decidable whether the ω-language definable by a finite state acceptor is empty.

5 Interaction and Relativization

5.1 Product and Projection

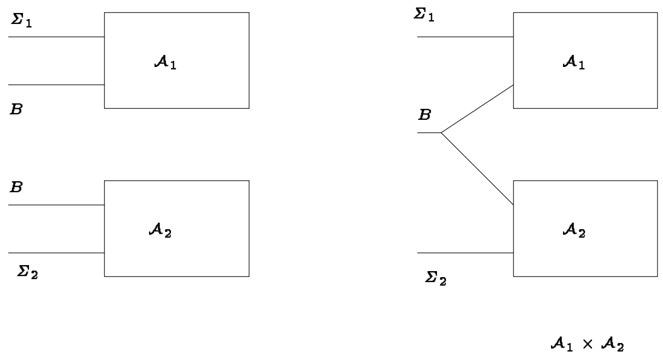


Fig. 2. Two acceptors and their product

Definition 4 (Product). Let T_i $(i = 1, 2)$ be labeled transitions systems with set Q_i of states, alphabet $\Sigma_i \times B$ and transition relations $\rightarrow_i$. Their product (notation $T_1 \times T_2$) is the following labeled transition system T

1. States: $Q = Q_1 \times Q_2$,
2. Alphabet: $\Sigma = \Sigma_1 \times \Sigma_2 \times B$
3. Transitions: $\langle q_1,\ q_2\rangle \xrightarrow{a_1, a_2, b} \langle q_1',\ q_2'\rangle$ iff $q_1 \xrightarrow{a_1, b}_1 q_1'$ in T_1 and $q_2 \xrightarrow{a_2, b}_2 q_2'$ in T_2.

The product of acceptors $\mathcal{A}_i = \langle T_i,\ INIT(\mathcal{A}_i) FAIR(\mathcal{A}_i)\rangle$ $i = 1,\ 2$ is is the acceptor $\mathcal{A} = \langle T,\ INIT(\mathcal{A}), FAIR(\mathcal{A})\rangle$ where T is the product of T_1 and T_2 and

4. $INIT(\mathcal{A}) = INIT(\mathcal{A}_1) \times INIT(\mathcal{A}_2)$ and $Q' \in FAIR(\mathcal{A})$ iff its projections onto $Q_1,\ Q_2$ are correspondingly in $FAIR(\mathcal{A}_1),\ FAIR(\mathcal{A}_2)$.

Remark. Note that the product of LTS which are deterministic and complete is not necessary complete.

Definition 5 (Projection). Let T be a labeled transition system over alphabet $\Sigma_1 \times \Sigma_2$. The projection of T on Σ_1 (notation $Project(T,\ \Sigma_1)$) is the LTS over alphabet Σ_1 with the same set of states as T and $q \xrightarrow{a} q'$ in $Project(T, \Sigma_1)$ if there is $b \in \Sigma_2$ such that $q \xrightarrow{a,b} q'$ is a transition of T. The projection of an ω-acceptor $\mathcal{A}$ is defined by taking the projection of its LTS. The initial states and fairness conditions of $Project(\mathcal{A}, \Sigma_1)$ are inherited from $\mathcal{A}$.

Remark. Note that if T is deterministic then its projection is not necessary deterministic. Hence the projection is not well defined on transducers.

Proposition 6. *Let $R_i(x_i, y)$ be the characteristic function of the ω-language defined by ω-acceptors $\mathcal{A}_i$ over alphabet $\Sigma_i \times B$. Then $R(x_1,\ x_2,\ y) = R_1(x_1,\ y) \wedge R_2(x_2, y)$ is the characteristic function of the ω-language accepted by $\mathcal{A}_1 \times \mathcal{A}_2$.*

Proposition 7. *Let $R(x,\ y)$ be the characteristic function of the ω-language defined by ω-acceptors $\mathcal{A}$ over alphabet $\Sigma_1 \times \Sigma_2$. Then $\exists y. R(x,\ y)$ is the characteristic function of the ω-language accepted by $Project(\mathcal{A},\ \Sigma_1)$.*

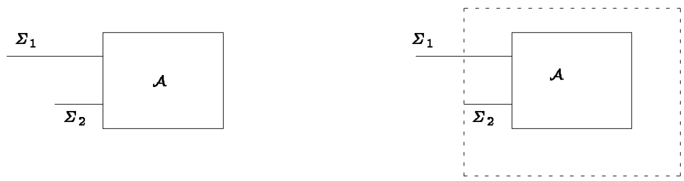


Fig. 3. Projection

5.2 Relativization with respect to acceptors/transducers

Let $\mathcal{A}$ be an acceptor over the alphabet $\Sigma_1 \times \Sigma_2$ and let $\mathcal{B}$ be an acceptor over the alphabet Σ_2. The **relativization** of $\mathcal{A}$ with respect to $\mathcal{B}$ (notation - $\mathcal{A}^{\mathcal{B}}$) is defined as $Project(\mathcal{A} \times \mathcal{B}, \Sigma_1)$. An ω-language L is *definable* by $\mathcal{A}$ with respect to oracle $\mathcal{B}$ if L is definable by $\mathcal{A}^{\mathcal{B}}$. In this case we say also that L is **$\mathcal{B}$-definable** by $\mathcal{A}$.

Note that if $\mathcal{A}$ is Σ_1-deterministic then $\mathcal{A}^{\mathcal{B}}$ is deterministic. We say that L is $\mathcal{B}$-definable by a deterministic ω-acceptor if there exists Σ_1-deterministic acceptor $\mathcal{A}$ such that L is B-definable by $\mathcal{A}$;

Remark. Above only a single occurrence of an oracle $\mathcal{B}$ is considered. The generalization for several occurrences (copies) of the oracle is immediate but notationally cumbersome. Therefore, we confine in the sequel with the simple occurrence version (see Theorem 9 and Theorem 10).

Let $\mathcal{A} = \langle T_1,\ q_1\rangle$ be a transducer of type $\Sigma_1 \times \Sigma_4 \to \Sigma_2 \times \Sigma_3$ and let $\mathcal{B} = \langle T_2,\ q_2\rangle$ be a transducer of type $\Sigma_3 \to \Sigma_4$, which computes a strong retrospective ω-operator. Then $\mathcal{A}^{\mathcal{B}}$ is defined as $\langle T,\ q\rangle$, where T is $Project(T_1 \times T_2, \Sigma_1 \times \Sigma_2)$ and q is the pair of the initial states q_1 and q_2. Under the above assumption it can be shown that $\mathcal{A}^{\mathcal{B}}$ is a transducer of type $\Sigma_1 \to \Sigma_2$ and, hence, computes an ω-retrooperator F. We say that F is computable by $\mathcal{A}$ with respect to oracle $\mathcal{B}$.

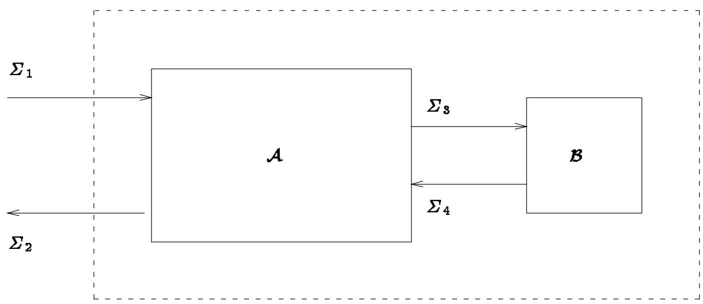


Fig. 4. Transducer $\mathcal{A}$ relativized wrt transducer $\mathcal{B}$

Remark. Note that if the retrospective operator computable by $\mathcal{B}$ is not strong retrospective then $\mathcal{A}^{\mathcal{B}}$ is not necessary a transducer.

Proposition 8 (Congruence). *Assume that (1) $\mathcal{B}_1$ and $\mathcal{B}_2$ define the same ω-language and (2) $\mathcal{A}_1$ and $\mathcal{A}_2$ define the same ω-language. Then $\mathcal{A}_1^{\mathcal{B}_1}$ and $\mathcal{A}_2^{\mathcal{B}_2}$ define the same ω-language. Similarly for transducers, if (1) $\mathcal{B}_1$ and $\mathcal{B}_2$ are transducers that compute the same strong retrospective operators and (2) transducers $\mathcal{A}_1$ and $\mathcal{A}_2$ compute the same retrospective operator then $\mathcal{A}_1^{\mathcal{B}_1}$ and $\mathcal{A}_2^{\mathcal{B}_2}$ compute the same retrospective operator.*

5.3 Relativization wrt ω-language L/strong retrooperator F

Let $\mathcal{B}$ be a transducer that computes a strong retrospective operator F; we say that an ω-language L' is F-definable by $\mathcal{A}$ if L' is $\mathcal{B}$-definable by $\mathcal{A}$. Proposition 8 shows that the particular choice of $\mathcal{B}$ does not matter. Similarly for F-computability, etc.

L is F-definable by a finite state acceptor if there is a finite state acceptor $\mathcal{A}$ such that L is F-definable by $\mathcal{A}$.

The role of Proposition 8 is rather to justify the robustness of the concepts above. The following theorem follows from Propositions 6 and 7 and it is crucial for the connection with logic.

Theorem 9. *Let $\mathcal{A}$ be an ω-acceptor over alphabet $\Sigma_1 \times \Sigma_2$ and let $\mathcal{B}$ be an ω-acceptor over alphabet Σ_2. Let $A(x, y)$ and $B(y)$ be the corresponding characteristic functions of the ω-languages definable by $\mathcal{A}$ and $\mathcal{B}$. Then $\exists y.\ A(x,\ y) \wedge B(y)$ is the characteristic function of the language definable by $\mathcal{A}^{\mathcal{B}}$.*

Remark. In the case of multiple occurrences of the oracle (see Section 5.2) the formula would be $\exists y_1 \exists y_2 \ldots \exists y_k.\ A(x,\ y_1, y_2, \ldots y_k) \wedge B(y_1) \wedge B(y_2) \ldots \wedge B(y_k)$.

Example 5 (A timed automaton with one SqClock). Consider the strong retrooperator SqClock from Example 2 and the transducer $Tran(SqClock)$ (see Proposition 2). SqClock-definability reminds definability by timed automata [3] up to the following peculiarities: (1) Discrete time; (2) Type of clock constraints. Actually, the operational semantics for timed automata as well as for hybrid automata [3, 4, 5] is also based on the product of a finite state automaton and a clock (oracle), though this is not mentioned explicitly.

5.4 Relativizing the Five Issues

Below F is a strong retrospective ω-operator and $R(x, y)$ is the characteristic function of its graph.

Theorem 10 *(Relativizing the Five Issues).*

Monadic logic: An ω-language L is F-definable by a finite state acceptor iff there is a monadic formula ψ such that $\exists x \exists y.\ \psi \wedge R(x\ y)$ is the characteristic function of L.

Nets: The set $\{F\} \cup \{$ pointwise extensions of **and**, $\neg$, unit delay$\}$ (see Example 4) is a finite basis for the class of all retrospective ω-operators which are F-computable by finite state transducers.

The following three statements may hold or fail depending on the choice of oracle F. It is clear that they hold when F has finite memory.

Determinization: An ω-language over alphabet Σ is F-definable by a finite state acceptor iff it is F-definable by a Σ-deterministic finite state acceptor.

Uniformization: An ω-language L over alphabet $\Sigma_1 \times \Sigma_2$ which is F-definable by a finite state acceptor contains the graph of a retrooperator iff L contains the graph of a retrooperator which is F-computable by a finite state transducer.

Emptiness Decidability: It is decidable whether the language F-definable by a finite state acceptor is empty.

6 Continuous Time

Unlike discrete time when an execution of a transition is considered as an instantaneous event, for continuous time the duration aspects become more relevant. Hence the attempts to use as appropriate syntax graphs in which both edges and nodes are labeled as in the transition $\langle q\ , a\rangle \xrightarrow{b} \langle q',\ a'\rangle$. ('Bitransition' seems to be more adequate). The intended semantics is that the edge represents an instantaneous activation of b (a jump) whereas the nodes represent lasting activations of a, a' (flows).

There are different formalizations of this approach in the literature and often they allow subsequent executions of jumps and subsequent executions of flows. We adopt essentially the approach in [4], but we require strong alternation of jumps and flows; in other words, the operational semantics amounts to the subsequent execution of 'bitransitions', each of them producing a pair ⟨ jump, flow ⟩. Below we present this semantics in a slightly modified way. The idea is to abstract first from the time dimension and handle the issue as in the discrete case.

A non-Zeno signal x over Σ belongs to the **image** of an ω-string $\langle a_0,\ b_0\rangle\langle a_1,\ b_0\rangle\ldots$ over alphabet $\Sigma\times\Sigma$ if there exists an unbounded increasing sequence $0=\tau_0<\tau_1<\ldots<\tau_i<\ldots$ of reals such that $x(\tau_i)=a_i$ and $x(\tau)=b_i$ for $\tau\in(\tau_i,\ \tau_{i+1})$. The image of an ω-language L is the union of the images of all ω-strings in L.

It is clear that if L is an ω-language over $\Sigma\times\Sigma$ then its image is a speed independent signal language over Σ.

Definition 11 (Signal acceptor). A signal acceptor of type Σ is an ω-acceptor over alphabet $\Sigma\times\Sigma$. A signal acceptor $\mathcal{A}$ is **deterministic** if $\mathcal{A}$ has only one initial state and for every state q of $\mathcal{A}$ and $a_1,\ a_2\in\Sigma$ there exists at most one state q' such that $q \xrightarrow{a_1,\ a_2} q'$.

Definition 12. A signal language L over Σ is defined by an signal acceptor $\mathcal{A}$ of type Σ if L is the image of the ω-language defined by $\mathcal{A}$ considered an an ω-acceptor.

Definition 13 (Signal transducer). A signal transducer of type $\Sigma_1\to\Sigma_2$ is a pair $\langle T,\ q\rangle$ where T is an LTS over the alphabet $\Sigma_1\times\Sigma_1\times\Sigma_2\times\Sigma_2$ for which there are functions $nextstate: Q\times\Sigma_1\times\Sigma_1\to Q$ and $out: Q\times\Sigma_1\to\Sigma_2$ such that

1. $q_1 \xrightarrow{a_1,a_2,b_1,b_2} q_2$ iff $q_2 = nextstate(q_1,\ a_1,\ a_2)$ and $b_1 = out(q_1,\ a_1)$ and $b_2 = out(q_2, a_2)$.
2. if $q_2 = nextstate(q_1,\ a_1,\ a_2)$ then $q_2 = nextstate(q_2,\ a_2,\ a_2)$.

Let $\mathcal{A}=\langle T,\ q\rangle$ be a signal transducer of type $\Sigma_1\to\Sigma_2$. Then the language defined by the signal acceptor[1] $\langle T,\ q,\ SAFE(T)\rangle$ is the graph of a speed independent retrooperator F on non-Zeno signals. We say that F is computable by $\mathcal{A}$.

Proposition 14 (Non-Zeno speed-independent operators vs transducers). *A non-Zeno signal operator is computable by a signal transducer iff it is a speed independent retrospective operator. A non-Zeno signal operator is computable by a finite state signal transducer iff it is retrospective operator with finite memory.*

Theorem 15 (Lifting the Five Issues From Discrete to Continuous Time).

Monadic logic: A signal language is definable by a finite state signal acceptor if and only if it is definable by a monadic formula.

Nets: The operators *leftlim*, *LVJ*, *LVS* (see Example 1) and the pointwise extensions of the boolean functions form a basis for signal operators computable by finite state signal transducers.

[1] recall that $SAFE(T)$ is the set of all subsets of the set of states of T

Determinization: A signal language is definable by a finite state signal acceptor iff it is definable by a deterministic finite state signal acceptor.

Uniformization: A signal language L over alphabet $\Sigma_1 \times \Sigma_2$ which is definable by a finite state signal acceptor contains the graph of a retrooperator iff L contains the graph of a retrooperator which is computable by a finite state signal transducer.

Emptiness Decidability: It is decidable whether the language definable by a finite state signal acceptor is empty.

7 Discussion

Relativization: Clearly, oracles with finite memory do not add expressive power; hence, the 'lifting' makes sense rather for oracles with infinite memory. For simplicity we considered single oracles but the generalization to multiple oracles and the reformulations of the corresponding results are straightforward. Note also that the 'classical' theory deals with constructive versions of the five issues and with corresponding algorithms of synthesis, analysis and verification of finite state systems. It is reasonable to investigate the impact of oracles for the constructive versions in terms of relative algorithms.

Continuous Time: We use non-Zeno signals as the main data structure. This is also the approach in [2], where in fact a more specific subclass of non-Zeno signals is considered.More popular is the use of timed sequences; however we find them less appropriate for our approach.

Comparison of two extensions: Note that lifting to continuous time essentially preserves the five classical issues as well as their constructive versions. However the particular choice of the oracles may preserve some of these issues and change others. That is not surprising if one takes into account experience with relativization in logic and recursion theory.

Comparison of LTS vs Logic and Nets: There are different versions for definitions of acceptors over continuous time. Unfortunately, none of these versions (including our own) is robust like the LTS machinery in the discrete time case. This is in full contrast with monadic logic and nets, where reference to discrete or continuous time are equally persuading.

Though we considered here only linear time models, many of the results can be extended to the case where time is not linear (branching).

Toward more complicated models: The first step is to combine continuous time with oracles and to examine the five issues. A work in this direction was done in [10]. The further development is to consider continuous activities. A simple way to include them into the picture is to consider three additional kinds of retrooperators: (1) "Laws" which map functions from $\mathbf{R} \to \mathbf{R}$ into functions from $\mathbf{R} \to \mathbf{R}$ (e.g., differential operators). (2) "Sensors", which map functions from $\mathbf{R} \to \mathbf{R}$ into signals and (3) "Actuators" which map signals to functions in $\mathbf{R} \to \mathbf{R}$. An obvious folklore idea (see Fig. 5) is to use the composition of an Actuator, a Law and a Sensor as an oracle from signals to signals. This paves the way for the Automata Theoretical machinery assumed the behavior of this oracle and of the functions are well understood. Of course such understanding may require tools beyond automata theory.

Related work: There is an extensive literature in the area and we mention here only the following sources which suggested the issues to be lifted from classical Automata Theory: Monadic Logic: [13], Decidability and Determinization: [3], Uniformization [8].

Note that nets as a specification formalism are not considered in those works. Nets are considered in [10] where a finite basis for signal operators computable by finite state transducers is given. The point of view that transducers and nets are conceptually more fundamental than acceptors (and in particular than nondeterministic acceptors) is implicit already in the classical works of Pitts-Mc.Culloch [9], Kleene [7]. It was consistently pursued in [6], see also Scott's argumentation in [12]. Finally, let us mention the Abadi - Lamport appeal [1] to abstain from invention and proliferation of new semantical models whenever old-fashioned recipes do the job.

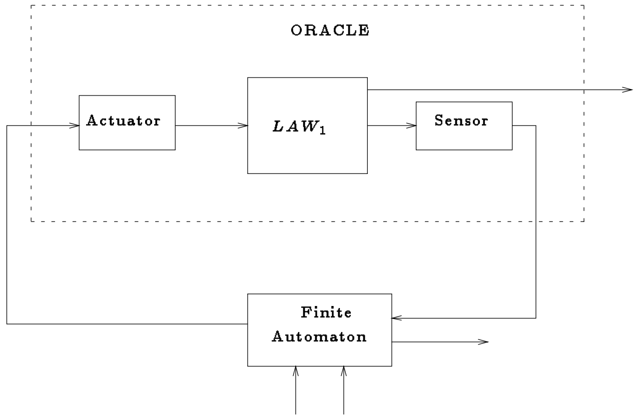


Fig. 5. A Hybrid System

Acknowledgments

Thanks to Oded Maler for discussions. His review of our results is almost literally reproduced in the introduction.

References

1. M. Abadi and L. Lamport. An Old-Fashioned Recipe for Real Time . In ACM Toplas 16, 5:1543-1571, 1994.
2. R. Alur, C. Courcubetis, T. Henzinger and P. Ho. Hybrid Automata, an algorithmic approach to the specification and verification of hybrid systems. LNCS 736, 1993.
3. R. Alur and D. Dill. A theory of timed automata. Theoretical Computer Science,126:183-235,1994.
4. R. Alur, T. Henzinger and P. Ho. Automatic Symbolic Verification of Embedded Systems, 1993.
5. T. Henzinger. A theory of hybrid automata. In *Proceedings of LICS*, 1996.
6. N. Kobrinski and B. A. Trakhtenbrot. Introduction to the theory of Finite Automata, North Holland 1965.
7. S. Kleene. Representation of events in nerve nets and finite automata. Automata Studies, 3-41, Princeton, 1956.
8. O. Maler, A. Pnueli and J. Sifakis. On the synthesis of discrete controller for timed systems. In *Theoretical Aspects of Computer Science*, LNCS vol 900, 1995.
9. R. McCulloch and W. Pitts. A logical calculus of the ideas immanent inn nervous activity. Bull.. Math. Biophysics, 5, 1943, pp. 115-133.
10. D. Pardo, A. Rabinovich and B. A. Trakhtenbrot. On Synchronous Circuits over Continuous Time. Technical Report Tel Aviv University, 1997. Available at `http://www.math.tau.ac.il/~rabino`.
11. B. Trakhtenbrot. Origins and Metamorphoses of the Trinity: Logics, Nets, Automata. In *Proceedings of LICS*, 1995.
12. D. Scott. Some definitional suggestions for automata theory. J. of Computer and System Science, 1967, pp.187-212.
13. T. Wilke. Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata, FTFRFT, 1994.