Chapter 3.

Stream X-machines.

This chapter will investigate further stream X-machines. Firstly, we shall identify the class of (partial) functions that this model computes. Secondly, we shall define two simple operations (i.e. sequential and parallel composition) that can be performed on stream X-machines. Finally, the minimality problem for stream X-machines will be discussed. Two types of minimality will be defined and the problem of finding the minimal machines will be addressed for some particular cases. Also, several X-machine specifications of a digital system (viz. the correlator) will be given.

3.1. (Generalised) stream functions.

In the previous chapter we explored the stream X-machine mainly as an acceptor, identifying classes of languages that can be accepted by stream X-machines with a certain type Φ. We shall now concentrate on stream X-machines with non-empty output alphabet and we shall seek to identify general features of the (partial) functions they compute, regardless of the type Φ used.

For the sake of simplicity, in what follows we shall modify slightly the definition 2.4.1 of a stream X-machine given in the previous chapter by considering that the blank belongs to the input alphabet, i.e. $\delta \in \Sigma$. Hence, $\Sigma' = \Sigma$ and the input code $\alpha: \Sigma^* \to X$ will be defined by

$$\alpha(s) = (1, m_0, s), \forall\ s \in \Sigma^*.$$

Of course, this does not change the nature of the model, nor does it affect the results in the previous chapter.

Recall that each $\varphi \in \Phi$ is a relation $\varphi: X \to X$, with $X = \Gamma^* \times M \times \Sigma^*$, defined by

$$\varphi(g, m, s) = \begin{cases} (g\ \rho(m, \text{head}(s)), \mu(m, \text{head}(s)), \text{tail}(s)), & \text{if } s \neq 1 \\ \varnothing, & \text{otherwise} \end{cases}$$

where $\mu: M \times \Sigma \leftrightarrow M$, $\rho: M \times \Sigma \leftrightarrow \Gamma$ are relations.

Then, it is clear that each relation φ is well determined by ρ and μ. For the sake of simplicity, in what follows we shall be referring to φ as a relation

$$\phi: M \times \Sigma \leftrightarrow \Gamma \times M,$$

where $\phi(m, \sigma) = (\rho(m, \sigma), \mu(m, \sigma)), \forall\ \sigma \in \Sigma, m \in M$.

Therefore $\phi(m, \sigma) = (\gamma, m')$ iff $\varphi(g, m, \sigma s) = (g\gamma, m', s)$, $\forall\ s \in \Sigma^*$, $g \in \Gamma^*$. Then, the type Φ will be referred to as a set

$$\Phi = \{\phi |\ \phi: M \times \Sigma \leftrightarrow \Gamma \times M\}.$$

If a stream X-machine is deterministic, then Φ is a set of (partial) functions and $\forall\ \phi, \phi' \in \Phi$, if $\exists\ q \in Q$, $m \in M$, $\sigma \in \Sigma$ such that

$(q, \phi) \in$ dom F, $(q, \phi') \in$ dom F, $(m, \sigma) \in$ dom ϕ and $(m, \sigma) \in$ dom ϕ',

then $\phi = \phi'$.

In other words, given a specific memory value and input there is at most one possible ϕ that can be applied from a given state. All the stream X-machines we shall be referring to in what follows will be deterministic. First, we introduce some useful notation and prove some preparatory results.

Definition 3.1.1.

Given a deterministic stream X-machine $\mathcal{M}$, we define the partial functions

$$v: Q \times M \times \Sigma \to Q,\ w: Q \times M \times \Sigma \to M,\ \lambda: Q \times M \times \Sigma \to \Gamma,$$

with

$$\text{dom } v = \text{dom } w = \text{dom } \lambda,$$

such that :

1. $(q, m, \sigma) \in$ dom v iff $\exists\ \phi \in \Phi$ such that $(q, \phi) \in$ dom F and $(m, \sigma) \in$ dom ϕ.
2. if $F(q, \phi) = q', \phi(m, \sigma) = (\gamma, m')$, then

   $v(q, m, \sigma) = q'$, $w(q, m, \sigma) = m'$ and $\lambda(q, m, \sigma) = \gamma$,

where ϕ is as defined above.

Obviously, since $\mathcal{M}$ is deterministic, such a ϕ is unique. Hence v, w and λ are well defined. $v(q, m, \sigma)$ and $w(q, m, \sigma)$ indicate the next state and the next memory value, respectively, produced by the machine when σ is received in q and m; $\lambda(q, m, \sigma)$ indicates the output symbol produced by the machine.

3. We also define

$$u: Q \times M \times \Sigma \to Q \times M,$$

by

$$u(q, m, \sigma) = (v(q, m, \sigma), w(q, m, \sigma)),\ \forall\ q \in Q,\ m \in M,\ \sigma \in \Sigma$$

(i.e. u indicates both the next state and next memory).

Then u is called the *transition function* and λ the *output function*.

Definition 3.1.2.

We extend u and λ to

$$u_e: Q \times M \times \Sigma^* \to Q \times M \text{ and } \lambda_e: Q \times M \times \Sigma^* \to \Gamma^*,$$

where u_e, and λ_e are defined recursively by:

$$u_e(q, m, 1) = (q, m), \text{ where 1 is the empty string}$$

$$u_e(q, m, s\sigma) = \begin{cases} u(u_e(q, m, s), \sigma), & \forall\ \sigma \in \Sigma,\ s \in \Sigma^* \text{ such that} \\ & u_e(q, m, s) \neq \varnothing \text{ and } u(u_e(q, m, s), \sigma) \neq \varnothing \\ \varnothing, \text{ otherwise} \end{cases}$$

$\lambda_e(q, m, 1) = 1$

$$\lambda_e(q, m, s\sigma) = \begin{cases} \lambda_e(q, m, s)\,\lambda(u_e(q, m, s), \sigma), & \forall\, \sigma \in \Sigma,\ s \in \Sigma^* \text{ such that} \\ & \lambda_e(q, m, s) \neq \varnothing \text{ and } \lambda(u_e(q, m, s), \sigma) \neq \varnothing \\ \varnothing, & \text{otherwise} \end{cases}$$

Then u_e is called the *extended transition function* and λ_e the *extended output function*. It is clear that dom u_e = dom λ_e. We also define two partial functions

$v_e: Q \times M \times \Sigma^* \to Q$ and $w_e: Q \times M \times \Sigma^* \to M$

by:

dom v_e = dom w_e = dom u_e

and

$v_e(q, m, s) = q'$ and $w_e(q, m, s) = m'$,

where $(q', m') = u_e(q, m, s)$, $\forall\ (q, m, s) \in$ dom u_e.

It is obvious that v_e and w_e are extensions of v and w respectively.

If a machine $\mathcal{M}$ is in the state q with the memory value m, then an input sequence s takes the machine to the state $q' = v_e(q, m, s)$ and the memory value $m' = w_e(q, m, s)$ while adding the sequence $\lambda_e(q, m, s)$ to the output string.

Similarly, we can define v, w, u and λ for generalised stream X-machines (i.e. in this case λ will be a partial function $\lambda: Q \times M \times \Sigma \to \Gamma^*$) and v_e, w_e, u_e and λ_e, their extensions.

Lemma 3.1.3.

Let $\mathcal{M}$ be a stream X-machine, $(q, m) \in Q \times M$ and $s \in \Sigma^*$. If $\lambda_e(q, m, s) \neq \varnothing$ then

$|\lambda_e(q, m, s)| = |s|$.

Note: |s| denotes the length of the string s.

Proof :

Follows by induction on the length of s. ⑥

Lemma 3.1.4.

Let $\mathcal{M}$ be a (generalised) stream X-machine, $(q, m) \in Q \times M$ and $s, s' \in \Sigma^*$. Then:

$\lambda_e(q, m, ss') = \lambda_e(q, m, s)\,\lambda_e(u_e(q, m, s), s')$,

$u_e(q, m, ss') = u_e(u_e(q, m, s), s')$.

Proof :

Follows by induction on the length of s'. ⑥

Observation 3.1.5.

Let q_0 be the initial state of a deterministic stream X-machine and $x_0 \in X$, $x_0 = (1, m_0, s)$, $s \in \Sigma^*$ and let $x \in X$ be the final value computed by the machine following the (unique) path (if any) emerging from q_0 having x_0 as the initial value. Then

$x = (\lambda_e(q_0, m_0, s), w_e(q_0, m_0, s), 1)$.

Hence

$Out(x) = \lambda_e(q_0, m_0, s)$.

Note: Recall that Out: $X \to \Gamma^*$ is defined by Out(g, m, s) = g.

Lemma 3.1.6.
Let $\mathcal{M}$ be a deterministic stream X-machine with T = Q (i.e. all the final states are terminal) and let $f = \alpha|\mathcal{M}|\beta\ \beta\varepsilon$ the function computed by $\mathcal{M}$. Then

$f(s) = \lambda_e(q_o, m_o, s), \forall\ s \in \Sigma^*$.

Proof:
We have $\alpha(s) = (1, m_o, s)$. From the observation above it follows that

$Out(x) = \lambda_e(q_o, m_o, s)$,

where $x \in X$ is the final value computed by the machine following the (unique) path (if any) emerging from q_o having $x_o = (1, m_o, s)$ as the initial value. If T = Q, then the output function β can be applied in any state, hence

$f(s) = \alpha|\mathcal{M}|\beta(s) = Out(x)$.

Hence

$f(s) = \lambda_e(q_o, m_o, s)$. ⑥

We can now give a characterisation of the functions that (generalised) stream X-machines compute. First, we need the following definitions.

Definition 3.1.7.
Let $f: \Sigma^* \to \Gamma^*$ be a partial function. Then f is called *segment preserving* if:

$\forall\ s, t \in \Sigma^*$, if $s, st \in$ dom f then $\exists\ u \in \Gamma^*$ such that $f(st) = f(s)u$.

Definition 3.1.8.
Let $f: \Sigma^* \to \Gamma^*$ be a partial function. If

$|f(s)| = |s|, \forall\ s \in$ dom f

then f is called *length preserving*.

Definition 3.1.9.
Let $f: \Sigma^* \to \Gamma^*$ be a partial function. Then f is called a *partial stream function* if

i) f is both segment preserving and length preserving and

ii) $\forall\ s, t \in \Sigma^*$, if $st \in$ dom f, then $s \in$ dom f.

Proposition 3.1.10.
Let $\mathcal{M}$ be a deterministic stream X-machine and let $f: \Sigma^* \to \Gamma^*$ be the partial function computed by it. Then

1. f is length-preserving and segment preserving.

2. If all the states of $\mathcal{M}$ are terminal (i.e. T = Q), then f is a partial stream function.

Proof:

1. If $f(s) \neq \varnothing$ and $f(st) \neq \varnothing$, then

$f(s) = \lambda_e(q_o, m_o, s)$

and

$f(st) = \lambda_e(q_o, m_o, st)$

and the result follows from lemmas 3.1.3 and 3.1.4.

2. It follows from $f(s) = \lambda_e(q_0, m_0, s)$, $\forall s \in \Sigma^*$ using lemma 3.1.4. ⑥

Similarly, we have the following results for generalised stream X-machines.

Definition 3.1.11.
Let $f: \Sigma^* \rightarrow \Gamma^*$ be a partial function. Then f is called a *partial generalised stream function* if

i) it is segment preserving and

ii) $\forall s, t \in \Sigma^*$, if $st \in$ dom f, then $s \in$ dom f.

Proposition 3.1.12.
Let $\mathcal{M}$ be a deterministic generalised stream X-machine and let $f: \Sigma^* \rightarrow \Gamma^*$ be the partial function computed by it. Then

1. f is segment-preserving.

2. If all the states of $\mathcal{M}$ are terminal (i.e. T = Q), then f is a generalised stream function.

Proof:
Follows similarly to proposition 3.1.10. ⑥

We have now a characterisation of the partial functions computed by (generalised) stream X-machine. Of course whether a particular function f that satisfies the conditions from proposition 3.1.10 (or 3.1.12) can be computed by a (generalised) stream X-machine with a certain type Φ depends on the nature of Φ (i.e. if f is non-Turing computable Φ has to be non-Turing computable, if f is computable Φ can be chosen to be computable, if f is fully computable, Φ can be chosen to be fully-computable). This problem was discussed in detail in the previous chapter. The conditions from proposition 3.1.10 and proposition 3.1.12. are satisfied by all functions computed by (generalised) stream X-machines regardless of the type Φ used.

If the stream X-machine model is to be used in testing real systems, it is natural that we would like to have as much information about the outputs produced as possible. For this reason we shall, from now on, be referring to stream X-machines with all the states terminal (i.e. T = Q). This means that the output produced by the machine can be viewed in any of its states (i.e. β can be applied in any state q of the machine), even though the machine is allowed to terminate its computation only in a terminal state. For example, if $\mathcal{M}$ is the specification of a program, the condition T = Q is achieved if the program displays the intermediary outputs, as well as the final ones. Of course, the intermediary outputs that need not be displayed can be removed after the program is tested. In what follows, a stream X-machine with all the states terminal will be referred to as a tuple

$$\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$$

(since T = Q the set of terminal states will be omitted).

3.2. Stream X-machine specifications of a correlator.

We interrupt our theoretical discussion and present some examples. In what follows we use stream X-machines to specify a digital system, viz. a correlator. The process of specifying a digital correlator is investigated by McEvoy & Tucker, [46], and several functional specifications of the system have been produced. In what follows we construct stream X-machine specifications for these models and contrast the functional and machine specifications.

The heart of the correlator is a function that compares pairs of data-elements from two n-element vectors and returns the number of elements that are the same.

Let A be any non-empty set. Consider the function corr: $A^n \times A^n \rightarrow \{0,...,n\}$ defined by

$$corr(a, b) = card(\{i | a_i = b_i\}),$$

where $a = (a_1, ..., a_n)$, $b = (b_1, ..., b_n)$. That is, corr(a, b) is the number of elements (cardinality) of the set of i's such that a_i and b_i are equal.

3.2.1. First model.

We can now define a simple stream correlator. Let w: $T \rightarrow A^n$ be a stream of vectors of length n, where T is the set of natural numbers. The idea is that at every clock cycle $t \in T$, a new data-word $w(t)$ is generated and delivered to the processor corr.

The system should satisfy the following requirements:

1. The system computes the correlation of two n-elements in a constant number of cycles (k).
2. The system reads a data vector at each tick of the clock.
3. The reference word $r = (r_1, ...r_n) \in A^n$ is fixed.
4. The output period of the system is 1 clock cycle.

We are now able to produce a specification of the system as a function:

$$corr_1: (T \rightarrow A^n) \rightarrow (T \rightarrow (\{0, ...,n\} \cup \{u\})),$$

$$corr_1(w)(t) = \begin{cases} u, & \text{if } t < k \\ corr(w(t-k), r), & \text{otherwise} \end{cases}$$

where k is the computation time and r is the reference word.

Observation 3.2.1.1.
Let X and Y be sets and

$$f: (T \rightarrow X) \rightarrow (T \rightarrow Y)$$

be a (total) function. If $\forall\ w,\ w' \in (T \rightarrow X)$ and $t \in T$,

$$w(0) = w'(0), ..., w(t-1) = w'(t-1), w(t) = w'(t) \Rightarrow f(w)(t) = f(w')(t)$$

(i.e. $\forall\ w \in (T \to X)$, f(w)(t) does not depend on w(t+1), w(t+2), ...), then

ℓ: $X^* \to Y^*$

defined by

$\ell(v) = f(w)(0) \ldots f(w)(t), \forall\ v = v_0 \ldots v_t$

with $v_i \in X$ and $w \in (T \to X)$ such that $w(i) = v_i$, i = 0, ...t, is a stream function. It is clear that ℓ can be uniquely determined from f and vice versa. Then we shall call ℓ the stream function *determined by* f.

Then we denote by

$Corr_1$: $(A^n)^* \to (\{0, ...,n\} \cup \{u\})^*$

the stream function determined by $corr_1$. A stream X-machine $\mathcal{M}_1$ which computes $Corr_1$ is as follows.

1. $\Sigma = A^n$
2. $\Gamma = \{0, 1, ...n\} \cup \{u\}$
3. The set of states is $Q = \{q_0, ..., q_k\}$; q_0 is the initial state.
4. $M = (A^n)^k$
5. $m_0 = (b_{1,0}, ..., b_{k,0})$, where $b_{i,0} \in A^n$, $i \in \{1, ...k\}$, are chosen at random.
6. $\Phi = \{\phi_1, \phi_2\}$, where ϕ_1, ϕ_2: $M \times \Sigma \to \Gamma \times M$ are functions defined by

$\phi_1((b_1, ..., b_{k-1}, b_k), w) = (u, (b_2, ..., b_k, w))$,

$\phi_2((b_1, ..., b_{k-1}, b_k), w) = (corr(b_1, r), (b_2, ..., b_k, w))$,

$\forall\ b_1, ..., b_k \in A^n\ \ \forall\ w \in A^n$,

where $r = (r_1, ...r_n)$ is the reference word.

7. F is represented in figure 3.1.

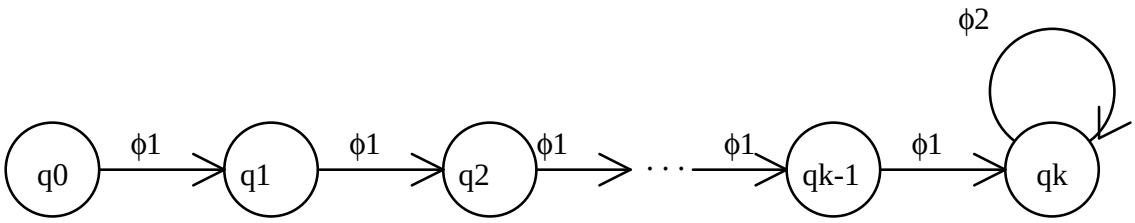


Figure 3.1.

The state set replaces the clock T. The system is considered to be in the state q_t, $t \in \{0, 1, ...,k-1\}$ if $t < k$ and in q_k if $t \geq k$. The memory holds the last k elements of the input vector stream which have been read.

3.2.2. Second model.

We modify the first model in the sense that, instead of reading the input vectors in parallel, their components are read serially, one per clock cycle. The new specification will be:

$corr_2$: $(T \to A) \to (T \to (\{0, ...,n\} \cup \{u\}))$,

$$corr_2(x)(t) = \begin{cases} u, & \text{if } t < k + n-1 \\ corr((x(t-k-n+1), ..., x(t-k)), r), & \text{otherwise} \end{cases}$$

The idea is that we are correlating on a part of the stream *x* of length n. It takes n steps before we have acquired enough elements from the stream *x* to form a word of length n and we can begin correlating. So the delay before results emerge is made up of the initialisation time n-1 and computation time of the function corr. After this time, each result emerging is the result of a correlation that started k steps earlier.

A stream X-machine $\mathcal{M}_2$ which computes $Corr_2$, the stream function determined by $corr_2$ will be the following:

1. $\Sigma = A$
2. $\Gamma = \{0, 1, ...n\} \cup \{u\}$
3. The set of states is $Q = \{q_0, ..., q_{k+n-1}\}$ and q_0 is the initial state.
4. $M = A^{n+k-1}$
5. $m_0 = (a_{1,0}, ..., a_{k+n-1,0})$, where $a_{i,0} \in A$, $i \in \{1, ... k+n-1\}$, are chosen at random.
6. $\Phi = \{\phi_1, \phi_2\}$, where $\phi_1, \phi_2: M \times \Sigma \to \Gamma \times M$ are functions defined by

$$\phi_1((a_1, ..., a_{k+n-1}), x) = (u, (a_2, ...,a_{k+n-1}, x))$$
$$\phi_2((a_1, ..., a_{k+n-1}), x) = (corr((a_1, ...,a_n), r), (a_2, ...,a_{k+n-1}, x))$$

$\forall\ a_1, ..., a_{k+n-1} \in A\ \ \forall\ x \in A$,
where $r = (r_1, ...r_n)$ is the reference word.

7. F is represented in figure 3.2.

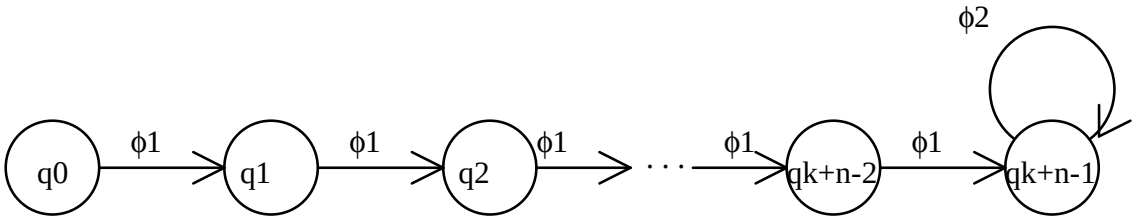


Figure 3.2.

3.2.3. Third model:

We will now remove the simplifying assumption of a fixed reference word. We define a correlator that allows the reference word r to be programmed by means of a stream $y \in (T \to A)$ and a control stream $s \in (T \to B)$ (i.e. B is the set of Booleans). The data items from stream *y* will be accepted or rejected according to

the value of the corresponding element of *s*. Only those elements of the *y* stream which arrive at the same time as a true element on stream *s* will be accepted. The elements of the *y* stream which are accepted will go to make up an n-element word. This leads to a much more complicated specification because we cannot assume that after n steps enough elements from the stream *y* will have been read for correlation to start.

Let $d_1, ..., d_n: T \times (T \to B) \to T$ be partial functions defined such that $y(d_i(t, s))$ is the i-th element of the reference word at the time t and let $d: (T \to B) \to T$ be the time needed for the reference word to be completely loaded. The specification of the new system in terms of functions is the following:

$$corr_3: ((T \to A)^2 \times (T \to B)) \to (T \to (\{0, ...,n\} \cup \{u\}),$$

$$corr_3(x, y, s)(t) = \begin{cases} u, \text{ if } d(s) = \varnothing \text{ or } t < k + d(s) \\ corr((x(t-k-n+1),.., x(t-k)), (y(d_1(t,s)), ..,y(d_n(t,s))), \text{ otherwise} \end{cases}$$

where

$d_i: T \times (T \to B) \to T, 1 \le i \le n,$

are defined by:

$$d_n(0, s) = \begin{cases} 0, & \text{if } s(0) = true \\ \varnothing, & \text{if } s(0) = false \end{cases}$$

$$d_n(t+1, s) = \begin{cases} t + 1, & \text{if } s(t+1) = true \\ d_n(t, s), & \text{if } s(t+1) = false \end{cases}$$

for $1 \le i < n$

$$d_i(0, s) = \varnothing$$

$$d_i(t+1, s) = \begin{cases} d_{i+1}(t, s), & \text{if } s(t+1) = true \\ d_i(t, s), & \text{if } s(t+1) = false \end{cases}$$

$$d: (T \to B) \to T$$

is defined by

$$d(s) = \begin{cases} \min\{t |\ d_1(s, t) \ne \varnothing\}, \text{ if } \exists\ t \in T \text{ such that } d_1(s, t) \ne \varnothing \\ \varnothing, \text{ otherwise} \end{cases}$$

and k is the computation time for corr.

Similarly to observation 3.2.1.1, we can define

$Corr_3$: $(A^2 \times B)^* \rightarrow (\{0, ..., n\} \cup \{u\})^*$,
the stream function determined by $corr_3$. A stream X-machine that computes $Corr_3$ is the following.

1. $\Sigma = A^2 \times B$.
2. $\Gamma = \{0, 1, ...n\} \cup \{u\}$.
3. The set of states is $Q = \{q_0, ..., q_{n+k-1}\}$; q_0 is the initial state.
4. $M = A^{n+k-1} \times A^n$.
5. $m_0 = ((a_{1,0}, ..., a_{k+n-1,0}), r_0, 0)$ with $r_0 = (r_{1,0}, ...r_{n,0})$,
where $r_{i,0} \in A$, $i \in \{1, ...n\}$ and $a_{j,0} \in A$, $j \in \{1, ...n+k-1\}$, are chosen at random.
6. $\Phi = \{\phi_1, \phi_2, \phi_3, \phi_4\}$, where $\phi_1, \phi_2, \phi_3, \phi_4$: $M \times \Sigma \rightarrow \Gamma \times M$ are partial functions defined by:

$dom\ \phi_1 = dom\ \phi_3 = (A^2 \times \{false\}) \times M$,
$dom\ \phi_2 = dom\ \phi_4 = (A^2 \times \{true\}) \times M$,

$\phi_1(((a_1, ..., a_{k+n-1}), (r_1, ...r_n)), (x, y, false)) =$
$(u, ((a_2, ...a_{k+n-1}, x), (r_1, ...r_n)))$

$\phi_2(((a_1, ..., a_{k+n-1}), (r_1, ...r_n)), (x, y, true)) =$
$(u, ((a_2, ...a_{k+n-1}, x), (r_2, ...r_n, y)))$

$\phi_3(((a_1, ..., a_{k+n-1}), (r_1, ...r_n)), (x, y, false)) =$
$(corr((a_1, ..., a_n), (r_1, ..., r_n)), ((a_2, ...a_{k+n-1}, x), (r_1, ...r_n)))$

$\phi_4(((a_1, ..., a_{k+n-1}), (r_1, ...r_n)), (x, y, true)) =$
$(corr((a_1, ..., a_n), (r_1, ..., r_n)), ((a_2, ...a_{k+n-1}, x), (r_2, ...r_n, y)))$

7. F is represented in figure 3.3.

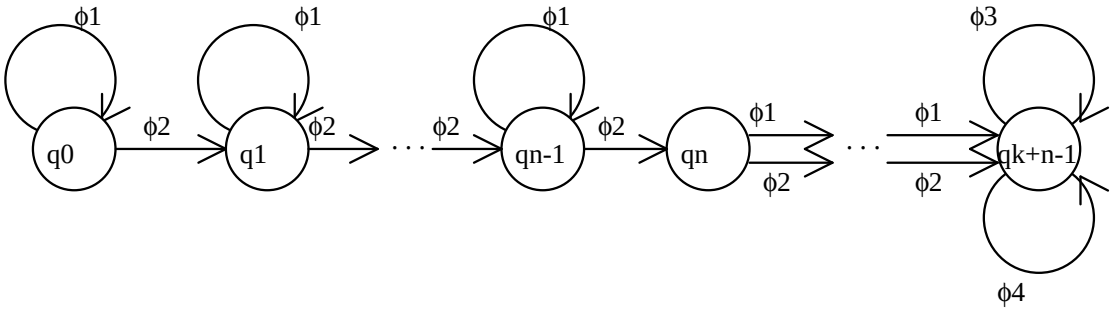


Figure 3.3.

Conclusions.

Some conclusions can be drawn from this example. We have specified in terms of functions and stream X-machines three versions of a fairly simple system, i.e. a

digital correlator. While the functional specification appeared to be sufficient for the simpler models (the first and second), they became too complicated and difficult to handle when they dealt with a more complex model (i.e. the third model requires several stream functions and their integration may be difficult to understand). In this last case, a stream X-machine specification appears to be much more intuitive and easy to understand.

3.3. Parallel and sequential composition of stream X-machines

We shall now introduce some basic operations that can be performed on stream X-machines (i.e. parallel and sequential composition), describing the result of these operations in terms of stream functions.

3.3.1. Parallel composition.

Definition 3.3.1.1.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Sigma', \Gamma', Q', M', \Phi', F', q_0', m_0')$ be two deterministic stream X-machines. Then a deterministic stream X-machine

$$\mathcal{M}'' = (\Sigma'', \Gamma'', Q'', M'', \Phi'', F'', q_0'', m_0'')$$

defined by:

1. $\Sigma'' = \Sigma \times \Sigma'$
2. $\Gamma'' = \Gamma \times \Gamma'$
3. $Q'' = Q \times Q'$
4. $M'' = M \times M'$
5. $q_0'' = (q_0, q_0')$
6. $m_0'' = (m_0, m_0')$
7. $\Phi'' = \{\phi'' = \psi(\phi, \phi') | \ \phi''$ is not the empty function$\}$,

where $\forall \phi \in \Phi, \phi' \in \Phi', \phi'' = \psi(\phi, \phi')$ is a partial function $\phi'': M'' \times \Sigma'' \to \Gamma'' \times M''$ defined by:

$$\phi''((m, m'),(\sigma', \sigma)) = \begin{cases} ((\gamma, \gamma'), (m_1, m_1')), \text{ if } \phi(m, \sigma) \neq \varnothing \text{ and } \phi'(m', \sigma') \neq \varnothing, \\ \quad \text{where } (\gamma, m_1) = \phi(m, \sigma) \text{ and } (\gamma', m_1') = \phi'(m', \sigma') \\ \varnothing, \text{ otherwise} \end{cases}$$

8. $F'': Q'' \times \Phi'' \to Q''$ is defined by

$$F''((q, q'),\phi'') = \begin{cases} (F(q, \phi), F'(q', \phi')), \text{ if } \exists \phi \in \Phi, \phi' \in \Phi' \text{ such that} \\ \quad \psi(\phi, \phi') = \phi'', F(q, \phi) \neq \varnothing \text{ and } F(q', \phi') \neq \varnothing \\ \varnothing, \text{ otherwise} \end{cases}$$

is called the *parallel composition* of $\mathcal{M}$ and $\mathcal{M}'$ (written $\mathcal{M}'' = \mathcal{M} \times \mathcal{M}'$).

Note: Since $\mathcal{M}$ and $\mathcal{M}'$ are deterministic and from the definition of $\mathcal{M}'$ it follows that if $\psi(\phi_1, \phi_1') = \psi(\phi_2, \phi_2') \in \Phi''$ and

$F(q, \phi_1) \neq \emptyset$, $F(q', \phi_1') \neq \emptyset$, $F(q, \phi_2) \neq \emptyset$, $F(q', \phi_2') \neq \emptyset$,
then $\phi_1 = \phi_2$ and $\phi_1' = \phi_2'$. Hence F" is well defined. Also, since $\mathcal{M}$ and $\mathcal{M}'$ are deterministic, $\mathcal{M}$" is deterministic.
The following proposition shows the relationship between the functions computed by two machines and the function computed by their parallel composition.

Proposition 3.3.1.2.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_o, m_o)$ and $\mathcal{M}' = (\Sigma', \Gamma', Q', M', \Phi', F', q_o', m_o')$ be two deterministic stream X-machines that compute $f: \Sigma^* \to \Gamma^*$ and $g: \Sigma'^* \to \Gamma'^*$ respectively. Then $\mathcal{M} \times \mathcal{M}'$ computes h, where $h: (\Sigma \times \Sigma')^* \to (\Gamma \times \Gamma')^*$ is defined by:

$$h(s, s') = \begin{cases} (f(s), g(s')) \text{ if } f(s) \neq \emptyset \text{ and } g(s') \neq \emptyset \\ \emptyset, \text{ otherwise} \end{cases}$$

$\forall\, s \in \Sigma,\ s'^* \in \Sigma'^*$ with $|s| = |s'|$.
Hence h is still a stream function.

Note: If $s = \sigma_1 \ldots \sigma_k$, $s' = \sigma_1' \ldots \sigma_k'$, with $\sigma_1, \ldots, \sigma_k \in \Sigma$, $\sigma_1', \ldots, \sigma_k' \in \Sigma'$, then $(s, s') = (\sigma_1, \sigma_1') \ldots (\sigma_k, \sigma_k')$.

Proof:
Follows by induction on $t = (s, s')$. ©

Example 3.3.1.3.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_o, m_o)$ and $\mathcal{M}' = (\Sigma', \Gamma', Q', M', \Phi', F', q_o', m_o')$, where $\Sigma = \{a, b\}$, $\Sigma' = \{c, d\}$, $\Gamma = \{x, y\}$, $\Gamma' = \{u, v\}$, $Q = \{q_o, q_1\}$, $Q' = \{q_o', q_1'\}$, $M = M' = \{0, 1\}$, $m_o = m_o' = 0$, $\Phi = \{\phi_1, \phi_2\}$, $\Phi' = \{\phi_1', \phi_2'\}$ and F and F' represented in figures 3.4 and 3.5 respectively.

$\phi_1, \phi_2: M \times \Sigma \to \Gamma \times M$, $\phi_1', \phi_2': M' \times \Sigma' \to \Gamma' \times M'$ are partial functions defined by:

dom $\phi_1 = \{(0, a)\}$; $\phi_1(0, a) = (x, 0)$;
dom $\phi_2 = \{(0, b)\}$; $\phi_2(0, b) = (x, 1)$;
dom $\phi_1' = \{(0, c)\}$; $\phi_1'(0, c) = (u, 0)$;
dom $\phi_2' = \{(0, d\}$; $\phi_2'(0, d) = (v, 1)$.

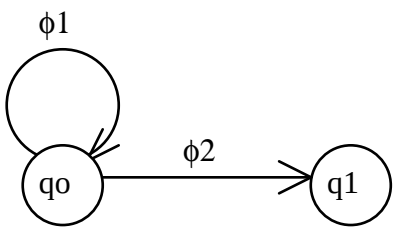


Figure 3.4.

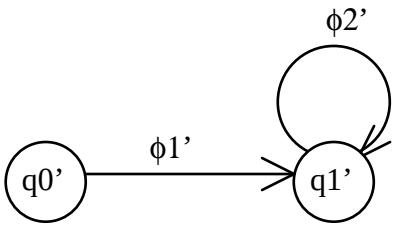


Figure 3.5.

$\mathcal{M}$ and $\mathcal{M}'$ will compute f and g respectively, where f: $\Sigma^* \rightarrow \Gamma^*$ and g: $\Sigma'^* \rightarrow \Gamma'^*$ are defined by:

dom f = $\{a^n | \ n \geq 0\} \cup \{a^n b | \ n \geq 0\}$;
$f(a^n) = x^n$;
$f(a^n b) = x^{n+1}$ and

dom g = $\{1\} \cup \{cd^n | \ n \geq 0\}$;
$g(1) = 1$;
$g(cd^n) = uv^n$.

Then $\mathcal{M}''$ = ($\Sigma \times \Sigma'$, $\Gamma \times \Gamma'$, $Q \times Q'$, $M \times M'$, Φ'', F'', (q_o, q_o'), (m_o, m_o')) is the parallel composition of $\mathcal{M}$ and $\mathcal{M}'$, where $\Phi'' = \{\phi_{11}'', \ \phi_{12}'', \phi_{21}'', \phi_{22}''\}$, and F'' is represented in figure 3.6.
The partial functions ϕ_{11}'', ϕ_{12}'', ϕ_{21}'' and ϕ_{22}'' are defined by:

dom $\phi_{11}'' = \{((0, 0), (a, c))\}$; $\phi_{11}''((0, 0), (a, c)) = ((x, u), (0, 0))$;
dom $\phi_{12}'' = \{((0, 0), (a, d))\}$; $\phi_{12}''((0, 0), (a, d)) = ((x, v), (0, 1))$;
dom $\phi_{21}'' = \{((0, 0), (b, c))\}$; $\phi_{21}''((0, 0), (b, c)) = ((x, u), (1, 0))$;
dom $\phi_{22}'' = \{((0, 0), (b, d))\}$; $\phi_{22}''((0, 0), (b, d)) = ((x, v), (1, 1))$.

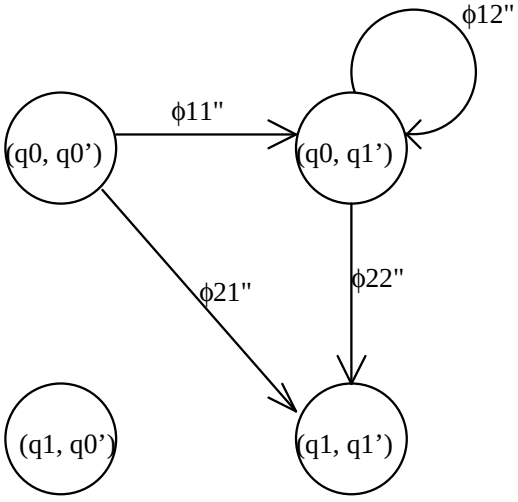


Figure 3.6.

Then $\mathcal{M}$" computes h: $(\Sigma \times \Sigma')^* \rightarrow (\Gamma \times \Gamma')^*$ defined by

dom h = $\{1\} \cup \{(a, c)(a, d)^n | \ n \geq 0\} \cup \{(b, c)\} \cup \{(a, c)(a, d)^n(b, d) | \ n \geq 0\}$;
h(1) = 1;
h((a, c)(a, d)^n) = (x, u)(x, v)^n;
h(b, c) = (x, u);
h((a, c)(a, d)^n(b, d) = (x, u)(x, v)$^{n+1}$.

3.3.2. Sequential composition.

Definition 3.3.2.1.
Let $\mathcal{M} = (\Sigma, \Omega, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Omega, \Gamma, Q', M', \Phi', F', q_0', m_0')$ two deterministic stream X-machines. Then a deterministic stream X-machine

$\mathcal{M}$" = (Σ, Γ, Q", M", Φ", F", q_0", m_0")

defined by:

1. Q" = Q × Q'
2. M" = M × M'
3. q_0" = (q_0, q_0')
4. m_0" = (m_0, m_0')
5. Φ" = {ϕ" = $\psi(\phi, \phi')$|ϕ" is not the empty function},

where $\forall\ \phi \in \Phi, \phi' \in \Phi', \phi'' = \psi(\phi, \phi')$ is a partial function ϕ": M" × $\Sigma \rightarrow \Gamma \times$ M" defined by

$$\phi''((m, m'),\sigma) = \begin{cases} (\gamma, (m_1, m_1')), \text{ if } \phi(m, \sigma) \neq \varnothing \text{ and } \phi'(m',\omega) \neq \varnothing, \\ \quad \text{where } (\omega, m_1) = \phi(m, \sigma) \text{ and } (\gamma, m_1') = \phi'(m',\omega) \\ \varnothing, \text{ otherwise} \end{cases}$$

6. F": Q" × Φ" → Q" is defined by

$$F''((q, q'),\phi'') = \begin{cases} (F(q, \phi), F'(q',\phi')), \text{ if} \exists\ \phi \in \Phi, \phi' \in \Phi' \text{ such that} \\ \quad \psi(\phi, \phi') = \phi'', F(q, \phi) \neq \varnothing \text{ and } F(q',\phi') \neq \varnothing \\ \varnothing, \text{ otherwise} \end{cases}$$

is called the *sequential composition* of $\mathcal{M}$ and $\mathcal{M}'$ (written $\mathcal{M}'' = \mathcal{M}\mathcal{M}'$).

Note: Similar to parallel composition, since $\mathcal{M}$ and $\mathcal{M}'$ are deterministic, F" is well defined and $\mathcal{M}$" is deterministic.

Proposition 3.3.2.2.
Let $\mathcal{M} = (\Sigma, \Omega, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Omega, \Gamma, Q', M', \Phi', F', q_0', m_0')$ two deterministic stream X-machines that compute f: $\Sigma^* \rightarrow \Omega^*$ and g: $\Omega^* \rightarrow \Gamma^*$ respectively. Then $\mathcal{M}\mathcal{M}'$ computes h: $\Sigma^* \rightarrow \Gamma^*$, h = fg. Hence h = fg is still a stream function.

Proof:
Follows by induction on s. ⑥

Example 3.3.2.3.
Let $\mathcal{M} = (\Sigma, \Omega, Q, M, \Phi, F, q_o, m_o)$ and $\mathcal{M}' = (\Omega, \Gamma, Q', M', \Phi', F', q_o', m_o')$, where $\Sigma = \{a, b\}$, $\Omega = \{c, d\}$, $\Gamma = \{x, y\}$, $Q = \{q_o, q_1\}$, $Q' = \{q_o', q_1'\}$, $M = M' = \{0, 1\}$, $m_o = m_o' = 0$, $\Phi = \{\phi_1, \phi_2\}$, $\Phi' = \{\phi_1', \phi_2'\}$ and F and F' represented in figures 3.7 and 3.8 respectively.

ϕ_1, ϕ_2: $M \times \Sigma \rightarrow \Omega \times M$, ϕ_1', ϕ_2': $M' \times \Omega \rightarrow \Gamma \times M'$ are partial functions defined by:

dom $\phi_1 = \{(0, a)\}$; $\phi_1(0, a) = (c, 0)$;
dom $\phi_2 = \{(0, b)\}$; $\phi_2(0, b) = (c, 1)$;
dom $\phi_1' = \{(0, c)\}$; $\phi_1'(0, c) = (x, 0)$;
dom $\phi_2' = \{(0, d\}$; $\phi_2'(0, d) = (y, 1)$.

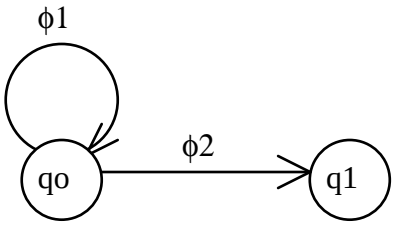


Figure 3.7.

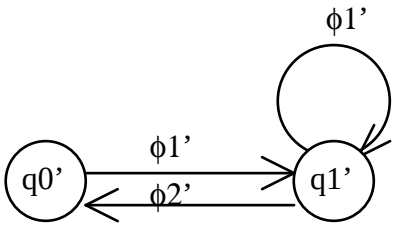


Figure 3.8.

$\mathcal{M}$ and $\mathcal{M}'$ compute f and g respectively, where f: $\Sigma^* \rightarrow \Omega^*$ and g: $\Omega^* \rightarrow \Gamma^*$ are defined by:

dom f = $\{a^n | n \geq 0\} \cup \{a^n b | n \geq 0\}$;
$f(a^n) = c^n$;
$f(a^n b) = c^{n+1}$ and

dom g = $\{1\} \cup \{c^{n+1} | n \geq 0\} \cup \{c^{n+1} d | n \geq 0\}$;
$g(1) = 1$;
$g(c^{n+1}) = x^{n+1}$;
$g(c^{n+1} d) = x^{n+1} y$.

Then, $\mathcal{M}$" = (Σ, Γ, Q $\times$ Q', M $\times$ M', Φ", F", (q_o, q_o'), (m_o, m_o')) is the sequential composition of $\mathcal{M}$ and $\mathcal{M}$', where Φ" = {ϕ_{11}", ϕ_{21}"} and F" is represented in figure 3.9.

ϕ_{11}" and ϕ_{21}" are defined by:

dom ϕ_{11}" = {((0, 0), a)}; ϕ_{11}"((0, 0), a) = (x, (0, 0));
dom ϕ_{21}" = {((0, 0), b)}; ϕ_{21}"((0, 0), b) = (x, (1, 0);

Note: ϕ_{11}" = $\psi(\phi_1, \phi_1')$, ϕ_{21}" = $\psi(\phi_2, \phi_1')$; $\psi(\phi_1, \phi_2')$ and $\psi(\phi_2, \phi_2')$ are the empty function.

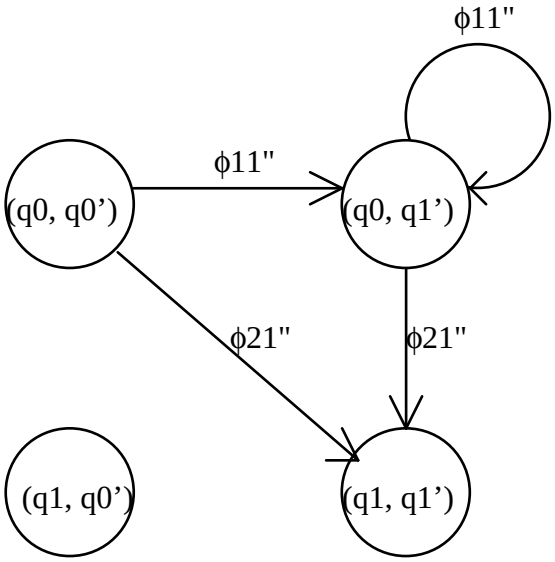


Figure 3.9.

It is clear that $\mathcal{M}$" computes h: $\Sigma^* \rightarrow \Gamma^*$ defined by

dom h = $\{a^n | n \geq 0\} \cup \{a^n b | n \geq 0\}$;
$h(a^n) = x^n$;
$h(a^n b) = x^{n+1}$.

3.4. Minimal stream X-machines and minimal coverings of stream X-machines.

In this section we shall discuss two types of stream X-machine minimality. First, we present some state machine theory that we shall be using later on.

3.4.1. Some state machine theory.

In this section we refer to (finite) state machines with all the states terminal. Unlike section 2.1.1, we shall also consider the case when the output alphabet is not empty.
We shall denote a deterministic state machine (or automaton) by a tuple

$\mathcal{A} = (\Sigma, \Gamma, Q, F, G, q_0)$,

where Σ and Γ are the finite input and output alphabets respectively, Q is the state set, q_0 the initial state and

$F: Q \times \Sigma \to Q$ and $G: Q \times \Sigma \to \Gamma$

are partial functions with

dom F = dom G.

F is called the *next state function* and G the *output function* of $\mathcal{A}$. We also define

$F_e: Q \times \Sigma^* \to Q$ and $G_e: Q \times \Sigma^* \to \Gamma^*$

by:

$$F_e(q, 1) = q,$$

$$F_e(q, s\sigma) = \begin{cases} F(F_e(q, s), \sigma), & \forall\, \sigma \in \Sigma, s \in \Sigma^* \text{ such that} \\ & F_e(q, s) \neq \varnothing \text{ and } F(F_e(q, s), \sigma) \neq \varnothing \text{ defined} \\ \varnothing, & \text{otherwise} \end{cases}$$

$$G_e(q,1) = 1$$

$$G_e(q, s\sigma) = \begin{cases} G_e(q, s)\, G(F_e(q, s), \sigma) & \forall\, \sigma \in \Sigma, s \in \Sigma^* \text{ such that} \\ & G_e(q, s) \neq \varnothing \text{ and } G(F_e(q, s), \sigma) \neq \varnothing \\ \varnothing, & \text{otherwise} \end{cases}$$

F_e and G_e are called the *extended next state function* and *extended output function*. Obviously dom F_e = dom G_e. Also, by induction it follows that

$F_e(q, ss') = F_e(F_e(q, s), s')$ and $G_e(q, ss') = G_e(q_0, s)\, G_e(F_e(q_0, s), s')$, $\forall\, q \in Q, s, s' \in \Sigma^*$.

Also, for any $q \in Q$, we denote by

$F_e.q: \Sigma^* \to Q$ and $G_e.q : \Sigma^* \to \Gamma^*$

the (partial) function defined by

$F_e.q\,(s) = F_e(q, s)$, $\forall\, s \in \Sigma^*$ and

$G_e.q\,(s) = G_e(q, s)$, $\forall\, s \in \Sigma^*$.

The (partial) *function computed* by $\mathcal{A}$, $f: \Sigma^* \to \Gamma^*$ is defined by

$f(s) = G_e(q_0, s)$, $\forall\, s \in \Sigma^*$.

It is obvious that $\forall\, q \in Q$, $G_e.q$ is a partial stream function. Hence any state machine computes a partial stream function. If the output set is empty (i.e. $\Gamma = \varnothing$), then an automaton will be denoted by a tuple

$\mathcal{A} = (\Sigma, Q, F, q_0)$.

In this case L = dom f will be the *language accepted* by the machine.

The majority of the concepts and results we shall be presenting are valid for a (possibly infinite) state machine. When the results depend on the state set being

finite, then the machine will be called a finite state machine. The concepts and results in this section are largely from Eilenberg, [12].

3.4.1.1. Morphisms.

A morphism is a particular type of state mapping between two state machines.

Definition 3.4.1.1.1.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, F, G, q_0)$ and $\mathcal{M}' = (\Sigma, \Gamma, Q', F', G', q_0')$ be two deterministic state machines having the same input and output alphabets. Then $g: \mathcal{M} \to \mathcal{M}'$ is called a *morphism* if g is a function $g: Q \to Q'$ such that:

$g(F(q, \sigma)) \subseteq F'(g(q),\sigma), \forall q \in Q$ and $\sigma \in \Sigma$,
$G(q, \sigma) \subseteq G'(g(q),\sigma), \forall q \in Q$ and $\sigma \in \Sigma$ and
$g(q_0) = q_0'$.

Note: $g(F(q, \sigma)) \subseteq F'(g(q), \sigma)$ means that either $g(F(q, \sigma)) = \varnothing$ or $g(F(q, \sigma)) \neq \varnothing$ and $g(F(q, \sigma)) = F'(g(q),\sigma)$.

Definition 3.4.1.1.2.
A morphism $g: \mathcal{M} \to \mathcal{M}'$ is called *proper* if:

$g(F(q, \sigma)) = F'(g(q),\sigma), \forall q \in Q$ and $\sigma \in \Sigma$,
$G(q, \sigma) = G'(g(q),\sigma), \forall q \in Q$ and $\sigma \in \Sigma$.

Observation 3.4.1.1.3.
If the output set is empty (i.e. $\Gamma = \varnothing$) then a morphism will be a function $g: Q \to Q'$ that satisfies

$g(F(q, \sigma)) \subseteq F'(g(q),\sigma)$ and
$g(q_0) = q_0'$.

A proper morphism satisfies

$g(F(q, \sigma)) = F'(g(q),\sigma)$.

Definition 3.4.1.1.4.
A morphism g is called an *isomorphism* if it is proper and bijective.

In what follows some properties of morphisms will be stated. First, we give the following definition.

Definition 3.4.1.1.5.
Let f, g be two partial functions $f, g: \Sigma^* \to \Gamma^*$. Then we say that f is included in g (written $f \subseteq g$) if:

dom $f \subseteq$ dom g and
$f(s) = g(s), \forall s \in$ dom f.

Lemma 3.4.1.1.6.
If $g: \mathcal{M} \to \mathcal{M}'$ is a morphism then $\forall q \in Q$, $G_e.q \subseteq G_e'.g(q)$. If g is proper, then $G_e.q = G_e'.g(q)$.

Proof:
Follows by induction on $s \in \Sigma^*$. ⑥

Proposition 3.4.1.1.7.
Let $\mathcal{A}$, $\mathcal{A}'$ be two deterministic state machines, f and f' the functions computed by them and $g: \mathcal{M} \to \mathcal{M}'$ a morphism. Then $f \subseteq f'$. If g is proper, then f = f'.

Proof:
$G_e.q_0 \subseteq G_e'.q_0'$, $f = G_e.q_0$, $f' = G_e'.q_0'$. If g is proper, then $G_e.q_0 = G_e'.q_0'$. ⑥

3.4.1.2. State machine minimality.

In this section, the concept of minimal state machine will be defined. It will be shown that a minimal machine can be uniquely derived from the function that it computes. Obviously, if Q is a finite set, then a minimal machine is the machine with the minimum number of states that computes a certain function. However, the concept of minimal machine and its uniqueness (up to isomorphism) does not rely on the finiteness of the state set. The concepts and results in what follows are largely from Eilenberg, [12].

Definition 3.4.1.2.1.
Let $\mathcal{A} = (\Sigma, \Gamma, Q, F, G, q_0)$ be a deterministic state machine. Then a state $q \in Q$ is called *accessible* if $\exists\, s \in \Sigma^*$ such that $q = F_e(q_0, s)$. Also $\mathcal{A}$ is called *accessible* if all the states are accessible.

Definition 3.4.1.2.2.
Let $\mathcal{A} = (\Sigma, \Gamma, Q, F, G, q_0)$ be a deterministic state machine. Then
$\mathcal{A}' = (\Sigma, \Gamma, Q', F', G', q_0')$ defined by:

1. $Q' = \{q \in Q \mid \exists\, s \in \Sigma^*$ such that $q = F_e(q_0, s)\}$,
2. $F'(q, \sigma) = F(q, \sigma)$, $\forall\, q \in Q'$ and $\sigma \in \Sigma$,
3. $G'(q, \sigma) = G(q, \sigma)$, $\forall\, q \in Q'$ and $\sigma \in \Sigma$,
4. $q_0' = q_0$

is called *the accessible part* of $\mathcal{A}$, denoted $\mathcal{A}' = Acc(\mathcal{A})$. Also, Q' will be denoted by Acc(Q). It is clear that $\mathcal{A}'$ is accessible.

Definition 3.4.1.2.3.
Let $\mathcal{A}$ be a deterministic state machine and $S \subseteq \Sigma^*$. Then we define an equivalence relation $\sim_S$ on Q by $q \sim_S q'$ iff $G_e(q, s) = G_e(q', s)$, $\forall\, s \in S$. If $q \sim_S q'$, we say that q and q' are *S-equivalent*. If $S = \Sigma^*$ we say that q and q' are *equivalent* (written $q \sim q'$).

Observation 3.4.1.2.4.
If the output alphabet is empty (i.e. $\Gamma = \emptyset$) then
$q \sim_S q'$ iff $(\forall\, s \in S, (q, s) \in \text{dom } F_e$ iff $(q', s) \in \text{dom } F_e)$.

In other words, $\forall s \in S$, there exists a path labelled s from q iff there exists a path labelled s from q'.

Definition 3.4.1.2.5.
A deterministic state machine $\mathcal{M}$ is called *reduced* if $\forall q, q' \in Q, q \sim q' \Rightarrow q = q'$.

Definition 3.4.1.2.6.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, F, G, q_0)$ be a deterministic state machine. Then we define $\mathcal{M}' = \mathcal{M}/\sim$, $\mathcal{M}' = (\Sigma, \Gamma, Q', F', G', q_0')$ by:

1. $Q' = Q / \sim = \{[q] | q \in Q\}$, where [q] is the equivalence class of q,
2. $F'([q], \sigma) = [F(q, \sigma)], \forall q \in Q$ and $\sigma \in \Sigma$,
3. $G'([q], \sigma) = G(q, \sigma), \forall q \in Q'$ and $\sigma \in \Sigma$,
4. $q_0' = [q_0]$

Then $\mathcal{M}'$ is called the *reduced part* of $\mathcal{M}$, denoted $\mathcal{M}' = \text{Red}(\mathcal{M})$. It is clear that $\mathcal{M}'$ is reduced.

Note: F' and G' are well defined since

$$q \sim q' \Rightarrow G(q, \sigma) = G(q', \sigma) \text{ and } F(q, \sigma) \sim F'(q, \sigma).$$

Definition 3.4.1.2.7.
A deterministic state machine $\mathcal{M}$ is called *minimal* if it is accessible and reduced.

It will be proven that the minimal state machine that computes a certain function is unique up to an isomorphism and it can be derived uniquely from the function it computes. First we need the following definition.

Definition 3.4.1.2.8.
Let $f: \Sigma^* \to \Gamma^*$ be a partial stream function. Then $\forall s \in \text{dom } f$ we define a new partial function $f.s: \Sigma^* \to \Gamma^*$ by

$$f.s(x) = f(s)^{-1} f(sx)$$

Note: $\forall s, t \in \Sigma^*$ if $\exists u \in \Sigma^*$ such that $t = su$, then $s^{-1}t = u$.
It is clear that f.s is also a partial stream function.

Then we have the following result.

Proposition 3.4.1.2.9.
Let $f: \Sigma^* \to \Gamma^*$ be a partial stream function and let $\mathcal{M} = (\Sigma, \Gamma, Q, F, G, q_0)$ be a deterministic state machine defined as follows:

1. $Q = \{f.s | s \in \text{dom } f\}$,
2. F is defined by:

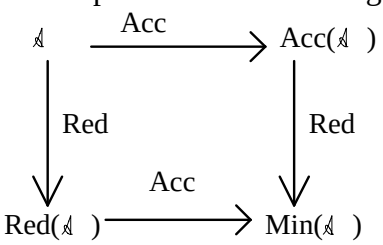
$$F(f.s, \sigma) = \begin{cases} f.s\sigma, & \text{if } s\sigma \in \text{dom } f \\ \emptyset, & \text{otherwise} \end{cases}$$

3. G is defined by:
   $G(f.s, \sigma) = f.s(\sigma)$,
4. $q_0 = f$

Then $\mathcal{A}$ is a minimal deterministic state machine. Furthermore, if $\mathcal{A}'$ is a minimal deterministic state machine that computes f, then $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.

Proof:
The fact that $\mathcal{A}$ is accessible and reduced follows from the construction of $\mathcal{A}$. If $\mathcal{A}' = (\Sigma, \Gamma, Q', F', G', q_0')$ is a minimal state machine that computes f, then we define a function g: $Q' \to Q$ by $g(q) = G_e.q$. Since $\mathcal{A}'$ is accessible, it follows that g is surjective. Since $\mathcal{A}'$ is reduced, it follows that g is injective. It can be verified easily that g is a proper morphism. ⑥

Therefore, any two minimal state machines that compute the same function are isomorphic and the following diagram commutes.



Furthermore, the minimal machine can be determined uniquely from the function it computes.

If the state set is finite Acc($\mathcal{A}$) and Red($\mathcal{A}$) can be determined by some finite algorithm (see Eilenberg [12]). Hence, the minimal machine that computes the same function as a certain automaton $\mathcal{A}$ can be determined algorithmically. Also, if f is a partial function computed by a finite state machine $\mathcal{A}$, then

$$\text{card}(\{f.s \mid s \in \text{dom } f\}) \leq \text{card}(Q),$$

where Q is the state set of $\mathcal{A}$. Then, we can give the following characterisation of the functions computed by finite state machines.

Corollary 3.4.1.2.10.
Let f: $\Sigma^* \to \Gamma^*$ a partial function. Then there exists a finite state machine $\mathcal{A}$ that computes f if and only if f is a partial stream function and the set $\{f.s \mid s \in \text{dom } f\}$ is finite.

3.4.2. Stream X-machine minimality

Let us return to stream X-machines. In this section we shall be discussing the minimisation problem for stream X-machines. First, we make the following remark.

It is clear that a computation that a stream X-machine performs is completely determined by: the state set Q, the memory set M, the transition function u, the output function λ, the initial state q_0, the initial memory value m_0 and the set of terminal states T (T will be ignored in what follows since will consider that all the

states are terminal, i.e. T = Q). Of course these will not determine uniquely the type Φ and the transition diagram F (there might be machines with different Φ's and F's that have the same Q, M, u, λ, q_0, m_0), but they determine uniquely the function computed. Therefore, if we are interested only in the functionality of the machine and we do not restrict the type Φ used to a particular set of functions, then we can treat a stream X-machine as an (infinite) state machine

$$\mathcal{M} = (\Sigma, \Gamma, P, u, \lambda, p_0),$$

in which an actual state p is a pair (q, m), where q is the current state and m is the current memory value (i.e. the state set is $P = Q \times M$ with the initial state $p_0 = (q_0, m_0)$). Therefore, the minimisation techniques presented in section 3.4.1.2 can be applied and the resulting machine will be minimal w.r.t. $Q \times M$ (i.e. the new state set Q' and memory M' are chosen such that $Q' \times M'$ is the set P' of the minimal (infinite) state machine with the same functionality as $\mathcal{M} = (\Sigma, \Gamma, P, u, \lambda, p_0)$). However, this type of minimisation is usually of little interest for the following reasons:

M is usually infinite (in practice very large), hence the minimal state machine cannot be determined algorithmically.

The basic functions Φ' obtained using this type of minimisation can be much more complicated then the initial ones. Additionally, in many cases the X-machine model is used to build more complex models from simpler components (the ϕ's) and hence the memory and the set Φ of the minimal machine should be the same as the initial one.

However, this type of state machine minimisation could be used in special cases (i.e. when M is finite and not too large or when the construction of the minimal state machine is obvious), the purpose being to minimise the resources (i.e. $Q \times M$) with which a certain functionality can be achieved.

Having said that, we shall now discuss a more useful type of stream X-machine minimality, i.e. given a certain stream X-machine $\mathcal{M}$ with the type Φ that computes the function f what is the 'smallest' (i.e. in terms of states) stream X-machine with the same type Φ that computes f ? Obviously, we can convert an X-machine into a finite state machine by treating the elements of Φ as abstract symbols. But the minimisation of an X-machine is more complicated then that of a finite state machine since each element in Φ has a well defined semantics (it is a partial function) rather than being merely an input symbol. However, we shall show that, if Φ satisfies certain conditions, then the minimisation of a stream X-machine can be reduced to that of a finite state machine. First we need the following definitions and preparatory results.

As we have mentioned, given an X-machine $\mathcal{M}$, we can convert it into a finite state machine $\mathcal{A}$ by treating the elements of Φ as abstract input symbols. We are, in effect, "forgetting" the memory structure and the semantics of the elements of Φ. We call this the *associated automaton* of $\mathcal{M}$.

Definition 3.4.2.1.
Let $\mathcal{M}$ = (Σ, Γ, Q, M, Φ, F, q_0, m_0) be a deterministic X-machine. Then the automaton $\mathcal{A}$ = (Φ, Q, F, q_0) over the alphabet Φ is called the *associated automaton* of $\mathcal{M}$.

Recall that Φ denotes the (possibly infinite) set of basic functions that could be used by the machine. However, only a finite subset $\Phi' \subseteq \Phi$ is actually used, hence $\mathcal{A}$ will be an automaton over the input alphabet Φ'. For the sake of simplicity, Φ' will be not mentioned explicitly (of course, it can be derived from the definition of F).

Obviously $\mathcal{A}$ is an automaton with empty output alphabet. We shall extend F to F_e and define $F_e.q$ as in section 3.4.1. It is clear that dom $F_e.q_0$ is the language accepted by $\mathcal{A}$ and $\forall$ q $\in$ Q, dom $F_e.q$ is the language accepted by $\mathcal{A}_q$ = (Φ, Q, F, q), where $\mathcal{A}_q$ is the automaton obtained from $\mathcal{A}$ by considering q as initial state. Then, we have the following result.

Lemma 3.4.2.2.
Let $\mathcal{M}$ = (Σ, Γ, Q, M, Φ, F, q_0, m_0) and $\mathcal{M}'$ = (Σ, Γ, Q', M, Φ, F', q_0', m_0) be two deterministic stream X-machines with the same type Φ and initial memory m_0, $\mathcal{A}$ and $\mathcal{A}'$ their associated automata and let q $\in$ Q, q' $\in$ Q'. If dom $F_e.q$ = dom $F_e.q'$, then $\lambda_e(q, m, s) = \lambda_e'(q', m, s)$, $\forall$ m $\in$ M, s $\in \Sigma^*$. In particular, if $\mathcal{A}$ and $\mathcal{A}'$ accept the same language, then f = f', where f and f' are the partial functions computed by $\mathcal{M}$ and $\mathcal{M}'$ respectively.

Proof:
By induction on k = |s| it follows easily that:
$\forall$ q $\in$ Q, q' $\in$ Q' such that dom $F_e.q$ = dom $F_e.q'$, then $\forall$ m $\in$ M, s $\in \Sigma^*$ either
i). $\lambda_e(q, m, s) = \lambda_e'(q', m, s) = \varnothing$ or
ii). $\lambda_e(q, m, s) = \lambda_e'(q', m, s) \neq \varnothing$, $w_e(q, m, s) = w_e'(q', m, s)$ and
dom $F_e.q_1$ = dom $F_e.q_1'$, where $q_1 = v_e(q, m, s)$ and $q_1' = v_e(q', m, s)$. ⑥

In general, the converse implication is not true, a counter-example will be given later on (see example 3.4.2.7).

Similar to the definition of an accessible state for state machines, we shall say that a state q of a stream X-machine is reachable if there exists an input sequence which takes the machine from initial state q_0 and initial memory m_0 to q. We shall also say that a memory value m is attainable in q if there exists an input sequence which takes the machine from q_0 and m_0 to q and m.

Definition 3.4.2.3.
A state q is called *reachable* if $\exists$ s $\in \Sigma^*$ such that $u_e(q_0, m_0, s) = (q, m)$ for some m $\in$ M.

Definition 3.4.2.4.
Given a stream X-machine $\mathcal{M}$, $q \in Q$ and $m \in M$, m is *attainable* in q if there is an input sequence $s \in \Sigma^*$ such that $u_e(q_0, m_0, s) = (q, m)$.

Definition 3.4.2.5.
Given a stream X-machine $\mathcal{M}$ and $q \in Q$,

$Att(q) = \{m \in M | \text{ m is attainable in q}\}$

We can now formalise the concept of a minimal stream X-machine. In what follows we shall be referring to deterministic stream X-machines with the same memory ($M = M'$), type ($\Phi = \Phi'$), initial memory value ($m_0 = m_0'$), input ($\Sigma = \Sigma'$) and output ($\Gamma = \Gamma'$) alphabets. Then we have the following definition.

Definition 3.4.2.6.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine which computes a partial function f. Then $\mathcal{M}$ is called *Q-minimal w.r.t.* Φ if it satisfies the following:

1. If $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ is a deterministic stream X-machine with the same type Φ and initial memory value m_0 which computes the same partial function f, then $\mathcal{M}'$ has at least the same number of states as $\mathcal{M}$ (i.e. $card(Q') \geq card(Q)$).

2. By removing any arc or number of arcs from $\mathcal{M}$, the function computed will change.

Note: An equivalent form of condition 2 is the following:

2'. If $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ is a deterministic stream X-machine with the same type Φ and initial memory value m_0 which computes the same partial function f and there exists a bijective morphism $g: \mathcal{A}' \rightarrow \mathcal{A}$, where $\mathcal{A}$ and $\mathcal{A}'$ are the associated automata of $\mathcal{M}$ and $\mathcal{M}'$ respectively, then g is an isomorphism.

The Q-minimality as presented above has some obvious properties:

1. If $\mathcal{M}$ is Q-minimal w.r.t. Φ, then each state is reachable.

2. If $\mathcal{M}$ and $\mathcal{M}'$ are two stream X-machine such that $\mathcal{M}$ is Q-minimal w.r.t. Φ and their associated automata $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic, then $\mathcal{M}'$ is Q-minimal w.r.t. Φ.

3. If $\mathcal{M}$ is Q-minimal w.r.t. Φ, then the associated automaton $\mathcal{A}$ is minimal.

However, the converse implication (i.e. if $\mathcal{A}$ is minimal, then $\mathcal{M}$ is Q-minimal w.r.t. Φ) is not always true. For example, let $\mathcal{M}$ be the stream X-machine from example 3.4.2.7.

Example 3.4.2.7.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$, with $\Sigma = \{a, b\}$, $\Gamma = \{x, y\}$, $Q = \{q_0, q_1\}$, $M = \{0, 1\}$, $m_0 = 0$, $\Phi = \{\phi_1, \phi_2\}$ and the next state function F as represented in figure 3.10.

ϕ_1, ϕ_2: $M \times \Sigma \rightarrow \Gamma \times M$ are partial functions defined by:

dom ϕ_1 = {(0, a)}, $\phi_1(0, a) = (x, 0)$;

dom ϕ_2 = {(1, b)}, $\phi_2(1, b) = (y, 1)$

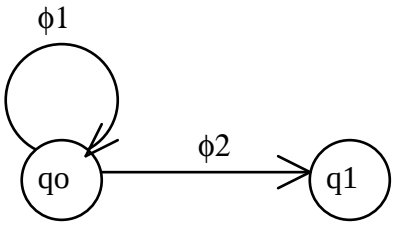


Figure 3.10.

It is clear that the associated automaton $\mathcal{A}$ of $\mathcal{M}$ is minimal but q_1 is not reachable. Even if each state of the machine is reachable, the minimality of the associated automaton does not guarantee that the machine is Q-minimal. For example, let $\mathcal{M}$ be the stream X-machine from example 3.4.2.8.

Example 3.4.2.8.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_o, m_o)$, where $\Sigma = \{a, b\}$, $\Gamma = \{x, y\}$, $Q = \{q_o, q_1\}$, $M = \{0, 1\}$, $m_o = 0$, $\Phi = \{\phi_1, \phi_2\}$, with ϕ_1, ϕ_2: $M \times \Sigma \rightarrow \Gamma \times M$ being the partial functions defined in example 3.4.2.7, and the next state function F represented in figure 3.11.

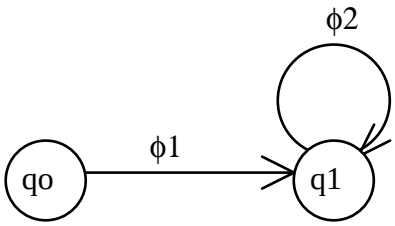


Figure 3.11.

It is clear that the associated automaton $\mathcal{A}$ of $\mathcal{M}$ is minimal and both q_o and q_1 are reachable. However, since

Att(q_1) = {0} and

$\neg\exists\ \sigma \in \Sigma$ such that $(0, \sigma) \in$ dom ϕ_2,

ϕ_2 can be removed from figure 3.11, without changing the functionality of the machine.

However, if Φ satisfies some additional conditions, the Q-minimality can be reduced to the minimality of the associated automaton.

Definition 3.4.2.9.
Let $\phi_1, \phi_2 \in \Phi$. Then ϕ_1, ϕ_2 are said to be *output-distinguishable* if:

$\forall$ m $\in$ M, $\sigma \in \Sigma$, if $\phi_1(m, \sigma) = (\gamma_1, m_1')$ and $\phi_2(m, \sigma) = (\gamma_2, m_2')$, with $m_1', m_2' \in$ M, $\gamma_1, \gamma_2 \in \Gamma$, then $\gamma_1 \neq \gamma_2$.

Definition 3.4.2.10.
A type Φ is called *output-distinguishable* if:

$\forall \phi_1, \phi_2 \in \Phi$, then either $\phi_1 = \phi_2$ or ϕ_1 and ϕ_2 are output-distinguishable.

What this is saying is that we must be able to distinguish between any two different processing functions (the ϕ's) by examining outputs. If we cannot then we will not be able to tell them apart.

Definition 3.4.2.11.
Let $\phi \in \Phi$. Then ϕ is said to be *complete* if:

$\forall$ m $\in$ M, $\exists \sigma \in \Sigma$ such that $(m, \sigma) \in$ dom ϕ.

Definition 3.4.2.12.
A type Φ is called *complete* if:

$\forall \phi \in \Phi$, ϕ is complete.

This condition will ensure that all the accessible states in $\mathcal{A}$ are reachable in $\mathcal{M}$. Also, it will prevent the situation in which there exist q $\in$ Q and $\phi \in \Phi$, such that

$(Att(q) \times \Sigma) \cap$ dom $\phi = \varnothing$.

For example, all three X-machine models of the correlator have the type complete and output-distinguishable.

We can prove now that if the associated automaton $\mathcal{A}$ is minimal and the type Φ is complete and output-distinguishable, then $\mathcal{M}$ is Q-minimal w.r.t. Φ. This will be proved by showing that if two machines with Φ complete and output-distinguishable compute the same function, then their associated automata accept the same language.

Lemma 3.4.2.13.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ be two deterministic stream X-machines with Φ complete and output-distinguishable and let q $\in$ Q and q' $\in$ Q' and m $\in$ M. If $\lambda(q, m, s) = \lambda'(q', m, s)$, $\forall$ s $\in \Sigma^*$ then q and q' are Φ^*-equivalent as states in $\mathcal{A}$ and $\mathcal{A}'$ respectively. In particular, if $\mathcal{M}$ and $\mathcal{M}'$ compute the same function, then $\mathcal{A}$ and $\mathcal{A}'$ accept the same language.

Proof:
We prove by induction that

$\forall k > 0$, $\lambda_e(q, m, s) = \lambda_e'(q', m, s)$, $\forall s \in \Sigma^*$ with $|s| \le k$, $\Rightarrow$ q and q' are Φ^k-equivalent.

Let us prove that this is true for k = 1. Let $q \in Q$, $q' \in Q'$ and $m \in M$ such that

$\lambda_e(q, m, \sigma) = \lambda_e'(q', m, \sigma)$, $\forall \sigma \in \Sigma$.

Let $\phi \in \Phi$ be such that $F(q, \phi) \neq \varnothing$ (i.e. there exists an arc labelled ϕ from q). Since Φ is complete, $\exists \sigma \in \Sigma$ such that $(m, \sigma) \in \text{dom } \phi$. Then let

$\phi(m, \sigma) = (\gamma, m_1)$, with $\gamma \in \Gamma$, $m_1 \in M$.

Then $\lambda(q, m, \sigma) = \gamma$, hence $\lambda'(q', m, \sigma) = \gamma$. Then $\exists \phi' \in \Phi$ such that $F'(q', \phi') \neq \varnothing$ and

$\phi'(m, \sigma) = (\gamma, m_1')$ with $m_1' \in M$.

Since Φ is output-distinguishable, we have $\phi' = \phi$.

Then

$\forall \phi \in \Phi$, if $F(q, \phi) \neq \varnothing$ then $F'(q', \phi) \neq \varnothing$.

Similarly we can prove that

$\forall \phi \in \Phi$, if $F'(q', \phi) \neq \varnothing$ then $F(q, \phi) \neq \varnothing$.

Hence q and q' are Φ-equivalent.

Let us assume that the statement above is true for $k \ge 1$. Let $q \in Q$, $q' \in Q'$ and $m \in M$ such that

$\lambda_e(q, m, s) = \lambda_e'(q', m, s) \forall s \in \Sigma^*$ with $|s| \le k+1$.

It is clear that q and q' are Φ^{k+1}-equivalent iff:

i) q and q' are Φ-equivalent and

ii) $\forall \phi \in \Phi$ such that $F(q, \phi) \neq \varnothing$, q_1 and q_1' are Φ^k-equivalent, where $q_1 = F(q, \phi)$ and $q_1' = F(q, \phi)$.

Since q and q' are Φ^k-equivalent, they are Φ-equivalent. Let $\phi \in \Phi$ such that $F(q, \phi) \neq \varnothing$ and let $q_1 = F(q, \phi)$ and $q_1' = F'(q', \phi)$. Since Φ is complete, $\exists \sigma \in \Sigma$ such that $(m, \sigma) \in \text{dom } \phi$. Then let

$\phi(m, \sigma) = (\gamma, m_1)$, with $\gamma \in \Gamma$, $m_1 \in M$.

Then $u(q, m, \sigma) = (q_1, m_1)$ and $u'(q', m, \sigma) = (q_1', m_1)$. Since

$\lambda_e(q, m, \sigma s) = \lambda(q, m, \sigma)\, \lambda_e(q_1, m_1, s)$,

$\lambda_e'(q, m, \sigma s) = \lambda'(q, m, \sigma)\, \lambda_e'(q_1', m_1, s)$ $\forall s \in \Sigma^*$ and

$\lambda_e(q, m, \sigma s) = \lambda_e'(q', m, \sigma s)$ $\forall s \in \Sigma^*$ with $|s| \le k$,

we have

$\lambda_e(q_1, m_1, s) = \lambda_e'(q_1', m_1, s)$ $\forall s \in \Sigma^*$ with $|s| \le k$.

Hence q_1 and q_1' are Φ^k-equivalent. ⑥

Theorem 3.4.2.14.

Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0, m_0)$ be two deterministic stream X-machines which compute the same partial function (f = f') and let $\mathcal{A}$ and $\mathcal{A}'$ their associated automata. If Φ is output-distinguishable and complete and $\mathcal{A}$ and $\mathcal{A}'$ are minimal, then $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.

Proof:
From lemma 3.4.2.13. it follows that $\mathcal{A}$ and $\mathcal{A}'$ accept the same language. Since $\mathcal{A}$ and $\mathcal{A}'$ are minimal, it follows that $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic. ©

Corollary 3.4.2.15.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine with Φ output-distinguishable and complete. If the associated automaton $\mathcal{A}$ is minimal then $\mathcal{M}$ is Q-minimal w.r.t. Φ.

Proof:
Let f the function computed by $\mathcal{M}$ and let $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ be a Q-minimal stream X-machine w.r.t. Φ which computes f and $\mathcal{A}'$ its associated automaton. Then $\mathcal{A}'$ is minimal. From theorem 3.4.2.14 it follows that $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic. Hence $\mathcal{M}$ is Q-minimal w.r.t. Φ. ©

Therefore, if Φ is output-distinguishable and complete the Q-minimality can be reduced to the minimality of the associated automaton and the Q-minimal machine which computes a certain function is unique up to an isomorphism of the associated automata. This could be very significant for a stream X-machine testing theory. Indeed, if the specification and the implementation of a system can be described as two stream X-machines $\mathcal{M}$ and $\mathcal{M}'$ with the type Φ complete and output-distinguishable, then testing that the implementation satisfies the specification will be reduced to showing that the minimal automata of $\mathcal{M}$ and $\mathcal{M}'$ respectively are isomorphic. On the other hand, it is fairly clear that any type Φ can be transformed into a complete and output-distinguishable type by augmenting the input and output alphabets. These ideas will be discussed in more detail in the next chapter.

Obviously, if Φ is output-distinguishable and complete the Q-minimal machine that computes the same function as a certain stream X-machine can be determined algorithmically (i.e. we apply an algorithm which minimises the associated automaton).

3.4.3. Minimal coverings.

Let $\mathcal{M}$ be a stream X-machine specification of a system and f be the partial function it computes. In some situations it might be acceptable to add extra functionality to the system, as long as the 'minimal' functionality required determined by f remains unchanged. Therefore, any machine $\mathcal{M}'$ which computes f' such that $f \subseteq f'$ will be an acceptable specification of the same system. In this context, a natural question that arises is how we can determine such as $\mathcal{M}'$ with the minimal number of states. In what follows we shall address this problem for stream X-machines with Φ complete and output-distinguishable. First, let us formalise this problem.

Definition 3.4.3.1.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine which computes a partial function f. Then a deterministic stream X-machine $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ with the same type Φ and initial memory value m_0 is called a *covering of* $\mathcal{M}$ *w.r.t.* Φ if $f \subseteq f'$, where f' is the partial function computed by $\mathcal{M}'$.

Definition 3.4.3.2.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine which computes a partial function f. Then a deterministic stream X-machine $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$, is called a *minimal covering of* $\mathcal{M}$ *w.r.t.* Φ if:

1. $\mathcal{M}'$ is a covering of $\mathcal{M}$ w.r.t. Φ.
2. If $\mathcal{M}''$ is a covering of $\mathcal{M}$ w.r.t. Φ, then $\mathcal{M}''$ has at least the same number of states as $\mathcal{M}'$ (i.e. $card(Q'') \geq card(Q')$).
3. If $\mathcal{M}''$ is a deterministic stream X-machine obtained from $\mathcal{M}'$ by removing any arc or number of arcs, then $\mathcal{M}''$ is no longer a covering of $\mathcal{M}$ w.r.t. Φ.

Note: Similar to the definition of Q-minimality, condition 3 can be formalised in terms of morphisms.

Since the state set of $\mathcal{M}$ is finite, it is clear that there exists at least one minimal covering of $\mathcal{M}$. We will now show how all the minimal coverings of a stream X-machine with Φ output-distinguishable and complete can be constructed. Obviously, it is sufficient to solve this problem for machines whose associated automata are minimal (otherwise, the associated automaton of the machine is minimised first). First, we introduce some new concepts and prove some preparatory results.

Definition 3.4.3.3.
Let $\mathcal{A} = (\Phi, Q, F, q_0)$ and $\mathcal{A}' = (\Phi, Q', F', q_0')$ be two deterministic state machines with empty output alphabet. Then we say that $\mathcal{A}' \geq \mathcal{A}$ if there exists a function $g: Q \rightarrow Q'$ such that

$g(q_0) = q_0'$ and
$dom\ F_e.q \subseteq dom\ F_e'.g(q), \forall\ q \in Q.$

In other words, $g: Q \rightarrow Q'$ will satisfy $g(q_0) = q_0'$ and $\forall\ q \in Q$, if $\exists$ a path in $\mathcal{A}$ labelled $\phi_1 ... \phi_k$ emerging from q then $\exists$ a path in $\mathcal{A}'$ labelled $\phi_1 ... \phi_k$ emerging from $q' = g(q)$.

Obviously, if $\mathcal{A}' \geq \mathcal{A}$ then $L \subseteq L'$, where L and L' are the languages accepted by $\mathcal{A}$ and $\mathcal{A}'$ respectively. If $\mathcal{A}$ is accessible, then the converse implication is also true.

Lemma 3.4.3.4.
Let $\mathcal{A} = (\Phi, Q, F, q_0)$ and $\mathcal{A}' = (\Phi, Q', F', q_0')$ be two deterministic state machines, $\mathcal{A}$ accessible and L and L' respectively the languages accepted by them. If $L \subseteq L'$, then $\mathcal{A}' \geq \mathcal{A}$.

Proof:
We define g as follows. Obviously, we take $g(q_0) = q_0'$. Let $q \in Q - \{q_0\}$. If $\mathcal{A}$ is accessible, then $\exists\, \nu \in \Phi^+$ such that $F_e(q_0, \nu) = q$. Then, we take

$$g(q) = F_e'(q_0', \nu).$$

It is easy to verify that dom $F_e.q \subseteq$ dom $F_e'.g(q)$. ⑥
Note: Clearly, g might not be unique.

Let $\mathcal{M}$ and $\mathcal{M}'$ be two deterministic stream X-machines with the same memory, type and initial memory value and let $\mathcal{A}$ and $\mathcal{A}'$, respectively, be their associated automata. If $\mathcal{A}' \geq \mathcal{A}$ then it can be proven easily that the partial function computed by $\mathcal{M}$ will be included in the one computed by $\mathcal{M}'$ (i.e. it follows by induction similarly to lemma 3.4.2.2). If Φ is complete and output-distinguishable and $\mathcal{A}$ is accessible, then the converse implication is also true.

Lemma 3.4.3.5.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ and $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ be two deterministic stream X-machines with Φ complete and output-distinguishable, f and f', respectively, be the partial functions computed by them and $\mathcal{A}$ and $\mathcal{A}'$, respectively, be their associated automata. If $\mathcal{A}$ is accessible and $f \subseteq f'$, then $\mathcal{A}' \geq \mathcal{A}$.

Proof:
By induction (similar to the proof of lemma 3.4.2.13), it follows that $L \subseteq L'$, where L and L' are the languages accepted by $\mathcal{A}$ and $\mathcal{A}'$ respectively. Hence, using lemma 3.4.3.4, we have $\mathcal{A}' \geq \mathcal{A}$. ⑥

Therefore, if $\mathcal{M}$ is a deterministic stream X-machine with Φ complete and output-distinguishable and its associated automaton $\mathcal{A}$ is minimal, the problem of finding the minimal coverings of $\mathcal{M}$ can be reduced to finding all the machines $\mathcal{M}'$ such that:

1. $\mathcal{A}' \geq \mathcal{A}$, where $\mathcal{A}'$ is the associated automaton of $\mathcal{M}'$.
2. If $\mathcal{M}''$ is a deterministic stream X-machine with the associated automaton $\mathcal{A}''$ such that $\mathcal{A}'' \geq \mathcal{A}$, then $\mathcal{A}''$ has at least as many states as $\mathcal{A}'$.
3. If $\mathcal{A}'''$ is an automaton obtained by removing an arc or a number of arcs from $\mathcal{A}'$, then $\mathcal{A}''' \geq \mathcal{A}$ is not satisfied.

We shall show that the set of minimal coverings can be effectively constructed using special kinds of decompositions of the state set of $\mathcal{M}$.

Definition 3.4.3.6.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine. Then a set $H = \{H_i\}_{i \in I}$, $H_i \neq \emptyset$, $\forall\, i \in I$, is called an *admissible decomposition* of Q if:

i) $\bigcup_{i \in I} H_i = Q$ and

ii) $\forall i \in I$ and $\forall \phi \in \Phi$, $\exists j \in I$ such that if $F(q, \phi) \neq \emptyset$, then $F(q, \phi) \in H_j$, $\forall q \in H_i$.

Definition 3.4.3.7.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine. Then two states $q, q' \in Q$ are called *domain-compatible* if:

$\forall \phi, \phi' \in \Phi$ such that $F(q, \phi) \neq \emptyset$ and $F(q', \phi') \neq \emptyset$, then either $\phi = \phi'$ or $\text{dom } \phi \cap \text{dom } \phi' = \emptyset$.

Definition 3.4.3.8.
A set $H = \{H_i\}_{i \in I}$, $H_i \neq \emptyset$, $\forall i \in I$, is called a *domain-consistent decomposition* of Q if:

i) $\bigcup_{i \in I} H_i = Q$ and

ii) $\forall i \in I$, $q, q' \in H_i$, q and q' are domain-compatible.

If $\mathcal{M}$ is a stream X-machine with associated automaton $\mathcal{A}$, then any stream X-machine $\mathcal{M}'$ whose associated automaton $\mathcal{A}'$ satisfies $\mathcal{A}' \geq \mathcal{A}$ determines an admissible and domain-consistent decomposition of the state set of $\mathcal{M}$.

Lemma 3.4.3.9.
Let $\mathcal{M}$ and $\mathcal{M}'$ be two deterministic stream X-machines and $\mathcal{A}$ and $\mathcal{A}'$ be their associated automata such that $\mathcal{A}' \geq \mathcal{A}$. Let $h: Q' \rightarrow \wp Q$ be defined by

$h(q') = \{q \in Q | \text{ dom } F_e.q \subseteq \text{dom } F_e'.q'\}$.

Then the set

$H = \{h(q') | q' \in Q' \text{ and } h(q') \neq \emptyset\}$

is an admissible and domain-consistent decomposition of Q.

Proof:
Let $q' \in Q'$ such that $h(q') \neq \emptyset$ and $q_1, q_2 \in h(q')$. Hence $\text{dom } F_e.q_1 \subseteq \text{dom } F_e'.q'$ and $\text{dom } F_e.q_2 \subseteq \text{dom } F_e'.q'$. Then let $\phi, \phi' \in \Phi$ $\phi \neq \phi'$ such that $F(q_1, \phi) \neq \emptyset$ and $F(q_2, \phi') \neq \emptyset$. Thus $F'(q', \phi) \neq \emptyset$ and $F'(q', \phi') \neq \emptyset$. Since $\mathcal{M}'$ is deterministic, it follows that $\text{dom } \phi \cap \text{dom } \phi' = \emptyset$. Hence H is domain-compatible.
Since $\forall q' \in Q'$, $\phi \in \Phi$, if $q \in h(q')$ and $F(q, \phi) \neq \emptyset$ then $F(q, \phi) \in h(F'(q', \phi))$ $\forall q \in Q$, H is also admissible. ⑥

Conversely, an admissible and domain-consistent decomposition of the state set of $\mathcal{M}$ determines at least one machine $\mathcal{M}'$ whose associated automaton $\mathcal{A}'$ satisfies $\mathcal{A}' \geq \mathcal{A}$.

Definition 3.4.3.10.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine and $H = \{H_i\}_{i \in I}$ be an admissible and domain-consistent decomposition of Q. Then, we can define at least one deterministic stream X-machine

$\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$

as follows:

1. $Q' = \{H_i\}_{i \in I}$
2. F' is defined by:

$$F'(H_i, \phi) = \begin{cases} H_j, \text{ if } \exists\, q \in H_i \text{ such that } F(q, \phi) \neq \emptyset, \\ \quad \text{where j is chosen such that } F(q, \phi) \in H_j \ \forall\, q \in H_i \\ \emptyset, \text{ if } F(q, \phi) = \emptyset \ \forall\, q \in H_i \end{cases}$$

3. $q_0' = H_j$, where H_j is chosen such that $q_0 \in H_j$.

The set of all the machines $\mathcal{M}'$ defined as above will be denoted by $\mathcal{M}/H$.

Since H is admissible F' is well defined. Also since H is domain-consistent, $\mathcal{M}'$ is deterministic. Indeed, let $i \in I$ and $\phi, \phi' \in \Phi$ such that $\phi \neq \phi'$. Then, if $F'(H_i, \phi) \neq \emptyset$ and $F'(H_i, \phi') \neq \emptyset$ then $\exists\, q, q' \in H_i$ such that $F(q, \phi) \neq \emptyset$ and $F(q', \phi') \neq \emptyset$. Since q and q' are compatible, $dom\phi \cap dom\,\phi' = \emptyset$.

Obviously, the definition of F' leads to more than one machine if $\exists\, i, j, k \in I$, $j \neq k$, and $\phi \in \Phi$ such that $F(H_i, \phi) \subseteq H_j$ and $F(H_i, \phi) \subseteq H_k$. Also, if $\exists\, i, j \in I$ such that $q_0 \in H_i$ and $q_0 \in H_j$, then the definition above leads to more than one machine. Obviously, if H is a partition (i.e. $H_i \cap H_j = \emptyset$, $\forall\, i, j \in I$, $i \neq j$), then $\mathcal{M}/H$ contains only one element.

Lemma 3.4.3.11.
Let $\mathcal{M}$ and $\mathcal{M}' \in \mathcal{M}/H$ as above and $\mathcal{A}$ and $\mathcal{A}'$ the associated automata of $\mathcal{M}$ and $\mathcal{M}'$ respectively. Then $\mathcal{A}' \geq \mathcal{A}$.

Proof:
We define $g: Q \to Q'$ by $g(q) = H_i$, where i is chosen such that $q \in H_i$. By induction it follows that $dom\, F_e.q \subseteq dom\, F_e'.g(q)\ \forall\, q \in Q$. ⑥

We can now prove the result we want.

Theorem 3.4.3.12.
Let $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ be a deterministic stream X-machine with Φ complete and output-distinguishable such that its associated automaton $\mathcal{A}$ is minimal. Also, let $\mathcal{M}' = (\Sigma, \Gamma, Q', M, \Phi, F', q_0', m_0)$ be a minimal covering of $\mathcal{M}$ w.r.t. Φ and $\mathcal{A}'$ the associated automaton of $\mathcal{M}'$. Then there exists $H = \{H_i\}_{i \in I}$ an admissible and domain-consistent decomposition of Q and $\mathcal{M}'' \in \mathcal{M}/H$ such that $\mathcal{A}'$ and $\mathcal{A}''$ are isomorphic, where $\mathcal{A}''$ is the associated automaton of $\mathcal{M}''$.

Proof:
Since $\mathcal{A}$ is minimal, $\mathcal{A}$ is accessible. From lemma 3.4.3.5 it follows that $\mathcal{A}' \geq \mathcal{A}$. Let $h: Q' \to \mathcal{P}Q$ defined by $h(q') = \{q \in Q \mid dom\, F_e.q \subseteq dom\, F_e'.q'\}$. Then the set

$$H = \{h(q') \mid q' \in Q' \text{ and } h(q') \neq \emptyset\}$$

is an admissible and domain-consistent decomposition of Q (see lemma 3.4.3.9).

From lemma 3.4.3.11 it follows that any element of $\mathcal{M}$/H is a covering of $\mathcal{M}$. Since $\mathcal{M}$' is a minimal covering, it follows that card(Q') ≤ card(H). Now, from the construction of the set H, we have card(H) ≤ card(Q'). Hence card(H) = card(Q') and the function h will have the following properties:

$h(q') \neq \emptyset \; \forall \; q' \in Q'$

$h(q_1') \neq h(q_2') \; \forall \; q_1', q_2' \in Q'$ with $q_1' \neq q_2'$.

Hence $h_r: Q' \rightarrow H$ defined by $h_r(q') = h(q')$ is a bijective function.
Let $k: H \rightarrow Q'$, $k = h_r^{-1}$. Obviously, k is also bijective.

Then we construct $\mathcal{M}'' = (\Sigma, \Gamma, Q'', M, \Phi, F'', q_o'', m_o)$ as follows:

1. Q" = H
2. F" is defined by:

$$F''(H_i, \phi) = \begin{cases} h(F'(k(H_i), \phi)), \text{ if } \exists\, q \in H_i \text{ such that } F(q, \phi) \neq \emptyset \\ \emptyset, \text{ if } F(q, \phi) = \emptyset \; \forall \; q \in H_i \end{cases}$$

3. $q_o'' = h(q_o')$

It follows easily that $\mathcal{M}'' \in \mathcal{M}$/H. Also,

$F''(H_i, \phi) \subseteq h(F'(k(H_i), \phi))$.

Since $h\,k = 1_{Q'}$, it follows that

$k(F''(H_i, \phi)) \subseteq F'(k(H_i), \phi)$.

Hence $k: \mathcal{A}'' \rightarrow \mathcal{A}'$ is a bijective morphism. Since $\mathcal{M}'$ is a minimal covering of $\mathcal{M}$ w.r.t. Φ, k is an isomorphism. ©

Therefore, if $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_o, m_o)$ is a deterministic stream X-machine with Φ complete and output-distinguishable and its associated automaton $\mathcal{A}$ is minimal, then the set of all minimal coverings of $\mathcal{M}$ w.r.t. Φ will be a subset of $\{\mathcal{M}' \in \mathcal{M}/H \mid$ H is any admissible and output-consistent decomposition of Q with minimum possible number of elements$\}$ (i.e. ∀ H' another admissible and domain-consistent decomposition of Q, card(H) ≤ card(H')). Thus, if we can determine algorithmically whether dom ϕ ∩ dom ϕ' is empty or not ∀ ϕ, ϕ' ∈ Φ, then the set of all minimal coverings can also be determined algorithmically. The algorithm we shall be giving in what follows involves a great deal of trial and error. It consists of the following steps:

1. Construct all the admissible and domain-consistent decompositions H of Q with minimum number of elements (let this number be n_o). Since $n_o \leq n$, it is clear that there exist only a finite number of such decompositions.

2. For each such H, construct all $\mathcal{M}' \in \mathcal{M}$/H. The set of all such $\mathcal{M}'$ will be denoted by Ξ.

3. Eliminate all the machines $\mathcal{M}' \in$ Ξ such that:

∃ $\mathcal{M}'' \in$ Ξ such that the associated automaton of $\mathcal{M}''$, $\mathcal{A}''$, can be obtained by removing one or more arcs from $\mathcal{A}'$, the associated automaton of $\mathcal{M}'$. The remaining elements of Ξ are all minimal coverings of $\mathcal{M}$.

The algorithm is illustrated by the following example.

Example 3.4.3.13.
Consider the stream X-machine $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_o, m_o)$, where $\Sigma = \{a, b\}$, $\Gamma = \{x, y\}$, $Q = \{q_o, q_1, q_2\}$, $M = \{0, 1\}$, $m_o = 0$, $\Phi = \{\phi_1, \phi_2\}$ and the next state function F represented in figure 3.12.
ϕ_1, ϕ_2: $M \times \Sigma \rightarrow \Gamma \times M$ partial functions defined by:

dom $\phi_1 = \{(0, a), (1, b)\}$; $\phi_1(0, a) = (x, 1), \phi_1(1, b) = (x, 0)$;
dom $\phi_2 = \{(0, b), (1, a)\}$; $\phi_2(0, b) = (y, 1), \phi_2(1, a) = (x, 0)$;
dom $\phi_3 = \{(0, a), (1, b)\}$; $\phi_3(0, a) = (y, 1), \phi_3(1, b) = (y, 0)$;

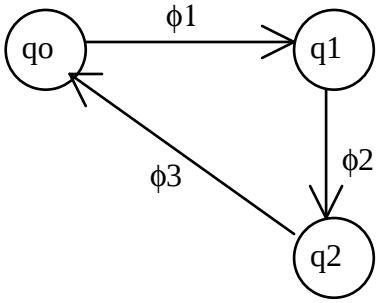


Figure 3.12.

$\mathcal{M}$ computes the partial function f: $\Sigma^* \rightarrow \Gamma^*$ defined by

dom f = {1, a, aa, aaa};
f(1) = 1;
f(a) = x;
f(aa) = xx;
f(aaa) = xxy.

It is clear that Φ is complete and output-distinguishable. Also, q_o is compatible with q_1 and q_1 is compatible with q_2 but q_o and q_2 are not compatible. Then, the admissible and domain-consistent decompositions of Q with the lowest cardinality are H_1, H_2 and H_3, where:

$H_1 = \{\{q_o\}, \{q_1, q_2\}\}$,
$H_2 = \{\{q_o, q_1\}, \{q_2\}\}$ and
$H_3 = \{\{q_o, q_1\}, \{q_1, q_2\}\}$.

Then $\mathcal{M}/H_1 = \{\mathcal{M}_1\}$, $\mathcal{M}/H_2 = \{\mathcal{M}_2\}$, $\mathcal{M}/H_3 = \{\mathcal{M}_3, \mathcal{M}_4\}$, where:

$\mathcal{M}_1 = (\Sigma, \Gamma, H_1, M, \Phi, F_1, \{q_o\}, m_o)$ with F_1 represented in figure 3.13;
$\mathcal{M}_2 = (\Sigma, \Gamma, H_2, M, \Phi, F_2, \{q_o, q_1\}, m_o)$ with F_2 represented in figure 3.14;
$\mathcal{M}_3 = (\Sigma, \Gamma, H_3, M, \Phi, F_3, \{q_o, q_1\}, m_o)$ with F_3 represented in figure 3.15;
$\mathcal{M}_4 = (\Sigma, \Gamma, H_3, M, \Phi, F_4, \{q_o, q_1\}, m_o)$ with F_4 represented in figure 3.16.

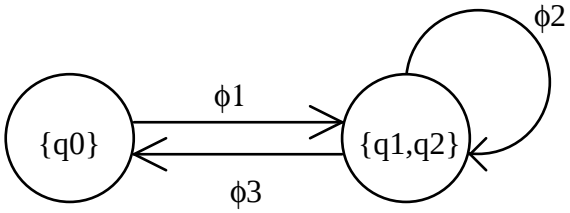


Figure 3.13.

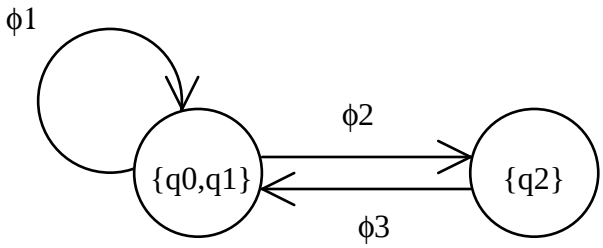


Figure 3.14.

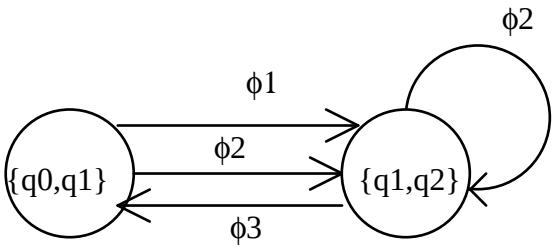


Figure 3.1.5.

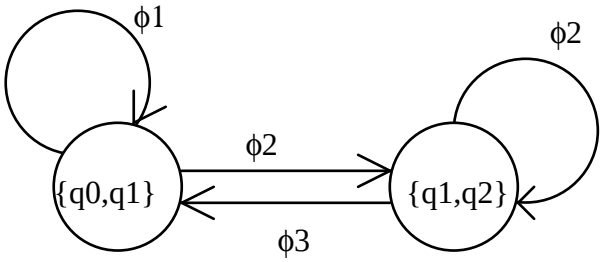


Figure 3.16.

It is clear that $\mathcal{M}_1$ and $\mathcal{M}_2$ can be obtained by removing a single arc from $\mathcal{M}_3$ and $\mathcal{M}_4$ respectively. Therefore, the set of all minimum coverings of $\mathcal{M}$ is $\{\mathcal{M}_1, \mathcal{M}_2\}$. $\mathcal{M}_1$ and $\mathcal{M}_2$ compute f_1: $\Sigma^* \rightarrow \Gamma^*$ and f_2: $\Sigma^* \rightarrow \Gamma^*$ respectively defined by:

dom $f_1 = \{1\} \cup \{a(ab)^n | \ n \geq 0\} \cup \{a(ab)^n a | \ n \geq 0\} \cup \{a(ab)^n b | \ n \geq 0\} \cup \{a(ab)^n aa | \ n \geq 0\}$,

$f_1(1) = 1$,
$f_1(a(ab)^n) = x(xy)^n$,
$f_1(a(ab)^n a) = x(xy)^n x$,
$f_1(a(ab)^n b) = x(xy)^n y$,
$f_1(a(ab)^n aa) = x(xy)^n xy$;

dom $f_2 = \{(ab)^n | \ n \geq 0\} \cup \{(ab)^n a | \ n \geq 0\} \cup \{(ab)^n b | \ n \geq 0\} \cup \{(ab)^n aa | \ n \geq 0\} \cup \{(ab)^n bb | \ n \geq 0\} \cup \{(ab)^n aaa | \ n \geq 0\}$,

$f_2((ab)^n) = (xx)^n$,
$f_2((ab)^n a) = (xx)^n x$,
$f_2((ab)^n b) = (xx)^n y$,
$f_2((ab)^n aa) = (xx)^n xx$,
$f_2((ab)^n bb) = (xx)^n yy$,
$f_2((ab)^n aaa) = (xx)^n xxy$.

Clearly, $f \subseteq f_1$ and $f \subseteq f_2$.

If the type Φ is not complete and output-distinguishable, then the algorithm above can still be used to find coverings of a certain machine $\mathcal{M}$ with the number of states less or equal to the number of states of $\mathcal{M}$, but in this case the machines obtained are not guaranteed to be minimal coverings.

Also, it is useful to notice that, if Φ is complete and output-distinguishable, then the two concepts presented above (i.e. Q-minimality and minimal covering) do not depend on the initial memory of the machine. Indeed, if a machine is Q-minimal w.r.t. Φ, then by changing the initial memory, it still remains Q-minimal. Also, if $\mathcal{M}'$ is a minimal covering of $\mathcal{M}$ w.r.t. Φ, then by changing the initial memory of both machines, $\mathcal{M}'$ will still be a minimal covering of $\mathcal{M}$.