Algebraic Semantics of Register Transfer Level in Synthesis of Stream Calculus-Based Computing Big Data in Livestream

Pham Van Dang[1,2,3(✉)], Phan Cong Vinh[1,2,3], and Nguyen Bao Khang[4]

[1] Graduate University of Science and Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam
{pvdang,pcvinh}@ntt.edu.vn
[2] Institute of Applied Mechanics and Informatics, Ho Chi Minh City, Vietnam
[3] Faculty of Information Technology, Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam
[4] British International School Ho Chi Minh City, Ho Chi Minh City, Vietnam
knguyen22@bisvietnam.com

Abstract. This paper represents verification algorithms and register transfer level (RTL) specification as algebraic aspects proposed to validate the results of RTL synthesis. Major properties of this approach, the conception of an algebraic semantics-based model (ASM), to be interpreted as a Chu space, is viewed as an algebraic semantics foundation for the RTL formalization and the conception of algebraic semantics-based specification automata (ASA_{SPEC}) are given for formal correctness of the results of RTL synthesis. Approaching formal verification is focused on functional equivalence examining to define if the algebraic RTL automata (ASA_{RTL}) are equivalent to ASA_{SPEC}. To put it another way, the comparison is determined as an assessing that examines the synthesis algorithm is produced an effective RTL specification.

Keywords: Algebraic semantics · Algebraic semantics-based model (ASM) · Algebraic semantics-based specification automata (ASA_{SPEC}) · Algebraic RTL automata (ASA_{RTL}) · Big data in livestream (BDL) · Register transfer level (RTL) · Stream calculus-based computing BDL · Control step (CStep) · Functional unit (FU)

1 Introduction

In recent years, several researchers of many areas have been analyzing techniques known as partial order methods, which can obviously reduce the running time of formal validation by avoiding redundant exploration of executive scenarios [1, 4]. This paper performs development an algebraic semantics-based model (ASM) for the RTL formalization and via the algebraic semantics, proposes a formal verification approach to proving the result correctness of RTL synthesis in stream calculus-based computing BDL. To put it another way, approach of this paper is to model the behaviors using algebraic semantics as a

P. Cong Vinh and H. Mahfooz Ul Haque (Eds.): ICTCC 2023, LNICST 586, pp. 19–35, 2024.
https://doi.org/10.1007/978-3-031-59462-5_2

foundation and utilize this model to construct attracting properties of the model that should remain both at the behaviors and RTLs. After that RTL synthesis, this paper will verify the same properties continue to remain for designing the RTL. In addition, two achievable targets from this approach are accurately examining an RTL specification regarding a behavioral specification, bisimulation of two RTL synthesis results of the same algorithms.

In stream calculus-based computing big data in livestream (BDL), correctness validation is usually a hard and open problem in generally giving all researchers. It can be done via two fundamental methods to be formal verification and simulation [1–6]. In the formal verification approach, demonstrating of theorem and examining of model are considered the most fashionable techniques to show whether the design of RTL synthesized from the behavioral specification of algorithm is mathematically correct. Finding formal verification approaches to supply exact and rapid validation clearly is a so valuable goal. In the research [7], a categorical method (to be interpreted as algebraic semantics) is utilized to build a vigorous formal basic for the contextual perception model (to be interpreted as algebraic semantics-based model) so as to acquire the algebraic semantics.

This paper is structured as follows. Section 1 presents introduction and objectives of this paper. Section 2 overviews related work. Section 3 describes stream algebra and uses some fundamental operators for BDL. Section 4 specifies algebraic semantics for RTL. Section 5 describes verification algorithms for the results of RTL synthesis. Section 6 outlines conclusions and future work. Finally, there are references.

2 Related Work

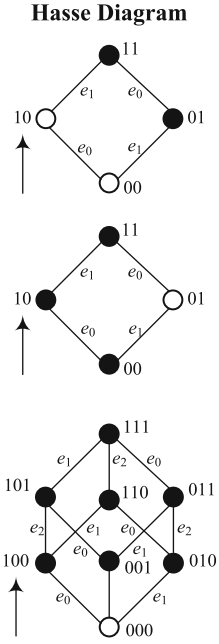
Normally, a Chu space is a matrix over a set {0, 1}, to be interpreted as, an array whose entries are drawn from {0, 1}. Formally, in [8–15], a Chu space is viewed as a binary relation between two sets of events E and states S. It is written as a triple (E, S, R), in which $R : E \times S \rightarrow \{0, 1\}$ is the binary relation as a properties function of a $E \times S$ subset.

Here, E and S are examined as two generic mathematical sets. Figure 1 gives some Chu spaces examples. The E elements are symbolized by $e_0, e_1, e_2, e_3, ...e_n, 0 \leq i \leq n$ and the elements of S are symbolized by $s_0, s_1, s_2, s_3, ...s_m, 0 \leq j \leq m$.

In [2, 3, 12], they have symbolized E as the events set and S as the states set. A state is determined about an occurrence relation $R(e, s)$ which is true where the event e has happened in the state$s$. Therefore, each state s is an E subset holding the events that have happened in the state$s$. Algebraic semantics is viewed as a Chu space C specified by the triple(E, S, R), in which $E = \{e_0, e_1, e_2, e_3, ..., e_n\}, 1 \leq i \leq n$ is an events set, $S = \{s_0, s_1, s_2, s_3, ..., s_n\}, 1 \leq i \leq n$ is a states set, and $R : E \times S \rightarrow \{0, 1\}$ represents the occurrence relation, to be interpreted as, $R(e, s) = 1$ if the event e has happened in the state s and$R(e, s) = 0$, otherwise. For any$e \in E$, each state $s_i \in S = 2^E$ is determined in terms of R by$s_i = \{e|R(e, s_i) = 1\}$.

The algebraic semantics in Fig. 2 has three events, $\{e_0, e_1, e_2\}$ and seven states $\{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, and represents a system, where any event $\in \{e_0, e_1, e_2\}$ occurs and according to data produced by event that one of the remaining two events will happen. For example, in state s_0, the event e_2 has happened; s_2 describes the state in which e_1 has happened after e_2 or e_2 has happened after e_1.

An algebraic semantics can be represented in a matrix, as a Hasse diagram or as a logical formula [16, 17]. In the matrix description, each entry pair (e, s) holds the value of the occurrence relation. Therefore, the matrix rows relate to the algebraic semantics states and the matrix columns relate to the algebraic semantics events. In a Hasse diagram, an algebraic semantics is represented by the partial order existing between the states. In the logical formula, the matrix is viewed as a truth table, the logical description $f_{\mathcal{D}}$ of the algebraic semantics $\mathcal{D}$ is determined by $f_{\mathcal{D}} = \bigvee_{0<i<n} (\wedge\{e|R(e, s_i) = 1\}) \wedge \{\wedge\overline{e}|R(e, s_i) = 0\}$, where $n = |S|$ and $e \in E$ and $\overline{e}$ is the complement of e.



R	e_0	e_1
s_0	1	1
s_1	0	1

(a)

R	e_0	e_1
s_0	1	1
s_1	0	0
s_2	1	0

(b)

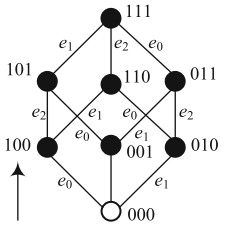
R	e_0	e_1	e_2
s_0	0	1	0
s_1	0	0	1
s_2	1	0	0
s_3	0	1	1
s_4	1	1	0
s_5	1	0	1
s_6	1	1	1

(c)

Fig. 1. Chu spaces and its descriptions by matrix (a), (b) and (c).

In which, the events $e \in E$ are seen as variables, the states $s \in S$ are seen as in terms of logical formula and the R relation defines either the variable is complemented $R(e, s_i) = 0$ or not $R(e, s_i) = 1$. Logical formula f is true for each state $s \in S$ which is allowed in the algebraic semantics. From that, we are obtained an ASM in form as follows ASM $= (E, S, R)$, where $E = \{e_0, e_1, e_2\}$ holds the set of events, $S = \{\{e_1\}, \{e_2\}, \{e_0\}, \{e_1, e_2\}, \{e_0, e_1\}, \{e_0, e_2\}, \{e_0, e_1, e_2\}\}$ holds the states set, $R(E, S)$ is the relation between event E and state S represented by the binary matrix (see Fig. 2).

Some the fundamental relations between events as follows relations of independence, precedence, conflict, and disjunctive enabling determined in [16–18] are used in this paper.



Matrix

R	e_0	e_1	e_2		**Logical formula** f_{ASM}
s_0	0	1	0	$\{e_1\}$	$\{\overline{e_0}, e_1, \overline{e_2}\}$
s_1	0	0	1	$\{e_2\}$	$\{\overline{e_0}, \overline{e_1}, e_2\}$
s_2	1	0	0	$\{e_0\}$	$\{e_0, \overline{e_1}, \overline{e_2}\}$
s_3	0	1	1	$\{e_1, e_2\}$	$\{\overline{e_0}, e_1, e_2\}$
s_4	1	1	0	$\{e_0, e_1\}$	$\{e_0, e_1, \overline{e_2}\}$
s_5	1	0	1	$\{e_0, e_2\}$	$\{e_0, \overline{e_1}, e_2\}$
s_6	1	1	1	$\{e_0, e_1, e_2\}$	$\{e_0, e_1, e_2\}$

Fig. 2. Algebraic semantics and its descriptions

Relation (∇)**:** The relation of independence ($e_0 \nabla e_1$) describes the independent execution of two events e_0 and e_1 of every allowed state. To put it another way, all subsets of E are effective states. Figure 3(a) presents the algebraic semantics and its relevant logical formula.

∇	e_0	e_1	$f_\nabla =$
s_0	0	1	$\{\overline{e_0}, e_1\} +$
s_1	0	0	$\{\overline{e_0}, \overline{e_1}\} +$
s_2	1	1	$\{e_0, e_1\} +$
s_3	1	0	$\{e_0, \overline{e_1}\}$
			$= \mathbf{1}$

(a) Relation matrix of independence

$\prec$	e_0	e_1	$f_\prec =$
s_0	1	0	$\{e_0, \overline{e_1}\} +$
s_1	0	0	$\{\overline{e_0}, \overline{e_1}\} +$
s_2	1	1	$\{e_0, e_1\}$
			$= e_0 + \overline{e_1}$

(b) Relation matrix of precedence

den	e_0	e_1	e_2	$f_{\text{den}} =$
s_0	0	1	0	$\{\overline{e_0}, e_1, \overline{e_2}\} +$
s_1	0	0	0	$\{\overline{e_0}, \overline{e_1}, \overline{e_2}\} +$
s_2	0	1	1	$\{\overline{e_0}, e_1, e_2\} +$
s_3	0	0	1	$\{\overline{e_0}, \overline{e_1}, e_2\} +$
s_4	1	0	1	$\{e_0, \overline{e_1}, e_2\} +$
s_5	1	1	1	$\{e_0, e_1, e_2\} +$
s_6	1	1	0	$\{e_0, e_1, \overline{e_2}\} +$
				$= \overline{e_0} + e_1 + e_2$

(d) Relation matrix of disjunctive enabling

#	e_0	e_1	$f_\# =$
s_0	0	1	$\{\overline{e_0}, e_1\} +$
s_1	0	0	$\{\overline{e_0}, \overline{e_1}\} +$
s_2	1	0	$\{e_0, \overline{e_1}\}$
			$= \overline{e_0} + \overline{e_1}$

(c) Relation matrix of conflict

Fig. 3. Some fundamental relations between the events of the states in Chu space

Relation ($\prec$)**:** The relation of precedence ($e_0 \prec e_1$) represents the event occurrence e_0 followed by the event occurrence e_1. To put it another way, this precedence relation is utilized to specify the sequential events execution. Figure 3(b) represents the algebraic semantics and its relevant logical formula.

Relation (#): The relation of conflict ($e_0 \# e_1$) describes either the e_0 event occurrence or the e_1 event occurrence. To put it another way, both the events e_0 and e_1 can never occur

in e_0 same computation of the algebraic semantics. Figure 3(c) represents the algebraic semantics and its relevant logical formula.

Relation (den): The relation of disjunctive enabling $\text{den}(e_0, e_1, e_2)$ represents the occurrence of event e_1 or e_2 enabling the occurrence of e. In addition, this relation can be used along with the relation of conflict to specify permission for events following the lines of an *if ...then...else* statement. Figure 3(d) represents the algebraic semantics and its relevant logical formula.

3 Stream Algebra for BDL

Stream algebra is represented as sets with operators. Handling the frames infinite sequences is called streams as single entities, an algebra of streams is built in two ways as in analysis, including stream differentiation and integration. In algebra, where operators and establishes identities is computed [19, 20].

The stream algebra has been proposed in [21–25] consisting of some operators such as the operators of merging, dropping, taking, zipping, splitting, copying, registering and assignment [19]. A stream σ is defined by the function in the following.

$$\sigma : \mathbb{N} \to \mathbb{A}^{\omega}$$

Here, $\mathbb{N} = \{0, 1, 2, 3, ...\}$ is natural numbers set and $\mathbb{A}$ is set of frames. A frame may be indicated as a framework, which holds of data types as text, image, audio, video, space, time, geometry and so on. To be interpreted as$\sigma = (\sigma(0), \sigma(1), \sigma(2), ...)$.

For formally defining stream, the stream derivative notation is used for a stream definition as follows. For every integer $n \geq 0$, with differential equation: $\left(\sigma^{(n)}\right)' = \sigma^{(n+1)}$, initial value $\sigma(0)$. The initial value of stream σ is defined namely $\sigma(0)$, and the stream derivative signified by $\sigma\prime$ and is defined by $\left(\sigma^{(n)}\right)' = \sigma^{(n+1)}$, for any integers $n \geq 0$. To put it another way, the initial value and stream derivative equivalent the head and tail of stream σ, respectively. The stream behavior σ includes two aspects, it enables for the initial value inspection $\sigma(0)$ and it can make a development to the new stream $\sigma\prime$, including the original stream from which the first element has been dropped. The initial value of $\sigma\prime$, which is $\sigma\prime(0) = \sigma(1)$, in its turn can be inspected, but note that we had to move from σ to $\sigma\prime$ first in order to do so. Now a behavioral differential equation defines a stream by specifying its initial value together with a its derivative description, which tells us how to continue.

Representing the fundamental definitions on streams that they will be used. Moreover, the authors of researches also propound a brief overview of a coinductive streams calculus [22, 25. The streams set are defined over frames set $\mathbb{A}$ by$\mathbb{A}^{\omega} = \{\sigma | \sigma : \mathbb{N} \to \mathbb{A}\}$. The Greek letters $\sigma, \tau, \gamma, \pi, \epsilon, ...$ are used to symbolize streams. Elements $\sigma \in \mathbb{A}^{\omega}$ are signified by$\sigma = (\sigma(0), \sigma(1), \sigma(2), ...)$. The stream derivate of a stream σ is $\sigma' = (\sigma(1), \sigma(2), \sigma(3), ...)$ and the initial value of σ is$\sigma(0)$. For $n \geq 0$ and$\sigma \in \mathbb{A}^{\omega}$, higher-order derivate is determined by $\sigma^{(0)} = \sigma$ and$\sigma^{(n+1)} = \left(\sigma^{(n)}\right)'$. This paper provides$\sigma(n) = \sigma^{(n)}(0)$.

In this paper, we use some fundamental operators for stream calculus-based computing BDL in the research [19, 20, 25] calculated in Al algorithm the following subsections.

4 Algebraic Semantics for RTL

In this paper, an algebraic semantics for an RTL specification and a verification algorithm (see Fig. 4(a)) have been developed for validating the results of RTL synthesis, using the existing concepts covered in this section.

4.1 Events Set

Definition 1 (Computation). A finite actions sequence is a computation over E events set and the computations set is symbolized by $\mathcal{C}(E)$.

4.2 States Set

Let S be the states set. State $s \in S$ is determined by a transition relation $T(e, s_i)$, also sometimes denoted by $s_i \xrightarrow{e} s_j$, where the event $e \in E$ can make a transition from the state $s_i \in S$ to $s_j \in S$. Therefore, each state s_j is a subset of event E, holding the events that can make a transition from the s_i. To be interpreted as:

$$s_j = \left\{ e | e \in E : s_i \xrightarrow{e} s_j \right\}$$

To put it another way, the triple (E, S, T) is an algebraic semantics. For all e_i in E and s_i in$S$, a computation of the ASM(E, S, T) is$\mathcal{T} = \left\{ e_1, e_2, e_3, ..., e_n : s_1 \xrightarrow{e_1} s_2, s_2 \xrightarrow{e_2} s_3, ..., s_n \xrightarrow{e_n} s_{n+1} \right\}$. From this consideration, the conception of algebraic automata is developed in the next subsections.

4.3 Algebraic Automata

Definition 2 (Algebraic automata). Algebraic automata are a triple $\mathcal{A} = (S, s_0, T)$, where S is the finite states set, s_0 is the initial state, T is a transition function from $S \times E$ into $S \bigcup \{\perp\}$.

If$T(s, e) = \perp$, no transition labelled by event e can be fired from state$s$. Note that $\perp$ can be considered as a sink state. A computation $\mathcal{T} = \{e_1, e_2, e_3, ..., e_n\}$ is agreed by the automata if there exists $s_1, s_2, s_3, ..., s_n, \in S$ such that$T(s_0, e_1) = s_1$,$\forall 1 < i \leq n, T(s_{i-1}, e_i) = s_i$. This is symbolized by$s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} s_2 ... s_{n-1} \xrightarrow{e_n} s_n$. If it is not the case, there exists$1 \leq k \leq n$, the states sequence such that$s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} s_2 ... s_{k-1} \xrightarrow{e_k} \perp$.

Such a path via automata is known as the automata run over the computation $\mathcal{T}$. The computations set agreed by $\mathcal{A}$ is symbolized by $\mathcal{C}(\mathcal{A})$. The four algebraic automata of four CSteps include four states and such the four states respectively (see Fig. 4(a), 4(b) and 4(c)).

The FUs are normally built into a library representation the map between its functions and the related operators. The FUs can be implementations of single-purpose operators or multiple-purpose operators with control input signals for choosing the desired for

operators [19, 20]. In Fig. 4(a), the eight single-purpose FUs are required to implement the nine operators of *split*, *take*, *zip*, *drop*, $+$, $\cdot$, $:=$, $<>$ and $\blacksquare$ in the following.

In Fig. 4(a), it represents RTL of Al algorithm with four CSteps. CStep 0 performs four operators as follows operators of taking, merging, splitting and copying. Then, results are stored in two registers 1 and 2. Next, CStep 1 performs three operators as follows operators of zipping, merging and dropping using values of two registers 1 and 2. Then, results are stored in two reused registers 1 and 2. CStep 2 performs a comparison using values of two registers 1 and 2 of CStep 1. Then, results are stored in reused registers 1 or 2. Finally, CStep 3 performs an assignment operator using values of two registers 1 or 2 from CStep 2, then sending result of assignment operator for output. In Fig. 4(b), it represents matrixes with four CSteps and registers respectively, in particular, representing State 0 $\left(s_0^{Se/Re}\right)$ to State 3 $\left(s_3^{Se/Re}\right)$ of CStep 0 with two registers and such CStep 1, 2 and 3 also are represented. In Fig. 4(c), it is a visual graph that describes matrixes in Fig. 4(b) representing algebraic automata for Al algorithm with four CSteps.

This paper presents now an algebraic automata product in Sect. 4.4 that permits a more complex process modularity specification. In which, each sub-process can be modelled by the algebraic automata and completeness process modeling can be acquired by computing the algebraic automata product of all sub-process represented details in the next section.

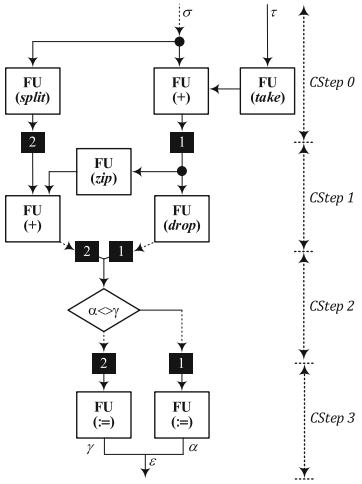
4.4 Algebraic Automata Product

This paper represents the two processes *Se* and *Re* that can be linked as in Fig. 5(a) and 5(b). Let $\mathcal{S}e = \left(S^{Se}, s_p^{Se}, T^{Se}\right)$ and $\mathcal{R}e = \left(S^{Re}, s_q^{Re}, T^{Re}\right)$ be the two algebraic automata that model the processes *Se* and *Re*, respectively. This paper defines algebraic automata product of the two processes of *Se* and *Re* as $\mathcal{S}e \times \mathcal{R}e$ in order to model the process attained by connecting *Se* to *Re*. For asynchronizing *Se* outputs with *Re* inputs so that when a data moves on between *Se* and *Re* through a register then this transfer must be occurred. This shows the determination of the $\mathcal{S}e$ and $\mathcal{R}e$ product, over the similar event set E: $\mathcal{S}e \times \mathcal{R}e = \left(S, s_p, T\right)$, in which $S = S^{Se} \times S^{Re}$, $s_p = \left(s_p^{Se} \times s_q^{Re}\right)$, where p, q are replaced for i, j, k, or$l$.

A transition relation T is determined in the way as follows. Let $s_i = \left(s_i^{Se} \times s_j^{Re}\right)$ be in state S and e in event E. If there exist $s_i^{Se} + 1 \in S^{Se}$ and $s_j^{Re} + 1 \in S^{Re}$ such that $T^{Se}\left(s_i^{Se}, e\right) = s_{i+1}^{Se}$ and $T^{Re}\left(s_j^{Re}, e\right) = s_{j+1}^{Re}$, we set $T\left(\left(s_i^{Se}, s_j^{Re}\right), e\right) = (s_{i+1}^{Se}, s_{j+1}^{Re})$. Otherwise, we set $T\left(\left(s_i^{Se}, s_j^{Re}\right), e\right) = \perp$.

Suppose that the output/input width *Se* is equivalent to the input/output width *Re* respectively, so these processes can be linked each other in Fig. 5, in which Fig. 5(a) is a matrix representing sixteenth states of four registers and then Fig. 5(b) is a visual graph representing RTLs of sixteenth states with four registers. Each state in $\mathcal{S}e \times \mathcal{R}e$ is a pair including a state from $\mathcal{S}e$ and a state from $\mathcal{R}e$. The run of $\mathcal{S}e \times \mathcal{R}e$ over the agreed computation $\mathcal{T} = \{(1, 1, 0, 1)(1, 1, 1, 0)(1, 1, 1, 1)\}$ is symbolized as below:

$$\left(s_3^{Se}, s_0^{Re}\right) \overset{(1,1,0,1)}{\rightarrow} \left(s_3^{Se}, s_1^{Re}\right) \overset{(1,1,1,0)}{\rightarrow} \left(s_3^{Se}, s_2^{Re}\right) \overset{(1,1,1,1)}{\rightarrow} \left(s_3^{Se}, s_3^{Re}\right)$$



(a) Chart of RTL of Al algorithm with four CSteps along with FUs and operators

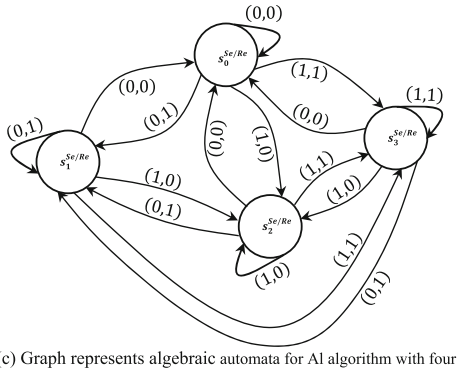
CStep 0	Registers 1	2
State 0 $\left(s_0^{Se/Re}\right)$	0	0
State 1 $\left(s_1^{Se/Re}\right)$	0	1
State 2 $\left(s_2^{Se/Re}\right)$	1	0
State 3 $\left(s_3^{Se/Re}\right)$	1	1

CStep 1	Registers 1	2
State 0 $\left(s_0^{Re/Se}\right)$	0	0
State 1 $\left(s_1^{Re/Se}\right)$	0	1
State 2 $\left(s_2^{Re/Se}\right)$	1	0
State 3 $\left(s_3^{Re/Se}\right)$	1	1

CStep 2	Registers 1	2
State 0 $\left(s_0^{Re/Se}\right)$	0	0
State 1 $\left(s_1^{Re/Se}\right)$	0	1
State 2 $\left(s_2^{Re/Se}\right)$	1	0
State 3 $\left(s_3^{Re/Se}\right)$	1	1

CStep 3	Registers 1	2
State 0 $\left(s_0^{Re/Se}\right)$	0	0
State 1 $\left(s_1^{Re/Se}\right)$	0	1
State 2 $\left(s_2^{Re/Se}\right)$	1	0
State 3 $\left(s_3^{Re/Se}\right)$	1	1

(b) Matrixes represent algebraic automata of Al algorithm with four CSteps

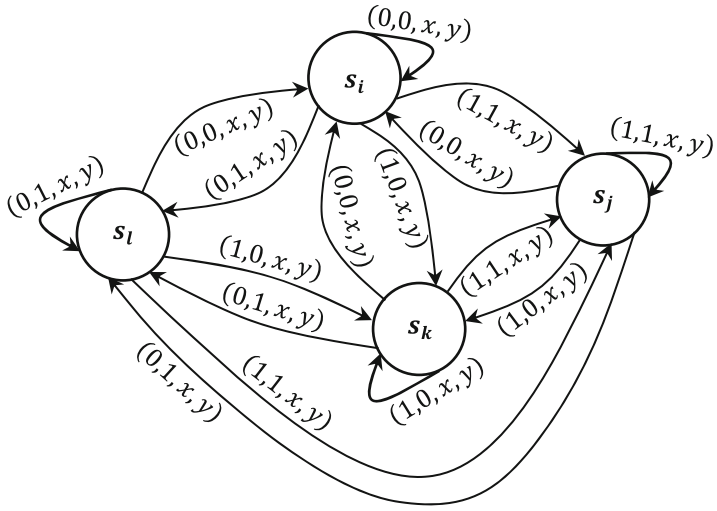


(c) Graph represents algebraic automata for Al algorithm with four CSteps

Fig. 4. The representation of algebraic automata for Al algorithm with four CSteps

$\mathcal{Se} \times \mathcal{Re}$	Registers			
	1	**2**	**3**	**4**
$s_i = \left(s_i^{Se} \times s_j^{Re}\right)$	0	0	x	y
$s_j = \left(s_j^{Se} \times s_k^{Re}\right)$	0	1	x	y
$s_k = (s_k^{Se} \times s_l^{Re})$	1	0	x	y
$s_l = (s_l^{Se} \times s_i^{Re})$	1	1	x	y
In which: $x, y \in \{0,1\}$ and $x \neq y$; $i, j, k, l = \overline{[0,3]} \subset \mathbb{N}^* = \mathbb{N} \cup (0)$.				

(a) This matrix represents sixteenth states of four registers.



(b) Graph represents RTLs of sixteenth states

Fig. 5. Connection of the processes *Se* and *Re* and its algebraic automata product

This signifies that in the state $\left(s_3^{Se}, s_0^{Re}\right)$ on event (1, 1, 0, 1), the algebraic automata product $\mathcal{Se} \times \mathcal{Re}$ carries out by executing $\mathcal{Se}$ from s_3^{Se} and in parallel, executing $\mathcal{Re}$ from s_0^{Re} and in parallel, and so on.

4.5 Algorithms Equivalence

Let Al and $Al^{'}$ be two algorithms and $\mathcal{A}$ and $\mathcal{A}^{'}$ be their related algebraic automata. Al and $Al^{'}$ are equivalent $\iff \mathcal{C}(\mathcal{A}) = \mathcal{C}(\mathcal{A}^{'})$. To put it another way, the algorithms Al and $Al^{'}$ are equal to each other if they cannot be separated by their external behaviors.

4.6 Algebraic Semantics

An algebraic semantics (known as algebraic aspects) can be explained as algebraic automata with events set E and possible states set S. Going into a state s, the algebraic automata execute some transitions over events to reach a substitute state of s. Each

possible computation of the algebraic semantics relates to each run of the algebraic automata and the algebraic semantics execution specifies the algebraic automata runs set.

Algebraic automata in terms of an algebraic semantics model are represented by the events set E and the events happen to each state, to be interpreted as, the transition relation T. A further practical approach reply on relations between states is that each algebraic automata are modelled as a relations set between states and for each such relation, this paper has an algebraic semantics correspondingly. The algebraic semantics of each relation between states is viewed as an algebraic automata characteristic. Therefore, a combination of the characteristics results in the algebraic automata.

5 Verification Algorithms for the Results of RTL Synthesis

5.1 The Algorithm Steps

The verification algorithm steps are represented graphically in following Fig. 6. The algorithm steps are viewed in the following.

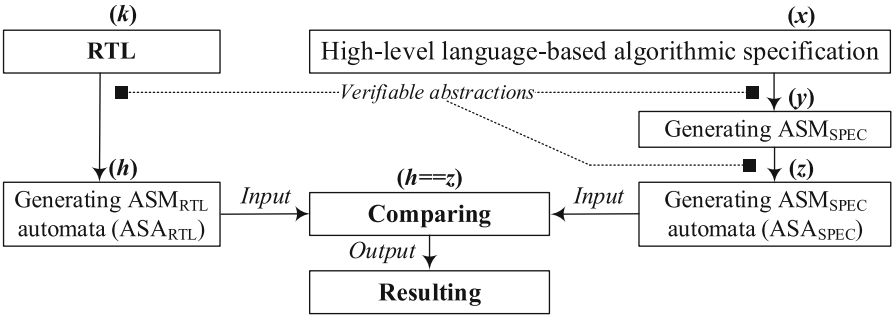


Fig. 6. An algebraic semantics-based verification algorithm for the results of RTL synthesis

5.2 High-Level Language-Based Algorithmic Specification

An appropriate high-level language-based algorithmic specification is described, this means that a program is generated in this step. A main job during this step is the actualization of the modes of various scheduling. Thus, in the very-high hardware description language example indicated in the Fig. 8, this paper has an Al algorithm with seven events associated with the Al algorithm statements, where each statement is viewed as an event.

This paper uses the udStreamsA algorithm in the [19] research to distribute its list of events for the operators of merging, dropping, taking, zipping, splitting and assignment [19] with single CStep in this algorithm to become Al algorithm below (see Fig. 7). The diverse synthesis steps of stream calculus-based computing BDL are illustrated via an example that maps input streams ($\sigma, \tau \in \mathbb{A}^{\omega}$) onto the output streams tuples ($\alpha, \gamma, \epsilon \in \mathbb{A}^{\omega}$) as defined by the algorithm specification, namely Al algorithm (Fig. 8).

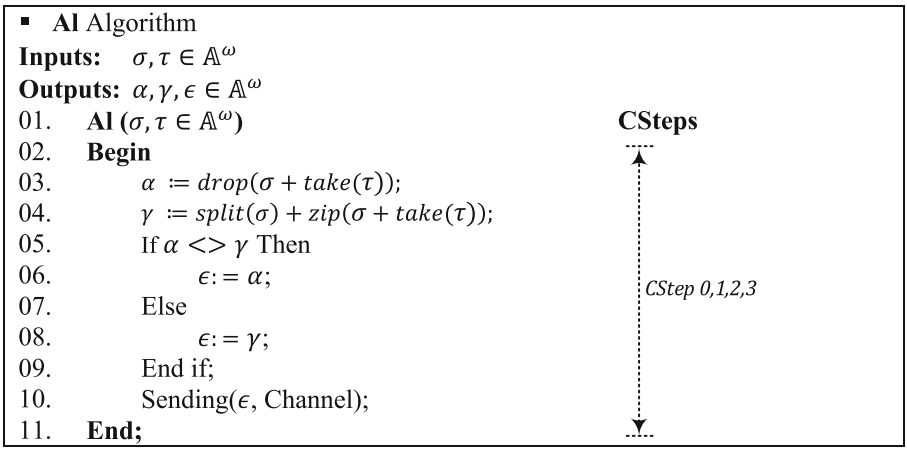


Fig. 7. An Al algorithm for four CSteps

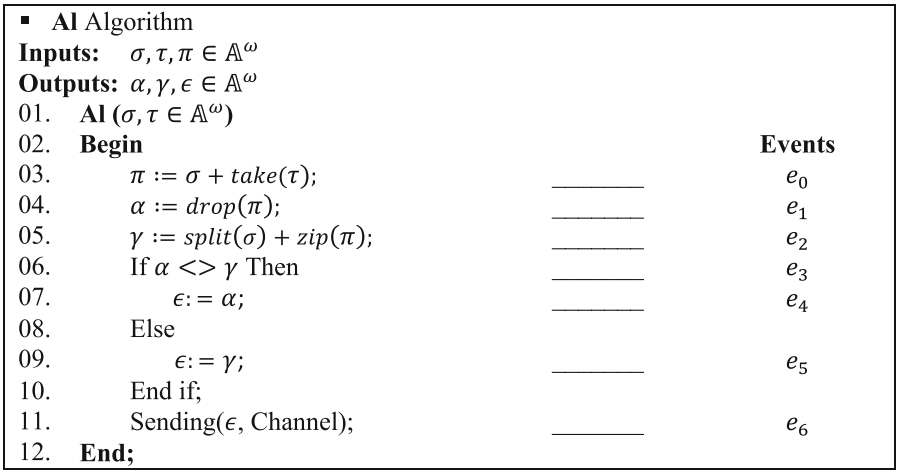


Fig. 8. An Al algorithm and its list of events

Consider the synthesis of stream calculus-based computing BDL via the above example that computes the streams ($\sigma, \tau \in \mathbb{A}^{\omega}$) and finally mapping the streams to outputs α, γ and ϵ as determined by the Al algorithm (see Fig. 8).

5.3 Generating ASM_{SPEC}

The major problem is the partial order specification creation of the program statements generated in Sect. 5.2. To put it another way, this specification is utilized to show the data dependency of statements necessary for the creation of the possible algebraic semantics-based specification of the automata, namely ASA_{SPEC}. To fulfil this task, we need to

refer to some fundamental relations between events, such as relations of independence, precedence, conflict and disjunctive enabling, as described in Sect. 2.

In approach of this paper to create ASM_{SPEC}, this paper utilizes the relations between events. The relations can be extracted from the system specification given in a high-level programming language as in Sect. 5.2. Let there be an Al algorithm with events set E, together with R relations between events, which were extracted from that algorithm specification in the following. In Fig. 8 (Al algorithm), the paper has seven events, where e_0 precedes e_1, e_0 precedes e_2, e_1 is independent of e_2, e_1 precedes e_3, e_2 precedes e_3, e_3 precedes e_4, e_3 precedes e_5, e_4 and e_5 are in conflict, and the execution of e_4 or e_5 enables the occurrence of e_6. The combination of these gives us the algebraic semantics for the Al algorithm above. Formally, ASM_{SPEC} is described as:

$$\text{ASM}_{\text{SPEC}} = \{e_0 \prec e_1, e_0 \prec e_2, e_1 \nabla e_2, e_1 \prec e_3, e_2 \prec e_3, e_3 \prec e_4, e_3 \prec e_5, e_4 \# e_5, \text{den}(e_6, e_4, e_5)\}$$

From the list of events of Al algorithm and dependency relation between the events generated in ASM_{SPEC}, such algebraic automata of the ASM_{SPEC} is generated (Fig. 9).

ASM_{SPEC} automata (ASA_{SPEC})	e_0	e_1	e_2	e_3	e_4	e_5	e_6	Logical formula (ASM_{SPEC})
s_0	0	0	0	0	0	0	0	$f = f_{e_0 \prec e_1} \wedge f_{e_0 \prec e_2} \wedge f_{e_1 \nabla e_2} \wedge$ $f_{e_1 \prec e_3} \wedge f_{e_2 \prec e_3} \wedge f_{e_3 \prec e_4} \wedge$ $f_{e_3 \prec e_5} \wedge f_{e_4 \# e_5} \wedge f_{\text{den}(e_6, e_4, e_5)} =$ $(e_0 + \overline{e_1}) \wedge (e_0 + \overline{e_2}) \wedge 1 \wedge$ $(e_1 + \overline{e_3}) \wedge (e_2 + \overline{e_3}) \wedge$ $(e_3 + \overline{e_4}) \wedge (e_3 + \overline{e_5}) \wedge$ $(\overline{e_4} + \overline{e_5}) \wedge (\overline{e_6} + e_4 + e_5) =$ $\overline{e_1}\overline{e_2}\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0e_2\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6}$ $+e_0e_1\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0e_1e_2\overline{e_4}\overline{e_5}\overline{e_6} + e_0e_1e_2e_3\overline{e_4}\overline{e_6}$ $+e_0e_1e_2e_3\overline{e_4}e_5 + e_0e_1e_2e_3\overline{e_5}\overline{e_6} + e_0e_1e_2e_3e_4\overline{e_5}$
s_1	1	0	0	0	0	0	0	
s_2	1	0	1	0	0	0	0	
s_3	1	1	0	0	0	0	0	
s_4	1	1	1	0	0	0	0	
s_5	1	1	1	1	0	0	0	
s_6	1	1	1	1	0	1	0	
s_7	1	1	1	1	0	1	1	
s_8	1	1	1	1	1	0	0	
s_9	1	1	1	1	1	0	1	

Fig. 9. ASM_{SPEC} automata (ASA_{SPEC}) of the Al algorithm

5.4 Result of RTL Synthesis

This is a result of RTL synthesis of the algorithmic behavioral specification. This result is generated from the synthesis phase and is moved to the verification phase. The RTL module [19, 26, 27] is defined by the following:

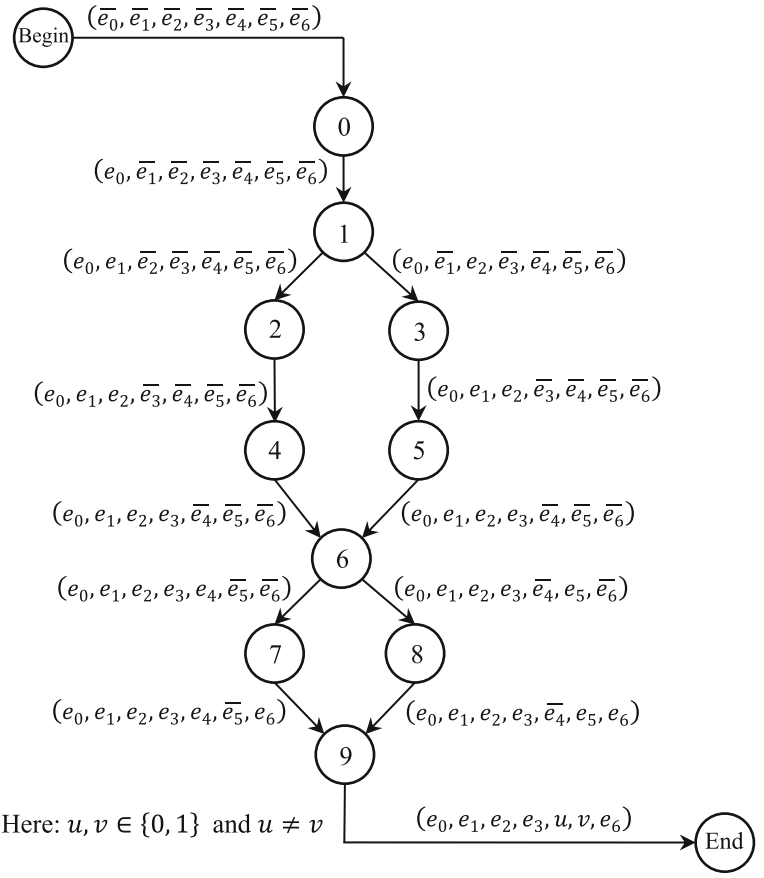


Fig. 10. RTL automata of the Al algorithm

- Processing unit that holds the declaration of the components that make up the processing unit.
- Control unit that determines the internal command sequence that must be emitted by the control unit.
- Fixed assignment that determines an operation that must be repeated every clock cycle.

The control unit is made up of steps. Each one is numbered and must be executed in a single clock unit. The Fig. 10 shows this control unit for the Al algorithm.

5.5 Generating ASA_{RTL}

ASA_{RTL} Of this paper need to be created from the result of the RTL synthesis. Thus, from the RTL synthesis of Al algorithm, as indicated in Fig. 10, ASA_{RTL} are generated in the following (see Fig. 11).

Algebraic RTL automata (ASA_{RTL})	e_0	e_1	e_2	e_3	e_4	e_5	e_6
s_0	0	0	0	0	0	0	0
s_1	1	0	0	0	0	0	0
s_2	1	0	1	0	0	0	0
s_3	1	1	0	0	0	0	0
s_4	1	1	1	0	0	0	0
s_5	1	1	1	1	0	0	0
s_6	1	1	1	1	1	0	0
s_7	1	1	1	1	0	1	0
s_8	1	1	1	1	0	1	1
s_9	1	1	1	1	1	0	1
s_{10}	1	1	1	1	u	v	1
Where: $u, v \in \{0, 1\}$ and $u \neq v$							

Fig. 11. Algebraic RTL automata (ASA_{RTL}) of the Al algorithm

5.6 Result Comparison and Discussion

5.6.1 Result Comparison Between ASA_{SPEC} and ASA_{RTL}

In comparing the ASA_{SPEC} and ASA_{RTL}, to verify the correctness of a result of RTL synthesis needs to determine the following.

Definition 3 (Correctness of a result of RTL synthesis). An algebraic semantics-based result of RTL synthesis is correct iff it satisfies all the algebraic specification requirements.

Theorem 1. An algebraic semantics-based RTL synthesis result is correct iff $\mathcal{C}(\text{RTL}) = \mathcal{C}(\text{SPEC})$ as definition 1 in Sect. 4.1. To put it another way, the computations set agreed by RTL is equivalent to the set of all computations agreed by SPEC.

Proof. For the *"if"* part, when $\mathcal{C}(\text{RTL})$ is equivalent to $\mathcal{C}(\text{SPEC})$, then an RTL synthesis result is correct according to definition 3. To demonstrate the *"only if"* part, this paper needs to demonstrate that if $\mathcal{C}(\text{RTL}) \neq \mathcal{C}(\text{SPEC})$, then the result of the RTL synthesis is not correct. Therefore, there are two cases in the following.

- If $\mathcal{C}(\text{RTL}) \subset \mathcal{C}(\text{SPEC})$ then there exists a computational requirement $\mathcal{T} \in \mathcal{C}(\text{SPEC}) \setminus \mathcal{C}(\text{RTL})$ that is not synthesized in the RTL result. By definition 3, the result of RTL synthesis is not correct.
- If $\mathcal{C}(\text{SPEC}) \subset \mathcal{C}(\text{RTL})$ then the specification and RTL synthesis are not equivalent. To take the consequence of this, their computing behaviors are distinctive. To put it another way, the result of RTL synthesis is not correct.

5.6.2 Discussion

The considerations above show that if the synthesis algorithm has created an effective RTL result and then comparing and contrasting are performed here as an assessing that examines whether algebraic RTL automata are equivalent to algebraic automata. Indeed, Fig. 9 and Fig. 11 show that $\mathcal{C}(\text{RTL}) = \mathcal{C}(\text{SPEC})$. Therefore, the formal verification approach shows the correctness of the results of the RTL synthesis for the Al algorithm.

6 Conclusions and Future Work

This paper specified an algebraic semantics for an RTL specification and a verification algorithm developed to validate the results of RTL synthesis. Major properties of this approach, the conception of an ASM (to be interpreted as a Chu space) is considered as an algebraic semantics foundation for the RTL formalization and the conception of algebraic automata are devoted for formal correctness of the result of RTL synthesis. This formal verification approach focused on functional equivalence examining to define if the algebraic RTL automata (to be interpreted as ASA_{RTL} automata) are equivalent to algebraic specification automata (to be interpreted as ASA_{SPEC} automata). For verifying the correctness of a result of RTL synthesis, we have proved that the result using an ASM is correct iff $\mathcal{C}(\text{RTL}) = \mathcal{C}(\text{SPEC})$. To put it another way, the set of all computations agreed by RTL automata (to be interpreted as ASA_{RTL} automata) is equivalent to the ones agreed by specification automata (to be interpreted as ASA_{SPEC} automata).

Furthermore, in software engineering, model-based verification (MBV) is viewed as a systematic approach to find lacks in requirements, designs or coding. MBV integrates a mathematical models form to offer an approaching logically analyzing rather than a proof of correctness strategy. MBV requires generating crucial models of the system behavior and analyzing these models based on formally representing of expected properties. Unfortunately, the specific techniques and engineering practices of applying MBV to stream calculus-based computing BDL verification have yet to be fully explored and documented. A number of challenges to the acceptance of MBV for stream calculus-based computing BDL verification have been identified, including the lack of good process support and tools for formal modeling and analysis. An essential ingredient of stream calculus is recognized as the stream bisimulation conception, which efficiently supports a stream calculus-based computing BDL approach to MBV.

References

1. Godefroid, P., Peled, D., Staskauskas, M.: Using partial-order methods in the formal validation of industrial concurrent programs. In: Presented at the Proceedings of the 1996 ACM SIGSOFT International Symposium on Software testing and analysis, San Diego, California, USA (1996)
2. Hansen, C., Kunzmann, A., Rosenstiel, W.: Verification by simulation comparison using interface synthesis. In: Proceedings Design, Automation and Test in Europe, pp. 436–443 (1998). https://doi.org/10.1109/DATE.1998.655894
3. Hansen, C., Nascimento, F.A.M.D., Rosenstiel, W.: Verifying high level synthesis results using a partial order based model. In: International High Level Design Validation and Test Workshop, San Diego, CA, USA, pp. 135–141 (1998)

4. Kordon, F., Hillah, L.M., Hulin-Hubard, F., Jezequel, L., Paviot-Adet, E.: Study of the efficiency of model checking techniques using results of the MCC from 2015 To 2019. Int. J. Softw. Tools Technol. Transfer **23**(6), 931–952 (2021). https://doi.org/10.1007/s10009-021-00615-1
5. Souri, A., Norouzi, M.: A state-of-the-art survey on formal verification of the internet of things applications. J. Serv. Sci. Res. **11**(1), 47–67 (2019). https://doi.org/10.1007/s12927-019-0003-8
6. Alur, R.: Formal verification of hybrid systems. In: Presented at the Proceedings of the ninth ACM International Conference on Embedded Software, Taipei, Taiwan (2011). https://doi.org/10.1145/2038642.2038685
7. Anh, V.T., Vinh, P.C., Cuong, P.Q.: Contextual perception in internet of mobile things: a categorical structure. Internet Things **22**, 100799 (2023). https://doi.org/10.1016/j.iot.2023.100799
8. Maruyama, Y.: Chu duality theory and coalgebraic representation of quantum symmetries. J. Pure Appl. Algebra **226**(9), 106960 (2022). https://doi.org/10.1016/j.jpaa.2021.106960
9. Myojin, S., Babaguchi, N.: A logical consideration on fraudulent email communication. Artif. Life Robot. **25**(3), 475–481 (2020). https://doi.org/10.1007/s10015-020-00597-4
10. Ivanov, L.: Automatic generation of Chu space model expressions for verification. In: 51st Midwest Symposium on Circuits and Systems, pp. 613–616 (2008). https://doi.org/10.1109/MWSCAS.2008.4616874
11. Peled, D.: Partial-Order Reduction. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 173–190. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_6
12. Droste, M., Zhang, G.-Q.: Bifinite chu spaces. EPI Sciences Overlay Journal, lmcs:1183 - Logical Methods Comput. Sci. **6**(1) (2010). https://doi.org/10.2168/LMCS-6(1:3)2010
13. Xu-Tao, Du., Xing, C.-X., Zhou, L.-Z.: Modeling and verifying concurrent programs with finite chu spaces. J. Comput. Sci. Technol. **25**(6), 1168–1183 (2010). https://doi.org/10.1007/s11390-010-9397-y
14. Fields, C., Glazebrook, J.F.: Information flow in context-dependent hierarchical Bayesian inference. J. Exp. Theor. Artif. Intell. **34**(1), 111–142 (2022). https://doi.org/10.1080/0952813X.2020.1836034
15. Huang, F.-P., Droste, M., Zhang, G.-Q.: A monoidal category of bifinite chu spaces. Electron. Notes Theor. Comput. Sci. **212**, 285–297 (2008). https://doi.org/10.1016/j.entcs.2008.04.068
16. Buchenrieder, K.: Rapid prototyping of embedded hardware/software systems. Des. Autom. Embed. Syst. **5**(3), 215–221 (2000). https://doi.org/10.1023/A:1008950900254
17. Scheide, W., Georg, D., Helmut, K.: Using Conjoint Analysis for Software Engineering. In: CONSYSE '97 International Workshop on Conjoint Systems Engineering: ITpress Verlag, Konferenzbeitrag (1997)
18. Ulidowski, I., Phillips, I., Yuen, S.: Reversing event structures. New Gener. Comput. **36**(3), 281–306 (2018). https://doi.org/10.1007/s00354-018-0040-8
19. Van Pham, D., Phan, V.C., Nguyen, B.K.: Formally specifying and coinductive approach to verifying synthesis of stream calculus-based computing big data in livestream. Internet Things **23**, 100878 (2023). https://doi.org/10.1016/j.iot.2023.100878
20. Van Pham, D., Phan, V.C.: Overview of the stream theory-based big data in livestream. Mobile Netw. Appl. (2023). https://doi.org/10.1007/s11036-023-02180-0
21. Rutten, J.J.M.M.: Elements of stream calculus. Electron. Notes Theor. Comput. Sci. **45**, 358–423 (2001). https://doi.org/10.1016/S1571-0661(04)80972-1
22. Niqui, M., Rutten, J.J.M.M.: Stream processing coalgebraically. Sci. Comput. Program. **78**(11), 2192–2215 (2013). https://doi.org/10.1016/j.scico.2012.07.013

23. Helala, M.A., Qureshi, F.Z., Pu, K.Q.: A stream algebra for performance optimization of large scale computer vision pipelines. IEEE Trans. Pattern Anal. Mach. Intell. **44**(2), 905–923 (2022). https://doi.org/10.1109/TPAMI.2020.3015867
24. Hansen, H.H., Kupke, C., Rutten, J.J.M.M.: Stream differential equations: specification formats and solution methods. Log. Methods Comput. Sci. **13**(1) (2017). https://doi.org/10.23638/LMCS-13(1:3)2017
25. Rutten, J.J.M.M.: The Method of Coalgebra: exercises in coinduction. CWI, Amsterdam, The Netherlands, p. 261 (2019). ISBN: 978-90-6196-568-8
26. Carchiolo, V., Malgeri, M., Mangioni, G.: Hardware/software synthesis of formal specifications in codesign of embedded systems. ACM Trans. Des. Autom. Electron. Syst. **5**(3), 399–432 (2000). https://doi.org/10.1145/348019.348093
27. Bailey, B., Gajski, D.: RTL semantics and methodology. In: International Symposium on System Synthesis (IEEE Cat. No.01EX526), pp. 69–74 (2001). https://doi.org/10.1145/500001.500017