On some equivalence notions of synchronous systems

Miklós Bartha [*]
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada

Branislav Cirovic
Department of Electronics Engineering
College of the North Atlantic
St. John's, NL, Canada

Abstract

An important optimization tool in the design of synchronous systems is retiming, which in many cases allows a significant reduction in the length of the systems' clock period. Even though the internal structure of systems changes upon retiming, their input-output behavior remains essentially the same. The original system and the one after the retiming can simulate each other in a suitable way. The equivalence notion arising from this kind of mutual simulation is called simulation equivalence, and the aim of this paper is to characterize simulation equivalence in an algebraic setting. It is shown that simulation equivalence is a congruence relation of the algebra of synchronous schemes, and that this congruence is the smallest one containing retiming equivalence and finitary strong equivalence. An axiomatization of these equivalences is presented in the general framework of strictly monoidal categories with feedback.

1 Introduction

The increasing demands of speed and performance in modern signal and image processing applications necessitate a revolutionary super-computing technology. In most real-time digital signal processing applications, general purpose parallel computers cannot offer satisfactory processing speed due to severe system

[*]Partially supported by Natural Science and Engineering Research Council of Canada, Discovery Grant #170493-03

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 systems was derived from that of a systolic system, which has turned out to be one of the most attractive tools in massive parallel computing. A large number of systolic systems have been designed, many of them manufactured. Transformation methodologies for the design and optimization of systolic systems have been developed, and yet, a rigorous mathematical foundation of systolic and synchronous systems has not been provided until recently. Important equivalence relations of synchronous systems, such as retiming equivalence, strong equivalence, and simulation equivalence still lack a proper characterization and decision algorithms. The present paper aims at providing a suitable characterization in a precise algebraic framework, which characterization will easily lead to appropriate decision algorithms.

As introduced in [15], a synchronous system is partitioned into functional elements (combinational logic) and registers (clocked memory). Such a system can be described by an edge-weighted directed graph G, in which the vertices represent functional elements and the edges correspond to interconnections between the functional elements. The weight of each edge in G is a non-negative integer, which indicates the number of registers placed along the interconnection between the two functional elements that correspond to the endpoints of the edge. The external interface is represented in G by a distinguished vertex, called the host.

In a synchronous system, every functional element has a fixed primitive operation associated with it. These operations are designed to manipulate some simple data (e.g. signals) in the usual algebraic sense. The registers and functional elements are organized by a common clock, which renders the following stepwise behavior to the system. A state (also called configuration) is an assignment of data to all registers. With each clock tick, the current configuration is mapped into a new configuration in such a way that every functional element performs the primitive operation associated with it. The operands (result) of the operation performed by each functional element are taken from (is forwarded to) the nearest registers lying on the interconnections arriving at (going out of) the functional element. At the same time, data are advanced one register in the queue of registers along each interconnection. If there is no register along an interconnection, then data are always propagated through that interconnection during a single clock cycle. To avoid circular rippling of data within the system, it is assumed that every oriented cycle in the graph of the system contains at least one edge having strictly positive weight.

In the forthcoming sections we are going to present a formal algebraic model for the study of synchronous systems, and characterize some of their basic equivalences as congruence relations in the algebra of synchronous schemes. The reader is referred to [12] for the universal algebraic terminology used.

2 Schemes, flowcharts, and their algebras

As introduced in [3], a *synchronous scheme* over a ranked alphabet $\Sigma = \{\Sigma_n | n \geq 0\}$ is a finite directed graph F having the following additional structure.

1. Each vertex v is labeled by either a symbol in Σ, or one of the symbols in:
$$\{\mathrm{ic}_j \mid 1 \leq j \leq q\} \cup \{\mathrm{oc}_i \mid 1 \leq i \leq p\} \cup \{\perp\},$$
where p and q are fixed non-negative integers. If the label of v is in Σ, then v is called a *box*. Boxes represent functional elements in synchronous systems. Vertices labeled by the symbols $\{\mathrm{ic}_j \mid 1 \leq j \leq q\}$ and $\{\mathrm{oc}_i \mid 1 \leq i \leq p\}$ are called *input and output channels*, respectively, and every vertex labeled by $\perp$ is called a loop vertex. We shall assume that each label, not only those in Σ, has a fixed rank associated with it, so that $rank(\mathrm{oc}_i) = 1$, $rank(\mathrm{ic}_j) = 0$, and $rank(\perp) = 1$. Then the in-degree of v (that is, the number of edges arriving at v) equals the rank of the symbol labeling v. Moreover, the only edge arriving at each loop vertex is a loop around that vertex.

2. The edges arriving at any vertex v labeled by a symbol of rank n are ordered, which order is captured by saying that these edges enter v at the first,..., n-th *input port*.

3. Each edge e is assigned a non-negative integer weight $w(e)$, which specifies the number of registers placed along the interconnection represented by e. It is required that, in each oriented cycle of F, there exists at least one edge e with $w(e) > 0$. This requirement will be referred to as the *exclusion of rippling*.

A synchronous *system* is a pair $(F, \mathcal{I})$, where F is a synchronous Σ-scheme (scheme, for short), and $(\Sigma, \mathcal{I})$ is a Σ-algebra. If F is a scheme having p output and q input channels, then we write $F : p \to q$. Let F^R denote the directed graph obtained from F by reversing the direction of each edge in it. When forgetting the weight of the edges, F^R becomes a flowchart in the sense of [10]. This flowchart will be denoted by *fl*(F).

Vertex v in scheme F is said to be *accessible* if there exists a directed path in F^R from some output channel leading to v. Scheme F is accessible if all of its boxes and loop vertices are accessible. System S is accessible if the scheme of S is such.

Now we turn to defining the algebra of synchronous schemes and some other related algebras. Each of these algebras is sorted by the set $N \times N$ of all pairs of non-negative integers, and we shall refer to any element a in an underlying set of sort (p, q) as $a : p \to q$. Our algebras use a common set of constants and operations, which are listed below.

Constants. $1 : 1 \to 1$, $0 : 0 \to 0$, $0_1 : 0 \to 1$, $x : 2 \to 2$, and $\epsilon : 2 \to 1$.
Sum. If $F_1 : p_1 \to q_1$ and $F_2 : p_2 \to q_2$, then $F_1 + F_2 : p_1 + p_2 \to q_1 + q_2$.

Composition. If $F : p \to q$ and $G : q \to r$, then $F \cdot G : p \to r$.
Feedback. If $F : 1 + p \to 1 + q$, then $\uparrow F : p \to q$.

The algebra $\mathrm{Syn}(\Sigma)$ of synchronous schemes is defined as follows.

— The constants are "all-wire" schemes not containing boxes, registers, or loop vertices. Each one of them, except for x, is uniquely determined by this description as an appropriate mapping. The constant x is defined as the transposition mapping $2 \to 2$.

— The sum of schemes $F_1 : p_1 \to q_1$ and $F_2 : p_2 \to q_2$ is essentially their disjoint union with an appropriate relabeling of the input-output channels of F_2.

— The composite of schemes $F : p \to q$ and $G : q \to r$ is obtained by gluing them together at the input (output) channels of F (respectively, G). The weights of the edges that are joined during this procedure are added up, and output wires of G "cut" by an isolated input channel of F are deleted. See [3] for the details.

— The feedback of scheme $F : 1 + p \to 1 + q$ is obtained by joining the edge e arriving at oc_1 with each of the edges starting from ic_1 (if any), adding up their weights as in the case of composition, and incrementing each of these sums by 1. The incrementation amounts to putting an extra register along each of the newly created interconnections. Again the output wire e is deleted if ic_1 is isolated. In case e comes directly from ic_1, a new loop vertex is created and the weight of its loop is set to $w(e) + 1$.

Notice the different treatment of loop vertices in the definition of feedback above compared to [2, 3] and [7]. In those papers there is exactly one loop vertex in each scheme, regardless of whether this vertex has actually been generated by the feedback operation or not. This is a minor change, however, and the axiomatization of schemes presented in those works can be adjusted in a staightforward manner to accomodate multiple loop vertices. This issue will be dealt with in Section 7.

Sometimes it is advantageous to make the presence of registers more explicit in schemes. For this reason, augment Σ by a new symbol ∇ of rank 1, and consider the graph F^R for any scheme $F : p \to q$. Interpret the weight of each edge e in F^R by subdividing e with a sequence of $w(e)$ ∇-boxes to obtain a flowchart $\mathit{fl}_\nabla(F) : p \to q$ over the alphabet $\Sigma_\nabla = \Sigma \cup \{\nabla\}$. Clearly, the connection $\chi : F \mapsto \mathit{fl}_\nabla(F)$ is one-to-one, but not all Σ_∇-flowcharts are covered by χ because of the restriction imposed by the exclusion of rippling. As another slight deficiency, the equation $\epsilon \cdot \nabla = (\nabla + \nabla) \cdot \epsilon$, which would be necessary to exclude branches originating from ∇-boxes, is not true in the algebra of ordinary Σ_∇-flowcharts.

Let $\mathrm{Fl}(\Sigma_\nabla)$ denote the algebra of Σ_∇-flowcharts as introduced in [2], with the modified treatment of loop vertices described above. Take the quotient of $\mathrm{Fl}(\Sigma_\nabla)$ determined by the equation $\epsilon \cdot \nabla = (\nabla + \nabla) \cdot \epsilon$, and restrict the underlying sets of this algebra to flowcharts that are in the range of the function

χ. Let $\mathrm{Fl}_{\nabla}(\Sigma)$ denote the resulting partial algebra. We shall make use of this algebra in the proof of Theorem 4.2. To ensure a uniform treatment for flowchart and synchronous schemes, flowcharts will also be considered as edge-weighted schemes in which all edges have zero weight.

3 Simulation equivalence of synchronous systems

The following definition of simulation equivalence between two accessible synchronous systems is quoted from [15].

Definition 1 System S_1 can simulate system S_2 if, for every sufficiently old configuration c_2 of S_2, there exists a configuration c_1 of S_1 such that S_1 and S_2 exhibit the same input-output behavior when started from configurations c_1 and c_2, respectively. Systems S_1 and S_2 are *simulation equivalent* if they can simulate each other. Schemes F_1 and F_2 are simulation equivalent, if the systems $S_1 = (F_1, \mathcal{I})$ and $S_2 = (F_2, \mathcal{I})$ are such under all interpretations $\mathcal{I}$.

To avoid ambiguity, we need to spell out the meaning of the term "sufficiently old" in the definition of simulation equivalence above. It means that, starting from an *arbitrary* initial configuration c of S_2, there exists a non-negative integer $n(c)$ such that, no matter how the inputs to S_2 are chosen in the first $n(c)$ clock cycles, the resulting configuration c_2 has the property specified in Definition 1. The corresponding configuration c_1 of S_1 will then depend on c and the inputs to S_2 in the first $n(c)$ clock cycles. The important point here is that the choice of c is arbitrary. This is to ensure transitivity of simulation, so that simulation equivalence indeed becomes an equivalence relation of synchronous systems (schemes). Simulation equivalence of either systems or schemes will be denoted by $\sim$.

Examples In all of the examples below, fix the interpretation $\mathcal{I}$ to be the free Σ-algebra T_{Σ}, and assume that Σ has at least two symbols, one of which is a constant. The symbol g used in the examples has rank 1. For better understanding, the schemes appearing in Figures 1, 2, 3, and 4 have their registers represented by ∇-boxes rather than integer weights.

Example 1. Consider the systems S_1 and S_2 in Fig. 1.

Clearly, S_2 can simulate S_1. We do not even have to bother making a configuration c of S_1 sufficiently old, because the configuration $f(c)$ will do for S_2. In order to simulate S_2 with S_1, however, we need to run S_2 for one clock cycle in order to obtain the value $f(x_1)$ in its sole register, where x_1 is the input to S_2 in the first clock cycle. Then, by assigning the value x_1 to the register of S_1, the simulation of S_2 is properly established. Thus, $S_1 \sim S_2$.

Example 2. See Fig. 2 for the systems S_1 and S_2.

Again, S_2 can trivially simulate S_1 by assigning the same value to its two registers. When simulating S_2 with S_1, however, we cannot assume that these values are the same, at least not initially. After just one clock cycle, however, the difference between the initial contents of the two registers "flushes out", so

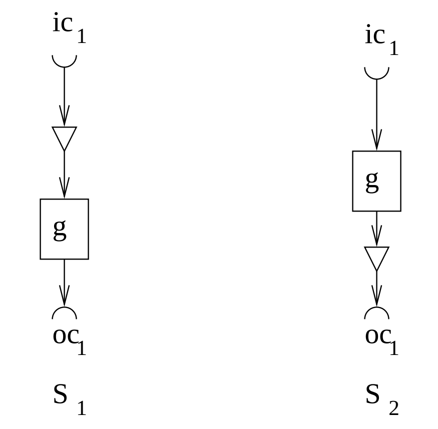

Figure 1: The schemes of Example 1

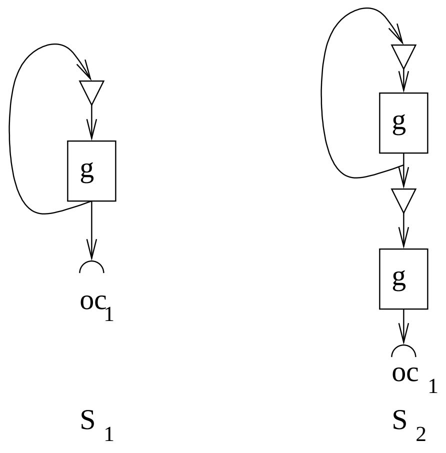

Figure 2: The schemes of Example 2

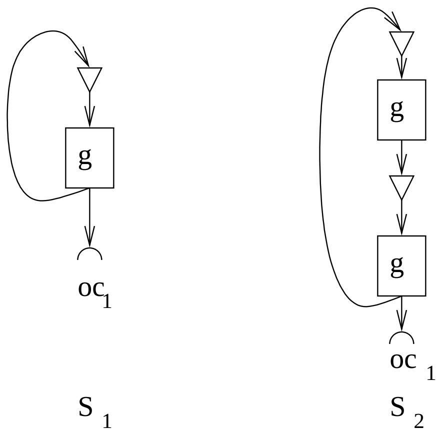

Figure 3: The schemes of Example 3

that an appropriate configuration of S_1 can be adjusted. Thus, $S_1 \sim S_2$.

Example 3. See Fig. 3 for the systems S_1 and S_2.

Now S_1 cannot simulate S_2, because any possible difference between the initial contents of the two registers in S_2 "regenerates" itself after each clock tick. The contents of these two registers are therefore inherently different, which makes it impossible to set the simulating configuration of S_1 in an appropriate way. Thus, $S_1 \not\sim S_2$.

Example 4. See Fig. 4 for the systems S_1 and S_2.

Observe that the two outputs of S_1 are the same in each clock cycle, regardless of the initial configuration. The two outputs of S_2, however, will always be out of pace compared to one another, the output at oc_2 "lagging one g" behind the output at oc_1. Thus, $S_1 \not\sim S_2$.

By definition, simulation equivalence is of a semantic nature. As stated in [15]:

"Two synchronous systems may be equivalent even though their internal organizations are radically different. For example, one system might be a tree and the other a mesh."

Our goal is to demonstrate that this statement over-estimates the complexity of simulation equivalence, which can in fact be characterized in syntactical terms. This characterization even admits a decision algorithm for simulation equivalence, which will be presented in a forthcoming paper.

We start out by a theorem that reveals an important algebraic property of simulation equivalence. The proof of the theorem is a routine check to verify that simulation equivalence is compatible with the scheme operations sum, composition, and feedback.

Theorem 3.1 *Simulation equivalence is a strong congruence relation of the par-*

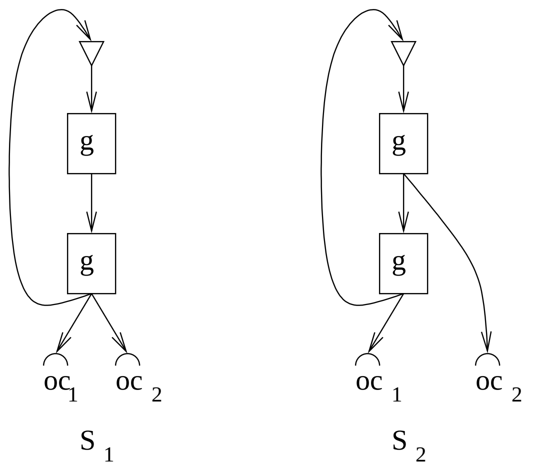

Figure 4: The schemes of Example 4

tial subalgebra $Syn_a(\Sigma)$ *of* $Syn(\Sigma)$ *determined by all accessible schemes.*

Theorem 3.1 would remain true in the whole algebra Syn(Σ), should we want to extend Definition 1 directly to all schemes. This is not our intention, however, for reasons outlined in Section 7.

4 Retiming synchronous systems

Let F be scheme and u be a box in F labeled by $\sigma \in \Sigma_n$ such that the edges $e_1, \ldots, e_n$ arriving at the input ports of u have positive weights. *Retiming* u then means subtracting 1 from $w(e_i)$ for all $1 \leq i \leq n$, and adding 1 to the weight of each edge going out from u. Retiming will also be allowed on loop vertices, in which case the label of u is $\perp$, and $n = 1$. *Elementary retiming* is the binary relation $\rightarrow_r$ on Syn(Σ) by which $F \rightarrow_r F'$ if F' results from F by retiming a single box or loop vertex in it. The smallest equivalence relation containing $\rightarrow_r$ is called *retiming equivalence*, and is denoted by $\sim_r$.

A *retiming count vector* for scheme F is an assignment R of integers to all of its boxes and loop vertices. Extend R to all vertices of F by fixing $R(v) = 0$ for each input/output channel. We say that R is *legal* if for every edge $e : u \rightarrow v$ in F, $w(e) + R(u) - R(v) \geq 0$. If R is legal, then it takes F into a scheme F' that has the same flowchart structure as F, but the weight $w'(e)$ of each edge $e : u \rightarrow v$ is $w(e) + R(u) - R(v)$.

A characterization of $\sim_r$ by retiming count vectors was given in [15] as follows.

Proposition 4.1 *Schemes F and F' are retiming equivalent iff there exists a legal retiming count vector R taking F into F'.*

It is clear by the definitions that if R is legal for F, then $-R$ is legal for F', and $-R$ takes F' back to F. It is also easy to see by Proposition 4.1 that retiming equivalence is a congruence relation of $\mathrm{Syn}(\Sigma)$.

Based on Proposition 4.1, the following theorem was proved in [15] as the "Retiming Lemma", relying on the simple graph model of synchronous systems. Here we present an algebraic proof for the more sophisticated synchronous scheme model.

Theorem 4.2 *In $Syn_a(\Sigma)$, $\sim_r \subseteq \sim$.*

Proof. It is sufficient to prove that elementary retiming in synchronous schemes is simulation equivalent. That is, retiming a single box in an accessible scheme F results in an accessible scheme $F' \sim F$. On the analogy of Example 2, this statement is obvious for retiming a loop vertex in F. If F consists of a box $\sigma \in \Sigma_n$ as a stand-alone scheme $1 \to n$ having a single layer of registers on its input side, then the statement is adequately reflected by Example 1. If the box σ is being retimed in a context C, then try to model C as an algebraic expression in $\mathrm{Syn}(\Sigma)$ over one variable $1 \to n$, so that $F = C(S_1)$ and $F' = C(S_2)$, where S_1 and S_2 are single-box schemes before and after a suitable elementary retiming, as in Example 1. If this was possible, then the statement of the theorem would immediately follow from Theorem 3.1. Unfortunately, despite the fact that $\mathrm{Syn}(\Sigma)$ is generated by Σ, the context C cannot always be modeled in this way. The reason is that some of the registers present in S_1 and S_2 may only be generated algebraically by applying the feedback operation as part of the context C, which is illegal. In other words, $\sim_r$ cannot be specified as the smallest congruence relation of $\mathrm{Syn}(\Sigma)$ containing single-box retiming.

On the other hand, it is straightforward to design C in the algebra $\mathrm{Fl}_\nabla(\Sigma)$. Even though $\mathrm{Fl}_\nabla(\Sigma)$ is only a partial algebra, on the analogy of Theorem 3.1 it is easy to establish $\sim$ as a strong congruence in it. With this observation, the proof of Theorem 4.2 is now complete.

5 Strong equivalence of synchronous schemes

Let F be a synchronous scheme. The relation of *having the same strong behavior* is defined on the vertices of F as the largest label-preserving equivalence μ_F such that if $u\mu_F v$, with the label of u and v having rank $n \geq 1$, then for every $1 \leq i \leq n$, $u_i \mu_F v_i$ holds for the vertices u_i and v_i that are connected to the i-th input port of u and v by edges e_i^u and e_i^v; moreover, $w(e_i^u) = w(e_i^v)$. The equivalence μ_F gives rise to a minimal scheme F/μ_F in the usual way (cf. [10]), and schemes F_1, F_2 are said to be *strong equivalent* if $F_1/\mu_{F_1} = F_2/\mu_{F_2}$. Strong equivalence will be denoted by $\sim_s$.

By the standard definition in graph theory, a *directed walk* in graph G is an alternating sequence of vertices and edges, which starts and ends with a vertex, and in which each edge points from the vertex immediately preceding it to the vertex immediately following it. Let F be a scheme, and $\alpha = v_0 e_1 \ldots e_n v_n$ be a directed walk in F^R. By the *pattern* of α we mean the sequence $p(\alpha) = \sigma_0(i_1, w_1)\ldots(i_n, w_n)\sigma_n$, where σ_j, $0 \leq j \leq n$, is the label of vertex v_j, i_j identifies the input port of v_{j-1} to which the edge e_j is connected, and $w_j = w(e_j)$. In general, a pattern of walks is a sequence $p = \sigma_0(i_1, w_1)\ldots(i_n, w_n)\sigma_n$ such that $\sigma_j \in \Sigma \cup \{\mathrm{ic}_k | k \geq 1\} \cup \{\mathrm{oc}_l | l \geq 1\}$, $1 \leq i_j \leq rank(\sigma_{j-1})$ and $w_j \geq 0$. We say that pattern p is *feasible* for vertex u if there exists a directed walk α in F^R starting from u such that $p = p(\alpha)$. In this case, $\mathrm{end}(u, p)$ denotes the last vertex of α.

It is easy to see that, for every two vertices u and v of F, $u\mu_F v$ is equivalent to saying that an arbitrary pattern p is feasible for u iff p is feasible for v.

Now we introduce the relation of *having the same finitary strong behavior* on the set of vertices of scheme F, denoted θ_F. For two vertices u and v, $u\theta_F v$ if for every pattern p, p is feasible for u iff p is feasible for v, and, $\mathrm{end}(u, p) = \mathrm{end}(v, p)$ whenever p is feasible and sufficiently long. Clearly, θ_F is also an equivalence relation, and $\theta_F \subseteq \mu_F$. This equivalence, too, gives rise to a minimal scheme F/θ_F, and schemes F_1, F_2 are said to be *finitary strong equivalent* if $F_1/\theta_{F_1} = F_2/\theta_{F_2}$. Finitary strong equivalence will be denoted by $\sim_f$.

Consider the scheme equation

$$\epsilon \cdot \sigma = (\sigma + \sigma) \cdot \epsilon_n,$$

where $\sigma \in \Sigma_n$ stands for a box scheme $1 \to n$ with no registers, and ϵ_n is the mapping $2n \to n$ by which $\epsilon_n(in + j) = j$ for $i = 0, 1$ and $1 \leq j \leq n$. The mapping ϵ_n is essentially the branch $\epsilon : 2 \to 1$ performed on a "block" of size n, which is easy to assemble from the constants 1, ϵ, and x using sum and composition. (See [3] for the details.) The system of these equations for all $\sigma \in \Sigma$ determines the relation of *box unfolding (reduction)* on $\mathrm{Syn}(\Sigma)$, whereby the left-hand side schemes unfold into the right-hand side ones and vice versa, when reduction takes place.

The following two theorems characterize finitary strong equivalence, and relate it to simulation equivalence.

Theorem 5.1 *Finitary strong equivalence is the smallest congruence relation of Syn(Σ) containing box unfolding/reduction.*

Strong equivalence and finitary strong equivalence have nothing to do with the exclusion of rippling in synchronous schemes, therefore they are meaningful for all Σ-flowcharts. Obviously, Theorem 5.1 also remains true in the algebra $\mathrm{Fl}(\Sigma)$ of all Σ-flowcharts.

Theorem 5.2 *In Syn$_a(\Sigma)$, $\sim_f \subseteq \sim$.*

The reader should make note of the fact that finitary strong equivalence does not coincide with the congruence relation of $\mathrm{Syn}(\Sigma)$ defined by the identity

$$\epsilon \cdot h = (h + h) \cdot \epsilon_n \ , \ \text{ where } h : 1 \to n.$$

Unfolding/reduction is only allowed for individual boxes according to $\sim_f$, and not for all schemes $h : 1 \to n$ as suggested by the identity above. Consequently,

$\sim_f$ fails to be fully invariant. In contrast, $\sim_s$ is fully invariant, and it can be captured by the system of feedback theory identities described in [4]. Nevertheless, $\sim_f$ is still powerful enough to unfold cycle-free accessible schemes into finite trees, hence the name "finitary strong equivalence".

6 The main result

As we have seen in Theorems 4.2 and 5.2, retiming equivalence and finitary strong equivalence are both contained in simulation equivalence. It is natural to ask if there are any other ingredients present in $\sim$, or it is completely determined by the join of these two congruences. This question is answered by Theorem 6.1 below.

Theorem 6.1 *Simulation equivalence is the smallest strong congruence relation of $Syn_a(\Sigma)$ containing retiming equivalence and finitary strong equivalence.*

In [6], the join of $\sim_r$ and $\sim_s$ has been introduced by the name *strong retiming equivalence*. On this analogy, the join of $\sim_r$ and $\sim_f$ is called *finitary strong retiming equivalence*.

Theorem 6.1 is a direct consequence of Theorems 4.2, 5.2, and Statements 1, 2, and 3 below.

Statement 1. If $F_1 \sim F_2$, then F_1 and F_2 are strong retiming equivalent.

Statement 2. If $F_1 \sim F_2$, then $fl(F_1) \sim_f fl(F_2)$. (Recall that $fl(F)$ denotes the Σ-flowchart determined by F.)

Statement 3. If F_1 and F_2 are strong retiming equivalent such that $fl(F_1) \sim_f fl(F_2)$, then F_1 and F_2 are finitary strong retiming equivalent.

Statement 1 was proved in [9]. The proofs of Statements 2 and 3 both rely on the following lemma, the first part of which was already shown in [6].

Lemma 6.2 *If schemes F_1 and F_2 are strong retiming equivalent, then there exist schemes G_1 and G_2 such that $F_i \sim_s G_i$, $i = 1, 2$, and $G_1 \sim_r G_2$. Moreover, if $fl(F_1) \sim_f fl(F_2)$, then G_1 and G_2 can be chosen in such a way that $F_i \sim_f G_i$.*

In the light of Lemma 6.2, Statement 3 is straightforward. The proof of Statement 2 is based on Lemma 6.2, Theorem 5.1, and some other results found in [9, 6]. This proof is challenging.

7 Axiomatizing simulation equivalence

In this section we present an equational axiomatization of the equivalence relations considered, and relate our work to the results obtained in [14] on strictly monoidal categories with feedback. In order to make the connection in a smooth way, we shall simplify the discussion by assuming that the underlying monoidal categories are just magmoids [1] satisfying the block permutation axiom [2, 11]. Such magmoids will be called symmetric.

A *magmoid with feedback* is a symmetric magmoid M equipped with a feedback operation $\uparrow\colon M(1+p,1+q) \to M(p,q)$ satisfying the following axioms.

S1: $\uparrow(f+g) = \uparrow f + g$ for $f: 1+p \to 1+q$, $g: r \to s$;

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

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

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

These axioms were introduced originally in [2] to axiomatize flowchart schemes, but they coincide with a reduced version of the axioms naturality (S3, S4), weak naturality (S2), and superposing (S1) of [14] in the present simplified context. The following two axioms have been added in [13].

S: $\uparrow^m((g+p)\cdot f) = \uparrow^l(f\cdot(g+q))$ for $f: l+p \to m+q$, $g: m \to l$;

X: $\uparrow x = 1$.

Axioms S and X are called sliding and yanking, respectively. It was proved in [2] that the set of axioms {S1, S2, S3, S4, S5, X}, where

S5: $\uparrow 1 = 0$ and $\epsilon\cdot\bot = \bot+\bot$,

provide an axiomatization of flowchart schemes, with the assumption that the loop vertex is unique. Dropping this assumption simply eliminates S5 from the required axioms. Since S is trivially satisfied in the algebra of flowchart schemes (with the loop vertex being unique or not), sliding comes as a consequence of {S1, S2, S3, S4, X}. See also [13, Lemma 2.1] in the general framework of monoidal categories. Sliding has also been considered in [8] by the name circular feedback as one of the basic axioms for feedback theories (with no delay).

Yanking was replaced in [3] by the axiom

S6: $\uparrow^2((\epsilon+p)\cdot f) = \uparrow(f\cdot(\epsilon+q))$ for $f: 1+p \to 2+q$,

and it was proved that {S1,. . .,S6}, together with the axiom

S7: $0_1\cdot\nabla = 0_1$

and another axiom S8: $\uparrow(\epsilon\cdot\nabla^n) = \bot$, which is irrelevant in our present discussion, provide an axiomatization of synchronous schemes.

Now we turn to describing the *Circ* construction in [14]. If M is a symmetric magmoid, then $Circ(M)$ is the magmoid in which the morphisms $p \to q$ are isomorphism classes of pairs (f,l), where $f: l+p \to l+q$ in M, and where an isomorphism from (f,l) to (g,m) is an isomorphism $\gamma: m \to l$ such that $(\gamma+p)\cdot f = g\cdot(\gamma+q)$. Without loss of generality we can assume that $m=l$ and γ is a permutation $l \to l$. Morphisms in $Circ(M)$ are called circuits. The identity circuit $p \to p$ is p in M, and composition of circuits is essentially cascade product of automata. Feedback of circuits is defined by observing the obvious rule $\uparrow(\uparrow^l f) = \uparrow^{l+1} f$ (vanishing). See [5, 14] for the details. It was proved in [14] that for every symmetric magmoid M, $Circ(M)$ is a magmoid with feedback. Moreover, $Circ(M)$ is freely generated by M.

Let us extend the definition of $\sim_f$ by incorporating the equations $0_1\cdot\sigma = 0_n$

for all $\sigma \in \Sigma_n$. This allows for eliminating certain inaccessible boxes in Syn(Σ). Based on Theorem 6.1, redefine $\sim$ as finitary strong retiming equivalence, that is, the join of $\sim_f$ and $\sim_r$. Consider the symmetric magmoid $Df(\Sigma)$ of data flowchart schemes as introduced in [3], and let $T(\Sigma)$ be the free algebraic theory generated by Σ, also considered as a symmetric magmoid with the constants 0_1 and ϵ having their natural interpretation.

For an arbitrary collection E of identities in our algebraic language, let η_E denote the congruence relation of magmoids with feedback determined by E. Combining the axiomatization results of [2, 3] with the *Circ* construction, we obtain the following results.

Theorem 7.1

1. $Circ(T(\Sigma))/\eta_{\mathrm{S6,S7}} \cong Syn(\Sigma)/{\sim_f}$;
2. $Circ(T(\Sigma))/\eta_S \cong Syn(\Sigma)/{\sim}$;
3. $Circ(T(\Sigma))/\eta_X \cong Fl(\Sigma)/{\sim_f}$;
4. $Circ(Df(\Sigma))/\eta_{\mathrm{S6,S7}} \cong Syn(\Sigma)$;
5. $Circ(Df(\Sigma))/\eta_S \cong Syn(\Sigma)/{\sim_r}$;
6. $Circ(Df(\Sigma))/\eta_X \cong Fl(\Sigma)$.

Notice that the equation $\uparrow (0_1 + 1) = 0_1$ cannot be proved from the axioms S1,. . .,S4, therefore the axiom S7 must be considered in 1 and 4 above. Equation S7, however, is provable from S, so that S6 and S7 need not be considered when S is present. (Essentially, S6 and S7 are the special instances of S in which the morphism g is chosen as the constant ϵ and 0_1, respectively.) Moreover, as we have seen earlier, S comes free whenever X is present.

The axioms S5 and S8 are not valid in Syn(Σ) by our present model. One can easily prove, however, that {S1,. . .,S4, S6, S7} is a proper axiomatization of our current Syn(Σ), which is in accordance with 4 above.

8 Conclusion

We have provided an algebraic characterization of simulation equivalence of synchronous systems. First we showed that simulation equivalence is a congruence relation of the algebra of synchronous schemes. Then we identified the two major components of this relation: retiming equivalence and finitary strong equivalence. We have found that both of these equivalences are easy to characterize in syntactical terms as appropriate congruence relations of the algebra of schemes. Finally, we have proved that simulation equivalence is the smallest congruence relation containing these two components. We have also provided an axiomatization of our equivalences relying on the *Circ* construction in [14].

Acknowledgements. The authors thank a referee for pointing out the work of Katis, Sabadini, and Walters. The first author is also grateful to Bob Walters for a helpful communication on this matter.

References

[1] A. Arnold and M. Dauchet, Théorie des magmoïdes, *RAIRO Inform. Théor.* **12** (1978), 235–257 and **13** (1979), 135–154.

[2] M. Bartha, A finite axiomatization of flowchart schemes, *Acta Cybernet.* **2** (1987), 203–217.

[3] M. Bartha, An equational axiomatization of systolic systems, *Theoret. Comput. Sci.* **55** (1987), 265–289.

[4] M. Bartha, Foundations of a theory of synchronous systems, *Theoret. Comput. Sci.* **100** (1992), 325–346.

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

[6] M. Bartha, Strong retiming equivalence of synchronous systems, submitted for publication.

[7] S. L. Bloom and Z. Ésik, Axiomatizing schemes and their behaviors, *J. Comput. System Sci.* **31** (1985), 375–393.

[8] V. E. Casanescu and Gh. Stefanescu, Feedback, iteration and repetition, Research Report 42, National Institute for Scientific and Technical Creation, Bucharest, 1988.

[9] B. Cirovic, Equivalence relations of synchronous systems, PhD Dissertation, Memorial University of Newfoundland, 2000.

[10] C. C. Elgot, 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.), pp. 175–230, North Holland, Amsterdam, 1975.

[11] C. C. Elgot and J. C. Shepherdson, An equational axiomatization of the algebra of reducible flowchart schemes, IBM Research Report RC 8221.

[12] G. Grätzer, *Universal Algebra*, Springer-Verlag, Berlin, 1968, 1979.

[13] A. Joyal, R. Street, and D. Verity, Traced monoidal categories, *Math. Proc. Camb. Phil. Soc.* **119** (1996), 447–468.

[14] P. Katis, N. Sabadini, and R. F.C. Walters, Feedback, trace, and fixed-point semantics, *Theoret. Informatics Appl.* **36** (2002) 181–194.

[15] C. E. Leiserson, J. B. Saxe, Optimizing synchronous systems, *J. VLSI Comput. Systems* **1** (1983), 41–67.