EQUIVALENCE RELATIONS OF SYNCHRONOUS SCHEMES

CENTRE FOR NEWFOUNDLAND STUDIES

TOTAL OF 10 PAGES ONLY
MAY BE XEROXED

(Without Author's Permission)

BRANISLAV CIROVIC

001311


CENTRE FOR NFLD. STUDIES
MEMORIAL UNIVERSITY
OF NEWFOUNDLAND

Equivalence Relations of Synchronous Schemes

BY BRANISLAV CIROVIC

A thesis submitted to the School of Graduate Studies in
partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Department of Computer Science
Memorial University of Newfoundland

April 2000

St. John's Newfoundland

Abstract

Synchronous systems are single purpose multiprocessor computing machines, which provide a realistic model of computation capturing the concepts of pipelining, parallelism and interconnection. The syntax of a synchronous system is specified by the synchronous scheme, which is in fact a directed, labelled, edge-weighted multigraph. The vertices and their labels represent functional elements (the combinational logic) with some operation symbols associated with them, while the edges represent interconnections between functional elements. Each edge is weighted and the non-negative integer weight (register count) is the number of registers (clocked memory) placed along the interconnection between two functional elements. These notions allowed the introduction of transformations useful for the design and optimization of synchronous systems: *retiming* and *slowdown*.

Two synchronous systems are *strongly equivalent* if they have the same stepwise behavior under all interpretations. *Retiming* a functional element in a synchronous system means shifting one layer of registers from one side of the functional element to the other. *Retiming equivalence* is obtained as the reflexive and transitive closure of this primitive retiming relation. *Slowdown* is defined as follows: for any system $G = (V, E, w)$ and any positive integer c, the *c-slow* system $cG = (V, E, w')$ is the one obtained by multiplying all the register counts in G by c. *Slowdown equivalence* is obtained as the symmetric and transitive closure of this primitive slowdown relation. *Strong retiming equivalence* is the join of two basic equivalence relations on synchronous systems, namely strong equivalence and retiming equivalence. *Slowdown retiming equivalence* is the join of retiming and slowdown equivalence. *Strong slowdown retiming equivalence* is the join of strong, slowdown and retiming equivalence. It is proved that both slowdown retiming and strong slowdown retiming equivalence of synchronous systems (schemes), are decidable.

According to [Leiserson and Saxe, 1983a, 1983b], synchronous systems S and S'

are *equivalent* if for every sufficiently old configuration c of S, there exists a configuration c' of S' such that when S started in configuration c and S' started in configuration c', the two systems exhibit the same behavior. It is proved that two synchronous systems (schemes) are *Leiserson* equivalent if and only if they are strong retiming equivalent.

The semantics of synchronous systems can be specified by algebraic structures called feedback theories. Synchronous schemes and feedback theories have been axiomatized equationally in [Bartha, 1987]. Here we extend the existing set of axioms in order to capture strong retiming equivalence.

One of the fundamental features of synchronous systems is *synchrony*, that is, the computation of the network is synchronized by a global (basic) clock. Other, slower clocks can be defined in terms of boolean-valued flows. In order to describe the behavior of schemes with multiple regular clocks, we extend the existing algebra of schemes to include multiclocked schemes, and study the general idea of Leiserson equivalence in the framework of this algebra.

Contents

List of Figures and Tables

Table Page

Acknowledgements

First of all, I would like to express my deep and sincere gratitude to my supervisor, Dr. Miklós Bartha for his willingness and determination to supervise my work for more than three years, for his encouragement, expertise, understanding and patience. It was he who led me through the beautiful world of theoretical computer science and taught me to express my ideas in an organized and precise language of mathematics.

I would like to thank the other members of my comittee, Dr. Krishnamurthy Vidyasankar, Dr. Paul Gillard and Dr. Anthony Middleton for their assistance and the services they provided.

I would also like to thank Dr. Nicolas Halbwachs from IMAG Research Lab, Grenoble, who generously provided the LUSTRE compiler and related tools.

The Deparment of Computer Science at Memorial University of Newfoundland has not only provided the warm and stimulative environment for my research but has also given me the opportunity to teach several undergraduate courses, which I appreciate very much.

I am grateful to Memorial University of Newfoundland Graduate School and the Department of Computer Science at Memorial University of Newfoundland for providing me with the financial assistance. I recognize that this research would not have been possible without that support.

1 Introduction

The increasing demands of speed and performance in modern signal and image processing applications necessitate a revolutionary super-computing technology. According to [Kung, 1988], sequential systems will be inadequate for future real-time processing systems and the additional computational capability available through VLSI concurrent processors will become a necessity. In most real-time digital signal processing applications, general-purpose parallel computers cannot offer satisfactory processing speed due to severe system overheads. Therefore, special-purpose array processors will become the only appealing alternative. *Synchronous systems* are such multiprocessor structures which provide a realistic model of computation capturing the concepts of pipelining, parallelism and interconnection. They are single purpose machines which directly implement as low-cost hardware devices a wide variety of algorithms, such as filtering, convolution, matrix operations, sorting etc.

The concept of a synchronous system [Leiserson and Saxe, 1983a] was derived from the concept of a *systolic system* [Kung and Leiserson, 1978] which has turned out to be one of the most attractive current concepts in massive parallel computing. In recent years many systolic systems have been designed, some of them manufactured; transformation methodologies for the design and optimization of systolic systems have been developed and yet the rigorous mathematical foundation of a theory of synchronous systems has been missing. Important *equivalence relations* of synchronous systems such as slowdown retiming and strong slowdown retiming still lack decision algorithms. Some of the fundamental concepts, like Leiserson's definition of *equivalency* of synchronous systems are still informal and operational.

A more sophisticated model of synchronous systems was introduced in [Bartha, 1987]. In that model the graph of a synchronous system becomes a flowchart scheme in the sense of [Elgot, 1975], with the only difference that all edges are weighted and reversed. For this reason, such graphs are called *synchronous schemes*. With that

approach it becomes possible to study synchronous systems in an exact algebraic (and/or category theoretical) framework, adopting the sophisticated techniques and constructions developed for flowchart schemes and iteration theories.

This thesis addresses the problems stated above, which, to the best of our knowledge, are unsolved so far. The thesis is organized as follows: in Chapter 2 we introduce the concepts of systolic arrays, synchronous systems, flowchart and synchronous schemes, and give a short summary of the most important definitions and results in the field. In Chapter 3 we define the slowdown retiming and strong slowdown retiming equivalence relations of synchronous systems and show that both relations are decidable. In Chapter 4 we compare Leiserson's definition of equivalency of synchronous systems with strong retiming equivalence of synchronous schemes, and show them to be identical. Chapter 5 deals with the equational axiomatization of synchronous schemes. The goal is to define the retiming identity, which together with the feedback theory identities captures the strong retiming equivalence of synchronous schemes.

Finally, in Chapter 6 we introduce the generalized algebra of multiclocked schemes which is intended to describe the behavior of synchronous schemes with multiple clocks motivated by the clock analysis of the Synchronous Dataflow Programming Language LUSTRE.

2 Preliminaries

2.1 Systolic Arrays

In [Kung and Leiserson, 1978] the authors proposed multiprocessor structures called systolic arrays (systems), which provide a realistic model of computation, capturing the concepts of pipelining, parallelism and interconnection. The goal was to design multiprocessor machines which have simple and regular communication paths, employ pipelining and can be implemented directly as low-cost hardware devices. Systolic systems are not general purpose computing machines. A systolic computing system is a subsystem that performs its computations on behalf of a *host* which can be viewed as a Turing-equivalent machine that provides input and receives output from the systolic system. Kung [1988] defined a systolic array as follows:

DEFINITION 2.1 A *systolic array* is a computing network possessing the following features:

- *Synchrony* The data are rythmically computed (timed by a global clock) and passed through the network.
- *Modularity and Regularity* The array consists of modular processing units with homogeneous interconnections. Moreover, the computing network can be extended indefinitely.
- *Spatial locality and temporal locality* The array manifests a locally communicative interconnection structure, i.e., spatial locality. There is at least one unit-time delay alloted so that signal transactions from one node to the next can be completed, i.e., temporal locality.
- *Pipelinability* ($O(n)$ execution-time speedup) The array exhibits a *linear rate pipelinability*, i.e., it should achieve an $O(n)$ speedup in terms of processing

rate, where n is the number of processing elements. The efficiency of the array is measured by the following:

$$speedup\ factor = \frac{T_S}{T_P}$$

where T_S is the processing time in a single processor, and T_P is the processing time in the array processor.

A systolic device is typically composed of many interconnected processors. Two processors that comminicate must have a data path between them and free global communication is disallowed. The farthest a datum can travel in unit time is from one processor to an adjacent processor(s). Figure 2.1 illustrates several mesh-connected network configurations.

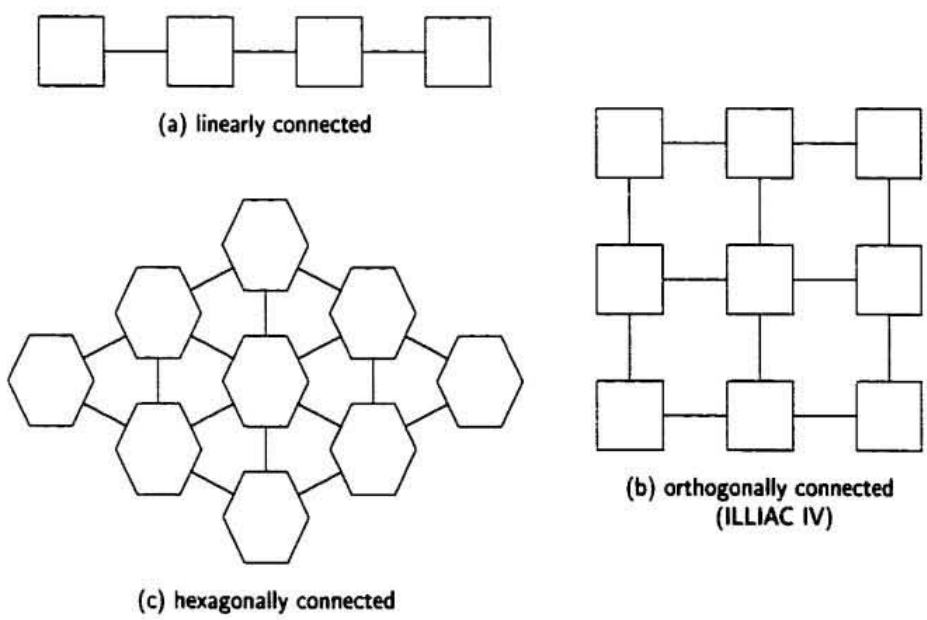


Figure 2.1: Mesh-connected systolic arrays.

Many algorithms such as filtering, convolution, matrix operations and sorting can be implemented as systolic arrays. The following example (from [Kung and Leiserson, 1978]) demonstrates matrix-vector multiplication in a linear systolic array.

EXAMPLE 2.1

Consider the problem of multiplying a matrix $\mathbf{A} = (a_{ij})$ with a vector $\mathbf{x} = (x_1, \ldots, x_n)^T$. The elements in the product $\mathbf{y} = (y_1, \ldots, y_n)^T$ can be computed by the following reccurences

$$\begin{aligned} y_i^1 &= 0, \\ y_i^{k+1} &= y_i^k + a_{ik}x_k, \\ y_i &= y_i^{n+1}. \end{aligned}$$

The single operation common to all the computations for matrix vector multiplication is the inner product step, $C = C + A \cdot B$. Processor which implements the inner product step has three registers R_A, R_B and R_C. Each register has two connections, one for input and one for output. Figure 2.2 shows the geometry for this processor.

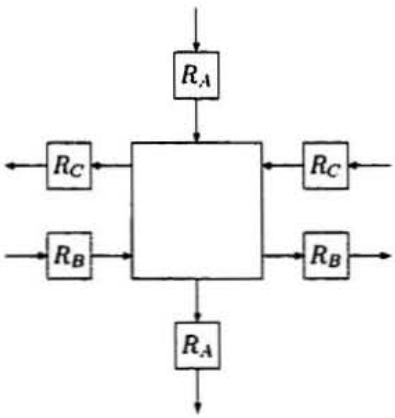


Figure 2.2: Geometry for the inner product step processor.

Suppose $\mathbf{A}$ is $n \times n$ band matrix with band width $w = p + q - 1$. (See Figure 2.3 for the case when $p = 2$ and $q = 3$.) Then the above reccurences can be evaluated by pipelining the x_i and y_i through a systolic array consisting of w linearly connected processors which compute the inner product step $y = y + A \cdot x$. The linearly connected systolic array for the band matrix-vector multiplication problem in Figure 2.3 has four inner product step processors. See Figure 2.4.

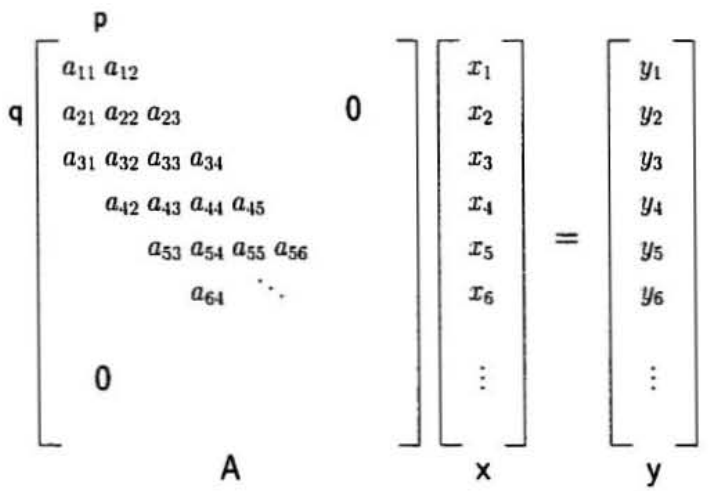


Figure 2.3: Multiplication of a vector by a band matrix with $p = 2$ and $q = 3$.

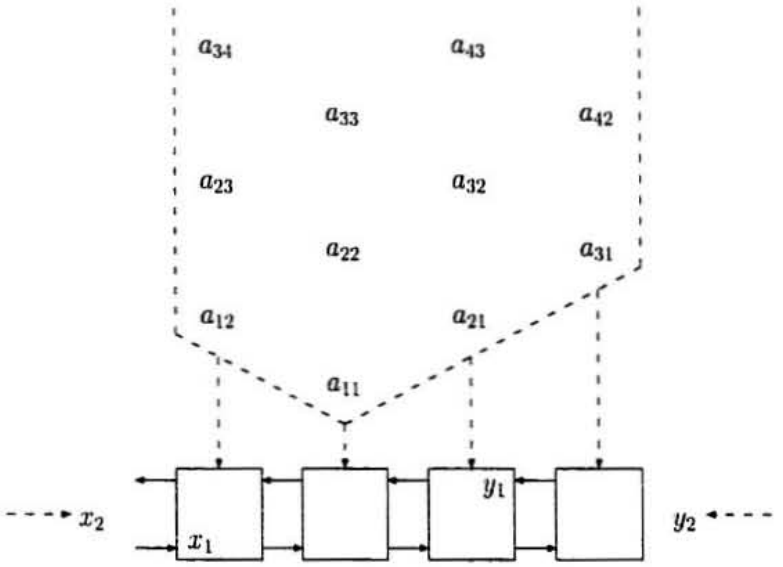


Figure 2.4: The linearly connected systolic array for the matrix vector multiplication problem in Figure 2.3.

The general scheme of computation can be viewed as follows. The y_i which are initially zero, are pumped to the left while x_i are pumped to the right and the a_{ij} are marching down. All the moves are synchronized.

Figure 2.5 illustrates the first seven pulsations of the systolic array. Observe that at any given time alternate processors are idle. Indeed, by coalescing pairs of adjacent processors, it is possible to use $w/2$ processors in the network for a general band matrix with band width w.

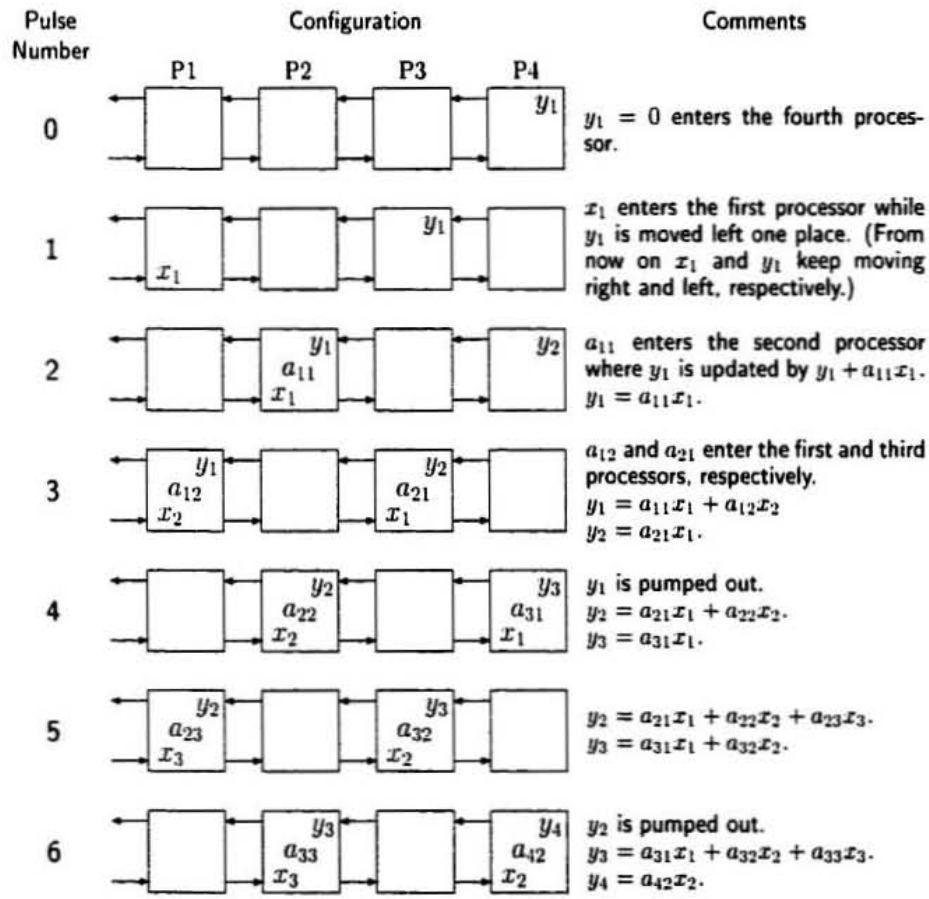


Figure 2.5: The first seven pulsations of the linear systolic array in Figure 2.4.

We now specify the operation of the systolic array more precisely. Assume that the processors are numbered by integers $1, 2, \ldots, w$ from the left end processor to the right end processor. Each processor has three registers R_A, R_B and R_C, which hold entries in A, x and y, respectively. Initially, all registers contain zeros.

Each pulsation of the systolic array consists of the following operations, but for odd numbered pulses only odd numbered processors are activated and for even numbered pulses only even numbered processors are activated.

- *Shift*

  1. R_A gets a new element in the band of matrix **A**.
  2. R_x gets the contents of register R_x from the left neighboring node. (The R_x in processor P1 gets a new component of **x**.
  3. R_y gets the contents of register R_y from the right neighboring node. (Processor P1 outputs its R_y contents and the R_y in processor w gets zero.)

- *Multiply and Add*

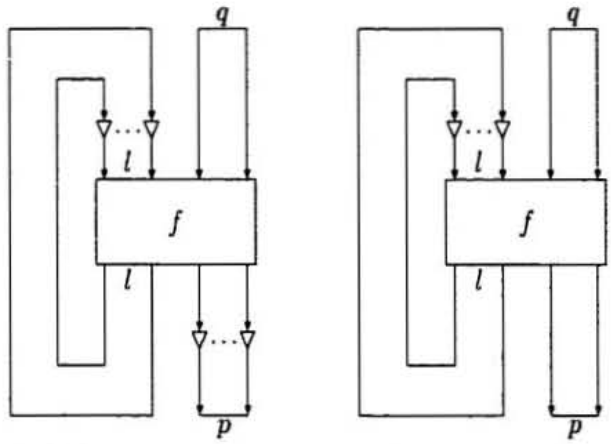
  $$R_y = R_y + R_A \cdot R_x$$

Using the inner product step processor the three shift operations in step 1 can be done simultaneously, and each pulsation of the systolic array takes a unit of time. Suppose the bandwidth of **A** is $w = p + q - 1$. It is readily seen that after w units of time the components of the product $\mathbf{y} = \mathbf{Ax}$ are pumped out from the left processor at the rate of one output every two units of time. Therefore, using the proposed systolic network all the n components of **y** can be computed in $2n + w$ time units, as compared to the $O(wn)$ time needed for a sequential algorithm on a uniprocessor computer.

2.2 The Structure of Systolic Arrays

Processors in a systolic system are composed of a constant number of *Moore automata*. Recall that a finite state Moore automaton is defined as a six-tuple $A = (S, l, q, p, \delta, \lambda)$, where S is a finite set, l, p, q are nonnegative integers; $\delta : S^{l+q} \rightarrow S^l$ is the state transition function, and $\lambda : S^{l+q} \rightarrow S^p$ is the output function. Considering A as an ordinary automaton, then S^l is the set of states, S^q and S^p are the input and

output alphabet, respectively. The standard graphical representation of A is given in Figure 2.6(a), where the triangles symbolize the l state components (registers), and $f = \langle \delta, \lambda \rangle : S^{l+q} \rightarrow S^{l+p}$ is the combinational logic. This type of automaton has the property that its outputs are dependent upon its state but not upon its inputs.



(a) Moore automaton (b) Mealy automaton

Figure 2.6: The difference between Moore and Mealy automata

In this mathematical model, *time* can be regarded as independent variable which takes on integer values and is a count of the *number* of clock cycles or state changes. The states $S^l(t+1)$ and outputs $S^p(t+1)$ of a Moore automaton at time $t+1$ are uniquely determined by its states $S^l(t)$ and its inputs $S^q(t)$ at time t by

$$\begin{aligned} S^l(t+1) &= \delta(S^l(t), S^q(t)) \\ S^p(t+1) &= \lambda(S^l(t), S^q(t)) \end{aligned}$$

A *Mealy automaton* is similarly defined as a six-tuple $A = (S, l, q, p, \delta, \lambda)$, where all is the same as in Moore automata except that the output at time t is dependent on input at time t, that is

$$\begin{aligned} S^l(t+1) &= \delta(S^l(t), S^q(t)) \\ S^p(t+1) &= \lambda(S^l(t), S^q(t+1)) \end{aligned}$$

The standard graphical representation of Mealy automaton is shown in Figure 2.6(b). In both automata the state is clocked through registers, but since the input signals are allowed to propagate through to the output unconstrained, a change in the signal on an input can affect the output without an intervening clock tick. When Mealy machines are connected in series, signals *ripple* through the combinational logic of several machines between clock ticks. If the signals feed back on themselves before being stopped by a register, they can latch or oscilate. Even if the problems associated with feedback have been precluded, the settling of combinational logic can make the clock period long in systems with rippling logic. *Systolic systems* contain only Moore automata, while *Semisystolic systems* may contain both Moore and Mealy automata. The exclusion of Mealy automata guarantees that the clock period does not grow with system size, and makes the *number* of clock ticks be a measure of time that is largely independent of system size.

A systolic system can be simply viewed as a set of interconnected Moore automata. The structure of such a system $S(n)$ is given by a *communication graph* $G = (V, E)$ of n interconnected automata where the vertices in V represent the automata and the edges in E represent represent interconnections between the automata. The weights of edges in systolic systems are strictly positive, while the weights of edges in semisystolic systems may be zero. An example of a semisystolic system and its communication graph is shown in Figure 2.7.

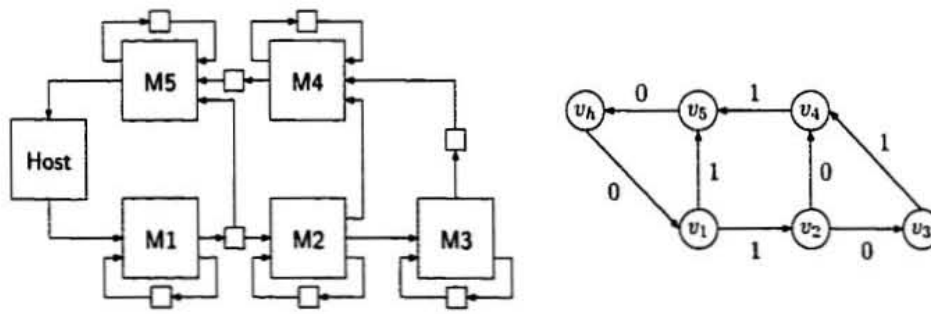


Figure 2.7: Semi-systolic array and its communication graph.

The automata operate synchronously by means of a common clock, and *time* in the system is measured as the number of clock cycles. All the automata in V are Moore automata (Moore and Mealy in the case of semisystolic system) with the exception of one called the *host* which can be viewed as a Turing equivalent machine that provides input to and receives output from the system. Based on the communication graph, the *neighborhood* of an automaton $v \in V$ is the set of automata with which it communicates:

$$N(v) = \{w \mid (v, w) \in E \ \ or \ \ (w, v) \in E\}.$$

For $S(n)$ to be systolic, it is further required that the Moore machines be small in the following sense. There must exist constants c_1, c_2, c_3 and c_4 such that for all n and all $v \in V - \{host\}$,

- $| S_v^i | \leq c_1$ The number of states of each Moore (Mealy) automaton is bounded,
- $| S_v^q | \leq c_2$ The number of input symbols is bounded
- $| S_v^p | \leq c_3$ The number of output symbols is bounded
- $|N(v)| \leq c_4$ The number of neighbors of each automaton is bounded, i.e., the communication graph has bounded degree.

The "smallness" conditions help ensure that the number of clock cycles is a good measure of time in the systolic model. A problem arises, however, when the time required to propagate a signal between machines becomes longer than the time required for the longest combinational-logic delay through a machine. The period of the clock must be at least as long as the longest propagation delay between machines, which means that the independence of the clock period from system size will not be realized for systems with long interconnections. Systolic arrays, which have only nearest-neighbor connections, are especially attractive for VLSI because propagation delay is insignificant.

2.3 Synchronous Systems

The systolic design methodology manages communication costs effectively because the only communication permitted during a clock cycle is between a processing element and its neighbors in the communication graph of the system. This constraint is in direct contrast with, for example, the propagation of a carry signal which *ripples* down the length of an adder. Such combinational rippling and global control such as *broadcasting* are forbidden in systolic designs. Global communication is more easily described in terms of rippling logic. In a systolic system the effect of broadcasting must be achieved by multiple local communications. The primary reason for introducing the concept of a *Synchronous System* was the design issue. In [Leiserson and Saxe, 1983a] the authors demonstrated how a synchronous system can be designed with rippling logic, and then converted through *Systolic Conversion Theorem* to a systolic implementation that is functionally equivalent to the original system - the principal difference being the shorter clock period of the systolic implementation.

A synchronous system can be modelled as a finite, rooted, edge-weighted, directed multigraph $G = \langle V, E, v_h, w \rangle$. The vertices V of the graph model the functional elements (combinational logic) of the system. Every functional element is assumed to have a fixed primitive operation associated with it. These operations are designed to manipulate some simple data in the common algebraic sense. Each vertex $v \in V$ is weighted with its numerical propagation delay $d(v)$. A distinguished root vertex v_h, called the *host*, is included to represent the interface with the external world, and it is given zero propagation delay. The directed edges E of the graph model interconnections between functional elements. Each edge e in E is a triple of the form (u, v, w), where u and v are (possibly identical) vertices of G connecting an output of some functional element to an input of some functional element and $w = w(e)$ is the nonnegative integer *weight* of the edge. The weight (register count) is the number of registers (clocked memory) along the interconnection between the two functional elements. If e is an edge in the graph that goes from vertex u to vertex v, we shall

use the notation $u \xrightarrow{e} v$. For a graph G, we shall view a path p in G as a sequence of vertices and edges. If a path p starts at vertex u and ends at a vertex v, we use the notation $u \xrightarrow{p} v$. A *simple path* contains no vertex twice, and therefore the number of vertices exceeds the number of edges by exactly one. We extend the register count function w in a natural way from single edges to arbitrary paths. For any path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, we define the *path weight* as the sum of the weights of the edges of the path:

$$w_p = \sum_{i=0}^{k-1} w(e_i)$$

Similarly, propagation delay function d can be extended to simple paths. For any simple path $p = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, we define the *path delay* as the sum of the delays of the vertices of the path:

$$d_p = \sum_{i=0}^{k} d(v_i)$$

In order that a graph $G = \langle V, E, v_h, w \rangle$ has well-defined physical meaning as a circuit, we place the following restriction on propagation delays $d(v)$ and register counts $w(e)$:

D. *The propagation delay $d(v)$ is nonnegative for each vertex $v \in V$.*

W. *In any directed cycle of G, there is some edge with strictly positive register count.*

We define a *synchronous system* as a system that satisfies conditions **D** and **W**. The reason for including condition **W** is that whenever an edge e between two vertices u and v has zero weight, a signal entering vertex u can ripple unhindered through vertex u and subsequently through vertex v. If the rippling can feed back upon itself, problems of asynchronous latching, oscilation and race conditions can arise. By prohibiting zero-weight cycles, condition **W** prevents these problems from occuring, provided that the system clock runs slowly enough to allow the outputs of all the functional elements to settle between each two consecutive ticks. The following definitions are adopted from [Leiserson and Saxe, 1983a, 1983b].

DEFINITION 2.2 A synchronous system is *systolic* if for each edge (u, v, w) in the comminication graph of S, the weight w is strictly greater than zero.

DEFINITION 2.3 A *configuration* of a system is some assignment of values to all its registers. With each clock tick, the system maps the current configuration into a new configuration. If the weight of an edge happens to be zero, no register impedes the propagation of a signal along the corresponding interconnection.

DEFINITION 2.4 Let c be a configuration of a synchronous system S and let c' be a configuration of a synchronous system S'. The system S started in configuration c has the same *behavior* as the system S' started in configuration c' if for any sequence of inputs to the system from the host, the two systems produce the same sequence of outputs to the host.

DEFINITION 2.5 Let S and S' be synchronous systems. Suppose that for every sufficiently old configuration c of S, there exists a configuration c' of S' such that when S is started in configuration c and S' is started in configuration c', the two systems exhibit the same behavior. Then system S' can *simulate* S. If two synchronous systems can simulate each other, then they are equivalent.

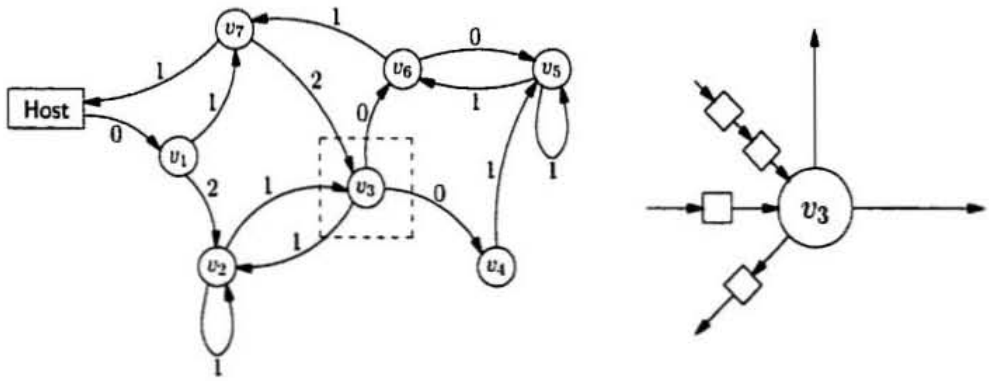
Two synchronous systems are *strongly equivalent*, or, in other words, have the same *strong behavior* if they have the same behavior under all interpretations. The interpretation of a functional element labeled with σ from some alphabet Σ, with p input channels and q output channels, is a mapping $\phi_\sigma : D^q \to D^p$, where the set D consists of certain data elements.

DEFINITION 2.6 For any synchronous circuit G, the minimum feasible *clock period* $\Phi(G)$ is the maximum amount of propagation delay through which any signal must ripple between clock ticks. Condition **W** guarantees that the clock period is well defined by the equation $\Phi(G) = \max\{d(p) \mid w(p) = 0\}$.

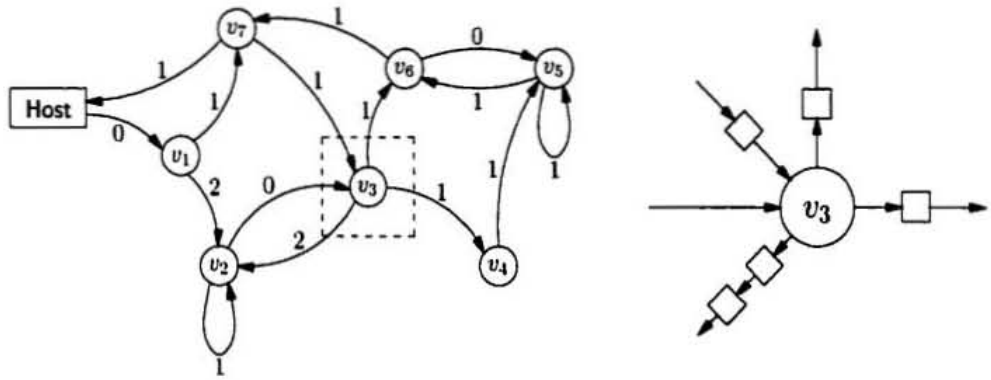
These notions allowed the introduction of transformations useful for the design and the optimization of synchronous systems: *retiming* and *slowdown*.

Retiming transformations can alter the computations carried out in one clock cycle of the system by relocating registers, that is, shifting one layer of registers from one side of a functional element to the other. Two systems are retiming equivalent if they can be joined by a sequence of such primitive retiming steps. Retiming is important technique which can be used to optimize clocked circuits by relocating registers so as to reduce combinational rippling.

Consider the communication graph G_1 in Figure 2.8(a). Suppose, for instance, that each vertex has a propagation delay of 3 esec. Then the clock period of S_1 must be at least 9 esec - the time for a signal to propagate from v_3 through v_6 to v_5.



(a) The communication graph G_1 of a synchronous system S_1.



(b) The communication graph G_2 of a system S_2, which is equivalent to the system S_1 from Figure 2.8(a), as viewed from the host. Internally, the two systems differ in that vertex v_3 lags by one clock tick in S_1 with respect to S_2.

Figure 2.8: Retiming transformation.

Retiming the vertex v_3 in G_1, that is, decreasing the number of registers by one on all incoming edges and increasing the number of registers by one on all outgoing edges, results in a communication graph G_2 of a synchronous system S_2 in Figure 2.8(b) which is, intuitively, equivalent to S_1 but with a shorter clock period - 6 esec.

Formally, retiming transformation is defined as follows: let S be a synchronous system, $V(G)$ the set of vertices of the underlying graph G and R a function from $V(G)$ into the set of all integers. We say that R is a *legal retiming vector* if for every edge (u, v, w) in G the value $w + R(v) - R(u)$ is nonnegative and $R(host) = 0$. Applying R to S simply means replacing the weight $w(e)$ of each edge $e : u \rightarrow v$ by $w'(e) = w(e) + R(v) - R(u)$.

In our example, the legal retiming vector R which takes S_1 into S_2 is:

$$R(host, v_1, v_2, v_3, v_4, v_5, v_6, v_7) = \{0, 0, 0, -1, 0, 0, 0, 0\}.$$

The impact of retiming on the behavior of synchronous systems is expressed by the so called *Retiming Lemma* in [Leiserson and Saxe, 1983a]:

LEMMA 2.1 (*Retiming Lemma*) Let S be a synchronous system with communication graph G, and let R be a function that maps each vertex v of G to an integer and the host to zero. Suppose that for every edge (u, v, w) in G the value $w + R(v) - R(u)$ is nonnegative. Let S' be the system obtained by replacing every edge $e = (u, v, w)$ in S with $e' = (u, v, w + R(v) - R(u))$. Then the systems S and S' are equivalent.

Slowdown is defined as follows: for any circuit $G = (V, E, w)$ and any positive integer c, the *c-slow* circuit $cG = (V, E, w')$ is the circuit obtained by multiplying all the register counts in G by c. That is, $w'(e) = cw(e)$ for every edge $e \in E$. All the data flow in cG is slowed down by a factor of c, so that cG performs the same computations as G, but takes c times as many clock ticks and communicates with the host only on every c^{th} clock tick. In fact, cG acts as a set of c independent versions of G, communicating with the host in round-robin fashion. For example, the 2-slow circuit S_3 of S_1 is shown in Figure 2.9.

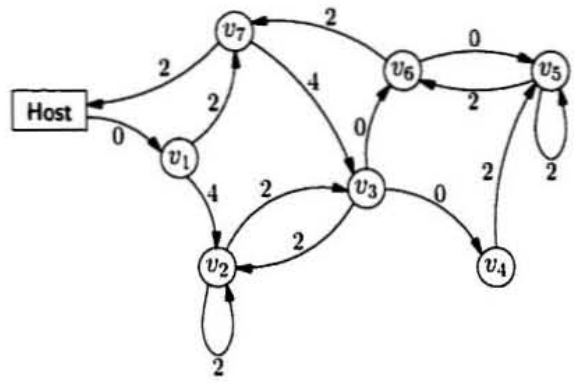


Figure 2.9: Slowdown transformation. The communication graph $G_3 = 2G_1$ of a system S_3 obtained by multiplying all the register counts in G_1 from Figure 2.8(a) by 2. All the data flow in S_3 is slowed down by a factor of 2, so that S_3 performs the same computations as S_1, but takes 2 times as many clock ticks and communicates with the host only on every second tick.

The impact of slowdown on the behavior of synchronous systems is the following: the main adventage of c-slow circuits is that they can be retimed to have shorter clock periods than any retimed version of the original. For many applications, throughput is the issue, and multiple, interleaved streams of computation can be effectively utilized. A c-slow circuit that is systolic offers maximum throughput. Another interesting observation is that not every synchronous system can be retimed to get an equivalent systolic system. According to *Systolic Conversion Theorem* [Leiserson and Saxe, 1983a], synchronous system S with communication graph G can be retimed to systolic system S' if the *constraint graph* $G - 1$, which is the graph obtained from G by replacing every edge (u, v, w) with $(u, v, w - 1)$ has no cycles of negative weight. However, for any synchronous system that cannot be directly retimed to get a systolic system, there might be a slowdown transformation such that, after this transformation is applied, one gets a synchronous system that can be retimed to get an equivalent systolic system. It can be proved that such slowdown transformation is possible only if the underlying automaton is Moore automaton. Retiming vector $R(v)$ is defined for every vertex v as the weight of the *shortest path* from v to *host* in $G-1$. Consider the constraint graph $G_1 - 1$ in Figure 2.10 of a synchronous system S_1. Since $G_1 - 1$

contains a cycle $host \rightarrow v_1 \rightarrow v_7 \rightarrow host$ of negative weight, S_1 cannot be directly retimed to get an equivalent systolic system.

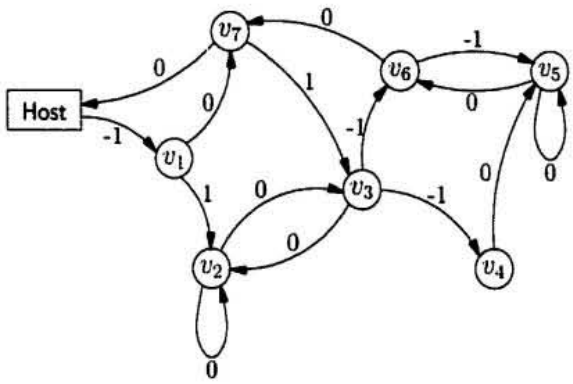


Figure 2.10: The constraint graph $G_1 - 1$ of a synchronous system S_1 in Figure 2.8(a).

On the other hand, the constraint graph $G_3 - 1 = 2G_1 - 1$ does not have cycles of negative weight. Consequently, there exists a legal retiming vector which transforms synchronous system S_3 into systolic system S_4 in Figure 2.11:

$$R(host, v_1, v_2, v_3, v_4, v_5, v_6, v_7) = \{0, 2, 2, 1, 4, 3, 2, 1\}.$$

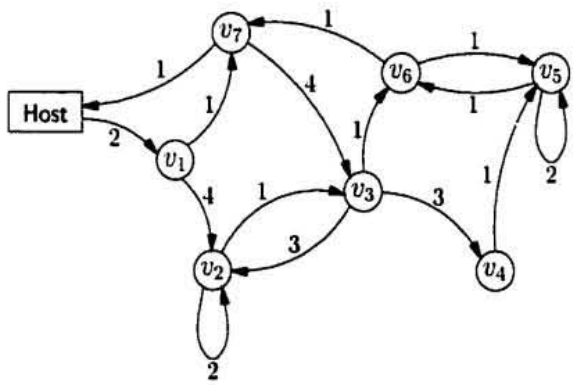


Figure 2.11: The systolic system S_4 obtained from the 2-slow synchronous system S_3 by retiming.

2.4 Flowchart and Synchronous Schemes

Two major objections must be made about the model of synchronous system presented in the previous subsection:

[1] According to Definition 2.1, a synchronous system is an *infinite* edge-weighted directed multigraph represented by its finite aproximations that must be regular in a certain sense. Therefore the single finite graph G should be called a finite system only, or rather a *scheme*.

[2] The multigraph representation of a synchronous scheme is inadequate in the sense that it does not relate the two endpoints of a given edge to designated labelled input-output "ports" of the corresponding vertices. This question is clearly important, because i/o ports of the functional elements (processors) behave differently in general. Also, it is advantageous to replace the host by a fixed (finite) number of *input-output channels* as distinguished vertices, thus avoiding the unnecessary constraint that those cycles closed only by the host should contain an edge with positive weight.

These two criticisms suggest reconsidering synchronous systems in the framework of Elgot's [1975] well-known model of monadic computations (flowchart algorithms). This standpoint motivated the definition of *synchronous flowchart schemes* in [Bartha, 1987] or simply *synchronous scheme*. Since synchronous schemes are defined in terms of flowchart schemes we introduce the fundamental definitions and properties of flowchart schemes that will play an important role in the sequel.

DEFINITION 2.7 A *signature* or *ranked alphabet* is a set Σ, whose elements are called *operation symbols*, together with a mapping $ar : \Sigma \to \mathbb{N}$, called the *arity function*, assigning to each operation symbol a natural number, called its finite *arity*. If the operation symbols are grouped into subsets according to their arity: $\Sigma_n = \{\sigma \in \Sigma \mid ar(\sigma) = n\}$, then the signature Σ is uniquely determined by the family $(\Sigma_n \mid n \in \mathbb{N})$.

A *realization* of an n-ary operation symbol in a set A is an n-ary operation on A. Given a signature Σ, a Σ-*algebra* A is a pair $\mathfrak{A} = (A, \Sigma^A)$ consisting of a set A, called the *carrier* of A, and a family $\Sigma^A = (\sigma^A \mid \sigma \in \Sigma)$ of realizations σ^A of operation symbols σ from Σ.

DEFINITION 2.8 A Σ-*flowchart scheme* (FΣ-scheme) F is a finite directed graph augmented by the following data.

(1) A subset $X \subseteq F$ of vertices of outdegree 0. The elements of X are called *exits* of F.

(2) A labeling function, by which every nonexit vertex v is assigned a symbol $\sigma \in \Sigma$ in such a way that the rank of σ equals the outdegree of v.

(3) For each vertex u, a linear order of the set of edges leading out of u. By the notation $u \to_i v$ we shall indicate that the target of the i^{th} edge leaving u is vertex v.

(4) A begin function, which maps some finite set B into the set of vertices of F. The begin function specifies a set of marked entries into F.

For simplicity, the marking set B above will be identified with the set $[n] = \{1, \ldots, n\}$. Similarly, the exit vertices will be labeled by the numbers in $[p] = \{1, \ldots, p\}$. An n-entry and p-exit FΣ-scheme F is denoted $F : n \to p$. If $F : n \to p$ and $G : p \to q$ are FΣ-schemes, then one can form their composite $F \cdot G : n \to q$ by identifying the exits of F with the entries of G in a natural way, assuming that F and G are disjoint graphs. This kind of composition gives rise to a category with all nonnegative integers as objects and with all FΣ-schemes as morphisms. The category obtained in this way is known as the horizontal structure of flowchart schemes.

The vertical structure of FΣ-schemes [Bloom and Ésik, 1993] is the category $\mathbf{Fl}_\Sigma$ constructed as the coproduct (disjoint union) of the categories $\mathbf{Fl}_\Sigma(n, p)$, $n, p \in \mathbb{N}$ defined below.

- For each pair $(n,p) \in \mathbb{N} \times \mathbb{N}$, $\mathbf{Fl}_\Sigma(n,p)$ has as objects all FΣ-schemes $n \to p$.
- A morphism $F \to F'$ between FΣ-schemes $F, F' : n \to p$ is a mapping α from the set of vertices of F into that of F' which preserves:
  1. the sequence of entry and exit vertices;
  2. the labeling of the boxes;
  3. the edges in the sense that if $u \to_i v$ holds in F, then $\alpha(u) \to_i \alpha(v)$ will hold in F'.
- Composition of morphisms is defined in $\mathbf{Fl}_\Sigma(n,p)$ as that of mappings, and the identity morphisms are the identity maps.

Sometimes it is useful to consider an FΣ-scheme $F : n \to p$ as a separate partial algebraic structure over the set of vertices of F [Grätzer, 1968]. In this structure there are n constants, namely the entry vertices of F. Furthermore, for each $\sigma \in \Sigma_q$ there are q unary operations $\langle \sigma, i \rangle, i \in [q]$ if $q \geq 1$, one unary operation $\langle \sigma, 0 \rangle$ if $q = 0$. If $i \geq 1$, then the operation $\langle \sigma, i \rangle$ is defined on vertex u of F if and only if u is labeled by σ, and in that case $\langle \sigma, i \rangle(u)$ is the unique vertex v for which $u \to_i v$. The operation $\langle \sigma, 0 \rangle$ is interpreted as if there was a loop around each vertex labeled by the constant symbol σ, i.e. $\langle \sigma, 0 \rangle$ is an appropriate restriction of the identity function. No operation is defined on the set of exit vertices.

A strong congruence relation of F (as a partial algebra) by which the exit vertices form singleton groups is called a *scheme congruence* of F. Clearly, every scheme morphism $\alpha : F \to F'$ induces a scheme congruence θ on F. By the homomorphism theorem, if α is onto then $F/\theta \cong F'$, where the isomorphism and the factor scheme F/θ have their usual algebraic meaning [Grätzer, 1968]. In the sequel we shall not distinguish between isomorphic FΣ-schemes.

Let v be a vertex of an FΣ-scheme $F : n \to p$. Starting from v, F can be *unfolded* into a possibly infinite Σ-tree $T(F,v)$. Recall from [Bloom and Ésik, 1993] that an

infinite Σ-tree has all of its nodes labeled by the symbols of Σ in such a way that the number of descendents of each node u is equal to the arity of the symbol labeling u. The branches of the tree $T(F, v)$ correspond to maximal walks in F starting from v, where a maximal walk is one that ends at a vertex of outdegree zero or it proceeds to the infinity. The walk is allowed to return to itself arbitrary many times. The nodes of $T(F, v)$, being copies of the vertices of F, are labeled either by the symbols of Σ or by the variable symbols $x_1, \ldots, x_p$ chosen from the fixed variable set $X = \{x_1, \ldots, x_n, \ldots\}$, in the case of exit vertices.

EXAMPLE 2.2 Consider the flowchart scheme in Figure 2.12.

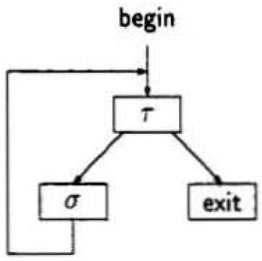


Figure 2.12: Flowchart scheme.

Syntactical description of F can be given by the equation $y = \tau(\sigma(y), x_1)$. The term on the right-hand side has the tree representation shown in Figure 2.13.

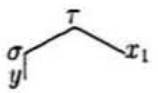


Figure 2.13: The tree representation of a term.

Solving the equation means replacing y by $\tau(\sigma(y), x_1)$ as many times as possible. The process results in the infinite labelled tree shown in Figure 2.14.

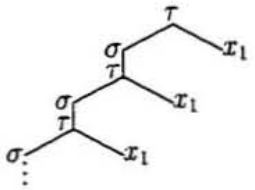


Figure 2.14: Unfolding the flowchart scheme.

For two vertices u, v of F, we say that u and v have the same strong behavior if $T(F,u) = T(F,v)$. Unfolding F starting from each entry vertex simultaneously yields an n-tuple of trees, which is called the *strong behavior* of F, denoted $T(F)$. By definition, if θ is a scheme congruence of F and $u \equiv v(\theta)$, then u and v have the same strong behavior. Consequently, if $\alpha : F \to F'$ is a morphism in $\mathbf{Fl}_\Sigma$, then $T(F) = T(F')$.

An FΣ-scheme is called *accessible* if every nonexit vertex of F can be reached from at least one entry vertex by a directed path. In the algebraic setting F is accessible if, with the exception of the exit vertices, F is generated by its constants. For an accessible FΣ-scheme F, define the equivalence μ_F on the set of vertices of F in the following way:

$$u \equiv v(\mu_F) \quad \text{if} \quad T(F,u) = T(F,v).$$

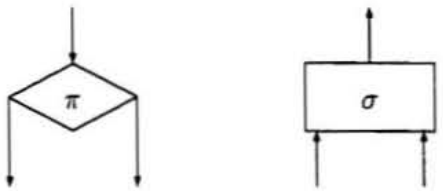
Obviously, μ_F is a scheme congruence, and it is the largest among all the scheme congruences of F. The scheme F/μ_F is therefore called *minimal*.

Let G be a graph and denote by $V(G)$ the set of vertices of G. A subset $S \subseteq V(G)$ is *strongly connected* if for every $u, v \in S$ there exists a directed path in G from u to v going through vertices of S only. A *strong component* of G is a strongly connected subset S which is maximal in the sense that if S' is strongly connected and $S \subseteq S'$, then $S = S'$. An n-entry FΣ-scheme F is *tree-reducible* if F is accessible and the graph obtained from F by deleting its exit vertices and contracting each of its strong components into a single vertex consists of n disjoint trees. Every accessible FΣ-scheme F can be unfolded into a tree-reducible scheme by finite means. To this end, it is sufficient to unfold the partial order of the strong components of F with its exit vertices deleted into a set of disjoint trees. The resulting tree-reducible FΣ-scheme will be denoted by $utr(F)$. The unfolding determines a morphism $utr(F) \to F$ in the category $\mathbf{Fl}_\Sigma$.

DEFINITION 2.9 A *synchronous scheme* S (SΣ-scheme for short) consists of a finite underlying FΣ-scheme, denoted $fl(S)$, and a weight function by which every edge of

S is assigned a nonnegative integer. We assume that the direction of an edge e in S is the opposite of the direction of the same edge in $fl(S)$.

Let us point out the semantical differences between flowchart and synchronous schemes. A flowchart scheme of sort $p \to q$ is interpreted as a flowchart algorithm called monadic computation [Elgot, 1975] with p entries and q exits. Accordingly, the flow of information in the flowchart scheme follows the direction of the arrow between p and q. For, example the scheme $\varepsilon : 2 \to 1$ should be interpreted as a join of two different paths in the flowchart. In a logical circuit, however, the meaning of ε is a branch; thus, in this case the information flows in the opposite direction. Reasoning from the point of view of category theory, the difference is the following. Concerning flowchart schemes, the object n in the theory T is treated as the nth *copower* of the object 1 ($n = \sum_{i=1}^{n} 1$), while in the case of synchronous schemes n would rather be the nth *power* of 1 ($n = \prod_{i=1}^{n} 1$), as in the original definition of algebraic theories in [Lawvere, 1963]. See also Figure 2.15. However, if we followed the product formalism, then the sort of a mapping $[p] \to [q]$ would confusingly become $q \to p$. Therefore, we rather adhere to the coproduct formalism and express the product-like (functional) semantics only by designing our schemes in an upside-down fashion.



(a) Flowchart scheme $F : D \to 2D$ (b) Synchronous scheme $S : D^2 \to D$

Figure 2.15: The difference between flowchart and synchronous schemes.

The category $\mathbf{Syn}_\Sigma$ of SΣ-schemes consists of the following. The objects are all accessible SΣ-schemes. A morphism $S \to S'$ in $\mathbf{Syn}_\Sigma$ is a morphism $fl(S) \to fl(S')$ in $\mathbf{Fl}_\Sigma$ that preserves the weight of the edges. Accordingly, a scheme congruence of S is one of $fl(S)$ that is compatible with the weight function.

Categories $\mathbf{TFl}_\Sigma$ and $\mathbf{TSyn}_\Sigma$ are full subcategories of $\mathbf{Fl}_\Sigma$ and $\mathbf{Syn}_\Sigma$ respectively, determined by the subset of tree-reducible schemes.

We define the signature Σ_∇ as $\Sigma \cup \{\nabla\}$, where ∇ is a unary operation symbol (register). With any SΣ-scheme S we then associate the FΣ_∇-scheme $fl_\nabla(S)$, which is obtained from $fl(S)$ by replacing every edge e in it by a chain of n ∇-labeled vertices, where n is the weight of e. As in the case of FΣ-schemes, we include the infinite unfoldings of SΣ-schemes in $\mathbf{Syn}_\Sigma$. Obvious details of this procedure are omitted. See Figure 2.16.

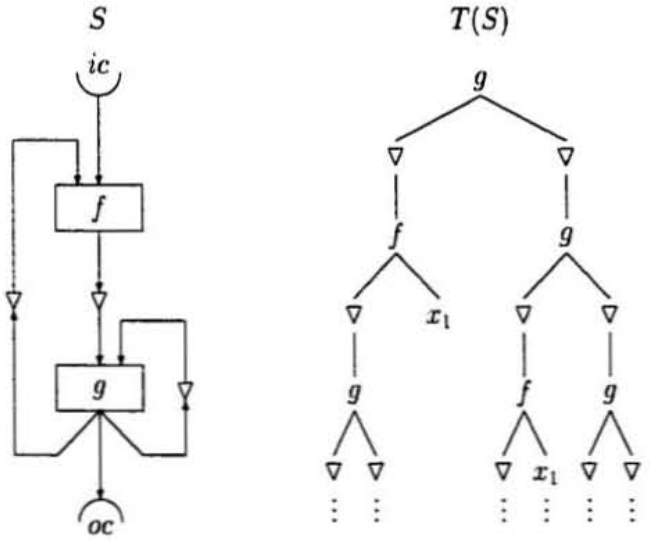


Figure 2.16: Synchronous scheme S and its (infinite) unfolding tree $T(S)$.

The importance of the concept of *tree unfolding* is that it captures the strong behavior of synchronous (flowchart) schemes. Two schemes can be syntactically different and yet exhibit the same strong behavior as shown in Figure 2.17.
Transformations of retiming and slowdown are defined for synchronous schemes in the same way as for synchronous systems.

If R is a legal retiming vector for S and S' is the scheme obtained by applying R to S, then we shall write $R : S \to S'$. Retiming count vectors thus define a category on the set of SΣ-schemes as objects. The composition of two arrows R and R' is $R + R'$ and the identities are the zero vectors $\vec{0}$.

If c is a positive integer such that $S' = cS$, i.e., S' is obtained from S by c-slowdown, then we shall write $c : S \to S'$. Slowdown transformations also define a category on the set of SΣ-schemes as objects. The composition of two arrows c_1 and c_2 is c_1c_2 and the identities are designated by $c = 1$.

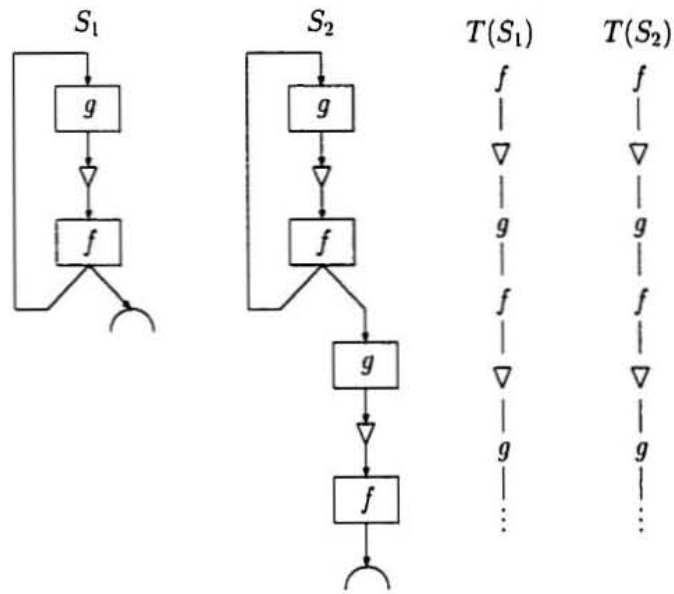


Figure 2.17: Schemes S_1 and S_2 while different represent the same computational process. They can be unfolded into the same tree $T(S_1) = T(S_2)$.

The following Definition and Lemma are adopted from [Bartha, 1994].

DEFINITION 2.10 The relation of *strong retiming equivalence* on the set Syn_Σ is the smallest equivalence relation containing $\longrightarrow_s$ and $\longrightarrow_r$, where $\longrightarrow_s$ denotes the binary relation induced by reduction (unfolding) and $\longrightarrow_r$ denotes the binary relation induced by retiming transformation. Strong retiming equivalence is denoted by $\sim$.

LEMMA 2.11 $\sim = \longleftarrow_s \circ \sim_r \circ \longrightarrow_s$

FACT: The relation of strong retiming equivalence is decidable for synchronous schemes [Bartha, 1994].

3 Equivalence Relations of Synchronous Schemes and their Decision Problems

In this section we introduce the equivalences of slowdown retiming and strong slowdown retiming as the join of slowdown equivalence and retiming equivalence, and the join of these two plus strong equivalence, respectively, on the set of SΣ-schemes. Concerning slowdown, $\longrightarrow_{sl}$ will stand for the partial order induced on Syn_Σ by slowdown constants. We shall use the preorders defined by the categories $\mathbf{Fl}_\Sigma$ and $\mathbf{Syn}_\Sigma$ as simple binary relations over the sets Fl_Σ and Syn_Σ of all finite accessible FΣ-schemes and SΣ-schemes, respectively. In both cases, this preorder will be denoted by $\longrightarrow_s$. Concerning retiming, $\sim_r$ will stand for the equivalence relation induced on Syn_Σ by legal retiming count vectors.

3.1 Slowdown Retiming Equivalence

DEFINITION 3.1.1 The relation of *slowdown retiming equivalence* on the set Syn_Σ is the smallest equivalence relation containing $\longrightarrow_{sl}$ and $\sim_r$.

Slowdown retiming equivalence of synchronous schemes will be denoted by $\sim_{SR}$. The relations $\sim_{sl} = (\longrightarrow_{sl} \cup \longleftarrow_{sl})^*$ (symmetric and transitive closure) and $\sim_r$ are called *slowdown equivalence* and *retiming equivalence*, respectively. In order to decide the relation $\sim_{SR}$ we are going to prove the following equation:

$$\sim_{SR} = \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} . \tag{1}$$

Equation (1) says that if two SΣ-schemes S_1 and S_2 are slowdown retiming equivalent, then they can be slowed down into appropriate schemes S_1' and S_2' that are already retiming equivalent.

LEMMA 3.1.2 $\sim_r \circ \longrightarrow_{sl} \subseteq \longrightarrow_{sl} \circ \sim_r$

PROOF. Let S, P and Q be SΣ-schemes such that $S \sim_r P$ and $P \longrightarrow_{sl} Q$. Then there exist a legal retiming vector $R : S \to P$ and a positive integer c such that Q is the c-slow scheme cP obtained by multiplying all the register counts in P by c. Scheme S can be slowed down by multiplying all the register counts in S by c, i.e., $S \longrightarrow_{sl} cS = P_1$. Define a legal retiming vector R_1 as $R_1(v) = cR(v)$ for all vertices v in P_1. We claim that R_1 takes P_1 to Q. Indeed, for any edge $u \xrightarrow{e} v$ in S, the weight $w_R(e)$ of the corresponding edge in P after retiming R is defined by the equation

$$w_R(e) = w(e) + R(v) - R(u)$$

The slowdown $c : P \to Q$ transforms $w_R(e)$ into

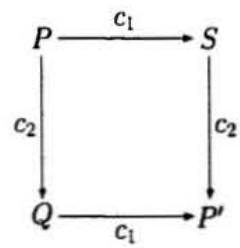
$$cw_R(e) = cw(e) + cR(v) - cR(u)$$

in Q. On the other hand, for any edge $u \xrightarrow{e} v$ in S, slowdown transforms $w(e)$ into $cw(e)$ in P_1, and retiming R_1 takes this number to

$$\begin{aligned} cw_{R_1}(e) &= cw(e) + R_1(v) - R_1(u) \\ &= cw(e) + cR(v) - cR(u) \quad \blacksquare \end{aligned}$$

LEMMA 3.1.3 $\longleftarrow_{sl} \circ \longrightarrow_{sl} \subseteq \longrightarrow_{sl} \circ \longleftarrow_{sl}$

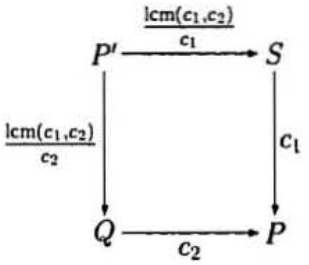
PROOF. Let S, P and Q be SΣ-schemes such that $P \longrightarrow_{sl} S$ and $P \longrightarrow_{sl} Q$. Then there exist positive integers c_1 and c_2 such that S and Q are c-slow schemes c_1P and c_2P obtained by multiplying all the register counts in S by c_1 and c_2 respectively. Then the following diagram commutes



since $S = c_1P$, $Q = c_2P$ and $c_2S = c_2c_1P = c_1c_2P = c_1Q = P'$. $\blacksquare$

LEMMA 3.1.4 $\longrightarrow_{sl} \circ \longleftarrow_{sl} \subseteq \longleftarrow_{sl} \circ \longrightarrow_{sl}$

PROOF. Let S, P and Q be SΣ-schemes such that $S \longrightarrow_{sl} P$ and $Q \longrightarrow_{sl} P$. Then there exist positive integers c_1 and c_2 such that $P = c_1 S = c_2 Q$. Let e be an edge in P. Then $w_P(e) = c_1 w_S(e) = c_2 w_Q(e)$ and $\frac{c_1}{\mathrm{lcm}(c_1,c_2)} w_S(e) = \frac{c_2}{\mathrm{lcm}(c_1,c_2)} w_Q(e)$, where $\mathrm{lcm}(c_1, c_2)$ is the *least common multiple* of c_1 and c_2. Therefore, there exists scheme P' such that $S = \frac{\mathrm{lcm}(c_1,c_2)}{c_1} P'$ and $Q = \frac{\mathrm{lcm}(c_1,c_2)}{c_2} P'$, i.e., the following diagram commutes:



since $P = c_1 S = c_1 \frac{\mathrm{lcm}(c_1,c_2)}{c_1} P' = c_2 \frac{\mathrm{lcm}(c_1,c_2)}{c_2} P' = c_2 Q = P$. ■

COROLLARY 3.1.5 $\sim_{sl} \;=\; \longleftarrow_{sl} \circ \longrightarrow_{sl} \;=\; \longrightarrow_{sl} \circ \longleftarrow_{sl}$

PROOF. Follows directly from Lemmas 3.1.3 and 3.1.4. ■

COROLLARY 3.1.6 $\sim_{SR} = \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl}$

PROOF. It is sufficient to prove that the relation $\rho = \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl}$ is transitive. We have

$$\begin{aligned}
\rho \circ \rho &= \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \\
&\subseteq \longrightarrow_{sl} \circ \sim_r \circ \longrightarrow_{sl} \circ \longleftarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \quad \text{(by Lemma 3.1.3)} \\
&\subseteq \longrightarrow_{sl} \circ \longrightarrow_{sl} \circ \sim_r \circ \sim_r \circ \longleftarrow_{sl} \circ \longleftarrow_{sl} \quad \text{(by Lemma 3.1.2)} \\
&= \rho \;\blacksquare
\end{aligned}$$

3.2 Decidability of Slowdown Retiming Equivalence

PROPOSITION 3.2.1 *The relations $\sim_{sl}$ and $\sim_r$ are decidable.*

PROOF. For any two SΣ-schemes S and S', $S \sim_{sl} S'$ if and only if there exists scheme P such that $S \longrightarrow_{sl} P$ and $S' \longrightarrow_{sl} P$, i.e. there exist positive integers c and c' such

that $cw(e) = c'w'(e')$ for all edges e in S and corresponding edges e' in S'. Therefore, in order to decide $\sim_{sl}$ it is sufficient to verify that the ratio $\frac{w(e)}{w'(e')}$ is the same for all corresponding edges e, e'.

As to the relation $\sim_r$, recall from [Murata, 1977] that a *fundamental circuit* of a directed graph is a cycle of the corresponding undirected graph. Let S and S' be two SΣ-schemes with weight functions w and w', and assume that S and S' share a common underlying graph G. For every fundamental circuit z of this graph, let us fix a cyclic order of the vertices of z in order to distinguish a positive and a negative direction of the edges belonging to z. For every edge $e \in z$, let $sign_z(e) = 1(-1)$ if the direction of e is positive (respectively, negative) with respect to the given order.

LEMMA 3.2.2 *We claim that S and S' are retiming equivalent, i.e. there exists a legal retiming count vector taking S to S' if and only if:*

(1) For every fundamental circuit z of G

$$\sum_{e \in z} sign_z(e) \cdot w(e) = \sum_{e \in z} sign_z(e) \cdot w'(e)$$

(2) All simple paths from an entry to an exit vertex have the same weight, where by *simple path* we mean an alternating sequence of vertices and edges $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$ in which no vertex is repeated.

By [Murata, 1977, Theorem 1], (1) is necessary and sufficient for the existence of a retiming vector R that satisfies the condition of being legal, except that $R(v)$ need not be zero for all exit vertices. Suppose that such an R exists, and let p be a simple path composed of vertices and edges $v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \cdots \xrightarrow{e_{k-1}} v_k$, where v_0 is an entry and v_k is an exit vertex. Then we have:

$$\begin{aligned} w'(p) &= \sum_{i=0}^{k-1} w'(e_i) = \sum_{i=0}^{k-1} (w(e_i) + R(v_{i+1}) - R(v_i)) \\ &= \sum_{i=0}^{k-1} w(e_i) + \sum_{i=0}^{k-1} (R(v_{i+1}) - R(v_i)) = w(p) + R(v_k) - R(v_0). \end{aligned}$$

Then $w(p) = w(p')$ if and only if $R(v_0) = R(v_k)$. Obviously, one of $R(v_0)$ and $R(v_k)$ can be chosen freely, so that the condition $R(v_0) = R(v_k)$ is equivalent to saying that assignment $R(v_0) = R(v_k) = 0$ is possible. ■

COROLLARY 3.2.3 *Let S and S' be synchronous schemes such that $S \sim_r S'$. Then for any directed cycle c in S and S', we have $w(c) = w'(c)$.*

PROOF. Follows immediately from Lemma 3.2.2 (2). ■

If S and S' are tree-reducible schemes, the relation $\sim_r$ is yet simpler to decide. Since every fundamental circuit must remain in some strong component, in condition (1), it suffices to check that every directed cycle of a common underlying graph G has the same total weight by the weight functions of S and S'. ■

THEOREM 3.2.4 *The relation of slowdown retiming equivalence is decidable for synchronous schemes.*

PROOF. Let G and G' be SΣ-schemes. By Corollary 3.1.6, G and G' are slowdown retiming equivalent if and only if there exist SΣ-schemes S and S' such that $G \longrightarrow_{sl} S$, $G' \longrightarrow_{sl} S'$ and $S \sim_r S'$. Since $S = c_1 G$ and $S' = c_2 G'$, we must have $c_1 G \sim_r c_2 G'$, i.e. there exists a legal retiming vector R such that, according to Lemma 3.2.2, all fundamental circuits z and simple entry-to-exit paths p have the same total weight. In other words, the ratios $\frac{w(z)}{w'(z)}$ and $\frac{w(p)}{w'(p)}$ must be the same, where w and w' are weight functions of G and G' respectively.

According to the argument above, one can decide the slowdown retiming equivalence of G and G' by the following algorithm.

Algorithm A.

Compute $w(z_i)$ and $w'(z_i)$ for every fundamental circuit $z_i, 1 \leq i \leq n$ and $w(p_j)$ and $w'(p_j)$ for every simple entry-to-exit path $p_j, 1 \leq j \leq m$ in G and G', respectively, and

check the ratios $\frac{w(z_i)}{w'(z_i)}$ and $\frac{w(p_j)}{w'(p_j)}$. Then, G and G' are slowdown retiming equivalent if and only if these ratios are the same for all $1 \leq i \leq n$ and $1 \leq j \leq m$. ■

Let us briefly discuss the complexity of Algorithm A. It is easy to see that the expensive part of Algorithm A is the finding of all fundamental circuits and entry to exit paths. Johnson's algorithm [Johnson, 1975] for finding all the elementary circuits of a directed graph has a time bound of $O(|(n+e)(c+1)|)$ on any graph with n vertices, e edges and c elementary ciruits. In order to find all entry-to-exit paths the Floyd-Warshall algorithm might be used. It is well known that the Floyd-Warshall algorithm runs in $O(|V^3|)$ time on any graph with vertex set V.

EXAMPLE 3.1 Consider the schemes in Fig. 3.1.

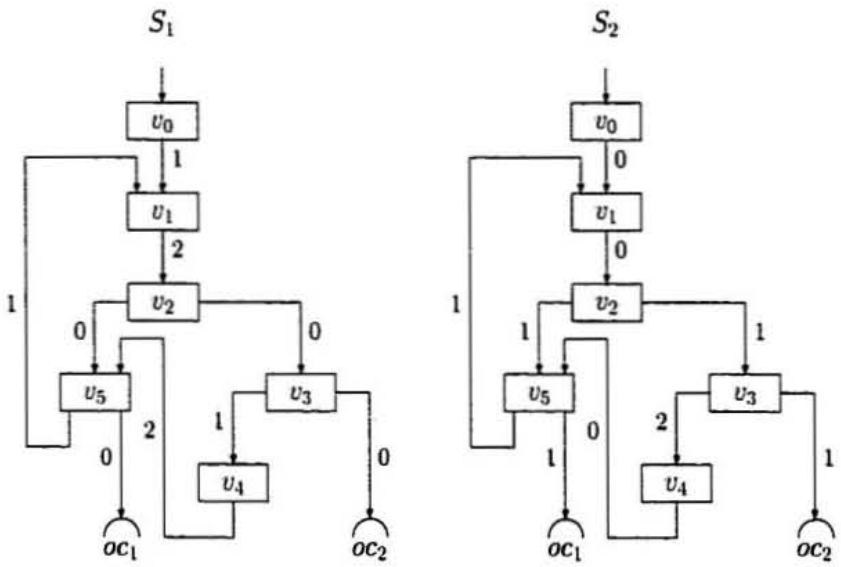


Figure 3.1.

Schemes S_1 and S_2 are not retiming equivalent since both directed cycles and both entry-to-exit paths have different total weights. In order to decide the slowdown retiming equivalence relation for the given schemes it suffices to solve the following system of linear equations:

$$
\begin{aligned}
c_1 w_1(z_1) &= c_2 w_2(z_1) \\
c_1 w_1(z_2) &= c_2 w_2(z_2) \\
c_1 w_1(p_1) &= c_2 w_2(p_1) \\
c_1 w_1(p_2) &= c_2 w_2(p_2)
\end{aligned}
$$

where z_1 is the directed cycle $v_1 \to v_2 \to v_5 \to v_1$; z_2 is the directed cycle $v_1 \to v_2 \to v_3 \to v_4 \to v_5 \to v_1$; p_1 is the entry-to-exit path $v_0 \to v_1 \to v_2 \to v_5 \to oc_1$ and p_2 is the entry-to-exit path $v_0 \to v_1 \to v_3 \to oc_2$ with $w_1(z_1) = 3$, $w_2(z_1) = 2$, $w_1(z_2) = 6$, $w_2(z_2) = 4$, $w_1(p_1) = 3$, $w_2(p_1) = 2$, $w_1(p_2) = 3$ and $w_2(p_2) = 2$. Since

$$
c_1 = c_2 \frac{w_2(z_1)}{w_1(z_1)} = c_2 \frac{w_2(z_2)}{w_1(z_2)} = c_2 \frac{w_2(p_1)}{w_1(p_1)} = c_2 \frac{w_2(p_2)}{w_1(p_2)} = c_2 \frac{2}{3}
$$

the solution exists, $c_1 = 2$, $c_2 = 3$. By multiplying all the registers counts in S_1 and S_2 by c_1 and c_2, respectively, one gets schemes S_1' and S_2'. See Figure 3.2.

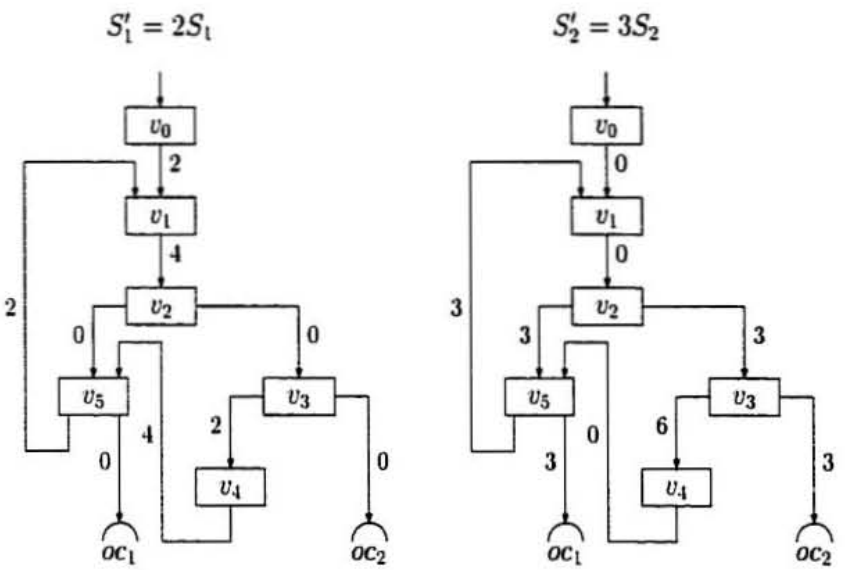


Figure 3.2.

It is trivial to check that schemes S_1' and S_2' are retiming equivalent. Consequently, original schemes S_1 and S_2 are slowdown retiming equivalent.

3.3 Strong Slowdown Retiming Equivalence

DEFINITION 3.3.1 The relation of *strong slowdown retiming equivalence* on the set Syn_Σ is the smallest equivalence relation containing $\longrightarrow_{sl}$, $\longrightarrow_s$ and $\sim_r$.

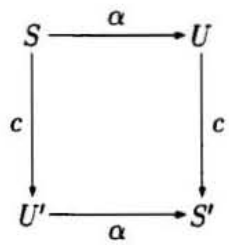
Strong slowdown retiming equivalence of synchronous schemes will be denoted by $\sim_{SSR}$. The relation $\sim_s = (\longrightarrow_s \cup \longleftarrow_s)^*$ (symmetric and transitive closure) is called *strong equivalence*. For the definitions of *slowdown* and *retiming* equivalence see Definition 3.1.1. In order to decide the relation $\sim_{SSR}$ we are going to prove the following equation

$$\sim_{SSR} = \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s = \longleftarrow_s \circ \sim_{SR} \circ \longrightarrow_s \qquad (2)$$

Equation (2) says that if two accessible SΣ-schemes S_1 and S_2 are strong slowdown retiming equivalent, then they can be unfolded into appropriate schemes S_1' and S_2' that are already slowdown retiming equivalent.

LEMMA 3.3.2 $\longrightarrow_s \circ \longrightarrow_{sl} \subseteq \longrightarrow_{sl} \circ \longrightarrow_s$

PROOF. Let S, U and S' be SΣ-schemes such that $S \longrightarrow_s U$ and $U \longrightarrow_{sl} S'$. Then there exist a scheme morphism $\alpha : S \to U$ and a positive integer c such that S' is a c-slow scheme cU obtained by multiplying all the register counts in U by c. Then the following diagram commutes



LEMMA 3.3.3 $\sim_r \circ \longleftarrow_s \subseteq \longleftarrow_s \circ \sim_r$ [Bartha, 1994]

PROOF. Let S, S' and U be SΣ-schemes such that $S \sim_r U$ and $S' \longrightarrow_s U$. Then there exists a legal retiming count vector $R : S \to U$ and a scheme morphism $\alpha : S' \to U$.

Since $fl(U) = fl(S)$, S can be unfolded into a scheme U' for which $fl(U') = fl(S')$ and $\alpha : U' \to S$. For every vertex v of U', define $R'(v) = R(\alpha(v))$. It is now easy to check that the retiming R' takes U' to S'. ■

COROLLARY 3.3.4 $\sim_{SSR} = \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s$

PROOF. It is sufficient to prove that the relation $\rho = \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s$ is transitive. Observe that $\longrightarrow_s \circ \longleftarrow_s \subseteq \longleftarrow_s \circ \longrightarrow_s$ and $\longrightarrow_{sl} \circ \longleftarrow_s \subseteq \longleftarrow_s \circ \longrightarrow_{sl}$, because the category $\mathbf{Syn}_\Sigma$ has all pullbacks and pushouts [MacLane, 1971]. By applying Lemmas 3.3.2, 3.3.3, 3.1.3 and 3.1.2 we have:

$$
\begin{aligned}
\rho \circ \rho &= \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \circ \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longleftarrow_s \circ \longrightarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_s \circ \longleftarrow_{sl} \circ \longrightarrow_{sl} \circ \longrightarrow_s \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longrightarrow_{sl} \circ \longleftarrow_s \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_{sl} \circ \sim_r \circ \longrightarrow_s \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longrightarrow_{sl} \circ \longleftarrow_s \circ \sim_r \circ \longrightarrow_{sl} \circ \longleftarrow_{sl} \circ \sim_r \circ \longrightarrow_s \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longrightarrow_{sl} \circ \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \circ \longleftarrow_{sl} \circ \longrightarrow_s \\
&\subseteq \longleftarrow_s \circ \longleftarrow_s \circ \longrightarrow_{sl} \circ \longrightarrow_{sl} \circ \sim_r \circ \sim_r \circ \longleftarrow_{sl} \circ \longleftarrow_{sl} \circ \longrightarrow_s \circ \longrightarrow_s \\
&= \rho \ \blacksquare
\end{aligned}
$$

Repeating the proof of Corollary 3.3.4 working in the subset $TSyn_\Sigma$ of tree reducible SΣ-schemes, we obtain the following result.

COROLLARY 3.3.5 $\sim_{SSR} = utr \circ \longleftarrow_s \circ \longrightarrow_{sl} \circ \sim_r \circ \longleftarrow_{sl} \circ \longrightarrow_s \circ utr^{-1}$, *where the relation* $\longrightarrow_s$ *is restricted to the subset of tree-reducible schemes.*

3.4 Decidability of Strong Slowdown Retiming Equivalence

PROPOSITION 3.4.1 *The relations* $\sim_{sl}, \sim_s$ *and* $\sim_r$ *are decidable.*

PROOF. See Proposition 3.2.1 and Proposition 5.2 [Bartha, 1994]. ■

THEOREM 3.4.2 *Let F and F' be tree-reducible* SΣ*-schemes such that $F \sim_{SR} F'$, and assume that θ is a tree-preserving scheme congruence of F. Then $F/\theta \sim_{SR} F'/\theta$, provided that θ is a scheme congruence of F', too.*

PROOF. Since slowdown transformations preserve the congruence θ, Theorem 6.2.5 [Bartha, 1994] directly applies. ■

THEOREM 3.4.3 *The relation of strong slowdown retiming equivalence is decidable for synchronous schemes.*

PROOF. Let G and G' be strong slowdown retiming equivalent SΣ-schemes. By Corollary 3.3.5 there exist some schemes F and F' such that $F \longrightarrow_s utr(G)$, $F' \longrightarrow_s utr(G')$ and $F \sim_{SR} F'$. See Figure 3.3a. Thus in the category $\mathbf{TSyn}_\Sigma$ there are morphisms $F \to utr(G)$ and $F' \to utr(G')$, which determine two morphisms $fl(F) \to fl(utr(G))$ and $fl(F') \to fl(utr(G'))$ in $\mathbf{TFl}_\Sigma$. Let ϕ and ϕ' denote the scheme congruences of $fl(F)$ induced by these two morphisms.

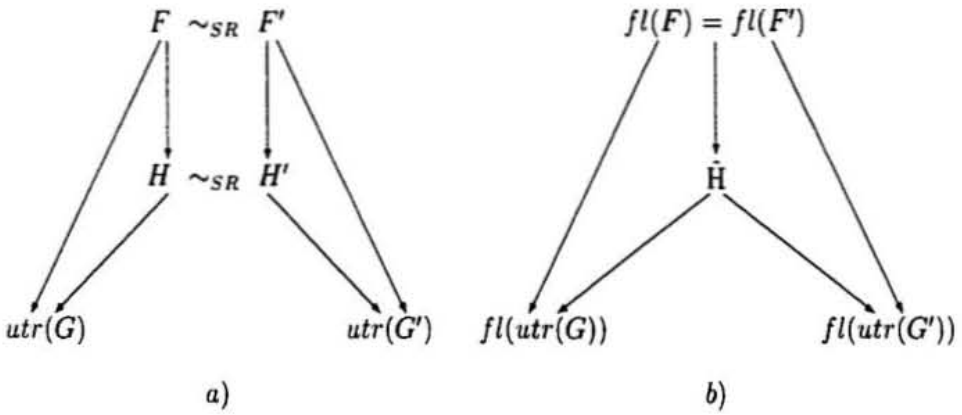


Figure 3.3: The proof of Theorem 3.4.3 in a diagram.

Now construct the product of $fl(utr(G))$ and $fl(utr(G'))$ as a tree-reducible FΣ-scheme $\hat{\mathrm{H}}$. Then there exists a morphism $fl(F) \to \hat{\mathrm{H}}$ that makes the diagram of Figure 3.3b commute. For the scheme congruence θ induced by this morphism, we thus have $\theta \subseteq \phi$ and $\theta \subseteq \phi'$. On the other hand, ϕ and ϕ' are also SΣ-scheme congruences of F

and F', respectively, for which $F/\phi = utr(G)$ and $F'/\phi' = utr(G')$. It follows that θ, too, is an SΣ-scheme congruence of both F and F'. Theorem 3.4.2 then implies that

$$H = F/\theta \sim_{SR} F'/\theta = H'.$$

According to the argument above, one can decide the slowdown retiming equivalence of G and G' by the following algorithm.

Algorithm B.

Step 1. See if $fl(G) \sim_s fl(G')$. If not, then G and G' are not strong slowdown retiming equivalent. Otherwise go to Step 2.

Step 2. Construct schemes H and H', which are the unfoldings of G and G' to the extent determined by the product of $fl(utr(G))$ and $fl(utr(G'))$ in $\mathbf{TFl}_\Sigma$, and test whether H and H' are slowdown retiming equivalent.

The schemes G and G' are strong slowdown retiming equivalent if and only if the result of the test performed in Step 2 of Algorithm B is positive. ■

EXAMPLE 3.2 Consider the schemes in Fig. 3.4.

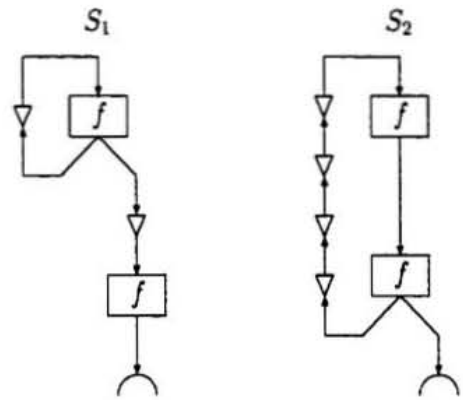


Figure 3.4

Since $fl(S_1) \sim_s fl(S_2)$ we construct the product $\hat{H}$ of $fl(utr(S_1))$ and $fl(utr(S_2))$ as follows:

[1] The set of vertices $V(\hat{H}) = V(S_1) \times V(S_2)$ with the restriction that $(u, v) \in V(\hat{H})$ if and only if $T(S_1, u) = T(S_2, v)$, for some $u \in V(S_1)$ and $v \in V(S_2)$. This restriction implies that u and v have the same label in S_1 and S_2 respectively. This common label becomes the label of vertex (u, v) in the product scheme.

[2] The entry (exit) vertices of $\hat{H}$ are those pairs consisting of an entry (exit) vertex in S_1 and the corresponding vertex in S_2.

[3] $(u_1, v_1) \rightarrow_i (u_2, v_2)$ in $\hat{H}$ if $u_1 \rightarrow_i u_2$ in S_1 and $v_1 \rightarrow_i v_2$ in S_2.

[4] Make the scheme $\hat{H}$ accessible by deleting non-accessible vertices.

Observe that $S_1 = utr(S_1)$ and $S_2 = utr(S_2)$. We have

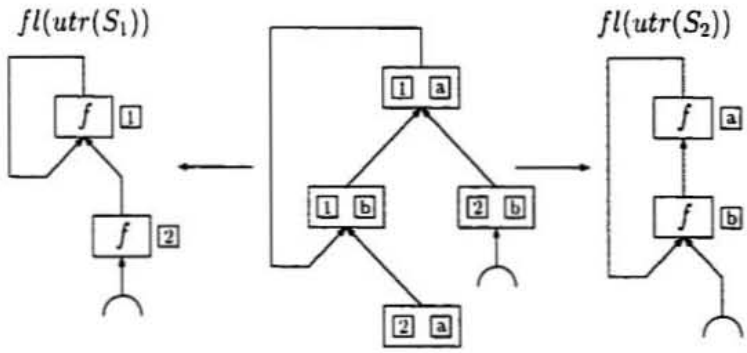


Figure 3.5: Construction of product of $fl(utr(S_1))$ and $fl(utr(S_2))$.

That is

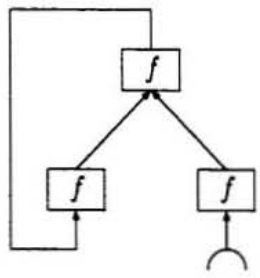


Figure 3.6: Scheme $\hat{H}$ as a product of $fl(utr(S_1))$ and $fl(utr(S_2))$.

Now we construct schemes H_1 and H_2, which are the unfoldings of S_1 and S_2 to the extent determined by the product scheme $\hat{H}$.

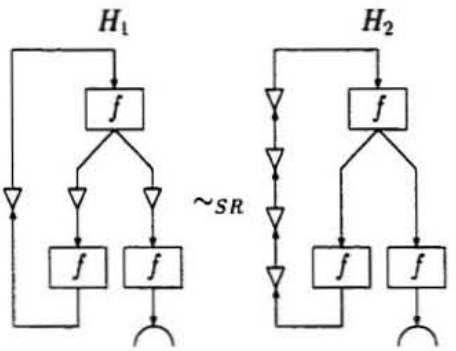


Figure 3.7: Schemes H_1 and H_2 are slowdown retiming equivalent.

It is trivial to verify that the slowdown constants are $c_1 = 2$ and $c_2 = 1$. Hence

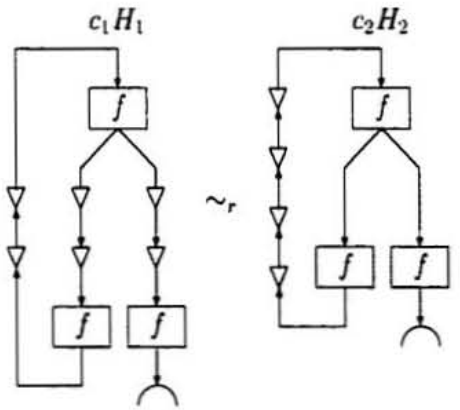


Figure 3.8: Schemes c_1H_1 and c_2H_2 are retiming equivalent.

Therefore, schemes S_1 and S_2 are strong slowdown retiming equivalent.

Let us briefly discuss the complexity of Algorithm B. The product scheme $\hat{H}$ can be constructed in $O(|V^2|)$. In order to construct schemes H_1 and H_2 one has to insert ∇ nodes into the product scheme $\hat{H}$ with reversed flow. This can be done starting from the exit vertices of H_1 and H_2 and following the unfolding trees $T(S_1)$ and $T(S_2)$ in $O(|V|)$. At this point one has to decide whether or not H_1 and H_2 are slowdown retiming equivalent. Algorithm A from Section 3.2, whose complexity is $O(|V^3|)$ applies.

4 Leiserson's Equivalence vs. Strong Retiming Equivalence

In this section the object of study is the relationship between Leiserson's (intuitive) definition of *equivalency of synchronous systems* (Definition 2.5) and *strong retmining equivalence of synchronous schemes* introduced in [Bartha, 1994].

We assume that the initial contents of the registers associated with the weights is $\perp$ (undefined datum).

EXAMPLE 4.1 Synchronous systems in Figure 4.1 are equivalent in the sense of Leiserson, that is S_1 and S_2 can simulate each other.

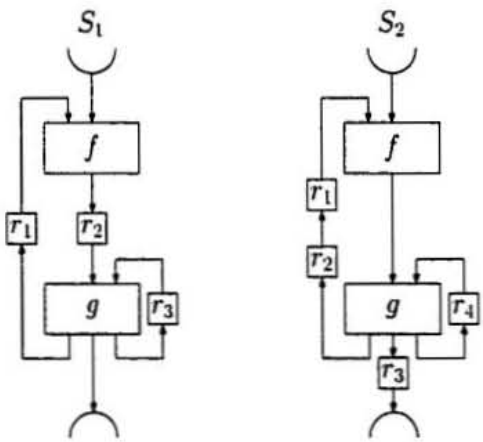


Figure 4.1.

The first three pulsations of the synchronous scheme S_1 are:

$r_1 = r_2 = r_3 = \perp$

Input: x_1

Output: $g(\perp, \perp)$

$r_1 = r_3 = g(\perp, \perp)$, $r_2 = f(\perp, x_1)$

Input: x_2

Output: $g(f(\perp, x_1), g(\perp, \perp))$

$r_1 = r_3 = g(f(\perp, x_1), g(\perp, \perp))$, $r_2 = f(g(\perp, \perp), x_2)$

Input: x_3

Output: $g(f(g(\perp, \perp), x_2), g(f(\perp, x_1), g(\perp, \perp)))$

$r_1 = r_3 = g(f(g(\perp, \perp), x_2), g(f(\perp, x_1), g(\perp, \perp)))$, $r_2 = f(g(f(\perp, x_1), g(\perp, \perp)), x_3)$

The first three pulsations of the synchronous scheme S_2 are:

$r_1 = r_2 = r_3 = r_4 = \perp$

Input: x_1

Output: $\perp$

$r_2 = r_3 = r_4 = g(f(\perp, x_1), \perp)$, $r_1 = \perp$

Input: x_2

Output: $g(f(\perp, x_1), \perp)$

$r_2 = r_3 = r_4 = g(f(\perp, x_2), g(f(\perp, x_1), \perp))$, $r_1 = g(f(\perp, x_1), \perp)$

Input: x_3

Output: $g(f(\perp, x_2), g(f(\perp, x_1), \perp))$

$r_2 = r_3 = r_4 = g(f(g(f(\perp, x_1), \perp), x_3), g(f(\perp, x_2), g(f(\perp, x_1), \perp)))$,

$r_1 = g(f(\perp, x_2), g(f(\perp, x_1), \perp))$

To demonstrate that S_2 can simulate S_1, let S_1 proceed one cycle from its initial configuration, then set $r_1 = g(\perp, \perp)$ and $r_2 = r_3 = r_4 = g(f(\perp, x_1), g(\perp, \perp))$ in S_2. From then on, for any sequence of inputs $x_2, x_3, \ldots$ scheme S_2 exhibits the same behavior as scheme S_1.

Similarly, after the first cycle of S_2, define $r_1 = \perp$, $r_2 = f(\perp, x_1)$ and $r_3 = \perp$ in S_1. Then, for any sequence of inputs $x_2, x_3, \ldots$ scheme S_1 will exhibit the same behavior as scheme S_2.

Let $Y = \{y_1, \ldots, y_n, \ldots\}$ be a fixed set of variables. For a ranked alphabet Σ, T_Σ will denote the set of finite Σ-trees. If S is any set of variable symbols then $T_\Sigma(S)$ denotes the set of Σ-trees over S, that is $T_\Sigma(S) = T_{\Sigma(S)}$, where $\Sigma(S)$ is the ranked alphabet obtained from Σ by adding all the elements of S as variables of rank 0 to it.

DEFINITION 4.3 A *finite state top-down tree transducer* M [Engelfriet, 1975] is a quintuple $(\Sigma, \Delta, Q, Q_d, R)$, where

- Σ is a ranked alphabet (of *input symbols*),
- Δ is a ranked alphabet (of *output symbols*),
- Q is a finite set of states, such that $Q \cap (\Sigma \cup \Delta) = \emptyset$,
- Q_d is a subset of Q (of *designated initial states*), and
- R is a finite set (of *rules*) such that $R \subseteq (Q \times \Sigma) \times T_\Delta(Q \times Y)$.

A rule of R will be written in the form $\alpha \longrightarrow p$, where $\alpha = (q, \sigma)$ with $q \in Q$, $\sigma \in \Sigma_n$ and $p \in T_\Delta(Q \times Y)$. In this rule, however, only the variables $y_1, \ldots, y_n$ are allowed to occur at the leaves of tree p. To emphasize this restriction, the above rule will rather be specified as

$$q\sigma(y_1, \ldots, y_n) \longrightarrow p(q_1 y_1, \ldots, q_n y_n) \tag{1}$$

Intuitively, the transducer works as follows. It starts processing an input tree $t \in T_\Sigma$ at its root in any of the designated initial states. Processing a node v labelled by $\sigma \in \Sigma_n$ is carried out by first finding a rule of the form (1), then replacing v by the tree p and continue processing the n subtrees under v in states $q_1, \ldots, q_n$, attaching them to the leaves of p labelled by $q_1 y_1, \ldots, q_n y_n$, respectively. Note that the rules are allowed to be nondeterministic. The relation $R \subseteq T_\Sigma \times T_\Delta$ induced by M will be denoted by $\mathfrak{R}(M)$ [Engelfriet, 1975], i.e., two trees t_1 and t_2 are related with respect to $\mathfrak{R}(M)$ if M maps t_1 into t_2.

In our transducers we shall allow the input tree to be infinite, which makes the processing of the tree also infinite, but still well-defined. Moreover, we shall augment

the input alphabet Σ by the variable symbols $X = \{x_1, \ldots, x_n, \ldots\}$ of rank 0.

DEFINITION 4.4 For a fixed $n \in \mathbb{N}$, the finite state top-down transducer $\mathcal{T}_n$ is defined by the following data

Input and output alphabet are the same: Σ_∇

$Q = [-n, n]$;

$Q_d = \{0\}$;

R is the set of rules defined below

(1) $i\sigma(y_1, \ldots, y_k) \to \nabla^l(\sigma((i+l)y_1, \ldots, (i+l)y_k))$ for $\sigma \in (\Sigma_\nabla)_k$, $l \geq 0$, $i+l \leq n$

(2) $i\nabla(y_1) \to (i-1)y_1$

(3) $0x_i \to x_i$, for $i \in \mathbb{N}$.

EXAMPLE 4.2 The finite state top-down tree transducer $\mathcal{T}_1$ can translate the tree $\nabla h(\nabla f \nabla x_1, \nabla g \nabla f \nabla x_2)$ into $\nabla\nabla h(f\nabla x_1, g\nabla f\nabla x_2)$ as follows (see also Figure 4.2):

$$
\begin{aligned}
0\nabla h(\nabla f\nabla x_1, \nabla g\nabla f\nabla x_2) &\Longrightarrow \nabla 0h(\nabla f\nabla x_1 \nabla g\nabla f\nabla x_2) && \text{rule (1)}\\
&\Longrightarrow \nabla\nabla 1h(\nabla f\nabla x_1, \nabla g\nabla f\nabla x_2) && \text{rule (1)}\\
&\Longrightarrow \nabla\nabla h(1\nabla f\nabla x_1, 1\nabla g\nabla f\nabla x_2) && \text{rule (1)}\\
&\Longrightarrow \nabla\nabla h(0f\nabla x_1, 0g\nabla f\nabla x_2) && \text{rules (2),(2)}\\
&\Longrightarrow \nabla\nabla h(f0\nabla x_1, g0\nabla f\nabla x_2) && \text{rules (1),(1)}\\
&\Longrightarrow \nabla\nabla h(f\nabla 0x_1, g\nabla 0f\nabla x_2) && \text{rules (1),(1)}\\
&\Longrightarrow \nabla\nabla h(f\nabla x_1, g\nabla f0\nabla x_2) && \text{rules (3),(1)}\\
&\Longrightarrow \nabla\nabla h(f\nabla x_1, g\nabla f\nabla 0x_2) && \text{rule (1)}\\
&\Longrightarrow \nabla\nabla h(f\nabla x_1, g\nabla f\nabla x_2) && \text{rule (3)}
\end{aligned}
$$

If $X = \{x_1, \ldots, x_n, \ldots\}$ is a set of variable symbols, then $T_\Sigma^\infty(X)$ denotes the set of (infinite) Σ-trees over X. An infinite tree $t \in T_\Sigma^\infty(X)$ is called *regular* if it has a finite number of different subtrees. Obviously, t is regular if and only if it can be obtained as the unfolding of an appropriate $F\Sigma$-scheme F, i.e. $t = T(F)$.

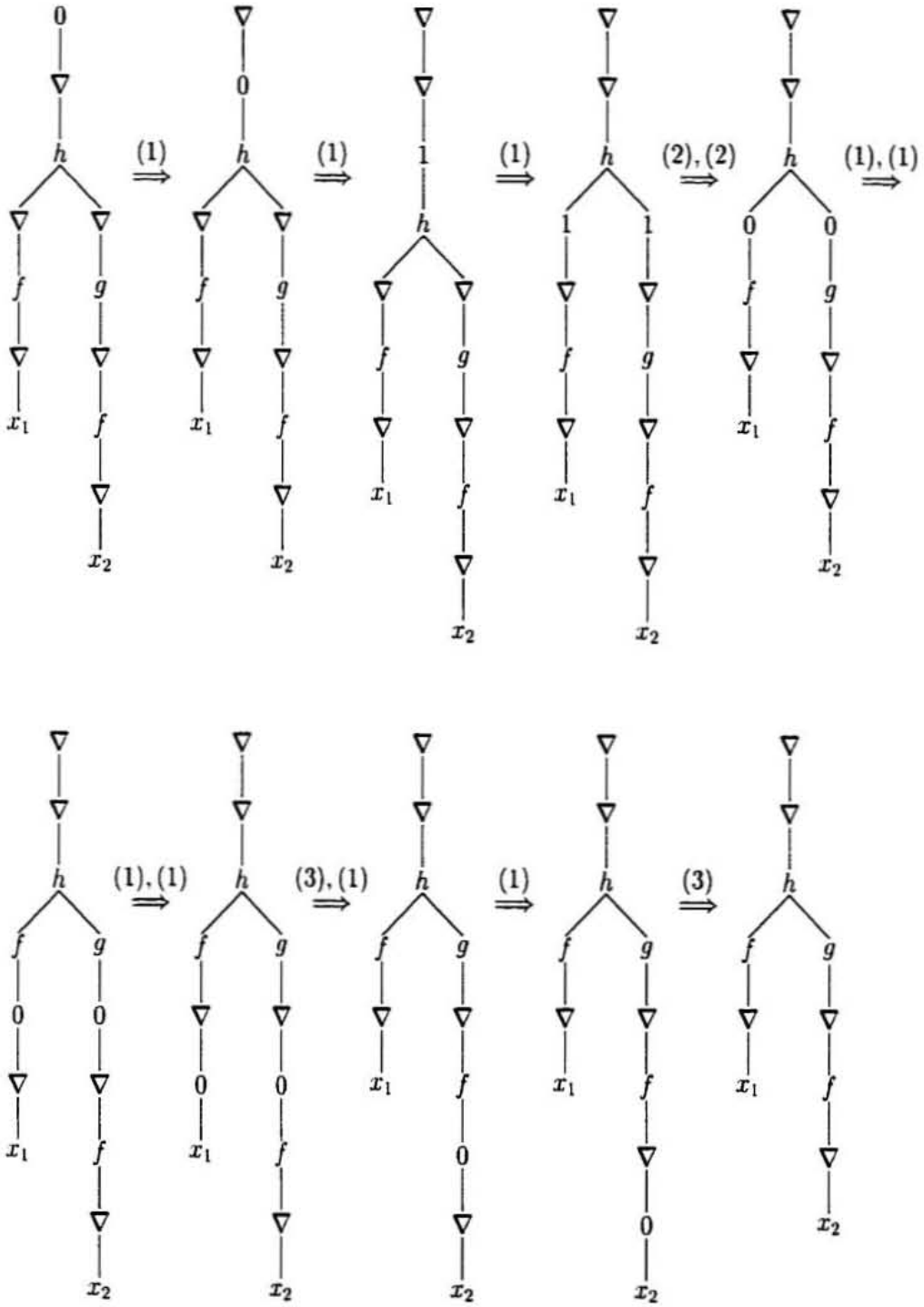

0
h
f
g
x1
f
x2
(1)
0
h
f
g
x1
f
x2
(1)
1
h
f
g
x1
f
x2
(1)
h
1
1
f
g
x1
f
x2
(2), (2)
h
0
0
f
g
x1
f
x2
(1), (1)
h
f
g
0
0
x1
f
x2
(1), (1)
h
f
g
0
0
x1
f
x2
(3), (1)
h
f
g
x1
f
0
x2
(1)
h
f
g
x1
f
0
x2
(3)
h
f
g
x1
f
x2

Figure 4.2.

DEFINITION 4.5 Two regular infinite trees $t_1, t_2 \in T^{\infty}_{\Sigma_\nabla}(X)$ are *retiming equivalent*, in notation $t_1 \sim t_2$, if there exist SΣ-schemes S_1, S_2 such that $t_1 = T(S_1)$, $t_2 = T(S_2)$ and $S_1 \sim S_2$.

THEOREM 4.6 *The relation of retiming equivalence on regular infinite Σ_∇-trees can be characterized as*:

$$\sim = \bigcup_{n \geq 0} \mathfrak{R}(\mathcal{T}_n).$$

PROOF SKETCH. ($\Rightarrow$) Let $t_1 = T(S_1)$ and $t_2 = T(S_2)$ for some strong retiming equivalent SΣ-schemes S_1 and S_2. Then S_1 and S_2 can be unfolded into schemes S_1' and S_2' that are already retiming equivalent, that is, there exists a legal retiming vector R taking S_1' into S_2'. The number of states $[-n, n]$ of the finite state top-down tree transducer $\mathcal{T}_n$ which takes t_1 into t_2 is determined by the maximum absolute value of $R(v)$, i.e., $n = \max\{|R(v)| \mid v \in V(S_1)\}$.

($\Leftarrow$) Let $t_1 = T(S_1)$ and $t_2 = T(S_2)$ for some synchronous schemes S_1 and S_2, and let $\mathcal{T}_n'$ be a finite state top-down tree transducer which maps t_1 into t_2 with the following feature: non ∇ nodes are additionally labeled with the state in which they are processed. We will denote the resulting tree as lt_1. It is obvious that transducer $\mathcal{T}_n'$ forces the common underlying flowchart scheme structure on both schemes S_1 and S_2. Let $\hat{\mathrm{H}}$ denote the product of $fl(utr(S_1))$ and $fl(utr(S_2))$. Construct schemes H_1 and H_2, which are the unfoldings of S_1 and S_2 determined by the product scheme $\hat{\mathrm{H}}$ as follows: reverse the flow of $\hat{\mathrm{H}}$ and, starting from its exit vertices, insert ∇ nodes into it following the structure of t_1 and t_2 respectively. Now, starting from exit vertices of H_1 and following the structure of lt_1, label non ∇ nodes in H_1 with labels from corresponding nodes in lt_1. Since $\mathcal{T}_n'$ maps t_1 into t_2, the corresponding nodes in H_1 and lt_1 have the same labels and these labels determine the legal retiming vector which maps H_1 into H_2. Therefore, S_1 and S_2 are strong retiming equivalent. ■

The following is the example where it is not possible to translate one tree into another using the finite state top-down tree transducer $\mathcal{T}_n$ for any $n \in \mathbb{N}$. Notice the difference between the number of ∇ nodes along the corresponding paths.

EXAMPLE 4.3 The input tree is $\nabla h(\nabla f \nabla x_1, \nabla g \nabla f \nabla x_2)$ and the goal output tree is $h(\nabla f \nabla x_1, \nabla g \nabla f \nabla x_2)$.

$$
\begin{aligned}
0\nabla h(\nabla f \nabla x_1, \nabla g \nabla f \nabla x_2) &\Longrightarrow (-1)h(\nabla f \nabla x_1, \nabla g \nabla f \nabla x_2) && \text{rule (2)} \\
&\Longrightarrow h((-1)\nabla f \nabla x_1, (-1)\nabla g \nabla f \nabla x_2) && \text{rule (1)} \\
&\Longrightarrow h(\nabla(-1) f \nabla x_1, \nabla(-1) g \nabla f \nabla x_2) && \text{rule (1),(1)} \\
&\Longrightarrow h(\nabla f(-1)\nabla x_1, \nabla g(-1)\nabla f \nabla x_2) && \text{rule (1),(1)} \\
&\Longrightarrow h(\nabla f \nabla(-1) x_1, \nabla g \nabla(-1) f \nabla x_2) && \text{rule (1),(1)} \\
&\Longrightarrow h(\nabla f \nabla(-1) x_1, \nabla g \nabla f(-1)\nabla x_2) && \text{crash, rule (1)} \\
&\Longrightarrow h(\nabla f \nabla(-1) x_1, \nabla g \nabla f \nabla(-1) x_2) && \text{crash, rule (1)} \\
&\Longrightarrow h(\nabla f \nabla(-1) x_1, \nabla g \nabla f \nabla(-1) x_2) && \text{crash}
\end{aligned}
$$

DEFINITION 4.7 We define the finite state top-down tree transducer $\mathcal{O}_n$ which takes as input $t \in T^{\infty}_{\Sigma_\nabla}(X)$ such that $t = T(S)$ for some SΣ-scheme S, and translates it into the output of scheme S at the n^{th} clock tick, assuming an initial configuration with $\perp$'s assigned to all registers, as follows:

$\Sigma = \Sigma_\nabla(X)$

$\Delta = \Sigma \cup \{\perp\} \cup \{x^i \mid i \geq 1, x \in X\}$

$Q = \{0, 1, \ldots, n-1\}$

$Q_d = \{n-1\}$

R is the set of rules defined below

(1) $i\sigma(y_1, \ldots, y_n) \rightarrow \sigma(iy_1, \ldots, iy_n)$ for $\sigma \in \Sigma_n$

(2) $i\nabla(y_1) \rightarrow (i-1)y_1$ if $i \geq 1$

(3) $0\nabla(y_1) \rightarrow \perp$

(4) $ix_j \rightarrow x_j^{i+1}$ for $i \in \mathbb{N}$.

Notice that $\mathcal{O}_n$ is deterministic. The variable symbol x_j^i stands for the input arriving from input channel j in the the i^{th} clock cycle.

If the starting configuration c is different, then introduce unary symbols of the form (∇, p), where $p \in T_\Sigma$ is a finite tree representing the contents of a register according to c. Modify the above rules (2) and (3) as:

(2') $i(\nabla, p)(y_1) \to (i-1)y_1$ if $i \geq 1$

(3') $0(\nabla, p)(y_1) \to p$

Call this transducer $\mathcal{O}_n(c)$, where c is the starting configuration.

Let $H_n(t)$ denote the net output height of an infinite tree $t \in T^\infty_{\Sigma_\nabla}(X)$ in the n^{th} step, i.e. the height of $\mathcal{O}_n(t)$, and let $H_n(c)(t)$ denote the total output height of an infinite tree $t \in T^\infty_{\Sigma_\nabla}(X)$ in the n^{th} step starting in configuration c, i.e. the height of $\mathcal{O}_n(c)(t)$. Note: $H_n(c)(t) - H_n(t) \leq k_c$ for a fixed bound k_c depending on configuration c.

LEMMA 4.8 *Let S and S' be Leiserson equivalent $S\Sigma$-schemes Then S and S' are strong retiming equivalent.*

PROOF. Recall the definition of Leiserson equivalency (Definition 2.5). Assume, by way of contradiction, that S and S' are Leiserson equivalent, but $S \not\approx S'$. Then

$$(1) \quad fl(T(S)) = fl(T(S'))$$
$$(2) \qquad T(S) \not\approx T(S').$$

Condition (1) is necessary for two schemes to be Leiserson equivalent. For if $fl(T(S)) \neq fl(T(S'))$ then, no matter what the configurations of S and S' are, they will never exhibit the same behavior, that is, produce the same sequence of outputs for the same sequence of inputs.

By virtue of Lemma 3.2.2 and Lemma 2.11 (characterization of strong retiming equivalence), (2) can only happen if

(i) There exists a finite branch in $fl(T(S))$ leading to variable x_j^i such that the corresponding branches in $T(S)$ and $T(S')$ have a different number of registers along them; or

(ii) There exists an infinite branch in $fl(T(S))$ such that the absolute difference of the number of registers along the corresponding branches in $T(S)$ and $T(S')$ is ∞, i.e., $\lim_{n\to\infty} \mid H_n(T(S)) - H_n(T(S')) \mid = \infty$.

If (i) is the case, then it is easy to see that the input x_j^i will always appear in different clock cycles in the output sequences of S and S'. Therefore the equation

$$\mathcal{O}_n(c)(T(S)) = \mathcal{O}_n(c')(T(S'))$$

will not hold for every $n \geq 0$, no matter how the configurations c and c' are chosen. This contradicts the hypothesis that S and S' are Leiserson equivalent.

In case (ii), according to our hypothesis, there exist configurations c and c' for S and S' respectively, such that

$$\mathcal{O}_n(c)(T(S)) = \mathcal{O}_n(c')(T(S'))$$

for all $n \geq 0$. Therefore $H_n(c)(T(S)) = H_n(c')(T(S'))$. On the other hand, by assumption we also have:

$$\lim_{n\to\infty} \mid H_n(T(S)) - H_n(T(S')) \mid = \infty.$$

This is a contradiction since there exists a fixed bound k_c such that $H_n(c)(t) - H_n(t) \leq k_c$ for all infinite trees $t \in T^{\infty}_{\Sigma_\nabla}(X)$. ■

LEMMA 4.9 *Let S_1 and S_2 be strong retiming equivalent* SΣ*-schemes. Then S_1 and S_2 are Leiserson equivalent.*

PROOF. According to Lemma 2.11, there exist SΣ-schemes S_1' and S_2' such that S_i and S_i' are strongly equivalent for $i = 1, 2$, and $S_1' \sim_r S_2'$. By definition, if two SΣ-schemes are strongly equivalent, then they are Leiserson equivalent. On the other hand, Lemma 2.1 (*Retiming Lemma*) assures that S_1' and S_2' are Leiserson equivalent. ■

THEOREM 4.10 *Two synchronous schemes S_1 and S_2 are Leiserson equivalent if and only if they are strong retiming equivalent.*

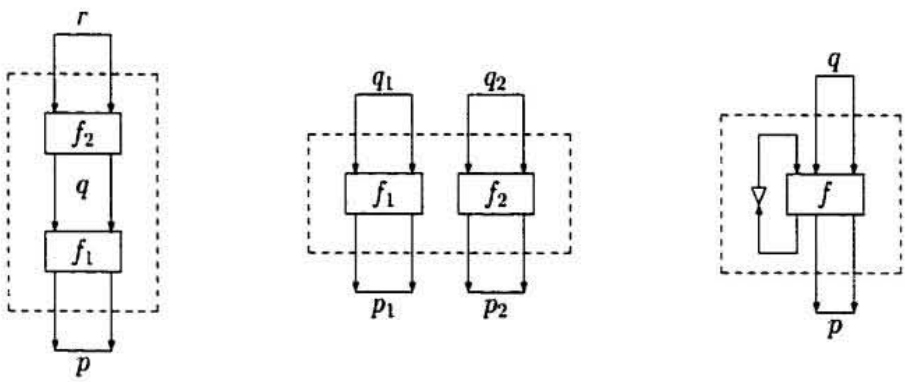
PROOF. Follows directly from Lemmas 4.8 and 4.9. ■

4 Retiming Identities

4.1 The Algebra of Synchronous Schemes

It was observed in [Elgot and Shepardson, 1979] that flowchart schemes can be treated as morphisms in a strict monoidal category [MacLane, 1971] over the set of objects $\mathbb{N} = \{0, 1, 2, \ldots\}$. Arnold and Dauchet [1978, 1979] reformulated these categories as $\mathbb{N} \times \mathbb{N}$ sorted algebras called magmoids. In a magmoid M, we have an underlying set $M(p, q)$ corresponding to each pair (p, q) of nonnegative integers, and the basic operations are the following:

- *Composition*: maps $M(p, q) \times M(q, r)$ into $M(p, r)$ for each triple $p, q, r \in \mathbb{N}$, denoted by $\cdot$. See Figure 5.1(a).
- *Sum*: maps $M(p_1, q_1) \times M(p_2, q_2)$ into $M(p_1 + p_2, q_1 + q_2)$ for every choice of the nonnegative integers p_1, p_2, q_1, q_2, denoted by $+$. See Figure 5.1(b).
- *Feedback*: maps $M(1+p, 1+q)$ into $M(p, q)$ for each pair $(p, q) \in \mathbb{N} \times \mathbb{N}$, denoted by $\uparrow$. See Figure 5.1(c). The application of $\uparrow$ *creates* triangles (boxes of sort $1 \to 1$) which represent registers.



(a) Composition. $f_1 \cdot f_2 : p \to r$

(b) Sum. $f_1 + f_2 : p_1 + p_2 \to q_1 + q_2$

(c) Feedback. $\uparrow f : p \to q$

Figure 5.1: The interpretation of operations.

There are two constants in M, 0 and 1, standing for the identity arrows 1_0 and 1_1, respectively. By the strict monoidal property, 1_p ($p \geq 1$) then corresponds to the element $\sum_{i=1}^{p} 1$ in $M(p,p)$. We use the notation p for $\sum_{i=1}^{p} 1$, and adopt the categorical terminology $f : p \to q$ to mean that f is an element (morphism) of sort (p,q) in M. The operations and constants are subject to the obvious identities M1, ..., M5 below.

The magmoid operations are, however, not sufficient to express even the most elementary schemes, i.e., mappings. For this reason, some further constants are to be introduced. Usually the constants π_p^i for all $p \in \mathbb{N}$ and $i \in [p] = \{1, 2, \ldots, p\}$ are chosen. The constant $\pi_p^i : 1 \to p$ represents the mapping $[1] \to [p]$ which sends 1 to i. This choice is natural, because the semantics of flowchart schemes is defined in algebraic theories [Lawvere, 1963], and the constants π_p^i are included in the type of the coresponding $\mathbb{N} \times \mathbb{N}$ sorted algebras. However, regarding the pure syntax of schemes only, the choice of the constants π_p^i is not the simplest one. Indeed, every mapping can be expressed by the help of the *transposition* $x : 2 \to 2$, the *join* (or *branch*) $\varepsilon : 2 \to 1$, and the *zero* $0_1 : 0 \to 1$ using the magmoid operations. These constants are also natural for us, even from the semantic point of view, because we consider schemes to be logical circuits. In this case the constants x, ε and 0_1 are interpreted as the simplest switching elements in the circuits, see Figure 5.2.

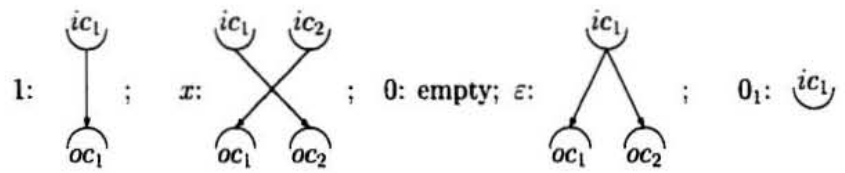


Figure 5.2: The interpretation of constants.

In accordance with [Bartha, 1987], S denotes the type consisting of the operations $\cdot, +$ and $\uparrow$ and constants 0, 1, x, ε and 0_1, and D is the subtype of S not containing $\uparrow$. This way we have defined the S-algebra $\mathrm{Sf}(\Sigma)$, where $\mathrm{Sf}(\Sigma)(p,q)$ is the

set of all SΣ-schemes of sort $p \to q$ over a doubly ranked alphabet Σ. Recall that $\Sigma = \{\Sigma(p,q) \mid (p,q) \in \mathbb{N} \times \mathbb{N}\}$ where the sets $\Sigma(p,q)$ are pairwise disjoint. With each $\sigma(p,q) \in \Sigma(p,q)$ we associate an atomic SΣ-scheme with $p+q+1$ vertices ($2p+2q$ ports) shown in Figure 5.3.

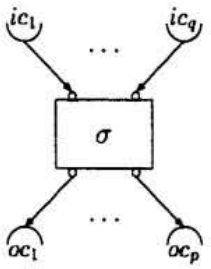
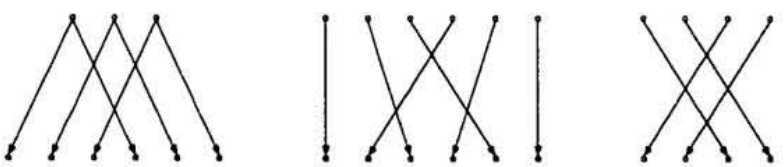


Figure 5.3: $\sigma \in \Sigma(p,q)$ as an atomic scheme.

The following mappings will play an important role in the sequel:

- $\varepsilon_k : k \to 1$ is the unique one of its sort.
- $w_p(q) : p \cdot q \to q$. For any $p, q \in \mathbb{N}$, $w_p(q)$ takes a number of the form $(j-1) \cdot q + i$ ($j \in [p], i \in [q]$) to i. See Figure 5.4a.
- $\kappa(n,p) : p \cdot n \to n \cdot p$ is the permutation (sometimes called a perfect shuffle) which rearranges p blocks of length n into n blocks of length p, i.e., $\kappa(n,p)$ takes $(j-1) \cdot n + i$ ($j \in [p], i \in [n]$) to $(i-1) \cdot p + j$. See Figure 5.4b.
- $\beta\#s$. If $\beta : r \to r$ is any permutation and s is a sequence $(n_1, \ldots, n_r)$ of nonnegative integers with $n = \sum_{i=1}^{r} n_i$, then $\beta\#s : n \to n$ is the block by block performance of β on s, i.e, $\beta\#s$ sends $j + \sum_{i=1}^{k} n_i$, where $j \in [n_{k+1}]$ to the number $y + j$, where y is the sum of numbers n_i such that $\beta(i) < \beta(k+1)$. See Figure 5.4c.



a) mapping $w_2(3)$ b) mapping $\kappa(3,2)$ c) mapping $x\#(2,2)$

Figure 5.4: Examples of mappings $w_p(q)$, $\kappa(n,p)$ and $\beta\#s$.

4.2 Equational Axiomatization of Synchronous Schemes

The syntactical and semantical features of synchronous systems can be conveniently separated. The syntax is specified by a synchronous scheme. The semantics is then specified by an algebra, which associates a fixed operation with each operation symbol. The set of identities SF has been developed in [Bartha, 1987]. In this section we augment SF with a new axiom R, intended to capture the retiming equivalence of synchronous schemes and develop the system of identities F_rT to serve as a basis of identities of feedback theories being the semantics of synchronous schemes. The first set of identities towards the axiomatization of schemes is MG:

1. MG = {M1, ..., M5} is the set of magmoid identities, where

   M1: $f \cdot (g \cdot h) = (f \cdot g) \cdot h$ if $f : p \to q, g : q \to r, h : r \to s$;

   M2: $f + (g + h) = (f + g) + h$ if $f : p_1 \to q_1, g : p_2 \to q_2, h : p_3 \to q_3$;

   M3: $p \cdot f = f \cdot q = f$ if $f : p \to q$;

   M4: $f + 0 = 0 + f = f$ if $f : p \to q$;

   M5: $(f_1 \cdot g_1) + (f_2 \cdot g_2) = (f_1 + f_2) \cdot (g_1 + g_2)$ if $f_i : p_i \to q_i, g_i : q_i \to r_i, i = 1, 2$.

2. DF = MG ∪ {P, D1, D2, D3}, where

   P: $f_1 + f_2 = x\#(p_1, p_2) \cdot (f_2 + f_1) \cdot x\#(q_2, q_1)$ if $f_i : p_i \to q_i, i = 1, 2$.

P is the block permutation axiom introduced by Elgot and Shepherdson [1980]. This axiom postulates a *symmetry* [MacLane, 1971] for the strict monoidal category determined by the axioms MG.

D1: $(\varepsilon + 1) \cdot \varepsilon = (1 + \varepsilon) \cdot \varepsilon$;

D2: $x \cdot \varepsilon = \varepsilon$;

D3: $(1 + 0_1) \cdot \varepsilon = 1$.

3. SF = DF ∪ {S1, S2, ..., S9}, where

   S1: $\uparrow(f_1 + f_2) = \uparrow f_1 + f_2$ if $f_1 : 1 + p_1 \to 1 + q_1, f_2 : p_2 \to q_2$;

   S2: $\uparrow^2((x + p) \cdot f) = \uparrow^2(f \cdot (x + q))$ if $f : 2 + p \to 2 + q$;

   S3: $\uparrow(f \cdot (1 + g)) = (\uparrow f) \cdot g$ if $f : 1 + p \to 1 + q, g : q \to r$;

   S4: $\uparrow((1 + g) \cdot f) = g \cdot \uparrow f$ if $f : 1 + q \to 1 + r, g : p \to q$;

   S5: $\uparrow 1 = 0$;

   S6: $\varepsilon \cdot \bot = \bot + \bot$, where $\bot = \uparrow\varepsilon$;

   S7: $\uparrow(f \cdot (\varepsilon + q)) = \uparrow^2((\varepsilon + p) \cdot f)$ if $f : 1 + p \to 2 + q$;

   S8: $0_1 \cdot \nabla = 0$, where $\nabla = \uparrow x$;

   S9: $\uparrow(\varepsilon \cdot \nabla^n) = \bot \ \forall n \in \mathbb{N}$, where ∇^n denotes the n-fold composite of ∇.

4. RF = SF ∪ R, where

   R: $\uparrow^{p_1}(f \cdot (g + q_2)) = \uparrow^{q_1}((g + p_2) \cdot f)$ if $f : p_1 + p_2 \to q_1 + q_2$ and $g : q_1 \to p_1$.
   For the interpretation of axiom R see Figure 5.5.

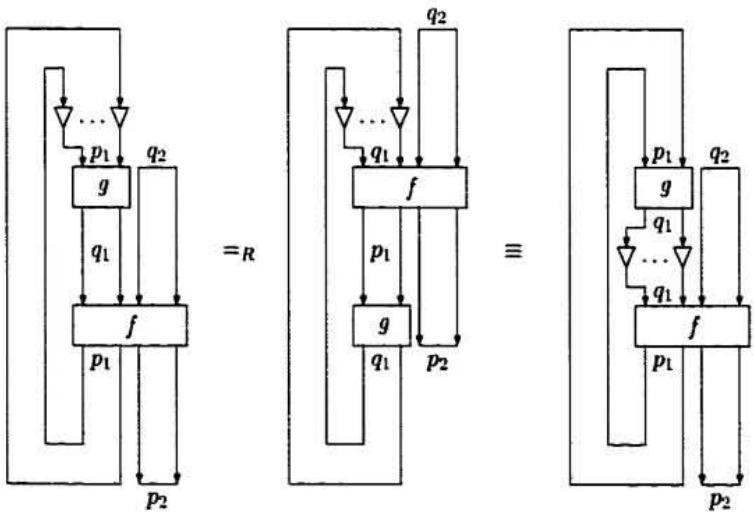


Figure 5.5: Retiming identity.

CLAIM: The following identity is provable from RF (See also Figure 5.6):

$$\mathrm{R}*: \sum_{i=1}^{p} \nabla \cdot f = f \cdot \sum_{i=1}^{q} \nabla \quad \text{for} \quad f : p \to q$$

PROOF.

$$\begin{aligned}
\sum_{i=1}^{p} \nabla \cdot f &\overset{def}{=} \uparrow^{p}(f \cdot (p+q)) \overset{\mathrm{P}}{=} \uparrow^{p}(f \cdot x\#(p,q)) \\
&\overset{\mathrm{R}}{=} \uparrow^{q}(x\#(q,p) \cdot f) \overset{\mathrm{P}}{=} \uparrow^{q}((q+p) \cdot f) \\
&\overset{def}{=} f \cdot \sum_{i=1}^{q} \nabla \ \blacksquare
\end{aligned}$$

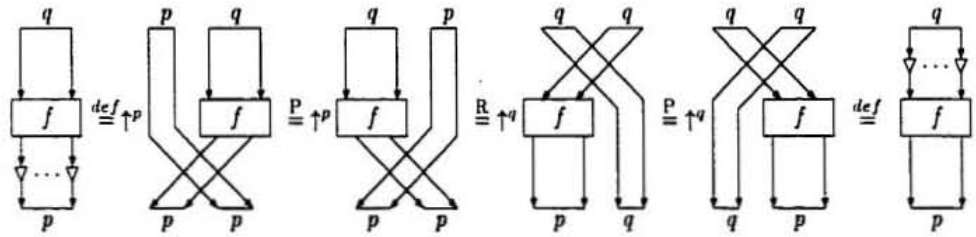


Figure 5.6: Proof of identity R* in a diagram.

Note, however, that identity R* alone is not sufficient to capture the retiming equivalence of synchronous schemes. Consider $S_1 = \uparrow(\varepsilon \cdot f \cdot g)$ and $S_2 = \uparrow((g+1) \cdot \varepsilon \cdot f)$. See Figure 5.7.

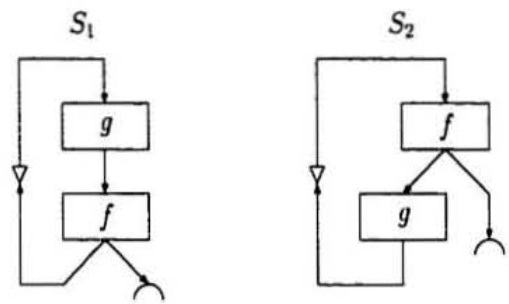


Figure 5.7.

Schemes S_1 and S_2 obviously exhibit the same behavior, yet equation $S_1 = S_2$ is not provable from SF ∪ R*. The only axiom that interchanges the composition is P7

which is not applicable in this case. On the other hand, $\uparrow(\varepsilon \cdot f \cdot g) = \uparrow((g+1) \cdot \varepsilon \cdot f)$ follows directly from R.

Let Q be a type of $\mathbb{N} \times \mathbb{N}$ sorted algebras and Σ be a doubly ranked alphabet. If E is a set of Q-identities, then we denote by $\mathcal{K}_Q(E)$ the variety of all Q-algebras in which the identities E are valid. If $\mathfrak{A}$ is a Q-algebra, then $\Phi_{\mathfrak{A}}(E)$, or simply $\Phi(E)$, denotes the congruence relation of $\mathfrak{A}$ induced by E, i.e., the smallest congruence relation for which $\mathfrak{A}/\Phi(E)$ (the quotient of $\mathfrak{A}$ by $\Phi(E)$) becomes an algebra in $\mathcal{K}_Q(E)$.

THEOREM 5.2.1 *The congruence relation* $\Phi(\mathrm{R})$ *induced by axiom* R *in the algebra* $\mathrm{Sf}(\Sigma)$ *is the retiming equivalence relation of synchronous schemes.*

PROOF. As retiming equivalence is the smallest equivalence containing the primitive retiming relation (retiming one box only), and $\Phi(\mathrm{R})$ is also an equivalence, it is sufficient to show that if SΣ-scheme S' is obtained from S via one primitive retiming step, then $\mathrm{R} \vdash S = S'$ in the algebra $\mathrm{Sf}(\Sigma)$.

Let $S, S' : p \to q$ be SΣ-schemes such that S' is obtained from S by retiming a single box. S can be represented as $\uparrow^{q_1}((g+p)\cdot F)$, $F : p_1 + p \to q_1 + q$ representing the "surroundings" and $g : q_1 \to p_1$ representing the single box. Then $S' = \uparrow^{p_1}(F \cdot (g+q))$ follows from S by a single application of axiom R. See also Figure 5.8. ■

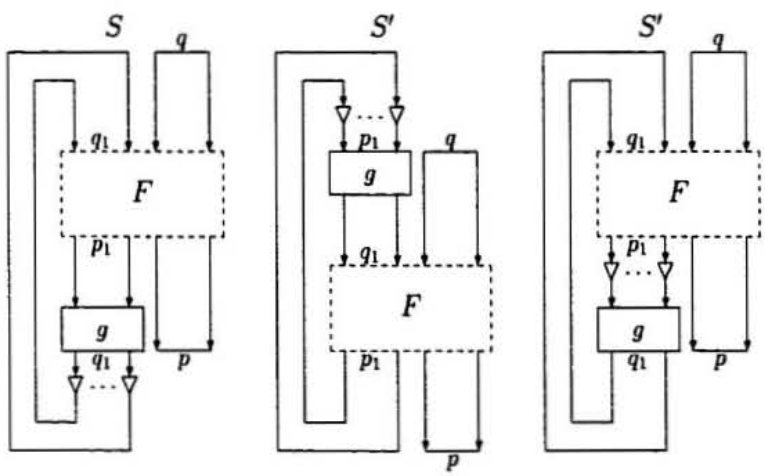


Figure 5.8: Congruence $\Phi(\mathrm{R})$ as the retiming equivalence relation.

THEOREM 5.2.2 [Bartha, 1987] $\mathrm{Sf}(\Sigma)$ *is freely generated by* Σ *in* $\mathcal{K}_S(\mathrm{SF})$.

THEOREM 5.2.3 $\mathrm{Sf}(\Sigma)/\Phi(\mathrm{R})$ *is freely generated by* Σ *in* $\mathcal{K}_S(\mathrm{RF})$.

PROOF. It is well known that if free algebra over the equational class of algebras exists then it is isomorphic to a quotient algebra of terms, where the quotient is taken with respect to the congruence induced by the set of axioms (equations). Therefore:

$$\mathrm{Sf}(\Sigma) \cong \text{T-S}\Sigma/\Phi(\mathrm{SF})$$

where T-SΣ denotes the term algebra over Σ and $\Phi(\mathrm{SF})$ denotes the congruence relation induced by the set of axioms SF. Let $\Phi(\mathrm{R})$ denote the congruence relation induced by the retiming axiom R. Then, by the second isomorphism theorem:

$$\mathrm{Sf}(\Sigma)/\Phi(\mathrm{R}) \cong (\text{T-S}\Sigma/\Phi(\mathrm{SF}))/\Phi(\mathrm{R}) \cong \text{T-S}\Sigma/\Phi(\mathrm{SF} \cup \{\mathrm{R}\}). \ \blacksquare$$

In our axiomatic treatment, algebraic theories can be introduced by the help of identities $\mathrm{TH} = \{\mathrm{T1}, \mathrm{T2}\}$, where

$$\begin{aligned} &\mathrm{T1}: \quad 0_1 \cdot f = 0_q \ \text{ for } \ f : 1 \to q \\ &\mathrm{T2}: \quad w_p(p) \cdot f = \left(\sum_{i=1}^{p} f\right) \cdot w_p(q) \ \text{ for } \ f : p \to q \end{aligned}$$

We define the identity R1 as follows:

$$\mathrm{R1}: \ \uparrow^{p_1}(f \cdot (g + q_2)) = \uparrow((g + p_2) \cdot f) \quad \text{for} \quad f : p_1 + p_2 \to 1 + q_2, g : 1 \to p_1$$

THEOREM 5.2.4 *In the presence of the theory axiom* TH *it is sufficient to consider axiom* R1 rather than R.

PROOF. We have to prove that

$$\mathrm{R}: \ \uparrow^{p_1}(f \cdot (g + q_2)) = \uparrow^{q_1}((g + p_2) \cdot f) \quad \text{for} \quad f : p_1 + p_2 \to q_1 + q_2, g : q_1 \to p_1$$

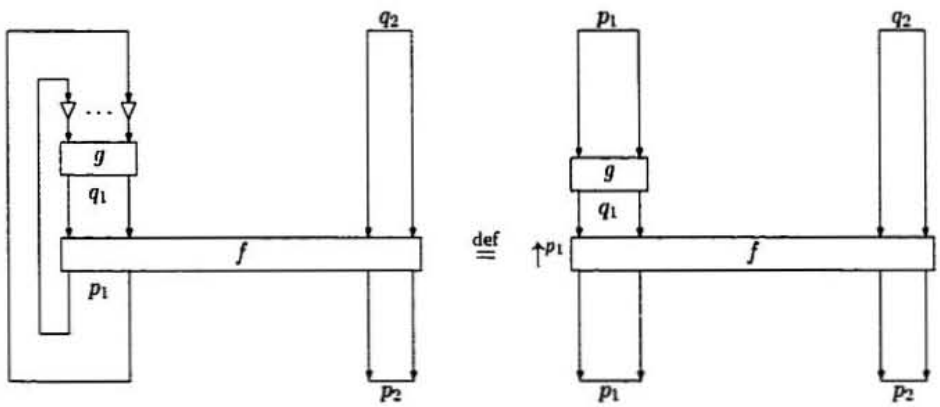
is a consequence of $\mathrm{SF} \cup \mathrm{TH} \cup \{\mathrm{R1}\}$. The proof is an induction argument on q_1. If $q_1 = 1$ then axiom R is of the form

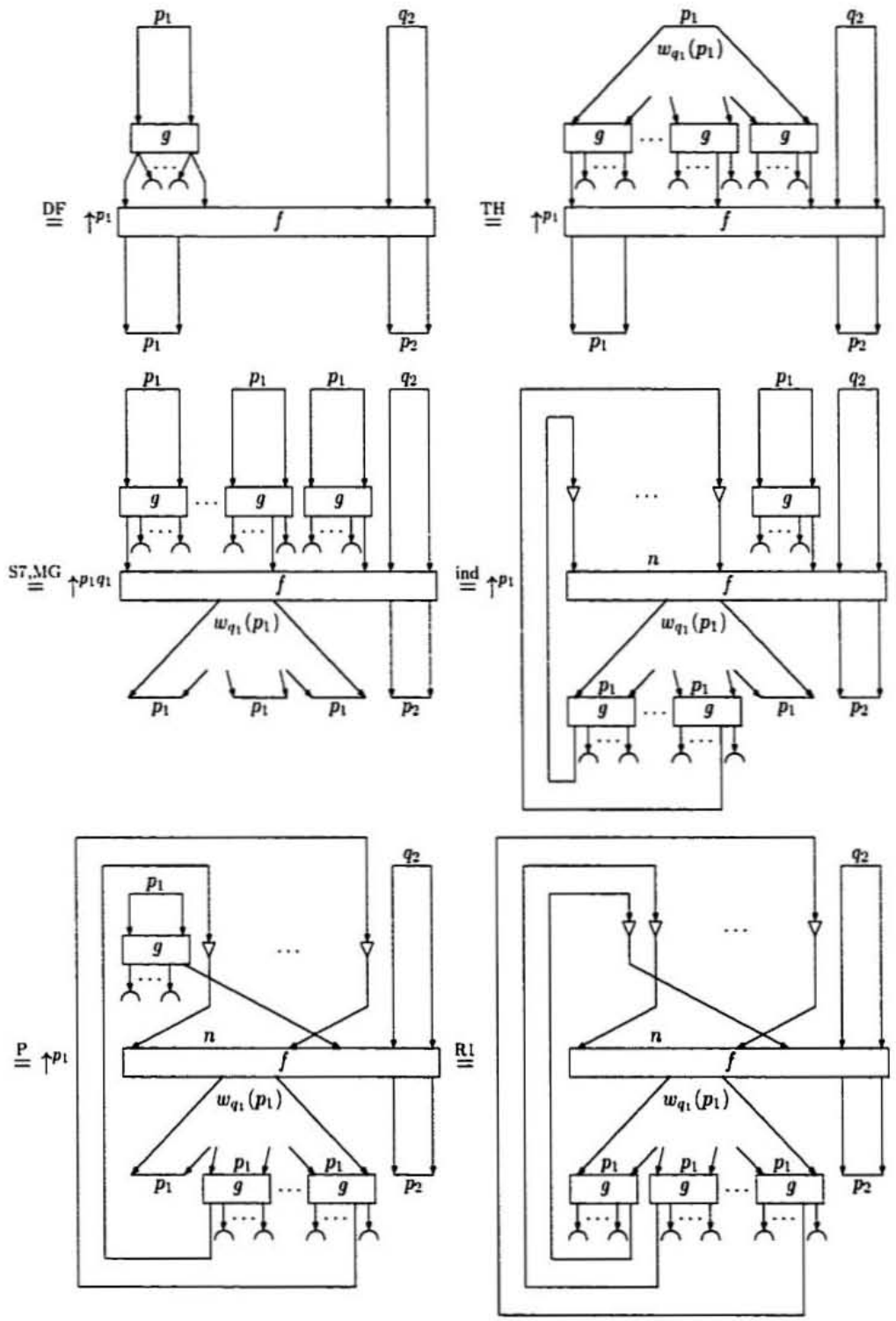
$$\uparrow^{p_1}(f \cdot (g + q_2)) = \uparrow^{q_1}((g + p_2) \cdot f) = \uparrow((g + p_2) \cdot f)$$

that is, R reduces to R1. Now assume that $q_1 \geq 1$ and that the theorem is true for $q_1 = n$. Then for $q_1 = n+1$ we have

$$
\begin{aligned}
& \uparrow^{p_1}(f \cdot (g + q_2)) \\
\overset{\text{DF}}{=} \quad & \uparrow^{p_1}(f \cdot ((1 + \cdots + 0_1 + \cdots + 0_1 + \cdots + 1) \cdot w_{q_1}(p_1) \cdot g + q_2)) \\
\overset{\text{TH}}{=} \quad & \uparrow^{p_1}(f \cdot ((1 + \cdots + 0_1 + \cdots + 0_1 + \cdots + 1) \cdot \sum_{i=1}^{n+1} g \cdot (w_{q_1}(p_1) + q_2))) \\
\overset{\text{S7,MG}}{=} \quad & \uparrow^{p_1 q_1}((w_{q_1}(p_1) + p_2) \cdot f \cdot ((1 + \cdots + 0_1) \cdot g + \cdots + (0_1 + \cdots + 1) \cdot g) + q_2)) \\
\overset{\text{ind}}{=} \quad & \uparrow^{p_1}(\uparrow^{n}(((1 + \cdots + 0_1 + 0_1) \cdot g + \cdots + (0_1 + \cdots + 1 + 0_1) \\
& g + p_2) \cdot w_{q_1}(p_1) \cdot f) \cdot (n + (0_1 + \cdots + 0_1 + 1) \cdot g + q_2)) \\
\overset{\text{P}}{=} \quad & \uparrow^{p_1}(\uparrow^{n}(x\#((1 + \cdots + 0_1 + 0_1) \cdot g + \cdots + (0_1 + \cdots + 1 + 0_1) \cdot g + p_2) \\
& w_{q_1}(p_1) \cdot f)(x\#((0_1 + \cdots + 0_1 + 1) \cdot g + n) + q_2)) \\
\overset{\text{R1}}{=} \quad & \uparrow^{q_1}(x\#((0_1 + \cdots + 0_1 + 1) \cdot g + (1 + \cdots + 0_1 + 0_1) \cdot g + \cdots + \\
& (0_1 + \cdots + 1 + 0_1) \cdot g + p_2) \cdot w_{q_1}(p_1) \cdot f \cdot (nx + q_2)) \\
\overset{\text{P}}{=} \quad & \uparrow^{q_1}(((1 + \cdots + 0_1) \cdot g + \cdots + (0_1 + \cdots + 1) \cdot g) + p_2) \cdot w_{q_1}(p_1) \cdot f) \\
\overset{\text{TH}}{=} \quad & \uparrow^{q_1}((1 + \cdots + 0_1 + \cdots + 0_1 + \cdots + 1) \cdot w_{q_1}(p_1) \cdot g + p_2) \cdot f) \\
\overset{\text{DF}}{=} \quad & \uparrow^{q_1}((g + p_2) \cdot f) \ \blacksquare
\end{aligned}
$$

See also Figure 5.9.




DF
TH
ST,MG
ind
P
RI

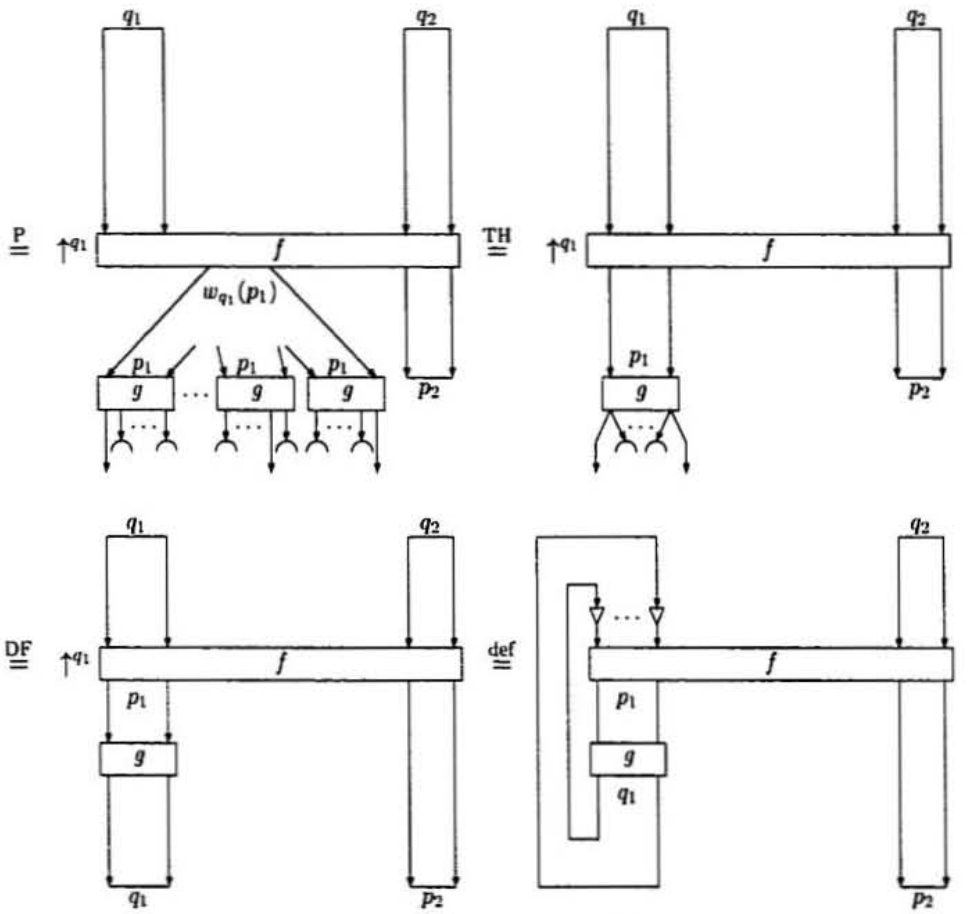


Figure 5.9: Proof of Theorem 5.2.4 in a diagram.

Concerning feedback theories, we introduce the well-known *commutative identity* [Ésik, 1980] in the following alternative way:

$$\text{C: } w_n(p)\cdot \uparrow^l f = \uparrow^{l\cdot n}(f * (\rho_1,\ldots,\rho_l))\cdot w_n(q) \text{ if } f : l+p \to l+q,$$

for all $n \in \mathbb{N}$ under every choice of mapping $\rho_1,\ldots,\rho_l : n \to n$, where

$$f * (\rho_1,\ldots,\rho_l) = \alpha(l,n,p)^{-1}\cdot\left(\sum_{i=1}^{n} f\right)\cdot\alpha(l,n,q)\cdot\left(\sum_{i=1}^{l}\rho_i + n\cdot q\right)$$

and $\alpha(l,n,m) = (\kappa(2,n)\#(l,m)^n)\cdot(\kappa(l,n)+n\cdot m)$. See Figure 5.10 for an instance of C in the case $n=3$, $l=2$ and $p=q=1$.

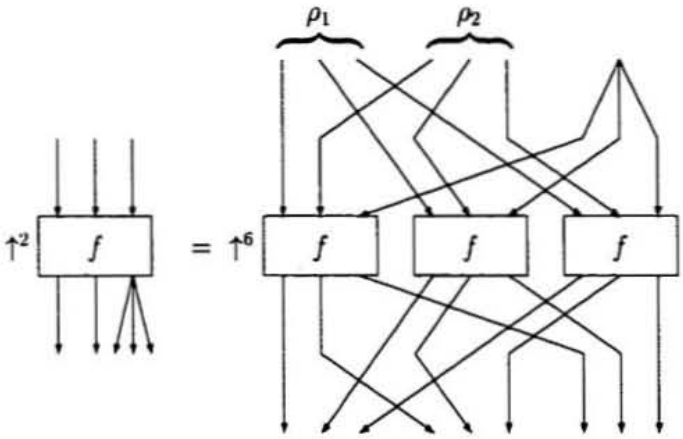


Figure 5.10: The axiom C for $n = 3, l = 2$ and $p = q = 1$.

DEFINITION 5.2.5 We define the *strong retiming feedback theory* F_rT as

$$F_rT = SF \cup THC \cup \{R1\}$$

where $THC = TH \cup C$.

COROLLARY 5.2.6 Strong retiming equivalence of synchronous schemes can be characterized as a congruence relation $\Phi(F_rT)$ induced on the set of $S\Sigma$-schemes by the axiom set F_rT.

PROOF. Follows immediately from Theorem 5.2.1 and Definition 5.2.5. ■

COROLLARY 5.2.7 The free algebra in $\mathcal{K}_S(F_rT)$ generated by Σ has a characterization by equivalence classes of infinite Σ_∇-trees according to their retiming equivalence.

PROOF. Follows immediately from the fact that the free algebra in $\mathcal{K}_S(FT)$, where $FT = SF \cup THC$ is a feedback theory, generated by Σ has a characterization by equivalence classes of infinite Σ_∇-trees and Theorem 4.6. ■

5 The Algebra of Multiclocked Schemes

In this chapter we study the general case of multiclocked synchronous schemes. The motivation comes from the synchronous dataflow programming language LUSTRE [Halbwachs, Caspi, Raymond and Pilaud, 1991] proposed as a tool for programming reactive systems as well as for describing hardware and program verification.

5.1 The LUSTRE Programming Language

Reactive systems have been defined as computing systems which continuously interact with a given physical environment, when this environment is unable to synchronize logically with the system. This class of systems has been proposed [Harel and Pnueli 1985, Berry 1989] to distinguish them from *transformational* systems - i.e., classical programs whose data are available at their beginning and which provide results when terminating - and from *interactive* systems which interact continuously with environments that possess synchronization capabilities. The dataflow aspect of LUSTRE makes it very close to usual description tools in these domains (block-diagrams, networks of operators, dynamical samples-systems, ...), and its synchronous interpretation makes it well suited for handling time in programs.

In LUSTRE, any constant, variable and expression denotes a *flow*, i.e., a pair made of a possibly infinite sequence of values and a *clock*, representing a sequence of time. A flow takes the n-th value of its sequence of values at the n-th clock tick. A LUSTRE program describes a network of operators controlled by a *global* (*basic*) clock. When executing, this network receives, at each clock tick, a set of inputs and calculates the set of outputs. The language is based on the perfect synchrony hypothesis, which means that all computations or communications take no time and that the net is supposed to react instantenously and to produce its outputs at the same time it receives its inputs. Other, slower clocks can be defined in terms of boolean flows. The clock defined by a boolean flow is the sequence of times at which

the flow takes the value *true*. For example, table 6.1 shows the time scales defined by the flow C whose clock is the basic clock, flow C_1 whose clock is defined by C and flow C_2 whose clock is defined by C_1.

basic time scale	1	2	3	4	5	6	7	8
C flow	*false*	*true*	*true*	*false*	*false*	*true*	*false*	*true*
C time scale		1	2			3		4
C_1 flow		*false*	*true*			*true*		*true*
C_1 time scale			1			2		3
C_2 flow			*true*			*false*		*true*
C_2 time scale			1					2

Table 6.1: Boolean clocks and flows.

LUSTRE has only few elementary basic types: boolean, integer, real and one type constructor: *tuple*. Complex types can be imported from a host language and handled as abstract types. Constants are those of basic types and those imported from the host language. Corresponding flows have constant sequences of values and their clock is the basic one. Variables must be defined with their types and variables which do not correspond to inputs should be given one and only one definition, in the form of equations (expressions). The equation "`X = E;`" defines variable `X` as being identical to expression `E` in the sense that `E` denotes the flow of variables of the same type $e_1, e_2, e_3, \ldots$ and $x_i = e_i$ for all $i \geq 1$ where $x_1, x_2, x_3, \ldots$ denotes the flow `X` with the same clock as `E`.

Usual operators over basic types are available (arithmetic: `+`, `-`, `*`, `/`, `div`, `mod`; boolean `not`, `and`, `or`; relational: `=`, `<`, `<=`, `>`, `>=`; conditional: `if then else`) and functions can be imported from the host language. These are called *data operators* and only operate on operands sharing the same clock.

What follows is the description of the context-free syntax of LUSTRE using a simple variant of Backus-Naur-Form (BNF). <*Italic*> type style words enclosed in

angle brackets are used to denote the syntactic categories and **Typewriter** type style words or characters are used to denote reserved words, delimiters or lexical elements of the language, other than identifiers. ε denotes the empty string.

<LUSTRE_program> ::= <sequence_of_nodes>

<sequence_of_nodes> ::= <node> | <node><sequence_of_nodes>

<node> ::= **node** <identifier> (<input_decl>) **returns** (<output_decl>);
<declaration_sequence>
let
<block>
tel.

<input_decl> ::= <variable_list>:<type> | <variable_list>:<type>;<input_decl> |
(<input_decl>) **when** $<variable>_B$;$<variable>_B$:**bool**

<output_decl> ::= <input_decl>

<variable_list> ::= <variable> | <variable>,<variable_list>

<type> ::= **int** | **bool** | **real**

<declaration_sequence> ::= ε | <declaration><declaration_sequence>

<declaration> ::= **var** <variable_list>:<type>;

<block> ::= <command>; | <command>;<block>

<command> ::= <variable> = <expression> | <tuple> = <expression> |
<assertion>

<expression> ::= <constant> | <variable> | <integer_expr> | <boolean_expr> |
<conditional_expr> | <temporal_expr> | <node_call>

<constant> ::= <numeral> | <boolean_constant>

<numeral> ::= <integer> | <real>

<integer> ::= <digit> | <digit><integer>

<real> ::= <integer>.<integer>

<boolean_constant> ::= **true** | **false**

<integer_expr> ::= <term> | <integer_expr><arithmetic_op><term>

<term> ::= <numeral> | <variable>

<arithmetic_op> ::= **+** | **-** | ***** | **/** | **div** | **mod**

<*boolean_expr*> ::= <*boolean_term*> | **not** <*boolean_expr*> |
<*boolean_expr*><*boolean_op*><*boolean_term*>

<*boolean_term*> ::= <*boolean_constant*> | <*variable*> | <*comparison*>

<*boolean_op*> ::= **and** | **or** | **xor**

<*comparison*> ::= <*integer_expr*>$_1$<*relation*><*integer_expr*>$_2$

<*relation*> ::= **=** | **<>** | **<** | **<=** | **>** | **>=**

<*conditional_expr*> ::= **if** <*boolean_expr*> **then** <*expression*>$_1$ **else** <*expression*>$_2$

<*temporal_expr*> ::= **pre** <*expression*> | <*expression*>$_1$ **->** <*expression*>$_2$ |
<*expression*>$_1$ **when** <*expression*>$_2$ | **current** <*expression*>

<*node_call*> ::= <*identifier*>(<*variable_list*>)

<*assertion*> ::= **assert** <*boolean_expr*>

<*variable*> ::= <*identifier*>

<*tuple*> ::= <*variable_list*>

<*identifier*> ::= <*letter*> | <*identifier*><*letter*> | <*identifier*><*digit*>

<*digit*> ::= **0** | ... | **9**

<*letter*> ::= **a** | ... | **z** | **A** | ... | **Z**

LUSTRE's specific operators are "temporal" operators **pre**, **->**, **when** and **current** which operate specifically on flows. A flow of values from a data domain D is a pair (d, τ) where d is a sequence over D and $\tau = [\tau_1, \ldots, \tau_n]$ is a clock of d. The basic data domains consist of finite and infinite sequences of integers and boolean values extended with the value $\perp$ to represent the absence of a value, which is treated like any other value - in particular, it is not smaller than other values in the domain ordering. The *clock* element $[\tau_1, \ldots, \tau_n]$ represents a clock that ticks as defined by the simple clock τ_1 and has been sampled using the clocks $\tau_2, \ldots, \tau_n$. The last element of this sequence τ_n is always the basic clock. An element $(d, [\tau_1, \ldots, \tau_n])$ represents the flow that produces the i-th element of d at the instant when the i-th tick of τ_1 appears.

The operator **pre** is the delay operator. It memorises the last value of a flow and outputs it when it receives a new value, transforming a sequence $e_1 e_2 \ldots$ with clock τ into the sequence $\perp e_1 e_2 \ldots$ with the same clock.

Table 6.2 shows the behavior of the **pre** operator in schematic form.

E	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
pre(E)	$\perp$	e_1	e_2	e_3	e_4	e_5	e_6	e_7

Table 6.2: The "*previous*" operator.

The initialization operator -> maps flows $\mathtt{E} = (e_1 e_2 \ldots, \tau)$ and $\mathtt{F} = (f_1 f_2 \ldots, \tau)$ to the flow $(e_1 f_2 f_3 \ldots, \tau)$. The -> operator only gives well-defined output as long as the input flows have the same clock. Table 6.3 shows the behavior of the -> operator in schematic form.

E	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
F	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
E -> F	e_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8

Table 6.3: The "*followed by*" operator.

The expression **E when B** samples values from **E** when **B** is true. Here **E** and **B** must be on the same clock and **B** must be a boolean flow. The clock of the flow defined by **E when B** consists of those instants when **B** is true. Formally, if $\mathtt{E} = (e, \tau)$ and $\mathtt{B} = (b, \tau)$, where $e = e_1 e_2 \ldots$ and $b = b_1 b_2 \ldots$, then **E when B** $= (e$ **when** $b, [b\tau])$, where e **when** b is the sequence $e_{i_1} e_{i_2} \ldots$ such that the numbers i_j are exactly the ones in increasing order for which b_{i_j} is true.

The **current** operator performs up-sampling, or interpolation, of a flow. For $\mathtt{E} = (e, [b\tau])$, **current(E)** $= (\mathbf{cur}(e, b), \tau')$, where $\mathbf{cur}(e, b)$ is the sequence e' for which

$$e'_i = \begin{cases} e_i & \text{if } b_i \text{ is true} \\ e'_{i-1} & \text{if } b_i \text{ is false} \end{cases}$$

Note that, according to the above recursive definition of e', $e'_0 = \perp$, by definition. As to the sequence τ',

$$\tau' = \begin{cases} \tau & \text{if } \tau \text{ is not empty} \\ [b] & \text{if } \tau \text{ is empty} \end{cases}$$

Table 6.4 shows the behavior of the **when** and **current** operators in schematic form.

E	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
B	*false*	*true*	*true*	*false*	*false*	*true*	*false*	*true*
Y = E when B		e_2	e_3			e_6		e_8
Z = current Y	$\perp$	e_2	e_3	e_3	e_3	e_6	e_6	e_8

Table 6.4: Sampling and Interpolating.

LUSTRE program is a finite sequence of *nodes* which consist of a declaration of input/output variables and a set of equations defining the output flows. The following node is the standard example how to define the basic clock counter (**COUNTER**) and its application in defining the regular clock which ticks on every third tick of the basic clock (**REGULAR_CLOCK_3**).

```
node COUNTER(val_init, val_incr: int; reset: bool) returns(n: int);
let
   n = val_init -> if reset then val_init else pre(n) + val_incr;
tel.

node REGULAR_CLOCK_3 () returns (clock_3: bool);
var n_3: int;
let
   n_3 = COUNTER(1, 1, pre(n_3) = 3);
   clock_3 = if (n_3 = 1) then true else false;
tel.
```

5.2 The Algebra of Schemes with Multiple Regular Clocks

In this subsection, motivated by the clock analysis of LUSTRE, we develop the algebra $\mathrm{Sf}_s(\Sigma)$ of synchronous schemes with multiple regular clocks, i.e., clocks that tick every first, second, third etc. instant of the basic clock. The arbitrary clocks are intentionally omitted since the issue becomes technically too complex.

DEFINITION 6.1 The algebra $\mathrm{Sf}_s(\Sigma)$ of generalized synchronous schemes consists of:

- **Objects**: (S, n) of sort $p \to q$, $S : p \to q$ in $\mathrm{Sf}(\Sigma)$ and $n \in \mathbb{N}$.

(S, n) stands for generalized scheme. Each input signal is repeated n times and outputs are read in $kn + 1$ cycles only, where $k = 0, 1, 2, \ldots$

- **Constants**: $1 = (1, 1), x = (x, 1), 0 = (0, 1), \varepsilon = (\varepsilon, 1), 0_1 = (0_1, 1)$
- **Operations**:
  1. *Composition*: $(f, m) \cdot (g, n) = (\text{SLOW}_{n'}(f) \cdot \text{SLOW}_{m'}(g), \text{lcm}(m, n))$
     if $f : p \to q, g : q \to r$, where $\text{lcm}(m, n)$ is the least common multiple of m and n with $m' = \frac{\text{lcm}(m,n)}{n}$, $n' = \frac{\text{lcm}(m,n)}{m}$ and $\text{SLOW}_c(S)$ is the c-slow of S.
  2. *Sum*: $(f, m) + (g, n) = (\text{SLOW}_{n'}(f) + \text{SLOW}_{m'}(g), \text{lcm}(m, n))$
     if $f : p_1 \to q_1, g : p_2 \to q_2$.
  3. *Feedback*: $\uparrow(f, n) = (\uparrow_n f, n)$
     if $f : 1 + p \to 1 + q$, where $\uparrow_n$ means feedback with interjecting n registers.

THEOREM 6.2 *The algebra* $\text{Sf}_s(\Sigma)$ *satisfies scheme identities* RF.

PROOF.

M1 $(f, x) \cdot ((g, y) \cdot (h, z)) = ((f, x) \cdot (g, y)) \cdot (h, z)$
if $f : p \to q$, $g : q \to r$, $h : r \to s$ and $x, y, z \in \mathbf{N}$

$$
\begin{aligned}
& (f, x) \cdot ((g, y) \cdot (h, z)) \\
= \; & (f, x) \cdot (\text{SLOW}_{\frac{\text{lcm}(y,z)}{y}}(g) \cdot \text{SLOW}_{\frac{\text{lcm}(y,z)}{z}}(h), \text{lcm}(y, z)) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(x,\text{lcm}(y,z))}{x}}(f) \cdot \text{SLOW}_{\frac{\text{lcm}(x,\text{lcm}(y,z))}{\text{lcm}(y,z)}}(\text{SLOW}_{\frac{\text{lcm}(y,z)}{y}}(g) \cdot \text{SLOW}_{\frac{\text{lcm}(y,z)}{z}}(h)), \\
& \text{lcm}(x, \text{lcm}(y, z))) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(x,\text{lcm}(y,z))}{x}}(f) \cdot (\text{SLOW}_{\frac{\text{lcm}(x,\text{lcm}(y,z))}{y}}(g) \cdot \text{SLOW}_{\frac{\text{lcm}(x,\text{lcm}(y,z))}{z}}(h)), \\
& \text{lcm}(x, \text{lcm}(y, z))) \\
= \; & (\text{SLOW}_x(\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x,y),z)}{x^2}}(f) \cdot \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x,y),z)}{xy}}(g)) \cdot \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x,y),z)}{z}}(h), \\
& \text{lcm}(\text{lcm}(x, y), z)) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x,y),z)}{x^2}}(f) \cdot \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x,y),z)}{xy}}(g), \frac{\text{lcm}(\text{lcm}(x, y), z))}{x}) \cdot (h, z)
\end{aligned}
$$

$$= ((f,x)\cdot(g,y))\cdot(h,z)$$

M2 $(f,x)+((g,y)+(h,z)) = ((f,x)+(g,y))+(h,z)$
if $f : p_1 \to q_1$, $g : p_2 \to q_2$, $h : p_3 \to q_3$ and $x, y, z \in \mathbb{N}$

$$\begin{aligned}
& (f,x)+((g,y)+(h,z)) \\
= \;& (f,x)+(\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(g)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(h),\mathrm{lcm}(y,z)) \\
= \;& (\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,\mathrm{lcm}(y,z))}{x}}(f)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,\mathrm{lcm}(y,z))}{\mathrm{lcm}(y,z)}}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(g)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(h)), \\
& \mathrm{lcm}(x,\mathrm{lcm}(y,z))) \\
= \;& (\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,\mathrm{lcm}(y,z))}{x}}(f)+(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,\mathrm{lcm}(y,z))}{y}}(g)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,\mathrm{lcm}(y,z))}{z}}(h)), \\
& \mathrm{lcm}(x,\mathrm{lcm}(y,z))) \\
= \;& (\mathrm{SLOW}_{x}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z)}{x^2}}(f)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z)}{xy}}(g))+\mathrm{SLOW}_{\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z)}{z}}(h), \\
& \mathrm{lcm}(\mathrm{lcm}(x,y),z)) \\
= \;& (\mathrm{SLOW}_{\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z)}{x^2}}(f)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z)}{xy}}(g),\frac{\mathrm{lcm}(\mathrm{lcm}(x,y),z))}{x})+(h,z) \\
= \;& ((f,x)+(g,y))+(h,z)
\end{aligned}$$

M3 $(p,1)\cdot(f,y) = (f,y)\cdot(q,1)$ and $y \in \mathbb{N}$ if $f : p \to q$

$$\begin{aligned}
(p,1)\cdot(f,y) &= (\mathrm{SLOW}_y(p)\cdot f, y) \\
&= (p\cdot f, y) \\
&= (f\cdot q, y) \\
&= (f\cdot \mathrm{SLOW}_y(q), y) \\
&= (f,y)\cdot(q,1)
\end{aligned}$$

M4 $(f,x)+(0,1) = (0,1)+(f,x)$ if $f : p \to q$ and $x \in \mathbb{N}$

$$\begin{aligned}
(f,x)+(0,1) &= (f+\mathrm{SLOW}_x(0), x) \\
&= (f+0, x) \\
&= (0+f, x)
\end{aligned}$$

$$= (\text{SLOW}_x(0) + f), x)$$
$$= (0,1) + (f, x)$$

M5 $((f_1, x_1) \cdot (g_1, y_1)) + ((f_2, x_2) \cdot (g_2, y_2)) = ((f_1, x_1) + (f_2, x_2)) \cdot ((g_1, y_1) + (g_2, y_2))$
if $f_i : p_i \to q_i$, $g_i : q_i \to r_i$, $i = 1, 2$ and $x_1, y_1, x_2, y_2 \in \mathbb{N}$

$$\begin{aligned}
& ((f_1, x_1) \cdot (g_1, y_1)) + ((f_2, x_2) \cdot (g_2, y_2)) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(x_1,y_1)}{x_1}}(f_1) \cdot \text{SLOW}_{\frac{\text{lcm}(x_1,y_1)}{y_1}}(g_1), \text{lcm}(x_1, y_1)) + \\
& (\text{SLOW}_{\frac{\text{lcm}(x_2,y_2)}{x_2}}(f_2) \cdot \text{SLOW}_{\frac{\text{lcm}(x_2,y_2)}{y_2}}(g_2), \text{lcm}(x_2, y_2)) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{\text{lcm}(x_1,y_1)}}(\text{SLOW}_{\frac{\text{lcm}(x_1,y_1)}{x_1}}(f_1) \cdot \text{SLOW}_{\frac{\text{lcm}(x_1,y_1)}{y_1}}(g_1)) + \\
& \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{\text{lcm}(x_2,y_2)}}(\text{SLOW}_{\frac{\text{lcm}(x_2,y_2)}{x_2}}(f_2) \cdot \text{SLOW}_{\frac{\text{lcm}(x_2,y_2)}{y_2}}(g_2)), \\
& \text{lcm}(\text{lcm}(x_1, y_1), \text{lcm}(x_2, y_2))) \\
= \; & ((\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{x_1}}(f_1) \cdot \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{y_1}}(g_1)) + \\
& (\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{x_2}}(f_2) \cdot \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,y_1),\text{lcm}(x_2,y_2))}{y_2}}(g_2)), \\
& \text{lcm}(\text{lcm}(x_1, y_1), \text{lcm}(x_2, y_2))) \\
= \; & ((\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{x_1}}(f_1) + \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{x_2}}(f_2)) \cdot \\
& (\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{y_1}}(g_1) + \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{y_2}}(g_2)), \\
& \text{lcm}(\text{lcm}(x_1, x_2), \text{lcm}(y_1, y_2))) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{\text{lcm}(x_1,x_2)}}(\text{SLOW}_{\frac{\text{lcm}(x_1,x_2)}{x_1}}(f_1) + \text{SLOW}_{\frac{\text{lcm}(x_1,x_2)}{x_2}}(f_2)) \cdot \\
& \text{SLOW}_{\frac{\text{lcm}(\text{lcm}(x_1,x_2),\text{lcm}(y_1,y_2))}{\text{lcm}(y_1,y_2)}}(\text{SLOW}_{\frac{\text{lcm}(y_1,y_2)}{y_1}}(g_1) + \text{SLOW}_{\frac{\text{lcm}(y_1,y_2)}{y_2}}(g_2)), \\
& \text{lcm}(\text{lcm}(x_1, x_2), \text{lcm}(y_1, y_2))) \\
= \; & (\text{SLOW}_{\frac{\text{lcm}(x_1,x_2)}{x_1}}(f_1) + \text{SLOW}_{\frac{\text{lcm}(x_1,x_2)}{x_2}}(f_2), \text{lcm}(x_1, x_2)) \cdot \\
& (\text{SLOW}_{\frac{\text{lcm}(y_1,y_2)}{y_1}}(g_1) + \text{SLOW}_{\frac{\text{lcm}(y_1,y_2)}{y_2}}(g_2), \text{lcm}(y_1, y_2)) \\
= \; & ((f_1, x_1) + (f_2, x_2)) \cdot ((g_1, y_1) + (g_2, y_2))
\end{aligned}$$

P $(f_1, x_1) + (f_2, x_2) = x\#((p_1, 1), (p_2, 1)) \cdot ((f_2, x_2) + (f_1, x_1)) \cdot x\#((q_2, 1), (q_1, 1))$
if $f_i : p_i \to q_i$, $i = 1, 2$ and $x_1, x_2 \in \mathbb{N}$

$$
\begin{aligned}
& (f_1, x_1) + (f_2, x_2) \\
= \; & (\mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2), \mathrm{lcm}(x_1, x_2)) \\
= \; & (x\#(p_1, p_2) \cdot (\mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1)) \cdot x\#(q_2, q_1), \mathrm{lcm}(x_1, x_2)) \\
= \; & (x\#((p_1, 1), (p_2, 1)) \cdot (\mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2) + \\
& \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1)) \cdot x\#((q_2, 1), (q_1, 1)), \mathrm{lcm}(x_1, x_2)) \\
= \; & x\#((p_1, 1), (p_2, 1)) \cdot ((f_2, x_2) + (f_1, x_1)) \cdot x\#((q_2, 1), (q_1, 1))
\end{aligned}
$$

D1 $((\varepsilon, 1) + (1, 1)) \cdot (\varepsilon, 1) = ((1, 1) + (\varepsilon, 1)) \cdot (\varepsilon, 1)$

Follows directly from SF and the definition of constants.

D2 $(x, 1) \cdot (\varepsilon, 1) = (\varepsilon, 1)$

Follows directly from SF and the definition of constants.

D3 $((1, 1) + (0_1, 1)) \cdot (\varepsilon, 1) = (1, 1)$

Follows directly from SF and the definition of constants.

S1 $\uparrow((f_1, x_1) + (f_2, x_2)) = \uparrow(f_1, x_1) + (f_2, x_2)$
if $f : 1 + p_1 \to 1 + q_1, f_2 : p_2 \to q_2$ and $x_1, x_2 \in \mathbb{N}$

$$
\begin{aligned}
& \uparrow((f_1, x_1) + (f_2, x_2)) \\
= \; & \uparrow(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2), \mathrm{lcm}(x_1, x_2)) \\
= \; & (\uparrow_{\mathrm{lcm}(x_1,x_2)}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2)), \mathrm{lcm}(x_1, x_2)) \\
= \; & ((\uparrow_{\mathrm{lcm}(x_1,x_2)} \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_1}}(f_1)) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(x_1,x_2)}{x_2}}(f_2), \mathrm{lcm}(x_1, x_2)) \\
= \; & (\uparrow_{x_1} f_1, x_1) + (f_2, x_2) \\
= \; & \uparrow(f_1, x_1) + (f_2, x_2)
\end{aligned}
$$

S2 $\uparrow^2(((x, 1) + (p, 1)) \cdot (f, c)) = \uparrow^2((f, c) \cdot ((x, 1) + (q, 1)))$
if $f : 2 + p \to 2 + q$ and $c \in \mathbb{N}$

$$
\begin{aligned}
& \uparrow^2(((x,1)+(p,1))\cdot(f,c)) \\
= \ & \uparrow^2((x+p)\cdot(f,c)) \\
= \ & \uparrow^2((\mathrm{SLOW}_c(x)+\mathrm{SLOW}_c(p))\cdot f,c) \\
= \ & (\uparrow_c^2((x+p)\cdot f,c) \\
= \ & (\uparrow_c^2(f\cdot(x+q)),c) \\
= \ & \uparrow^2(f\cdot(\mathrm{SLOW}_c(x)+\mathrm{SLOW}_c(q)),c) \\
= \ & \uparrow^2((f,c)\cdot((x,1)+(q,1)))
\end{aligned}
$$

S3 $\uparrow((f,x)\cdot((1,1)+(g,z)))=\uparrow(f,x)\cdot(g,z)$
if $f:1+p\to 1+q$, $g:q\to r$ and $x,z\in\mathbb{N}$

$$
\begin{aligned}
& \uparrow((f,x)\cdot((1,1)+(g,z))) \\
= \ & \uparrow((f,x)\cdot(\mathrm{SLOW}_z(1)+g),z)) \\
= \ & \uparrow(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{x}}(f)\cdot(\mathrm{SLOW}_{zz}(1)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{z}}(g)),\mathrm{lcm}(x,z)) \\
= \ & (\uparrow_{\mathrm{lcm}(x,z)}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{x}}(f)\cdot(1+\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{z}}(g))),\mathrm{lcm}(x,z)) \\
= \ & ((\uparrow_{\mathrm{lcm}(x,z)}\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{x}}(f))\cdot\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,z)}{z}}(g),\mathrm{lcm}(x,z)) \\
= \ & (\uparrow_x f,x)\cdot(g,z) \\
= \ & \uparrow(f,x)\cdot(g,z)
\end{aligned}
$$

S4 $\uparrow(((1,1)+(g,y))\cdot(f,z))=(g,y)\cdot\uparrow(f,z)$
if $f:1+q\to 1+r$, $g:p\to q$ and $y,z\in\mathbb{N}$

$$
\begin{aligned}
& \uparrow(((1,1)+(g,y))\cdot(f,z)) \\
= \ & \uparrow((\mathrm{SLOW}_y(1)+g,y)\cdot(f,z)) \\
= \ & \uparrow((\mathrm{SLOW}_{yz}(1)+\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(g))\cdot\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(f),\mathrm{lcm}(y,z)) \\
= \ & (\uparrow_{\mathrm{lcm}(y,z)}((1+\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(g))\cdot\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(f)),\mathrm{lcm}(y,z)) \\
= \ & (\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(g)\cdot(\uparrow_{\mathrm{lcm}(y,z)}\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(f)),\mathrm{lcm}(y,z))
\end{aligned}
$$

$$
\begin{aligned}
&= (g, y) \cdot (\uparrow_z f, z) \\
&= (g, y) \cdot \uparrow(f, z)
\end{aligned}
$$

S5 $\uparrow(1, 1) = (0, 1)$

Follows directly from SF and the definition of constants.

S6 $(\varepsilon, 1) \cdot (\bot, 1) = (\bot, 1) + (\bot, 1)$ where $\bot = \uparrow\varepsilon$

Follows directly from SF and the definition of constants.

S7 $\uparrow((f, x) \cdot ((\varepsilon, 1) + (q, 1))) = \uparrow^2(((\varepsilon, 1) + (p, 1)) \cdot (f, x))$
if $f : 1 + p \to 2 + q$ and $x \in \mathbb{N}$

$$
\begin{aligned}
&\uparrow((f, x) \cdot ((\varepsilon, 1) + (q, 1))) \\
= \;&\uparrow((f, x) \cdot (\varepsilon + q)) \\
= \;&\uparrow(f \cdot (\text{SLOW}_x(\varepsilon) + \text{SLOW}_x(q)), x) \\
= \;&\uparrow(f \cdot (\varepsilon + q), x) \\
= \;&(\uparrow_x(f \cdot (\varepsilon + q)), x) \\
= \;&(\uparrow_x^2((\varepsilon + p) \cdot f), x) \\
= \;&\uparrow^2((\text{SLOW}_x(\varepsilon) + \text{SLOW}_x(p)) \cdot f, x) \\
= \;&\uparrow^2(((\varepsilon, 1) + (p, 1)) \cdot (f, x))
\end{aligned}
$$

S8 $(0_1, 1) \cdot (\nabla, 1) = (0_1, 1)$ where $\nabla = \uparrow x$

Follows directly from SF and the definition of constants.

S9 $\uparrow((\varepsilon, 1) \cdot (\nabla, 1)^n) = (\bot, 1)$
where $(\nabla, 1)^n$ denotes the n-fold composite of $(\nabla, 1)$

Follows directly from SF and the definition of constants.

R $\uparrow^{p_1}((f, x) \cdot ((g, y) + (q_1, 1))) = \uparrow^{q_1}(((g, y) + (p_2, 1)) \cdot (f, x))$
if $f : p_1 + p_2 \to q_1 + q_2, g : q_1 \to p_1$ and $x, y \in \mathbb{N}$

$$
\begin{aligned}
& \uparrow^{p_1}((f,x)\cdot((g,y)+(q_1,1))) \\
= \;& \uparrow^{p_1}((f,x)\cdot(g+\mathrm{SLOW}_y(q_1),y)) \\
= \;& \uparrow^{p_1}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{x}}(f)\cdot(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{y}}(g)+\mathrm{SLOW}_{xy}(q_1)),\mathrm{lcm}(x,y)) \\
= \;& (\uparrow^{p_1}_{\mathrm{lcm}(x,y)}(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{x}}(f)\cdot(\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{y}}(g)+q_1)),\mathrm{lcm}(x,y)) \\
= \;& (\uparrow^{q_1}_{\mathrm{lcm}(x,y)}((\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{y}}(g)+p_2)\cdot\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{x}}(f)),\mathrm{lcm}(x,y)) \\
= \;& \uparrow^{q_1}((\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{y}}(g)+\mathrm{SLOW}_{xy}(p_2))\cdot\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{x}}(f),\mathrm{lcm}(x,y)) \\
= \;& \uparrow^{q_1}(((g,y)+(p_2,1))\cdot(f,x)) \;\blacksquare
\end{aligned}
$$

DEFINITION 6.3 We define the *L-eqivalence* of generalized synchronous schemes as follows. Let (F,m) and (G,n) be generalized synchronous schemes. Suppose that for every sufficiently old configuration c of (F,m), there exists a configuration c' of (G,n) such that when (F,m) is started in configuration c with each input signal repeated m times and (G,n) is started in configuration c' with each input signal repeated n times, the two schemes exhibit the same behavior, i.e., the outputs in cycles $km+1$ and $kn+1$, $k=0,1,2,\ldots$ are the same. Then scheme (G,n) can *simulate* (F,m). If two generalized synchronous systems can simulate each other, then they are *L-equivalent*.

Unfortunately, not all (S,n) schemes are suitable. Consider the following example:

Basic clock	1	2	3	4	5	6	7	8	9	10	11
Input	x_1	x_1	x_2	x_2	x_3	x_3	x_4	x_4	x_5	x_5	x_6
$(\nabla,2)$ Output	$\perp$		x_1		x_2		x_3		x_4		x_5
$(\nabla^2,2)$ Output	$\perp$		x_1		x_2		x_3		x_4		x_5
$(\nabla^3,2)$ Output	$\perp$		$\perp$		x_1		x_2		x_3		x_4

Table 6.5: The behavior of $(\nabla,2)$, $(\nabla^2,2)$ and $(\nabla^3,2)$ during first eleven pulsations.

Then

$$(\nabla,2) \approx (\nabla,2)$$

$$(\nabla,2) \approx (\nabla^2,2) \text{ but}$$

$$(\nabla,2)\cdot(\nabla,2) = (\nabla^2,2) \not\approx (\nabla^3,2) = (\nabla,2)\cdot(\nabla^2,2)$$

where $\approx$ denotes the L-equivalence relation. In other words, the L-equivalence is not preserved by all (S,n) schemes. For this reason we introduce the following restriction to $\mathrm{Sf}_s(\Sigma)$.

DEFINITION 6.4 The algebra $\mathfrak{Sf}_s(\Sigma)$ consists of all (S,n) schemes such that S is strong retiming equivalent to some appropriate n-slow SΣ-scheme S'.

It is now obvious that $(\nabla,2) \notin \mathfrak{Sf}_s(\Sigma)$ since there is no 2-slow scheme which is Leiserson equivalent to $(\nabla,2)$. On the other hand, $(\nabla^2,2) \in \mathfrak{Sf}_s(\Sigma)$ since $(\nabla^2,2) \sim \mathrm{SLOW}_2(\nabla) = \nabla^2$.

THEOREM 6.5 *The characteristic function*

$$c(S,n) = \begin{cases} 1 & \text{if } (S,n) \in \mathfrak{Sf}_s(\Sigma) \\ 0 & \text{otherwise} \end{cases}$$

is a recursive function.

PROOF. (a) *Biaccessible schemes.* Recall from [Bloom and Tindel, 1979] that a flowchart scheme is *biaccessible* if it is accessible and every vertex is the starting point of some path whose endpoint is an exit vertex. An SΣ-scheme S is biaccessible if the FΣ-scheme $fl(S)$ is such. In other words, S is biaccessible if it is accessible and every vertex can be reached from some input channel by a directed path.

Let (S,n) be a biaccessible generalized scheme. If $(S,n) \in \mathfrak{Sf}_s(\Sigma)$ then there exists SΣ-scheme S' such that $S \sim nS'$. According to Theorem 6.1.5 [Bartha, 1994],

$$S \sim nS' \Leftrightarrow S_{max} \sim_s nS'_{max}$$

where S_{max} and S'_{max} are SΣ-schemes such that $R_{max} : S \to S_{max}$, $R_{max} : nS' \to nS'_{max}$ and

$$R_{max}(v) = \min\{w(p) \mid p \text{ is an input path leading to } v\}$$

is a legal retiming vector. Since $S_{max} \sim_s nS'_{max}$, S_{max} is an n-slow of some SΣ-scheme F, i.e., $S_{max} = nF$. Therefore, in order to decide whether $(S, n) \in \mathfrak{Sf}_s(\Sigma)$ or not, it is sufficient to compute total weights of all entry-to-exit paths $w(p)$ and directed cycles $w(z)$ in S, and check ratios $\frac{w(p)}{n}$ and $\frac{w(z)}{n}$. If these ratios are integers then $(S, n) \in \mathfrak{Sf}_s(\Sigma)$, otherwise $(S, n) \notin \mathfrak{Sf}_s(\Sigma)$.

(b) *The general case.* First of all, observe that *ground* schemes, i.e., schemes without input channels, belong to $\mathfrak{Sf}_s(\Sigma)$. This is indeed true by Theorem 4.6, since in infinite trees without variables it is always possible to rearrange the registers (∇-nodes) from any regular pattern to any other regular pattern by using the transformation $\mathcal{T}_n$. See also Figure 6.1. Hence, given generalized scheme (S, n), it is sufficient to isolate ground subschemes and test only the biaccessible part. ■

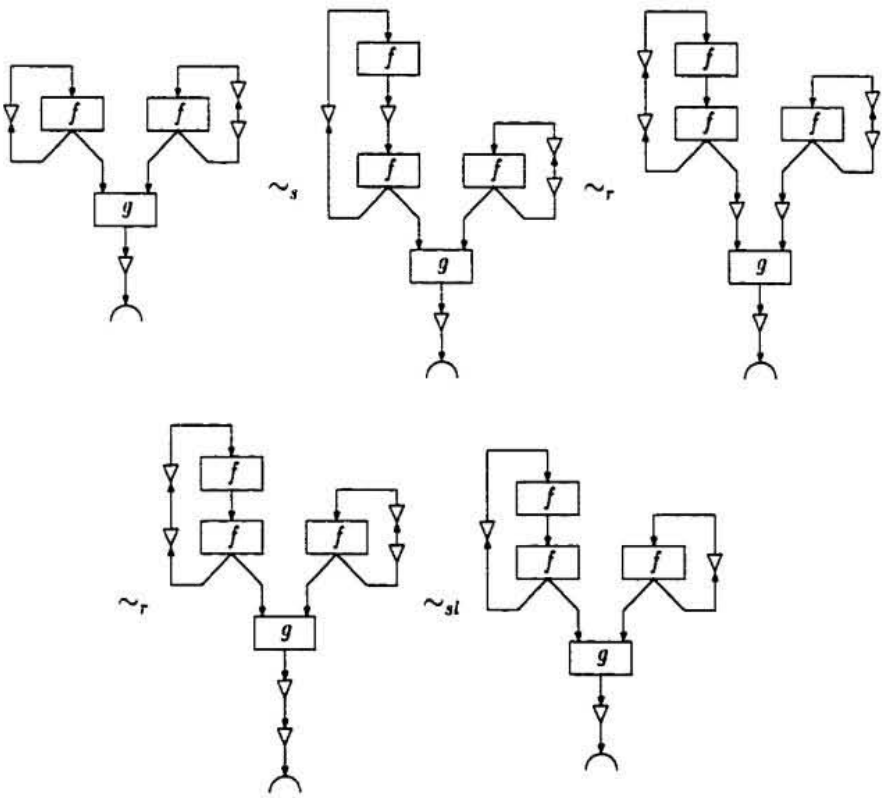


Figure 6.1: Ground schemes belong to $\mathfrak{Sf}_s(\Sigma)$.

THEOREM 6.6 *The algebra* $\mathfrak{Sf}_s(\Sigma)$ *is a subalgebra of* $\mathrm{Sf}_s(\Sigma)$.

PROOF.

1. *Composition*

   Let $(F,m),(G,n) \in \mathfrak{Sf}_s(\Sigma)$. Then $(F,m) \sim mF'$ and $(G,n) \sim nG'$ for some appropriate schemes F' and G'. We have:

   $(F,m) \cdot (G,n) =$

   $$(\mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{m}}(F) \cdot \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{n}}(G), \mathrm{lcm}(m,n)) \sim \mathrm{SLOW}_{\mathrm{lcm}(m,n)}(F' \cdot G')$$

   Hence $(F,m) \cdot (G,n) \in \mathfrak{Sf}_s(\Sigma)$.

2. *Sum*

   Let $(F,m),(G,n) \in \mathfrak{Sf}_s(\Sigma)$. Then $(F,m) \sim mF'$ and $(G,n) \sim nG'$ for some appropriate schemes F' and G'. We have:

   $(F,m) + (G,n) =$

   $$(\mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{m}}(F) + \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{n}}(G), \mathrm{lcm}(m,n)) \sim \mathrm{SLOW}_{\mathrm{lcm}(m,n)}(F' + G')$$

   Hence $(F,m) + (G,n) \in \mathfrak{Sf}_s(\Sigma)$.

3. *Feedback*

   Let $(F,m) \in \mathfrak{Sf}_s(\Sigma)$. Then $(F,m) \sim mF'$ for some appropriate scheme F'. We have:

   $$\uparrow(F,m) = (\uparrow_m F, m) \sim \mathrm{SLOW}_m(\uparrow F')$$

   Hence $\uparrow(F,m) \in \mathfrak{Sf}_s(\Sigma)$. ■

COROLLARY 6.7 $\mathfrak{Sf}_s(\Sigma)$ satisfies scheme identities RF.

PROOF. Follows directly from Theorem 6.2 and Theorem 6.6. ■

DEFINITION 6.8 We define the relation Θ_L on $\mathfrak{Sf}_s(\Sigma)$ as follows. Let $(F,m),(G,n) \in \mathfrak{Sf}_s(\Sigma)$. Then:

$$(F,m) \equiv (G,n)(\Theta_L) \Leftrightarrow \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{m}}(F) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{n}}(G)$$

THEOREM 6.9 *The relation Θ_L is a congruence relation of $\mathfrak{Sf}_s(\Sigma)$.*

PROOF. (a) Θ_L is an equivalence relation.

1. *Reflexivity*

   Let $(F, m) \in \mathfrak{Sf}_s(\Sigma)$. Since $F \sim F$ we have $(F, m) \equiv (F, m)(\Theta_L)$.

2. *Symmetry*

   Let $(F, m), (G, n) \in \mathfrak{Sf}_s(\Sigma)$ and $(F, m) \equiv (G, n)(\Theta_L)$. Then

   $(F, m) \equiv (G, n)(\Theta_L) \Rightarrow$

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{m}}(F) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{n}}(G) \Rightarrow$

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{n}}(G) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(m,n)}{m}}(F) \Rightarrow$

   $(G, n) \equiv (F, m)(\Theta_L)$

3. *Transitivity*

   Let $(F, x), (G, y), (H, z) \in \mathfrak{Sf}_s(\Sigma)$ and $(F, x) \equiv (G, y)(\Theta_L)$, $(G, y) \equiv (H, z)(\Theta_L)$. Then

   $(F, x) \equiv (G, y)(\Theta_L)$ and $(G, y) \equiv (H, z)(\Theta_L) \Rightarrow$

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{x}}(F) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y)}{y}}(G)$ and $\mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{y}}(G) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(y,z)}{z}}(H) \Rightarrow$

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{x}}(F) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{y}}(G)$ and

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{y}}(G) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{z}}(H) \Rightarrow$

   $\mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{x}}(F) \sim \mathrm{SLOW}_{\frac{\mathrm{lcm}(x,y,z)}{z}}(H) \Rightarrow$

   $(F, x) \equiv (H, z)(\Theta_L)$

(b) Θ_L satisfies the substitution property.

1. *Composition*

   Let $(F_1, m_1) \equiv (G_1, n_1)(\Theta_L)$ and $(F_2, m_2) \equiv (G_2, n_2)(\Theta_L)$. Then

   $(F_1, m_1) \cdot (F_2, m_2) =$

   $(\mathrm{SLOW}_{\frac{a}{m_1}}(F_1) \cdot \mathrm{SLOW}_{\frac{a}{m_2}}(F_2), a) \equiv_{\Theta_L}$

   $(\mathrm{SLOW}_{\frac{\mathrm{lcm}(a,b)}{m_1}}(F_1) \cdot \mathrm{SLOW}_{\frac{\mathrm{lcm}(a,b)}{m_2}}(F_2), \mathrm{lcm}(a, b)) =$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{c}}\text{SLOW}_{\frac{c}{m_1}}(F_1) \cdot \text{SLOW}_{\frac{\text{lcm}(a,b)}{d}}\text{SLOW}_{\frac{d}{m_2}}(F_2), \text{lcm}(a,b)) \sim$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{c}}\text{SLOW}_{\frac{c}{n_1}}(G_1) \cdot \text{SLOW}_{\frac{\text{lcm}(a,b)}{d}}\text{SLOW}_{\frac{d}{n_2}}(G_2), \text{lcm}(a,b)) =$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{n_1}}(G_1) \cdot \text{SLOW}_{\frac{\text{lcm}(a,b)}{n_2}}(G_2), \text{lcm}(a,b)) \equiv_{\Theta_L}$

$(\text{SLOW}_{\frac{b}{n_1}}(G_1) \cdot \text{SLOW}_{\frac{b}{n_2}}(G_2), \text{lcm}(n_1,n_2)) = (G_1,n_1) \cdot (G_2,n_2)$

where $a = \text{lcm}(m_1,m_2)$, $b = \text{lcm}(n_1,n_2)$, $c = \text{lcm}(m_1,n_1)$ and $d = \text{lcm}(m_2,n_2)$.

2. *Sum*

Let $(F_1,m_1) \equiv (G_1,n_1)(\Theta_L)$ and $(F_2,m_2) \equiv (G_2,n_2)(\Theta_L)$. Then

$(F_1,m_1) + (F_2,m_2) =$

$(\text{SLOW}_{\frac{\text{lcm}(a)}{m_1}}(F_1) + \text{SLOW}_{\frac{a}{m_2}}(F_2), a) \equiv_{\Theta_L}$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{m_1}}(F_1) + \text{SLOW}_{\frac{\text{lcm}(a,b)}{m_2}}(F_2), \text{lcm}(a,b)) =$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{c}}\text{SLOW}_{\frac{c}{m_1}}(F_1) + \text{SLOW}_{\frac{\text{lcm}(a,b)}{d}}\text{SLOW}_{\frac{d}{m_2}}(F_2), \text{lcm}(a,b)) \sim$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{c}}\text{SLOW}_{\frac{c}{n_1}}(G_1) + \text{SLOW}_{\frac{\text{lcm}(a,b)}{d}}\text{SLOW}_{\frac{d}{n_2}}(G_2), \text{lcm}(a,b)) =$

$(\text{SLOW}_{\frac{\text{lcm}(a,b)}{n_1}}(G_1) + \text{SLOW}_{\frac{\text{lcm}(a,b)}{n_2}}(G_2), \text{lcm}(a,b)) \equiv_{\Theta_L}$

$(\text{SLOW}_{\frac{b}{n_1}}(G_1) + \text{SLOW}_{\frac{b}{n_2}}(G_2), b) = (G_1,n_1) + (G_2,n_2)$

where $a = \text{lcm}(m_1,m_2)$, $b = \text{lcm}(n_1,n_2)$, $c = \text{lcm}(m_1,n_1)$, and $d = \text{lcm}(m_2,n_2)$.

3. *Feedback*

Let $(F,m) \equiv (G,n)(\Theta_L)$. Then

$\uparrow(F,m) = (\uparrow_m F, m) \equiv_{\Theta_L} \text{SLOW}_{\frac{\text{lcm}(m,n)}{m}}(\uparrow_m F, m) \sim$

$\text{SLOW}_{\frac{\text{lcm}(m,n)}{n}}(\uparrow_n G, n) \equiv_{\Theta_L} (\uparrow_n G, n) = \uparrow(G,n)$ ■

THEOREM 6.10 *Two generalized synchronous schemes (S_1,n_1) and (S_2,n_2) are L-equivalent if and only if they are Θ_L equivalent.*

PROOF. Follows directly from Definition 6.8, Theorem 6.9 and Theorem 4.10. ■

Conclusion

The notion of a synchronous system allowed the introduction of transformations useful for the design and optimization of such systems: slowdown and retiming. Retiming is important transformation which can be used to optimize clocked circuits by relocating registers so as to reduce combinational rippling. It has an interesting property that if two systems can be joined by series of primitive retiming steps, i.e., shifting one layer of registers from one side of a functional element to the other, then those two systems exhibit the same behavior, as proved in [Leiserson and Saxe, 1983a]. Concerning slowdown transformation, the main adventage of c-slow circuits, i.e, circuits obtained from the original circuit by multiplying all the register counts by some positive integer c, is that they can be retimed to have shorter clock periods than any retimed version of the original. Slowdown transformation does not preserve the equivalence of schemes in the strictest sense. The c-slow circuits perform the same computation as original circuit, but take c times as many clock ticks and communicate with the host only on every cth clock tick. The impact of slowdown on the behavior of synchronous schemes is the following: not any two synchronous schemes are retiming equivalent. However, for two synchronous schemes that cannot be directly retimed to each other, there might be appropriate slowdown transformations such that, after these transformations are applied, one gets synchronous schemes that are already retiming equivalent. A new relation is obtained by taking the join of the retiming and slowdown relations and is called slowdown retiming equivalence. One of the contributions of Chapter 3 is the proof of the fact that the slowdown retiming equivalence relation is decidable for synchronous schemes. Two synchronous schemes are said to be strongly equivalent if they exhibit the same behavior under all interpretations, that is if they can be unfolded into exactly the same tree. The new equivalence relation can be obtained as a join of strong and retiming equivalence. In [Bartha, 1994] it was proved that strong retiming equivalence relation is decidable for synchronous schemes. The next

major contribution of Chapter 3 is the proof that the strong slowdown retiming equivalence relation, which is the join of strong, slowdown and retiming equivalence, is also decidable.

The concept of equivalency of synchronous systems was introduced in [Leiserson and Saxe, 1983a] in a rather intuitive and informal manner. The most important contribution of the Thesis is the proof in Chapter 4 that two notions, Leiserson equivalence and strong retiming equivalence, coincide. The very same notion of the equivalency of synchronous schemes has also been characterized in terms of finite state top-down tree transducers.

The syntax of a synchronous scheme is specified by a directed, labelled, edge-weighted multigraph. The semantics of a synchronous scheme can then be specified by the algebraic structures called feedback theories. Synchronous schemes have been axiomatized equationally in [Bartha, 1987] capturing their strong behavior. The major contribution of Chapter 5 is the introduction of retiming identities and the construction of the feedback theory capturing the strong retiming behavior of synchronous schemes.

The motivation for the results of Chapter 6 stems from two sources: multiphase clocking (clocking schemes that use more phases and consequently offer more flexibility in adjusting the relative timings of the functional elements) has been left as a further topic in [Leiserson and Saxe, 1983a] and the notion of multiple clocks defined in terms of boolean-valued flows of the synchronous dataflow programming language LUSTRE. The major contribution of Chapter 6 is the construction of the general algebra of multiclocked schemes. For simplicity, only schemes with multiple regular clocks, i.e., clocks that tick every first, second, third etc. instant of the basic clock, have been considered. The arbitrary clocks are intentionally omitted since the issue becomes too complex technically. Also, the intuitive notion of L-equivalency between two generalized schemes is introduced and shown to coincide with the formal characterization of Θ_L equivalency.

References

[1] ARNOLD, A. AND DAUCHET, M. (1978, 1979), Theéorie des magmoïdes, *RAIRO Inform. Theór. Appl.* **12**, 235-257 and **13**, 135-154.

[2] BARTHA, M. (1987), An equational axiomatization of systolic systems, *Theoretical Computer Science*, **55**, 265-289.

[3] BARTHA, M. (1989), Interpretations of synchronous flowchart schemes, in "Proceedings, 7th Conference on the Fundamentals of Computation Theory, Szeged" (Edited by J. Csirik, J. Demetrovics and F. Gécseg), *Lecture Notes in Computer Science*, **380**, 25-34, Springer-Verlag, Berlin.

[4] BARTHA, M. (1992a), Foundations of a theory of synchronous systems, *Theoretical Computer Science*, **100**, 325-346, Elsevier.

[5] BARTHA, M. (1992b), An algebraic model of synchronous systems, *Information and Computation*, **97**, 97-131.

[6] BARTHA, M. AND GOMBÁS, É. (1994), Strong retiming equivalence of synchronous systems, Technical Report No. 9404, MUN.

[7] BERRY, G. (1989), Real time programming: Special purpose or general purpose languages, in *IFIP World Computer Congress*, San Francisco.

[8] BILSTEIN, J. AND DAMM, W. (1981), Top-down tree-transducers for infinite trees, in "Proceedings, 6th Colloquium on Trees in Algebra and Programming, Genoa" (Edited by E. Astesiano and C. Böhm), *Lecture Notes in Computer Science*, **112**, 97-131, Springer-Verlag, Berlin.

[9] BLOOM, S. L. AND ÉSIK, Z. (1993), Iteration Theories, The Equational Logic of Iterative Processes, Springer-Verlag, Berlin.

[10] BLOOM, S. L. AND TINDELL, R. (1979), Algebraic and graph theoretic characterizations of structured flowchart schemes, *Theoretical Computer Science*, **9**, 265-286, Elsevier.

[11] ELGOT, C. C. (1975), Monadic computations and iterative algebraic theories, in "Logic Colloquium '73, Studies in Logic and the Foundations of Mathematics" (H. E. Rose and J. C. Shepherdson, Eds.), 175-230, North-Holland, Amsterdam.

[12] ELGOT, C. C., BLOOM, S. L. AND TINDELL, R. (1978), On the algebraic structure of rooted trees, *Journal of Computer and System Sciences*, **16**, 228-242.

[13] ELGOT, C. C. AND SHEPHERDSON, J. C. (1979), A semantically meaningfull characterization of reducible flowchart schemes, *Theoretical Computer Science*, **8**(3), 325-357.

[14] ELGOT, C. C. AND SHEPHERDSON, J. C. (1980), An Equational Axiomatization of the Algebra of Reducible Flowchart Schemes, IBM Research Report RC 8221.

[15] ENGELFRIET, J. (1975), Bottom-up and Top-down Tree Transformations - A Comparison, *Mathematical Systems Theory*, **9**(3), 198-231.

[16] ÉSIK, Z. (1980), Identities in iterative and rational theories, *Computational Linguistics and Computer Languages*, **14**, 183-207.

[17] GRÄTZER, G. (1968, 1979), Universal Algebra, Springer-Verlag, Berlin.

[18] HALBWACHS, N., CASPI, P., RAYMOND, P. AND PILAUD, D. (1991), The synchronous dataflow programming language LUSTRE, *Proceedings of the IEEE*, **79**(9), 1305-1320.

[19] HAREL, D. AND PNUELI, A. (1985), On the development of reactive systems, in "Logic and Models of Concurrent Systems", NATO, Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems, Springer-Verlag.

[20] JENSEN, T. P. (1995), Clock Analysis of Synchronous Dataflow Programs, in "Proc. of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation", San Diego CA, 156-167.

[21] JOHNSON, D. B. (1975), Finding all the elementary circuits of a directed graph, SIAM, *Journal on Computing*, **4**(1), 77-84.

[22] KUNG, H. T. AND LEISERSON, C. E. (1978), Systolic arrays for VLSI, in "Sparse Matrix Proceedings", SIAM, Philadelphia, 256-282.

[23] KUNG, S. Y. (1988), VLSI array processors, Prentice Hall, Englewood Cliffs, N.J.

[24] LAWVERE, F. W. (1963), Functional semantics of algebraic theories, *Proc. Nat. Acad. Sci. U.S.A.*, **50**(5), 869-872.

[25] LEISERSON, C. E. (1982), Area-efficient VLSI Computation, ACM-MIT Press Doctoral Dissertation Award Series 1, ACM-MIT, New York.

[26] LEISERSON, C. E. AND SAXE, J. B. (1983a), Optimizing Synchronous Systems, *Journal of VLSI and Computer Systems*, **1**(1), 41-67.

[27] LEISERSON, C. E., ROSE, F. M. AND SAXE, J. B. (1983b), Optimizing Synchronous Circuitry by Retiming, Proceedings of 3rd Caltech Conference on VLSI (Edited by Randal Bryant), Computer Science Press, 87-116.

[28] MACLANE. S. (1971), Categories for Working Mathematician, Springer-Verlag, Berlin.

[29] MURATA, T. (1977), Circuit theoretic analysis and synthesis of marked graphs. *IEEE Trans. on Circuits and Systems* vol. CAS-24 **7**, 400-405.

[30] WRIGHT, J. B., THATCHER, J. W., WAGNER, E. G. AND GOGUEN, J. A. (1976), Rational algebraic theories and fixed-point solutions, in "Proceedings, 17th IEEE Symposium on Foundations of Computer Science, Houston, Texas." 147-158.

Index

CENTRE FOR NFLD. STUDIES
JUL 18 1978
MEMORIAL UNIVERSITY
OF NEWFOUNDLAND