UTP Semantics for BigrTiMo

Wanling Xie[1], Huibiao Zhu[1]⋆, Shengchao Qin[2]

[1] Shanghai Key Laboratory of Trustworthy Computing,
School of Computer Science and Software Engineering,
East China Normal University, Shanghai, China
[2] School of Computing, University of Teesside,
Middlesbrough, Tees Valley, TS1 3BA, UK

Abstract. BigrTiMo [1], a process algebra that combines the rTiMo calculus [2] and the Bigraph model [3], is capable of specifying a rich variety of properties for structure-aware mobile systems. Compared with rTiMo, our BigrTiMo calculus can specify not only time, mobility and local communication, but also remote communication. In this paper, we study the semantic foundation of this highly expressive modelling language and propose a denotational semantic model for it based on Hoare and He's Unifying Theories of Programming (UTP) [4]. Compared to the standard UTP model, in addition to the communication, the novelty of the proposed UTP model in this paper covers time, location and global shared variable. Moreover, we give an example to show the contribution of BigrTiMo and illustrate how to use our semantic model and the trace-merging definition proposed in our paper under this example. We also demonstrate the proofs of some algebraic laws proposed in [1] based on our denotational semantics.

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. [5] have first introduced a process algebra called TiMo (Timed Mobility) model for mobile systems, where it is possible to add time constraints to the basic actions (i.e., migration action and communication action) and the model of time is based on local clocks. Aman et al. [2] have extended TiMo 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.

The above calculi only can model the local communication (the two communication components should be at the same location), however, in real applications, with the development of the internet, the two communication parties may not only communicate locally, but also communicate remotely (the two components

⋆ Corresponding Author. Email: hbzhu@sei.ecnu.edu.cn

can be at the different locations). In order to model the remote communications, we have extended rTiMo into BigrTiMo [1] by introducing a Bigraph model [3].

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) [6]. In [1], we have presented the operational semantics and algebraic semantics for BigrTiMo. And in this paper, we will investigate the denotational semantics which provides the mathematical meanings to programs. The approach of denotational semantics is under a purely mathematical basis, thus, it is more abstract. Compared with operational semantics, denotational semantics expresses *what a program does*. Our approach is based on Unifying Theories of Programming (UTP) proposed by Hoare and He in 1998 [4]. Compared to the standard UTP model [4], in addition to communication, the novelty of the UTP model in this paper covers time, location and global shared variable. Moreover, we give an example to show the contribution of BigrTiMo and illustrate how to use our semantic model and the trace-merging definition proposed in our paper under this example. We also demonstrate the proofs of some algebraic laws proposed in [1] based on our denotational semantics.

The remainder of this paper is organized as follows. Section 2 gives an introduction to the BigrTiMo calculus. In Section 3, we first present the semantic model and healthiness conditions that a BigrTiMo program should satisfy. We then explore the denotational semantics of BigrTiMo. In Section 4, we demonstrate the proofs of some algebraic laws based on the denotational semantics. Section 5 concludes the paper and discusses some possible future work.

2 BigrTiMo

In this section, we introduce BigrTiMo. In Section 2.1, we give a brief review of the bigraph. In Section 2.2, we introduce the syntax of BigrTiMo and give an example to show the contribution of our BigrTiMo calculus.

2.1 Review of Bigraph

A bigraph is a mathematical structure with two graphs, including a *placing graph* and a *linking graph* [3]. The placing graph is a forest which is used to model nested locality of components and the linking graph is a hypergraph that represents connectivity between components. Fig. 1 illustrates an example of a bigraph. Fig. 2 presents the corresponding placing and linking graphs of Fig. 1.

The encompassing rectangle represents a *region* and the grey rectangles are used to represent *holes*. Region and hole are the root and leaf node respectively in the placing graph, and enable the composition of placing graphs, e.g., a hole of a bigraph can be replaced by a region of another bigraph with the aid of composition operator defined in [3]. *Ports* are represented as black dots on the node, and are used to connect the edges or names, i.e., the node m_1 has two ports which connect to the edge e and the outer name y. The edge e and inner

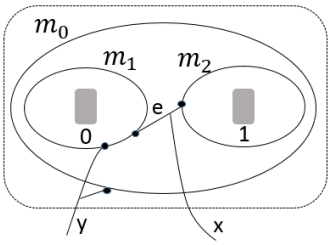

Fig. 1. Bigraph

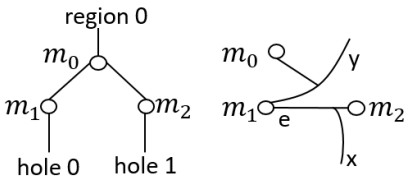

Fig. 2. Placing and linking graphs

name x and outer name y are contained in the linking graph. We can also merge inner names and outer names using the bigraph composition operator.

We below give the definition of a bigraph.

Definition 1. *A **bigraph** is a 5-tuple* $(V, E, nupt, prnt, link)$ *where,*

- V *is the set of node identifiers and* E *is the set of edge identifiers.*
- $nupt : V \to \mathbb{N}$ *is a map that assigns their numbers of ports (i.e., a natural number) to nodes.*
- $prnt : m \uplus V \to V \uplus n$ *is the parent map which is used to assign a parent (i.e., a node or a region) to the children (i.e., a hole or a node).* $m = \{0, ..., |m|-1\}$ *denotes the set of holes and* $n = \{0, ..., |n| - 1\}$ *denotes the set of regions. The symbol* $\uplus$ *stands for the disjoint union of sets.*
- $link : X \uplus P \to E \uplus Y$ *is the link map which assigns edges and outer names to inner names and ports.* X *and* Y *denote the set of inner names and the set of outer names respectively.* P *denotes the set of ports of the bigraph and is formalized as* $P = \{(l, i) | i \in \{0, 1, ..., nupt(l) - 1\}\}$*, where* l *is a node in the set* V *in a bigraph. For convenience, we introduce a map* $pts : V \to \mathbb{P}(\mathbb{N})$ *that takes a node and returns the set of ports of that node.*

The bigraph stands for a specific snapshot of the world but there is no information on how it can evolve to another bigraph. Bigraph Reaction Rules (BRRs) are defined in [3] to create dynamics of bigraphs. A BRR is the form of $R \to R'$ where R and R' are bigraphs called *redex* and *reactum*, respectively. Let $r = R \to R'$ be a BRR and B a bigraph. In order to execute rule r in B we should first decompose B into $C \circ R \circ d$ where C stands for the context and d stands for the parameters inside the holes of R. We compose C with R' and with d to obtain the result B', e.g., $B = C \circ R \circ d \Rightarrow B' = C \circ R' \circ d$.

Consider a BRR **MOVE(pc0, room2)** that moves a **pc0** node from its current location to a **room2** node. Fig. 3 describes the context C and parameters d where the rule is applied. A bigraph B changed to B' is illustrated at the top of Fig. 3. At the bottom of this figure, we give the decomposition of each bigraph in the context C, redex R, reactum R' and parameters d.

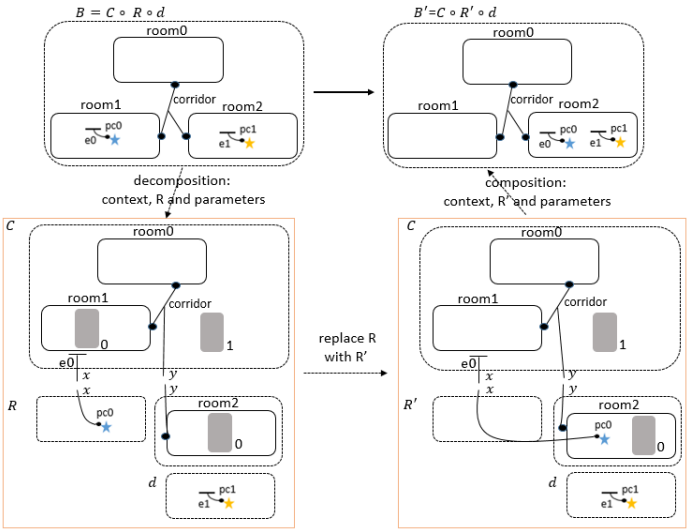

Fig. 3. Example of application of **MOVE** reaction rule

2.2 The Syntax of BigrTiMo

We have presented the BigrTiMo calculus in [1]. Compared with rTiMo, our BigrTiMo can model not only the location of components but also the connectivity of components. Thus, a BigrTiMo process not only can communicate locally with other processes (like an rTiMo process), but also can communicate remotely with other processes (if the locations of the two components are connected, i.e., they share a communication link in the bigraph). In addition, a BigrTiMo process can migrate from one location to another location (if the desired location is contained in the bigraph and the current location of the process is connected to the desired location), and perform the bigraph reaction rules to update the bigraph.

The syntax of BigrTiMo is given in Table 1. In BigrTiMo, actions are controlled by using real-time constraints. Timeouts are specified by a superscript $\triangle t$. The communication channels are point-point, i.e. each connecting two processes, and synchronous. A synchronous channel with buffer size 0 sends/recieves messages synchronously and a communication $a.v$ takes place when both actions $a!\langle v\rangle$ and $a?(v)$ are enabled simultaneously. We first introduce the process parts:

1. nil denotes the process that terminates without taking any time.
2. $a^{\triangle t}!\langle v\rangle$ *then* P *else* Q stands for an output process. When the message v is sent via channel a 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.
3. $a^{\triangle t}?(u)$ *then* P *else* Q indicates an input process. When the process receives a message within t time units, the next process is P. If the input action does

not occur before the deadline t, it gives up and switches to the process Q. The input process binds the variable u within P (but not within Q).

4. $go^{\triangle t} l$ *then* P *else* Q denotes a migration process. If the migration action happens successfully after delaying t time units, then the next process is P located at location l. Otherwise, it switches to the alternative process Q whose location does not change.
5. $control^{\triangle t}(r)$ *then* P is an update process where r is a BRR which is performed to update the state of the shared bigraph. After delaying t time units, the update action takes place and the next process is P.
6. $P \mid\mid Q$ stands for parallel composition.
7. $l[[P]]$ specifies a process P running at location l.

Table 1. BigrTiMo Syntax

Process	$P, Q ::= nil$	(termination)
	$\mid a^{\triangle t}!\langle v\rangle$ *then* P *else* Q	(output)
	$\mid a^{\triangle t}?(u)$ *then* P *else* Q	(input)
	$\mid go^{\triangle t} l$ *then* P *else* Q	(move)
	$\mid control^{\triangle t}(r)$ *then* P	(update)
	$\mid P \mid\mid Q$	(parallel composition)
Located process	$L ::= l[[P]]$	
Network	$N ::= 0 \mid L \mid L \mid\mid N$	
Configuration	$G ::= empty \mid \langle N, B\rangle \mid \langle N, B\rangle \mid\mid G$	

We next introduce the network and configuration parts. 0 denotes an empty network. A network can be a located process L or can be built via its component $L \mid\mid N$. *empty* denotes an empty configuration. A BigrTiMo configuration is a tuple $\langle N, B\rangle$ denoted by G where N is a BigrTiMo network, B is a shared bigraph and the locations in N are all contained in the set of the node identifiers in the bigraph B. A configuration also can be built via component $\langle N, B\rangle \mid\mid G$.

The shared bigraph in BigrTiMo is globally accessible and it can be read and written by different actions. In order to ensure the consistency of the shared bigraph, it can only be updated in sequential programs (i.e., $G1; G2$ denotes the behavior that runs $G1$ and $G2$ sequentially). Moreover, when programs execute atomically, sequential programs from different configurations are not allowed to execute simultaneously.

In order to support our algebraic expansion laws proposed in [1], we have presented three types of guarded choice. And we can convert every BigrTiMo program into the guarded choice form.

1. **Instantaneous Guarded Choice**.
   The notation $\langle [\![\{g_1 \rightarrow N_1, ..., g_n \rightarrow N_n\}, B\rangle$ stands for an instantaneous guarded choice, which executes its guard g_i under the bigraph B initially

and then performs the corresponding program $\langle N_i, B\rangle$ afterward. The guard g_i is an instantaneous guard which means that it takes place without any time delay, and it can be a communication guard or an event guard. A communication guard can be expressed as as $a!\langle v\rangle@l$, $a?(u)@l$ or $a.[v/u]@(l,l')$, where $a!\langle v\rangle@l$ (or $a?(u)@l$) indicates that the output (or input) action happens at location l, and $a.[v/u]@(l,l')$ denotes that the communication occurs where one communication end is from the location l and the other is from the location l', and the variable u is replaced by the message v. The event guards are $go(l')@l$ and $control(r)@l$ which represent that the migration action and update action happen at the location l, respectively.

2. **Delay Guarded Choice**.
   $\langle \#t \to N, B\rangle$ is a delay guarded choice and $\#t$ means delaying t time units.
3. **Hybrid Guarded Choice**.
   The hybrid guarded choice has the following form where the notation $\oplus$ denotes the disjointness of timed behaviors.

$$\begin{aligned}&\langle [\![_{i\in I}\{g_i \to N_i\}, B\rangle\\ \oplus\ &\exists t' \in (0...t) \bullet (\langle \#t' \to [\![_{i\in I}\{g_i \to N_i'\}, B\rangle)\\ \oplus\ &\langle \#t \to N', B\rangle\end{aligned}$$

Example 1. Consider users with smartphones or personal computers communicating with each other. A city may contain housing area, office area and subway station and so on. Some areas may contain wireless hotspots which can be connected to the internet. If a personal computer or a smartphone is contained in a wireless hotspot it can connect to it. A user can walk from one location to anther location if the two locations are connected.

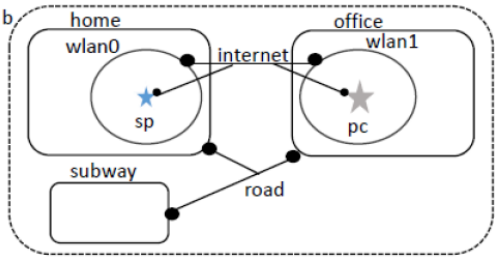

Fig. 4. The example of bigraph b

Fig. 4 depicts a bigraph city named b. This figure shows a city with three areas, namely *home*, $office$ and *subway*. All areas are connected with a link named *road* which models physical adjacency between the corresponding physical locations. A user can walk from *home* to *subway* (the two locations are connected with a link *road*). Some areas contain nodes $wlan0$ and $wlan1$ (white

circles) which model the wireless hotspots, i.e., $home$ contains $wlan0$ and $office$ contains $wlan1$. A smartphone modelled as a node sp (blue star) is contained in $wlan0$. A personal computer modelled as a node pc (grey star) is contained in $wlan1$. The nodes $wlan0$, $wlan1$, sp and pc all share a link named $internet$ modelling the connectivity of the corresponding entities to the internet.

The definition of the bigraph b is showed as below.

$b = (V, E, nupt, prnt, link)$ where:

$V = \{home, office, subway, wlan0, wlan1, sp, pc\}$

$E = \{internet, road\}$

$nupt(l) = 1$ where $l \in V$

$$prnt(l) = \begin{cases} 0, & if\ l \in \{home, office, subway\}; \\ home, & if\ l = wlan0; \\ office, & if\ l = wlan1; \\ wlan0, & if\ l = sp; \\ wlan1, & if\ l = pc. \end{cases}$$

$$link(p) = \begin{cases} internet, & if\ p \in \{(wlan0, 0), (wlan1, 0), (sp, 0), (pc, 0)\}; \\ road, & if\ p \in \{(home, 0), (office, 0), (subway, 0)\}. \end{cases}$$

Consider two BigrTiMo processes $appsf$ and $appbs$ which are hosted at sp and pc respectively. The role of each process is described as below.

- $appsf$ is a process to give instructions to a staff to receive a message (this message is used to inform the staff to go to a office) from a boss, then move to $subway$, move to $office$, connect to the internet.
- $appbs$ is a process to give instructions to a boss to send a message to a staff.

The BigrTiMo syntax of the whole mobile system is:

$\langle sp[[appsf]] \parallel pc[[appbs]], b\rangle$.

The BigrTiMo syntax of two processes is as below:

$$appsf = bs^{\triangle 1}?(u_1)\ then\ (control^{\triangle 3}(r_1)\ then\ control^{\triangle 2}(r_2)\ then\ control^{\triangle 1}(r_3)\ then\ nil)\ else\ nil$$

$$appbs = bs^{\triangle 1}!\langle work\rangle\ then\ nil\ else\ nil$$

where $r_1 = \mathbf{MOVE}(\mathbf{sp}, \mathbf{subway})$, $r_2 = \mathbf{MOVE}(\mathbf{sp}, \mathbf{office})$, $r_3 = \mathbf{CONNECT}(\mathbf{sp}, \mathbf{wlan1})$. □

3 Denotational Semantics of BigrTiMo

In this section, we present the denotational semantics for BigrTiMo. We use $\mathbf{beh}(\langle l[[P]], B\rangle)$ to describe the behavior of a process P running at location l in the given bigraph B after it is activated. In Section 3.1, we give the semantic model for BigrTiMo. In Section 3.2, we explore the behaviors of the basic commands. In Section 3.3, we investigate the behavior of the parallel composition.

3.1 The Semantic Model

In this subsection, the denotational semantic model for BigrTiMo is investigated. Similar to the semantic model for rTiMo [7], here, we also 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. "$st = wait$" 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 program in terms of a trace of snapshots which record the sequence of the actions. In our semantic model, we introduce a variable tr to denote the trace for the sequence of the actions. Inspired by the semantic model for CSP# [8] which covers both communication and shared variables, a snapshot in our semantic model can be expressed as (t, loc, σ, κ) where:

- t indicates the time when the action takes place.
- loc records the locations at which the action takes place. And the form of loc is (l_1, l_2) or a single location l, where (l_1, l_2) means that the two components are located at l_1 and l_2 respectively (i.e., one communication end is from l_1, and the other one is from l_2). For convenience, we have $(l, l) = l$.
- σ denotes the pre-state of the shared bigraph and the action is observed under this state.
- κ denotes the observed action, including inputs/outputs, synchronous communications, migration and update. Thus, κ has the following forms:
  1. for an input/output or a synchronous communication, 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 $\mathbf{Mess}(a.v) = v$.
  2. for a migration action, κ is a desired location.
  3. for an update action, κ is a bigraph σ' which is a post-state recording the final state of the global shared bigraph after the observation. And it is used to record the observation for the sequential programs.

We use the following projections to select the components of a snapshot:

$$\pi_1((t, loc, \sigma, \kappa)) =_{df} t \qquad \pi_2((t, loc, \sigma, \kappa)) =_{df} loc$$
$$\pi_3((t, loc, \sigma, \kappa)) =_{df} \sigma \qquad \pi_4((t, loc, \sigma, \kappa)) =_{df} \kappa$$

In addition to the communication and global shared variable, our calculus has other interesting features, including time constrains and location information. Thus, the observations of a BigrTiMo program can be described by a tuple:

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

– *time* and $time'$ respectively denote the start point and the end point of the time interval over which the observation is recorded. We use δ to represent the length of the time interval.
$$\delta =_{df} (time' - time)$$
In our discrete model, we regard the length of a time interval as a non-negative real number, i.e., δ is considered as a non-negative real number.
– *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.

We use the notations **head**(s) and **tail**(s) to denote the first snapshot of the trace s and the result of removing the first snapshot in the trace s, respectively.

Example 2. Let us consider the configuration in *Example* 1 in Section 2 again, where $N_1 = sp[[appsf]]$, $N_2 = pc[[appbs]]$, $G = \langle N_1 \parallel N_2, b\rangle$.

Fig. 5 shows the execution trace over the respective BRRs, where b is the initial bigraph and $b3$ is the final bigraph when the program terminates.

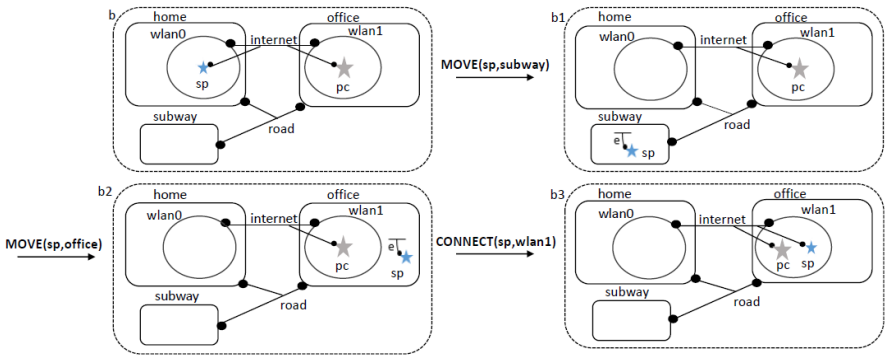

Fig. 5. Execution trace over the respective BRRs

Now we consider the trace of G. Assume that the activated time of G is at 0. According to the syntax of BigrTiMo, we know that the communication action occurs at time 0 (the two locations of the two components are connected by a communication link *internet*). After delaying three time units, the rule r_1 is performed to update the bigraph b into $b1$. After that, the rule r_2 is performed to update $b1$ into $b2$ after delaying two time units. Lastly, after delaying one time unit, the rule r_3 is performed to update $b2$ into $b3$.

A trace of $\langle N_1, b\rangle$ is given as below:

$\langle(0, sp, b, bs.work), (3, sp, b, b1), (5, sp, b1, b2), (6, sp, b2, b3)\rangle$.

A trace of $\langle N_2, b\rangle$ is: $\langle(0, pc, b, bs.work)\rangle$.

Hence, the one trace of G is as follows:

$\langle(0, (sp, pc), b, bs.work), (3, sp, b, b1), (5, sp, b1, b2), (6, sp, b2, b3)\rangle$. □

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 two healthiness conditions that BigrTiMo programs should satisfy, and they are similar to the standard UTP theory [4]. In our semantic model, the variable tr is used to record the execution trace of a program, so it cannot be shortened. 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 a BigrTiMo program behavior must therefore imply this fact. So it satisfies the healthiness condition **R1**.

R1 $P = P \wedge Inv(tr, time)$, where

$Inv(tr, time) =_{df} tr \preceq tr' \ \wedge \ time \leq time'$, which states that tr is a prefix of tr', and $time$ is less than or equal to $time'$.

Because of the requirement for synchronisation, a program may wait for communicating with another program via a specific channel. We take the output command $\langle l[[a^{\triangle t}!\langle v\rangle \ then \ P_1 \ else \ P_2]], B\rangle$ as an example: if process P_1 or P_2 is asked to start in a waiting state of the output action $a^{\triangle t}!\langle v\rangle$, then P_1 or P_2 keeps itself unchanged. And it satisfies the following healthiness condition.

R2 $P = \Pi \ \triangleleft \ st = wait \ \triangleright \ P$

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

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

We denote all healthiness conditions satisfied by the BigrTiMo program using the following H function. And function H is idempotent and monotonic [4].

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

From the definition of H function, we know that $H(X)$ satisfies the healthiness conditions **R1** and **R2**. The H function is used to define the denotational semantics for the BigrTiMo model.

3.2 Denotational Semantics of Basic Commands

We first investigate the denotational semantics of $\langle 0, B\rangle$. It is an empty configuration and its execution state, terminal time and trace all keep unchanged.

$\mathbf{beh}(\langle 0, B\rangle) =_{df} H\left(st' = st \ \wedge \ \delta = 0 \ \wedge \ tr' = tr\right)$

$G1; G2$ denotes the behavior that runs $G1$ and $G2$ sequentially. We now define the sequence operator for our semantic model.

Definition 2. $G1; G2 =_{df}$

$\exists s, t, r \bullet G1[s/st', t/time', r/tr'] \ \wedge \ G2[s/st, t/time, r/tr]$.

The semantics of sequential composition is given as below:

$\mathbf{beh}(G1; G2) =_{df} \mathbf{beh}(G1); \mathbf{beh}(G2)$

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

Instantaneous Guarded Choice. An instantaneous guard can be a communication guard or an event guard. The three types of the communication guard are $a!\langle v\rangle@l$, $a?(u)@l$ and $a.[v/u]@(l, l')$. The event guards are $go(l')@l$ and $control(r)@l$. The semantics of the communication guards is similar to the one in [7]. Due to space limitations, we only take the semantics of $a!\langle v\rangle@l$ as an example. And the semantics of the event guards $go(l')@l$ and $control(r)@l$ is novel in this paper.

$\mathbf{beh}(\langle [\![_{i\in I}\{g_i \to N_i\}, B\rangle) =_{df} \bigvee_{i\in I} \mathbf{beh}(\langle g_i \to N_i, B\rangle)$, where

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

$\mathbf{beh}(\langle a!\langle v\rangle@l \to N, B\rangle) =_{df} \mathbf{beh}(\langle a!\langle v\rangle@l, B\rangle); \mathbf{beh}(\langle N, B\rangle)$

where $\mathbf{beh}(\langle a!\langle v\rangle@l, B\rangle) =_{df} H \begin{pmatrix} st' = completed \ \wedge \ \delta = 0 \ \wedge \\ tr' = tr^\frown\langle(time', l, B, a.v)\rangle \end{pmatrix}$

(2) if $g = go(l')@l$, then

$\mathbf{beh}(\langle go(l')@l \to N, B\rangle) =_{df} \mathbf{beh}(\langle go(l')@l, B\rangle); \mathbf{beh}(\langle N, B\rangle)$

where $\mathbf{beh}(\langle go(l')@l, B\rangle) =_{df} H \begin{pmatrix} st' = completed \ \wedge \ \delta = 0 \ \wedge \\ tr' = tr^\frown\langle(time', l, B, l')\rangle \end{pmatrix}$

(3) if $g = control(r)@l$, then

$\mathbf{beh}(\langle control(r)@l \to N, B\rangle) =_{df} \mathbf{beh}(\langle control(r)@l, B\rangle); \mathbf{beh}(\langle N, B'\rangle)$

where $\mathbf{beh}(\langle control(r)@l, B\rangle) =_{df} H \begin{pmatrix} st' = completed \ \wedge \ \delta = 0 \ \wedge \\ tr' = tr^\frown\langle(time', l, B, B')\rangle \end{pmatrix}$

and $r = R \to R'$, $B = C \circ R \circ d$, $B' = C \circ R' \circ d$.

In the semantic of $\langle control(r)@l \to N, B\rangle$, the rule r is performed to update the bigraph B into B' without any time delay. Thus, $time' = time$ and a snapshot $(time', l, B, B')$ contributed by the update action is attached to the end of the program trace.

Delay Guarded Choice. It consists of only one time delay component.

$\mathbf{beh}(\langle \#t \to N, B\rangle) =_{df} \mathbf{beh}(\langle \#t, B\rangle); \mathbf{beh}(\langle N, B\rangle)$

where $\mathbf{beh}(\langle \#t, B\rangle) =_{df} H \begin{pmatrix} ((st' = wait \ \wedge \ \delta < t) \vee \\ (st' = completed \ \wedge \ \delta = t)) \ \wedge \ tr' = tr \end{pmatrix}$

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

$$\begin{aligned} G = {} & \langle [\![_{i\in I}\{g_i \to N_i\}, B\rangle \\ & \oplus \exists t' \in (0...t) \bullet (\langle \#t' \to [\![_{i\in I}\{g_i \to N_i'\}, B\rangle) \\ & \oplus \langle \#t \to N', B\rangle \end{aligned}$$

and the semantics of G is given below. The three branches are disjoint.

$$\mathbf{beh}(G) =_{df} \begin{pmatrix} \bigvee_{i\in I} \mathbf{beh}(\langle g_i \to N_i, B\rangle) \\ \bigvee \\ \exists t' \in (0...t) \bullet (\mathbf{beh}(\langle \#t', B\rangle); \bigvee_{i\in I} \mathbf{beh}(\langle g_i \to N_i', B\rangle)) \\ \bigvee \\ \mathbf{beh}(\langle \#t, B\rangle); \mathbf{beh}(\langle N', B\rangle) \end{pmatrix}$$

We then investigate the behavior of $\langle l[[go^{\triangle t} l' \ then \ P \ else \ Q]], B\rangle$, it indicates that if the desired location l' is contained in the set of the node identifiers in the bigraph B (denoted by V_B), as well as the location l and the desired location l' are connected in the bigraph B, then the migration action can happen successfully after delaying t time units. If the migration action takes place successfully, then the subsequent behavior of the program is the behavior of the process P at the location l' in the bigraph B. On the other hand, if the migration action does not take place, then the subsequent behavior of the program is the behavior of the process Q at the location l in the bigraph B.

$$\mathbf{beh}(\langle l[[go^{\triangle t} l' \ then \ P \ else \ Q]], B\rangle) =_{df}$$
$$\mathbf{beh}(\langle \#t, B\rangle); \begin{pmatrix} (\mathbf{beh}(\langle go(l')@l, B\rangle); \mathbf{beh}(\langle l'[[P]], B\rangle)) \\ \lhd goflag \rhd \\ \mathbf{beh}(\langle l[[Q]], B\rangle) \end{pmatrix}$$

where $goflag = l' \in V_B \ \wedge \ (\exists p \in pts(l). \exists p' \in pts(l'). link(p) = link(p'))_B$.

In the above semantics definition, $\mathbf{beh}(\langle \#t, B\rangle)$ describes the behaviors of delaying t time units. For t time units, its trace keeps unchanged. After delaying t time units, if the boolean $goflag$ is true, then the migration action takes place and generates a snapshot attached to the end of the program trace.

The semantics of the output and input commands in our BigrTiMo is similar to the one in rTiMo [7]. Due to space limitations, here, we only take the semantic of the output command as an example.

$$\mathbf{beh}(\langle l[[a^{\triangle t}!\langle v\rangle \ then \ P \ else \ Q]], B\rangle) =_{df}$$
$$\begin{pmatrix} \mathbf{beh}(\langle a!\langle v\rangle@l, B\rangle); \mathbf{beh}(l[[P]], B\rangle) \\ \bigvee \\ \exists t' \in (0...t) \bullet (\mathbf{beh}(\langle \#t', B\rangle); \mathbf{beh}(\langle a!\langle v\rangle@l, B\rangle); \mathbf{beh}(l[[P]], B\rangle)) \\ \bigvee \\ \mathbf{beh}(\langle \#t, B\rangle); \mathbf{beh}(l[[Q]], B\rangle) \end{pmatrix}$$

Compared to the commands proposed in [7], the novelty operator in this paper is the control command which is used to update the global shared bigraph. This command can only be executed in sequential programs, and when it executes atomically, sequential programs from different configurations are not allowed to execute simultaneously.

The control command $\langle l[[control^{\triangle t}(r) \ then \ P]], B\rangle$ performs the BRR r to update the current bigraph B. After delaying t time units, the update action takes place and the next process is P.

$$\mathbf{beh}(\langle l[[control^{\triangle t}(r) \ then \ P]], B\rangle) =_{df}$$
$$\mathbf{beh}(\langle \#t, B\rangle); \mathbf{beh}(\langle control(r)@l, B\rangle); \mathbf{beh}(\langle l[[P]], B'\rangle)$$

For t time units, the update action is in a waiting state and its trace is

unchanged. After delaying t time units, the update action takes place successfully and a snapshot $(time', l, B, B')$ contributed by the update action is attached to the end of the program trace.

3.3 Denotational Semantics of Parallel Composition

The parallel composition $\langle l[[P]] \mid\mid l'[[Q]], B\rangle$ executes the process P from l and the process Q from l' (l and l' can be same or different) under the global shared bigraph B in two ways: (1) synchronous channel output in one process takes place simultaneously with the corresponding channel input in the other process; (2) other actions of processes take place independently. The composition is described by the following definition.

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

$$\mathbf{beh}(\langle l[[P_1]], B\rangle) \mid\mid \mathbf{beh}(\langle l'[[P_2]], B\rangle) =_{df}$$

$$\left(\begin{array}{c} \exists st_1, st_1', st_2, st_2', time_1, time_1', time_2, time_2', tr_1, tr_1', tr_2, tr_2' \bullet \\ \mathbf{beh}(\langle l[[P_1]], B\rangle)[st_1, st_1', time_1, time_1', tr_1, tr_1'/ \\ st, st', time, time', tr, tr'] \wedge \\ \mathbf{beh}(\langle l'[[P_2]], B\rangle)[st_2, st_2', time_2, time_2', tr_2, tr_2'/ \\ st, st', time, time', tr, tr'] \wedge \\ Merge \end{array}\right)$$

The first two predicates in the above definition describe the two independent behaviors of the configurations $\langle l[[P_1]], B\rangle$ and $\langle l'[[P_2]], B\rangle$. The predicate $Merge$ mainly does the merging of the contributed traces of the two behavioral branches, as well as the merging of the execution states and terminal times.

We now give the definition of $Merge$.

$$Merge =_{df} \left(\begin{array}{c} (st_1' = completed \ \wedge \ st_2' = completed) \Rightarrow st' = completed \wedge \\ (st_1' = wait \ \vee \ st_2' = wait) \Rightarrow st' = wait \wedge \\ time' = \mathbf{max}\{time_1', time_2'\} \wedge \\ \exists s \in (tr_1' - tr_1) \mid\mid (tr_2' - tr_2) \bullet tr' = tr^\frown s \end{array}\right)$$

The final execution state of the behavior of the parallel composition is determined by the two parallel components together. And the terminal time of the parallel composition is the maximum between the two terminal times of the parallel components. The merging of the contributed traces of the two behaviors can be defined as follows. The result of merging two empty traces (represented as ϵ) is still empty, which is illustrated in **case-1**. If one of the two traces is empty and the other is nonempty, the result follows the nonempty one shown in **case-2**. And **case-3** shows that function $\mid\mid$ is symmetric.

case-1 $\epsilon \mid\mid \epsilon =_{df} \{\epsilon\}$ **case-2** $s \mid\mid \epsilon =_{df} \{s\}$ **case-3** $s \mid\mid t =_{df} t \mid\mid s$

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.

$$t_1 = \pi_1(\mathbf{head}(s)),\ l_1 = \pi_2(\mathbf{head}(s)),\ \sigma_1 = \pi_3(\mathbf{head}(s)),\ \kappa_1 = \pi_4(\mathbf{head}(s))$$
$$t_2 = \pi_1(\mathbf{head}(t)),\ l_2 = \pi_2(\mathbf{head}(t)),\ \sigma_2 = \pi_3(\mathbf{head}(t)),\ \kappa_2 = \pi_4(\mathbf{head}(t))$$

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. The bigraph is a global shared variable, so we have $\sigma_1 = \sigma_2 = \sigma$. In this case, neither of the two actions can be an update action, since when the update action is executed, no other action can be executed at this time. Thus, we only need to consider the following two cases: (1) the two actions both are communication actions (denoted by **case-4**); (2) at least one of the two actions is not a communication action: if κ_1 is a communication action, then κ_2 should be a migration action, and if κ_1 is a migration action, then κ_2 can be a migration action or a communication action (denoted by **case-5**).

$$\textbf{case-4}\quad s \mid\mid t =_{df} \begin{cases} \left(\begin{pmatrix} sc^\frown(\textbf{tail}(s) \mid\mid \textbf{tail}(t)) \\ \lhd\ \textbf{Mess}(\kappa_1) = \textbf{Mess}(\kappa_2)\ \rhd \emptyset \end{pmatrix} \right), & \text{if comflag=true;} \\ \left(\begin{pmatrix} \\ \lhd\ \textbf{Chan}(\kappa_1) = \textbf{Chan}(\kappa_2)\ \rhd \mathrm{T} \end{pmatrix} \right) & \\ \mathrm{T}, & \text{otherwise.} \end{cases}$$

where $comflag = (l_1 = l_2) \vee (\exists p \in pts(l_1) \cdot \exists p' \in pts(l_2) \cdot link(p) = link(p'))_\sigma$;

$sc = \langle (t_1, (l_1, l_2), \sigma, \kappa_1) \rangle$;

and $\mathrm{T} = \langle (t_1, l_1, \sigma, \kappa_1) \rangle^\frown(\textbf{tail}(s) \mid\mid t) \cup \langle (t_2, l_2, \sigma, \kappa_2) \rangle^\frown(s \mid\mid \textbf{tail}(t))$.

$comflag = true$ means that the two communication components are at the same location described by the first predicate $l_1 = l_2$, or the locations of the two components are connected in the current bigraph σ described by the second predicate. $comflag = false$ means that the two components cannot communicate with each other. If $comflag = false$, then we only need to attach $\textbf{head}(s)$ or $\textbf{head}(t)$ to the end of the program trace (denoted by T). If $comflag = true$, then we have the following descriptions.

- If $\textbf{Chan}(\kappa_1)$ equals to $\textbf{Chan}(\kappa_2)$ which means that the two channels are same, then we consider the messages. If $\textbf{Mess}(\kappa_1)$ equals to $\textbf{Mess}(\kappa_2)$ which means that the two messages are same, then a synchronization occurs and a snapshot sc 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 $\textbf{Chan}(\kappa_1)$ and $\textbf{Chan}(\kappa_2)$ are different, then a synchronization does not happen and we only need to attach $\textbf{head}(s)$ or $\textbf{head}(t)$ to the end of the program trace (denoted by T).

For the case that at least one of the two actions is not a communication action, then a synchronization does not take place (denoted by T).

case-5 $s \mid\mid t =_{df} \mathrm{T}$.

According to **case-3**, we know that function $\mid\mid$ is symmetric. Thus, we only need to consider the case $t_1 < t_2$ which means that κ_1 occurs before κ_2 (denoted by **case-6**). In this case, κ_1 (or κ_2) can be a communication action, a migration action or an update action. And we only need to attach the first snapshot of s to the end of the program trace.

case-6 $s \mid\mid t =_{df} \langle (t_1, l_1, \sigma_1, \kappa_1) \rangle^\frown(\textbf{tail}(s) \mid\mid t)$.

Example 3. Let us consider the configuration in *Example* 2 in Section 3.1 again, where $N_1 = sp[[appsf]]$, $N_2 = pc[[appbs]]$, $G = \langle N_1 \parallel N_2, b\rangle$.

As mentioned in *Example* 2, a trace of $\langle N_1, b\rangle$ is given as below:

$s = \langle(0, sp, b, bs.work), (3, sp, b, b1), (5, sp, b1, b2), (6, sp, b2, b3)\rangle$.

A trace of $\langle N_2, b\rangle$ is: $t = \langle(0, pc, b, bs.work)\rangle$.

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

$s \parallel t = \langle(0, (sp, pc), b, bs.work)\rangle^\frown(s' \parallel \epsilon)$

where $s' = \langle(3, sp, b, b1), (5, sp, b1, b2), (6, sp, b2, b3)\rangle$.

According to **case-2**, we can obtain $s' \parallel \epsilon = \{s'\}$.

Finally, we obtain one trace of G by merging s and t below:

$\langle(0, (sp, pc), b, bs.work), (3, sp, b, b1), (5, sp, b1, b2), (6, sp, b2, b3)\rangle$. □

4 Algebraic Properties

Program properties can be expressed as algebraic laws and equations. In [1], we have presented a set of algebraic laws for BigrTiMo. Our denotational semantics in this paper can support the proofs of these laws. From these proofs, we can see that our semantics definitions are very rigorous. Due to space limitations, we only take some representative laws as examples.

(Output1) $\langle l[[a^{\triangle t}!\langle v\rangle \ then \ P \ else \ Q]], B\rangle$

$= \ \langle a!\langle v\rangle@l \to l[[P]], B\rangle$

$\oplus \exists t' \in (0...t) \bullet (\langle \#t' \to a!\langle v\rangle@l \to l[[P]], B\rangle)$

$\oplus \langle \#t \to l[[Q]], B\rangle$, where $t > 0$.

From the law **(Output1)**, we can see that the output command can be converted into a hybrid guarded choice. And in this guarded choice, the first branch indicates that the output action occurs at the activation time of the output command. The second branch indicates that the output action takes place after delaying t' time units, where $t' \in (0...t)$. And the third branch indicates that the output action does not happen before the timeout t.

Proof. By the semantics definitions for the output command and hybrid guarded choice, we know that they have the same form. Thus, this law is correct. □

(Control1) $\langle l[[control^{\triangle t}(r) \ then \ P]], B\rangle = \langle \#t \to control(r)@l \to l[[P]], B\rangle$ where $t > 0$, $r = R \to R'$, $B = C \circ R \circ d$ and $B' = C \circ R' \circ d$.

From the law **(Control1)**, we can see that the control command can be converted into a delay guard followed by an instantaneous event guard, which indicates that after delaying t time units, the control action occurs successfully.

Proof. By the semantics definitions for the delay guarded choice and the instantaneous guarded choice, we have that $\mathbf{beh}(\langle \#t \to control(r)@l \to l[[P]], B\rangle)$ equals to $\mathbf{beh}(\langle \#t, B\rangle); \mathbf{beh}(\langle control(r)@l, B\rangle); \mathbf{beh}(\langle l[[P]], B'\rangle)$. According to the definition of the semantics for the control command, we see that they have the same form, so this law is correct. □

5 Conclusion

BigrTiMo is a process algebra for structure-aware mobile systems. In this paper, we have studied the denotational semantics for BigrTiMo via the concept of UTP. Compared to the standard UTP theory, in addition to communication, the novelty in our UTP model covers time, location and global shared variable. Moreover, we give an example to show the contribution of BigrTiMo and illustrate how to use our semantic model and the trace-merging definition proposed in our paper under this example. We also demonstrate the proofs of some algebraic laws based on the denotational semantics.

Recently, Hoare has proposed the challenging research topic for studying semantic linking where the starting point is from the algebra semantics [9]. Hoare and He have studied the derivation of operational semantics from the algebraic semantics [4, 10]. For future work, we want to explore linking theory of the semantics for BigrTiMo.

Acknowledgments

This work was partly supported by Shanghai Collaborative Innovation Center of Trustworthy Software for Internet of Things (No. ZF1213).

References

1. Xie, W., Zhu, H., Xu, Q.: BigrTiMo-a process algebra for structure-aware mobile systems. In: ICECCS 2017, Fukuoka, Japan, November 6-8. (2017) 50–59
2. Aman, B., Ciobanu, G.: Real-time migration properties of rTiMo verified in Uppaal. In: SEFM 2013, Madrid, Spain, September 25-27. (2013) 31–45
3. Milner, R.: The Space and Motion of Communicating Agents. Cambridge University Press (2009)
4. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice Hall International Series in Computer Science (1998)
5. Ciobanu, G., Koutny, M.: Timed mobility in process algebra and Petri nets. J. Log. Algebr. Program. **80**(7) (2011) 377–391
6. Hoare, T.: Unifying semantics for concurrent programming. In: Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky. (2013) 139–149
7. Xie, W., Xiang, S.: UTP semantics for rTiMo. In: UTP 2016, Reykjavik, Iceland, June 4-5. (2016) 176–196
8. Shi, L., Zhao, Y., Liu, Y., Sun, J., Dong, J.S., Qin, S.: A UTP semantics for communicating processes with shared variables. In: ICFEM 2013, Queenstown, New Zealand, October 29 - November 1. (2013) 215–230
9. Hoare, T., van Staden, S.: In praise of algebra. Formal Asp. Comput. **24**(4-6) (2012) 423–431
10. He, J., Hoare, C.A.R.: From algebra to operational semantics. Inf. Process. Lett. **45**(2) (1993) 75–80