




A UTP approach for rTiMo

Wanling Xie¹  Shuangqing Xiang¹ and Huibiao Zhu^{1,2}

¹ Shanghai Key Laboratory of Trustworthy Computing,
MOE International Joint Laboratory of Trustworthy Software,
International Research Center of Trustworthy Software,
East China Normal University, Shanghai 200062, China

² School of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

Abstract. rTiMo is a real-time version of TiMo (Timed Mobility), which is a process algebra for mobile distributed systems. In this paper, we investigate the denotational semantics for rTiMo. A trace variable tr is introduced to record the communications among processes as well as the location where the communication action takes place. Based on the formalized model, we study a set of algebraic laws, especially the laws about the migration and communication with real-time constraints. In order to facilitate the algebraic reasoning about the parallel expansion laws, we enrich rTiMo with a form of *guarded choice*. This can enable us to convert every parallel program to the guarded choice form. Moreover, we also provide a set of proof rules, which can be used to verify the correctness and real-time properties of programs.

Keywords: rTiMo; Mobile systems; UTP semantics; Hoare logic

1. Introduction

With the development of cloud computing, mobile applications play an important role in modern distributed systems. Analyzing and verifying the increasing complexity of mobile applications effectively is of great significance. Ciobanu et al. [CK11b] have first introduced a process algebra called TiMo (Timed Mobility) model for mobile systems, where it is possible to add timers to the basic actions in addition to process mobility and interaction. The model of time is based on local clocks in [CK11b]. Aman et al. [AC13] have extended the TiMo family [CK11a, CJ12, CK15] by introducing a real-time version named rTiMo in which a global clock is used. The rTiMo processes can move between different locations of a mobile distributed system and communicate locally with other processes.

Regarding a programming language, there are four well-known methods for presenting semantics, including operational semantics, denotational semantics, algebraic semantics and deductive semantics (originally called axiomatic semantics) [Hoa13]. The operational semantics of a programming language suggests a complete set of possible individual steps which may be taken in its execution [Plo04, Mil80]. An operational semantics usually consists of a set of transition rules. Denotational Semantics [Sto79] provides the mathematical meanings to programs. The approach of denotational semantics is under a purely mathematical basis. Thus, the denotational semantics is more abstract. Compared with operational semantics, denotational semantics expresses *what a program does*. An algebraic semantics [Hen88, HHH⁺87] consists of a set of algebraic laws and their deduced properties and it fits well with symbolic calculation of parameters and structures of an optimal design, which has been used in provably-correct compilation [He94, HHS93, DCS10]. Deductive semantics [Hoa69, O'H07] of a programming language provides a set of proof rules, which can be used to specify and verify correctness and general properties of a program.

Correspondence and offprint requests to: Huibiao Zhu (Email: hbzhu@sei.ecnu.edu.cn)
This paper extends the work published at UTP 2016 [XX16].

In [AC13], Aman et al. have investigated the operational semantics of rTiMo. This paper considers the denotational semantics for rTiMo, which can provide the precise understanding of rTiMo and deduce interesting properties of programs. In our semantic model, we introduce a variable tr to record the communications among processes. Based on the formalized denotational semantics, we investigate a set of algebraic laws, which not only comprises algebraic laws similar to traditional algebraic laws, but also contains algebraic laws about migration and communication with time constraints. In order to facilitate algebraic reasoning about parallel expansion laws, we enrich rTiMo with a guarded choice construct which is classified into three types: communication guarded choice, delay guarded choice and hybrid guarded choice. From a set of parallel expansion laws that we have explored, we can see that every parallel program can be converted to the guarded choice form. In addition, we also introduce a formalism based on Hoare Triples (precondition, program, postcondition) to specify and verify the correctness and the real-time behavior of programs.

The remainder of this paper is organized as follows. Section 2 introduces the syntax of rTiMo and the concept of guarded choice which is used to study the parallel expansion laws. Section 3 explores the denotational semantics for rTiMo. Section 4 presents a set of algebraic laws, including the basic algebraic laws and the parallel expansion laws which enable us to convert a parallel program to the guarded choice form. Section 5 provides a set of proof rules, which are used to verify the correctness and progress properties. Section 6 is the related work about the formal modeling and analysis of mobile distributed systems, rTiMo, UTP approach and Hoare Logic. Section 7 concludes the paper and discusses some possible future work.

2. The rTiMo calculus

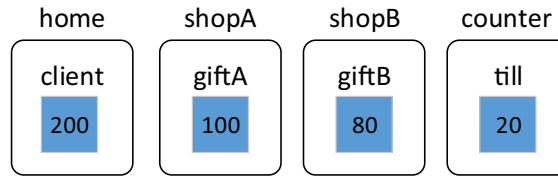
In this section, we will give a brief introduction to the rTiMo calculus. rTiMo is a real-time version of TiMo [CK11b], which is a process algebra for mobile distributed systems. rTiMo processes can move between different locations in a distributed environment and communicate locally with other processes. In other words, the processes from different locations cannot communicate with each other. The syntax of rTiMo is given in Table 1 as it has been introduced in [AC13]. In rTiMo, process migration and communication are controlled by using real-time constraints. Timeouts are specified by a superscript Δt where t is a timeout (deadline) of an action and $t \in \mathbb{R}_+$.

In rTiMo, the communication channels are point-point, i.e. each channel connects two processes, and synchronous. Synchronous communication inside the network is by handshake synchronization of input and output actions. rTiMo calculus enjoys some time properties such as (1) time determinism: the passage of time is deterministic; (2) maximal progress: data transmissions cannot be delayed, they must occur as soon as a possibility for communication arises. For example, let $N = l[[a^{\Delta t}!(v) \text{ then } P \text{ else } Q \mid a^{\Delta t}?(u) \text{ then } P' \text{ else } Q']]$, where $t > 0$, then the output action and the input action take place via channel a at the activation time of N without any time delay.

1. $a^{\Delta t}!(v) \text{ then } P \text{ else } Q$ indicates that an output process can send message v for a period of t time units via channel a . When the message v is sent successfully within t time units, the next process is P . If the communication does not happen before the timeout t , the communication attempt is aborted and the next process is Q .
2. $a^{\Delta t}?(u) \text{ then } P \text{ else } Q$ stands for an input process whose waiting time interval is t time units. When the process receives a message via channel a within t time units, the control passes to process P . If the communication does not take place before the deadline t , the waiting process gives up and it switches to the alternative process Q . The input process binds the variable u within P (but not within Q).
3. $go^{\Delta t}l \text{ then } P$ denotes a migration process that moves to location l precisely after t time units. In other words, after t time units, the migration action terminates and the next process is P running at the location l . And the terminal time of the migration action is the activated time of the process P .
4. 0 denotes the process that terminates without taking any time.
5. $P \mid Q$ stands for parallel composition.
6. $l[[P]]$ specifies a process P running at location l .
7. $L \mid L \mid N$ indicates a network can be a located process L or can be built from its components $L \mid N$.

Table 1. rTiMo syntax

Processes	$P, Q ::= a^{\Delta t}!(v) \text{ then } P \text{ else } Q \mid$	(output)
	$a^{\Delta t}?(u) \text{ then } P \text{ else } Q \mid$	(input)
	$go^{\Delta t} l \text{ then } P \mid$	(move)
	$0 \mid$	(termination)
	$P \mid Q$	(parallel)
Located processes	$L ::= l[[P]]$	
Networks	$N ::= L \mid L \mid N$	

**Fig. 1.** The initial network

We can also introduce the recursion in rTiMo. The semantics of the recursion can be defined according to Pratt's definition [Pra90], using the Knaster-Tarski fixed-point theorem [Tar55].

In order to support our algebraic expansion laws, we enrich rTiMo with a new concept, called guarded choice, which is classified into three types:

1. $\llbracket_{i \in I} \{g_i \rightarrow N_i\}$ is communication guarded choice where g_i is a communication guard. The guard is instantaneous which means that it happens without any time delay. The guard g_i can be expressed as $a!(v)@l$, $a?(u)@l$ or $a.[v/u]@l$, where $a!(v)@l$ ($a?(u)@l$) indicates that the output (input) action happens at location l and uses the channel a , and $a.[v/u]@l$ denotes that the communication over channel a takes place at location l and the variable u is replaced by the message v .
2. $\#t \rightarrow N$ is delay guarded choice, where $\#t$ means delaying t time units and $t \geq 0$. After delaying t time units, the subsequent network is N .
3. The third guarded choice is hybrid guarded choice, which has the following form where the notation \oplus denotes the disjointness of timed behaviors:

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

Example 2.1 Consider a simple shopping example. A client wants to buy a gift in the shopping mall in a short time and at a good price. The scenario involves four locations and four processes. The role of each process represented in Fig. 1 is as below:

- *client* is a process that initially resides in the *home* location and has an amount of 200 cash. It wants to buy a gift after comparing the prices of the gift in the two shops (*shopA* and *shopB*). After entering *shopA*, the *client* receives the price of this gift in *shopA* and then the *client* moves to the *shopB* location to receive the price of this gift in *shopB*. Finally, the *client* moves to the *counter* location to pay for the cheaper price and returns to the *home* location.
- *giftA* communicates with the *client* to offer the price of this gift in *shopA*.

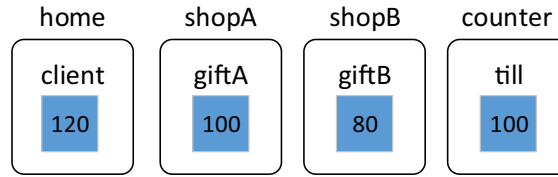


Fig. 2. A possible final network

- *giftB* communicates with the *client* to offer the price of this gift in *shopB*.
- *till* resides at the *counter* and has an initial amount of 20 cash. It can receive e-money paid by the *client*.

We use the following shorthand notations:

$a!(v) \rightarrow P$ represents $a^{\Delta\infty}!(v)$ then P else 0;

$a?(u) \rightarrow P$ represents $a^{\Delta\infty}?(u)$ then P else 0.

The rTiMo syntax of these processes is as below:

$$\begin{aligned} \text{client}(\text{init}) &= \text{go}^{\Delta 4} \text{shopA} \text{ then } \text{priceA}?(p_1) \rightarrow \text{go}^{\Delta 3} \text{shopB} \text{ then } \text{priceB}?(p_2) \rightarrow \\ &\quad \text{pay}!(\min\{p_1, p_2\}) \rightarrow \text{go}^{\Delta 5} \text{home} \text{ then } \text{client}(\text{init} - \min\{p_1, p_2\}) \end{aligned}$$

$$\text{giftA}(\text{price}) = \text{priceA}!(\text{price}) \rightarrow \text{giftA}(\text{price})$$

$$\text{giftB}(\text{price}) = \text{priceB}!(\text{price}) \rightarrow \text{giftB}(\text{price})$$

$$\text{till}(\text{cash}) = \text{pay}^{\Delta 1}?(payment) \text{ then } \text{till}(\text{cash} + payment) \text{ else } \text{till}(\text{cash}).$$

A possible final network after 12 time units is represented in Fig. 2. □

3. Denotational semantics

UTP [HH98] covers wide areas of fundamental theories of programs in a formalised style and acts as a consistent basis for the principles of programming language. It provides the denotational semantics for Dijkstra's non-deterministic sequential programming language based on predicate calculus of Tarki's theory of relations [Tar55]. A complete algebra for the sequential programming language is derived, which can transform every program to a normal form under a restricted notation. Many advanced programming language features are introduced, including parallel composition, reactive processes and declarative programming.

UTP Theory is regarded as an excellent methodology for studying various kinds of programs. The important advantages of denotational semantics are given as below:

1. "The behavior of each program can be predicated without actually executing it on a computer, and similarly the semantics of a program language can be understood as a whole without visualizing how programs run on a computer" [Wat91].
2. "Based on mathematical theory, we can reason about programs, for example, to prove that one program is equivalent to another" [Wat91].

In this section, we present the denotational semantics for the rTiMo model. We investigate the semantic model of rTiMo and healthiness conditions which a program should satisfy in Sect. 3.1. In Sect. 3.2, we explore the behaviors of the basic commands, including the located migration process, the located input process and the located output process. The behaviors of guarded choice introduced in Section 2, which includes communication guarded choice, delay guarded choice and hybrid guarded choice, are investigated in Sect. 3.3. In Sect. 3.4, we investigate the behavior of the parallel composition.

3.1. The semantic model

In this subsection, we explore the denotational semantics model for rTiMo. Our approach is based on the relational calculus [HH98]. The time model in our semantic model is discrete. We introduce a pair of variables st and st' into our semantic model in order to denote the execution state of a program. st represents the initial execution state of a program before its activation and st' stands for the final execution state of the program during the current observation. A program may have two execution states:

1. *completed state* : A program has reached the *completed state* when it terminates successfully. “ $st = completed$ ” indicates that the previous program has terminated successfully and control passes to the current program. “ $st' = completed$ ” indicates that the current program has terminated successfully and control passes to the next program.
2. *wait state* : A program may wait for communicating with another program via a specific channel or moving from one location to another after the given time units. “ $st = wait$ ” indicates that the predecessor of the current program is in a waiting state. Thus, the current program cannot be activated. “ $st' = wait$ ” indicates that the current program itself is in a waiting state. Thus, the next program cannot be activated.

We describe the behavior of a process in terms of a trace of snapshots which record the sequence of the communication actions that the process is able to engage in. In our semantic model, we introduce a variable tr to denote that trace. The behavior of a communication action is specified by a snapshot which can be expressed as a triple (t, κ, l) where:

- t indicates the time when the communication action takes place.
- κ denotes the message transmitted via a specific channel at the termination of a communication action. And the form of κ is $a.v$, where a indicates the communication channel and v is the message transmitted. We define $\mathbf{Chan}(\kappa)$ to obtain the communication channel and $\mathbf{Mess}(\kappa)$ to obtain the message, i.e., if $\kappa = a.v$, then $\mathbf{Chan}(a.v) = a$ and $\mathbf{Mess}(a.v) = v$.
- l records the location at which the communication action takes place.

Compared to many existing UTP theories [SZL⁺18, HH98], in addition to the communication, rTiMo calculus has other interesting features, including time constrains and location information. Thus, the observations of an rTiMo program can be described by a tuple:

$$(\overleftarrow{time}, \overrightarrow{time}, st, st', tr, tr'), \text{ where}$$

- \overleftarrow{time} and \overrightarrow{time} respectively denote the start point and the end point of the time interval over which the observation is recorded.
- st represents the initial execution state of the program before its activation and st' stands for its final execution state during the current observation.
- tr represents the initial trace of a program over the interval which is passed by its predecessor. tr' stands for the final trace of a program over the interval. $tr' - tr$ denotes the sequence of snapshots contributed by the program itself during the interval. The t in the snapshot contributed by the program during the interval ranges in $[\overleftarrow{time}, \overrightarrow{time}]$, that is to say, $t \in [\overleftarrow{time}, \overrightarrow{time}]$.

We now define $;$ operator for our semantic model to describe the behavior of sequential composition. It is similar to the definition of sequential composition in relational calculus.

Definition 3.1 $P; Q =_{df} \exists t, s, r \bullet P[t/\overrightarrow{time}, s/st', r/tr'] \wedge Q[t/\overleftarrow{time}, s/st, r/tr]$.

Example 3.2 Let $N_1 = l_1[[go^{\Delta 3} b_2 \text{ then } a^{\Delta 2}!\langle v_1 \rangle \text{ then } b^{\Delta 3}?(u_2) \text{ else } 0]]$,

$$N_2 = l_2[[a^{\Delta 6}?(u_1) \text{ then } b^{\Delta 2}!\langle v_2 \rangle \text{ else } 0]].$$

Above, we use the following shorthand notations:

$b^{\Delta 3}?(u_2)$ stands for $b^{\Delta 3}?(u_2) \text{ then } 0 \text{ else } 0$,

$b^{\Delta 2}!\langle v_2 \rangle$ stands for $b^{\Delta 2}!\langle v_2 \rangle \text{ then } 0 \text{ else } 0$.

Consider the trace of $N_3 = N_1 \mid N_2$. Assume the activation time of N_3 is at 0.

According to the syntax of rTiMo, the migration process moves to location l_2 from location l_1 after 3 time units. Thus, the communication action takes place at time 3.

One possible trace of N_1 is given as below:

$$\langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle$$

A possible trace of N_2 is shown next:

$$\langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle$$

Hence, the one trace of N_3 is as follows:

$$\langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle \quad \square$$

Every program will always satisfy some given healthiness conditions which are defined as equations according to an idempotent function ϕ on predicates. And for a predicate P denoting a healthy program, we have $P = \phi(P)$. We next consider the healthiness conditions rTiMo programs should satisfy, and they are similar to the standard UTP theory [HH98]. In our semantics model, the variable tr is used to record the execution trace of a program, so this variable cannot be shortened. A formula P satisfies the healthiness condition (H1).

(H1) $P = P \wedge Inv(tr)$, where $Inv(tr) =_{df} tr \preceq tr'$, which states that tr is a prefix of tr' .

As we have mentioned earlier on, a program may wait for communicating with another program via a specific channel or moving from one location to another after a given time delay. For the migration process $home[[go^{\Delta 5} shop \text{ then } P]]$ which indicates that process P moves to location $shop$ from location $home$ after 5 time units: if process P is asked to start in a waiting state of $go^{\Delta 5} shop$, then P keeps itself unchanged; i.e., it satisfies the following healthiness condition.

(H2) $P = \Pi \triangleleft st = wait \triangleright P$,

where $\Pi =_{df} (st' = st) \wedge (\overrightarrow{time} = \overleftarrow{time}) \wedge (tr' = tr)$

and $P \triangleleft b \triangleright Q =_{df} (b \wedge P) \vee (\neg b \wedge Q)$

The variable $time$ is used to record the progress of a program and no program can ever make time go backwards, thus, it cannot be smaller. The predicate P which describes an rTiMo program behavior must therefore imply this fact. \overleftarrow{time} and \overrightarrow{time} denote the start point and the end point of a program in our semantic model respectively, thus, P should satisfy the following healthiness condition.

(H3) $P = P \wedge (\overleftarrow{time} \leq \overrightarrow{time})$

The definition of H function can be given as follows:

$$H(X) =_{df} \Pi \triangleleft st = wait \triangleright (X \wedge Inv(tr) \wedge \overleftarrow{time} \leq \overrightarrow{time})$$

From the definition of H function, we know that $H(X)$ satisfies the healthiness conditions (H1), (H2) and (H3). The H function is used to define the denotational semantics for the rTiMo model.

Lemma 3.3 $H(X1 \vee X2); H(X) = (H(X1); H(X)) \vee (H(X2); H(X))$

The lemma above specifies a distributive law. The function H distributes through disjunction $H(X1 \vee X2) = H(X1) \vee H(X2)$. Similar to the standard sequential composition, we have $(X1 \vee X2); X = (X1; X) \vee (X2; X)$. These two laws together provide justification for this distributive law.

3.2. Denotational semantics for basic commands

In this subsection, we investigate the behaviors of the basic commands, which include the located migration process, the located input process and the located output process. We use $\mathbf{beh}(l[[P]])$ to describe the behavior of a process P running at location l .

We first investigate the denotational semantics of 0 at the location l . It is a termination process and the execution state, the terminal time and the trace all keep unchanged.

$$\mathbf{beh}(l[[0]]) =_{df} H \left(st' = st \wedge \overrightarrow{time} = \overleftarrow{time} \wedge tr' = tr \right)$$

$N; M$ denotes the behavior that runs N and M sequentially and its semantics is given as below:

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

We then explore the behavior of the located migration process $l[[go^{\Delta t} l' \text{ then } P]]$, it indicates that process P moves to location l' from location l after t time units.

$$\mathbf{beh}(l[[go^{\Delta t} l' \text{ then } P]]) =_{df} H \left(\left(\begin{array}{c} (st' = wait \wedge \overrightarrow{time} - \overleftarrow{time} < t) \\ \vee \\ (st' = completed \wedge \overrightarrow{time} = \overleftarrow{time} + t) \end{array} \right) \wedge tr' = tr \right); \mathbf{beh}(l'[[P]])$$

For t time units, the migration action is in a waiting state and its trace is unchanged. After t time units elapse, the migration action terminates successfully and the trace also remains unchanged. We record the terminal time of the migration action using $\overrightarrow{time} = \overleftarrow{time} + t$ which is also the activation time of the next action. After the completion of the migration action, the subsequent behavior of the program is the behavior of process P which runs at location l' .

The located input process $l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]]$ indicates that if the program successfully receives an input via channel a within t time units, then process P from location l gets the control. On the other hand, if the communication does not happen before the deadline t , the waiting process gives up and it switches to the alternative process Q located at l . The notation $tr_1 \hat{\ } tr_2$ is used to denote the concatenation of the two traces tr_1 and tr_2 .

$$\mathbf{beh}(l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]]) =_{df}$$

$$\left(\begin{array}{c} \exists m \in \mathbf{Type}(a) \bullet \left(H \left(\begin{array}{c} (st' = wait \wedge tr' = tr \wedge \\ 0 < \overrightarrow{time} - \overleftarrow{time} < t) \vee \\ (st' = completed \wedge \\ \text{append}(a?(m)@l) \wedge \\ 0 \leq \overrightarrow{time} - \overleftarrow{time} < t) \end{array} \right); \mathbf{beh}(l[[P[m/u]]]) \right) \\ \vee \\ H \left(st' = completed \wedge tr' = tr \wedge \overrightarrow{time} = \overleftarrow{time} + t \right); \mathbf{beh}(l[[Q]]) \end{array} \right)$$

where, $\text{append}(a?(m)@l) =_{df} tr' = tr \hat{\ } (\overrightarrow{time}, a.m, l) \wedge \neg(\forall t' \in (\overrightarrow{time}, \infty) \bullet tr' = tr \hat{\ } (t', a.m, l))$

Here, $\mathbf{Type}(a)$ represents the type of the messages that channel a can communicate. There are two alternative behavior branches to describe the execution of the located input process.

- **Case 1:** within t time units, the input action either waits for receiving a message via a specific channel, or communicates with the corresponding output action successfully. If the input action waits for receiving a message, the execution state is *wait* and the trace is unchanged. If the input action happens successfully, the execution state is *completed* and the snapshot $(\overrightarrow{time}, a.m, l)$ contributed by the input action is attached to the end of the program trace, which is described by the first predicate $tr' = tr \hat{\ } (\overrightarrow{time}, a.m, l)$ in $\text{append}(a?(m)@l)$. The predicate $\neg(\forall t' \in (\overrightarrow{time}, \infty) \bullet tr' = tr \hat{\ } (t', a.m, l))$ is used to ensure that the input action takes place as soon as it is enabled. The subsequent behavior of the program is determined by process P from location l . The notation $P[m/u]$ denotes P in which all free occurrences of a variable u are replaced by m .

- **Case 2:** the input action does not take place before the deadline t , the execution state is *completed* and the trace remains unchanged. In this case, the subsequent behavior of the program is the behavior of the alternative process Q from location l .

The located output process $l[[a^{\Delta t}!\langle v \rangle \text{ then } P \text{ else } Q]]$ means that if the program sends the message v along channel a within t time units successfully, the control passes into process P . On the other hand, if the communication does not happen before the timeout t , the communication gives up and the control passes into process Q .

$\mathbf{beh}(l[[a^{\Delta t}!\langle v \rangle \text{ then } P \text{ else } Q]]) =_{df}$

$$\left(\begin{array}{l} H \left(\begin{array}{l} (st' = \text{wait} \wedge tr' = tr \wedge 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \vee \\ (st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \\ 0 \leq \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \end{array} \right); \mathbf{beh}(l[[P]]) \\ \vee \\ H \left(st' = \text{completed} \wedge tr' = tr \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} + t \right); \mathbf{beh}(l[[Q]]) \end{array} \right)$$

where, $\text{append}(a!\langle v \rangle @l) =_{df} tr' = tr \hat{\ } (\overrightarrow{\text{time}}, a.v, l) \wedge \neg(\forall t' \in (\overrightarrow{\text{time}}, \infty) \bullet tr' = tr \hat{\ } (t', a.v, l))$

Similar to the located input process, there are also two branches to describe the execution of the located output process. The first branch describes the case that the output action takes place within t time units and the case that the output action does not happen before the deadline t is illustrated in the second branch.

3.3. Denotational semantics for guarded choice

As mentioned earlier, the guarded choice has three types: communication guarded choice, delay guarded choice and hybrid guarded choice. Now we give the denotational semantics for these three types of guarded choice below.

Communication Guarded Choice. We first consider the communication guarded choice, which is composed of a set of communication guarded components. There are three types of the communication guard: $a!\langle v \rangle @l$, $a?(u) @l$ and $a.[v/u] @l$, which all have been described earlier.

$\mathbf{beh}(\llbracket_{i \in I} \{g_i \rightarrow N_i\}\rrbracket) =_{df} \bigvee_{i \in I} \mathbf{beh}(g_i \rightarrow N_i)$, where

if $g = a!\langle v \rangle @l$, then

$\mathbf{beh}(g \rightarrow N) =_{df} H \left(st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} \right); \mathbf{beh}(N)$

where $\text{append}(a!\langle v \rangle @l)$ has been defined in the denotational semantics of the located output process in Sect. 3.2. The guard $a!\langle v \rangle @l$ is an instantaneous action, which means that it takes place without any time delay, thus the termination time of $a!\langle v \rangle @l$ equals to its activation time.

If $g = a?(u) @l$, then

$\mathbf{beh}(g \rightarrow N) =_{df}$

$\left(\exists m \in \mathbf{Type}(a) \bullet \left(H \left(st' = \text{completed} \wedge \text{append}(a?(m) @l) \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} \right); \mathbf{beh}(N[m/u]) \right) \right)$

where $\text{append}(a?(m) @l)$ has been defined in the denotational semantics of the located input process.

If $g = a.[v/u] @l$, then

$\mathbf{beh}(g \rightarrow N) =_{df} H \left(st' = \text{completed} \wedge \text{append}(a.[v/u] @l) \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} \right); \mathbf{beh}(N[v/u])$

where, $\text{append}(a.[v/u] @l) =_{df} tr' = tr \hat{\ } (\overrightarrow{\text{time}}, a.v, l) \wedge \neg(\forall t' \in (\overrightarrow{\text{time}}, \infty) \bullet tr' = tr \hat{\ } (t', a.v, l))$

Delay Guarded Choice. For the delay guarded choice, it consists of only one time delay component. And we can find that the denotational semantics of the delay guarded choice is similar to the denotational semantics of the located migration process introduced in Sect. 3.2.

$$\mathbf{beh}(\#t \rightarrow N) =_{df} H \left(\left(\begin{array}{l} (st' = wait \wedge \overrightarrow{time} - \overleftarrow{time} < t) \vee \\ (st' = completed \wedge \overrightarrow{time} = \overleftarrow{time} + t) \end{array} \right) \wedge tr' = tr \right); \mathbf{beh}(N)$$

Hybrid Guarded Choice. The hybrid guarded choice has the following form:

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

where, $\llbracket_{i \in I} \{g_i \rightarrow N_i\}$ is communication guarded choice, $\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N_i\})$ consists of a delay guard followed by a communication guarded choice where $t' \in (0 \dots t)$ and $\#t \rightarrow N'$ is delay guarded choice.

$$\mathbf{beh}(N) =_{df} \left(\begin{array}{c} \bigvee_{i \in I} \mathbf{beh}(g_i \rightarrow N_i) \\ \bigvee \\ \bigvee_{1 \leq i \leq n} \left(H \left(\begin{array}{l} (st' = wait \wedge tr' = tr \wedge \\ 0 < \overrightarrow{time} - \overleftarrow{time} < t) \vee \\ (st' = completed \wedge \mathit{append}(g_i) \wedge \\ 0 < \overrightarrow{time} - \overleftarrow{time} < t) \end{array} \right); \mathbf{beh}(N'_i) \right) \\ \bigvee \\ H \left(st' = completed \wedge tr' = tr \wedge \overrightarrow{time} = \overleftarrow{time} + t \right); \mathbf{beh}(N') \end{array} \right)$$

From the denotational semantics of the hybrid guarded choice, we can find that the three branches correspond to three cases which are progressive over time. The first branch describes the case that the action takes place at the activation time. The second branch describes the case that the action waits for some time units before it happens. And the case that the action does not take place before the deadline t is described by the third branch. As we know, the three branches are progressive over time. Thus, from the time-related predicate, we can find that the three branches are disjoint.

3.4. Denotational semantics for parallel composition

In this subsection, we explore the behavior of a network which is composed of a set of located processes running in parallel. Let P and Q be the processes. The parallel composition $l[[P]] \mid l'[[Q]]$ performs process P from location l and process Q from location l' running in parallel, where l and l' can be the same or different locations. Its behavior is the composition of the behaviors of the two parallel components by merging their traces together. The composition is described by the following definition.

$\mathbf{beh}(l[[P]] \mid l'[[Q]]) = \mathbf{beh}(l[[P]]) \mid \mathbf{beh}(l'[[Q]])$, where

$$l[[P_1]] \mid l'[[P_2]] =_{df} \left(\begin{array}{l} \left(\exists st_1, st'_1, st_2, st'_2, tr_1, tr'_1, tr_2, tr'_2, \overleftarrow{time}_1, \overrightarrow{time}_1, \overleftarrow{time}_2, \overrightarrow{time}_2 \bullet \right. \\ tr_1 = tr_2 = tr \wedge st_1 = st_2 = st \wedge \overleftarrow{time}_1 = \overleftarrow{time}_2 = \overleftarrow{time} \wedge \\ P_1[st_1, st'_1, tr_1, tr'_1, \overleftarrow{time}_1, \overrightarrow{time}_1 / st, st', tr, tr', \overleftarrow{time}, \overrightarrow{time}] \wedge \\ P_2[st_2, st'_2, tr_2, tr'_2, \overleftarrow{time}_2, \overrightarrow{time}_2 / st, st', tr, tr', \overleftarrow{time}, \overrightarrow{time}] \wedge \\ \left. \text{Merge} \right) \end{array} \right)$$

The first three predicates in the definition $l[[P_1]] \mid l'[[P_2]]$ describe the two independent behaviors of process P_1 from location l and P_2 from location l' running in parallel. The last predicate *Merge* mainly does the merge of the contributed traces of the two behavioral branches for recording the communications, which is defined as below.

$$Merge =_{df} \left(\begin{array}{l} (st'_1 = completed \wedge st'_2 = completed) \Rightarrow st' = completed \wedge \\ (st'_1 = wait \vee st'_2 = wait) \Rightarrow st' = wait \wedge \\ \exists s \in (tr'_1 - tr) \mid (tr'_2 - tr) \bullet tr' = tr \hat{\ } s \wedge \\ \overrightarrow{time} = \mathbf{max}\{\overrightarrow{time}_1, \overrightarrow{time}_2\} \end{array} \right)$$

The final execution state of the behavior of the parallel composition is determined by the two parallel components together. The contributed traces of the two behaviors for recording the communication can be merged. And the terminal time of the parallel composition is the maximum between the two terminal times of the parallel components.

We first introduce some notations which will be used in the trace-merging definitions. The notation **head**(s) is used to denote the first snapshot of the trace s and **tail**(s) is used to denote the result of removing the first snapshot in the trace s . We use the projections to select the components of a snapshot:

$$\pi_1((t, \kappa, l)) =_{df} t \quad \pi_2((t, \kappa, l)) =_{df} \kappa \quad \pi_3((t, \kappa, l)) =_{df} l$$

The merging of the contributed traces of the two behaviors for recording the communication can be defined as follows. The result of merging two empty traces (represented as ϵ) is still empty which is illustrated in the first definition. For the two traces which are required to be merged, if one of them is empty and the other is nonempty, the result of trace merging follows the nonempty one and the second and third definitions describe the case.

$$\mathbf{case\ 1} \quad \epsilon \mid \epsilon =_{df} \{\epsilon\} \quad \mathbf{case\ 2} \quad s \mid \epsilon =_{df} \{s\} \quad \mathbf{case\ 3} \quad \epsilon \mid t =_{df} \{t\}$$

If both traces are nonempty, then we can use the following cases to merge the two traces. We below obtain the first snapshot in the two traces respectively.

$$\begin{aligned} t_1 &= \pi_1(\mathbf{head}(s)), & \kappa_1 &= \pi_2(\mathbf{head}(s)), & l_1 &= \pi_3(\mathbf{head}(s)), \\ t_2 &= \pi_1(\mathbf{head}(t)), & \kappa_2 &= \pi_2(\mathbf{head}(t)), & l_2 &= \pi_3(\mathbf{head}(t)). \end{aligned}$$

We first consider the case that $t_1 = t_2$ which means that the two actions κ_1 and κ_2 take place at the same time (denoted by **case 4**).

$$\mathbf{case\ 4} \quad s \mid t =_{df} \left\{ \begin{array}{l} \left(\left(\langle (t_1, \kappa_1, l_1) \rangle \hat{\ } (\mathbf{tail}(s) \mid \mathbf{tail}(t)) \triangleleft \mathbf{Mess}(\kappa_1) = \mathbf{Mess}(\kappa_2) \triangleright \emptyset \right) \right. \\ \quad \left. \triangleleft \mathbf{Chan}(\kappa_1) = \mathbf{Chan}(\kappa_2) \triangleright \right. \\ \left. \langle (t_1, \kappa_1, l_1) \rangle \hat{\ } (\mathbf{tail}(s) \mid t) \cup \langle (t_2, \kappa_2, l_2) \rangle \hat{\ } (s \mid \mathbf{tail}(t)) \right) \\ \left. \langle (t_1, \kappa_1, l_1) \rangle \hat{\ } (\mathbf{tail}(s) \mid t) \cup \langle (t_2, \kappa_2, l_2) \rangle \hat{\ } (s \mid \mathbf{tail}(t)), \right. \end{array} \right. \begin{array}{l} \text{, if } l_1 = l_2; \\ \text{otherwise.} \end{array}$$

The communications in rTiMo are local, thus, in the fourth case, we first consider whether the two locations l_1 and l_2 are same or not. If l_1 and l_2 are different, then a synchronization does not take place. We only need to attach **head**(s) or **head**(t) to the end of the program trace. If the two locations are same, then we have the following descriptions.

- If $\mathbf{Chan}(\kappa_1)$ equals to $\mathbf{Chan}(\kappa_2)$ which means that the two channels are same, then we consider the messages. If $\mathbf{Mess}(\kappa_1)$ equals to $\mathbf{Mess}(\kappa_2)$ which means that the two messages are same, then a synchronization occurs and a snapshot (t_1, κ_1, l_1) contributed by this communication is generated. On the other hand, if the two messages are different, then the result of trace merging is empty set \emptyset .
- If $\mathbf{Chan}(\kappa_1)$ and $\mathbf{Chan}(\kappa_2)$ are different, then a synchronization does not happen and we only need to attach **head**(s) or **head**(t) to the end of the program trace.

We now consider the case $t_1 < t_2$ which means that κ_1 occurs before κ_2 (denoted by **case 5**). In this case, we only need to attach the first snapshot of s to the end of the program trace.

case 5 $s \mid t =_{df} \langle (t_1, \kappa_1, l_1) \rangle \hat{\ } (\mathbf{tail}(s) \mid t)$.

Finally, we consider the case $t_1 > t_2$ which means that κ_2 occurs before κ_1 (denoted by **case 6**). In this case, we only need to attach the first snapshot of t to the end of the program trace.

case 6 $s \mid t =_{df} \langle (t_2, \kappa_2, l_2) \rangle \hat{\ } (s \mid \mathbf{tail}(t))$.

Example 3.5 Consider the network $N_1 \mid N_2$ in Example 3.2, where the trace of N_1 is below:

$$s = \langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle$$

And the trace of N_2 is below:

$$t = \langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle$$

According to the trace-merging definition **case 4**, we can obtain

$$s \mid t = \langle (3, a.v_1, l_2) \rangle \hat{\ } (s' \mid t')$$

where $s' = \langle (3, b.v_2, l_2) \rangle$, $t' = \langle (3, b.v_2, l_2) \rangle$

According to the trace-merging definition **case 4**, we can obtain

$$s' \mid t' = \langle (3, b.v_2, l_2) \rangle \hat{\ } (\epsilon \mid \epsilon)$$

According to the trace-merging definition **case 1**, we can obtain

$$\epsilon \mid \epsilon = \{\epsilon\}$$

Finally, we obtain the trace of the network by merging s and t below:

$$\langle (3, a.v_1, l_2), (3, b.v_2, l_2) \rangle$$

□

4. Algebraic properties

Our work towards the formalization of rTiMo aims to deduce its interesting properties, which are usually expressed using algebraic laws and equations [HH98]. In this section, we explore a set of algebraic laws for rTiMo including basic algebraic laws and a set of parallel expansion laws. The basic algebraic laws are introduced in Sect. 4.1. In Sect. 4.2, we investigate the parallel expansion laws. And from these laws, we can find that every parallel program can be converted to the guarded choice form.

We have presented the three types of guarded choice, including communication guarded choice, delay guarded choice and hybrid guarded choice. Then, we find that the parallel composition of any two components can be converted into the guarded choice form, i.e., the law (**para-6**) in Sect. 4.2 is for the parallel composition of a located output process and a located migration process and from this law, we can find that it can be converted into a hybrid guarded choice. And in Sect. 4.2, we also investigate the laws for the parallel composition of any two guarded choice components. And from these laws, we find that the form of the parallel result is still contained in the three types of guarded choice. For example, the law (**para-9**) is for the parallel composition of a communication guarded choice and delay guarded choice and from this law, we can see that it can be converted into a communication guarded choice.

4.1. Basic algebraic laws

If a migration action has a timer which equals to 0, then process P migrates from location l to l' without any time delay.

$$\mathbf{(move-1)} \quad l[[go^{\Delta 0} l' \text{ then } P]] = l'[[P]]$$

A communication action has a timer which equals to 0, the process $a^{\Delta 0} * \text{ then } P \text{ else } Q$ continues as the alternative process Q . Here, $*$ $\in \{!\langle v \rangle, ?\langle u \rangle\}$.

$$\mathbf{(output-1)} \quad l[[a^{\Delta 0} !\langle v \rangle \text{ then } P \text{ else } Q]] = l[[Q]]$$

(input-1) $l[[a^{\Delta 0}(u) \text{ then } P \text{ else } Q]] = l[[Q]]$

For the located migration process $l[[go^{\Delta t}l' \text{ then } P]]$, it first delays t time units, then process P moves to location l' .

(move-2) $l[[go^{\Delta t}l' \text{ then } P]] = \#t \rightarrow l'[[P]]$ where $t > 0$.

For the located output process, if the output action happens at the start of the program, the subsequent process is P from location l . On the other hand, the output action needs to wait for the specific input action triggered. If the waiting time t' ranges in $(0 \dots t)$, the subsequent process is still P . The subsequent process is Q from location l when the output action delays t time units, which means that the communication gives up and the control passes into the alternative process Q .

(output-2) $l[[a^{\Delta t}! \langle v \rangle \text{ then } P \text{ else } Q]]$

$$\begin{aligned} &= a!\langle v \rangle @l \rightarrow l[[P]] \\ &\oplus \exists t' \in (0 \dots t) \bullet (\#t' \rightarrow (a!\langle v \rangle @l \rightarrow l[[P]]) \\ &\oplus \#t \rightarrow l[[Q]], \text{ where } t > 0. \end{aligned}$$

Proof

RHS

= {Def of Hybrid Guarded Choice}

$$\begin{aligned} &\left(\begin{array}{l} H(st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}}); \mathbf{beh}(l[[P]]) \\ \vee \\ H \left(\begin{array}{l} (st' = \text{wait} \wedge tr' = tr \wedge 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \vee \\ (st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \\ 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \end{array} \right); \mathbf{beh}(l[[P]]) \\ \vee \\ H(st' = \text{completed} \wedge tr' = tr \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} + t); \mathbf{beh}(l[[Q]]) \end{array} \right) \\ &= \{(H(X1); Y) \vee (H(X2); Y) = H(X1 \vee X2); Y\} \\ &\left(\begin{array}{l} H \left(\begin{array}{l} (st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}}) \vee \\ (st' = \text{wait} \wedge tr' = tr \wedge 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \vee \\ (st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \\ 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \end{array} \right); \mathbf{beh}(l[[P]]) \\ \vee \\ H(st' = \text{completed} \wedge tr' = tr \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} + t); \mathbf{beh}(l[[Q]]) \end{array} \right) \end{aligned}$$

= {Logical Equivalence}

$$\left(\begin{array}{l} H \left(\begin{array}{l} (st' = \text{wait} \wedge tr' = tr \wedge 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t) \vee \\ (st' = \text{completed} \wedge \text{append}(a!\langle v \rangle @l) \wedge \\ (\overrightarrow{\text{time}} = \overleftarrow{\text{time}} \vee 0 < \overrightarrow{\text{time}} - \overleftarrow{\text{time}} < t)) \end{array} \right); \mathbf{beh}(l[[P]]) \\ \vee \\ H(st' = \text{completed} \wedge tr' = tr \wedge \overrightarrow{\text{time}} = \overleftarrow{\text{time}} + t); \mathbf{beh}(l[[Q]]) \end{array} \right)$$

$$\begin{aligned}
&= \{(\overrightarrow{time} = \overleftarrow{time} \vee 0 < \overrightarrow{time} - \overleftarrow{time} < t) = 0 \leq \overrightarrow{time} - \overleftarrow{time} < t\} \\
&\left(\begin{array}{l} H \left(\begin{array}{l} (st' = wait \wedge tr' = tr \wedge 0 < \overrightarrow{time} - \overleftarrow{time} < t) \vee \\ (st' = completed \wedge append(a!\langle v \rangle @ l) \wedge \\ 0 \leq \overrightarrow{time} - \overleftarrow{time} < t) \end{array} \right); \mathbf{beh}(l[[P]]) \\ \vee \\ H \left(st' = completed \wedge tr' = tr \wedge \overrightarrow{time} = \overleftarrow{time} + t \right); \mathbf{beh}(l[[Q]]) \end{array} \right) \\
&= \{\text{Def of Output Process}\} \\
&LHS \qquad \qquad \qquad \square
\end{aligned}$$

$$\begin{aligned}
\mathbf{(input-2)} \quad &l[[a^{\Delta t}?(u) \text{ then } P \text{ else } Q]] \\
&= a?(u)@l \rightarrow l[[P]] \\
&\oplus \exists t' \in (0..t) \bullet (\#t' \rightarrow (a?(u)@l \rightarrow l[[P]])) \\
&\oplus \#t \rightarrow l[[Q]], \text{ where } t > 0.
\end{aligned}$$

The located input process has the similar description with the located output process.

4.2. Algebraic laws for parallel composition

The located process $l[[0]]$ is the identity of parallel composition.

$$\mathbf{(para-1)} \quad N \mid l[[0]] = N = l[[0]] \mid N$$

The parallel composition \mid is symmetric and associative.

$$\mathbf{(para-2)} \quad N \mid M = M \mid N$$

$$\mathbf{(para-3)} \quad N \mid (M \mid R) = (N \mid M) \mid R$$

$$\begin{aligned}
\mathbf{(para-4)} \quad &\text{Let } N_1 = l[[a^{\Delta t}!\langle v \rangle \text{ then } P \text{ else } Q]], N_2 = l[[a^{\Delta t}?(u) \text{ then } P' \text{ else } Q']], \\
&N_3 = N_1 \mid N_2, \text{ where } t > 0.
\end{aligned}$$

$$\text{Then, } N_3 = a.[v/u]@l \rightarrow (l[[P]] \mid l[[P'[v/u]]])$$

As mentioned earlier, rTiMo calculus enjoys the time property maximal progress: data transmissions cannot be delayed, they must occur as soon as a possibility for communication arises. In the law **(para-4)**, the output action and the input action both are enabled at the current activation time of N_3 , thus, the communication takes place at this time without time delay. In this law, an output process, from location l , succeeds in sending the message v over channel a to an input process from location l without any time delay. Both processes continue to execute at location l , the first one as P , the second one as $P'[v/u]$.

$$\begin{aligned}
\mathbf{(para-5)} \quad &\text{Let } N_1 = l[[go^{\Delta t_1} l_1 \text{ then } P_1]], N_2 = l[[go^{\Delta t_2} l_2 \text{ then } P_2]], \\
&N_3 = N_1 \mid N_2, \text{ where } t_1 > 0 \text{ and } t_2 > 0.
\end{aligned}$$

Then, we have the following three cases:

$$t_1 < t_2: N_3 = \#t_1 \rightarrow (l_1[[P_1]] \mid \#(t_2 - t_1) \rightarrow l_2[[P_2]])$$

$$t_1 = t_2: N_3 = \#t_1 \rightarrow (l_1[[P_1]] \mid l_2[[P_2]])$$

$$t_1 > t_2: N_3 = \#t_2 \rightarrow (\#(t_1 - t_2) \rightarrow l_1[[P_1]] \mid l_2[[P_2]])$$

Law **(para-5)** is about the one for the parallel composition of two migration processes. There are three cases in this law. The first case is that t_1 is the smaller one and in this case N_3 delays t_1 time units first, process P_1 moves to location l_1 and the second process P_2 still needs to wait $t_2 - t_1$ time units. For the case $t_1 = t_2$, after delaying t_1 (t_2) time units, P_1 moves to location l_1 and P_2 moves to location l_2 . In the case $t_1 > t_2$, N_3 delays t_2 time units first, process P_2 then moves to location l_2 and the first process P_1 still needs to wait $t_1 - t_2$ time units.

(para-6) Let $N_1 = l[[a^{\Delta t_1}! \langle v \rangle \text{ then } P \text{ else } Q]]$, $N_2 = l[[go^{\Delta t_2} l' \text{ then } P']]$,

$$N_3 = N_1 \mid N_2, \text{ where } t_1 > 0 \text{ and } t_2 > 0.$$

Then we have three cases for N_3 :

$$t_1 < t_2: N_3 = a! \langle v \rangle @ l \rightarrow (l[[P]] \mid N_2)$$

$$\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (a! \langle v \rangle @ l \rightarrow l[[P]] \mid \#(t_2 - t') \rightarrow l'[[P']]))$$

$$\oplus \#t_1 \rightarrow (l[[Q]] \mid \#(t_2 - t_1) \rightarrow l'[[P']])$$

$$t_1 = t_2: N_3 = a! \langle v \rangle @ l \rightarrow (l[[P]] \mid N_2)$$

$$\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (a! \langle v \rangle @ l \rightarrow l[[P]] \mid \#(t_2 - t') \rightarrow l'[[P']]))$$

$$\oplus \#t_1 \rightarrow (l[[Q]] \mid l'[[P']])$$

$$t_1 > t_2: N_3 = a! \langle v \rangle @ l \rightarrow (l[[P]] \mid N_2)$$

$$\oplus \exists t' \in (0 \dots t_2) \bullet (\#t' \rightarrow (a! \langle v \rangle @ l \rightarrow l[[P]] \mid \#(t_2 - t') \rightarrow l'[[P']]))$$

$$\oplus \#t_2 \rightarrow (l[[a^{\Delta t_1 - t_2}! \langle v \rangle \text{ then } P \text{ else } Q]] \mid l'[[P']]).$$

For the parallel composition of a located output process and a located migration process indicated in law **(para-6)**, we have to consider the three cases $t_1 < t_2$, $t_1 = t_2$ and $t_1 > t_2$. Since the three cases have the similar descriptions, here, we take the case $t_1 < t_2$ as an example.

In the case $t_1 < t_2$, if the output action occurs at the start of the program, the located output process evolves as process P from location l and the located migration process keeps itself unchanged. On the other hand, the output action enters a waiting state, if the waiting time t' ranges in $(0 \dots t_1)$, N_3 delays t' time units first, the output action happens and the located output process continues as P from location l , the located migration process should still wait $t_2 - t'$ time units. If the output action does not take place before the timeout t_1 , N_3 delays t_1 time units, the located output process continues as the alternative process Q from location l and the located migration process still needs to wait $t_2 - t_1$ time units.

(para-7) Let $N_1 = l[[a^{\Delta t_1}! \langle v \rangle \text{ then } P_1 \text{ else } Q_1]]$, $N_2 = l[[b^{\Delta t_2}?(u) \text{ then } P_2 \text{ else } Q_2]]$,

$$N_3 = N_1 \mid N_2, \text{ where } 0 < t_1 < t_2.$$

Then, $N_3 = a! \langle v \rangle @ l \rightarrow (l[[P_1]] \mid N_2) \parallel b?(u) @ l \rightarrow (N_1 \mid l[[P_2]])$

$$\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow ($$

$$a! \langle v \rangle @ l \rightarrow (l[[P_1]] \mid l[[b^{\Delta t_2 - t'}?(u) \text{ then } P_2 \text{ else } Q_2]])$$

$$\parallel b?(u) @ l \rightarrow (l[[a^{\Delta t_1 - t'}! \langle v \rangle \text{ then } P_1 \text{ else } Q_1]] \mid l[[P_2]]))$$

$$\oplus \#t_1 \rightarrow (l[[Q_1]] \mid l[[b^{\Delta t_2 - t_1}?(u) \text{ then } P_2 \text{ else } Q_2]])$$

In law **(para-7)**, the located output process and the located input process do not share the same communication channel, which means that there is no message communication between them. The law **(para-7)** describes the following three cases:

- **Case 1:** At the start point of the program, at least one of output action over channel a ($a! \langle v \rangle @ l$) and input action over channel b ($b?(u) @ l$) occurs, the first action of N_3 is either $a! \langle v \rangle @ l$ or $b?(u) @ l$. When $a! \langle v \rangle @ l$ is the first action of N_3 , the located output process continues to be process P_1 from location l and the located input process remains unchanged. If the first action of N_3 is $b?(u) @ l$, the located output process keeps itself unchanged and the located input process continues as process P_2 from location l .

Table 2. Parallel composition of two guarded choice components

	Communication GC	Delay GC	Hybrid GC
Communication GC	✓	✓	✓
Delay GC	✓	✓	✓
Hybrid GC	✓	✓	✓

- **Case 2:** N_3 enters the waiting state, if the waiting time t' , which denotes the time $a!(v)@l$ or $b?(u)@l$ happens, ranges in $(0...t_1)$, N_3 delays t' time units first and the next action is either $a!(v)@l$ or $b?(u)@l$.
- **Case 3:** If neither $a!(v)@l$ nor $b?(u)@l$ takes place before the timeout t_1 , then N_3 delays t_1 time units first, the located output process continues as process Q from location l and the timer for the located input process is $\Delta^{t_2-t_1}$.

Next we investigate the laws for the parallel composition of two guarded choice components. And from these laws, we can find that every parallel program can be converted into the guarded choice form. We have three types about the guarded choice, including communication guarded choice, delay guarded choice and hybrid guarded choice. Thus, there are six cases about the parallel composition of two guarded choice components, which are illustrated in Table 2, where *GC* denotes *Guarded Choice*. The first case is the parallel composition of two communication guarded choice components and the laws for the first case are given in **(para-8-1)** and **(para-8-2)**.

(para-8-1) Let $N = \prod_{i \in I} \{g_i \rightarrow N_i\}$ and $M = \prod_{j \in J} \{h_j \rightarrow M_j\}$.

Assume that there is no message communication between N and M .

Then, $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N_i \mid M)\} \prod_{j \in J} \{h_j \rightarrow (N \mid M_j)\}$

Law **(para-8-1)** reflects the parallel composition of two communication guarded choices, in which two communication components do not share the same communication channels, which means there is no message communication between them. The case that two parallel communication components can communicate with each other is illustrated in law **(para-8-2)**.

(para-8-2) Let $N1 = \prod_{i \in I} \{g_i \rightarrow N1_i\}$ and $M1 = \prod_{j \in J} \{h_j \rightarrow M1_j\}$,

$$N = N1 \prod_{k \in K} \{a_k!(v_k)@l_k \rightarrow N2_k\},$$

$$M = M1 \prod_{k \in K} \{a_k?(u_k)@l_k \rightarrow M2_k\}.$$

Assume that there are no communications between $N1$ and $M1$.

Then, $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N1_i \mid M)\} \prod_{j \in J} \{h_j \rightarrow (N \mid M1_j)\}$

$$\prod_{k \in K} \{a_k.[v_k/u_k]@l_k \rightarrow (N2_k \mid M2_k)\}$$

The second case is the parallel composition of communication guarded choice and delay guarded choice. And the law **(para-9)** is for the second case. In the law **(para-9)**, the communication guard g_i is executed first, the subsequent network evolves as $(N_i \mid \#t \rightarrow N)$.

(para-9) $\prod_{i \in I} \{g_i \rightarrow N_i\} \mid \#t \rightarrow N = \prod_{i \in I} \{g_i \rightarrow (N_i \mid \#t \rightarrow N)\}$

The third case is the parallel composition of communication guarded choice and hybrid guarded choice, which is indicated in the laws **(para-10-1)** and **(para-10-2)**.

(para-10-1) Let $N = \prod_{i \in I} \{g_i \rightarrow N_i\}$,

$$M = \prod_{j \in J} \{h_j \rightarrow M_j\}$$

$$\oplus \exists t' \in (0...t) \bullet (\#t' \rightarrow (\prod_{j \in J} \{h_j \rightarrow M_j\}))$$

$$\oplus \#t \rightarrow M'.$$

Assume that there is no communication between N and M .

Then, $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N_i \mid M)\} \prod_{j \in J} \{h_j \rightarrow (N \mid M_j)\}$

In the law **(para-10-1)**, there is no communication between the two components, thus, the first action can be g_i or can be h_j . The case that there is a message communication between the two components is showed in the law **(para-10-2)**.

(para-10-2) Let $N1 = \prod_{i \in I} \{g_i \rightarrow N1_i\}$, $N = N1 \prod_{k \in K} \{a_k!(v_k)@l_k \rightarrow N2_k\}$,

$$\begin{aligned} M1 &= \prod_{j \in J} \{h_j \rightarrow M1_j\} \\ &\oplus \exists t' \in (0 \dots t) \bullet (\#t' \rightarrow (\prod_{j \in J} \{h_j \rightarrow M1'_j\})) \\ &\oplus \#t \rightarrow M1', \\ M &= \prod_{j \in J} \{h_j \rightarrow M1_j\} \prod_{k \in K} \{a_k?(u_k)@l_k \rightarrow M2_k\} \\ &\oplus \exists t' \in (0 \dots t) \bullet (\#t' \rightarrow (\prod_{j \in J} \{h_j \rightarrow M1'_j\})) \\ &\oplus \#t \rightarrow M1', \end{aligned}$$

Assume that there is no communication between $N1$ and $M1$.

Then, $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N1_i \mid M)\} \prod_{j \in J} \{h_j \rightarrow (N \mid M1_j)\} \prod_{k \in K} \{a_k.[v_k/u_k]@l_k \rightarrow (N2_k \mid M2_k)\}$

The fourth case is the parallel composition of two delay guarded choices. And the laws for the fourth case are given in **(para-11-1)**, **(para-11-2)** and **(para-11-3)**.

(para-11-1) $\#t_1 \rightarrow N_1 \mid \#(t_1 + t_2) \rightarrow N_2 = \#t_1 \rightarrow (N_1 \mid \#t_2 \rightarrow N_2)$

(para-11-2) $\#t_1 \rightarrow N_1 \mid \#t_1 \rightarrow N_2 = \#t_1 \rightarrow (N_1 \mid N_2)$

(para-11-3) $\#(t_1 + t_2) \rightarrow N_1 \mid \#t_2 \rightarrow N_2 = \#t_2 \rightarrow (\#t_1 \rightarrow N_1 \mid N_2)$

The fifth case is the parallel composition of delay guard choice and hybrid guarded choice. And the law for the fifth case is indicated in the law **(para-12)**.

(para-12) Let $N = \prod_{i \in I} \{g_i \rightarrow N_i\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\prod_{i \in I} \{g_i \rightarrow N'_i\})) \\ &\oplus \#t_1 \rightarrow N', \\ M &= \#t_2 \rightarrow M'. \end{aligned}$$

Then, $N \mid M$ has three cases:

$t_1 < t_2$: $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N_i \mid M)\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\prod_{i \in I} \{g_i \rightarrow N'_i\} \mid \#(t_2 - t') \rightarrow M')) \\ &\oplus \#t_1 \rightarrow (N' \mid \#(t_2 - t_1) \rightarrow M') \end{aligned}$$

$t_1 = t_2$: $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N_i \mid M)\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\prod_{i \in I} \{g_i \rightarrow N'_i\} \mid \#(t_2 - t') \rightarrow M')) \\ &\oplus \#t_1 \rightarrow (N' \mid M') \end{aligned}$$

$t_1 > t_2$: $N \mid M = \prod_{i \in I} \{g_i \rightarrow (N_i \mid M)\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_2) \bullet (\#t' \rightarrow (\prod_{i \in I} \{g_i \rightarrow N'_i\} \mid \#(t_2 - t') \rightarrow M')) \\ &\oplus \#t_2 \rightarrow (N1 \mid M') \end{aligned}$$

where,

$$\begin{aligned} N1 &= \llbracket_{i \in I} \{g_i \rightarrow N_i\} \\ &\oplus \exists t' \in (0 \dots t_1 - t_2) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N'_i\})) \\ &\oplus \#(t_1 - t_2) \rightarrow N' \end{aligned}$$

In the law **(para-12)**, there are three cases for the parallel composition of hybrid guarded choice and delay guarded choice, including $t_1 < t_2$, $t_1 = t_2$ and $t_1 > t_2$. Here, we take $t_1 < t_2$ as an example to explain the execution of the two parallel components. If the communication action in N takes place at the activation time of the program, then g_i is the first action of $N \mid M$ and the subsequent network is $N_i \mid M$. The second branch means that the communication action in N needs to wait t' time units where t' ranges in $(0 \dots t_1)$. The third branch indicates that the communication action in N does not happen before the timeout t_1 , then $N \mid M$ delays t_1 time units first, N evolves as N' and M evolves as $\#(t_2 - t_1) \rightarrow M'$.

The last case in Table 2 is the parallel composition of hybrid guarded choice and hybrid guarded choice, and the law for the sixth case is given in **(para-13-1)** and **(para-13-2)**.

(para-13-1) Let $N = \llbracket_{i \in I} \{g_i \rightarrow N_i\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N'_i\})) \\ &\oplus \#t_1 \rightarrow N', \\ M &= \llbracket_{j \in J} \{h_j \rightarrow M_j\} \\ &\oplus \exists t' \in (0 \dots t_2) \bullet (\#t' \rightarrow (\llbracket_{j \in J} \{h_j \rightarrow M'_j\})) \\ &\oplus \#t_2 \rightarrow M', \text{ where } t_1 < t_2. \end{aligned}$$

Assume that there is no message communication between N and M .

Then, $N \mid M = \llbracket_{i \in I} \{g_i \rightarrow (N_i \mid M)\} \llbracket_{j \in J} \{h_j \rightarrow (N \mid M_j)\}$

$$\begin{aligned} &\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow (N'_i \mid M1)\} \llbracket_{j \in J} \{h_j \rightarrow (N1 \mid M'_j)\})) \\ &\oplus \#t_1 \rightarrow (N' \mid M2) \end{aligned}$$

where, $N1 = \llbracket_{i \in I} \{g_i \rightarrow N_i\}$

$$\begin{aligned} &\oplus \exists t'' \in (0 \dots t_1 - t') \bullet (\#t'' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N'_i\})) \\ &\oplus \#(t_1 - t') \rightarrow N', \\ M1 &= \llbracket_{j \in J} \{h_j \rightarrow M_j\} \\ &\oplus \exists t'' \in (0 \dots t_2 - t') \bullet (\#t'' \rightarrow (\llbracket_{j \in J} \{h_j \rightarrow M'_j\})) \\ &\oplus \#(t_2 - t') \rightarrow M', \\ M2 &= \llbracket_{j \in J} \{h_j \rightarrow M_j\} \\ &\oplus \exists t'' \in (0 \dots t_2 - t_1) \bullet (\#t'' \rightarrow (\llbracket_{j \in J} \{h_j \rightarrow M'_j\})) \\ &\oplus \#(t_2 - t_1) \rightarrow M' \end{aligned}$$

In the law **(para-13-1)**, there is no message communication between N and M . $\llbracket_{i \in I} \{g_i \rightarrow (N'_i \mid M)\}$ in $N \mid M$ stands for that g_i in N takes place at the activation time of the program, and after that, N evolves as N_i and M keeps unchanged. $\llbracket_{j \in J} \{h_j \rightarrow (N \mid M_j)\}$ represents that h_j in M is the first action of the two parallel components, N keeps itself unchanged and M evolves as M_j . The second branch in $N \mid M$ means that N and M both need to wait for some time units before the communication actions happen. Thus, $N \mid M$ first delays t' time units, where t' ranges in $(0 \dots t_1)$. After delaying t' time units, the next action can be g_i or can be h_j . The third branch $\#t_1 \rightarrow (N' \mid M2)$ in $N \mid M$ indicates that the communication actions in N and M both do not take place within t_1 time units, thus, after delaying t_1 time units, N evolves as N' and M evolves as $M2$.

The case that N and M can communicate with each other is showed in **(para-13-2)**.

$$\begin{aligned}
\text{(para-13-2)} \quad \text{Let } N3 &= \llbracket_{i \in I} \{g_i \rightarrow N_i\} \\
&\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N'_i\})) \\
&\oplus \#t_1 \rightarrow N', \\
M3 &= \llbracket_{j \in J} \{h_j \rightarrow M_j\} \\
&\oplus \exists t' \in (0 \dots t_2) \bullet (\#t' \rightarrow (\llbracket_{j \in J} \{h_j \rightarrow M'_j\})) \\
&\oplus \#t_2 \rightarrow M', \\
N &= \llbracket_{i \in I} \{g_i \rightarrow N_i\} \rrbracket_{k \in K} \{a_k!(v_k)@l_k \rightarrow N4_k\} \\
&\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow N'_i\} \rrbracket_{k \in K} \{a_k!(v_k)@l_k \rightarrow N4'_k\})) \\
&\oplus \#t_1 \rightarrow N', \\
M &= \llbracket_{j \in J} \{h_j \rightarrow M_j\} \rrbracket_{k \in K} \{a_k?(u_k)@l_k \rightarrow M4_k\} \\
&\oplus \exists t' \in (0 \dots t_2) \bullet (\#t' \rightarrow (\llbracket_{j \in J} \{h_j \rightarrow M'_j\} \rrbracket_{k \in K} \{a_k?(u_k)@l_k \rightarrow M4'_k\})) \\
&\oplus \#t_2 \rightarrow M', \text{ where } t_1 < t_2.
\end{aligned}$$

Assume that there is no communication between $N3$ and $M3$.

$$\begin{aligned}
\text{Then, } N \mid M &= \llbracket_{i \in I} \{g_i \rightarrow (N_i \mid M)\} \rrbracket_{j \in J} \{h_j \rightarrow (N \mid M_j)\} \rrbracket_{k \in K} \{a_k.[v_k/u_k]@l_k \rightarrow (N4_k \mid M4_k)\} \\
&\oplus \exists t' \in (0 \dots t_1) \bullet (\#t' \rightarrow (\llbracket_{i \in I} \{g_i \rightarrow (N'_i \mid M1)\} \rrbracket_{j \in J} \{h_j \rightarrow (N1 \mid M'_j)\} \\
&\quad \rrbracket_{k \in K} \{a_k.[v_k/u_k]@l_k \rightarrow (N4'_k \mid M4'_k)\})) \\
&\oplus \#t_1 \rightarrow (N' \mid M2)
\end{aligned}$$

where $N1$, $M1$ and $M2$ have been defined in the law **(para-13-1)**.

5. Proof system for rTiMo

In this section, we present a proof system for rTiMo in order to prove the correctness of real-time systems formalized in rTiMo. Program correctness is expressed by so-called *correctness formulas* [AdBO09]. The formalism to specify real-time system which are described using rTiMo is proposed in Sect. 5.1. In Sect. 5.2, we will introduce some auxiliary axioms and rules, which also have been introduced in [AdBO09, Hoo94, Hoo91]. In Sect. 5.3, we give the proof rules for the basic commands. The proof rules for parallel composition are presented in Sect. 5.4.

5.1. Specification

The specifications in our paper are based on Hoare triples (precondition, program, postcondition) [Hoa69]. The formulas have the form $\{p\} N \{q\}$ where p and q are assertions and N is a network. The precondition p expresses assumptions about the values of local objects at the start of N , the beginning time of N and the timed occurrence of observable actions. And the postcondition q expresses assumptions about the values of the local objects at termination (if N terminates), the termination time (∞ if N does not terminate) and the timed occurrence of observable actions. Compared with classical Hoare triples, we add time in assertions, this means that our formalism can deal with partial correctness as well as progress properties. The assertions p and q in a formula $\{p\} N \{q\}$ are expressed in a first-order logic with the following primitives:

- Val denotes a set of logical value variables, i.e. $v_0, v_1, v_2, \dots, v_n$.
- $Time \cup \{\infty\}$ denotes a set of logical time variables, i.e. t_0, \dots, t_n , where $Time = \{t \in \mathbb{R} \mid t \geq 0\}$.
- A special variable $time$ denotes the global clock, ranging over $Time \cup \{\infty\}$. An occurrence of $time$ in precondition p stands for the starting time of network N whereas in postcondition q it represents the termination

time ($time = \infty$ denotes the nonterminating computation, i.e., a waiting process is in an infinite waiting state).

- Var denotes a set of program variables, ranging over Val . Let $Var(N)$ be the set of program variables in network N , and we require $Var(N_1) \cap Var(N_2) = \emptyset$ for parallel composition $N_1 \mid N_2$.
- $Chan$ denotes a set of communication channels. We define $Chan(N)$ to denote the set of communication channels in network N .

We next define some useful notations which will be used in our proof system:

- $A \text{ at } t$ denotes the observable action A happens at time t .
- $A \text{ during } I \equiv \forall t \in I : A \text{ at } t$ where A denotes the observable action and I denotes the time interval, i.e. $[t_0, t_1)$ ($t_0 < t_1$).
- $p[v/u]$ denotes that the substitution of v for each free occurrence of variable u in assertion p .

In rTiMo, the communication channels are point-point, i.e. each connecting two processes, and synchronous. To describe communication by message passing along synchronous channels, the assertion contains the following primitives, where $a \in Chan$, $v \in Val$ and $t \in Time$.

- $a!@l \text{ at } t$: it denotes a process at location l is waiting to send a value via channel a at time t .
- $(a!@l, v) \text{ at } t$: it denotes a process at location l starts sending value v via channel a at time t .
- $a?@l \text{ at } t$: it denotes that a process at location l is waiting to receive a value via channel a at time t .
- $(a?@l, v) \text{ at } t$: it denotes that a process at location l starts receiving value v via channel a at time t .

5.2. Auxiliary axioms and rules

In this subsection, we introduce some auxiliary axioms and rules presented in [AdBO09, Hoo94, Hoo91].

Nontermination axiom expresses that a program following a nonterminating computation has no effect.

Axiom 1. Nontermination

$$\{p \wedge time = \infty\} N \{p \wedge time = \infty\}$$

For invariance axiom, it expresses that an assumption satisfying certain restrictions is not affected by the execution of any program.

Axiom 2. Invariance

$$\{p\} N \{p\}$$

provided no free variable of p is subject to change in N .

We can use the substitution rule to replace a logical variable in the precondition by any arbitrary expression if this variable does not occur in the postcondition.

Rule 3. Substitution

$$\frac{\{p\} N \{q\}}{\{p[v/u]\} N \{q\}}$$

provided that u does not occur free in q .

Rule 4. Quantification

$$\frac{\{p\} N \{q\}}{\{\exists s : p\} N \{q\}}$$

provided s does not occur in q .

The conditional rule formalizes a case distinction according to the truth value of B .

Rule 5. Conditional

$$\frac{\{p \wedge B\} N_1 \{q\}, \{p \wedge \neg B\} N_2 \{q\}}{\{p\} \text{ if } B \text{ then } N_1 \text{ else } N_2 \text{ fi } \{q\}}$$

Our proof system contains the disjunction and conjunction rules which are identical to the classical rules.

Rule 6. Disjunction

$$\frac{\{p_1\} N \{q_1\}, \{p_2\} N \{q_2\}}{\{p_1 \vee p_2\} N \{q_1 \vee q_2\}}$$

Rule 7. Conjunction

$$\frac{\{p_1\} N \{q_1\}, \{p_2\} N \{q_2\}}{\{p_1 \wedge p_2\} N \{q_1 \wedge q_2\}}$$

Rule 8. Composition

$$\frac{\{p\} N_1 \{p_1\}, \{p_1\} N_2 \{q\}}{\{p\} N_1; N_2 \{q\}}$$

Rule 9. Consequence

$$\frac{p \rightarrow p_1, \{p_1\} N \{q_1\}, q_1 \rightarrow q}{\{p\} N \{q\}}$$

The consequence rule allows us to strengthen the precondition and weaken the postcondition in a correctness formula and enables the application of other proof rules.

The sequentialization rule is for disjoint parallel programs.

Rule 10. Sequentialization

$$\frac{\{p\} N_1; N_2 \{q\}}{\{p\} N_1 \mid N_2 \{q\}}$$

5.3. Proof rules for basic commands

In this subsection, we introduce the proof rules for the basic commands, including the empty command, output command, input command and move command.

0 terminates immediately and has no effect.

Axiom 11. 0

$$\{p\} 0 \{p\}$$

There are two possibilities about the rule for the output construct $l[[a^{\Delta t}!(v) \text{ then } P_1 \text{ else } P_2]]$. One possibility is that the output action takes place within t time units after the starting time t_0 , leading to assertion p_1 and after that process P_1 located at location l is executed leading to assertion q_1 . In this case, the output action can happen at the start point t_0 . On the other hand, the output action needs to wait for the specific input action triggered. The other possibility is that the output action does not happen before the timeout t , leading to assertion p_2 and after that process P_2 at location l is executed leading to q_2 .

Rule 12. Output

$$\frac{\begin{aligned} &(p \wedge \text{time} < \infty)[t_0/\text{time}] \wedge (((a!@l, v) \text{ at } t_0 \wedge \text{time} = t_0) \vee \\ &(\exists t' \in (t_0, t_0 + t) : a!@l \text{ during } [t_0, t') \wedge (a!@l, v) \text{ at } t' \wedge \text{time} = t')) \rightarrow p_1 \\ &(p \wedge \text{time} < \infty)[t_0/\text{time}] \wedge a!@l \text{ during } [t_0, t_0 + t) \wedge \text{time} = t_0 + t \rightarrow p_2 \\ &\{p_i\} l[[P_i]] \{q_i\}, \text{ for } i = 1, 2 \end{aligned}}{\{p \wedge \text{time} < \infty\} l[[a^{\Delta t}!(v) \text{ then } P_1 \text{ else } P_2]] \{q_1 \vee q_2\}}$$

There are also two possibilities about the rule for the input construct $l[[a^{\Delta t}?(u) \text{ then } P_1 \text{ else } P_2]]$. One possibility is that the input action takes place within t time units after the starting time t_0 leading to assertion p_1 and after that process P_1 located at location l is executed leading to assertion q_1 . In this case, the input action

either happens at the start point t_0 or waits for the specific output action triggered. The other possibility is that the input action does not happen before the deadline t , which leads to assertion p_2 and after that process P_2 located location l is executed leading to q_2 .

Rule 13. Input

$$\begin{array}{l} (p \wedge time < \infty)[t_0/time] \wedge (((a?@l, v) \text{ at } t_0 \wedge time = t_0) \vee \\ (\exists t' \in (t_0, t_0 + t) : a?@l \text{ during } [t_0, t'] \wedge (a?@l, v) \text{ at } t' \wedge time = t')) \rightarrow p_1[v/u] \\ (p \wedge time < \infty)[t_0/time] \wedge a?@l \text{ during } [t_0, t_0 + t] \wedge time = t_0 + t \rightarrow p_2 \\ \{p_i\} l[[P_i]] \{q_i\}, \text{ for } i = 1, 2 \\ \hline \{p \wedge time < \infty\} l[[a^{\Delta t?}(u) \text{ then } P_1 \text{ else } P_2]] \{q_1 \vee q_2\} \end{array}$$

According to the algebraic law **move-2** which has been introduced in Sect. 4, we know that the move construct $l[[go^{\Delta t}l' \text{ then } P]]$ equals to $\#t \rightarrow l'[[P]]$, where $\#t$ means delaying t time units. Thus, we have the following move rule:

Rule 14. Move

$$\begin{array}{l} \{p[time + t/time] \wedge time < \infty\} \#t \{p\}, \{p\} l'[[P]] \{q\} \\ \hline \{p[time + t/time] \wedge time < \infty\} \#t \rightarrow l'[[P]] \{q\} \end{array}$$

For the move rule, it first delays t time units, leading to assertion p , after which process P located at location l' is executed leading to assertion q .

5.4. Proof rules for parallel composition

The proof rule for parallel composition has the following general form where we use a combinator $Comb$ to combine two assertions. The parallel composition $l_1[[P_1]] \mid l_2[[P_2]]$ performs process P_1 from location l_1 and process P_2 from l_2 running in parallel, where l_1 and l_2 can be the same or different.

Rule 15. Parallel Composition

$$\begin{array}{l} \{p_1\} l_1[[P_1]] \{q_1\}, \{p_2\} l_2[[P_2]] \{q_2\}, \\ Comb(q_1, q_2) \rightarrow q \\ \hline \{p_1 \wedge p_2\} l_1[[P_1]] \mid l_2[[P_2]] \{q\} \end{array}$$

where we have the following assumptions:

- $Var(l_1[[P_1]]) \cap Var(l_2[[P_2]]) = \emptyset$ for parallel composition $l_1[[P_1]] \mid l_2[[P_2]]$.
- $Var(q_i) \subseteq Var(l_i[[P_i]])$ and $Chan(q_i) \subseteq Chan(l_i[[P_i]])$, for $i = 1, 2$.

We consider the following possibilities for $Comb$:

1. If $time$ does not occur in q_1 and q_2 , then we have the following definition:

$$Comb(q_1, q_2) \equiv q_1 \wedge q_2.$$

2. There is no doubt that the termination times of $l_1[[P_1]]$ and $l_2[[P_2]]$ may be different, that is to say that the values of $time$ in assertions q_1 and q_2 may be different. In order to obtain a general rule, we substitute logical variable t_1 and t_2 for $time$ in q_1 and q_2 respectively. Then the termination time of the parallel composition $l_1[[P_1]] \mid l_2[[P_2]]$ is the maximum between t_1 and t_2 .

$$Comb(q_1, q_2) \equiv q_1[t_1/time] \wedge q_2[t_2/time] \wedge time = \max(t_1, t_2).$$

The communication rule expresses that an output action, from location l , succeeds in sending the message v over channel a to an input action from location l without any time delay after the start point t_0 , leading to the assertions p_1 and p_2 respectively. After that P_1 and P_2 , which are both located at location l , run in parallel leading to assertion q .

Rule 16. Communication

$$\frac{\begin{array}{l} (p \wedge \text{time} < \infty)[t_0/\text{time}] \wedge (a!@l, v) \text{ at } t_0 \wedge \text{time} = t_0 \rightarrow p_1 \\ (p \wedge \text{time} < \infty)[t_0/\text{time}] \wedge (a?@l, v) \text{ at } t_0 \wedge \text{time} = t_0 \rightarrow p_2[v/u] \\ \{p_1 \wedge p_2\} l[[P_1]] \mid l[[P_2]] \{q\} \end{array}}{\{p \wedge \text{time} < \infty\} l[[P]] \mid l[[Q]] \{q\}}$$

where, $P = a^{\Delta t_1}!(v)$ then P_1 else Q_1 ,

$Q = a^{\Delta t_2}?(u)$ then P_2 else Q_2 .

Example 5.1 Let us consider the network $N_1 \mid N_2$ in Example 3.2 again, where

$$N_1 = l_1[[go^{\Delta 3}b_2 \text{ then } a^{\Delta 2}!(v_1) \text{ then } b^{\Delta 3}?(u_2) \text{ else } 0]],$$

$$N_2 = l_2[[a^{\Delta 6}?(u_1) \text{ then } b^{\Delta 2}!(v_2) \text{ else } 0]].$$

And we have the following shorthand notations:

$$b^{\Delta 3}?(u_2) \text{ stands for } b^{\Delta 3}?(u_2) \text{ then } 0 \text{ else } 0,$$

$$b^{\Delta 2}!(v_2) \text{ stands for } b^{\Delta 2}!(v_2) \text{ then } 0 \text{ else } 0.$$

Assume that the activated time of N_1 and N_2 is at 0. And according to the algebraic law **para-6** in Sect. 4, we can rewrite $N_1 \mid N_2$ as follows:

$$N_1 \mid N_2 = \#3 \rightarrow N_3, \text{ where}$$

$$N_3 = l_2[[a^{\Delta 2}!(v_1) \text{ then } b^{\Delta 3}?(u_2) \text{ else } 0]] \mid l_2[[a^{\Delta 3}?(u_1) \text{ then } b^{\Delta 2}!(v_2) \text{ else } 0]]$$

We prove the following correctness formula using the proof system which we have introduced:

$$\{\text{time} = 0\} N_1 \mid N_2 \{\text{time} = 3\}$$

To this end we apply the move rule, then we have

$$\frac{\{\text{time} = 0\} \#3 \{\text{time} = 3\}, \{\text{time} = 3\} N_3 \{q\}}{\{\text{time} = 0\} \#3 \rightarrow N_3 \{q\}}$$

Now we should prove that the assertion q in the correctness formula $\{\text{time} = 3\} N_3 \{q\}$ equals to or implies the desired assertion $\{\text{time} = 3\}$. To this end we apply the communication rule, then we can obtain

$$t_0 = 3 \wedge (a!@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3 \rightarrow (a!@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3$$

$$t_1 = 3 \wedge (a?@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3 \rightarrow (a?@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3$$

$$\{(a!@l_2, v_1) \text{ at } 3 \wedge (a?@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3\} l_2[[P_1]] \mid l_2[[P_2]] \{q\}$$

$$\{\text{time} = 3\} N_3 \{q\}$$

where, $P_1 = b^{\Delta 3}?(u_2)$ then 0 else 0,

$P_2 = b^{\Delta 2}!(v_2)$ then 0 else 0.

Since $(a!@l_2, v_1) \text{ at } 3 \wedge (a?@l_2, v_1) \text{ at } 3 \wedge \text{time} = 3 \rightarrow \text{time} = 3$, we then obtain the correctness formula:

$$\{time = 3\} b_2[[P_1]] \mid b_2[[P_2]] \{q\}$$

In order to obtain the assertion q in the above correctness formula, we apply the communication rule again, then we have

$$\frac{\begin{array}{l} t_0 = 3 \wedge (b?@b_2, v_2) \mathbf{at} 3 \wedge time = 3 \rightarrow (b?@b_2, v_2) \mathbf{at} 3 \wedge time = 3 \\ t_1 = 3 \wedge (b!@b_2, v_2) \mathbf{at} 3 \wedge time = 3 \rightarrow (b!@b_2, v_2) \mathbf{at} 3 \wedge time = 3 \\ \{(b?@b_2, v_2) \mathbf{at} 3 \wedge (b!@b_2, v_2) \mathbf{at} 3 \wedge time = 3\} 0 \mid 0 \{q\} \end{array}}{\{time = 3\} b_2[[P_1]] \mid b_2[[P_2]] \{q\}}$$

Since $(b?@b_2, v_2) \mathbf{at} 3 \wedge (b!@b_2, v_2) \mathbf{at} 3 \wedge time = 3 \rightarrow time = 3$, we then obtain the correctness formula:

$$\{time = 3\} 0 \mid 0 \{q\}$$

By the axiom 0, we have

$$\{time = 3\} 0 \{time = 3\}$$

Finally, by the parallel composition rule, we obtain:

$$\frac{\begin{array}{l} \{time = 3\} 0 \{time = 3\}, \{time = 3\} 0 \{time = 3\}, \\ Comb(time = 3, time = 3) \rightarrow time = 3 \end{array}}{\{time = 3\} 0 \mid 0 \{time = 3\}}$$

We now get the desired result, that is, the correctness formula $\{time = 0\} N_1 \mid N_2 \{time = 3\}$ holds. \square

6. Related work

In recent years, some work has been done to explore the formal modeling and analysis of mobile systems. Many concepts fundamental to mobile systems have been represented implicitly in π -calculus in [Mil93, Mil99]. The *ambient* calculus has been introduced [CG00], where crossing of boundaries is used to represent process mobility. In [Lak05], Lakos has presented a Petri Net formalism in which the mobility and the interplay between connectivity and locality can be investigated. The Petri Net formalism proposed in [Lak05] has been extended to Modular Petri Net [CP95, CP00] by adding the notion of nested modules called *locations*. And in [Lak09], Lakos has used the Petri Net formalism to model and analyze mobile IP, which is an Internet standard catering for mobile nodes using IP version 4 address. In addition, Lakos also has implemented the model by using the modular analysis tool Maria [Mäk02].

In [MT08], Ma et al. have proposed an extended version of elementary object system (EOS) [Val98], called EEOS. Based on the extended EOS (EEOS), they have explored a formal model, which is used to model and analyze a generic secure mobile-agent system. The model supports both strong mobility and secure mobility of a mobile agent. In [BSB11], Braghin et al. have presented an approach for the modeling and automated verification of mobile systems, which supports exhaustive analysis of security policies. In their paper, they used labeled Kripke structures (LKSs) to define the mobile system semantics. And the approach proposed in [BSB11] can model some essential features of mobile systems, including thread locations, location distribution and thread moving operations.

The calculus and approaches introduced above do not consider the time constraints, and the time-related aspects of process migration and interaction have been studied in [CK11b, AC13, CKS15, AC15b]. In [CK11b], Ciobanu et al. have first introduced a calculus named TiMo (Timed Mobility) in which the dynamic evolution of the whole system is based on local clocks. rTiMo [AC13] is a real-time version of TiMo [CK11b] and the model of time is based on a global clock in rTiMo calculus. In rTiMo, the processes can move between different locations and communicate locally with other processes. And the migration and communication are controlled by using the real-time constraints. Aman et al. have used the calculus rTiMo to model critical systems in [AC15a].

In [AC13], the operational semantics of rTiMo has been investigated. This paper studies the denotational semantics and algebraic properties of rTiMo via the concept of Unifying Theories of Programming (abbreviated as UTP) [HH98]. UTP covers wide areas of fundamental theories of programs in a formalized style and acts as a consistent basis for the principles of programming language.

The UTP approach has been successfully applied in investigating the semantics and algebraic laws of a variety of programming languages [MM05, DGJP04, HSM97, HHZ⁺15, Zhu05, Shi09, SZL⁺18]. Zhu has investigated the operational, denotational and algebraic semantics for Multithreaded Discrete Event Simulation Language (MDESL) in his thesis [Zhu05]. MDESL is a Verilog-like language [Gol96, Gor95], which has real-time and shared variable features. Shi has used the UTP approach to study the denotational semantics and algebraic semantics of CSP# [SLDC09] in her thesis [Shi09]. The CSP# is a CSP-like language and it directly supports shared variables which are not available in CSP (Communicating Sequential Processes) [Hoa85]. Woodcock et al. have applied UTP to study the safety critical Java memory models [CWW13]. He has proposed a new methodology for further studying the UTP theory [He16].

In addition, in this paper, we also investigate the deductive semantics for rTiMo based on Hoare Logic [AdBO09, Hoo94, Hoo91]. The deductive semantics provides a set of proof rules, which can be used to verify program correctness and general properties. Many approaches based on Hoare Logic have been presented and widely used in a variety of domain, ranging from reasoning about actions with loop [HZ16], to proving differential privacy [BGA⁺14], to linear systems [AMO13], to BPEL-like programs [LQQ08] and to dynamic networks [dB02].

7. Conclusion and future work

rTiMo is a real-time version of TiMo, which is a process algebra for mobile distributed systems. In this paper, we have studied the denotational semantics for rTiMo via the concept of UTP [HH98]. Based on the formalized model, a set of algebraic laws have been investigated, especially the algebraic laws which can describe the time-related features of rTiMo. In order to deal with the parallel expansion laws, we have introduced the concept of guarded choice. From a set of parallel expansion laws, we can see every parallel program can be converted to the guarded choice form. In addition, we also provide a set of proof rules in order to verify correctness and real-time properties of the real-time programs.

Recently, Hoare has proposed the challenging research topic for studying the semantics linking where the starting point is from the algebra semantics [Hoa13, HvS12, HvSM⁺16]. Hoare and He have studied the derivation of operational semantics from the algebraic semantics [HH98, HH93]. Zhu et al. studied the semantics linking theory for Verilog, System, PTSC (integrating Probability, Time and Shared-Variable Concurrency) and Web transaction [ZHB08, ZHQB15, ZYH⁺12, ZHLB11]. For the future work, we also want to explore the linking theory of the semantics for rTiMo. It is challenging for us to apply the concept of head normal form via the achieved algebraic laws of rTiMo.

Hoare also proposed a method to relate the algebra to the proof rules of the axiomatic semantics [Hoa13]. This gives us another challenging research plan for studying the semantics linking starting from the axiomatic semantics for rTiMo. It is challenging to give the translation between the algebraic semantics and axiomatic semantics for rTiMo.

For the future work, we will also consider the implementation of the Hoare-style reasoning system for rTiMo in proof assistant Isabelle/HOL [Pau94, vO01, ZZW⁺13] or any other theorem provers [FGH⁺14, QHL⁺14].

References

- [AC13] Aman B, Ciobanu G (2013) Real-time migration properties of rtimo verified in Uppaal. In: 11th international conference, SEFM 2013 software engineering and formal methods, Madrid, Spain, September 25–27, 2013, proceedings, pp. 31–45
- [AC15a] Aman B, Ciobanu G (2015) Timed mobility and timed communication for critical systems. In: Formal methods for industrial critical systems-20th international workshop, FMICS 2015, Oslo, Norway, June 22–23, 2015 proceedings, pp. 146–161
- [AC15b] Aman B, Ciobanu G (2015) Verification of bounded real-time distributed systems with mobility. In: Proceedings of the 9th workshop on verification and evaluation of computer and communication systems, VECoS 2015, Bucharest, Romania, September 10–11, 2015, pp 109–120
- [AdBO09] Apt KR, de Boer FS, Olderog ER (2009) Verification of sequential and concurrent programs. Texts in computer science. Springer
- [AMO13] Arthan R, Martin U, Oliva P (2013) A Hoare Logic for linear systems. Formal Asp Comput 25(3):345–363

- [BGA⁺14] Barthe G, Gaboardi M, Arias EJG, Hsu J, Kunz C, Strub PY (2014) Proving differential privacy in Hoare logic. In: IEEE 27th computer security foundations symposium, CSF 2014, Vienna, Austria, 19–22 July, 2014, pp. 411–424
- [BSB11] Braghin C, Sharygina N, Barone-Adesi K (2011) A model checking-based approach for security policy verification of mobile systems. *Formal Asp Comput* 23(5):627–648
- [CG00] Cardelli L, Gordon AD (2000) Mobile ambients. *Theor Comput Sci* 240(1):177–213
- [CJ12] Ciobanu G, Juravle C (2012) Flexible software architecture and language for mobile agents. *Concurrency and computation: practice and experience* 24(6):559–571
- [CK11a] Ciobanu G, Koutny M (2011) Timed migration and interaction with access permissions. In: FM 2011: Formal methods-17th international symposium on formal methods, Limerick, Ireland, June 20-24, 2011, proceedings, pp 293–307
- [CK11b] Ciobanu G, Koutny M (2011) Timed mobility in process algebra and Petri nets. *J Log Algebr Program* 80(7):377–391
- [CK15] Ciobanu G, Koutny M (2015) Pertimo: A model of spatial migration with safe access permissions. *Comput J* 58(5):1041–1060
- [CKS15] Ciobanu G, Koutny M, Steggles LJ (2015) Strategy based semantics for mobility with time and access permissions. *Formal Asp Comput* 27(3):525–549
- [CP95] Christensen S, Petrucci L (1995) Modular state space analysis of coloured Petri nets. In: 16th international conference application and theory of petri nets 1995, Turin, Italy, June 26–30, 1995, proceedings, pp 201–217
- [CP00] Christensen S, Petrucci L (2000) Modular analysis of Petri nets. *Comput J* 43(3):224–242
- [CWW13] Cavalcanti A, Wellings AJ, Woodcock J (2013) The safety-critical java memory model formalised. *Formal Asp Comput* 25(1):37–57
- [dB02] de Boer Frank S (2002) A Hoare logic for dynamic networks of asynchronously communicating deterministic processes. *Theor Comput Sci* 274(1-2):3–41
- [DCS10] Duran A, Cavalcanti A, Sampaio A (2010) An algebraic approach to the design of compilers for object-oriented languages. *Formal Asp Comput* 22(5):489–535
- [DGJP04] Desharnais J, Gupta V, Jagadeesan R, Panangaden P (2004) Metrics for labelled markov processes. *Theor Comput Sci* 318(3):323–354
- [FGH⁺14] Ferreira JF, Gherghina C, He G, Qin S, Chin W-N (2014) Automated verification of the FreeRTOS scheduler in Hip/Sleek. *STTT* 16(4):381–397
- [Gol96] Golze U (1996) VLSI chip design with the hardware description language VERILOG—an introduction based on a large RISC processor design. Springer, Berlin
- [Gor95] Gordon Michael JC (1995) The semantic challenge of Verilog HDL. In: Proceedings, 10th annual IEEE symposium on logic in computer science, San Diego, California, USA, June 26-29, 1995, pp. 136–145
- [He94] He J (1994) Provably correct systems: modelling of communication languages and design of optimized compilers. The McGraw-Hill international series in software engineering
- [He16] He J (2016) A new roadmap for linking theories of programming. In: Unifying theories of programming-6th international symposium, UTP 2016, Reykjavik, Iceland, June 4–5, 2016, Revised Selected Papers, pp 26–43
- [Hen88] Hennessy M (1988) Algebraic theory of processes. MIT Press series in the foundations of computing. MIT Press
- [HH93] He J, Hoare CAR (1993) From algebra to operational semantics. *Inf Process Lett* 45(2):75–80
- [HH98] Hoare CAR, He J (1998) Unifying Theories of Programming. Prentice Hall International Series in Computer Science
- [HHH⁺87] Hoare CAR, Hayes IJ, He J, Morgan C, Roscoe AW, Sanders JW, Sørensen IH, Spivey JM, Sufrin B (1987) Laws of programming. *Commun ACM* 30(8):672–686
- [HHS93] Hoare CAR, He J, Sampaio A (1993) Normal form approach to compiler design. *Acta Inf* 30(8):701–739
- [HHZ⁺15] Huang Y, He J, Zhu H, Zhao Y, Shi J, Qin S (2015) Semantic theories of programs with nested interrupts. *Front Comput Sci* 9(3):331–345
- [Hoa69] Hoare CAR (1969) An axiomatic basis for computer programming. *Commun ACM* 12(10):576–580
- [Hoa85] Hoare CAR (1985) Communicating sequential processes. Prentice-Hall,
- [Hoa13] Hoare T (2013) Unifying semantics for concurrent programming. In: Computation, logic, games, and quantum foundations. the many facets of samson abramsky-essays dedicated to samson abramsky on the occasion of his 60th Birthday, pp 139–149
- [Hoo91] Hooman J (1991) Compositional verification of real-time systems using extended hoare triples. In: Real-time: theory in practice, REX workshop, Mook, The Netherlands, June 3–7, 1991, proceedings, pp 252–290
- [Hoo94] Hooman J (1994) Extending Hoare Logic to real-time. *Formal Asp Comput* 6(6A):801–826
- [HSM97] He J, Seidel K, McIver A (1997) Probabilistic models for the guarded command language. *Sci Comput Program* 28(2-3):171–192
- [HvS12] Hoare T, van Staden S (2012) In praise of algebra. *Formal Asp Comput* 24(4-6):423–431
- [HvSM⁺16] Hoare T, van Staden S, Möller B, Struth G, Zhu H (2016) Developments in concurrent Kleene algebra. *J Log Algebr Methods Program* 85(4):617–636
- [HZ16] He J, Zhao X (2016) Reasoning about actions with loops via Hoare logic. *Front Comput Sci* 10(5):870–888
- [Lak05] Lakos C (2005) A Petri net view of mobility. In: Formal techniques for networked and distributed systems-FORTE 2005, 25th IFIP WG 6.1 international conference, Taipei, Taiwan, October 2–5, 2005, proceedings, pp 174–188
- [Lak09] Lakos C (2009) Modelling mobile IP with mobile Petri nets. *Transactions on petri nets and other models of concurrency III*. Lecture notes in computer science 5800, Springer 2009, ISBN 978-3-642-04854-8, 3:127–158
- [LQQ08] Luo C, Qin S, Qiu Z (2008) Verifying BPEL-like programs with Hoare Logic. *Front Comput Sci China* 2(4):344–356
- [Mäk02] Mäkelä M (2002) Maria: modular reachability analyser for algebraic system nets. In: Applications and theory of Petri nets 2002, 23rd international conference, ICATPN 2002, Adelaide, Australia, June 24-30, 2002, proceedings, pp 434–444
- [Mil80] Milner R (1980) A calculus of communicating systems (Lecture notes in computer science), vol 92. Springer
- [Mil93] Milner R (1993) Elements of interaction-turing award lecture. *Commun ACM*, 36(1):78–89
- [Mil99] Milner R (1999) Communicating and mobile systems-the Pi-calculus. Cambridge University Press, Cambridge
- [MM05] McIver A, Morgan C (2005) Abstraction and refinement in probabilistic systems. *SIGMETRICS Perform Eval Rev*, 32(4):41–47

- [MT08] Ma L, Tsai JJP (2008) Formal modeling and analysis of a secure mobile-agent system. *IEEE Trans Syst Man and Cyber Part A* 38(1):180–196
- [O’H07] O’Hearn PW (2007) Resources, concurrency, and local reasoning. *Theor Comput Sci* 375(1-3):271–307
- [Pau94] Paulson LC (1994) Isabelle-A Generic theorem prover (with a contribution by T. Nipkow), volume 828 of *Lecture notes in computer science*. Springer
- [Plo04] Plotkin GD (2004) A structural approach to operational semantics. *J Log Algebr Program* 60-61:17–139
- [Pra90] Vaughan RP (1990) Action logic and pure induction. In: *Logics in AI, European workshop, JELIA ’90*, Amsterdam, The Netherlands, September 10–14, 1990, proceedings, pp 97–120
- [QHL⁺14] Qin S, He G, Luo C, Chin W-N, Yang H (2014) Automatically refining partial specifications for heap-manipulating programs. *Sci Comput Program* 82:56–76
- [Shi09] Shi L (2009) Comparative studies, formal semantics and PVS encoding of CSP#. PhD thesis, East China Normal University, China
- [SLDC09] Sun J, Liu Y, Dong JS, Chen C (2009) Integrating specification and programs for system modeling and verification. In *TASE 2009, third IEEE international symposium on theoretical aspects of software engineering*, 29–31 July 2009, Tianjin, China, pp 127–135
- [Sto79] Stoy JE (1979) Foundations of denotational semantics. In: *Abstract software specifications, 1979 Copenhagen Winter School*, January 22 –February 2, 1979, proceedings, pp 43–99
- [SZL⁺18] Shi L, Zhao Y, Liu Y, Sun J, Dong JS, Qin S (2018) A UTP semantics for communicating processes with shared variables and its formal encoding in PVS. *Formal Asp Comput*
- [Tar55] Tarski A (1955) A lattice-theoretical fixpoint theorem and its applications. *Pac J Math*, 5(2):285–309
- [Val98] Valk R (1998) Petri nets as token objects: An introduction to elementary object nets. In: *19th international conference application and theory of petri nets 1998 ICATPN ’98*, Lisbon, Portugal, June 22-26, 1998, proceedings, pp 1–25
- [vO01] von Oheimb D (2001) Hoare logic for java in Isabelle/HOL. *Concurr Comput Pract Exp*. 13(13):1173–1214
- [Wat91] Watt DA (1991) *Programming language syntax and semantics*. Prentice Hall International series in computer science. Prentice Hall
- [XX16] Xie W, Xiang S (2016) UTP semantics for rTiMo. In *Unifying theories of programming-6th international symposium, UTP 2016*, Reykjavik, Iceland, June 4–5, 2016, Revised Selected Papers, pp. 176–196
- [ZHB08] Zhu H, He J, Bowen JP (2008) From algebraic semantics to denotational semantics for Verilog. *ISSE* 4(4):341–360
- [ZHLB11] Zhu H, He J, Li J, Bowen JP (2011) Algebraic approach to linking the semantics of web services. *ISSE* 7(3):209–224
- [ZHQB15] Zhu H, He J, Qin S, Brooke PJ (2015) Denotational semantics and its algebraic derivation for an event-driven system-level language. *Formal Asp Comput* 27(1):133–166
- [Zhu05] Zhu H (2005) Linking the semantics of a multithreaded discrete event simulation language. PhD thesis, London South Bank University UK
- [ZYH⁺12] Zhu H, Yang F, He J, Bowen JP, Sanders JW, Qin S (2012) Linking operational semantics and algebraic semantics for a probabilistic timed shared-variable language. *J Log Algebr Program* 81(1):2–25
- [ZZW⁺13] Zou L, Zhan N, Wang S, Fränzle M, Qin S (2013) Verifying Simulink diagrams via a hybrid hoare logic prover. In: *Proceedings of the international conference on embedded software, EMSOFT 2013*, Montreal, QC, Canada, September 29–Oct. 4, 2013, pp 9:1–9:10

Received 25 April 2017

Accepted in revised form 24 July 2018 by Jin Song Dong

Published online 10 August 2018