Denotational and Algebraic Semantics for the CaIT calculus

Ningning Chen and Huibiao Zhu

Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China
hbzhu@sei.ecnu.edu.cn

Abstract. The Internet of Things (IoT) has been wildly used in various fields of our lives, such as health care, smart environment, transportation, etc. However, the existing research on IoT mainly concentrates on its practical applications, and there is still a lack of work on modelling and reasoning about IoT systems from the perspective of formal methods. Therefore, the Calculus of the Internet of Things (CaIT) has been proposed to model the interactions among components and verify the network deployment to ensure the quality and reliability of IoT systems. Unfortunately, the CaIT calculus can only support point-to-point communication, while broadcast communication is more common in IoT systems. Therefore, this paper updates the CaIT calculus by replacing its communication primitive with the broadcast. Based on the Unifying Theories of Programming (UTP), we further explore its denotational semantics and algebraic semantics, with a special focus on broadcast communication, actions with the timeout (e.g. input actions and migration actions), and channel restriction. To facilitate the algebraic exploration of parallel expansion laws, we further extend the CaIT calculus with a new concept called guarded choice, which allows us to transform each program into the guarded choice form.

Keywords: the CaIT calculus · IoT · UTP · Denotational semantics · Algebraic Semantics.

1 Introduction

Equipped with “Things" capable of sensing, processing, and communicating, smart devices in IoT systems can collect information from the Internet or the physical environment anytime, anywhere, and provide advanced services to end users. [1, 2]. “Things" in IoT systems can be RFID tags that identify objects, sensors that capture changes in the environment, and actuators that provide information to the environment. With the increasing demand for applications and technologies of IoT, a variety of promising technologies (e.g. 5G, high speed, low latency networks, etc.) have been applied to the IoT paradigm. As a result, IoT is rapidly evolving towards IoT 2.0 to further meet the demands of a series of advanced technologies, such as machine learning, edge computing, and Industry

4.0 [3, 4]. However, most of the existing studies on IoT are mainly focused on its practical applications, and little work has been done to model interactions between components in IoT systems and to check IoT network deployments [5].

As a mathematics-based technique, the formal methods have been widely used to specify, verify, and analyze software and hardware systems, thereby ensuring the quality and reliability of systems. To the best of our knowledge, Lanese et al. have presented the first calculus for IoT named IoT-calculus, which aims to formalize a few fundamental characteristics of IoT systems: the partial topology of communications and the interaction between sensors, actuators, and computing processes [5]. Subsequently, Bodei et al. presented a secure untimed process calculus called IoT-LYSA, which employs static analysis technologies to track the sources and paths of IoT data and detect how they influence smart objects [6]. However, the above two calculi do not consider the effect of time on process actions. Thus, Lanotte et al. proposed the CaIT calculus with reduction semantics and a labeled transition system [7]. In contrast, the CaIT calculus can model discrete time behaviours with consistency and fairness properties by equipping process actions with timeouts. However, the CaIT calculus can only support point-to-point communication, rather than the more general broadcast communication. Thus, this paper refines the original CaIT calculus by replacing its point-to-point communication with broadcast communication [8].

The Unifying Theories of Programming (UTP) has been proposed by He and Hoare in 1998 [9], which has three methods to represent the semantics of a programing language: operational semantics [10], denotational semantics [11], and algebraic semantics [12]. The operational semantics of a programming language provides a complete set of individual steps to simulate the execution, which shows *how a program works* [10]. The denotational semantics gives meaning to a programming language from a purely mathematical point of view, which explains *what a program does* [11]. The algebraic semantics of a programming language includes a series of algebraic laws, which is well suited to the symbolic calculation of parameters and structures of an optimal design. The operational semantics of the CaIT calculus has been explored in [7]. This paper investigates the denotational and algebraic semantics of the CaIT calculus.

The main contributions of this paper are as follows:

- We enrich the CaIT calculus by replacing its point-to-point communication with broadcast communication.
- We explore the denotational semantics of the CaIT calculus, involving the basic commands, the guarded choice, the parallel composition, and channel restriction.
- We investigate the algebraic semantics of the CaIT calculus, especially channel restriction. By establishing the algebraic laws for the parallel composition of guarded choice components, we can describe any program as the guarded choice form.

The rest of this paper is organized as follows. In Section 2, we introduce the syntax of the CaIT calculus and the concept of guarded choice that is used to investigate the parallel expansion laws. In Section 3, we explore the denotational semantics of the CaIT calculus. In Section 4, we propose a set of algebraic laws for the CaIT calculus. The conclusion and future work are in Section 5.

2 The CaIT Calculus

In this section, we introduce the syntax of the CaIT calculus. To establish algebraic parallel expansion laws, we then give three kinds of guarded choices.

2.1 Syntax

Compared with the original CaIT calculus proposed in [7], we extend it by introducing more general broadcast communication and adding the migration operation to conveniently model node mobility. As shown in Table 1, the syntax of this calculus has a two-level structure: the first level illustrates the networks, and the second one models the processes. For greater clarity, the extensions have been bolded in Table 1.

Firstly, we define some functions for our exploration. Function $\mathbf{Rng}(c)$ is used to compute the transmission range of channel c. We can calculate the distance between locations l and l_1 with the function $\mathbf{Dis}(l, l_1)$. Two nodes at l and l_1 can communicate via channel c only if the distance between them is within the communication range of channel c, i.e., $\mathbf{Dis}(l, l_1) \leq \mathbf{Rng}(c)$. Otherwise, they cannot communicate via c.

Table 1. The syntax of CaIT calculus

Network			
	N, M	$::= 0$	Empty Network
		$\mid\ M \| N$	Parallel Composition
		$\mid\ (vc')M$	Channel Restriction
		$\mid\ {}_n[\Gamma \bowtie P]_l^u$	Node
Process			
	P, Q	$::= nil$	Termination
		$\mid\ P \| Q$	Parallel Composition
		$\mid\ [b]P\boldsymbol{,}Q$	Conditional Choice
		$\mid\ !\langle v\rangle^c; P$	**Output**
		$\mid\ \rho; P$	Intra-node Action
		$\mid\ \lfloor \pi; P \rfloor Q$	Action with Timeout
		$\mid\ while\ b\ do\ P$	Iteration
	ρ	$::= \sigma$	Delay
		$\mid\ s?y$	Reading Sensor
		$\mid\ a!v$	Writing Actuator
	π	$::= ?(x)^c$	**Input**
		$\mid\ move\_k$	**Migration**

Network Level:

(1) 0 stands for an empty network.

(2) $M\|N$ is the parallel composition.

(3) $(vc')M$ indicates that channel c' is private to the network M.

(4) ${}_{n}[\mathit{\Gamma} \bowtie P]_{l}^{u}$ denotes a network node, where n is the node ID, P is the process modelling the logic of this node, l records its current location, and $\mathit{\Gamma}$ is its physical interface. The physical interface $\mathit{\Gamma}$ can map the names of sensors and actuators to values. To ensure the security of nodes, each $\mathit{\Gamma}$ is private for a node. Given node ${}_{n}[\mathit{\Gamma} \bowtie P]_{l}^{u}$, only the corresponding *controller process* P can read the values of sensors in $\mathit{\Gamma}$. Analogously, the values of actuators in $\mathit{\Gamma}$ can be modified only by P. u is given to differentiate between stationary nodes (if $u = s$) and mobile nodes (if $u = m$).
Process Level:

(1) *nil* indicates this process terminates.

(2) $P \| Q$ stands for the parallel composition of processes P and Q.

(3) $[b]P$**,**Q illustrates the conditional choice, where b has the form as $[w = w']$. If w is equal to w', process P works; otherwise, process Q.

(4) $!\langle v\rangle^{c}; P$ denotes that after sending the value v via c immediately, P runs.

(5) $\rho; P$ models intra-node actions, where $\rho \in \{\sigma, s?(y), a!v\}$. $\sigma; P$ means that process P starts executing after waiting for one time unit. A node can obtain a value from sensor s and assign it to variable y, denoted by $s?y; P$. Due to the execution of $a!v; P$, the value of the actuator a should be modified to the newly written value v, further leading to the update of $\mathit{\Gamma}$, i.e., $\mathit{\Gamma}' = \mathit{\Gamma}[v/\mathit{\Gamma}(a)]$.

(6) $\lfloor \pi; P \rfloor Q$ stands for some actions which execute with the timeout, where $\pi \in \{?(x)^{c},\ move\_k\}$. For $\lfloor ?(x)^{c}; P \rfloor Q$, if a value can be received via channel c within one time unit, it continues as process P after that. Otherwise, Q runs after one time unit. $\lfloor move\_k; P \rfloor Q$ illustrates the node mobility. After delaying one time unit, if the node moves to the destination k successfully, this process terminates, and then P obtains the control at its new location k. Otherwise, Q starts at its original location after one time unit. $\pi; P$ indicates that process P begins only when action π has finished.

(7) *while b do P* is the iteration construct, where b is in the form $[w = w']$.

2.2 Guided Choice

To support our investigation of the algebraic parallel expansion laws, we extend the CaIT calculus with the following three types of guarded choices.
• **Instantaneous Guarded Choice:**

$$[\!]_{i \in I}\{g_i \rightarrow N_i\},$$

where, $g_i \in \{!\langle v\rangle^{c}@l,\ ?(x)^{c}@l,\ c.[v/x]@(l, l_1),\ s?y@l,\ a!v@l,\ move\_k@l,\ b_i\&\tau@l\}$.

The guard g_i is instantaneous, meaning that it executes without any time delay. $!\langle v\rangle^{c}@l$ indicates that a node at location l sends out the value v via channel c. And $c.[v/x]@(l, l_1)$ denotes that a node at l successfully sends v to another node at l_1 via c, where $[v/x]$ replaces the variable x with the received value v. $b_i\&\tau@l$ represents that a silent action occurs at location l if the Boolean expression b_i is true, where the silent action does nothing and terminates immediately. For $(vc')M$, it should guarantee that communication actions (output actions or synchronous communication actions) that take place on c' are not visible outside of M. Thus, we conceal them by replacing them with $true\&\tau@l$.

• **Delay Guarded Choice:**

$$\#t \rightarrow N$$

The delay guarded choice denotes that the subsequent network N starts after delaying t time units, where t ranges from 0 to 1.

• **Hybrid Guarded Choice:**

$$\begin{aligned} &\|_{i\in I}\{g_i \rightarrow N_i\} \\ &\oplus \exists t' \in (0\dots 1) \bullet \#t' \rightarrow N' \\ &\oplus \#1 \rightarrow N'' \end{aligned}$$

The third type is the hybrid guarded choice, which has three branches combined by the notation $\oplus$, where $\oplus$ denotes the disjointness of timed behaviours.

3 Denotational Semantics

Here, we present the denotational semantics of the CaIT calculus based on UTP. We first introduce the semantics model and give healthiness conditions that each program must satisfy. We then discuss the denotational semantics of basic commands, guarded choices, the parallel composition, and channel restriction.

3.1 Semantic Model

Compared with the previous UTP theories [9], the CaIT calculus captures some vital characteristics of IoT systems, such as time constraints, communications, the node mobility, etc. To better describe its behaviours, we define the following observation tuple for this calculus.

$$\big(time, time', st, st', tr, tr'\big)$$

(1) $time$ and $time'$ represent the start and end time points of an observation time interval, respectively. Δ is the time interval, where $\Delta = time' - time$.

(2) st and st' stand for the initial and final execution state of the program, respectively. For a program, it may have three types of execution states.

- ter : If a process terminates successfully, it runs into a ter state. "$st = ter$" means that the previous process terminates successfully, and the current process starts to execute. "$st' = ter$" denotes that the current process terminates successfully, and then the following process starts to work.
- $wait$: A process may be waiting to communicate with another process via a particular channel, as described by the $wait$ state. "$st = wait$" indicates that the previous process is in a $wait$ state. Thus, the current process cannot be executed. Similarly, "$st' = wait$" denotes that the current process reaches a $wait$ state. Therefore, the following process cannot be performed.
- div : When a process enters a div state and never terminates, its future behaviour becomes uncontrollable. $st = div$ means that the previous process diverges. Thus, the current process can never obtain control. $st' = div$ represents that the current process is uncontrolled. Thus, the next process never works.

(3) We introduce the notion of traces to record the communication behaviour of a process. We give a pair of variables tr and tr'. tr presents the initial trace of a process inherited from its predecessor. tr' is the final trace containing the contribution of the current process. $tr' - tr$ is the trace generated by the current process. $tr_1^\frown tr_2$ connects traces tr_1 and tr_2. A trace is a sequence of snapshots. The notation $\mathbf{head}(tr)$ stands for the first snapshot of trace tr. $\mathbf{tail}(tr)$ records the remainder after removing the first snapshot from trace tr. A snapshot is a quadruple formed by (t, l, o, f).

- t represents the time when the communication action occurs.
- l is the location of the node at which a communication action occurs.
- o means that a message is transmitted via a specific channel. The form of o is $c.v$, where c is the communication channel and v is the transmitted message. Further, we define the concepts $\mathbf{Chan}(o)$ and $\mathbf{Mess}(o)$ to obtain the communication channel and the transmitted message from o respectively, i.e., if $o = c.v$, then $\mathbf{Chan}(o) = c$ and $\mathbf{Mess}(o) = v$.
- To conveniently merge traces, we use the flag f to separate communication actions into two types. If $f = 1$, this action is an output action (e.g., $!\langle v\rangle^c$). Otherwise, $f = 2$ means that this is an input action (e.g., $?(x)^c$).

We introduce notations $\pi_i (i = 1, 2, 3, 4)$ to obtain the ith element of a snapshot, such as $\pi_1((t, l, o, f)) =_{df} t$.

3.2 Healthiness Conditions

Here, we give some healthiness conditions which every process must satisfy. **H1** indicates that traces cannot be shortened and time cannot go back. **H2** proposes the following two requirements. If the previous process runs into a *wait* state, the current process cannot be activated. Thus, all values keep unchanged. Otherwise, the current process executes. As mentioned before, $st = div$ means that the behaviour of the previous process becomes unpredictable. Thus, the current process has never started and the initial values are unobservable. Therefore, **H3** should be satisfied.

(H1) $P = P \wedge \mathbf{Inv}(tr, time)$,

where, $\mathbf{Inv}(tr, time) =_{df} tr \preceq tr' \wedge time \preceq time'$, and $tr \preceq tr'$ denotes that sequence tr is a prefix of sequence tr'.

(H2) $P = \prod \lhd st = wait \rhd P$,

where, $P \lhd b \rhd Q =_{df} (b \wedge P) \vee (\neg b \wedge Q)$, $\prod =_{df} (st' = st) \wedge (time' = time) \wedge (tr' = tr)$.

(H3) $P = \mathbf{Inv}(tr, time) \lhd st = div \rhd P$

To present the denotational semantics for the CaIT calculus, we give the following definition for $\mathbf{H}(X)$, which not only caters to the above three healthiness conditions but is idempotent and monotonic.

$$\mathbf{H}(X) =_{df} \mathbf{Inv}(tr, time) \lhd st = div \rhd \big(\prod \lhd st = wait \rhd \big(X \wedge \mathbf{Inv}(tr, time) \big) \big)$$

3.3 Denotational Semantics of Basic Commands

In this subsection, we explore the denotational semantics of the basic commands. By defining $\mathbf{beh}(N)$, we depict the behaviour of a network N.
(1) Termination: *nil* represents a termination process, so that the execution state, termination time, and trace remain unchanged.

$$\mathbf{beh}({}_n[\Gamma \bowtie nil]_l^u) =_{df} \mathbf{H}(st' = st \wedge time' = time \wedge tr' = tr)$$

(2) Sequential Composition: The behaviour of sequential composition is denoted by $N; M$, i.e., running N and M in sequence.

Definition 1 $N; M =_{df} \exists t, s, r \bullet N[t/time', s/st', r/tr'] \wedge M[t/time, s/st, r/tr]$

The denotational semantics of the sequential composition is given below:

$$\mathbf{beh}(N; M) =_{df} \mathbf{beh}(N); \mathbf{beh}(M)$$

(3) Delay: Given a delaying process ${}_n[\Gamma \bowtie \sigma; P]_l^u$, it denotes that process P starts to execute after one time unit.

$$\mathbf{beh}({}_n[\Gamma \bowtie \sigma; P]_l^u) =_{df} \mathbf{beh}(\#1); \mathbf{beh}({}_n[\Gamma \bowtie P]_l^u)$$

$\mathbf{beh}(\#t)$ describes the behaviour of waiting for t time units. $\Delta < t$ means that the process has not waited for t time units yet, thus it still needs to wait. In this case, its state and trace keep unchanged. The process stops waiting when $\Delta = t$, formed by $st' = ter$.

$$\mathbf{beh}(\#t) =_{df} \mathbf{H}\left((st' = wait \wedge \Delta < t \wedge tr' = tr) \vee (st' = ter \wedge \Delta = t \wedge tr' = tr)\right)$$

(4) Output: In ${}_n[\Gamma \bowtie !\langle v\rangle^c; P]_l^u$, the output command happens at the activeness time, modeled by $\mathbf{beh}(!\langle v\rangle^c@l)$. Then process P obtains the control.

$$\mathbf{beh}({}_n[\Gamma \bowtie !\langle v\rangle^c; P]_l^u) =_{df} \left(\mathbf{beh}(!\langle v\rangle^c@l); \mathbf{beh}({}_n[\Gamma \bowtie P]_l^u\right)$$

We only need to add the outputting snapshot $(time', l, c.v, 1)$ to the end of the trace tr. The behaviour of $\mathbf{beh}(!\langle v\rangle^c@l)$ is as below.

$$\mathbf{beh}(!\langle v\rangle^c@l) =_{df} \mathbf{H}\left(st' = ter \wedge \Delta = 0 \wedge tr' = tr^\frown\langle(time', l, c.v, 1)\rangle\right)$$

(5) Input: The semantics definition of an input command has three branches. The first branch (i.e., formula **(5.1)**) indicates that the input action happens at the triggering time. $\mathbf{beh}(?(m)^c@l)$ describes the behaviour of the input command, which means that node at location l receives a value m via channel c. The following behaviour is $\mathbf{beh}({}_n[\Gamma \bowtie P[m/x]]_l^u)$. In the second branch (i.e., formula **(5.2)**), the input command occurs after t' time units. The last branch (i.e., formula **(5.3)**) shows that this input command does not happen within one time unit. After delaying one time unit, the following behaviour is $\mathbf{beh}({}_n[\Gamma \bowtie Q]_l^u)$. Here, $Type(c)$ denotes the type of messages transmitted in channel c.

$$\mathbf{beh}({}_n[\Gamma \bowtie \lfloor ?(x)^c; P\rfloor Q]_l^u) =_{df}$$
$$\left(\begin{array}{ll} \exists m \in Type(c) \bullet \mathbf{beh}(?(m)^c@l); \mathbf{beh}({}_n[\Gamma \bowtie P[m/x]]_l^u) \vee & \mathbf{(5.1)} \\ \exists t' \in (0 \ldots 1) \bullet \mathbf{beh}(\#t'); \exists m \in Type(c) \bullet \mathbf{beh}(?(m)^c@l); & \\ \qquad \mathbf{beh}({}_n[\Gamma \bowtie P[m/x]]_l^u) \vee & \mathbf{(5.2)} \\ \mathbf{beh}(\#1); \mathbf{beh}({}_n[\Gamma \bowtie Q]_l^u) & \mathbf{(5.3)} \end{array}\right)$$

Now, we describe the behaviour of $\mathbf{beh}(?(m)^c@l)$. The input action is instantaneous, thus, $\varDelta = 0$. To record the input action, the snapshot $(time', l, c.m, 2)$ is added to the end of the trace.

$$\mathbf{beh}(?(m)^c@l) =_{df} \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr' = tr^\frown\langle(time', l, c.m, 2)\rangle \right)$$

(6) Migration: We explore the denotational semantic of the moving command $\lfloor move\_k; P\rfloor Q$. Here, δ is the maximum distance that the node n can move within one time unit, which is set for the node in advance. For one time unit, the migration action is in a waiting state and its trace is unchanged. After one time unit, if it moves to the destination k successfully (i.e., $\mathbf{Dis}(k, l) \leq \delta$), a silent action $\mathbf{beh}(\tau@l)$ and a moving command $\mathbf{beh}(move\_k@l)$ happen sequentially, and the following behaviour is $\mathbf{beh}({}_n[\varGamma \bowtie P]_k^m)$. Otherwise, a silent action $\mathbf{beh}(\tau@l)$ occurs, and then it continues as $\mathbf{beh}({}_n[\varGamma \bowtie Q]_l^m)$.

$$\begin{aligned}&\mathbf{beh}({}_n[\varGamma \bowtie \lfloor move\_k; P\rfloor Q]_l^m) =_{df} \\ &\qquad\left(\mathbf{beh}(\#1); \begin{pmatrix} \mathbf{beh}(\tau@l); \mathbf{beh}(move\_k@l); \mathbf{beh}({}_n[\varGamma \bowtie P]_k^m) \\ \lhd Dis(l,k) \leq \delta \rhd \\ \left(\mathbf{beh}(\tau@l); \mathbf{beh}({}_n[\varGamma \bowtie Q]_l^m)\right) \end{pmatrix}\right)\\ &\text{where, } \mathbf{beh}(move\_k@l) =_{df} \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr' = tr \right), \\ &\qquad \mathbf{beh}(\tau@l) =_{df} \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr' = tr \right).\end{aligned}$$

As mentioned before, traces are used to record the behaviour of communications. Thus, actions (i.e., the migration action) that do not involve communication need not be stored in traces.

(7) Reading Sensor: Now, we analyze the behaviour of reading a value from a sensor s, that is $\mathbf{beh}(s?\varGamma(s)@l))$. After that, the following behaviour is described by $\mathbf{beh}({}_n[\varGamma \bowtie P[\varGamma(s)/y]]_l^u)$.

$$\begin{aligned}&\mathbf{beh}({}_n[\varGamma \bowtie s?y; P]_l^u) =_{df} \mathbf{beh}(s?\varGamma(s)@l); \mathbf{beh}({}_n[\varGamma \bowtie P[\varGamma(s)/y]]_l^u)),\\ &\text{where, } \mathbf{beh}(s?\varGamma(s)@l)) =_{df} \ \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr' = tr \right).\end{aligned}$$

(8) Writing Actuator: If the original value of actuator a (i.e., $\varGamma(a)$) is not the newly written value v, $\mathbf{beh}(\tau@l)$ and $\mathbf{beh}(a!v@l)$ perform orderly, and then process P is executed under the new interface $\varGamma'$, where $\varGamma' = \varGamma[v/\varGamma(a)]$. Otherwise, P runs under $\varGamma$ after $\mathbf{beh}(\tau@l)$.

$$\begin{aligned}&\mathbf{beh}({}_n[\varGamma \bowtie a!v; P]_l^u) =_{df} \begin{pmatrix} \left(\mathbf{beh}(\tau@l); \mathbf{beh}(a!v@l); \mathbf{beh}({}_n[\varGamma' \bowtie P]_l^u)\right) \\ \lhd \varGamma(a) \neq v \rhd \left(\mathbf{beh}(\tau@l); \mathbf{beh}({}_n[\varGamma \bowtie P]_l^u)\right) \end{pmatrix},\\ &\text{where, } \mathbf{beh}(a!v@l) =_{df} \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr' = tr \right).\end{aligned}$$

(9) Conditional: The behaviour of the condition choice is as follows.

$$\mathbf{beh}({}_n[\varGamma \bowtie [b]P,Q]_l^u) =_{df} \left(\mathbf{beh}({}_n[\varGamma \bowtie \tau; P]_l^u) \lhd b \rhd \mathbf{beh}({}_n[\varGamma \bowtie \tau; Q]_l^u)\right)$$

(10) Iteration: Referring to the traditional programming language, we give the denotational semantics of the iteration construct. F is a monotonic function mapping processes to processes. Based on the previously proposed healthy formulas, we give its the weakest fixed point, i.e., $\mu_{HF}F(X)$.

$$\mathbf{beh}({}_n[\varGamma \bowtie while\ b\ do\ P]_l^u) =_{df} \mathbf{beh}({}_n[\varGamma \bowtie \mu_{HF}X \bullet\ [b](P; X), nil]_l^u)$$

3.4 Denotational Semantics of Guarded Choice

There are three types of guarded choices. In this subsection, we give their denotational semantics.

• **Instantaneous Guard Choice:** $\mathbf{beh}(\talloblong_{i\in I}\{g_i \rightarrow N_i\})$ indicates that an instantaneous action g_i is triggered, and the corresponding network N_i executes.

$$\mathbf{beh}(\talloblong_{i\in I}\{g_i \rightarrow N_i\}) =_{df} \bigvee_{i\in I} \mathbf{beh}(g_i \rightarrow N_i)$$

where, $g_i \in \{!\langle v\rangle^c@l,\ ?(x)^c@l,\ c.[v/x]@(l,l_1),\ s?y@l,\ a!v@l,\ move\_k@l,\ b_i\&\tau@l\}$.

▲ If $g_i = c.[v/x]@(l,l_1)$, then
$\mathbf{beh}(c.[v/x]@(l,l_1) \rightarrow N_i) =_{df} \mathbf{beh}(c.[v/x]@(l,l_1)); \mathbf{beh}(N_i[v/x])$,
where, $\mathbf{beh}(c.[v/x]@(l,l_1)) =_{df} \mathbf{H}\left(st' = ter \wedge \varDelta = 0 \wedge tr^\frown\langle(time', l, c.v, 1)\rangle\right)$.

The instantaneous guard $c.[v/x]@(l,l_1)$ denotes that a node at l sends value v to another node at l_1 via channel c successfully, described as $\mathbf{beh}(c.[v/x]@(l,l_1))$. According to the characteristics of broadcast communication, the behaviour of broadcast communication actions can be simply described as broadcast output commands. Therefore, the snapshot $(time', l, c.v, 1)$ is added to the end of the trace tr.

▲ If $g_i = b_i\&\tau@l$, then $\mathbf{beh}(b_i\&\tau@l \rightarrow N_i) =_{df} b_i \wedge (\mathbf{beh}(\tau@l); \mathbf{beh}(N_i))$,
where $beh(\tau@l)$ has been defined in Section 3.3 (6).

The denotational semantics of other types of instantaneous guards can be established similarly.

• **Delay Guarded Choice:**

$$\mathbf{beh}(\#t \rightarrow N) =_{df} \mathbf{beh}(\#t); \mathbf{beh}(N)$$

For the delay guarded choice, it waits for the given time units, and then N gains the control. $\mathbf{beh}(\#t)$ has been given in Section 3.3 (3).

• **Hybrid Guarded Choice:**

$$\begin{aligned} G =_{df}\ & \talloblong_{i\in I}\{g_i \rightarrow N_i\} \\ & \oplus \exists t' \in (0\ldots 1) \bullet \#t' \rightarrow N' \\ & \oplus \#1 \rightarrow N'' \end{aligned}$$

$\talloblong_{i\in I}\{g_i \rightarrow N_i\}$ is the instantaneous guarded choice. $\exists t' \in (0\ldots 1) \bullet \#t' \rightarrow N'$ has a delay guarded choice followed by the instantaneous guarded choice. $\#1 \rightarrow N''$ only contains the delay guarded choice.

$$\mathbf{beh}(G) =_{df} \begin{pmatrix} \bigvee_{i\in I} \mathbf{beh}(g_i \rightarrow N_i) \vee & \mathbf{(1)} \\ \exists t' \in (0\ldots 1) \bullet \mathbf{beh}(\#t'); \mathbf{beh}(N') \vee & \mathbf{(2)} \\ \mathbf{beh}(\#1); \mathbf{beh}(N'') & \mathbf{(3)} \end{pmatrix}$$

There are three branches corresponding to three cases that are progressive over time. In the first branch (*i.e.*, formula **(1)**), the instantaneous guard is triggered at the activation time of this process. The second branch (*i.e.*, formula **(2)**) denotes that the instantaneous guard is activated after waiting for t' time units. If the instantaneous guard cannot be triggered before one time unit, it is described by the third branch (*i.e.*, formula **(3)**). Obviously, the above three branches are disjoint.

3.5 Parallel Composition

In this subsection, we discuss the behaviour of two parallel networks. Here, a network may be a single node or the parallel composition of multiple nodes. The denotational of the parallel composition is presented below:

$$\mathbf{beh}(N_1\|N_2) =_{df} \mathbf{beh}(N_1)\|\mathbf{beh}(N_2) =_{df}$$
$$\left(\begin{array}{lr} \exists st_1, st_1', st_2, st_2', time_1, time_1', time_2, time_2', tr_1, tr_1', tr_2, tr_2' \bullet & \\ \quad\quad st_1 = st_2 = st \wedge time_1 = time_2 = time \wedge tr_1 = tr_2 = tr \wedge & \mathbf{(1)} \\ \mathbf{beh}(N_1)[st_1, st_1', time_1, time_1', tr_1, tr_1'/st, st', time, time', tr, tr'] \wedge & \mathbf{(2)} \\ \mathbf{beh}(N_2)[st_2, st_2', time_2, time_2', tr_2, tr_2'/st, st', time, time', tr, tr'] \wedge & \mathbf{(3)} \\ Merge & \mathbf{(4)} \end{array}\right)$$

The first formula (i.e., formula **(1)**) means that the initial value of the state, time, and trace of the two parallel components are the same. The following two formulas (i.e., formulas **(2)** and **(3)**) show the independent behaviour of two components. In the last one (i.e., formula **(4)**), the predicate $Merge$ is used to merge states, termination time, and the traces (snapshot sequences) contributed by the two behavioral branches.

$$Merge =_{df} \left(\begin{array}{l} ((st_1' = ter \wedge st_2' = ter) \Rightarrow st' = ter)\wedge \\ ((st_1' = div \vee st_2' = div) \Rightarrow st' = div)\wedge \\ ((st_1' = wait \wedge st_2' \neq div) \vee (st_1' \neq div \wedge st_2' = wait) \Rightarrow st' = wait) \\ \wedge time' = max(time_1', time_2')\wedge \\ \exists trace \in (tr_1' - tr_1)\|(tr_2' - tr_2) \bullet tr' = tr^\frown trace \end{array}\right)$$

If the final states of N_1 and N_2 are both ter states, the final state of $N_1\|N_2$ is ter. As long as one of them runs into a div state, the final state of $N_1\|N_2$ is div. If one of the two networks reaches a $wait$ state, and the other is not in a div state, $N_1\|N_2$ is in a $wait$ state. The termination time of the parallel composition is the maximum terminal time of N_1 and N_2.

Here, s and t represent the traces of N_1 and N_2, respectively. As mentioned before, a trace is a sequence of snapshots, and a snapshot is a quadruple (t, l, o, f). Next, we present some rules to merge traces. (**rule 1**) means that if both traces are empty (denoted by ϵ), the merged trace is still empty. (**rule 2**) indicates that the merged trace is not empty as long as one trace is not empty. (**rule 3**) denotes that the parallel composition of two traces is symmetrical.

- **(rule 1)** $\epsilon\|\epsilon =_{df} \{\epsilon\}$ • **(rule 2)** $s\|\epsilon =_{df} \{s\}$ • **(rule 3)** $s\|t =_{df} t\|s$

To emerge two nonempty traces, we propose the following notations to obtain the elements of their first snapshots (i.e., $\mathbf{head}(s)$ and $\mathbf{head}(t)$).

$$t_1 = \pi_1(\mathbf{head}(s)),\ l_1 = \pi_2(\mathbf{head}(s)),\ o_1 = \pi_3(\mathbf{head}(s)),\ f_1 = \pi_4(\mathbf{head}(s)),$$
$$t_2 = \pi_1(\mathbf{head}(t)),\ l_2 = \pi_2(\mathbf{head}(t)),\ o_2 = \pi_3(\mathbf{head}(t)),\ f_1 = \pi_4(\mathbf{head}(t)).$$

According to whether t_1 is equal to t_2, we discuss the following two cases:
(1) $t_1 = t_2$ means that communication actions o_1 and o_2 happen at the same time, resolved by (**rule 4**). Before merging them, we take out communication channels and transmitted messages from o_1 and o_2, respectively.

$$c_1 = \mathbf{Chan}(o_1), \quad m_1 = \mathbf{Mess}(o_1), \quad c_2 = \mathbf{Chan}(o_2), \quad m_2 = \mathbf{Mess}(o_2).$$

- **(rule 4)** We describe the details of (**rule 4**) through the following steps:

▲ **Step 1:** If their channels are different (i.e., $c_1 \neq c_2$), we only need to append $\mathbf{head}(s)$ or $\mathbf{head}(t)$ into the end of the merged trace, described as T'. Otherwise, we go to **Step 2** for further inspection.
▲ **Step 2:** If $f_1 = 1 \vee f_2 = 1$ is true, it means that one of them is a sender, and another one is a receiver, then we go to **Step 3**. Otherwise, they are both receivers, and we go to **Step 7**.
▲ **Step 3:** If $f_1 = 1$ is true, it denotes that the node at l_1 is the sender, and another one at l_2 is a receiver, we go to **Step 4**. Otherwise, we go to **Step 6**.
▲ **Step 4:** If the communication condition (i.e., $\mathbf{Dis}(l_1, l_2) \leq \mathbf{Rng}(c)$) is satisfied, then we go to **Step 5**. Otherwise, we merge traces as T'.
▲ **Step 5:** Both o_1 and o_2 should have the same message, i.e., $m_1 = m_2$. If so, $\mathbf{head}(s)$ is added to the end of the merged trace. Otherwise, the merged trace is an empty set $\emptyset$.
▲ **Step 6:** We discuss the case in which the node at l_2 is the sender, which is similar to steps 4 and 5.
▲ **Step 7:** Now, we explore the case in which two nodes are both receivers. If o_1 and o_2 have the same message, (i.e., $m_1 = m_2$), we extend the merged trace with the snapshot $(t_1, \{l_1, l_2\}, o_1, 2)$. Otherwise, the merged trace is $\emptyset$.

$$s \| t =_{df} \left(\begin{array}{l} \left(\begin{array}{l} \left(\begin{array}{c} \big((T_1 \lhd m_1 = m_2 \rhd \emptyset) \lhd \mathbf{Dis}(l_1, l_2) \leq \mathbf{Rng}(c_1) \rhd T' \big) \\ \lhd f_1 = 1 \rhd \\ \big((T_2 \lhd m_1 = m_2 \rhd \emptyset) \lhd \mathbf{Dis}(l_1, l_2) \leq \mathbf{Rng}(c_1) \rhd T' \big) \end{array} \right) \\ \quad \lhd f_1 = 1 \vee f_2 = 1 \rhd \big(T_3 \lhd m_1 = m_2 \rhd \emptyset \big) \end{array} \right) \\ \lhd c_1 = c_2 \rhd T' \end{array} \right),$$

where,

$$T' =_{df} \mathbf{head}(s)^\frown(\mathbf{tail}(s) \| t) \cup \mathbf{head}(t)^\frown(s \| \mathbf{tail}(t)),\ T_1 =_{df} \mathbf{head}(s)^\frown(\mathbf{tail}(s) \| \mathbf{tail}(t)),$$

$$T_2 =_{df} \mathbf{head}(t)^\frown(\mathbf{tail}(s) \| \mathbf{tail}(t)), \qquad T_3 =_{df} \langle (t_1, \{l_1, l_2\}, o_1, 2) \rangle^\frown(\mathbf{tail}(s) \| \mathbf{tail}(t)).$$

(2) If $t_1 \neq t_2$, o_1 and o_2 do not occur simultaneously, as shown in (**rule 5**).
• **(rule 5)** If $t_1 < t_2$, o_1 happens before o_2. In this case, $head(s)$ is added to the end of the merged trace. Otherwise, $head(t)$ extends the merged trace.

$$s \| t =_{df} \begin{cases} \mathbf{head}(s)^\frown(\mathbf{tail}(s) \| t), & \text{if } t_1 < t_2. \\ \mathbf{head}(t)^\frown(s \| \mathbf{tail}(t)), & \text{if } t_1 > t_2. \end{cases}$$

3.6 Channel Restriction

As mentioned before, $(vc')M$ indicates that channel c' is private to the network M. Thus, sending messages via c' is invisible outside of M. Based on the above subsections of Section 3, we can obtain the denotational semantics of M. To further gain the denotational semantics of $(vc')M$, we need to remove the snapshots which record output actions and synchronous communication actions occurring on c' from the trace of M.

$$\mathbf{beh}((vc')M) =_{df} \left(\begin{array}{l} \mathbf{beh}(M)[\mathbf{Re}(tr' - tr, c')/tr' - tr, div/str'] \\ \lhd \mathbf{Diverge}(\mathbf{beh}(M), c') \rhd \\ \mathbf{beh}(M)[\mathbf{Re}(tr' - tr, c')/tr' - tr] \end{array} \right),$$

where, $\mathbf{Diverge}(\mathbf{beh}(M), c') =_{df} \forall n \bullet \exists t \bullet t = (tr' - tr) \wedge \#(t \upharpoonright c') > n.$

If the condition $\mathbf{Diverge}(\mathbf{beh}(M), c')$ is true, it means that the behaviour of M (i.e., $\mathbf{beh}(M)$) diverges due to the concealment of channel c'. In other words, $(vc')M$ may generate an infinite sequence of hidden actions (i.e., output actions and synchronous communication actions occurring on c'). Here, $tr' - tr$ is the trace generated by M. $t \upharpoonright c'$ represents a sub-trace of t that contains only the behaviour of the hidden actions, and $\#(t \upharpoonright c')$ is its length. The following function $\mathbf{Re}(s, c')$ removes the hidden actions involving c' from a trace s, where the predefinition function π_i gets the ith element from the first snapshot $\mathbf{head}(s)$ (Page 6) of trace s.

$$\mathbf{Re}(s,c') =_{df} \left(\begin{array}{l} \left(\begin{array}{l} \mathbf{Re}(\mathbf{tail}(s), c') \\ \lhd \mathbf{Chan}(\pi_3(\mathbf{head}(s))) = c' \rhd (\mathbf{head}(s)^\frown \mathbf{Re}(\mathbf{tail}(s), c')) \end{array} \right) \\ \lhd \pi_4(\mathbf{head}(s)) = 1 \rhd (\mathbf{head}(s)^\frown \mathbf{Re}(\mathbf{tail}(s), c')) \end{array} \right)$$

Example 1. Now, we give the following example to illustrate the CaIT calculus.

$$N = (vc)(N_1 \| N_2) \| N_3 \qquad N_1 =_{n_1} [\mathit{\Gamma}_1 \bowtie !\langle v_1 \rangle^c ; \lfloor move\_l_1' ; !\langle v_2 \rangle^{c'} ; nil \rfloor nil]^m_{l_1}$$

$$N_2 =_{n_2} [\mathit{\Gamma}_2 \bowtie \lfloor ?(x)^c ; nil \rfloor nil]^{u2}_{l_2} \qquad N_3 =_{n_3} [\mathit{\Gamma}_3 \bowtie \sigma ; \lfloor ?(y)^{c'} ; nil \rfloor nil]^{u3}_{l_3}$$

For simplicity, we assume that the activeness time of N is 0, and all communication and migration conditions on distance are satisfied. For example, $Dis(l_1, l_2) \leq Rng(c)$ at time 0. The channel c is private to nodes n_1 and n_2, but c' is shared by three nodes. Next, we explore the trace of N to show the usability of our denotational semantics.
(1) At time 0, node n_1 sends the value v_1 via c to n_2. After one time unit, n_1 moves to l_1', and n_3 completes the delaying. After that, n_1 sends v_2 to n_2 at time 1. The traces of N_1, N_2, and N_3 are t, s, and w, respectively.

$$t = < (0, l_1, c.v_1, 1), (1, l_1', c'.v_2, 1) >$$

$$s = < (0, l_2, c.v_1, 2) > \qquad w = < (1, l_3, c'.v_2, 2) >$$

(2) Then, we merge the t and s by using the rules of parallel compositions. In detail, the snapshots in the above two shaded areas are merged into one snapshot shown in the following shaded area.

$$t \| s = < (0, l_1, c.v_1, 1), (1, l_1', c'.v_2, 1) >$$

(3) We conceal communication actions occurring on c to further obtain the trace T of $(vc)(N_1 \| N_2)$. Finally, we get the final trace of N by further merging w and T.

$$T = < (1, l_1', c'.v_2, 1) >$$

(4) Finally, we get the final trace of N by further merging w and T.

$$T \| w = < (1, l_1', c'.v_2, 1) >$$

4 Algebraic Semantics

4.1 Algebraic Laws of Basic Commands

We explore the algebraic laws of basic commands for the CaIT calculus. For **Input** and **Migration** commands, each one has two algebraic laws according to the different time intervals, i.e., their timers are 0 or greater than 0.

- **(Input0)** $_n[\mathit{\Gamma} \bowtie \lfloor ?(x)^c ; P \rfloor^0 Q]^u_l =_n [\mathit{\Gamma} \bowtie Q]^u_l$

- **(Migration0)** ${}_n[\Gamma \bowtie \lfloor move\_k; P\rfloor^0 Q]_l^m = [\!] \left\{\begin{array}{l} (Dis(l,k) \le \delta)\&\tau@l \to move\_k@l \to \\ \quad {}_n[\Gamma \bowtie P]_{k,\delta}^m, \\ \neg(Dis(l,k) \le \delta)\&\tau@l \to_n [\Gamma \bowtie Q]_l^m \end{array}\right\}$
- **(Delay)** ${}_n[\Gamma \bowtie \sigma; P]_l^u = \#1 \to_n [\Gamma \bowtie P]_l^u$
- **(Input)** ${}_n[\Gamma \bowtie \lfloor ?(x)^c; P\rfloor Q]_l^u = ?(x)^c@l \to_n [\Gamma \bowtie P]_l^u$

$$\oplus\, \exists t' \in (0\ldots 1) \bullet \#t' \to ?(x)^c@l \to_n [\Gamma \bowtie P]_l^u$$
$$\oplus\, \#1 \to_n [\Gamma \bowtie Q]_l^u$$

- **(Migration)** ${}_n[\Gamma \bowtie \lfloor move\_k; P\rfloor Q]_l^m = \#1 \to_n [\Gamma \bowtie \lfloor move\_k; P\rfloor^0 Q]_l^m$
- **(Output)** ${}_n[\Gamma \bowtie !\langle v\rangle^c; P]_l^u = !\langle v\rangle^c@l \to_n [\Gamma \bowtie P]_l^u$
- **(Reading Sensor)** ${}_n[\Gamma \bowtie s?y; P]_l^u = s?y@l \to_n [\Gamma \bowtie P]_l^u$
- **(Writing Actuator)**${}_n[\Gamma \bowtie a!v; P]_l^u = [\!] \left\{\begin{array}{l} (\Gamma(a) \ne v)\&\tau@l \to a!v@l \to_n [\Gamma' \bowtie P]_l^u, \\ \neg(\Gamma(a) \ne v)\&\tau@l \to_n [\Gamma \bowtie P]_l^u \end{array}\right\}$,

  where $\Gamma' = \Gamma[v/\Gamma(a)]$.
- **(Conditional)**${}_n[\Gamma \bowtie [b]P,Q]_l^u = [\!] \left\{\begin{array}{l} b\&\tau@l \to_n [\Gamma \bowtie P]_l^u, \\ \neg b\&\tau@l \to_n [\Gamma \bowtie Q]_l^u \end{array}\right\}$
- **(Iteration)** ${}_n[\Gamma \bowtie while\ b\ do\ P]_l^u = [\!] \left\{\begin{array}{l} b\&\tau@l \to_n [\Gamma \bowtie P; while\ b\ do\ P]_l^u, \\ \neg b\&\tau@l \to_n [\Gamma \bowtie nil]_l^u \end{array}\right\}$

4.2 Algebraic Laws of Parallel Composition

Now, we explore the algebraic laws of the parallel composition. The empty network 0 is the identity of parallel composition, described by (**par-1**). The parallel composition of networks is symmetric and associative, as shown in (**par-2**) and (**par-3**).

- **(par-1)** $N\|0 = N = 0\|N$
- **(par-2)** $N\|M = M\|N$
- **(par-3)** $N\|(M\|R) = (N\|M)\|R$

Then we explore the algebraic laws of the parallel composition of guarded choices. In Section 2.2, we have proposed three types of guarded choices. Thus, there should be nine parallel expansion laws. Since the parallel composition is symmetric, we only give seven parallel expansion laws, as shown in Table 2.

Table 2. Parallel composition of two guarded choices

	Instantaneous	**Delay**	**Hybrid**
Instantaneous	**(par-4-1), (par-4-2)**	**(par-5)**	**(par-6)**
Delay		**(par-7)**	**(par-8)**
Hybrid			**(par-9)**

We first investigate the parallel composition of two instantaneous guarded components, shown in (**par-4-1**) and (**par-4-2**).

- **(par-4-1)** $N = [\!]_{i\in I}\{g_i \to N_i\} \qquad M = [\!]_{j\in J}\{h_j \to M_j\}$

$$N\|M = [\!]_{i\in I}\{g_i \to N_i\|M\}[\!][\!]_{j\in J}\{h_j \to N\|M_j\}$$

Here, we suppose that there are no communications between N and M. (**par-4-1**) shows the parallel composition of two instantaneous guarded choice components without communications. In this case, the first action of $N\|M$ is either g_i or h_j, and it converts into $N_i\|M$ or $N\|M_j$ correspondingly.

- **(par-4-2)** $N = N_1 [\!] N_2 \quad N_1 = [\!]_{i\in I}\{g_i \to N_i\} \quad N_2 = [\!]_{w\in W}\{!\langle v_w\rangle^{c_w}@l_w \to N'_w\}$
  $M = M_1 [\!] M_2 \quad M_1 = [\!]_{j\in J}\{h_j \to M_j\} \quad M_2 = [\!]_{w\in W}\{?(x_w)^{c'_w}@l'_w \to M'_w\}$

We assume that there are no communications between N_1 and M_1, and the communication condition (i.e., $c_w = c'_w \wedge \mathbf{Dis}(l_w, l'_w) \le \mathbf{Rng}(c_w)$) is satisfied.

$$\begin{aligned} N\|M = &[\!]_{i\in I}\{g_i \to N_i\|M\} \\ &[\!]\ [\!]_{j\in J}\{h_j \to N\|M_j\} \\ &[\!]\ [\!]_{w\in W}\{c_w \cdot [v_w/x_w]@(l_w, l'_w) \to N'_w\|M'_w]\} \end{aligned}$$

There are three possibilities for the first action, i.e., g_i, h_j or communication action $c_w \cdot [v_w/x_w]@(l_w, l'_w)$. Due to the features of broadcast communication, $c_w \cdot [v_w/x_w]@(l_w, l'_w)$ can be seen as an output action continuing to communicate with other nodes.

- **(par-5)** $[\!]_{i\in I}\{g_i \to N_i\}\|\#t \to M = [\!]_{i\in I}\{g_i \to (N_i\|\#t \to M)\}$

Here, we discuss the parallel composition of the instantaneous guarded choice and the delay guarded choice. g_i is executed at the activation time, and then the process evolves as $N_i\|\#t \to M$.

- **(par-6)** $N = [\!]_{i\in I}\{g_i \to N_i\} \qquad M = [\!]_{j\in J}\{h_j \to M_j\}$
  $\oplus\#t' \to M'$
  $\oplus\#1 \to M''$, for $t' \in (0\ldots1)$.

$$N\|M = [\!]_{i\in I}\{g_i \to N_i\}\|[\!]_{j\in J}\{h_j \to M_j\}$$

Here, we investigate the parallel composition of the instantaneous guarded choice and the hybrid guarded choice. Clearly, the parallel composition is equal to the parallel composition of two instantaneous guarded choice components. We can obtain the final result based on the laws (i.e., (**par-4-1**) and (**par-4-2**)).

- **(par-7)** $N = \{\#t_1 \to N'\} \qquad M = \{\#t_2 \to M'\}$

This is the parallel composition of two delay guarded choice components. N_1 and N_2 wait for t_1 and t_2 time units, respectively. Here, both t_1 and t_2 range from 0 to 1. There are three possibilities:

▲ If $t_1 < t_2$, then $N\|M = \#t_1 \to (N'\|(\#(t_2 - t_1) \to M'))$.
N and M delay t_1 time units together, and then N turns into N'. But M still has to wait for $t_2 - t_1$ time units before converting into M'.

▲ If $t_1 = t_2$, then $N\|M = \#t_1 \to N'\|M'$.

▲ If $t_1 > t_2$, then$N\|M = \#t_2 \to (\#(t_1 - t_2) \to N'\|M')$.

- **(par-8)** $N = [\!]_{i\in I}\{g_i \to N_i\} \qquad M = \{\#t_1 \to M'\}$, for $t_1 \in [0\ldots1]$.
  $\oplus\ \#t' \to N'$
  $\oplus\ \#1 \to N''$, for $t' \in (0\ldots1)$.

Then, we analyze the parallel composition of the hybrid guarded choice and the delay guarded choice, which has the following two possibilities:

▲ If $t_1 < 1$, then $N\|M = [\!]_{i\in I}\{g_i \to N_i\|M\}$
$\oplus\#t' \to (N'\|\#(t_1 - t') \to M')$
$\oplus\#t_1 \to N_1\|M'$, for $t' \in (0\dots t_1)$,
where, $N_1 = [\!]_{i\in I}\{g_i \to N_i\}$
$\oplus\#t'' \to N'$
$\oplus\#(1 - t_1) \to N''$, for $t'' \in (0\dots 1 - t_1)$.

In this case, there are three alternative branches. In the first branch, g_i is triggered at the beginning time of the process. In the second one, g_i executes after waiting for t' time unit, and then this process is translated into $(N'\|\#(t_1 - t') \to M')$. For the last one, after t_1 time units, $N\|M$ evolves as $N_1\|M'$.

▲ If $t_1 = 1$, then $N\|M = [\!]_{i\in I}\{g_i \to N_i\|M\}$
$\oplus\#t'' \to (N'\|\#(1 - t'') \to M')$
$\oplus\#1 \to N''\|M'$, for $t'' \in (0\dots 1)$.

• **(par-9)** $N = [\!]_{i\in I}\{g_i \to N_i\}$
$\oplus\ \#t_1 \to [\!]_{i\in I}\{g_i \to N_i\}$
$\oplus\ \#1 \to N'$, for $t_1 \in (0\dots 1)$.

$M = [\!]_{j\in J}\{h_j \to M_j\}$
$\oplus\ \#t_2 \to [\!]_{j\in J}\{h_j \to M_j\}$
$\oplus\ \#1 \to M'$, for $t_2 \in (0\dots 1)$.

The parallel composition of networks N and M is as below.

$$N\|M = [\!]_{i\in I}\{g_i \to N_i\} \parallel [\!]_{j\in J}\{h_j \to M_j\}$$
$$\oplus\ \#t' \to ([\!]_{i\in I}\{g_i \to (N_i\|M_1)\} \parallel [\!]_{j\in J}\{h_j \to (N_1\|M_j)\})$$
$$\oplus\ \#1 \to N' \parallel M', \text{for } t' \in (0\dots 1).$$

where, $N_1 = [\!]_{i\in I}\{g_i \to N_i\}$
$\oplus\ \#t_1' \to [\!]_{i\in I}\{g_i \to N_i\}$
$\oplus\ \#(1 - t') \to N'$,

$M_1 = [\!]_{j\in J}\{h_j \to M_j\}$
$\oplus\ \#t_2' \to [\!]_{j\in J}\{h_j \to M_j\}$
$\oplus\#(1 - t') \to M'$,

for $t_1', t_2' \in (0\dots 1 - t')$.

In (**par-9**), we study the parallel composition of two hybrid guarded choice components with three branches. In the first branch, $N\|M$ equals $[\!]_{i\in I}\{g_i \to N_i\} \parallel [\!]_{j\in J}\{h_j \to M_j\}$ which has been analyzed in (**par-4-1**) and (**par-4-2**). The second branch means that an instantaneous action happens after t' time units, whose final results can be gained similar to the first case. In the third, no instantaneous action occurs before one time unit. After one time unit, $N\|M$ continues as $N'\|M'$.

• **(par-10)** According to the above laws, we find that any parallel program without restricted channels can be converted into the guarded choice form. To further describe the channel restriction (i.e., $(vc')M$), we give this law to replace output actions and synchronous communication actions happening in c' (i.e., $!\langle v\rangle^{c'}@l$ and $c'.[v/x]@(l, l_1)$) with the silent action $true\&\tau@l$, where l and l_1 are arbitrary locations in M.

▲ If $M = [\!]_{i\in I}\{g_i \to M_i\}$, then $(vc')M = [\!]_{i\in I}\{(vc')g_i \to (vc')M_i\}$.

▲ If $M = \#t \to M'$, then $(vc')M = \#t \to (vc')M'$.

▲ If $M = [\!]_{i\in I}\{g_i \to M_i\}$, then $(vc')M = [\!]_{i\in I}\{(vc')g_i \to (vc')M_i\}$
$\oplus\#t' \to M'$ $\qquad$ $\oplus\#t' \to (vc')M'$

$$\oplus \#1 \rightarrow M'' \qquad\qquad \oplus \#1 \rightarrow (vc')M'',$$

$$\text{where, } (vc')g_i = \begin{cases} true\&(\tau@l), & \text{if } g_i \in \{!\langle v\rangle^{c'}@l,\ c'.[v/x]@(l,l_1)\}. \\ g_i, & \text{otherwise.} \end{cases}$$

Example 2. Here, we continue to explore N mentioned in Example 1 (Page 12) to show the application of guarded choices and algebraic laws, where the first action of each parallel composition is represented in the shaded area.
(1.1) We get the first action of $N_1\|N_2$ by using **(par-4-2)** and **(par-1)**.

$$N_1\|N_2 = c.v_1@(l_1,l_2) \rightarrow N_1^1, \text{ where } N_1^1 =_{n_1} [\mathit{\Gamma}_1 \bowtie \lfloor move\_l_1'; !\langle v_2\rangle^{c'}; nil\rfloor nil]_{l_1}^{m}.$$

(1.2) Then we can obtain the first action of $(vc)(N_1\|N_2)$ by applying **(par-10)** to concealing the communication action happening on c.

$$(vc)(N_1\|N_2) = true\&(\tau@l_1) \rightarrow (vc)N_1^1$$

(1.3) Now, we can know the first action of N (i.e., $(vc)(N_1\|N_2)\|N_3$) with **(par-8)**.

$$(vc)(N_1\|N_2)\|N_3 = true\&(\tau@l_1) \rightarrow (vc)N_1^1\|N_3$$

(2) We further gain the first action of $(vc)N_1^1\|N_3$ according to **(par-7)** and **(par-5)**.

$$(vc)N_1^1\|N_3 = \#1 \rightarrow Dis(l_1,l_1') \leq \delta)\&\tau@l_1 \rightarrow move\_l_1'@l_1 \rightarrow (vc)N_1^2\|N_3^1,$$

$$\text{where, } (vc)N_1^2\|N_3^1 = (vc)_{n_1}[\mathit{\Gamma}_1 \bowtie !\langle v_2\rangle^{c'}; nil]_{l_1'}^{m}\|_{n_3}[\mathit{\Gamma}_3 \bowtie \lfloor ?(y)^{c'}; nil\rfloor nil]_{l_3}^{u_3}.$$

(3) We use **(par-4-2)** and **(par-1)** to get the first action of $(vc)N_1^2\|N_3^1$.

$$(vc)N_1^2\|N_3^1 = c'.v_2@(l_1',l_3)$$

After the above steps, we can finally gain the guard choice form of N. It indicates that each program of the CaIT calculus can be converted into a guard choice form, even if the channel restriction is involved. In addition, it means that a parallel program can be sequentialized by using our algebraic laws.

5 Conclusion and future work

The CaIT calculus has been proposed to specify and verify IoT systems with discrete time, while it can only support point-to-point communication. In this paper, we have enhanced the CaIT calculus by introducing the more common broadcast communication. Furthermore, we explored its denotational and algebraic semantics based on the UTP framework, focusing on broadcast communication, actions with the timeout, and even channel restriction. To establish the parallel expansion laws, we have presented three types of guarded choices so that each program can be transformed into the guarded choice form.

In the future, we will study the deductive semantics of the CaIT calculus via Hoare Logic [13]. We will further explore the semantics linking theory of the CaIT calculus and try to implement it in suitable tools like Coq, Isabelle/HOL, PVS, etc.

Acknowledgements. This work was partly supported by the National Natural Science Foundation of China (Grant Nos. 62032024, 61872145), “Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software (Grant No. 22510750100), and the Dean’s Fund of Shanghai Key Laboratory of Trustworthy Computing (East China Normal University).

References

1. Ying Zhang, Technology framework of the Internet of Things and its application, 2011 International Conference on Electrical and Control Engineering, 2011, pp. 4109-4112.
2. Ashton, K (2009) That “Internet of Things" Thing: In the Real World Things Matter More than Ideas. RFID Journal. http://www.rfidjournal.com/articles/view?4986
3. Jayavardhana Gubbi, Rajkumar Buyya: Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29(7): 1645-1660 (2013).
4. Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini: Internet of things: Vision, applications and research challenges. Ad Hoc Networks 10(7): 1497-1516 (2012).
5. Ivan Lanese, Luca Bedogni, Marco Di Felice: Internet of things: a process calculus approach. SAC 2013: 1339-1346.
6. Where Do Your IoT Ingredients Come From? COORDINATION 2016: 35-50.
7. Ruggero Lanotte, Massimo Merro: A semantic theory of the Internet of Things. Inf. Comput. 259(1): 72-101 (2018).
8. Anu Singh, C. R. Ramakrishnan, Scott A. Smolka: A process calculus for Mobile Ad Hoc Networks. Sci. Comput. Program. 75(6): 440-469 (2010).
9. Jifeng He, C. A. R. Hoare: Unifying theories of programming. RelMiCS 1998: 97-99.
10. Gordon D. Plotkin: A structural approach to operational semantics. J. Log. Algebraic Methods Program. 60-61: 17-139 (2004).
11. Joseph E. Stoy: Foundations of Denotational Semantics. Abstract Software Specifications 1979: 43-99.
12. Matthew Hennessy: Algebraic theory of processes. MIT Press series in the foundations of computing, MIT Press 1988, ISBN 978-0-262-08171-9, pp. I-VI, 1-270.
13. Krzysztof R. Apt, Frank S. de Boer, Ernst-RÃijdiger Olderog: Verification of Sequential and Concurrent Programs. Texts in Computer Science, Springer 2009, ISBN 978-1-84882-744-8, pp. i-xxiii, 1-502.