MINISTRY OF EDUCATION AND TRAINING

VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY

GRADUATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

PHAM VAN DANG

STREAM ALGEBRA-BASED ANALYTICS OF BIG DATA IN LIVESTREAM

SUMMARY OF DISSERTATION ON COMPUTER

Major: Computer Science

Code: 9 48 01 01

Ha Noi – Year 2025

The dissertation is completed at: Graduate University of Science and Technology, Vietnam Academy Science and Technology

Supervisors:

1. **Supervisor 1:** Prof. Phan Cong Vinh, Nguyen Tat Thanh University, Ho Chi Minh City, Viet Nam
2. **Supervisor 2:** Dr. Tran Trong Toan, Institute of Information Technology, Vietnam Academy of Science and Technology, Ho Chi Minh City, Viet Nam

Referee 1:..

....

Referee 2:..

....

Referee 3:..

....

The dissertation is examined by Examination Board of Graduate University of Science and Technology, Vietnam Academy of Science and Technology at…………………………………..(time ………………, date…………………………)

The dissertation can be found at:

1. Graduate University of Science and Technology Library
2. National Library of Vietnam

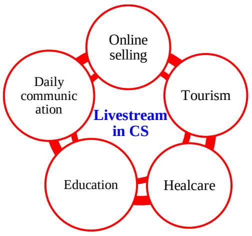
INTRODUCTION

1. Rationale for selecting the dissertation topic

The big data in livestream (BDL) are existing at various forms. However, so far, the author has not found any mathematical foundation that formally defines, explains, and verifies the operational mechanisms of BDL.

Livestream is a widely adopted formalization supporting various domains (Figure 1), generating vast amounts of data such as text, images, audio, and video real-time. Despite its rapid growth, there is currently no formally theoretical foundation that explains its operational mechanisms.

This dissertation develops the stream operators, composes them into stream expressions, algebraic aspects, scheduling, and algebraic semantics models to formalize the operational mechanism of BDL streams (Figure 2).



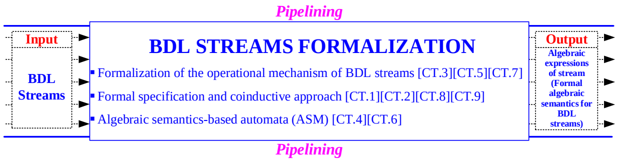

Pipelining
Input
BDL Streams
BDL STREAMS FORMALIZATION
Formalization of the operational mechanism of BDL streams [CT.3][CT.5][CT.7]
Formal specification and coinductive approach [CT.1][CT.2][CT.8][CT.9]
Algebraic semantics-based automata (ASM) [CT.4][CT.6]
Output
Algebraic expressions of stream (Formal algebraic semantics for BDL streams)
Pipelining

Figure 2. Formalization of operational mechanism of BDL streams within computing systems (CS)

Figure 1. Widely adopted livestream formalization give rise to BDL streams in CS

2. Scientific challenges

Based on the descriptions and analyses of Figures 1 and 2, along with stream theory, this dissertation encounters the following challenges in formalizing BDL streams:

- Challenge abstraction in the modeling BDL streams.
- Challenge reasoning in modeling BDL streams.
- Challenge formal verification of BDL streams.
- Challenge related to the intrinsically nature of BDL streams.

3. Dissertation objectives

This dissertation objective is to develop several algebraic aspects, optimize scheduling, and construct an algebraic semantics-based model (ASM) to formalize (modeling, reasoning, verification) the operational mechanism of streams, thereby fulfilling the algebraic semantics of BDL streams in CS.

4. New contributions of the dissertation

- To develop ten stream processing operators, their composition into stream expressions, extension of stream algebra and stream coalgebra, monoid algebra structure and monoid properties used to BDL streams in IoMT [CT.1][CT.2][CT.3][CT.5][CT.7][CT.8].

- To develop a scheduling for processing BDL streams and to optimize pipeline scheduling specifically used to BDL [CT.2][CT.9].
- To develop an algebraic semantics-based model (ASM) to specify the relation between the ASA_{RTL} and ASA_{SPEC} for verifying the consistency between the results of RTL and SPEC synthesis [CT.4][CT.6].

5. Structure of the dissertation

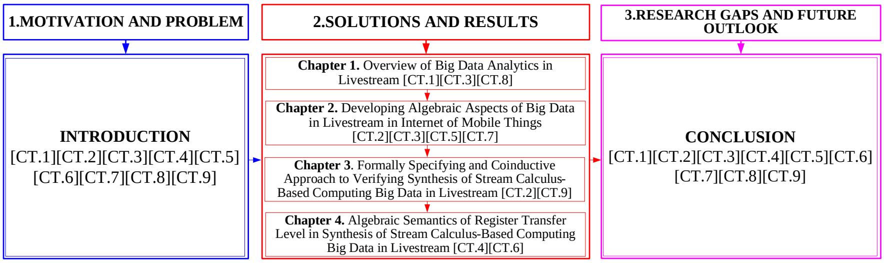

1.MOTIVATION AND PROBLEM
2.SOLUTIONS AND RESULTS
3.RESEARCH GAPS AND FUTURE OUTLOOK
INTRODUCTION [CT.1][CT.2][CT.3][CT.4][CT.5][CT.6][CT.7][CT.8][CT.9]
Chapter 1. Overview of Big Data Analytics in Livestream [CT.1][CT.3][CT.8]
Chapter 2. Developing Algebraic Aspects of Big Data in Livestream in Internet of Mobile Things [CT.2][CT.3][CT.5][CT.7]
Chapter 3. Formally Specifying and Coinductive Approach to Verifying Synthesis of Stream Calculus-Based Computing Big Data in Livestream [CT.2][CT.9]
Chapter 4. Algebraic Semantics of Register Transfer Level in Synthesis of Stream Calculus-Based Computing Big Data in Livestream [CT.4][CT.6]
CONCLUSION [CT.1][CT.2][CT.3][CT.4][CT.5][CT.6][CT.7][CT.8][CT.9]

Figure 3. Diagram of structure of the dissertation

CHAPTER 1: OVERVIEW OF BIG DATA ANALYTICS IN LIVESTREAM

1.1. Online generation of big data

Data lies at the core of research and innovation, driving advances in processes, algorithms, mathematical model, and formalization. The widespread adoption of smart devices, social networks, and the web has significantly increased the volume of online big data, particularly in BDL.

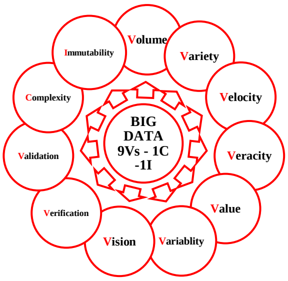


Figure 1.1. Core characteristics of big data: 9Vs – 1C – 1I

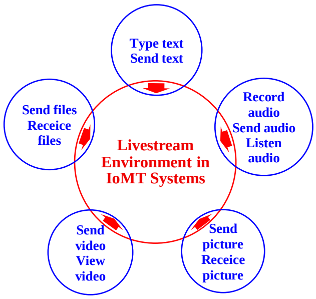

Livestream Environment in IoMT Systems
Type text Send text
Record audio Send audio Listen audio
Send picture Receice picture
Send video View video
Send files Receice files

Figure 1.2. Livestream activities on social media platforms

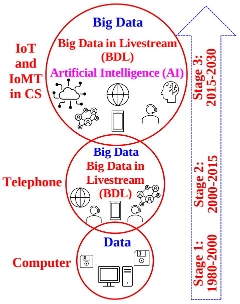


Figure 1.6. Evolution of data-generating devices [CT.7]

1.2. Core characteristics defining big data

Several technical and technological challenges are associated with the eleven core characteristics of big data: 9Vs, 1C, and 1I [CT.1] (Figure 1.1).

1.3. Livestream activities

The analysis of BDL streams involves continuous data processing, making it suitable for diverse data sources (structured, semi-structured, unstructured) and for real-time transmission. Livestream has emerged as a widely adopted mechanism in recent years (Figure 1.2).

1.4. Big data in livestream (BDL)

The evolution of electronic devices has been closely associated with the growth of big data sources across different stages (Figure 1.6) [CT.7]. BDL within the BD faces both theoretical and technical challenges from the eleven characteristics presented in Figure 1.1, and it is conceptualized as a linear frame sequence consisting of text, images, audio, video, etc. [CT.1][CT.2].

1.5. Similarities and differences between BD and BDL

Table 1.1. Comparative similarities and differences between BD and BDL

BD		Characteristics	BDL – Big data in livestream	
♦		7**V**, 1**C**, 1**I**	♦	
Stream abstraction	✓	**V**ariability	• Real-time rapidly changing stream • Large-scale real-time stream • Stream with high variability • Stream with interleaved frames	✓
		Vision	• Visually observable online stream • Stream responsive to visual perception • Real-time volatile stream • Stream supporting decision-making responsiveness	✓
Note: ✓ represents the difference; ♦ represents the similarity.				

1.6. Pipeline in BDL stream processing

1.7. Stream theory-based analytics of BDL streams

1.7.1. Fundamental concepts

a) Framed data within BDL streams

In [CT.1][CT.2], the basic notations and foundational concepts related to BDL streams are defined as follows: The symbols $\sigma, \tau, \rho, \alpha, \ldots$ representing streams, where $\sigma \in \mathbb{A}^{\omega}$ represents the sequence $(\sigma(0), \sigma(1), \ldots)$, and its derivative is defined as $\sigma' = (\sigma(1), \sigma(2), \ldots)$. The set of operator includes drop, take, zip, split, rev, merge, $\times$, $\bullet$, ▪ , and :=. The $\mathbb{A}^{\omega}$ denotes set of streams, while $\mathbb{A}$ denotes set of frames (e.g., text, images, audio, video, etc.).

b) Scheduling

c) Control step (CStep)

1.7.2. Strengths and limitations of stream algebra

1.7.3. Strengths and limitations of stream coalgebra
1.7.4. The role of ten stream operators for processing BDL streams
1.8. Formulating the research problems

In the field of stream processing, although stream theory has made significant contributions, there remain notable limitations regarding the availability of formalizations for BDL streams. This dissertation formulates three research problems, as illustrated in Figure 1.10.

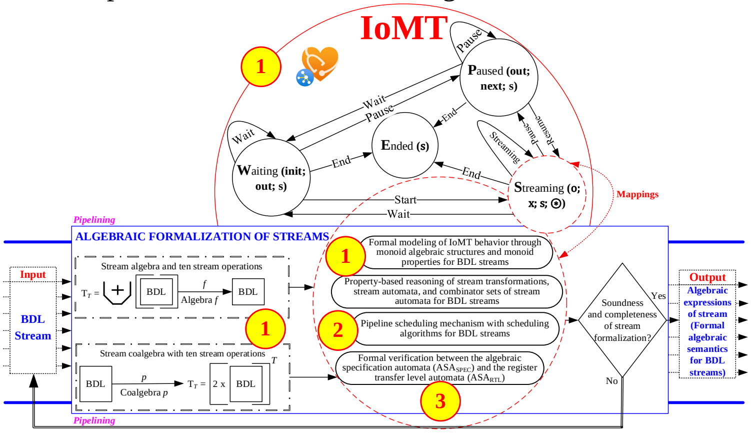

IoMT
1
Paused (out; next; s)
Pause
Wait
Pause
End
Resume
Pause
Streaming
Ended (s)
Wait
Waiting (init; out; s)
End
End
Streaming (o; x; s; ⊛)
Start
Wait
Mappings
Pipelining
ALGEBRAIC FORMALIZATION OF STREAMS
Input
BDL Stream
Stream algebra and ten stream operations
BDL
f
Algebra f
BDL
1
Stream coalgebra with ten stream operations
BDL
p
Coalgebra p
2 x BDL
1
Formal modeling of IoMT behavior through monoid algebraic structures and monoid properties for BDL streams
Property-based reasoning of stream transformations, stream automata, and combinator sets of stream automata for BDL streams
2
Pipeline scheduling mechanism with scheduling algorithms for BDL streams
Formal verification between the algebraic specification automata (ASA_{SPEC}) and the register transfer level automata (ASA_{RTL})
3
Soundness and completeness of stream formalization?
Yes
No
Output
Algebraic expressions of stream (Formal algebraic semantics for BDL streams)
Pipelining

Figure 1.10. The formalization framework of streams illustrates the three core research problems addressed in this dissertation concerning the formalization of the operational mechanism of BDL streams within IoMT.

CHAPTER 2: DEVELOPING ALGEBRAIC ASPECTS OF BIG DATA IN LIVESTREAM IN INTERNET OF MOBILE THINGS

2.1. Introduction to the computing system

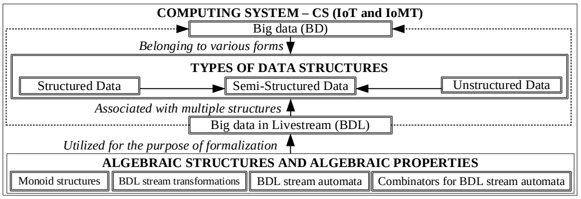

COMPUTING SYSTEM – CS (IoT and IoMT)
Big data (BD)
Belonging to various forms
TYPES OF DATA STRUCTURES
Structured Data
Semi-Structured Data
Unstructured Data
Associated with multiple structures
Big data in Livestream (BDL)
Utilized for the purpose of formalization
ALGEBRAIC STRUCTURES AND ALGEBRAIC PROPERTIES
Monoid structures
BDL stream transformations
BDL stream automata
Combinators for BDL stream automata

Figure 2.1. Formalization of BDL streams in CS

The rapid proliferation of electronic devices has led to the generation of massive data volumes, including BDL streams of complex characteristics (Figures 1.1 and 1.6). Based on the characteristics of streams (Subsection 1.2, Figures 1.1 and 2.1) and the limitations in formalization (Figure 1), this dissertation develops algebraic aspects to formalize stream operations.

2.2. Stream theory

2.2.1. Stream algebra-based analytics of BDL streams

BDL streams involves the initial value $\sigma(0)$, the stream derivative σ', behavioral properties, and differential equations. This dissertation develops ten stream operators (drop, take, zip, split, rev, merge, $\times$, $\bullet$, $\blacksquare$, and $:=$) as presented in Subsections 2.3.3.1 - 2.3.3.10, and composes them into stream expressions in Subsection 2.3.4 of Chapter 2. A unary stream operator $\mathbb{A}^{\omega} \to \mathbb{A}^{\omega}$, denoted by $\boxdot \in \{drop, take, zip, split, rev\}$, is defined by the following equation: $\boxdot: \sigma \in \mathbb{A}^{\omega} \to \boxdot\,(\sigma) \in \mathbb{A}^{\omega}$. A binary stream operator $\mathbb{A}^{\omega} \times \mathbb{A}^{\omega} \to \mathbb{A}^{\omega}$, denoted by $\circledast \in \{+,\ \times, :=\}$, is defined by the following equation: $\circledast: \sigma \in \mathbb{A}^{\omega} \times \tau \in \mathbb{A}^{\omega} \to \sigma \circledast \tau \in \mathbb{A}^{\omega}$ [CT.2]. The proposed stream function is defined as $\sigma: \mathbb{N} \to \mathbb{A}^{\omega}$, where $\sigma = (\sigma(0), \sigma(1), \dots)$, $\mathbb{A}^{\omega}$ denotes the set of streams, and $\mathbb{A}$ denotes the set of frames [CT.1][CT.2].

Theorem 2.1: For all streams $\sigma \in \mathbb{A}^{\omega}$ then $\sigma = \sigma(0) + (X \times \sigma')$.

2.2.2. Streams for BDL analysis

$\mathbb{A}^{\omega}$ is defined over $\mathbb{A}$, where the set $\mathbb{A}^{\omega} = \{\sigma \mid \sigma: \mathbb{N} \to \mathbb{A}\}$ represents BDL. Each stream $\sigma = (\sigma(0), \sigma(1), \dots) \in \mathbb{A}^{\omega}$ consists of an initial value $\sigma(0)$ and a derivative $\sigma' = (\sigma(1), \sigma(2), \dots)$. For every $n \geq 0$, the higher-order derivatives are defined as $\sigma^{(0)} = \sigma$ and $\sigma^{(n+1)} = \left(\sigma^{(n)}\right)'$ [CT.1, CT.2].

2.2.3. Stream coalgebra-based analytics of BDL streams

Definition 2.14 (BDL stream automata): A (finite or infinite) set of stream operators is denoted by T. A BDL stream in IoMT is defined as a stream automata of the type $\langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle$, where: T is the set of operators, IoMT is the set of states, $o_{IoMT}: IoMT \to (T \to 2)$ is the output function, and $e_{IoMT}: IoMT \to (T \to IoMT)$ is the evolution function.

Definition 2.15 (Stream coalgebra of BDL streams): The set of BDL streams denoted as $\{\langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle\}$, consisting of all finite or infinite stream carriers, forms a coalgebraic structure defined by a pair of operators $\langle O, E \rangle$ of the type: $\{\langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle\} \xrightarrow[\langle O,E \rangle]{} o_{IoMT} \times \{\langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle\}$, where the BDL stream element $\langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle$ is mapped to the pair $\langle \langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle(0), \langle IoMT, \langle o_{IoMT}, e_{IoMT} \rangle \rangle' \rangle$.

2.2.4. A simulation and verification problem to evaluate stream operators

a) Problem formulation for verifying the stream operators

Problem formulation: Given a stream σ_i consisting of multiple frames (Each frame being a tuple ⟨text, image, audio, video, etc.⟩) as input to six stream operators.

Scenarios: To verify the semantics that preserve the homogeneity of the original stream, maintain its subsets, and quantify key performance indicators (KPIs) including TP, LA, LP, BA, BP, FI, and IO given the six stream operators. In other words, given the six stream operators, the dissertation verifies their semantics and quantifies the KPIs for a stream σ_i composed of multiple frames.

b) Input/output analysis for simulation algorithm of operators

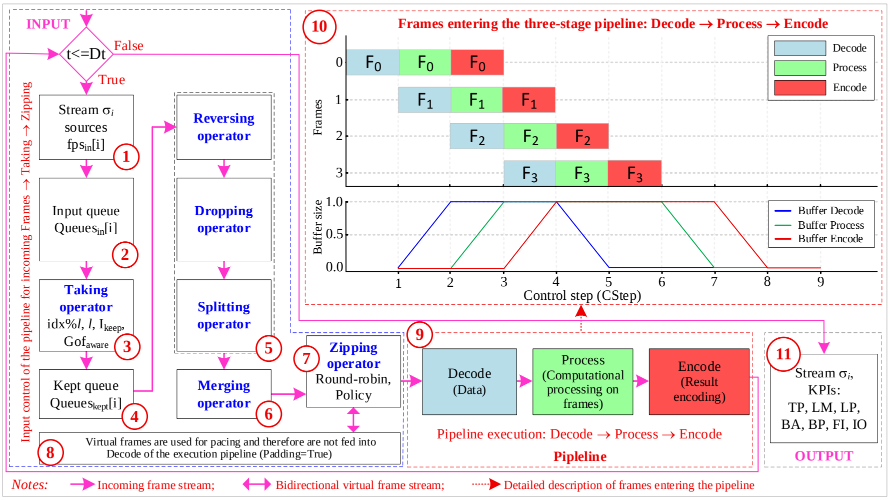

INPUT
t<=Dt
False
True
Input control of the pipeline for incoming Frames → Taking → Zipping
Stream σi sources fpsin[i]
Input queue Queuesin[i]
Taking operator idx%l, l, Ikeep, Gofaware
Kept queue Queueskept[i]
Reversing operator
Dropping operator
Splitting operator
Merging operator
Zipping operator Round-robin, Policy
Virtual frames are used for pacing and therefore are not fed into Decode of the execution pipeline (Padding=True)
Frames entering the three-stage pipeline: Decode → Process → Encode
Frames
Buffer size
Control step (CStep)
Decode
Process
Encode
Buffer Decode
Buffer Process
Buffer Encode
Decode (Data)
Process (Computational processing on frames)
Encode (Result encoding)
Pipeline execution: Decode → Process → Encode
Pipleline
Stream σi, KPIs: TP, LM, LP, BA, BP, FI, IO
OUTPUT
Notes: Incoming frame stream; Bidirectional virtual frame stream; Detailed description of frames entering the pipeline

Figure 2.10. Input control diagram of incoming frames in the pipeline

Figure 2.10 illustrates the input control mechanism of the pipeline, consisting of eleven blocks: Block ① represents the sources generating the streams σ_i; Block ② inserts frames into Queues$_{\text{in}}[i]$; Block ③ applies the taking operator to select frames by source; Block ④ stores the accepted frames into Queues$_{\text{kept}}[i]$; Block ⑤ includes the reversing, dropping, and splitting operators; Block ⑥ performs the merging operator to combine streams; Block ⑦ applies the zipping operator to interleave multiple sources before entering the pipeline; Block ⑧ performs padding to generate virtual frames when a source is empty; Blocks ⑨–⑩ represent the execution pipeline that processes frames F_0–F_3 at each time step. The three main components Taking ③, Zipping ⑦, and Pipeline ⑨ jointly govern Block ⑪, where the KPIs (TP, LM, LP, BA, BP, FI, and IO) are generated.

c) Simulation algorithm for stream processing operators

d) The roles of stream processing operators in the simulation scenarios

e) Data for configuration parameters of the simulation scenarios

The dissertation provides the simulation configuration parameter data in the Table 2.23. Simulation environment: colab.research.google.com and the Python programming language.

Table 2.23. Configuration parameter data for each simulation scenario

Scenarios	Simulation configuration parameters for the six stream processing operators													
	Take and Zip							Rev	Drop		Split		Merge
	T	M	fps_{in}	l	I_{keep}	$\text{Gof}_{\text{aware}}$	P	l	l	k	l	k	Use
KB1	5	1	[60]	4	[0,1,2,3]	False	Skip	5	8	2	8	[0,2]	False
KB2	5	2	[60,45]	4	[1,3]	False	Skip	4	4	2	4	[0,2]	True
KB3	5	3	[60,45,30]	4	[1,3]	True	Skip	3	3	2	3	[0,2]	True
KB4	5	4	[60, 45, 30, 20]	4	[0,2]	False	Skip	5	5	2	5	[0,2]	True
KB5	5	5	[60, 45, 30, 20, 15]	4	[1,3]	False	Skip	6	6	2	6	[0,2]	True
KB6	5	5	[60, 30, 45, 15, 10]	4	[1,3]	False	Skip	7	7	2	7	[0,2]	True

f) Running simulations and collecting KPIs results

Using the configuration parameters in Table 2.23, the simulation was executed to obtain the quantified KPIs results. The detailed simulation results for the six scenarios are presented in Table 2.24.

Table 2.24. Comparative KPIs across simulation scenarios

Scenarios	The KPIs are quantified based on the configuration parameters.						
	TP	LM	LP	BA	BP	FI	IO
KB1	60.0	35.0000	35.0	2.0949	3.0	1.0000	0.0
KB2	52.4	36.2023	40.0	1.8921	5.0	0.9794	0.5
KB3	77.4	37.2739	45.0	2.8551	5.0	0.9276	0.5
KB4	65.0	38.3692	45.0	2.3926	5.0	0.7958	0.5
KB5	84.8	40.0825	50.0	3.2927	6.0	0.8073	0.5
KB6	79.8	38.7093	45.0	3.0200	6.0	0.7463	0.5

Analysis of the KPI results in Table 2.24 shows that the KPIs clearly reflect the impact of each policy for taking and zipping on the operational efficiency of the pipeline. Specifically, five *questions/answers* are formulated to analyze how variations in the simulation configuration significantly affect TP, LM, LP, FI, IO as well as BA and BP.

2.3. Formalization framework for BDL streams in IoMT

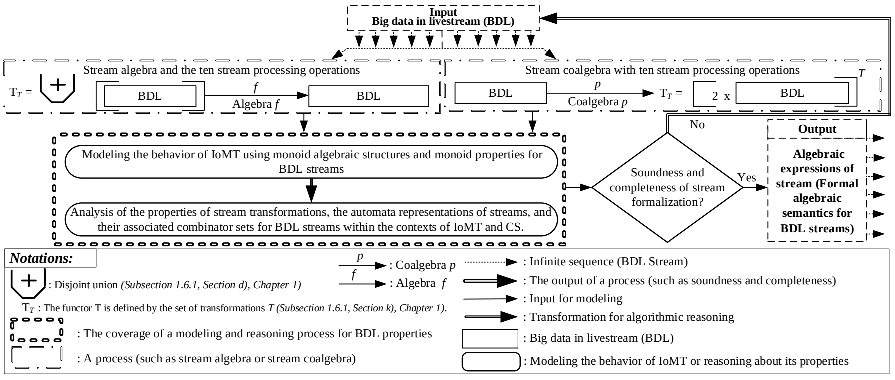

Input
Big data in livestream (BDL)
Stream algebra and the ten stream processing operations
T_T =
BDL
f
Algebra f
BDL
Stream coalgebra with ten stream processing operations
BDL
p
Coalgebra p
T_T =
2 x
BDL
T
No
Modeling the behavior of IoMT using monoid algebraic structures and monoid properties for BDL streams
Analysis of the properties of stream transformations, the automata representations of streams, and their associated combinator sets for BDL streams within the contexts of IoMT and CS.
Soundness and completeness of stream formalization?
Yes
Output
Algebraic expressions of stream (Formal algebraic semantics for BDL streams)
Notations:
: Disjoint union (Subsection 1.6.1, Section d), Chapter 1)
T_T : The functor T is defined by the set of transformations T (Subsection 1.6.1, Section k), Chapter 1).
: The coverage of a modeling and reasoning process for BDL properties
: A process (such as stream algebra or stream coalgebra)
: Coalgebra p
: Algebra f
: Infinite sequence (BDL Stream)
: The output of a process (such as soundness and completeness)
: Input for modeling
: Transformation for algorithmic reasoning
: Big data in livestream (BDL)
: Modeling the behavior of IoMT or reasoning about its properties

Figure 2.11. Formalization framework for BDL streams in IoMT

The formalization of BDL streams poses significant challenges in both stream algebra and stream coalgebra. This dissertation develops algebraic aspects to model the operational mechanism of streams [CT.1][CT.2][CT.3][CT.5][CT.7]. Stream computation is represented as a morphism $IoMT^{\omega} \rightarrow IoMT^{\omega}$, establishing a pipeline that satisfies the algebraic semantics of BDL streams within IoMT (Figure 2.11).

2.4. Monoid structure of BDL in IoMT

2.4.1. Monoid structure for modeling BDL streams in IoMT

Applying stream operators within a monoid structure serves as a foundational approach for processing BDL streams. This dissertation introduces the symbol $\circledast$ to denote binary stream operators, which are used to model the automated processing of individual frames in IoMT [CT.2].

Definition 2.16 (Monoid structure): A monoid is a triple $(IoMT, \circledast, 1)$, where $IoMT$ is a set of finite states or data stream types, specifically referring to BDL. The binary operator $\circledast\colon IoMT \times IoMT \rightarrow IoMT$ is defined over *IoMT* and represents the concatenation of frames in the context of stream processing. The element $1 \in IoMT$ is the identity element, such that the following two axioms are satisfied (2.23) and (2.24).

- Associativity: $(x \circledast y) \circledast z = x \circledast (y \circledast z), \forall x, y, z \in IoMT$ (2.23)
- Identity: $1 \circledast x = x \circledast 1 = x, \forall x \in IoMT.$ (2.24)

Definition 2.17 (Product of two monoids): Let $(IoMT_{1,2}, \circledast_{IoMT_{1,2}}, 1_{IoMT_{1,2}})$ be two monoid structures. Then, the product of these two monoids is defined as a monoid $(IoMT_1 \times IoMT_2, \circledast, 1)$, where: The binary stream operator $\circledast$ is given by: $(x, y) \circledast (x', y') = (x \circledast_{IoMT_1} x', y \circledast_{IoMT_2} y')$ for all $x, x' \in IoMT_1$ and $\forall y, y' \in IoMT_2$. The identity element is defined as $1 = (1_{IoMT_1}, 1_{IoMT_2})$.

Definition 2.18 (Monoid homomorphism for BDL streams): Let $(IoMT_{1,2}, \circledast, 1)$ be two monoids. A mapping $h\colon IoMT_1 \rightarrow IoMT_2$ is called a monoid homomorphism if it satisfies the following two conditions: Preservation of the identity element: $h(1) = 1$. Preservation of the stream operator: $h(x \circledast y) = h(x) \circledast h(y)$ for all $x, y \in IoMT_{1,2}$.

2.4.2. Formalization of BDL streams in IoMT

This dissertation describes the relation of monoid structure and monoid properties used to formalize the operational mechanism of BDL streams in IoMT, as illustrated in Figure 2.12.

- The monoid (IoMT, $\circledast$, 1) is an algebraic structure used to formalize operational mechanism of BDL in IoMT, along with its associated properties as presented in subsections 2.5.1 and 2.6 of chapter 2.

- Applicable to the set of finite states (Subsection 2.6.7, chapter 2).
- Applicable to types of data streams.
- Used to describe the operational mechanism of BDL streams in IoMT.
- Scientific and practical significance:
    - Real-time processing of BDL data streams in IoMT.
    - Maintaining livestream continuity even in the presence of data errors.
    - Reducing bandwidth consumption and improving processing speed.
- Order relations for monoid specification (Subsection 2.6.1, Chapter 2): Applied to monoid properties that satisfy the ordering of frames, streams, and states in IoMT.
- Monoid properties (Subsections 2.6.2–2.6.5, chapter 2): Used to specify the finiteness of BDL within the set of finite states of the automata.
- BDL transformation in IoMT ($IoMT_{1,2}$) (Subsection 2.6.6, Chapter 2): This transformation maps BDL from the source state set $IoMT_1$ to the target state set $IoMT_2$.
- Automata for BDL streams in IoMT (Subsection 2.6.7, Chapter 2): This subsection proposes a set of automata, including: formalization, a BDL counter, a BDL merger, a BDL flattening, and a BDL batch-splitting.
- Four combinators for BDL streams automata (Subsection 2.6.8, Chapter 2): These describe the sequential, parallel, merging, and feedback compositions of automata applied to BDL frames or streams.

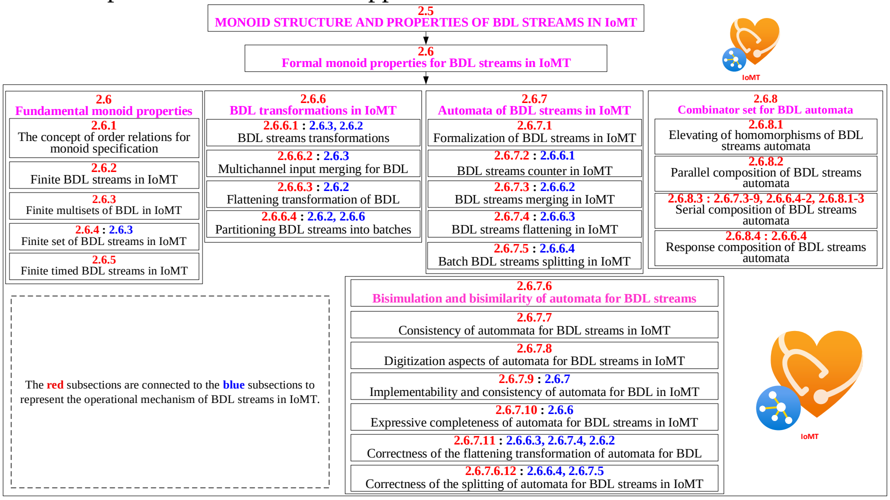

2.5
MONOID STRUCTURE AND PROPERTIES OF BDL STREAMS IN IoMT
2.6
Formal monoid properties for BDL streams in IoMT
IoMT
2.6
Fundamental monoid properties
2.6.1
The concept of order relations for monoid specification
2.6.2
Finite BDL streams in IoMT
2.6.3
Finite multisets of BDL in IoMT
2.6.4 : 2.6.3
Finite set of BDL streams in IoMT
2.6.5
Finite timed BDL streams in IoMT
2.6.6
BDL transformations in IoMT
2.6.6.1 : 2.6.3, 2.6.2
BDL streams transformations
2.6.6.2 : 2.6.3
Multichannel input merging for BDL
2.6.6.3 : 2.6.2
Flattening transformation of BDL
2.6.6.4 : 2.6.2, 2.6.6
Partitioning BDL streams into batches
2.6.7
Automata of BDL streams in IoMT
2.6.7.1
Formalization of BDL streams in IoMT
2.6.7.2 : 2.6.1
BDL streams counter in IoMT
2.6.7.3 : 2.6.6.2
BDL streams merging in IoMT
2.6.7.4 : 2.6.6.3
BDL streams flattening in IoMT
2.6.7.5 : 2.6.6.4
Batch BDL streams splitting in IoMT
2.6.8
Combinator set for BDL automata
2.6.8.1
Elevating of homomorphisms of BDL streams automata
2.6.8.2
Parallel composition of BDL streams automata
2.6.8.3 : 2.6.7.3-9, 2.6.6.4-2, 2.6.8.1-3
Serial composition of BDL streams automata
2.6.8.4 : 2.6.6.4
Response composition of BDL streams automata
2.6.7.6
Bisimulation and bisimilarity of automata for BDL streams
2.6.7.7
Consistency of autommata for BDL streams in IoMT
2.6.7.8
Digitization aspects of automata for BDL streams in IoMT
2.6.7.9 : 2.6.7
Implementability and consistency of automata for BDL in IoMT
2.6.7.10 : 2.6.6
Expressive completeness of automata for BDL streams in IoMT
2.6.7.11 : 2.6.6.3, 2.6.7.4, 2.6.2
Correctness of the flattening transformation of automata for BDL
2.6.7.6.12 : 2.6.6.4, 2.6.7.5
Correctness of the splitting of automata for BDL streams in IoMT
The red subsections are connected to the blue subsections to represent the operational mechanism of BDL streams in IoMT.
IoMT

Figure 2.12. Connections of monoid structure and properties for BDL streams

2.4.3. Formal automata for BDL streams in IoMT

This dissertation models the operational mechanism and computation of

BDL stream processing in IoMT using the notations (init, out, s, o, x, next, $\circledast$) on the state diagram (Figure 2.13), where the input and output are frames belonging to two monoids ($IoMT_1$,$\circledast$,1) and ($IoMT_2$,$\circledast$,1).

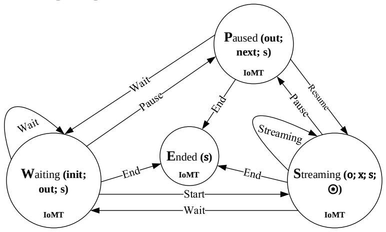


Figure 2.13. State transition diagram of a livestream system in IoMT

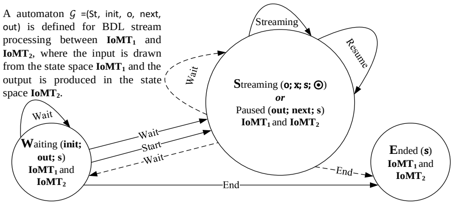

A automaton $\mathcal{G}$ =(St, init, o, next, out) is defined for BDL stream processing between **IoMT₁** and **IoMT₂**, where the input is drawn from the state space **IoMT₁** and the output is produced in the state space **IoMT₂**.

Figure 2.14. State transition diagram for the livestream system (IoMT_1 and IoMT_2)

The dissertation proposes an automaton $\mathcal{G} = (\text{St}, \text{init}, \text{o}, \text{next}, \text{out})$ for processing BDL streams in IoMT. The computation begins at an initial state init ∈ St, generates an initial output frame o ∈ $IoMT_2$, and at each step consumes input frames $x \in IoMT_1$, produces output $\text{out}(s, x) \in IoMT_2$, and transitions to the next state $\text{next}(s, x) \in \text{St}$.

2.4.4. Bisimulation and bisimilarity of BDL streams automata

The automata $\mathcal{G} =$ (St, init, o, next, out) takes input from $IoMT_1$ and produces output to $IoMT_2$ (Figure 2.14), where:

- A bisimulation $R \subseteq \text{St} \times \text{St}$ for $\mathcal{G}$ holds if for all states $s, t \in \text{St}$ and all inputs $x \in IoMT_1$, the relation $(s, t) \in R$ implies that the outputs are equal, i.e., $\text{out}(s, x) = \text{out}(t, x) \in R$ and the corresponding successor states also belong to R, i.e., $(\text{next}(s, x), \text{next}(t, x)) \in R$.
- The relation sRt denotes those states s and t are bisimilar, written $s \sim t$, if there exists a bisimulation R over the automaton $\mathcal{G}$ such that $(s, t) \in R$.
- For all states $s, t \in \text{St}$ and inputs $x \in IoMT$, if s is bisimilar to t ($s \sim t$), then their successors are also bisimilar, i.e., $\text{next}(s,\ x) \sim \text{next}(t,\ x)$.

CHAPTER 3: FORMALLY SPECIFYING AND COINDUCTIVE APPROACH TO VERIFYING SYNTHESIS OF STREAM CALCULUS-BASED COMPUTING BIG DATA IN LIVESTREAM

3.1. Introduction

Livestream from CS continuously generates real-time BDL streams, requiring technological and analytical support for efficient stream processing. Stream processing plays a critical role in specifying large-scale data systems. However, approaches to scheduling and formal verification of BDL streams remain largely unexplored.

3.2. Formal specification of BDL stream computation

Representing stream computation through stream operators enables efficient specification of pipeline aimed at reducing latency, increasing throughput, balancing load, and optimizing resource utilization. This specification is defined as a function from $\mathbb{A}^{\omega}$ to $\mathbb{A}^{\omega}$, supporting visual representation of RTL networks. Chapter 2 of the dissertation classifies the unary stream operators $\mathbb{A}^{\omega} \rightarrow \mathbb{A}^{\omega}$ and the binary stream operators $\mathbb{A}^{\omega} \times \mathbb{A}^{\omega} \rightarrow \mathbb{A}^{\omega}$, with the purpose of partitioning them into Algorithm 3.1.

3.3. RTL representation of algorithm 3.1

The algorithm 3.1 is used to explorer the RTL, illustrating the BDL synthesis process through the mapping of three input streams $\sigma, \tau, \rho \in \mathbb{A}^{\omega}$ to two output streams $\alpha, \gamma \in \mathbb{A}^{\omega}$.

3.3.1. Partitioning of BDL stream operators

The stream processing operators are partitioned into the stream diagram (Figure 3.2), corresponding to the code of Algorithm 3.1. Intermediate results are explicitly labeled on the diagram with the stream identifiers: a, b, c, d, e, f, g, h, i, j, k, l, m, and n.

3.3.2. Scheduling for BDL computation

The scheduling of BDL computation based on stream operators involves assigning individual stream operators to CSteps, sequentially indexed from 0, 1, 2,..., k, as illustrated in Figure 3.3.

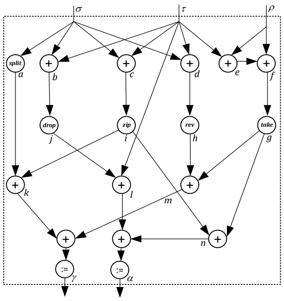


Figure 3.2. Stream diagram for partitioning stream operators

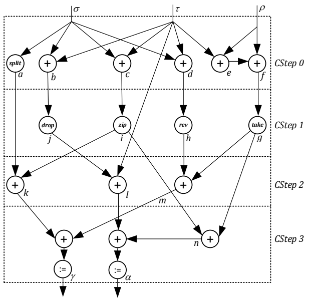


Figure 3.3. Diagram of stream operator partitioning for BDL computation into CSteps

3.3.3. Register allocation and binding

As part of register allocation and binding, the synthesis of stream-based computation involves executing two key tasks (Figure 3.4): Taks 1, allocation refers to determining the registers required to store intermediate results between two CSteps; Taks 2, binding refers to mapping intermediate results to registers at each CStep.

The process of register allocation and binding directly impacts hardware resource utilization and can significantly contribute to system optimization when performed efficiently. For the streams $\sigma, \tau, \rho, \alpha, \gamma \in \mathbb{A}^{\omega}$ (Figure 3.4), by applying Theorem 2.1, the following differential equations are obtained: $\alpha = X^3 \times (2\sigma + 4\tau + 2\rho)$ and $\gamma = X^3 \times (3\sigma + 3\tau + 2\rho)$.

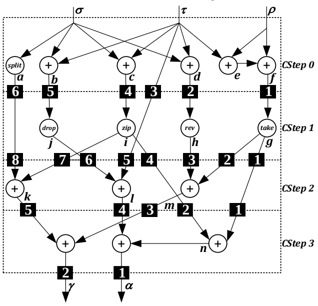


Figure 3.4. Register allocation and binding diagram of Algorithm 3.1

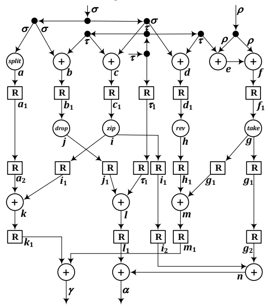


Figure 3.5. Stream computation diagram represented for RTL

Let the streams $\mu = (2\sigma + 4\tau + 2\rho)$ and $\delta = (3\sigma + 3\tau + 2\rho)$ be defined to produce the two output streams $\alpha = X^3 \times \mu$ and $\gamma = X^3 \times \delta$. The streams μ and δ are computed from σ, τ, and ρ, and are subsequently used to generate α and γ through the following differential equations:

Stream differential equation	Initial value
$\alpha^{(3)} = \mu, \gamma^{(3)} = \delta$	$\alpha(0) = \alpha(1) = \alpha(2), \gamma(0) = \gamma(1) = \gamma(2)$

The stream diagram in Figure 3.5 can also be interpreted as the RTL stream diagram shown in Figure 3.4. If the computation is scheduled across n CSteps, where $n \geq 0$, then the output streams α and γ are determined as follows: $\alpha = X^n \times \mu$ and $\gamma = X^n \times \delta$. The resulting output streams α and γ are given by:

Stream differential equation	Initial value
$\alpha^{(n)} = \mu$ $\gamma^{(n)} = \delta$	$\alpha^{(i)} = \mu(0), 0 \leq i \leq n-1$ $\gamma^{(i)} = \delta(0), 0 \leq i \leq n-1$

A bisimulation relation B over $\mathbb{A}^{\omega}$ is defined as a subset $B \subseteq \mathbb{A}^{\omega} \times \mathbb{A}^{\omega}$ such that $\forall \varphi, \beta \in \mathbb{A}^{\omega}$, the condition $\varphi\ B\ \beta$ implies that their initial values are equal, $\varphi(0) = \beta(0)$, and their derivatives satisfy $\varphi'\ B\ \beta'$.

Proposition 3.1 (Bisimulation): For all $n \geq 0$, any BDL output stream expressed as $\alpha = X^n \times \mu$ (or $\gamma = X^n \times \delta$) is in a bisimulation relation with the corresponding stream α (or γ).

Corollary 3.1 (Coinduction): For any output stream $X^n \times \mu$ in $\mathbb{A}^{\omega}$, if $X^i \times \mu \sim X^j \times \mu$ $(0 \leq i, j \leq n)$, then $X^i \times \mu = X^j \times \mu$. In other words, if two output streams $X^i \times \mu$ and $X^j \times \mu$ are bisimilar ($\sim$), then they are equal.

3.3.4. Functional unit allocation and binding

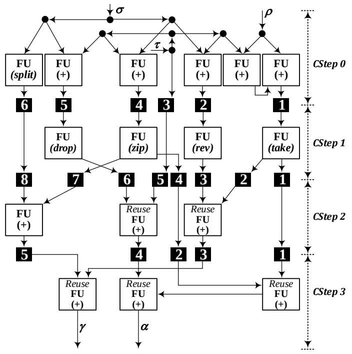


Figure 3.6. FU allocation and binding diagram

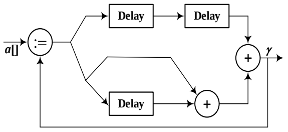


Figure 3.10. BDL computation diagram based on stream operators

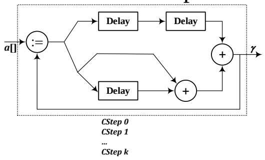


Figure 3.11. Single CStep scheduling diagram

In the allocation and binding of FUs, each FU is responsible for performing two main tasks as follows: Task 1, allocation refers to the assignment of stream processing operators to individual CSteps for execution; Task 2, binding refers to the implementation of stream processing operators on stream diagrams.

FUs are organized in a library to map functionalities to stream operators, supporting either single-purpose or multi-purpose usage through control signals. Figure 3.6 illustrates eight single-purpose FUs implementing the following operators: $drop, take, zip, split, rev, +, \bullet$ and $\blacksquare$.

3.3.5. Pipeline scheduling optimization problem for stream computation

Problem formulation: Given three streams $\sigma, \tau, \rho \in \mathbb{A}^{\omega}$ as inputs to the two stream algebra expressions of Algorithm 3.1:$\alpha := \big(zip(\sigma+\tau) + take\big(\rho + (\tau+\rho)\big) + (drop(\sigma+\tau)+\tau)\big)$ and $\gamma := \Big(split(\sigma) + zip(\sigma+\tau) + \Big(rev(\sigma+\tau) + take\big(\rho+(\tau+\rho)\big)\Big)\Big)$. This dissertation proposes the following scenario.

Scenario: Optimization of scheduling (reducing latency, increasing throughput, balancing load, and optimizing resources) based on a pipeline using linear programming, given two stream algebra expressions. In other words, given two stream algebra expressions and three input streams σ, τ, and ρ, the dissertation performs scheduling optimization as follows.

Simulation environment: colab.research.google.com and the Python programming language.

d) Developing a pipeline scheduling optimization model

The LP-based PSOM is specified through CSteps, represented by variables S_k at each node $k \in \mathbb{Z}^{+}$. The objective is to minimize latency

$C_{max} = \max(\{S_1, S_2, \dots, S_{13}\})$, subject to the constraint $C_{max} \geq S_k$, and non-overlapping constraints $|S_i - S_j| \geq 1$ or $|S_j - S_i| \geq 1$. Based on this formulation, the dissertation identifies 29 dependency constraints and 5 non-overlapping constraints for the LP-based PSOM [CT.9].

$$\begin{cases} C \geq S_k, \forall k = 1,2,3,\dots,13 \\ S_4 \geq S_1 \oplus 1 \\ S_5 \geq S_2 \oplus 1, S_5 \geq S_3 \oplus 2 \\ S_2 \geq S_3 \oplus 1 \\ S_6 \geq S_1 \oplus 1 \\ S_7 \geq S_6 \oplus 1 \\ S_9 \geq S_1 \oplus 1 \\ S_{10} \geq S_1 \oplus 1 \\ S_{11} \geq S_{10} \oplus 1, S_{11} \geq S_5 \oplus 1 \\ S_{12} \geq S_4 \oplus 1, S_{12} \geq S_5 \oplus 1, S_{12} \geq S_7 \oplus 1 \\ S_{13} \geq S_8 \oplus 1, S_{13} \geq S_9 \oplus 1, S_{13} \geq S_{11} \oplus 1 \\ |S_4 - S_9| \geq 1 \\ |S_9 - S_{10}| \geq 1 \\ |S_6 - S_{10}| \geq 1 \\ |S_6 - S_9| \geq 1 \\ |S_4 - S_{10}| \geq 1 \end{cases}$$

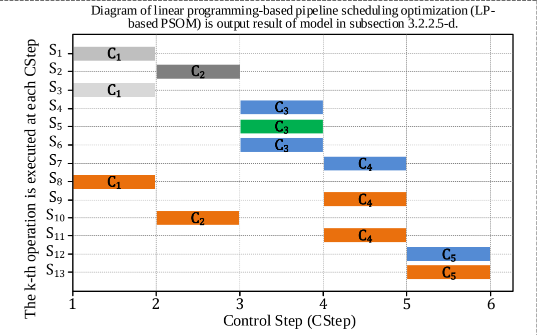


Figure 3.9. Pipeline scheduling optimization diagram [CT.9]

e) Pipeline scheduling optimization

f) Implementation of the pipeline scheduling optimization model

The dissertation implements the LP-based PSOM to obtain the optimized scheduling diagram shown in Figure 3.9 using algorithm 3.2: Each row represents a stream operator S_i, starting at its corresponding CStep; Five colors are used to indicate branches: gray (shared), light gray (intermediate), blue (α-branch), orange (γ-branch), and green (shared α, γ).

3.4. RTL representation of Algorithm 3.3

The BDL computation synthesis based on Algorithm 3.3 is illustrated through the stream array a, where $a[0] := drop(\sigma)$, $a[1] := zip(a[0])$, $a[2] := a[1] + a[0]$, and for $i \geq 3$, the recurrence $a[i] := a[i-1] + a[i-2] + a[i-3]$ is applied. These streams are then mapped to output γ.

3.4.1. Partitioning of BDL stream operators

The BDL computation diagram of Algorithm 3.3 (Figure 3.10) partitions three operators: $:=$, $+$, and unit delay. The unit delay stores the stream σ, emits the $(n-1)^{th}$ value, and stores the new n^{th} value at each step, for $n \geq 1$.

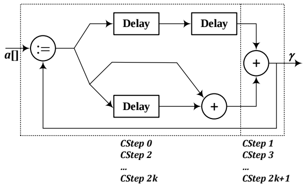


Figure 3.12. Scheduling diagram for two CSteps

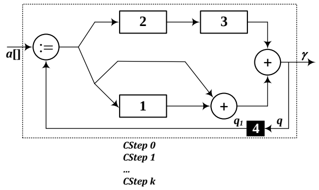


Figure 3.13. Register allocation and binding diagram for one CStep

3.4.2. Scheduling for BDL computation

Scheduling of BDL computations based on stream operators depends on execution speed (number of CStep) and hardware resources. Algorithm 3.3 can be scheduled using either a single CStep (Figure 3.11) or two CSteps (Figure 3.12).

3.4.3. Register allocation and binding

When scheduling a single CStep, an additional register labeled 4 is required for allocation and binding at the end of each CStep. The three-unit delays are replaced by three registers (labeled 1, 2, and 3) as illustrated in Figure 3.13 and described as follows:

Stream differential equation	Initial value
$a^{(3)}[] = a^{(2)}[] + a^{(1)}[] + a[]$	$a[0] = drop(\sigma), a^{(1)}[0] = zip(a[0])$ $a^{(2)}[0] = a^{(1)}[0] + a[0]$

The stream diagram in Figure 3.14 can also be regarded as an implementation of Algorithm 3.3, with the behavioral function defining the behavior of the output stream γ:

Stream differential equation	Initial value
$\gamma = a[] + X \times a[] + X^2 \times a[]$	$a[0] = drop(\sigma), a^{(1)}[0] = zip(a[0])$ $a^{(2)}[0] = a^{(1)}[0] + a[0]$

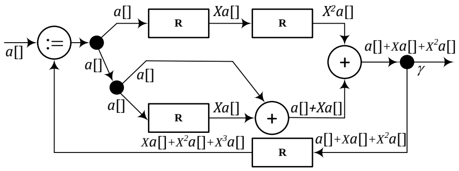


Figure 3.14. BDL computation diagram of stream operators for one CStep

Proposition 3.2 (Bisimulation): The two BDL output streams $a[] + X \times a[] + X^2 \times a[]$ and $X \times a[] + X^2 \times a[] + X^3 \times a[]$ are equivalent under bisimulation.

Corollary 3.2 (Coinductive): If $(a[] + X \times a[] + X^2 \times a[]) \sim (X \times a[] + X^2 \times a[] + X^3 \times a[])$, then $(a[] + X \times a[] + X^2 \times a[]) = (X \times a[] + X^2 \times a[] + X^3 \times a[])$.

When scheduling with two CSteps, the three-unit delays are replaced by three registers (1, 2, and 3). Two additional registers (4 and 5) are allocated at the end of each even-numbered CStep, with register 4 being reused at the end of each odd-numbered CStep (Figure 3.15). The behavioral differential equation of Algorithm 3.3 is described as follows:

Stream differential equation	Initial value
$a^{(3)}[] = X \times \left(a^{(2)}[] + a^{(1)}[] + a[]\right)$	$a[0] = drop(\sigma), a^{(1)}[0] = zip(a[0])$ $a^{(2)}[0] = a^{(1)}[0] + a[0]$

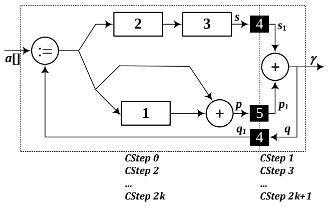


Figure 3.15. Diagram of register allocation and binding for 2 CSteps

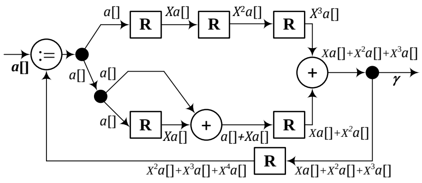


Figure 3.16. BDL computation diagram of stream operators describing 2 CSteps

The stream diagram in Figure 3.16 can also be regarded as an implementation of Algorithm 3.3 and defines the behavior of the BDL output stream γ as follows:

Stream differential equation	Initial value
$\gamma = X \times a[] + X^2 \times a[] + X^3 \times a[]$	$a[0] = \sigma, a^{(1)}[0] = a[0]$ $a^{(2)}[0] = a^{(1)}[0] + a[0]$

Substituting $a[] = 1 + X + 2X^2 + \left(4X^4 + 3X^5 + 2X^6 \times Rev_4(1 - X^2 - X^3 - X^4)\right)$ into the differential equation yields the behavior of the output stream γ in the form: $\gamma = 1 + X + 2X^2 + \left(4X^4 + 3X^5 + 2X^6 \times Rev_4(1 - X^2 - X^3 - X^4)\right) \times (X + X^2 + X^3)$.

Proposition 3.3 (Bisimulation): The two BDL output streams $X \times a[] + X^2 \times a[] + X^3 \times a[]$ and $X^2 \times a[] + X^3 \times a[] + X^4 \times a[]$ are equivalent under bisimulation.

Corollary 3.3 (Coinductive): If $(X \times a[] + X^2 \times a[] + X^3 \times a[]) \sim (X^2 \times a[] + X^3 \times a[] + X^4 \times a[])$, then $(X \times a[] + X^2 \times a[] + X^3 \times a[]) = (X^2 \times a[] + X^3 \times a[] + X^4 \times a[])$.

3.4.4. Functional unit allocation and binding

Under single CStep scheduling (Figure 3.17), two functional units perform two merge operators (+) and one assignment operator (:=), controlled by a $0/1$ signal to select the processing stream. If the signal is 0, the input stream of $a[]$ is output; if the signal is 1, $a[]$ is assigned the second input stream. Figure 3.17 illustrates the allocation and binding of FUs in Algorithm 3.3.

With two CStep scheduling (Figure 3.18), only two functional units are required: the even-numbered CStep performs merge (+) and assignment

$(:=)$, while the odd-numbered CStep reuses the merge unit.

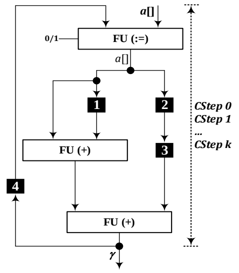


Figure 3.17. FU allocation and binding diagram for one CStep

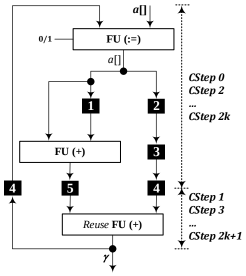
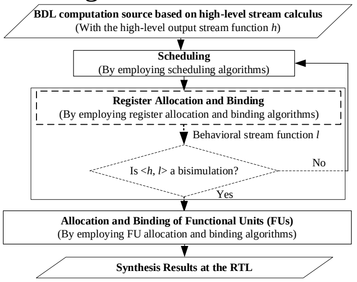


Figure 3.18. FU allocation and binding for 2 CSteps



BDL computation source based on high-level stream calculus
(With the high-level output stream function h)
Scheduling
(By employing scheduling algorithms)
Register Allocation and Binding
(By employing register allocation and binding algorithms)
Behavioral stream function l
Is <h, l> a bisimulation?
No
Yes
Allocation and Binding of Functional Units (FUs)
(By employing FU allocation and binding algorithms)
Synthesis Results at the RTL

Figure 3.19. Verification flowchart of the synthesis process for BDL stream computation

3.5. Coinductive approach for verifying synthesis of BDL computation

3.5.1. Synthesis verification

The verification technique for the BDL synthesis process based on stream calculus is performed post-execution, with the objective of being directly integrated into the synthesis steps. Unlike post-synthesis verification, the method illustrated in Figure 3.19 adopts a coinductive approach to ensure correctness.

3.5.2. Coinductive approach for verifying synthesis of BDL computation

h		l
σ (or τ)	$\sim$	$X^n \times \mu$ (or $X^n \times \delta$), with $n \geq 0$
$a[] + X \times a[] + X^2 \times a[]$		$X \times a[] + X^2 \times a[] + X^3 \times a[]$
$X \times a[] + X^2 \times a[] + X^3 \times a[]$		$X^2 \times a[] + X^3 \times a[] + X^4 \times a[]$

The verification technique utilizes a bisimulation relation between the output stream function h from the high-level BDL computation source and the behavioral stream function l after register allocation and binding (Figure 3.19). This relation, integrated into the synthesis process, is illustrated through Propositions 3.1–3.3.

It follows that the general bisimulation between h and l is given:

h with $m \in \mathbb{N}$	$\sim$	l with $n \in \mathbb{N}$
$X^m \times a[] + X^{m+1} \times a[] + X^{m+2} \times a[]$		$X^n \times a[] + X^{n+1} \times a[] + X^{n+2} \times a[]$

CHAPTER 4: ALGEBRAIC SEMANTICS OF REGISTER TRANSFER LEVEL IN SYNTHESIS OF STREAM CALCULUS-BASED COMPUTING BIG DATA IN LIVESTREAM

4.1. Introduction

Verifying correctness in BDL stream processing over livestream data is highly complex, particularly when validating the ASA_{RTL} and ASA_{SPEC}

automata based on algebraic semantics during the synthesis process.

4.2. The concept of Chu space

4.2.1. Chu space

4.2.2. Algebraic semantics as Chu space

In algebraic semantics, a Chu space $C = (X, r, A)$ is defined such that $r(e, s) = 1$ if event e occurs at state s, and $r(e, s) = 0$ otherwise. For each $e \in A$, a state $s_j \in 2^A$ is determined by $s_j = \{e \mid r(e, s_j) = 1\}$.

In logic, a matrix is regarded as a truth table. The logical specification f_{ASM} of the algebraic semantics model (ASM) is defined by the function: $f_{ASM} = \vee_{0 \le i \le n, 0 \le j \le l}\left(\wedge \{e_j \mid r(e_j, s_i) = 1\}\right) \wedge \{\wedge \bar{e}_j \mid r(\bar{e}_j, s_i) = 0\}$, where $n = |X| = |\{s_0, s_1, \dots, s_6\}|$, $l = |A| = |\{e_0, e_1, e_2\}|$, $e_j \in A$, and $\bar{e}_j$ denotes the complement of e_j.

4.3. Fundamental relations of state and event in Chu space

4.4. Algebraic semantics representation of the RTL for Algorithm 4.2

4.4.1. Description of the event set

Definition 4.1 (Computation): A finite sequential sequence of actions is called a computation process over the set of events A, and the set of all such computations is denoted by $\mathcal{C}(A)$.

4.4.2. Description of the state set

Let X be the set of states, where each $s \in X$ is defined through the transition relation $T(e, s_i)$, typically denoted as $s_i \xrightarrow{e} s_j$ with $e \in A$. Then, $s_j = \left\{e \mid e \in A\colon s_i \xrightarrow{e} s_j\right\}$. The triple (A, X, T) constitutes the algebraic semantics. For every $e_i \in A$ and $s_i \in X$, the ASM model (A, X, T) satisfies: $\mathcal{T} = \left\{e_1, e_2, \dots, e_n \mid s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \dots \xrightarrow{e_n} s_{n+1}\right\}$

4.4.3. Algorithm 4.2 applied to BDL stream processing

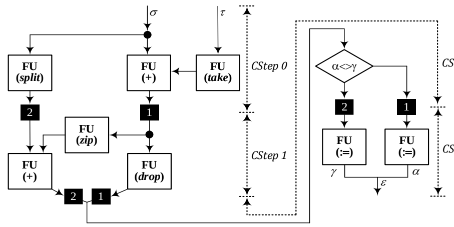
This chapter applies the udStreamsA algorithm [CT.2] to distribute events across BDL stream operators (merge, drop, take, zip, split, assign) within a single CStep, thereby forming Algorithm 4.2. The synthesis process maps the input $\sigma, \tau \in \mathbb{A}^\omega$ to the output $\alpha, \gamma, \epsilon \in \mathbb{A}^\omega$ in Algorithm 4.2.

4.4.4. The algebraic automata representation for Algorithm 4.2

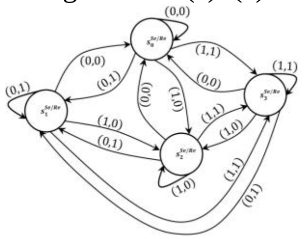
Definition 4.2 (Algebraic automata): An algebraic automaton is a triple $\mathcal{A} = (X, s_0, T)$, where X is a finite set of states, s_0 is the initial state, and T is a transition function from $X \times A$ to $X \cup \{\perp\}$.

The set of computations accepted by $\mathcal{A}$, denoted $\mathcal{C}(\mathcal{A})$ (Definition 4.1), is illustrated through four algebraic state automata corresponding to four

CSteps, each consisting of four states, as shown in Figures 4.10(a)–(c).



(a) RTL diagram of algorithm 4.2 with four CSteps, FUs, and operators



(c) Algebraic automata diagram for algorithm 4.2 with 4 CSteps

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

(b) Matrix representation of the algebraic automata for algorithm 4.2 with 4 CSteps

Figure 4.10. Representation of the algebraic automata for algorithm 4.2 with 4 CSteps

4.4.5. Algebraic automata product representation for Algorithm 4.2

The dissertation models two processes, Se and Re, using algebraic automata defined as $\mathcal{S}e = \left(X^{Se}, s_p^{Se}, T^{Se}\right)$ and $\mathcal{R}e = \left(X^{Re}, s_q^{Re}, T^{Re}\right)$, as shown in Figures 4.11(a) and 4.11(b). The product $\mathcal{S}e \times \mathcal{R}e$ models the connection between the two processes, ensuring data synchronization during register transfer. This product is defined over the event set A as $\mathcal{S}e \times \mathcal{R}e = \left(X, s_p, T\right)$, where $X = X^{Se} \times X^{Re}$, $s_p = \left(s_p^{Se} \times s_q^{Re}\right)$, with p, q optionally replaced by i, j, k, or l.

The transition relation T is defined as follows: for $s_i = \left(s_i^{Se} \times s_j^{Re}\right) \in X$ and $e \in A$, if $T^{Se}\left(s_i^{Se}, e\right) = s_{i+1}^{Se}$ and $T^{Re}\left(s_j^{Re}, e\right) = s_{j+1}^{Re}$, then $T\left(\left(s_i^{Se}, s_j^{Re}\right), e\right) = (s_{i+1}^{Se}, s_{j+1}^{Re})$; otherwise, $T\left(\left(s_i^{Se}, s_j^{Re}\right), e\right) = \perp$.

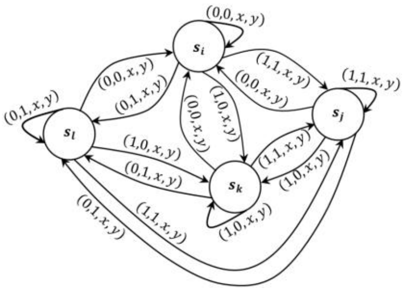
If the *output/input* of process Se corresponds to the *input/output* of process Re, the two processes are connected as illustrated in Figure 4.11: (a) the 16-state matrix of four registers and (b) the corresponding RTL diagram. Each state in the product $\mathcal{S}e \times \mathcal{R}e$ is a pair $(\mathcal{S}e, \mathcal{R}e)$ and is executed through the following computation sequence $\mathcal{T} = \{(1,1,0,1)(1,1,1,0)(1,1,1,1)\}$:

$$(s_3^{Se}, s_0^{Re}) \xrightarrow{(1,1,0,1)} (s_3^{Se}, s_1^{Re}) \xrightarrow{(1,1,1,0)} (s_3^{Se}, s_2^{Re}) \xrightarrow{(1,1,1,1)} (s_3^{Se}, s_3^{Re})$$

At state $\left(s_3^{Se}, s_0^{Re}\right)$ with event $(1,1,0,1)$, the product automata $\mathcal{S}e \times \mathcal{R}e$ is executed by concurrently running $\mathcal{S}e$ from s_3^{Se} and $\mathcal{R}e$ from s_0^{Re}, and the process continues similarly in subsequent iterations.

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

(a) This matrix represents the 16 states of four registers.



(b) Diagram representing the RTL of the 16 states

Figure 4.11. Connection of processes Se and Re and of algebraic automata product

4.5. RTL synthesis results produced by Algorithm 4.2

4.5.1. Steps of the verification algorithm

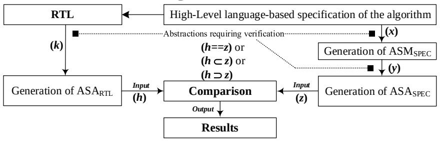


Figure 4.12. Diagram of algorithmic steps for verifying the ASM

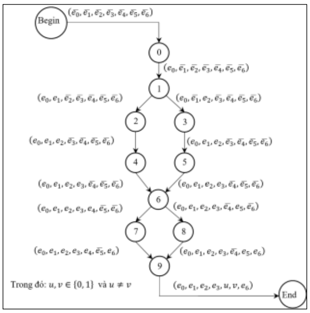
4.5.2. High-level language-based specification of the Algorithm

Specifying the algorithm using a high-level language is appropriate for descriptive purposes and facilitates program generation. The main task in this process is to update the various scheduling types. In *Figure 4.14*, the dissertation presents *Algorithm 4.2* with seven events e_0, e_1, e_2,…, e_7, each corresponding to a specific statement.

4.5.3. Creating algebraic semantics model (ASM)

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	$f = f_{e_0 \prec e_1} \wedge f_{e_0 \prec e_2} \wedge f_{e_1 \nabla e_2} \wedge f_{e_1 \prec e_3} \wedge f_{e_2 \prec e_3} \wedge f_{e_3 \prec e_4} \wedge f_{e_3 \prec e_5} \wedge f_{e_4 \# e_5} \wedge f_{\mathrm{den}(e_6,e_4,e_5)} = (e_0 + \overline{e_1}) \wedge (e_0 + \overline{e_2}) \wedge 1 \wedge (e_1 + \overline{e_3}) \wedge (e_2 + \overline{e_3}) \wedge (e_3 + \overline{e_4}) \wedge (e_3 + \overline{e_5}) \wedge (\overline{e_4} + \overline{e_5}) \wedge (\overline{e_6} + e_4 + e_5) = \overline{e_1}\overline{e_2}\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0 e_2\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0 e_1\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_6} + e_0 e_1 e_2\overline{e_4}\overline{e_5}\overline{e_6}\ e_0 e_1 e_2 e_3\overline{e_4}\overline{e_6} + e_0 e_1 e_2 e_3\overline{e_4} e_5 + e_0 e_1 e_2 e_3\overline{e_5}\overline{e_6}\ e_0 e_1 e_2 e_3 e_4\overline{e_5}$
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	

Figure 4.13. ASA_SPEC of algorithm 4.2



Begin
$(\overline{e_0}, \overline{e_1}, \overline{e_2}, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
0
$(e_0, \overline{e_1}, \overline{e_2}, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
1
$(e_0, e_1, \overline{e_2}, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
$(e_0, \overline{e_1}, e_2, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
2
3
$(e_0, e_1, e_2, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
$(e_0, e_1, e_2, \overline{e_3}, \overline{e_4}, \overline{e_5}, \overline{e_6})$
4
5
$(e_0, e_1, e_2, e_3, \overline{e_4}, \overline{e_5}, \overline{e_6})$
$(e_0, e_1, e_2, e_3, \overline{e_4}, \overline{e_5}, \overline{e_6})$
6
$(e_0, e_1, e_2, e_3, e_4, \overline{e_5}, \overline{e_6})$
$(e_0, e_1, e_2, e_3, \overline{e_4}, e_5, \overline{e_6})$
7
8
$(e_0, e_1, e_2, e_3, e_4, \overline{e_5}, e_6)$
$(e_0, e_1, e_2, e_3, \overline{e_4}, e_5, e_6)$
9
Trong đó: $u, v \in \{0, 1\}$ và $u \neq v$
$(e_0, e_1, e_2, e_3, u, v, e_6)$
End

Figure 4.14. ASA of algorithm 4.2

The local ordering specification and data dependency relations are derived from Algorithm 4.2 to construct ASA_SPEC, based on the event relations (Section 4.3) and the high-level programming language system specification (Section 4.5.2) to generate $\mathrm{ASM_{SPEC}} = \{e_0 \prec e_1, e_0 \prec e_2, e_1 \nabla e_2, e_1 \prec e_3, e_2 \prec e_3, e_3 \prec e_4, e_3 \prec e_5, e_4 \# e_5, \mathrm{den}(e_6, e_4, e_5)\}$ (Figure 4.13).

4.5.4. RTL synthesis results

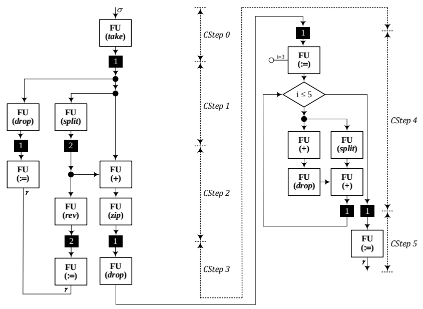
4.5.5. Algebraic RTL automata construction

ASA$_{\mathrm{RTL}}$ is generated from the RTL synthesis process of Algorithm 4.2, as illustrated in Figure 4.14, and presented in Figure 4.15.

4.6. Algebraic semantics for the RTL of Algorithm 4.4

4.6.1. Algorithm 4.4 for BDL streams processing

Algorithm 4.4 allocates events to the operators: merge, drop, take, zip, split, and assign within a single CStep. The BDL synthesis process is illustrated through the mapping of input streams $a[]$ and $\sigma \in \mathbb{A}^{\omega}$ to the output stream $\gamma \in \mathbb{A}^{\omega}$.



(a) RTL diagram of Algorithm 4.4 with 6 CSteps, FUs, and stream operators

ASA$_{\mathrm{RTL}}$	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
In which: $u, v \in \{0, 1\}$ and $u \neq v$							

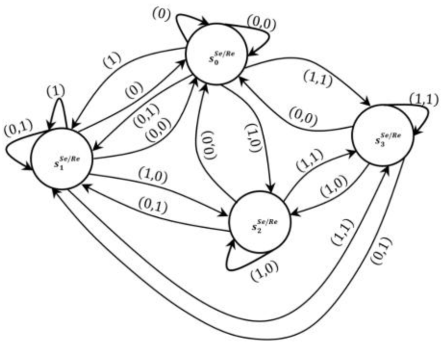
Figure 4.15. ASA$_{\mathrm{RTL}}$ of algorithm 4.2

	Register		Register	
CStep 0	1	CStep 1	1	2
0 ($s_0^{Se/Re}$)	0	0 ($s_0^{Re/Se}$)	0	0
1 ($s_1^{Se/Re}$)	1	1 ($s_1^{Re/Se}$)	0	1
		2 ($s_2^{Re/Se}$)	1	0
		3 ($s_3^{Re/Se}$)	1	1

CStep 2	2	1	CStep 3	1
0 ($s_0^{Re/Se}$)	0	0	0 ($s_0^{Re/Se}$)	0
1 ($s_1^{Re/Se}$)	0	1	1 ($s_1^{Re/Se}$)	1
2 ($s_2^{Re/Se}$)	1	0		
3 ($s_3^{Re/Se}$)	1	1		

CStep 4	1	CStep 5	1
0 ($s_0^{Re/Se}$)	0	0 ($s_0^{Re/Se}$)	0
1 ($s_1^{Re/Se}$)	1	1 ($s_1^{Re/Se}$)	1

(b) Matrix representation of the algebraic automata for Algorithm 4.4 with 6 CSteps



(c) Algebraic automata diagram of RTLs with 6 CSteps

Figure 4.16. Representation of algebraic automata for Algorithm 4.4 with 6 CSteps

4.6.2. The algebraic automata representation for Algorithm 4.4

Figure 4.16 illustrates the execution process of computations across 6 CSteps of Algorithm 4.4 through automata. Set of accepted computations, denoted $\mathcal{C}(\mathcal{A})$ (Definition 4.1), is detailed in Figures 4.16(a)-(c).

Figure 4.16(a) requires 15 single-purpose FUs to perform 10 operators ($split$, $take$, zip, rev, $drop$, $merge$ $+$, $copy$ $\bullet$, $assign$ $:=$, $\leq$, $register$ $\blacksquare$) across the CSteps as follows: CStep 0 performs the take operator and stores the result in register 1. CStep 1 executes copy, drop, and split, storing results in registers 1 and 2. CStep 2 executes assign, copy, merge, reverse, and zip, with values stored again in registers 1 and 2. CStep 3 performs assign and drop, storing the result in register 1. CStep 4 carries out $assign, \leq$, $merge$, $take$, and $drop$, with results written back to register 1. CStep 5 performs assign and outputs the result externally.

4.6.3. Product of algebraic automata for Algorithm 4.2

Figure 4.17(a) presents the 32-state matrix of five registers, while Figure 4.17(b) shows the corresponding RTL diagram. Each state in the product $Se \times Re$ is a pair composed of a state from Se and a state from Re. The product automata $Se \times Re$ executes the computation sequence $\mathcal{T} = \{(1,1,1,0,0)(1,1,1,0,1)(1,1,1,1,0)\}$ as follows:

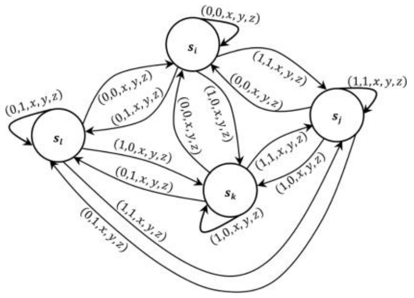
$$(s_3^{Se}, s_0^{Re}) \xrightarrow{(1,1,1,0,0)} (s_3^{Se}, s_1^{Re}) \xrightarrow{(1,1,1,0,1)} (s_3^{Se}, s_2^{Re}) \xrightarrow{(1,1,1,1,0)} (s_3^{Se}, s_3^{Re})$$

At state $\left(s_3^{Se}, s_0^{Re}\right)$ with event $(1,1,1,0,0)$, Se và Re execute concurrently from their respective states and continue iterating in manner.

Registers

$\mathcal{S}e \times \mathcal{R}e$	1	2	3	4	5
$s_i = \left(s_i^{Se} \times s_j^{Re}\right)$	0	0	x	y	z
$s_j = \left(s_j^{Se} \times s_k^{Re}\right)$	0	1	x	y	z
$s_k = (s_k^{Se} \times s_l^{Re})$	1	0	x	y	z
$s_l = (s_l^{Se} \times s_i^{Re})$	1	1	x	y	z
In which: $x, y, z \in \{0,1\}$ and $x \neq y$ và $y \neq z$; $i, j, k, l = \overline{[0,3]} \subset \mathbb{N}^* = \mathbb{N} \cup (0)$					

(a) The matrix represents the 32 states of 5 registers.



(b) Algebraic automata diagram of RTL representations with 32 states

Figure 4.17. Connection of processes Se and Re and the automata product

4.6.4. Algorithmic equivalence

4.6.5. Algebraic semantics of Algorithm 4.4

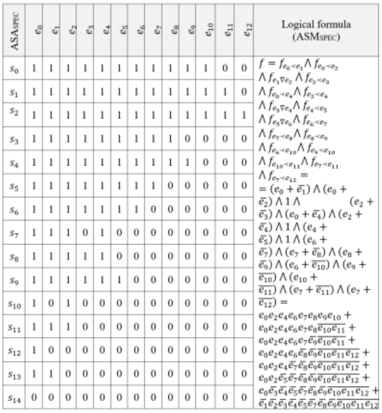
The algebraic automaton is modeled by the event set A and the transition relation T, which defines the set of state relations carrying algebraic semantics. The combination of these relations forms the algebraic automata.

4.7. Algorithm 4.4 for RTL synthesis results

4.7.1. Algorithm specification based on a high-level language

4.7.2. Model construction based on algebraic semantics (ASM)

This approach utilizes event relations to construct ASM_{SPEC} for Algorithm 4.4 in a high-level context, consisting of 13 events $e_0,\ldots, e_{12}$. The combination of these relations forms the resulting $\mathrm{ASM_{SPEC}} = \{e_0 \prec e_1, e_0 \prec e_2, e_1 \nabla e_2, e_2 \prec e_3, e_0 \prec e_4, e_2 \prec e_4, e_3 \nabla e_4, e_4 \prec e_5, e_5 \nabla e_6, e_6 \prec e_7, e_7 \prec e_8, e_8 \prec e_9, e_6 \prec e_{10}, e_9 \prec e_{10}, e_{10} \prec e_{11}, e_7 \prec e_{11}, e_7 \prec e_{12}\}$ (Figure 4.18.)



$\mathrm{ASA_{SPEC}}$	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	Logical formula ($\mathrm{ASM_{SPEC}}$)
s_0	1	1	1	1	1	1	1	1	1	1	1	0	0	$f = f_{e_0 \prec e_1} \wedge f_{e_0 \prec e_2} \wedge f_{e_1 \nabla e_2} \wedge f_{e_2 \prec e_3} \wedge f_{e_0 \prec e_4} \wedge f_{e_2 \prec e_4} \wedge f_{e_3 \nabla e_4} \wedge f_{e_4 \prec e_5} \wedge f_{e_5 \nabla e_6} \wedge f_{e_6 \prec e_7} \wedge f_{e_7 \prec e_8} \wedge f_{e_8 \prec e_9} \wedge f_{e_6 \prec e_{10}} \wedge f_{e_9 \prec e_{10}} \wedge f_{e_{10} \prec e_{11}} \wedge f_{e_7 \prec e_{11}} \wedge f_{e_7 \prec e_{12}} = = (e_0 + \overline{e_1}) \wedge (e_0 + \overline{e_2}) \wedge 1 \wedge (e_2 + \overline{e_3}) \wedge (e_0 + \overline{e_4}) \wedge (e_2 + \overline{e_4}) \wedge 1 \wedge (e_4 + \overline{e_5}) \wedge 1 \wedge (e_6 + \overline{e_7}) \wedge (e_7 + \overline{e_8}) \wedge (e_8 + \overline{e_9}) \wedge (e_6 + \overline{e_{10}}) \wedge (e_9 + \overline{e_{10}}) \wedge (e_{10} + \overline{e_{11}}) \wedge (e_7 + \overline{e_{11}}) \wedge (e_7 + \overline{e_{12}}) = e_0e_2e_4e_6e_7e_8e_9e_{10} + e_0e_2e_4e_6e_7e_8\overline{e_{10}}\overline{e_{11}} + e_0e_2e_4e_6e_7\overline{e_9}\overline{e_{10}}\overline{e_{11}} + e_0e_2e_4e_6\overline{e_8}\overline{e_9}\overline{e_{10}}\overline{e_{11}}\overline{e_{12}} + e_0e_2e_4\overline{e_7}\overline{e_8}\overline{e_9}\overline{e_{10}}\overline{e_{11}}\overline{e_{12}} + e_0e_2\overline{e_5}\overline{e_7}\overline{e_8}\overline{e_9}\overline{e_{10}}\overline{e_{11}}\overline{e_{12}} + e_0\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_7}\overline{e_8}\overline{e_9}\overline{e_{10}}\overline{e_{11}}\overline{e_{12}} + \overline{e_1}\overline{e_2}\overline{e_3}\overline{e_4}\overline{e_5}\overline{e_7}\overline{e_8}\overline{e_9}\overline{e_{10}}\overline{e_{11}}\overline{e_{12}}$
s_1	1	1	1	1	1	1	1	1	1	1	1	1	0	
s_2	1	1	1	1	1	1	1	1	1	1	1	1	1	
s_3	1	1	1	1	1	1	1	1	1	0	0	0	0	
s_4	1	1	1	1	1	1	1	1	1	1	0	0	0	
s_5	1	1	1	1	1	1	1	1	0	0	0	0	0	
s_6	1	1	1	1	1	1	1	0	0	0	0	0	0	
s_7	1	1	1	0	1	0	0	0	0	0	0	0	0	
s_8	1	1	1	1	1	0	0	0	0	0	0	0	0	
s_9	1	1	1	1	1	1	0	0	0	0	0	0	0	
s_{10}	1	0	1	0	0	0	0	0	0	0	0	0	0	
s_{11}	1	1	1	0	0	0	0	0	0	0	0	0	0	
s_{12}	1	0	0	0	0	0	0	0	0	0	0	0	0	
s_{13}	1	1	0	0	0	0	0	0	0	0	0	0	0	
s_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 4.18. ASA_{SPEC} of algorithm 4.4

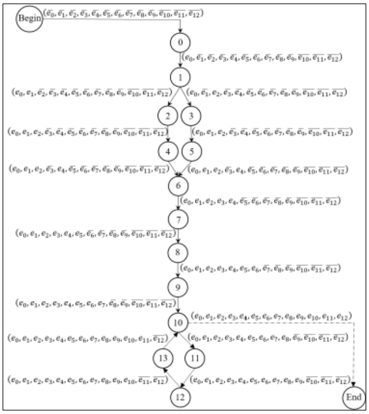

Begin
$(\overline{e_0},\overline{e_1},\overline{e_2},\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
0
$(e_0,\overline{e_1},\overline{e_2},\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
1
$(e_0,e_1,\overline{e_2},\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
$(e_0,\overline{e_1},e_2,\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
2
3
$(e_0,e_1,e_2,\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
$(e_0,e_1,e_2,\overline{e_3},\overline{e_4},\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
4
5
$(e_0,e_1,e_2,\overline{e_3},e_4,\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
$(e_0,e_1,e_2,\overline{e_3},e_4,\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
6
$(e_0,e_1,e_2,e_3,e_4,\overline{e_5},\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
7
$(e_0,e_1,e_2,e_3,e_4,e_5,\overline{e_6},\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
8
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,\overline{e_7},\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
9
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,\overline{e_8},\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
10
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},e_{11},e_{12})$
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},e_{11},\overline{e_{12}})$
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,\overline{e_9},\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
13
11
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,e_{10},\overline{e_{11}},\overline{e_{12}})$
$(e_0,e_1,e_2,e_3,e_4,e_5,e_6,e_7,e_8,e_9,\overline{e_{10}},\overline{e_{11}},\overline{e_{12}})$
12
End

Figure 4.19. ASA of algorithm 4.4

4.7.3. RTL synthesis results

4.7.4. Algebraic RTL automata construction

The ASA is derived from the RTL synthesis results of Algorithm 4.4, as illustrated in Figure 4.19 and represented as the ASA_{RTL} in Figure 4.20.

$\mathrm{ASA_{RTL}}$	e_1	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	$\mathrm{ASA_{RTL}}$	e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}
s_0	0	0	0	0	0	0	0	0	0	0	0	0	0	s_8	1	1	1	1	1	1	1	0	0	0	0	0	0
s_1	1	0	0	0	0	0	0	0	0	0	0	0	0	s_9	1	1	1	1	1	1	1	1	0	0	0	0	0
s_2	1	0	1	0	0	0	0	0	0	0	0	0	0	s_{10}	1	1	1	1	1	1	1	1	1	0	0	0	0
s_3	1	1	0	0	0	0	0	0	0	0	0	0	0	s_{11}	1	1	1	1	1	1	1	1	1	1	0	0	0
s_4	1	1	1	0	0	0	0	0	0	0	0	0	0	s_{12}	1	1	1	1	1	1	1	1	1	1	1	0	0
s_5	1	1	1	0	1	0	0	0	0	0	0	0	0	s_{13}	1	1	1	1	1	1	1	1	1	1	1	1	0
s_6	1	1	1	1	1	0	0	0	0	0	0	0	0	s_{14}	1	1	1	1	1	1	1	1	1	1	1	1	1
s_7	1	1	1	1	1	1	0	0	0	0	0	0	0														

Figure 4.20. $\mathrm{ASA_{RTL}}$ of algorithm 4.4

4.8. Comparison results

The correctness of the RTL results is verified by comparing the properties between ASA_{SPEC} and ASA_{RTL} (from Algorithm 4.2 or 4.4).

Definition 4.3 (Correctness of RTL synthesis results): An RTL synthesis result based on algebraic semantics is considered correct if it satisfies all the requirements specified in the algebraic specification.

Theorem 4.1: An RTL synthesis result based on algebraic semantics is correct if and only if $\mathcal{C}(\text{RTL}) = \mathcal{C}(\text{SPEC})$ (as defined in Definition 4.1).

The correctness verification between RTL and SPEC is conducted by comparing ASA_{RTL} with ASA_{SPEC}. Figures 4.13 and 4.15 (Algorithm 4.2), as well as Figures 4.18 and 4.20 (Algorithm 4.4), demonstrate that $\mathcal{C}(\text{SPEC})$ is equivalent to $\mathcal{C}(\text{RTL})$.

CONCLUSIONS

- **New scientific contributions**
  - **Development of an algebraic foundation for BDL streams in IoMT [CT.1][CT.2][CT.3][CT.5][CT.7][CT.8]**
    - Analyze stream theory (Stream algebra and stream coalgebra)
    - Develop ten stream processing operators
    - Combine stream processing operators into new stream expressions
    - Extend stream algebra and stream coalgebra used to BDL streams
    - Build monoid algebraic structure and monoid properties
    - Build a scenario to simulate and verify the evaluation of stream operators.
  - **Development of scheduling for BDL streams processing [CT.2][CT.9]**
    - Partition stream processing operators
    - Schedule for BDL stream computation
    - Allocate and bind registers
    - Allocate and bind functional units
    - Optimize scheduling using the pipeline to compute BDL streams.
  - **Development of the ASM [CT.4][CT.6]**
    - Develop the ASM model
    - Correlation between ASA_{RTL} and ASA_{SPEC}
- **Directions for future research**
    - Extend BDL stream processing operators
    - Build new algebraic structures for BDL streams
    - Advance stream theory for integration with big data systems
    - Research on big data processing for BDL streams within CS
    - Construct a generalized optimization model for pipeline scheduling
    - Develop operational algebraic semantics aspects of BDL in CS

LIST OF THE PUBLICATIONS RELATED TO THE DISSERTATION

[CT.1] **Dang Van Pham**, Vinh Cong Phan (2023), "Overview of The Stream Theory-Based Big Data in Livestream", Mobile Networks and Applications, ISSN: 1572-8153, Springer, 29(4), pp: 1239–1256, DOI: 10.1007/s11036-023-02180-0, **Scopus indexed(Q1), SCIE IF: 3.8(Q2)**.

[CT.2] **Dang Van Pham**, Vinh Cong Phan, Bao Khang Nguyen (2023), "Formally Specifying and Coinductive Approach to Verifying Synthesis of Stream Calculus-Based Computing Big Data in Livestream", Internet of Things, ISSN: 2542-6605, Elsevier, vol: 23, DOI: 10.1016/j.iot.2023.100878, **Scopus indexed(Q1), SCIE IF: 7.6(Q1)**.

[CT.3] **Dang Van Pham**, Vinh Cong Phan (2024), "Algebraic Aspects of Big Data in Livestream in Internet of Mobile Things", Mobile Networks and Applications, ISSN: 1572-8153, Springer, 29(1), pp: 243–261, DOI: 10.1007/s11036-024-02297-w, **Scopus indexed(Q1), SCIE IF: 3.8(Q2)**.

[CT.4] **Dang Van Pham**, Vinh Cong Phan, Bao Khang Nguyen (2024), "Algebraic Semantics of Register Transfer Level in Synthesis of Stream Calculus-Based Computing Big Data in Livestream", EAI ICTCC 2023 - 9th EAI International Conference on Nature of Computation and Communication. LNICST, ISSN: 1867-8211, Springer, Cham, vol: 586, pp: 19-35, ISBN: 978-3-031-59461-8, DOI: 10.1007/978-3-031-59462-5_2 **(Scopus, Q4)**.

[CT.5] **Dang Van Pham**, Vinh Cong Phan (2024), "Monoids Structure used for Computing Big Data in Livestream in Computing Systems", the 27th National Symposium of Selected ICT Problems – VNICT 2024, Nha Trang, Vietnam, 11-12/10/2024, pp: 363-369, ISBN: 978-604-67-3029-3 **(HĐCDGSNN)**.

[CT.6] **Dang Van Pham**, Vinh Cong Phan, Trong Toan Tran (2025), "Relationship between RTL Automata (ASA_{RTL}) and Algorithm Specification (ASA_{SPEC}) Based on Algebraic Semantics in Synthesis of Stream Calculus-Based Computing Big Data in Livestream", EAI ICCASA 2024 - 13th EAI International Conference on Context-Aware Systems and Applications. LNICST, ISSN: 1867-8211, Springer, Cham, vol: 664, ISBN: 978-3-032-12941-3 **(Scopus, Q4)**.

[CT.7] **Dang Van Pham**, Vinh Cong Phan, Toan Trong Tran, Bao Khang Nguyen (2024), "An Algebraic Structure for Computing Big Data in Livestream," 2024 13th International Conference on Control, Automation and Information Sciences (ICCAIS), ISSN: 2475-7896, ISBN: 979-8-3315-4204-7, IEEE, DOI: 10.1109/ICCAIS63750.2024.10814295 **(Scopus)**.

[CT.8] **Dang Van Pham**, Vinh Cong Phan, Trong Toan Tran (2025), "Analyzing and Comparing the Stream Theory: A Review", EAI ICTCC 2024 - 10th EAI International Conference on Nature of Computation and Communication. LNICST, ISSN: 1867-8211, Springer, Cham, vol: 668, ISBN: 978-3-032-12846-1, DOI: 10.1007/978-3-032-12846-1_6 **(Scopus, Q4)**.

[CT.9] **Dang Van Pham**, Vinh Cong Phan (2025), "Developing Linear Programming-Based Pipeline Scheduling Optimization Model Used to Big Data in Livestream", Proceeding of the 18th National Conference on Fundamental and Applied IT Research – FAIR'18, Ha Noi, 21-22/08/2025 **(HĐCDGSNN)**.