Automata and Their Interaction

Boris A. Trakhtenbrot

June 2000

1 Automata

a) Notational provisos
Z is a space (alphabet), T is the time-domain, i.e. the set N of nonnegative integers, or the set $R^{\geq 0}$ of nonnegative reals. A Z-path, is a function $\tilde{z}$ from T into Z, whose domain is a left-closed (possibly infinite) interval $[t, t+\delta)$ or $[t, t+\delta]$. $\tilde{Z}$ is the set of z-paths. The interval is said to be the *life-time* of $\tilde{z}$; its length δ is said to be the duration $|\tilde{z}|$ of $\tilde{z}$. A path is standard if $t = 0$. The standard shift of $\tilde{z}$ is the standard path whose value at time τ coincides with $\tilde{z}(t+\tau)$. In many respects we do not distinguish between paths with the same standard shift, and we preserve for them the same notation.

Unless stated otherwise, we confine ourselves to paths whose life-times are semi-intervals; their concatenation $\tilde{z}_1.\tilde{z}_2$ is defined in the usual way.

b) A reminder
A discrete-time deterministic automaton M is given by a (state- alphabet) X, an (action-alphabet) U and a map *nextstate* $: X \times U \longrightarrow X$. The associated *terminal transition map* Ψ of type $X \times \tilde{U} \longrightarrow X$ obviously extends *nextstate* from singleton action u to action sequence $\tilde{u} = u_1 ... u_l$. Finally, $\tilde{\Psi} : X \times \tilde{U} \longrightarrow \tilde{X}$ is the associated *full transition map*. When applied to state x and an action sequence $\tilde{u}$ it returns the state-sequence of length $l+1$, that starts with x and leads to the terminal state $x' = \Psi(x, \tilde{u})$.

Comments: In general, there are no restrictions on the cardinalities of U, X. For example, let X be the Euclidean space R^n, and let U consists of (names of) appropriate matrices. Then, *nextstate* may perform the corresponding linear transforms of R^n. But algebraic assumptions about X, U are not part of the model. The terminal and full transition maps are uniquely determined as soon as *nextstate* is given. But for continuous time domain $R^{\geq 0}$, *nextstate* does not make sense; hence, the need for a direct definition of terminal/full transitions.

2 Deterministic Automata

a) Basics
Even though the forthcoming formulations are general (hence, applicable for discrete time as well), they are intended mainly for the less routine continuous time-domain. (In [S1] these are also called Dynamic Systems or Machines).

An automaton M is given by a state- space X, an action-space U and a partial *terminal transition map* $\Psi : X \times \tilde{U} \rightarrow X$.

The intended semantics is: if state x occurs at time t, then $\tilde{u}$ with life-time $[t, t+\delta)$ produces state x' at time $t+\delta$.

A finite path $\tilde{u}$ is admissible for x iff $\Psi(x, \tilde{u})$ is defined.

An infinite path $\tilde{u}$ is admissible for x if all its finite prefixes are admissible for x.

Below, $\tilde{u}_1, \tilde{u}_2 \in \tilde{U}$ are finite paths, $\tilde{u}_1.\tilde{u}_2$ designates their concatenation, and ϵ is the empty action path.

Axioms

(i) Non-triviality. For each state x there is a non-empty action path $\tilde{u}$, which is admissible for x.

(ii) Semi-group.

$$\Psi(x, \epsilon) = x$$

$$[\Psi(x, \tilde{u}_1) = x' \ \& \ \Psi(x', \tilde{u}_2) = x''] \to \Psi(x, \tilde{u}_1.\tilde{u}_2) = x''$$

(iii) Restriction (Density). Assume that $\Psi(x, \tilde{u}) = x''$. If $\tilde{u} = \tilde{u}_1.\tilde{u}_2$ then there exists x' such that $\Psi(x, \tilde{u}_1) = x' \ \& \ \Psi(x', \tilde{u}_2) = x''$.

Semi-group closure. An automaton M may be defined as follows:

(i) Consider Ψ for some set Ω of admissible paths, and check the non-triviality and restriction axioms;

(ii) Extend Ψ (and preserve this notation!) via concatenations of paths from Ω.

Full transition map of M. This is a function $\tilde{\Psi} : X \times \tilde{U} \longrightarrow \tilde{X}$, which returns a path $\tilde{x}$ with the same life-time $[0, \delta)$ as that of $\tilde{u}$. Namely,

$$\tilde{\Psi}(x, \tilde{u}) = \tilde{x} \ \ iff \ \ \forall t \in [0, \delta).\Psi(x, \tilde{u}|t) = \tilde{x}(t)$$

It follows from the axioms, that for (deterministic!) M, the definition is correct, i.e. $\tilde{\Psi}$ is indeed a function.

The pair $(\tilde{u}, \tilde{x})$ is a finite *trajectory* of M, and $\tilde{x}$ is a finite *state-path*

Note, that unlike Ψ, the extension of $\tilde{\Psi}$ (hence also of state-paths and of trajectories) to infinite duration is straightforward.

b) Flows

Definition 1 *A flow on the state-space X is a function $f : X \times T \longrightarrow X$, that meets the following conditions: (i) $f(x, 0) = x$; (ii) if $f(x, t)$ is defined, so is $f(x, t')$ for each $0 < t' < t$ and*
(iii) additivity:

$$f(x, t_1) = x' \ \& f(x', t_2) = x'' \ \longrightarrow f(x, t_1 + t_2) = x''$$

Notation. nil is the (polymorphic) trivial flow: $\forall t[nil(x, t) = x]$

Example 1 $f(x, t) \overset{def}{=} x + t$.

Clearly, a flow is nothing but the transition map Ψ of an automaton M, whose action-space is a singleton. *Flow* is a pure semantical notion, without any commitment to specific syntax. Note, however, that in Control Theory the favorite way to describe flows is via differential equations.

Example 2 *Consider a finite-dimensional differential equation*

$$\dot{x} = F(x), \tag{1}$$

where x is an n-dimensional vector. Assume the existence of unique solutions of this equation for arbitrary initial conditions and for arbitrary time intervals. (This can be guaranteed under appropriate technical assumptions, which are not relevant here.) The associated flow f is defined as follows:

Given a time interval $[t_0, t]$ and an initial state $x(t_0) = x_0$, consider the corresponding solution x of equation (1) on that interval. Then, $f(x_0, t) \overset{def}{=} x(t)$.

c) Structured spaces (alphabets)

Assume $W = V \times Z$, so that each path $\tilde{w}$ is uniquely defined by the pair $\{\tilde{v}, \tilde{z}\}$ of its projections into V and Z; we do not distinguish between path $\tilde{w}$ and that pair, for which we use notation $\tilde{v}\tilde{z}$. Specific delimiters and/or ordering restrictions may be used for additional information. For example $\tilde{v}/\tilde{z}$ or $\tilde{v} \longrightarrow \tilde{z}$ may characterize $\tilde{v}$ as an argument value, and $\tilde{z}$ as the corresponding function value.

In the sequel, the state-space X will be structured only as the Cartesian product of a set of *components* $X_1, X_2, \ldots, X_m$. On the other hand, the action space U may be structured via a set of *components* $U_1, U_2, \ldots, U_k$ in two different formats:

a) Multiplicative (Cartesian) format: $U = U_1 \times U_2 \ldots \times U_k$

b) Additive (disjoint sum) format: $U = U_1 + U_2 \ldots + U_k$

Respectively, two formats of admissible action paths will be considered.

(i) In the multiplicative format such a path belongs to $\tilde{U}$.

(ii) In the additive format an additional assumption is needed:

Interleaving structure. Each admissible action-path is the concatenation of "straight" paths $\tilde{u}_i \in \tilde{U}_i$. Hence (by semigroup closure), it suffices to define transitions only for *straight* paths.

Clearly, for discrete (but not for continuous) time the interleaving structure is guaranteed for all paths that belong to $\tilde{U}$.

3 Behavior of Initialized Automata

a) Retrospection

Notation. $\alpha|\tau$ is the prefix of path α restricted to life-time $[0, \tau)$.

Definition 2 *f is a retrospective operator (shorthand: retrooperator) of type $\tilde{U} \to \tilde{Y}$ if it is defined on a prefix-closed subset of $\tilde{U}$ and satisfies the condition:*

$$\text{If } \tilde{y} = f(\tilde{u}), \text{ then } \tilde{y}|\tau = f(\tilde{u}|\tau) .$$

In other words, for each t the value $\tilde{y}(t)$ depends only on the values of $\tilde{u}$ on the right closed interval $[0, t]$. When $\tilde{y}(t)$ does not depend on $\tilde{u}(t)$, the operator f is said to be *strongly retrospective*.

Clearly, for each initial state $x_0 \in X$, the full transition map $\tilde{\Psi}$ induces a strong retrospective operator X, which is called the input/state behavior (i/o behavior) of the initialized automaton $< M, x_0 >$.

b) Accepted sets

The behavior of $< M, x_0 >$ can also be characterized by a set (language) which consists of all (or of an appropriate part of) the action paths (of the trajectories or the state-paths, respectively) admissible at x_0. This is the action (respectively, trajectory or state) set, accepted by $< M, x_0 >$. Most important is infinite behavior that obeys some reasonable fairness conditions. Fairness is beyond our subject, so, when referring to infinite paths, all admissible infinite paths are assumed.

If the action-space is structured as $U \times V \times ...$ then the accepted set is presented naturally by a characteristic relation $L(\tilde{u}, \tilde{v}, ...)$, called *relational behavior* of the automaton.

It may happen that for some partition of the arguments in L, the relational behavior is the graph of a function, so one could refer to the corresponding *functional behavior*. Note that for a given L there may happen to be different "functional" partitions of this kind.

c) Transducers

Implicit transducers. Consider, for example, a retrooperator F of type $\tau = \tilde{U} \times \tilde{A} \longrightarrow \tilde{B}$. Let $< M, x_{init} >$ be an initialized automaton with state-space X and action alphabet $U \times A \times B$.

Definition 3 *$< M, x_{init} >$ is a implicit transducer of type τ with input/output behavior (i/o behavior) F iff it accepts the graph of F (hence $\Psi_M(x_{init}, \tilde{u}\tilde{a}\tilde{b})$ is defined* iff *$\tilde{b} = F(\tilde{u}\tilde{a})$).*

Remark 1 *In the case above it might be convenient to use the mnemonic notation $\Psi_M(x_{init}, \tilde{u}\tilde{a} \,/\, \tilde{b})$ which points out the type of the intended behavior. Note, that an automaton M may happen to be typable in different ways as an implicit transducer.*

Explicit transducers: strong and weak readouts. Let M be an automaton with spaces X, U. Consider in addition: (i) a space Y of output (or measurement) values, and (ii) a map $h : X \longrightarrow Y$. The pair $< M, h >$ is said to be an explicit transducer with underlying automaton M and *strong readout* map h. Let $G : \tilde{U} \longrightarrow \tilde{X}$ be the *i/s* behavior of the underlying automaton M. Then, the *i/o*-behavior of the transducer $< M, h >$ is the retrooperator $F : \tilde{U} \longrightarrow \tilde{Y}$, defined as follows:

Assume $\tilde{x} = G(\tilde{u})$; then $F(\tilde{u})$ returns $\tilde{y}$ such that

$$\forall t \leq |\tilde{u}|.\ \tilde{y}(t) = h(\tilde{x}(t)) \tag{2}$$

Clearly, like G, the retrooperator F is also strongly retrospective.

A *weak readout* map h (not considered in [S1]!) is of type $X \times U \longrightarrow Y$. The definition of i/o-behavior in (2) should be modified, namely, $\tilde{y}(t) = h(\tilde{x}(t))$ is replaced by $\tilde{y}(t) = h(\tilde{x}(t), \tilde{u}(t))$. The i/o behavior is still retrospective, but strong retrospection is no longer guaranteed.

d) Comparing transducers and operators

Implicit transforms. Consider an explicit transducer $< M, h >$ with i/o behavior $F : \tilde{U} \times \tilde{A} \longrightarrow \tilde{B}$.

Assume that $< M, h >$ is specified by $\Psi : Q \times \tilde{U} \times \tilde{A} \longrightarrow Q$ and (for simplicity) by a strong readout $h : Q \longrightarrow B$.

Definition 4 *Let M' be the automaton with transition map $\Psi' : Q \times \tilde{U} \times \tilde{A} \times B \longrightarrow Q$, defined below ($\delta$ is the common length of the paths):*

$$\Psi'(q, \tilde{u}\tilde{a}/\tilde{b}, \delta) = q' \text{ iff } \tilde{\Psi}(q, \tilde{u}\tilde{a}, \delta) = \tilde{q} \ \ \& \ \ q' = \tilde{q}(\delta) \ \ \& \ \ \forall\, t < \delta.\ \tilde{b}(t) = h(\tilde{q}(t)).$$

Say that M' is the implicit transform of $< M, h >$, and $< M, h >$ is the explicit transform of M'.

It is easy to see that M' is an implicit transducer with i/o behavior F (the same as that of $< M, h >$).

Clearly, an implicit transducer is not necessarily the implicit transform of an explicit transducer.

Consider three properties of an operator F:

(i) F is a retrospective operator,

(ii) F is the i/o-behavior of an implicit transducer,

(iii) F is the i/o-behavior of an explicit transducer.

Proposition 1 *These properties are equivalent.*

That (ii) and (iii) imply (i) is trivial. The implication (iii) $\longrightarrow$ (ii) is due to implicit transforms. The other directions are not trivial.

4 Interaction

Interacting agents are (possibly infinite) automata with structured spaces (alphabets).

For discrete-time automata the binary interaction combinators are defined in a routine way, via derivation of the *nextstate*-map (*nextstate*-relation) for the composition M from the *nextstate*-maps (*nextstate*-relations) of the components M_i. However, one should be careful with continuous time, when one needs to deal directly with terminal, or even with full transition maps. For these reasons we start with *nextstate* for discrete time, and then provide motivated definitions for terminal and full transition maps, which cover both discrete and continuous time.

Note that in all cases below, if the components are deterministic, so is their composition. Finally, the combinators are commutative and associative; hence, compositions may be considered for arbitrary sets $\{M_1, M_2, M_3, \ldots\}$ of components.

5 Synchrony vs. Asynchrony

a) Synchrony
For simplicity consider M_i with state-space $X_i \times X_0$ and action-space $U_i \times U_0$. Call the X_i – ports, and the U_i – registers. Note, that the components are allowed to share registers (here - X_0) and ports (here - U_0).

Definition 5 (Synchronous composition: $M = M_1 \times M_2$). *The state-space of M is $X_1 \times X_0 \times X_2$, the action-space is $U_1 \times U_0 \times U_2$, and the transitions are as follows:*

$$M(x_1x_0x_2, u_1u_0u_2, x_1'x_0'x_2') \overset{def}{=} M_1(x_1x_0, u_1u_0, x_1'x_0') \ \& \ M_2(x_2x_0, u_2u_0, x_2'x_0') \tag{3}$$

In particular, for deterministic M_i

$$\begin{aligned} &nextstate(x_1x_0x_2, u_1u_0u_2) = x_1'x_0'x_2' \ \text{ iff} \\ &nextstate_1(x_1x_0, u_1u_0) = x_1'x_0' \ \& \ nextstate_2(x_2x_0, u_2u_0) = x_2'x_0' \end{aligned} \tag{4}$$

Warning. Note, that (4) cannot be extended to terminal transition maps in the form

$$\Psi(x_1x_0x_2, \tilde{u}_1\tilde{u}_0\tilde{u}_2) = x_1'x_0'x_2' \ \text{ iff } \ \Psi_1(x_1x_0, \tilde{u}_1\tilde{u}_0) = x_1'x_0' \ \& \ \Psi_2(x_2x_0, \tilde{u}_2\tilde{u}_0) = x_2'x_0' \tag{5}$$

The reason is that in (5), for given $\tilde{u}_1\tilde{u}_0\tilde{u}_2$, the M_i are required only to reach the same terminal (!) value x_0', which would guarantee that $\Psi(x_1x_0x_2, \tilde{u}_1\tilde{u}_0\tilde{u}_2)$ is defined. However, the Restriction Axiom for Automata requires more, namely, definedness for all prefixes of $\tilde{u}_1\tilde{u}_0\tilde{u}_2$.

One possible remedy might be to use full transitions instead of terminal ones.

Definition 6 (For both discrete or continuous time).

$$\tilde{\Psi}(x_1x_0x_2, \tilde{u}_1\tilde{u}_0\tilde{u}_2) = \tilde{x}_1\tilde{x}_0\tilde{x}_2 \ \text{ iff } \ \tilde{\Psi}_1(x_1x_0, \tilde{u}_1\tilde{u}_0) = \tilde{x}_1\tilde{x}_0 \ \& \ \tilde{\Psi}_2(x_2x_0, \tilde{u}_2\tilde{u}_0) = \tilde{x}_2\tilde{x}_0 \tag{6}$$

The way to handle terminal transitions is as follows:

Definition 7 (For both discrete and continuous time). *If the components M_i don't share registers, then*

$$\Psi(x_1x_2, \tilde{u}_1\tilde{u}_0\tilde{u}_2) = x_1'x_2' \ \text{ iff } \ \Psi_1(x_1, \tilde{u}_1\tilde{u}_0) = x_1' \ \& \ \Psi_2(x_2, \tilde{u}_2\tilde{u}_0) = x_2' \tag{7}$$

Let $act(M)$ denote the action set accepted by M.

Proposition 2 (Restorability of $act(M)$). *Assume that the M_i don't share registers.*
Let $L_i(\tilde{u}_i, \tilde{u}_0)$ be the characteristic predicate of $act(M_i)$. Then

$$L(\tilde{u}_1, \tilde{u}_0, \tilde{u}_2) = L_1(\tilde{u}_1, \tilde{u}_0) \ \& \ L_2(\tilde{u}_2, \tilde{u}_0)$$

is the characteristic predicate of $act(M)$

b) Asynchrony
This interaction combinator is considered for automata with interleaving structure (see Sec. 2.2c).
Consider first discrete time.

Definition 8 *(Asynchronous composition: $M = M_1 || M_2$). Assume M_1 with spaces $X_1 \times X_0$, $U_1 + U_0$, and M_2 with spaces $X_2 \times X_0$, $U_2 + U_0$. Then M has spaces $X_1 \times X_0 \times X_2$, $U_1 + U_0 + U_2$ and $nextstate(x_1x_0x_2, u_i) = x_1'x_0'x_2'$ holds in one of the cases:*

$$nextstate_1(x_1x_0, u_i) = x_1'x_0' \ \& \ x_2' = x_2 \quad if \quad i = 1 \tag{a}$$

$$nextstate_2(x_2x_0, u_i) = x_2'x_0' \ \& \ x_1' = x_1 \quad if \quad i = 2 \tag{b}$$

$$nextstate_1(x_1x_0, u_i) = x_1'x_0' \ \& \ nextstate_2(x_2x_0, u_i) = x_2'x_0' \quad if \quad i = 0 \tag{c}$$

Again, one should be careful about the extension from *nextstate* to terminal and full transition maps Ψ and $\tilde{\Psi}$.

The way to deal with terminal transitions, is as follows:

Definition 9 (No shared ports). *Assume M_1 with spaces $X_1 \times X_0$, U_1, and M_2 with spaces $X_2 \times X_0$, U_2. Then M has spaces $X_1 \times X_0 \times X_2$, $U_1 + U_2$, and (up to semi-group closure) the following terminal transition map: $\Psi(x_1 x_0 x_2, \tilde{u}_i) = x'_1 x'_0 x'_2$ holds in one of the cases:*

$$\Psi_1(x_1 x_0, \tilde{u}_i) = x'_1 x'_0 \ \& \ x'_2 = x_2 \quad if \quad i = 1 \tag{a}$$

$$\Psi_2(x_2 x_0, \tilde{u}_i) = x'_2 x'_0 \ \& \ x'_1 = x_1 \quad if \quad i = 2 \tag{b}$$

Definition 10 (No shared registers). *Assume M_1 with spaces X_1, $U_1 + U_0$ and M_2 with spaces X_2, $U_2 + U_0$. Then $\Psi(x_1 x_2, u_i) = x'_1 x'_2$ holds in one of the cases:*

$$\Psi_1(x_1, \tilde{u}_i) = x'_1 \ \& \ x'_2 = x_2 \quad if \quad i = 1 \tag{a}$$

$$\Psi_2(x_2, \tilde{u}_i) = x'_2 \ \& \ x'_1 = x_1 \quad if \quad i = 2 \tag{b}$$

$$\Psi_1(x_1, \tilde{u}_i) = x'_1 \ \& \ \Psi_2(x_2, \tilde{u}_i) = x'_2 \quad if \quad i = 0 \tag{c}$$

6 Nets vs. Webs

a) Basic architectures

This dichotomy reflects available communication mechanisms:

Nets: communication, if any, is via shared ports; all registers of a component are private.

Webs: communication, if any, is via shared registers; all ports of a component are private.

Note that for nets, because of the privacy of registers, it can be assumed (without loss of generality) that components have unique registers. Similarly for ports in a web. The two dichotomies induce four "architectures":

	$\times$	$\|\|$
private registers	synchronous net	asynchronous net
private ports	synchronous web	asynchronous web

Remark 2 *Terminal transition maps do not fit directly with architecture 3 (synchronous webs). In this case the formulation would require first the consideration of full transitions (see Warning in Sect. 3.1a).*

b) Petri graphs

This is a bipartite graph G (possibly directed) with k circle-nodes and m box-nodes. G is an atomic net if it consists of a single circle with its neighboring boxes; it is an atomic web if it consists of an unique box with its neighboring circles. Hence, two dual decompositions of G: into k atomic subnets or into m atomic subwebs. G is said to be a net or a web if it is equipped with the corresponding decomposition.

Figure 1 shows an atomic net with circle labeled Q and an atomic web with box labeled A.

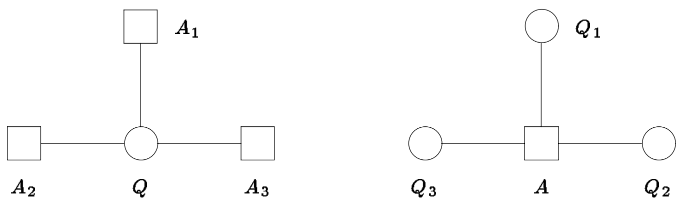

Figure 1: Nets versus Webs.

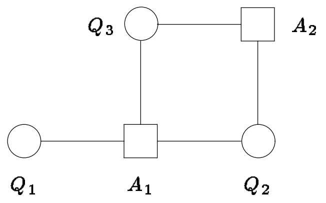

Figure 2: A Petri graph.

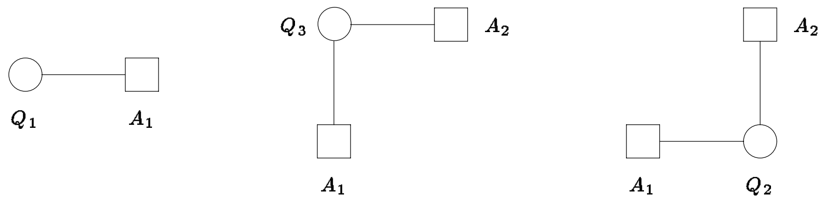

Figure 3: Decomposition into three atomic nets.

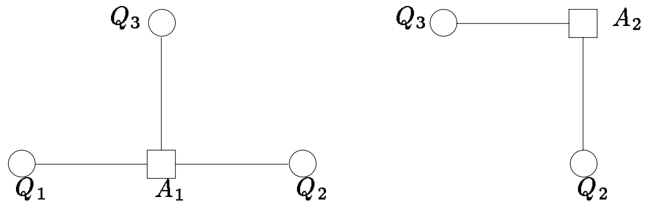

Figure 4: Decomposition into two atomic webs.

Example 3 *Consider the Petri graph shown in Figure 2. It has two dual decompositions. Figure 3 shows the decomposition into three atomic nets which "communicate" via shared boxes. Figure 4 shows the decomposition into two atomic webs which "communicate" via shared circles.*

c) Nets and webs of automata

Appropriately labeled nets or webs offer a suggestive graphical syntax for paradigms which involve sharing and/or communication.

A net of automata $N(M_1, ..., M_k)$ is an entity given by:

(i) A net N with numbered circles: $c_i : i = 1, 2, ..., k$.

(ii) A map *env* which *correctly* assigns to each c_i an automaton M_i; this means that the ports of M_i are in $1-1$ correspondence with the neighboring boxes of N. It is assumed that each M_i has a single register, which is its private register.

Semantics: if $M_1 \times ... \times M_k = M$, then say that M is decomposed as (specified by) the synchronous net $N(M_1, ..., M_k)$. Similarly, for asynchrony.

Webs of automata $W(M_1, ..., M_m)$ are handled in the dual way. In particular:

(i) $b_1, ..., b_m$ is an enumeration of the boxes in W.

(ii) Each of the automata has an unique (private!) port, and *env* correctly assigns automata to boxes.

Remark 3 *Synchronous nets are commonly used in hardware specification, where ports correspond to 'pins' of the physical component devices ([G]). The relational behavior of a device may be specified by defining a predicate $Dev(a_1, ...)$, which holds iff $a_1, ...$ are allowable values on the corresponding lines (ports). Note that the values on the lines can be modelled with infinite paths. The constraint imposed by the whole system (the relational behavior of the system) is obtained by:*

(i) conjoining the constraints (predicates) imposed by the components (compare with Proposition 2), and

(ii) existentially quantifying the variables corresponding to the internal lines. (As mentioned earlier, hiding issues are omitted in this paper.)

Recall that in the Petri Net community circles are called places, and boxes are called transitions. However, having in mind the *env* maps above, we will refer occasionally to the circles (boxes) of a Petri graph as registers (ports).

d) Mutual modeling of asynchronous nets and webs
Look at the Petri graph of Figure 2 and assume it pictures a net N over three components. See (easy!) that, dually, it pictures also an appropriate web W, such that N and W specify the same automaton. However, modeling webs as nets is much harder, and a more sophisticated approach is needed. The moral: webs are in some sense more expressive than nets. However, nets may be preferred for compositionality reasons [Ma].

e) Directed Petri-graphs
Assume that the edges in G are oriented (directed). A box b is an output-box of circle c, if there is a directed edge (channel) from c to b. Similarly, for input boxes of c. Hence, for given c we have the partition $< in(c), out(c) >$ into its input and output boxes. If b is an output-box of some $c \in G$, then it is said to be an output box of G; otherwise it is an input-box of G. Hence, we also have a partition of all boxes of G into $in(G)$ and $out(G)$. We call this partition the port-type of G, and use for it the notation $in(G) \longrightarrow out(G)$. Similarly, is defined the register type of G. In the sequel we focus on directed nets of automata, whose ports are partitioned into input and output ports, and require that *env* respects the status of these ports (i.e. input ports correspond to neighboring input boxes etc.).

Orientation of the edges may be used for additional information about the components of the net or of the web. But note that it does not affect the semantics of synchrony and asynchrony.