Time and Asynchrony in Distributed Computing

Ichiro Satoh

Department of Computer Science
Keio Univesity
3-14-1 Hiyoshi, Kohokuku, Yokohama 223, Japan

This dissertation was submitted in Janurary 1996 to Keio University
in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

This thesis formulates temporal aspects of distributed systems through developing a new process calculus and its two further expressive extensions.

The calculus is defined by extending existing non-timed process calculi. It provides a simple but powerful framework for describing synchronously communicating time-dependent processes. It has two new constructions for delayed processing and timed restriction, in addition to operational constructors found in many non-timed process calculi: sequential execution, parallel composition, synchronous communication, message scope restriction, and recursion definition. As a verification method, we develop several time-sensitive equation theories for processes and study their basic properties.

The two extensions reinforce the framework with the ability to express temporal properties of distributed computing. One of the extensions consists of some supplementary language constructors concerning with non-blocking message sending and process locations. It can model the notion of asynchrony and delay in communication among remotely located processes. On the basis of it, we give algebraic order relations which can order two behaviorally equivalent processes with respect to their relative execution speeds. The relations offer suitable verification and optimization techniques for asynchronously communicating systems, in particular very-large distributed systems.

The other extension lets the original calculus embody the expressive capability of multiple inaccurate clocks. It allows us to analyze the influence of inaccuracy and difference among physical clocks upon distributed systems. We develop an equivalence relation which can equate two processes when their behaviors are completely matched and their timings are different within a given bound. It provides a method to verify distributed systems with time uncertainties and non-strict time constraints.

We present some comparison with related work and discuss open problems and further applications of this work.

Acknowledgements

I would first of all like to thank my advisor, Professor Mario Tokoro, whose ideas, inspirations, and criticisms have contributed substantially to this work. Also, I would like to thank Professor Norihisa Doi, Professor Hikoe Enomoto, Professor Masaaki Shibuya, and Professor Naoki Yonezaki for providing useful comments and suggestions.

Many people have commented on previous presentations of this material and I would like to thank them all. Also, I would like to thank all the members of Tokoro laboratory for fruitful discussions and conversations. In particular, Dr. Vasco Vasconcelos and Kensuki Fukuda have provided me stimulating comments and discussions.

Also, it would not be right to ignore the excellent facilities and support provided by Keio University and its Computer Science Department.

Lastly, I am profoundly grateful to my parents for the encouragement and support they have provided over the years.

Ichiro Satoh
January, 1996

Contents

Chapter 1

Introduction

Time is an important and interesting issue in science and philosophy, although we do not have any sense of time. Time in computer science is a matter of major concern for several reasons. First, every computer is a physical entity, instead of any imaginary thing like the Turing machine and the lambda calculus. Therefore, computation in all real computers is never instantaneous and must take an amount of time. One of the main theme in the history of computer science and computer industry has been to reduce the execution time of computation as well as to enrich its functionalities.

The birth of distributed computing systems brings us the need of having a more deeply understanding time issue. A distributed computing system consists of autonomous computers communicating over communication networks with narrow bandwidth, high latency, and failure. When a computer sends a message to another, the other receives it at a later time, instead of the same time. It is even possible that one gives information to another and the other will not receive it. Therefore, computers are difficult to know the current status of other computers including time information. This limitation prevents distributed systems from having any absolute time reference which is available among all computers.

This does not create any problem only if each computer could always run independently. However, computers in distributed systems are often required to cooperate with each other through message passing and synchronization. When the timings of message arrival and synchronization are not correct, cooperations among computers may become inefficient or even fail. Also, in the majority cases, computers have to know time information from their own physical clocks, which never measure at the

same rate.[1] Differences in the time information of computers may lead the timings of the cooperations to failure. Thus, it is necessary to analyze the timings of these cooperations quantitatively in the development of distributed systems.

The existence of communication delay also affects the style of computation in distributed systems. For efficiency reasons, communication among remote computers is often realized as an asynchronous form, instead of a synchronous one. This is because in synchronous communication settings, sender processes must be blocked for at least the round-trip time of messages, including communication delay.

Time is highly related with fault tolerance in distributed systems. All elements of distributed systems are not reliable. For example, when some computers may go down and communication channels may lose messages, other computers may wait for messages which will never come. Therefore, computers need the ability to detect such dead-lock situations. However, by means of logical methods, computers are difficult to detect these situations. Hence, we usually use a special technique by using the notion of time, called *timeout handling*. Time is the last resort that computers can depend on in distributed computing environments, for the sake of their surviving against various faults and unexpected latencies.

It is very important to remember that distributed systems often cooperate with their external environments, — called the *real world* — which include various non-computational entities such as sensors, actuators, humans, and nature. To cooperate with the real-world at proper timings, these systems need to know when particular events occur in the real world. The time of the real-world is *real*, instead of any logical step. Distributed systems which interact with the real-world must have the notion of real-time.

There is a class of computer systems where their computation is subjected to certain time constraints. Systems in this class need to perform their required responses at their required timings. Such systems are often called *real-time systems* or *time-critical systems*.[2] Some of the systems are distributed. Examples include distributed

[1] Many methods to synchronize computer clocks to a known degree of accuracy have been studied. However, these methods need special protocols such as Network Time Protocol[48], but they are not always available in all distributed systems.

[2] The term "real-time system" is often used as different meanings. Therefore, it is difficult to find any accurate definition to encompass all the characteristics of real time systems. Here we shall use the term "real-time system" to indicate the class of systems which need to do their required responses on time or within specified time limits. Also, if they cannot do, they need to perform their

systems controlling factory automation, traffic control systems, telephone switches, robots, medical intensive care units, and numerous others. Since these systems often control safety-critical devices, their timing failures may result in a fatal loss of life and finance. Therefore, they must satisfy their certain time constraints which are required at their specifications. To develop such systems, we need to predicate the temporal properties of the systems quantitatively and then verify whether the systems can really satisfy their temporal specifications.

As we discussed previously, time plays a very important role in distributed computing. However, time is often treated as an insignificant or afterthought thing in most of the existing theories of computation, even theories for distributed computation. There are several plausible reasons for narrowing the scope of theories such as to get a better understanding of fundamental concepts or to get tractable theories. On the other hand, our claim is that, to establish the correct and efficient distributed systems, the need of strictly reasoning about time is inevitably derived. The aim of this thesis is to formulate time aspects of distributed computing through developing new process calculi.

1.1 Basic Framework

It is hard to design and understand distributed systems because they are complex and dynamic. In any event, we must design distributed systems that perform as we intend, and understand existing distributed systems well enough for modification as needs change. It is very useful to formulate a theoretical model that can simplify the complexity of distributed systems and postulate a set of rules to predicate the behaviors of distributed systems. Over the past twenty years, a variety of formalisms have been developed for the specification and the verification of distributed computing systems, for example temporal logic [22, 47, 60], dynamic logic [74], automata [46], state machine [72], Petri nets [63], process calculi [4, 10, 35, 51], and so on. They have been used as a framework to reason about behavioral properties of distributed systems such as control flow, computed values, and ordering events.

However, as mentioned previously, the correctness and efficiency of distributed

specified exception handling.

systems often depend on their temporal properties as well as their behavioral ones. There is a potential demand for developing a theoretical framework to analyze the temporal properties of distributed systems as well as their behavioral ones. This thesis studies a concurrency theory for reasoning about various temporal aspects of distributed systems through formulating new process calculi.

Here we should explain the reason why our theory is based on a process calculus approach. A process calculus is a mathematical framework which describes parallel computing around interprocess communications. Besides, the computation of distributed systems is essentially based on communications among autonomous processors through exchanging messages. Therefore, a process calculus is considered as a powerful framework to reason about distributed computing around communications. We also believe that the theory should provide a description language having the structure of a programming language, in order to ease the burden of synthesizing a specification into a fully-realized system. A process calculus approach allows us to describe systems in a fashion concordant with programming languages as compared to other approaches such as temporal logic, dynamic logic, and Petri net.

The theory needs to have language constructions to specify temporal properties found in distributed systems such as execution time, delayed processing, and timeout handling. However, ordinary process calculi do not have the notion of time and thus cannot describe these properties quantitatively. Indeed, there have been a few process calculi which have the notion of time. However, the purposes of existing time extended calculi are just to describe parallel computing systems instead of distributed systems. Therefore, they are difficult to model peculiar temporal properties of distributed systems such as multiple inaccurate clocks and communication delay.

This thesis aims at the establishment of a theoretical framework to specify and verify temporal aspects of distributed systems including inaccurate clocks and communication delay, through developing new process calculi and their verification methodologies.

1.2 Approach

This section considers possibilities of formalizing distributed computation. Since distributed computing systems are various and complex, there is no single formalism

that can accurately characterize all the properties of distributed systems. We will formalize distributed computing according to the following strategy; we first categorize distributed systems according to features of distributed systems. Next, we formulate different formalisms that can exactly include the attributes that affect our interesting aspects in distributed systems.

A distributed computing system consists of a collection of autonomous processors linked by communication networks. The primary concern of this thesis is to formulate temporal aspects of distributed systems around interprocess communications. First we should categorize communications found in distributed systems. It is useful to distinguish between *asynchronous* and *synchronous* communications.[3] The former consists of two blocking primitives: *blocking message send* and *blocking message receive*. Execution of either of the two primitives is deferred until the both primitives can take place simultaneously. On the other hand, the latter is characterized by *non-blocking message send* and *blocking message receive*.[4] Sender processes can send messages without synchronizing with anything and continue their next steps.

From a view of time, distributed computing can be classified into two sorts; whether every process has the same time source or not. The former means that processes at different computers share the same clock. Also, when processes follow different but well-synchronized clocks, the processes should be considered to share the same time source. The latter means that processes follow different clocks.

According to the above criteria: whether communication is synchronous or asynchronous, and whether the time reference of each process is the same or different, we will formulate three formal models for distributed systems as shown below:

- The first model aims at describing time-dependent systems with synchronous communication using the same clock. It is formulated by extending an existing non-timed process calculus with two time-dependent constructors. It also inherits all the operational constructors of the original calculus: sequential execution, parallel composition, synchronous communication, message scope restriction,

[3] In this thesis we concentrate on modeling only point-to-point communications because other communications for example multicast/broadcast communications can be considered as enhancements of point-to-point communications.

[4] Just as *send* operation can be blocking or non-blocking, so can *receive* operation. The non-blocking *receive* is not common because it is more difficult to be implemented.

and recursion definition. It provides a simple but powerful specification language for real-time systems in a single processor and small distributed systems.

- The second model is intended to describe a distributed system with asynchronous communication using the same clocks. This system corresponds to a distributed system where processes are allocated remotely and communicate asynchronously but a clock synchronization mechanism is well supported. The model has some supplement language constructors concerning with non-blocking message sending and blocking message receiving, in order to capture asynchronous message passing. Also, it introduces the notion of process location to specify latency in interprocess communications.

- The third model is purposed to capture a distributed system with synchronous communication using different clocks. This system corresponds to a distributed system where any clock synchronization mechanism is unsupported or inefficient. The model is characterized by having the ability to express multiple inaccurate clocks quantitatively, in addition to the expressive power of the first model.

The first model is defined as a time-extended process calculus and provides a cardinal basis of the other models studied in this thesis. The second and third models are formulated by enriching the first model with expressive capabilities of some peculiar properties of distributed computing systems such as asynchronous communication, communication delay, and multiple clocks.

From the above criteria, there might be a possibility to formalize a distributed system with asynchronous communication with different clocks. The second and third models are extended calculi of the first one but the both extensions are independent and never interferes with each other. Therefore, we can easily combine the second and the third models and then establish a framework to describe such a distributed system by combining them. Such a framework might be expressive but complex. For concentrating on formalizing temporal aspects in distributed systems, we avoid to treat such a combination in this thesis.

1.3 About This Thesis

This thesis is organized as follows: Chapter 2 defines an elementary process calculus for time-dependent systems where processes communicate synchronously and follow the same clock. The calculus is called *RtCCS* (*Real-time Calculus of Communicating Systems*) and is formulated based on an existing untimed process calculus, CCS [51]. *RtCCS* has a minimal but expressive set of constructors for describing real-time systems and provides a basis of developing further formalisms for distributed systems studied in the following chapters. Also, we present the definition of three time-sensitive equivalence relations: timed strong equivalence, timed observation equivalence, and timed observation congruence. We study some basic properties of the relations. We present some examples to demonstrate the utilities of the calculus and the relations.

Chapter 3 defines another process calculus by extending the calculus developed in Chapter 2 with the ability to express distributed systems where processes communicate asynchronously and follow the same clock. The calculus is called $RtCCS_A$. It allows us to analyze the temporal and behavioral properties of asynchronous interactions among remotely located processes. On the basis of the time-sensitive equivalences studied in Chapter 2, we also define an equivalence relation and order relations. The equivalence can equate two processes according to the results of asynchronous interactions with testing processes residing at remote locations. The order relations order between two behaviorally equivalent processes according to their execution speeds. We study basic properties of these relations.

Chapter 4 proposes a process calculus that is intended to capture a distributed system where processes communicate synchronously and follow different clocks. This calculus is an extension of the calculus presented in Chapter 2 with expressive capabilities of multiple inaccurate clocks. It is called $RtCCS_L$. Furthermore, we define a pseudo time-sensitive equivalence. It can equate two behaviorally equivalent processes when their timings are different within a given bound. We investigate basic properties of the relation. Some examples are shown to demonstrate its usefulness.

In Chapter 5, we compare the results presented in this thesis with some related work on formal methods for distributed computing and real-time computing. Finally, in Chapter 6 we summarize the achievements of this thesis and give some directions

on our future work.

Chapter 2

A Time Extended Process Calculus

This chapter defines an elementary process calculus for time-dependent computing systems where communication is synchronous and all processes follow the same clock.[1] The calculus is a minimal extension of an existing non-timed process calculus with quantitatively expressive capabilities for temporal aspects of concurrent computing systems. The underlying motivation of the calculus is to establish a theoretical basis for further expressive extensions developed in the following chapters. Also, this chapter formulates several equivalence relations as a verification technique for time-dependent systems.

2.1 Introductory Remarks

There is a large class of distributed computing systems which have certain time constraints which must be satisfied. The correctness of systems in this class depends not only on the logical results of computation, but also on the time at which the results are produced. In order to construct correct programs for these systems, it is necessary to analyze the temporal properties of the systems in advance. However, the construction and debugging of programs for such systems are far more complex and difficult than those of ordinary concurrent programs. Also, even when verifying

[1] This chapter is a modified version of articles that have been published earlier in [64, 65].

ordinary distributed computing systems, it is important to analyze their temporal properties. Hence, we need the support of a formal method for reasoning about the temporal properties of distributed computing systems as well as their functionally behavioral properties.

This chapter is to formulate an elementary process calculus for specifying time-dependent systems where communication is synchronous and every process shares the same clock. The calculus is defined based on an existing process calculus, CCS [51].[2] This is because CCS is a simple but powerful framework to describe non-deterministic and concurrent systems. However, since CCS lacks the notion of time, we extend CCS with temporal expressive capabilities. The newly proposed calculus is called *RtCCS* (*Real-time Calculus of Communicating Systems*).

2.2 Basic Framework

This calculus is an extension of CCS with the notion of time. We explain our basic ideas in the calculus below.

Time Values

We assume that time is interval between events instead of any absolute time, and is dense instead of discrete time.[3] Time values are denoted as positive real numbers.

The Passage of Time

Time passes in all processes at the same speed. Also, all processes follow the same clock, or different but well synchronized clocks. The advance of time is modeled as a special transition which is labeled by a quantity of time to indicate the amount of the advance, for example $E \xrightarrow{\langle t \rangle} E'$. It means that process E become E' after t real time.

Action Atomicity

To preserve the pleasant properties of the original calculus, all communication and

[2] Our extensions are essentially independent of CCS itself and thus can easily be applied into other process calculi.

[3] We will present another calculus based on discrete time in Chapter 4.

internal actions are assumed to be instantaneous. Instead, we introduce special language constructions to express the execution time of behaviors as shown below.

Time-Dependent Behaviors

We introduce two new prefix operators whose contents are dependent on the passage of time, called *delay operator* and *time restriction operator*.

delay operator: This is to suspend a process for a specified period, written as $\langle t \rangle$, where t is the amount of the suspension. For instance, $\langle t \rangle.E$ means a process which is idle for t real-time and then behaves like E.

time restriction operator: This is to restrict the execution of a process, written as prefix $[t]$, where t is the deadline time. For example, $[t].E$ behaves like process E if E can execute an internal or communication action within t time, whereas $[t].E$ terminates if E does not perform any action within t time.

Non Unnecessarily Idling

We assume that when an internal or communication action is enabled, processes must perform the action without imposing unnecessary idling. This assumption lets us measure the minimum cost in synchronization among parallel processes, and enables the calculus to preserve the observation properties of CCS.

2.3 The Language and its Semantics

The syntax of the elementary calculus is coincide with all the construction of CCS except for two new time-dependent operators, called *delay operator* and *time restriction operator*. The semantics of the calculus can computationally encompass that of CCS and embody the notion of time.

Notation

We first define the notations of time values. Time is assumed to be dense in the calculus developed in this chapter.

Definition 2.3.1 Let $\mathcal{R}^{+0}$ denote the set of the positive reals. We call $\mathcal{R}^{+0}$ *real time domain.* □

We assume that the domain has the following binary operators: $+$ (*addition*), $=$ (*equal*), and $<$ (*total order*), where these operators are the same as the usual mathematical operators. Next, we define symbols to present the events of processes.

Definition 2.3.2

- Let $\mathcal{A}$ be a infinite set of names denoting communication actions. Its elements are denoted as a,b,...
- Let $\overline{\mathcal{A}}$ be a infinite set of co-names. Its elements are denoted as $\overline{a}$,$\overline{b}$,... where $\overline{a}$ is the complementary action of a, and $\overline{\overline{a}}$ is equal to a.
- Let $\Lambda \equiv \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of communication action names. Elements of the set are written as $\lambda, \lambda', \ldots$.
- Let τ denote an internal action.
- Let Γ be the set of actions corresponding the amount of the passage of time. Elements of the set are denoted as $\langle t_1 \rangle, \langle t_2 \rangle, \ldots$, where $t_1, t_2, \ldots \in \mathcal{R}^{+0}$.
- Let $Act \equiv \Lambda \cup \{\tau\}$ be the set of operational actions. Its elements are denoted as α,β,... . □

τ-action represents all handshake communications and is considered to be unobservable from outside environments.

Syntax

The syntax of the calculus is identical with that of CCS except for two time-dependent operators.

Definition 2.3.3 The set $\mathcal{E}$ of *RtCCS* expressions ranged over by $E, E_1, E_2, \ldots$ is defined recursively by the following abstract syntax:

$$\begin{aligned} E \;::=\; & \mathbf{0} \;\mid\; X \;\mid\; \alpha.E \;\mid\; E_1 + E_2 \;\mid\; E_1|E_2 \;\mid\; \mathbf{rec}\, X : E \\ & \mid\; E[f] \;\mid\; E \setminus L \;\mid\; \langle t \rangle.E \;\mid\; [t].E \end{aligned}$$

where t is an element of $\mathcal{R}^{+0}$ and f is a relabeling function $f : Act \to Act$ and L is a subset of Λ. We assume that X is always *guarded*.[4] We often denote $E_1 + \cdots + E_n$ as $\sum_{i\in\{0,\ldots,n\}} E_i$. □

We define *relabeling function* $f : Act \to Act$.

Definition 2.3.4 We define a *relabeling* function to be any function $f : Act \to Act$ which respects complements:

$$f(\overline{\lambda}) \;=\; \overline{f(\lambda)} \qquad f(\tau) \;=\; \tau$$

where λ is an element of Λ and τ is an internal action. □

Remarks The informal meaning of each process constructor is as follows:

- *Terminate Process*, $\mathbf{0}$. This process is a stopped process which can perform no internal nor communication action.
- *Action Prefix*, $\alpha.E$. This is a process to perform action α and then behaves like E, where α is an input, output, or internal action.
- *External Choice*, $E_1 + E_2$. This represents a process which may behave as either E_1 or E_2.
- *Parallel Composition*, $E_1 \mid E_2$. This represents that process E_1 and E_2 may run in parallel.
- *Action Relabeling*, $E[f]$. This process behaves like E but with the actions in E relabeled by function f.
- *Action Restriction*, $E \setminus L$. This process behaves like E but it is prohibited to communicate with external processes at actions in $L \cup \bar{L}$.
- *Recursion*, $\mathbf{rec}\, X : E$. This expression binds free occurrences of X in E.
- *Delay*, $\langle t\rangle.E$. This represents a process which is suspended for t real time and then behaves like E.

[4] X is called *guarded* in E if each occurrence of X is only within some subexpressions $\alpha.E'$ in E where α is not an empty element; c.f. *unguarded* expressions, e.g. $\mathbf{rec}\, X : X$ or $\mathbf{rec}\, X : X + E$.

- *Time Restriction*, $[t].E$. This represents a process such that it behaves like E if E can execute an internal or communication action within t real time, whereas it behaves like a terminate process if E does not perform any action within t real time.

Definition 2.3.5 An occurrence of variable X in an expression $E \in \mathcal{E}$ is *bound* if it occurs in a subexpression of form $\mathbf{rec}\, X : F$. Otherwise it is *free*. E is an *open* expression if it contains a free occurrence of a variable, and a *closed* expression otherwise. □

Operational Semantics

The operational semantics of the calculus is defined in terms of a labeled transition system [59]. It consists of two kinds of transition rules. One of them defines the semantics of functional behaviors of processes, called *behavioral transition*, written as $\stackrel{\alpha}{\longrightarrow}$ ($\longrightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$) and the another defines the passage of time on processes, called *temporal transition*, written as $\stackrel{\langle t \rangle}{\longrightarrow}$ ($\longrightarrow \subseteq \mathcal{E} \times \Gamma \times \mathcal{E}$).

Definition 2.3.6 *RtCCS* is a labeled transition system $\langle\ \mathcal{E},\ Act \cup \Gamma,\ \{\ \stackrel{\mu}{\longrightarrow} \subseteq \mathcal{E} \times \mathcal{E} \mid \mu \in Act \cup \Gamma\ \}\ \rangle$. The transition relation $\longrightarrow$ is defined by two kinds of structural induction rules given in Figure 1 and 2. □

In giving the rules, we adopt the convention that the transition below the horizontal line may be inferred from the transitions above the line.

For simplicity, we often use a more readable form $A \stackrel{\text{def}}{=} P$, instead of $\mathbf{rec}\, X : E$, where A is an element of the set of process constants.

Definition 2.3.7 The labeled transition relation $\stackrel{\mu}{\longrightarrow}$ is reformulated by the same rules as $\stackrel{\mu}{\longrightarrow}$ presented in Definition 2.3.6 where $\mu \in Act \cup \Gamma$ and the following rules:

$$\frac{P \stackrel{\alpha}{\longrightarrow} P'}{A \stackrel{\alpha}{\longrightarrow} P'}\ (A \stackrel{\text{def}}{=} P) \qquad\qquad \frac{P \stackrel{\langle t \rangle}{\longrightarrow} P'}{A \stackrel{\langle t \rangle}{\longrightarrow} P'}\ (A \stackrel{\text{def}}{=} P)$$

□

$$\frac{-}{\alpha.E \xrightarrow{\alpha} E} \qquad \frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 + E_2 \xrightarrow{\alpha} E_1'} \qquad \frac{E_2 \xrightarrow{\alpha} E_2'}{E_1 + E_2 \xrightarrow{\alpha} E_2'}$$

$$\frac{E_1 \xrightarrow{\alpha} E_1'}{E_1 \mid E_2 \xrightarrow{\alpha} E_1' \mid E_2} \qquad \frac{E_2 \xrightarrow{\alpha} E_2'}{E_1 \mid E_2 \xrightarrow{\alpha} E_1 \mid E_2'} \qquad \frac{E_1 \xrightarrow{\lambda} E_1',\ E_2 \xrightarrow{\overline{\lambda}} E_2'}{E_1 \mid E_2 \xrightarrow{\tau} E_1' \mid E_2'}$$

$$\frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]} \qquad \frac{E \xrightarrow{\alpha} E',\ \alpha \notin L \cup \overline{L}}{E \setminus L \xrightarrow{\alpha} E' \setminus L} \qquad \frac{E\{\mathbf{rec}\, X : E/X\} \xrightarrow{\alpha} E'}{\mathbf{rec}\, X : E \xrightarrow{\alpha} E'}$$

$$\frac{E \xrightarrow{\alpha} E'}{[t].E \xrightarrow{\alpha} E'}(t > 0) \qquad \frac{E \xrightarrow{\alpha} E'}{\langle 0 \rangle.E \xrightarrow{\alpha} E'}$$

Figure 1: Inference Rules for Behavioral Transition

$$\frac{-}{\mathbf{0} \xrightarrow{\langle t \rangle} \mathbf{0}} \qquad \frac{-}{\lambda.E \xrightarrow{\langle t \rangle} \lambda.E}$$

$$\frac{E_1 \xrightarrow{\langle t \rangle} E_1',\ E_2 \xrightarrow{\langle t \rangle} E_2'}{E_1 + E_2 \xrightarrow{\langle t \rangle} E_1' + E_2'} \qquad \frac{E_1 \xrightarrow{\langle t \rangle} E_1',\ E_2 \xrightarrow{\langle t \rangle} E_2'}{E_1 \mid E_2 \xrightarrow{\langle t \rangle} E_1' \mid E_2'}(E_1 \mid E_2 \not\xrightarrow{\tau})$$

$$\frac{E \xrightarrow{\langle t \rangle} E'}{E[f] \xrightarrow{\langle t \rangle} E'[f]} \qquad \frac{E \xrightarrow{\langle t \rangle} E'}{E \setminus L \xrightarrow{\langle t \rangle} E' \setminus L} \qquad \frac{E\{\mathbf{rec}\, X : E/X\} \xrightarrow{\langle t \rangle} E'}{\mathbf{rec}\, X : E \xrightarrow{\langle t \rangle} E'}$$

$$\frac{E \xrightarrow{\langle t \rangle} E'}{\langle 0 \rangle.E \xrightarrow{\langle t \rangle} E'} \qquad \frac{-}{\langle t + t' \rangle.E \xrightarrow{\langle t \rangle} \langle t' \rangle.E}\ (t + t' > 0)$$

$$\frac{-}{[0].E \xrightarrow{\langle t \rangle} [0].E} \qquad \frac{E \xrightarrow{\langle t \rangle} E'}{[t + t'].E \xrightarrow{\langle t \rangle} [t'].E'}\ (t + t' > 0)$$

Figure 2: Inference Rules for Temporal Transition

Remark In [64, 65] we proposed an early version of the calculus presented in this chapter. We show a relationship between the early version and the present one. The early version is characterized by introducing a special binary operator, $\langle\ ,\ \rangle_t$ called *timeout* operator. For instance, $\langle E_1, E_2\rangle_t$ behaves as process E_1 if E_1 can execute an internal or communication action within t units of time, whereas $\langle E_1, E_2\rangle_t$ behaves as process E_2 if E_1 does not perform any action within t units of time. The operator can be encoded into *RtCCS*'s operators as follows:

$$\langle E_1, E_2\rangle_t \ \equiv\ [t].E_1 + \langle t\rangle.E_2$$

where $\equiv$ is a syntactic equation over expressions. On the contrary, we can encode new time-dependent operators $[t]$ and $\langle t\rangle$ studied in this thesis into expressions in the calculus presented in [64, 65] by using the timeout operator as follows: $\langle t\rangle.E \ \equiv\ \langle \mathbf{0}, E\rangle_t$ and $[t].E \ \equiv\ \langle E, \mathbf{0}\rangle_t$.

Basic Properties of Timed Semantics

We present some basic properties of the semantics of the calculus.

Proposition 2.3.8 *Maximal Progress*

$$P \xrightarrow{\ \tau\ } P' \textit{ then, there is not some } P'',\ P \xrightarrow{\ \langle t\rangle\ } P'' \text{ where } t > 0. \qquad \square$$

Proof. The proof is by transition induction on $P \xrightarrow{\ \tau\ } P'$. We consider the transition rules applied in the last step of the inference. There are four cases: (i) Let $P \equiv \tau.P'$ Then, by induction there are no t and P'' such that $P \xrightarrow{\ \langle t\rangle\ } P''$, because there is not any $\langle t\rangle$-transition for $\tau.P'$. (ii) Let $P \equiv P_1 + P_2$. We assume $P_1 + P_2 \xrightarrow{\ \tau\ } P'$ is inferred from $P_1 \xrightarrow{\ \tau\ } P'$. By induction, there are no t and P'' such that $P_1 \xrightarrow{\ \langle t\rangle\ } P''$. Hence, $P_1 + P_2 \xrightarrow{\ \langle t\rangle\ } P'$ cannot be inferred. The case that $P_1 + P_2 \xrightarrow{\ \tau\ } P'$ is inferred from $P_2 \xrightarrow{\ \tau\ } P'$ is symmetrical. (iii) Let $P \equiv P_1|P_2$ and $P_1|P_2 \xrightarrow{\ \tau\ } P'$. To infer $P_1|P_2 \xrightarrow{\ \langle t\rangle\ } P'$, we have $P_1|P_2 \not\xrightarrow{\ \tau\ } P'$. (iv) Let $P \equiv P' \setminus L$, $P \equiv P'[f]$, $P \equiv \mathbf{rec}\, X : E$, $P \equiv \langle 0\rangle.P'$, and $P \equiv [t].P'$ $(t > 0)$. The result follows by induction. $\square$

If a process has an executable communication or an internal action, it must perform the action immediately without imposing unnecessary idling.

Proposition 2.3.9 *Time Determinacy*

$$P \xrightarrow{\langle t \rangle} P' \text{ and } P \xrightarrow{\langle t \rangle} P'' \text{ then, } P' \text{ and } P' \text{ are syntactically identical.} \quad \square$$

Proof. The proof is by transition induction on $P \xrightarrow{\langle t \rangle} P_1$ and $P \xrightarrow{\langle t \rangle} P_2$. There are five cases: (i) Let $P \equiv \mathbf{0}$, $P \equiv \alpha.P'$. Then, it is trivial. (ii) Let $P \equiv P_1 + P_2$, $P_1 \xrightarrow{\langle t \rangle} P_1'$, and $P_2 \xrightarrow{\langle t \rangle} P_2'$. Then, P' and P'' are required to be $P_1' + P_2'$. (iii) Let $P \equiv P_1 | P_2$. It is similar to the case of summation. (iv) Let $P \equiv \langle \dot{t} \rangle.\dot{P}$, $P \xrightarrow{\langle t-\dot{t} \rangle} P_1$, and $P \xrightarrow{\langle t-\dot{t} \rangle} P_2$. By induction, we have $P_1 \equiv P_2$. (v) Let $P \equiv P' \setminus L$, $P \equiv P'[f]$, $P \equiv \mathbf{rec}\, X : E$, and $P \equiv [\dot{t}].\dot{P}$. By inductive hypothesis of P, we get the result. $\square$

A process may reach different states non-deterministically after performing a communication action or an internal one. On the other hand, time is deterministic in the sense that the passage of time does not interfere with any non-determinism.

Proposition 2.3.10 *Time Continuity*

$$P \xrightarrow{\langle t_1 + t_2 \rangle} P' \text{ then, for some } \dot{P},\ P \xrightarrow{\langle t_1 \rangle} \dot{P} \text{ and } \dot{P} \xrightarrow{\langle t_2 \rangle} P'. \quad \square$$

Proof. The result can directly be proved by transition induction on $P \xrightarrow{\langle t \rangle} P'$. $\square$

If a process proceeds from one instant to the other, it must reach all the intermediate instants between them.

2.4 Time-Sensitive Equivalence Relations

There have been many equivalence relations for verifying (non-timed) communicating processes. For example, *trace equivalence* identifying processes with the same behavioral language, for example see [11], *failure equivalence* identifying processes with the same set of failures (impossible actions) after a trace, for example see [35, 29], and *bisimulation equivalence* identifying processes which can simulate to each other, for example [58, 51]. These non time-sensitive equivalence relations provide powerful methods to verify untimed concurrent systems. In order to complete the calculus, we will formulate time-sensitive equivalence relations by extending untimed bisimulation.

Timed Strong Equivalence

Since the operational semantics of the calculus is given in terms of a labeled transition system, the notion of bisimulation can directly be introduced into the calculus. We present a time-sensitive equivalence relation over process expressions. It is an extension of CCS's strong bisimulation with the notion of time.

Definition 2.4.1 A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *strong bisimulation* if $(P_1, P_2) \in \mathcal{R}$ implies, for all $\mu \in Act \cup \Gamma$,

(i) $\forall P_1'$: $P_1 \xrightarrow{\mu} P_1'$ *then* $\exists P_2'$: $P_2 \xrightarrow{\mu} P_2'$ *and* $(P_1', P_2') \in \mathcal{R}$.
(ii) $\forall P_2'$: $P_2 \xrightarrow{\mu} P_2'$ *then* $\exists P_1'$: $P_1 \xrightarrow{\mu} P_1'$ *and* $(P_1', P_2') \in \mathcal{R}$.

P_1 and P_2 are *strongly equivalent*, written $P_1 \sim_{\mathcal{T}} P_2$, if there exists a strong bisimulation $\mathcal{R}$ such that $(P_1, P_2) \in \mathcal{R}$. □

Intuitively, if P_1 and P_2 are strongly equivalent, they cannot be distinguished from one another in their behaviors and timings.

Note that the above definition is equal to that of CCS's strong bisimulation ($\sim$) except that $\mu \in Act \cup \Gamma$ appears in place of $\mu \in Act$.

Proposition 2.4.2 (1) $\sim_{\mathcal{T}}$ is symmetric, reflexive, and transitive. (2) $\sim_{\mathcal{T}}$ is the largest strong bisimulation. □

Proof. Definition 2.4.1 almost directly tells us this. □

We show a set of laws which are sound with respect to the strong equivalence below.

Proposition 2.4.3

(1) $P_1 + P_2 \sim_{\mathcal{T}} P_2 + P_1$
(2) $P_1 + (P_2 + P_3) \sim_{\mathcal{T}} (P_1 + P_2) + P_3$
(3) $P + P \sim_{\mathcal{T}} P$
(4) $P + \mathbf{0} \sim_{\mathcal{T}} P$ □

Proof. We shall only prove (1); the others are just as easy. Let $P_1 + P_2 \xrightarrow{\mu} Q_1$. We will prove only the case of $\mu = \langle t \rangle$. We need to show that $\mathcal{R}$ is a strong bisimulation, where $\mathcal{R} = \{ (P_1 + P_2, P_2 + P_1) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$. So let $P_1 + P_2 \xrightarrow{\langle t \rangle} Q_1$. It is enough to find Q_2 such that $P_2 + P_1 \xrightarrow{\langle t \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$ (with a symmetric argument). $P_1 \xrightarrow{\langle t \rangle} P_1'$ and $P_2 \xrightarrow{\langle t \rangle} P_2'$ are required from Definition 2.3.6. Hence, we know $P_1 + P_2 \xrightarrow{\langle t \rangle} P_1' + P_2'$ and $P_2 + P_1 \xrightarrow{\langle t \rangle} P_2' + P_1'$ and clearly $(P_1' + P_2', P_2' + P_1') \in \mathcal{R}$. By a symmetric argument, we complete the proof. □

Proposition 2.4.4

(1) $P_1 \mid P_2 \sim_{\mathcal{T}} P_2 \mid P_1$
(2) $P_1 \mid (P_2 \mid P_3) \sim_{\mathcal{T}} (P_1 \mid P_2) \mid P_3$
(3) $P \mid \mathbf{0} \sim_{\mathcal{T}} P$
(4) $P \setminus L \sim_{\mathcal{T}} P$ if $\mathcal{L}(P) \cap (L \cup \overline{L}) = \emptyset$
(5) $(P_1 + P_2) \setminus L \sim_{\mathcal{T}} P_1 \setminus L + P_2 \setminus L$
(6) $(P_1 \mid P_2) \setminus L \sim_{\mathcal{T}} P_1 \setminus L \mid P_2 \setminus L$ if $\mathcal{L}(P_1) \cap \overline{\mathcal{L}(P_2)} \cap (L \cup \overline{L}) = \emptyset$
(7) $(\alpha.P)[f] \sim_{\mathcal{T}} f(\alpha).P[f]$
(8) $(P_1 + P_2)[f] \sim_{\mathcal{T}} P_1[f] + P_2[f]$
(9) $(P_1 \mid P_2)[f] \sim_{\mathcal{T}} P_1[f] \mid P_2[f]$ □

Proof. We shall only prove (1); the other cases are just as easy. It is enough to show that $\mathcal{R} = \{ (P_1|P_2, P_1|P_2) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$ is a strong bisimulation. So let $P_1|P_2 \xrightarrow{\mu} Q_1$; it is enough to find Q_2 such that $P_2|P_1 \xrightarrow{\mu} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$ (with a symmetric argument). There are two main cases.

Case 1 $\mu \in Act$.

Subcase 1.1 $P_1 \xrightarrow{\mu} P_1'$ and $Q_1 \equiv P_1'|P_2$. Then, we have $P_2|P_1 \xrightarrow{\mu} Q_2 \equiv P_2|P_1'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$

Subcase 1.2 $P_2 \xrightarrow{\mu} P_2'$ and $Q_1 \equiv P_1|P_2'$. Similar.

Subcase 1.3 $\mu = \tau$, $P_1 \xrightarrow{\lambda} P_1'$, and $P_2 \xrightarrow{\overline{\lambda}} P_2'$. Then, $P_2|P_1 \xrightarrow{\tau} Q_2 \equiv P_2'|P_1'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$

Case 2 $\mu = \langle t \rangle$. Then $P_1|P_2 \xrightarrow{\langle t \rangle} Q_1$ must be inferred from Definition 2.3.6. $Q_1 \equiv P_1'|P_2'$ where $P_1 \xrightarrow{\langle t \rangle} P_1'$ and $P_2 \xrightarrow{\langle t \rangle} P_2'$. Hence we have $P_2|P_1 \xrightarrow{\langle t \rangle} Q_2 \equiv P_2'|P_1'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

This proves the first part of Definition 2.4.1. The second part follows by a symmetric argument and we have then shown that $\mathcal{R}$ is a strong bisimulation. □

Proposition 2.4.5

(1) $\langle t\rangle.(P_1 + P_2) \sim_{\mathcal{T}} \langle t\rangle.P_1 + \langle t\rangle.P_2$

(2) $\langle t\rangle.(P_1|P_2) \sim_{\mathcal{T}} \langle t\rangle.P_1|\langle t\rangle.P_2$

(3) $\langle t_1 + t_2\rangle.P \sim_{\mathcal{T}} \langle t_1\rangle.\langle t_2\rangle.P$

(4) $[t].(P_1 + P_2) \sim_{\mathcal{T}} [t].P_1 + [t].P_2$

(5) $[t_1 + t_2].P \sim_{\mathcal{T}} [t_1].P + [t_1 + t_2].P$

(6) $\langle t_1\rangle.[t_2].P \sim_{\mathcal{T}} [t_1 + t_2].\langle t_1\rangle.P$

(7) $[t_1].[t_2].P \sim_{\mathcal{T}} \begin{cases} [t_1].P & \textit{where } t_1 \leq t_2 \\ [t_2].P & \textit{otherwise} \end{cases}$ □

Proof. (1) Let $\langle t\rangle.(P_1+P_2)\xrightarrow{\mu}Q_1$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{\, (\langle t\rangle.(P_1 + P_2), \langle t\rangle.P_1 + \langle t\rangle.P_2) \,|\, \forall P_1, P_2 \in \mathcal{R} \,\} \cup Id$. It is enough to find Q_2 such that $\langle t\rangle.P_1 + \langle t\rangle.P_2\xrightarrow{\langle i\rangle}Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. We will prove only the case of $\mu = \langle \dot{t}\rangle$. There are two cases to infer $\langle t\rangle.(P_1 + P_2)\xrightarrow{\langle t\rangle}Q_1$.

Case 1 $t \geq \dot{t}$. Then, from Definition 2.3.6, $\langle t\rangle.(P_1 + P_2)\xrightarrow{\langle \dot{t}\rangle} Q_1 \equiv \langle t - \dot{t}\rangle.\ (P_1 + P_2).\ \langle t\rangle.P_1\ +\langle t\rangle.P_2\xrightarrow{\langle \dot{t}\rangle}\ Q_2\equiv\ \langle t - \dot{t}\rangle.\ P_1+\ \langle t - \dot{t}\rangle.\ P_2$ is required. Hence, we have $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t < \dot{t}$, $P_1\xrightarrow{\langle \dot{t}-t\rangle}P_1'$, and $P_2\xrightarrow{\langle \dot{t}-t\rangle}P_2'$. Then $\langle t\rangle.(P_1 + P_2)\xrightarrow{\langle \dot{t}\rangle}.P_1' + P_2'$. Also we have $\langle t\rangle.P_1 + \langle t\rangle.P_2\xrightarrow{\langle \dot{t}\rangle}P_1' + P_2'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

By a symmetric argument, we complete the proof.

(2) Let $\langle t\rangle.(P_1|P_2)\xrightarrow{\mu}Q_1$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{\, (\langle t\rangle.(P_1|P_2), \langle t\rangle.P_1|\langle t\rangle.P_2) \,|\, \forall P_1, P_2 \in \mathcal{R} \,\} \cup Id$. It is enough to find Q_2 such that $\langle t\rangle.P_1|\langle t\rangle.P_2\xrightarrow{\langle \dot{t}\rangle}Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. We will prove only the case of $\mu = \langle \dot{t}\rangle$. There are two cases to infer $\langle t\rangle.(P_1|P_2)\xrightarrow{\langle t\rangle}Q_1$.

Case 1 $t \geq \dot{t}$. Then $\langle t\rangle.(P_1|P_2)\xrightarrow{\langle \dot{t}\rangle}\ Q_1 \equiv \langle t - \dot{t}\rangle.\ (P_1|P_2).\ \langle t\rangle.P_1\ |\langle t\rangle.P_2\xrightarrow{\langle \dot{t}\rangle}\ Q_2\equiv\ \langle t - \dot{t}\rangle.\ P_1|\ \langle t - \dot{t}\rangle.\ P_2$ is required from Definition 2.3.6. Hence, we have $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t < \dot{t}$, $P_1 \xrightarrow{\langle \dot{t}-t \rangle} P_1'$, and $P_2 \xrightarrow{\langle \dot{t}-t \rangle} P_2'$. Then $\langle t \rangle.(P_1|P_2) \xrightarrow{\langle \dot{t} \rangle} .P_1'|P_2'$. Also we have $\langle t \rangle.P_1|\langle t \rangle.P_2 \xrightarrow{\langle \dot{t} \rangle} P_1'|P_2'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

By a symmetric argument, we complete the proof.

(3) Let $\langle t_1 + t_2 \rangle.P \xrightarrow{\mu} Q_1$. We will prove only the case of $\mu = \langle t \rangle$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{ (\langle t_1 + t_2 \rangle.P, \langle t_1 \rangle.\langle t_2 \rangle.P) \mid \forall P_1, P_2 \in \mathcal{R} \} \cup Id$. It is enough to find Q_2 such that $\langle t_1 \rangle.\langle t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. There are three cases.

Case 1 $t < t_1$. Then $\langle t_1 + t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_1 \equiv \langle t_1 + t_2 - t \rangle.P$. We have $\langle t_1 \rangle.\langle t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_2 \equiv \langle t_1 - t \rangle.\langle t_2 \rangle.P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t_1 \leq t \leq t_1 + t_2$. Then $\langle t_1 + t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_1 \equiv \langle t_1 + t_2 - t \rangle.P$. We have $\langle t_1 \rangle.\langle t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_2 \equiv \langle t_2 - (t - t_1) \rangle.P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

Case 3 $t_1 + t_2 < t$ and $P \xrightarrow{\langle t - t_1 - t_2 \rangle} P'$ Then $\langle t_1 + t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_1 \equiv P'$. We have $\langle t_1 \rangle.\langle t_2 \rangle.P \xrightarrow{\langle t \rangle} Q_2 \equiv P'$ and clearly (P', P') is a strong bisimulation.

The case of $\mu = \alpha$ is trivial. By a symmetric argument, we complete the proof.

(4) Let $[t].(P_1 + P_2) \xrightarrow{\mu} Q_1$. We consider only the case of $\mu = \langle \dot{t} \rangle$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{ ([t].(P_1 + P_2), [t].P_1 + [t].P_2) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$. It is enough to find Q_2 such that $[t].P_1 + [t].P_2 \xrightarrow{\langle \dot{t} \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. We have two cases to $[t].(P_1 + P_2) \xrightarrow{\mu} Q_1$.

Case 1 $t \geq \dot{t}$, $P_1 \xrightarrow{\langle \dot{t} \rangle} P_1'$, and $P_2 \xrightarrow{\langle \dot{t} \rangle} P_2'$. Then, from Definition 2.3.6, $[t].(P_1 + P_2) \xrightarrow{\langle \dot{t} \rangle} Q_1 \equiv [t - \dot{t}].\ (P_1 + P_2).\ [t].P_1 + [t].P_2 \xrightarrow{\langle \dot{t} \rangle} Q_2 \equiv [t - \dot{t}].\ P_1 + [t - \dot{t}].P_2$ is required. Hence, we have $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t < \dot{t}$, $P_1 \xrightarrow{\langle \dot{t} \rangle} P_1'$, and $P_2 \xrightarrow{\langle \dot{t} \rangle} P_2'$. Then $[t].(P_1 + P_2) \xrightarrow{\langle \dot{t} \rangle} Q_1 \equiv P_1' + P_2'$. Also we have $[t].P_1 + [t].P_2 \xrightarrow{\langle \dot{t} \rangle} Q_1 \equiv P_1' + P_2'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

By a symmetric argument, we complete the proof.

(5) Let $[t_1 + t_2].P \xrightarrow{\mu} Q_1$. We will prove only the case of $\mu = \langle t \rangle$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{ ([t_1 + t_2].P, [t_1].P + [t_1 + t_2].P) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$. It is enough to find Q_2 such that $[t_1].P + [t_1 + t_2].P \xrightarrow{\langle t \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. There are two cases.

Case 1 $t \leq t_1$ and $P \xrightarrow{\langle t \rangle} P'$. Then $[t_1 + t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv [t_1 + t_2 - t].P$. We have $[t_1].P + [t_1 + t_2].P \xrightarrow{\langle t \rangle} Q_2 \equiv [t_1 - t].P + [t_1 + t_2 - t].P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t > t_1$ and $P \xrightarrow{\langle t \rangle} P'$. Then $[t_1 + t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv [t_1 + t_2 - t].P$. We have $[t_1].P + [t_1 + t_2].P \xrightarrow{\langle t \rangle} Q_2 \equiv [t_1 + t_2 - t].P$ and clearly (Q_1, Q_2) is a strong bisimulation.

By a symmetric argument, we complete the proof.

(6) Let $\langle t_1 \rangle.[t_2].P \xrightarrow{\mu} Q_1$. We will prove only the case of $\mu = \langle t \rangle$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{ (\langle t_1 \rangle.[t_2].P, [t_1 + t_2].\langle t_1 \rangle.P) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$. It is enough to find Q_2 such that $[t_1 + t_2].\langle t_1 \rangle.P \xrightarrow{\langle t \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. There are three cases.

Case 1 $t < t_1$. Then $\langle t_1 \rangle.[t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv \langle t_1 - t \rangle.[t_2].P$. We have $[t_1 + t_2].\langle t_1 \rangle.P \xrightarrow{\langle t \rangle} Q_2 \equiv [t_1 + t_2 - t].\langle t_1 - t \rangle.P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t_1 \leq t$ and $P \xrightarrow{\langle t - t_1 \rangle} P'$. Then $\langle t_1 \rangle.[t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv [t_2 - (t - t_1)].P'$. We have $[t_1 + t_2].\langle t_1 \rangle.P \xrightarrow{\langle t \rangle} Q_2 \equiv [t_1 + t_2 - t].P'$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

By a symmetric argument, we complete the proof.

(7) Let $[t_1].[t_2].P \xrightarrow{\mu} Q_1$. We will prove only the case of $\mu = \langle t \rangle$ and $t_1 \geq t_2$. We need to show that $\mathcal{R}$ is a strong bisimulation where $\mathcal{R} = \{ ([t_1].[t_2].P, [t_2].P) \mid \forall P_1, P_2 \in \mathcal{P} \} \cup Id$. It is enough to find Q_2 such that $[t_2].P \xrightarrow{\langle t \rangle} Q_2$ and $(Q_1, Q_2) \in \mathcal{R}$. There are three cases.

Case 1 $t < t_2$. Then $[t_1].[t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv \langle t_1 - t \rangle.[t_2 - t].P$. We have $[t_2].P \xrightarrow{\langle t \rangle} Q_2 \equiv [t_2 - t].P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

Case 2 $t_2 \leq t$ and $P \xrightarrow{\langle t \rangle} P'$. Then $[t_1].[t_2].P \xrightarrow{\langle t \rangle} Q_1 \equiv [t_1].[0].P$. We have $[t_2].P \xrightarrow{\langle t \rangle} Q_2 \equiv [0].P$ and clearly $(Q_1, Q_2) \in \mathcal{R}$.

By a symmetric argument, we complete the proof. □

Proposition 2.4.6

(1) $\langle 0\rangle.P \sim_{\mathcal{T}} P$
(2) $[0].P \sim_{\mathcal{T}} \mathbf{0}$ □

Proof. From Definition 2.3.6 and 2.4.1, we directly get these results. □

Hereafter, we assume that $\langle 0\rangle.P$ is syntactically equal to P and $[0].P$ to $\mathbf{0}$.

Proposition 2.4.7 Let $P_1 \sim_{\mathcal{T}} P_2$. Then we have the following properties:

(1) $\alpha.P_1 \sim_{\mathcal{T}} \alpha.P_2$
(2) $P_1 + P \sim_{\mathcal{T}} P_2 + P$
(3) $P_1|P \sim_{\mathcal{T}} P_2|P$
(4) $P_1 \setminus L \sim_{\mathcal{T}} P_2 \setminus L$
(5) $P_1[f] \sim_{\mathcal{T}} P_2[f]$
(6) $\langle t\rangle.P_1 \sim_{\mathcal{T}} \langle t\rangle.P_2$
(7) $[t].P_1 \sim_{\mathcal{T}} [t].P_2$ □

Proof. We prove only (6) and (7) and can proof the other operations in the same way. First we consider (6). It is enough to show that $\mathcal{R} = \{(\langle t\rangle.P_1, \langle t\rangle.P_2) \mid P_1 \sim_{\mathcal{T}} P_2\} \cup Id$ is a strong bisimulation. Let $\langle t\rangle.P_1 \xrightarrow{\mu} Q_1$. We treat the case of $\mu = \langle t\rangle$. Then, $\langle t\rangle.P_1 \xrightarrow{\langle t'\rangle} Q_1$. There are two cases:

Case 1 $\langle t\rangle.P_1 \xrightarrow{\langle t'\rangle} Q_1$ and $t \geq t_1$. Then $\langle t\rangle.P_1 \xrightarrow{\langle t'\rangle} \langle t - t'\rangle.P_1$. Then $\langle t\rangle.P_1 \xrightarrow{\langle t'\rangle} \langle t - t'\rangle.P_1$. From Definition 2.3.6, $\langle t\rangle.P_2 \xrightarrow{\langle t'\rangle} \langle t - t'\rangle.P_2$ and clearly $(\langle t - t'\rangle.P_1, \langle t - t'\rangle.P_2) \in \mathcal{R}$ is a strong bisimulation.

Case 2 $\langle t\rangle.P_1 \xrightarrow{\langle t'\rangle} Q_1$ and $t < t_1$. Then, $\langle t\rangle.P_1 \xrightarrow{\langle t\rangle} P_1$ and $P_1 \xrightarrow{\langle t'-t\rangle} Q_1$. From Definition 2.3.6, $\langle t\rangle.P_2 \xrightarrow{\langle t\rangle} P_2$ is required. Also, because $P_1 \sim_{\mathcal{T}} P_2$, we have $P_2 \xrightarrow{\langle t'-t\rangle} Q_2$ and clearly (Q_1, Q_2).

By a symmetric argument, we establish both the case (i) and (ii) of Definition 2.4.1.

Next we will prove (7). It is enough to show that $\mathcal{R} = \{([t].P_1, [t].P_2) \mid P_1 \sim_{\mathcal{T}} P_2,\ t \geq 0\}$ is a strong bisimulation. Let $[t].P_1 \xrightarrow{\mu} Q_1$. We treat the case of $\mu = \langle t\rangle$. There are three cases:

Case 1 $\mu \in Act$ and $P_1 \xrightarrow{\mu} P_1'$ Then, $[t].P_1 \xrightarrow{\mu} P_1'$ is required. Since $P_1 \sim_{\mathcal{T}} P_2$, we have $[t].P_2 \xrightarrow{\mu} P_2'$ with $P_1' \sim_{\mathcal{T}} P_2'$. The result follows directly.

Case 2 $t \geq t'$, $[t].P_1 \xrightarrow{\langle t' \rangle} Q_1$, $P_1 \xrightarrow{\langle t' \rangle} P_1'$, Then, $Q_1 \equiv [t - t'].P_1'$ is required. Since $P_1 \sim_{\mathcal{T}} P_2$, we have $P_2 \xrightarrow{\langle t' \rangle} P_2'$ with $P_1' \sim_{\mathcal{T}} P_2'$. Also, from Definition 2.3.6, $[t].P_2 \xrightarrow{\langle t' \rangle} [t - t'].P_2'$. Hence, $(Q_1, Q_2) \in \mathcal{R}$.

Case 3 $t > t'$. Then $[t].P_1 \xrightarrow{\langle t' \rangle} \mathbf{0}$ is inferred from Definition 2.3.6. Also, we know $[t].P_2 \xrightarrow{\langle t' \rangle} \mathbf{0}$.

By a symmetric argument, we complete the proof. □

This tells us that $\sim_{\mathcal{T}}$ is substitutive everywhere except under the recursion operation. We have not defined the strong equivalence relation over expressions with variables. Here we extend to expressions with variables.

Definition 2.4.8 Let E and F include one free variable X at most. Then $E \sim_{\mathcal{T}} F$ implies for all P of process expressions $\mathcal{P}$, $E\{P/X\} \sim_{\mathcal{T}} F\{P/X\}$ □

Proposition 2.4.9 Let E and F include one free variables X at most. Then $E \sim_{\mathcal{T}} F$ implies $\mathbf{rec}\, X : E \sim_{\mathcal{T}} \mathbf{rec}\, X : F$.

Proof. By structural induction on E and F. □

Proposition 2.4.10 Let $A \stackrel{\text{def}}{=} P$, then $A \sim_{\mathcal{T}} P$. □

Proof. By Definition 2.3.3, $A \stackrel{\text{def}}{=} P$ corresponds to the recursively definition. We have that A and P have exactly the same derivative tree. □

Theorem 2.4.11 *Strong equivalence $\sim_{\mathcal{T}}$ is preserved by all operators.* □

Proof. By Proposition 2.4.7 and 2.4.9. □

By this proposition we can guarantee that if two processes are strongly equivalent, the processes are substitutable for each other in any context. The property is very useful in compositional constructions and verifications for time-dependent systems.

Remarks

We presented a relationship between our strong equivalence ($\sim_{\mathcal{T}}$) and CCS' strong equivalence ($\sim$). It seems that we can always assume that if $P \sim Q$ holds in CCS, then $P \sim_{\mathcal{T}} Q$ holds in *RtCCS*. There is however a counter example: unguarded recursion expressions, e.g., $\mathbf{rec}\, X : X$ expression. This is because in *RtCCS* such expressions are not allowed. However, we think that such expressions are very unrealistic in concurrent and parallel processes in real-world. Our strong equivalence can equate processes in *RtCCS* which are equivalent according to CCS's strong equivalence.

Timed Observation Equivalence

The strong equivalence has several useful algebraic properties. However this is under the assumption that an observer is able to observe all actions, including an internal action (i.e. a invisible action) — τ action — which indeed should not be observed. We present a weaker equivalence.

The transition relation $\longrightarrow$ defined in chapter 2 does not distinguish between observable and unobservable actions. We define two transition relations due to the non-observability of τ.

Definition 2.4.12

(i) $P \stackrel{\mu}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$

(ii) $P \stackrel{\widehat{\mu}}{\Longrightarrow} Q \stackrel{\text{def}}{=} P(\stackrel{\tau}{\longrightarrow})^* \stackrel{\mu}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* Q$ if $\mu \neq \tau$ and otherwise $P(\stackrel{\tau}{\longrightarrow})^* Q$. □

We are now ready to define an equivalence with the concept of observability.

Definition 2.4.13 A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a *weak bisimulation* if $(P_1, P_2) \in \mathcal{R}$ implies, for all $\mu \in Act \cup \Gamma$,

(i) $\forall P_1'$: $P_1 \stackrel{\mu}{\Longrightarrow} P_1'$ *then* $\exists P_2'$: $P_2 \stackrel{\widehat{\mu}}{\Longrightarrow} P_2'$ *and* $(P_1', P_2') \in \mathcal{R}$.

(ii) $\forall P_2'$: $P_2 \stackrel{\mu}{\Longrightarrow} P_2'$ *then* $\exists P_1'$: $P_1 \stackrel{\widehat{\mu}}{\Longrightarrow} P_1'$ *and* $(P_1', P_2') \in \mathcal{R}$.

P_1 and P_2 are *observation-equivalent*, written $P_1 \approx_{\mathcal{T}} P_2$, if there exists a weak bisimulation $\mathcal{R}$ such that $(P_1, P_2) \in \mathcal{R}$. □

Intuitively, if P_1 and P_2 are observation-equivalent, each action of P_1 must be matched by a sequence of actions of P_2 with the same visible contents and timing, and conversely. The equivalence can equate two processes that are not distinguishable by their observable behaviors and the timings of their behaviors.

Proposition 2.4.14 (1) $\approx_{\mathcal{T}}$ is symmetric, reflexive, and transitive. (2) $\approx_{\mathcal{T}}$ is the largest strong bisimulation. □

Proof. Definition 2.4.13 almost directly tells us this. □

Proposition 2.4.15 For all $P_1, P_2 \in \mathcal{P}$, $P_1 \ \sim_{\mathcal{T}} \ P_2$ implies $P_1 \approx_{\mathcal{T}} P_2$. □

Proof. Definition 2.4.1 and 2.4.13 directly tells that every strong bisimulation is also a observation-bisimulation. □

In terms of $\sim_{\mathcal{T}}$, we have already presented many of equational laws. They are also valid for $\approx_{\mathcal{T}}$ because of $\sim_{\mathcal{T}} \subseteq \approx_{\mathcal{T}}$. Therefore we show only other equational laws which are invalid for the strong equivalence, but valid for the observation equivalence.

Proposition 2.4.16

(1) $\tau.P \ \approx_{\mathcal{T}} \ P$
(2) $\alpha.\tau.P \ \approx_{\mathcal{T}} \ \alpha.P$
(3) $\tau.P + P \ \approx_{\mathcal{T}} \ \tau.P$
(4) $\alpha.(\tau.P + Q) \ \approx_{\mathcal{T}} \ \alpha.(\tau.P + Q) + \alpha.Q$

Proof. We shall prove (1) only. The other cases can be proved from the application of Definition 2.4.13, just as Proposition 2.4.3. We will prove that $\mathcal{R} = \{ (\tau.P, P) \mid P \in \mathcal{P} \} \cup Id$ is a weak bisimulation. So, let $\tau.P \stackrel{\langle t \rangle}{\Longrightarrow} P_1$ then it is enough to find $P \stackrel{\widehat{\langle t \rangle}}{\Longrightarrow} P_2$ and $(P_1, P_2) \in \mathcal{R}$, whereas, let $P \stackrel{\mu}{\Longrightarrow} P_2$ then it is enough to find P_1. $\tau.P \stackrel{\widehat{\mu}}{\Longrightarrow} P_1$ and $(P_1, P_2) \in \mathcal{R}$. If $\tau.P \stackrel{\tau}{\Longrightarrow} P$, $P \stackrel{\widehat{\tau}}{\Longrightarrow} P$ and clearly $(P, P) \in \mathcal{R}$. If $P \stackrel{\mu}{\Longrightarrow} P'$, then $\tau.P \stackrel{\widehat{\tau}}{\Longrightarrow} \stackrel{\widehat{\mu}}{\Longrightarrow} P'$, and clearly $(P, P) \in \mathcal{R}$. □

Proposition 2.4.17

(1) $\langle t \rangle.\tau.P \ \approx_{\mathcal{T}} \ \langle t \rangle.P$

(2) $[t].\tau.P \approx_{\mathcal{T}} P$
(3) $[t_1].P + \langle t_1\rangle.\tau.[t_2].P \approx_{\mathcal{T}} [t_1 + t_2].\alpha.P$ □

Proof. Easy application of Definition 2.4.13, as Proposition 2.4.5 □

Proposition 2.4.18 Let $P_1 \approx_{\mathcal{T}} P_2$. Then we have the following properties:

(1) $\alpha.P_1 \approx_{\mathcal{T}} \alpha.P_2$
(2) $P_1|Q \approx_{\mathcal{T}} P_2|Q$
(3) $P_1 \setminus L \approx_{\mathcal{T}} P_2 \setminus L$
(4) $P_1[f] \approx_{\mathcal{T}} P_2[f]$
(5) $\langle t\rangle.P_1 \approx_{\mathcal{T}} \langle t\rangle.P_2$ □

Proof. As Proposition 2.4.11. □

Proposition 2.4.19 Let E and F include one free variables X at most. Then $E \approx_{\mathcal{T}} F$ implies $\mathbf{rec}\, X : E \approx_{\mathcal{T}} \mathbf{rec}\, X : F$.

Proof. By structural induction on E and F. □

Proposition 2.4.20 Let $A \stackrel{\text{def}}{=} P$, then $A \approx_{\mathcal{T}} P$. □

Proof. By Definition 2.3.3, $A \stackrel{\text{def}}{=} P$ corresponds to the recursively definition. We have that A and P have exactly the same derivative tree. □

Theorem 2.4.21 Observation-equivalence $\approx_{\mathcal{T}}$ is preserved by all operators except for summation and time restriction operators. □

Proof. By Proposition 2.4.13 and 2.4.19. □

Unfortunately, like the weak equivalence of CCS[51], our weak equivalence is not congruent. We show counterexamples:

$$P \approx_{\mathcal{T}} \tau.P \qquad \text{implies} \qquad [t].P \not\approx_{\mathcal{T}} [t].\tau.P$$

Timed Observation Congruence

To have a congruent relation based on the observation-equivalence, we need a little restriction on the definition of the equivalence.

Definition 2.4.22 P_1 and P_2 are *observation-congruent* if for all $\mu \in Act \cup \Gamma$

(i) $\forall P_1'$: $P_1 \stackrel{\mu}{\Longrightarrow} P_1'$ *then* $\exists P_2'$: $P_2 \stackrel{\widehat{\mu}}{\Longrightarrow} P_2'$ *and* $P_1' \approx_{\mathcal{T}} P_2'$.

(ii) $\forall P_2'$: $P_2 \stackrel{\mu}{\Longrightarrow} P_2'$ *then* $\exists P_1'$: $P_1 \stackrel{\widehat{\mu}}{\Longrightarrow} P_1'$ *and* $P_1' \approx_{\mathcal{T}} P_2'$.

We write $P_1 =_{\mathcal{T}} P_2$ if P_1 and P_2 are *observation-congruent*. □

Proposition 2.4.23 For all $P_1, P_2 \in \mathcal{P}$

$$\textit{if } P_1 \sim_{\mathcal{T}} P_2 \textit{ then } P_1 =_{\mathcal{T}} P_2, \textit{ and if } P_1 =_{\mathcal{T}} P_2 \textit{ then } P_1 \approx_{\mathcal{T}} P_2. \qquad \square$$

Proof. By Definition 2.4.1, 2.4.13, and 2.4.22, $=_{\mathcal{T}}$ lies between $\sim_{\mathcal{T}}$ and $\approx_{\mathcal{T}}$. □

Theorem 2.4.24 *$=_{\mathcal{T}}$ is preserved by all operators.*

Proof. Same as Proposition 2.4.11. □

$=_{\mathcal{T}}$ is a congruence relation and very close to timed observation equivalence. The results provide a powerful method for proving the substitutability between two equivalent processes. We can grantee that if two processes are observation-congruent, they are substitutable for each other, even though their internal implementations are different.

2.5 Examples

This section demonstrates how to apply *RtCCS* to reason about time-dependent communicating systems.

Communication Protocol through N Processors

We consider a communication protocol through N processors (nodes) connected linearly. As the first step of our analysis, we describe a communication protocol between two neighborhood processors.

Communication Protocol between Two Neighborhood Processors

The communication protocol between two neighborhood processors consists of a sender process (written as S), and a receiver process (written as R), linked through a unreliable medium (written as M).

- Upon reception of data (action *send*), the sender process sends a data message to the medium (action $\overline{put}$). The process then waits for an acknowledgment message (action *ack*). If the acknowledgment cannot be received within a specified period of time (t real time), the sender retransmits the data message to the medium.
- After receiving a data message (action *put*), the medium transmits the data message to the receiver process (action $\overline{get}$). The delay of the transmission is d_1 real time. Also, when receiving an acknowledge message (action *ok*), it sends the message to the sender process (action $\overline{ok}$) after d_2 real time.
- The receiver process gets a data message (action *get*) and then returns an acknowledgment message to the sender through the medium and gives its neighborhood processor the data (action $\overline{receive}$).

We describe the protocol system in *RtCCS* as follows:

$$\begin{aligned} S &\stackrel{\text{def}}{=} send.S' \\ S' &\stackrel{\text{def}}{=} \overline{put}.([t].ack.S + \langle t \rangle.S') \\ M &\stackrel{\text{def}}{=} put.\langle d_1 \rangle.\overline{get}.M + ok.\langle d_2 \rangle.\overline{ack}.M \\ R &\stackrel{\text{def}}{=} get.\overline{ok}.(R|\overline{receive}.\mathbf{0}) \end{aligned}$$

Note that $\overline{put}.([t].ack.S + \langle t \rangle.S')$ plays a role of timeout handling for action *ack*.

The communication protocol is described as $(S|M|R) \setminus \{get, put, ack, ok\}$. By using the timed observation equivalence, the composition can be transformed into a process which is behaviorally and temporally equivalent but has a simpler structure.

$$\begin{aligned} (S \mid M \mid R) \setminus \{get, put, ack, ok\} &\approx_{\mathcal{T}} C \\ \text{where } C &\stackrel{\text{def}}{=} send.\langle d_1 \rangle.(\langle d_2 \rangle.C \mid \overline{receive}.\mathbf{0}) \end{aligned}$$

Let C be the behavioral and temporal specification of the protocol. Then, the above result means that the implementation of the protocol satisfies its specification successfully.

Communication Protocol through Linearly Connected N-Processors

Next, we linearly link N communication protocols through with $N-1$ node processors. That is, i-th protocol is linked with its its both neighborhood protocols through processors (written as $P_{i-1,i}$ and $P_{i,i+1}$). Let p be the execution of time in each processor.

$$P_{i,i+1} \stackrel{\text{def}}{=} receive_i.\langle p \rangle.\overline{send_{i+1}}.P_{i,i+1}$$

$P_{i,i+1}$ receives action $receive_i$ from i-th protocol. After an internal execution for p real time, it sends action $\overline{send_{i+1}}$. Communication through N protocols. is described as the following parallel composition:

$$\begin{aligned} Sys \stackrel{\text{def}}{=} & (C[receive_1/receive] \mid (P_{1,2} \mid C[send_2/send, receive_2/receive] \mid P_{2,3} \mid \cdots \\ & \cdots \mid C[send_{N-1}/send, receive_{N-1}/receive] \mid P_{N-1,N} \mid C[send_N/send]) \\ & \setminus \bigcup_{1 \le i < N} \{send_i\} \setminus \bigcup_{1 < i \le N} \{receive_i\} \end{aligned}$$

Moreover, Sys can be transformed into simpler expression Sys by means of the timed observation equivalence as shown below:

$$\begin{aligned} Sys &\approx_{\mathcal{T}} Sys' \text{ where} \\ P' &\stackrel{\text{def}}{=} receive.(\langle d_1 + d_2 \rangle.P' \mid \langle (N-1) \times p + N \times d_1 \rangle.\overline{send}.\mathbf{0}) \end{aligned}$$

We can analyze the behavioral and temporal properties of the whole communication system through simpler expression Sys more easily. For example, from Sys' we know that the communication delay among N protocols is $(N-1) \times p + N \times d_1$ and the front processor can become to receive data again after $d_1 + d_2$.

2.6 Concluding Remarks

This section introduced a new calculus for describing time-dependent systems. The calculus is a minimal extension of Milner's CCS by introducing two timed primitives:

delay operator and time restriction operator. The calculus can successfully encompass the pleasant properties of CCS and express various temporal properties of real time systems in a single processor and small distributed systems, such as execution time, delayed processing, and timeout handling.

In a large part of this chapter we have studied time-sensitive equivalence relations over time dependent processes: strong equivalence, observation equivalence, and observation congruence. The strong equivalence shows that if processes are strongly equivalent, they cannot be distinguished from one another in their time properties as well as their functional behaviors. The observation equivalence and the observation congruence can equate two processes that are not distinguishable in their observable behaviors and the timings of the behaviors. In particular, the strong equivalence and the observation congruence are proved to be congruent and thus provide a powerful method for proving the reusability of real-time processes.

Chapter 3

Locality in Communication

This chapter presents an extension of the calculus presented in Chapter 2 with the ability to express asynchronous message passing, communication delay, and the notion of process location.[1] We develop algebraic relations for asynchronously communicating processes located remotely.

3.1 Introductory Remarks

In Chapter 2 we established a process calculus for time-dependent systems where communication is synchronous — a process sends a message to another and the other must receive the message at the same time.

However, in a large distributed system, processors are allocated remotely. The spatial distance among distributed processes manifests communication delay. The delay seriously affects the timings of interprocess communications. For example, due to the delay, a process sends a message to another and the other will always receive the message at a later time. However, the amount of the delay cannot often be predicated exactly. Moreover, the delay often affects the style of communications among processes. In synchronous communication settings, a sender process must be blocked for at least the round-trip time of message transmission, including communication delay. For efficiency reasons, communications among distributed processes are often based on *asynchronous* forms instead of *synchronous* ones. However, asynchronous

[1] This chapter is a modified version of an article that has been published earlier in [71].

communication often results in another non-deterministic property.

Therefore, delay and asynchrony in communication create serious difficulties in the design and development of distributed systems. To construct correct and efficient programs for distributed systems, we need to analyze the influences of delay and asynchrony on the behavioral and temporal properties of the systems. This chapter addresses this problem and proposes a theoretical framework for specifying and verifying distributed systems with these features.

The framework consists of two parts: a description language and a verification method for asynchronous interactions among distributed processes. The language is formulated based on the calculus studied in Chapter 2. However, the original calculus is based on synchronous communication. This chapter extends the original calculus with the ability to express asynchronous message passing and delayed processing. Furthermore, we introduce the notion of process location and specify communication delay as temporal distance between the locations of sender and receiver processes. On the other hand, the verification method is formulated on the basis of algebraic relations over processes described in the language. Throughout this chapter, we assume that every process shares the same time reference. Also, in this thesis, for focusing temporal aspects in distributed systems, we avoid to introduce the expressive capability for a distributed system where topological connections between processes can change dynamically. However, the calculus can easily model such a system by incorporating with the port-passing mechanism developed in [52].

3.2 Basic Framework

We extend the calculus presented in Chapter 2 with the ability to specify asynchronous interactions among remote located processes. The extended calculus is called $RtCCS_A$. We summarize the basic idea of the extensions before giving its formal definition.

Extension: Spatial Location and Communication Delay

Communication delay is a function of geographic distance between sender processes and receiver ones. The amount of communication delay is highly dependent on the locations of sender and receiver processes. We introduce the notion of process location

to specify communication delay appropriately. Every process expression is enriched with a *location postfix*, written as $(E) : \ell$, which means that process E is allocated at location ℓ. The length of communication delay is given as a set of the possible distances between the locations of sender and receiver processes.

Extension: Asynchronous Communication
In distributed systems, communication between remotely located processes is often realized by means of asynchronous message passing. However, the calculus developed in the previous chapter is based on synchronous communication. We extend the following two primitives for asynchronous message passing: *non-blocking message send* and *blocking message receive*.

Non-blocking Message Send: This is represented by the creation of a process corresponding to an asynchronous message. The created process does not do anything except for being received by an input port for the message. The approach is basically the same to those developed in [5, 7, 36, 42]. However, our approach makes such a created process guarded by a delay operator to be suspended for the transmission latency of the message. Moreover, each message has its target location. For example, a process residing at location ℓ which sends message a to a process at location ℓ' is written as $(\ell' \uparrow a.E) : \ell$, where E is its body program.

Blocking Message Receive: This is modeled as an input-action prefix which is basically the same as the input action primitive of *RtCCS*. We assume that each process can receive only the messages that arrive at its location. For example, $(\downarrow a.E) : \ell$ represents a process which is at location ℓ and receives a message which arrives at the location and then behaves like $(E) : \ell$. We also assume that the order of message arrival is indeterminate.

3.3 Definition

The syntax and the semantics of this calculus are formulated based on those of the calculus presented in Chapter 2.

Notation

We first define new notations which we will use hereafter. Communication delay is given as a function parameterized by the locations of sender and receiver processes, written as Δ.

Definition 3.3.1

- Let $\mathcal{L}\mathrm{oc}$ be an infinite set of location names. Its elements are denoted as ℓ,ℓ_1,ℓ_2,....
- Let $\Delta \subseteq \mathcal{L}\mathrm{oc} \times \mathcal{L}\mathrm{oc} \to 2^{\mathcal{R}^{+0}}$ be a *communication delay* function.

Note that, to model unpredictable communication delay, $\Delta(\ell, \ell')$ specifies all the possible amount of communication delay from location ℓ to ℓ'. For example, $\Delta(\ell_1, \ell_2) = [10, 12]$ means that the communication delay from location ℓ_1 to ℓ_2 varies from 10 to 12 real time.

Syntax

In order to clarify our exposition, we divide the expressions into two groups: expressions for describing local processes written as $\mathcal{E}$, and expressions for describing asynchronous interactions among remotely located processes written as $\mathcal{P}$.

Definition 3.3.2 The set $\mathcal{E}$ of local process expressions is defined by the following expressions:

$$\begin{array}{rcl} E & ::= & \ell\!\uparrow a.E \;\mid\; \langle t\rangle.E \;\mid\; X \;\mid\; \mathbf{rec}\, X : E \;\mid\; F \\ F & ::= & \mathbf{0} \;\mid\; \downarrow a.E \;\mid\; F_1 + F_2 \;\mid\; [t].F \end{array}$$

where a is a message name in $\mathcal{A}$, t is an element of $\mathcal{R}^{+0}$, and ℓ is a location name in $\mathcal{L}\mathrm{oc}$. Hereafter we shall often use the more readable notation $X \stackrel{\mathrm{def}}{=} E$ instead of $\mathbf{rec}\, X : E$. □

Definition 3.3.3 The set $\mathcal{P}$ of remote process expressions is defined by the following expressions:

$$P \;::=\; (E) : \ell \qquad \mid \qquad P_1 \,|\, P_2 \qquad \mid \qquad P \setminus N$$

where N is a subset of Λ, and f is an action mapping $f : Act \to Act$. □

We give the intuitive meanings of some important constructors in the language below. The meanings of other constructors are equal to those of *RtCCS*.

- $(E):\ell$ means that process E is residing at location ℓ.
- $(\ell' \uparrow a.E) : \ell$ represents a process which is residing at location ℓ. The process asynchronously sends message a to a process at location ℓ' and behaves like E.
- $(\downarrow a.E):\ell$ represents a process that is at location ℓ and receives message a which arrives at location ℓ, and then behaves like E.

We define a function to extract location names from P expressions.

Definition 3.3.4 The function $|\cdot|_{Loc} : \mathcal{P} \to 2^{\mathcal{L}\mathrm{oc}}$, which presents the location names of processes, is defined as follows:

$$|(E):\ell|_{Loc} = \{\ell\}, \quad |P_1|P_2|_{Loc} = |P_2|_{Loc} \cup |P_2|_{Loc}, \quad |P \setminus N|_{Loc} = |P|_{Loc} \qquad \square$$

Semantics

The semantics of $RtCCS_A$ is defined through two steps: *location translation rule* and *labeled transition system*. The former translates location-dependent expressions into expressions in *RtCCS* process, written as $(E) : \ell$. The latter is equal to the operational semantics of *RtCCS*. The translated expressions can be interpreted as *RtCCS* expressions.

Location Translation

Before defining the location translation rule, we explain its basic idea. The following examples will be enough to understand it.

$$\begin{aligned} (\ell_R \uparrow a.E_S):\ell_S &\longrightarrow \sum_{t \in \Delta(\ell_S,\ell_R)} \tau.\langle t \rangle.\overline{a_{\ell_R}}.\mathbf{0} \mid (E_S):\ell_S \\ (\downarrow a.E_R):\ell_R &\longrightarrow a_{\ell_R}.(E_R):\ell_R \end{aligned}$$

The first rule defines the semantics of non-blocking message sending by translating into *RtCCS* process expressions. The rule transforms $(\ell_R \uparrow a.E) : \ell_S$ into a parallel composition between two processes: $(E) : \ell_S$ and $\sum_{t \in \Delta(\ell_S,\ell_R)} \tau.\ \langle t \rangle.\ \overline{a_{\ell_R}}.\ \mathbf{0}$. The

former corresponds to the subsequent computation. The latter corresponds to the asynchronous message itself. It is received by a process at location ℓ_R after idling for t real time to model the transmission delay of the message, where t is an arbitrary one of the possible communication delay given as set $\Delta(\ell_S, \ell_R)$. The second rule associates an input action name with the location name of the process. As a result, receiver processes can accept only the messages which arrive at their own locations.

Definition 3.3.5 The location translation rule $(E) : \ell$ is recursively defined as follows:

$$(\mathbf{0}):\ell \;\longrightarrow\; \mathbf{0} \tag{1}$$

$$(X):\ell \;\longrightarrow\; X \tag{2}$$

$$(\downarrow a.E):\ell \;\longrightarrow\; a_\ell.(E):\ell \tag{3}$$

$$(E_1 \,+\, E_2):\ell \;\longrightarrow\; (E_1):\ell \,+\, (E_2):\ell \tag{4}$$

$$(\mathbf{rec}\, X : E):\ell \;\longrightarrow\; \mathbf{rec}\, X : (E):\ell \tag{5}$$

$$([t].E):\ell \;\longrightarrow\; [t].(E):\ell \tag{6}$$

$$(\ell' \uparrow a.E):\ell \;\longrightarrow\; \bigoplus_{t\in\Delta(\ell,\ell')} \langle t\rangle.\overline{a_{\ell'}}.\mathbf{0} \,|\, (E):\ell \tag{7}$$

$$(\langle t\rangle.E):\ell \;\longrightarrow\; \langle t\rangle.(E):\ell \tag{8}$$

where ℓ and ℓ' are location names and Δ is a communication delay function. We write $\sum_{i\in\{1,\ldots,n\}} \tau.E_i$ as $\oplus_{i\in\{1,\ldots,n\}} E_i$. We rewrite form $P \setminus N$ to $P \setminus N'$ where $N' \stackrel{\text{def}}{=} \{\, a_\ell \,|\, a \in N,\ \ell \in \mathcal{L}\mathrm{oc}\,\}$. □

Hereafter, we will often omit the $\longrightarrow$ translation if it is directly understood from the context.

Operational Semantics

The location translation rules can map all $RtCCS_A$ expressions into $RtCCS$ ones. The semantics of the translated expressions is defined through the semantics of $RtCCS$.

Definition 3.3.6 $RtCCS_A$ including no $(E):\ell$ is a labeled transition system defined from the transition relation rules given in Definition 2.3.6. □

In asynchronous communication settings, sender processes cannot observe how the messages which they send will be treated by the other processes. We introduce special labeled transitions for asynchronous communication.

Definition 3.3.7 *RtCCS*$_A$ is a labeled transition system redefined from the transition relation rules given in Definition 2.3.6 and the following rules:

$$\frac{-}{\overline{a_\ell}.\mathbf{0} \mid P \xrightarrow[\ell]{\uparrow a} P} \qquad \frac{-}{P \xrightarrow[\ell]{\langle t\rangle \ell' \downarrow a} (\langle t\rangle.\ell' \uparrow a.\mathbf{0}) : \ell \mid P}$$

where $t \in \mathcal{R}^{+0}$. We often denote $P \xrightarrow[\ell]{\langle 0\rangle \ell' \downarrow a} P'$ as $P \xrightarrow[\ell]{\ell' \downarrow a} P'$. We often write $P \xrightarrow[\ell]{\langle t\rangle \epsilon \downarrow a} \langle t\rangle.\overline{a_\ell}.\mathbf{0} \mid P$ instead of the second labeled transition rule. □

We give intuitive meaning of the above transitions.

- $P \xrightarrow[\ell]{\uparrow a} P'$ means that an observer at location ℓ receives message a from P and P behaves like P'.
- $P \xrightarrow[\ell]{\langle t\rangle \ell' \downarrow a} P'$ means that after t real time, an observer at location ℓ sends message a to a process residing at location ℓ' and P becomes P'.

Example 3.3.8 Let $\Delta(\ell_1, \ell_2) = \{4\}$ and $\Delta(\ell_2, \ell_1) = \{5\}$.

(1) *Asynchronous Output*

$$\begin{aligned}
(\langle 3\rangle.\ell_2 \uparrow a.\downarrow b.\mathbf{0}) : \ell_1 &\longrightarrow \langle 3\rangle.(\langle 4\rangle.\overline{a_{\ell_2}}.\mathbf{0} \mid b_{\ell_1}.) \\
&\xrightarrow{\langle 7\rangle} \overline{a_{\ell_2}}.\mathbf{0} \mid b_{\ell_1}.\mathbf{0} \\
&\xrightarrow[\ell_2]{\uparrow a} \mathbf{0} \mid b_{\ell_1}.\mathbf{0}
\end{aligned}$$

(2) *Asynchronous Input*

$$\begin{aligned}
(\downarrow a.\ell_1 \uparrow b.\mathbf{0}) : \ell_2 &\longrightarrow a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \mid \mathbf{0}) \\
&\xrightarrow[\ell_1]{\langle 3\rangle \ell_2 \downarrow a} \langle 3\rangle.(\ell_2 \uparrow a.\mathbf{0}) : \ell_1 \mid a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \mid \mathbf{0}) \\
&\longrightarrow \langle 3\rangle.\langle 4\rangle.\overline{a_{\ell_2}}.\mathbf{0} \mid a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \mid \mathbf{0}) \\
&\xrightarrow{\langle 7\rangle} \overline{a_{\ell_2}}.\mathbf{0} \mid a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \mid \mathbf{0}) \\
&\cdots
\end{aligned}$$

(3) *Asynchronous Communications between Two Processes*

$$\begin{aligned}(\langle 3\rangle.\ell_2\uparrow a.\downarrow b.\mathbf{0}):\ell_1 \,|\, (\downarrow a.\ell_1\uparrow b.\mathbf{0}):\ell_2 &\longrightarrow \langle 3\rangle.\langle 4\rangle.\overline{a_{\ell_2}}.\mathbf{0} \,|\, b_{\ell_1}.\mathbf{0} \,|\, a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \,|\, \mathbf{0}) \\ &\xrightarrow{\langle 7\rangle} \overline{a_{\ell_2}}.\mathbf{0} \,|\, b_{\ell_1}.\mathbf{0} \,|\, a_{\ell_2}.(\langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \,|\, \mathbf{0}) \\ &\xrightarrow{\tau} \mathbf{0} \,|\, b_{\ell_1}.\mathbf{0} \,|\, \langle 5\rangle.\overline{b_{\ell_1}}.\mathbf{0} \,|\, \mathbf{0} \\ &\xrightarrow{\langle 5\rangle} \mathbf{0} \,|\, b_{\ell_1}.\mathbf{0} \,|\, \overline{b_{\ell_1}}.\mathbf{0} \,|\, \mathbf{0} \\ &\xrightarrow{\tau} \mathbf{0} \,|\, \mathbf{0} \,|\, \mathbf{0} \,|\, \mathbf{0}\end{aligned}$$

The above transition relation does not distinguish between observable and unobservable actions. Therefore we give weak transitions as follows:

Definition 3.3.9

- $P \overset{\langle d\rangle\ell'\downarrow a}{\underset{\ell}{\Longrightarrow}} P'$ is defined as $P\ (\xrightarrow{\tau})^*\overset{\langle d\rangle\ell'\downarrow a}{\underset{\ell}{\longrightarrow}}(\xrightarrow{\tau})^*\ P'$
- $P \overset{\uparrow a}{\underset{\ell}{\Longrightarrow}} P'$ is defined as $P\ (\xrightarrow{\tau})^*\overset{\uparrow a}{\underset{\ell}{\longrightarrow}}(\xrightarrow{\tau})^*\ P'$
- $P \overset{\langle t\rangle}{\Longrightarrow} P'$ is defined as $P \overset{\langle t_1\rangle}{\Longrightarrow}\cdots\overset{\langle t_n\rangle}{\Longrightarrow} P'$ where $t = t_1 + \cdots + t_n$

where $t \in \mathcal{R}^{+0}$ and $\xrightarrow{\tau}$ and $\overset{\langle t\rangle}{\Longrightarrow}$ are defined in Definition 2.4.12. □

3.4 Time-Sensitive Bisimulation

This section introduces an equivalence relation for remotely interacting processes.[2]

Definition 3.4.1 A symmetric relation $\mathcal{R} \subseteq (\mathcal{P}\times\mathcal{P})\times 2^{\mathcal{L}\mathrm{oc}}$ is a *remote bisimulation* on location set L ($L \subseteq \mathcal{L}\mathrm{oc}$) if $(P_1, P_2) \in \mathcal{R}^L$ implies, for all $a, b \in \mathcal{A}$, $t \in \mathcal{R}^{+0}$, $\ell_L^a, \ell_L^b \in L$, and $\ell \in \mathcal{L}\mathrm{oc}$;

(i) $\forall {P_1}'\colon P_1 \overset{\ell\downarrow a}{\underset{\ell_L^a}{\Longrightarrow}}\overset{\langle t\rangle}{\Longrightarrow}\overset{\uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_1}'$ *then*

$$\exists {P_2}'\colon P_2 \overset{\ell\downarrow a}{\underset{\ell_L^a}{\Longrightarrow}}\overset{\langle t\rangle}{\Longrightarrow}\overset{\uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_2}' \quad \textit{and} \quad ({P_1}', {P_2}') \in \mathcal{R}^L.$$

(ii) $\forall {P_2}'\colon P_2 \overset{\ell\downarrow a}{\underset{\ell_L^a}{\Longrightarrow}}\overset{\langle t\rangle}{\Longrightarrow}\overset{\uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_2}'$ *then*

$$\exists {P_1}'\colon P_1 \overset{\ell\downarrow a}{\underset{\ell_L^a}{\Longrightarrow}}\overset{\langle t\rangle}{\Longrightarrow}\overset{\uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_1}' \quad \textit{and} \quad ({P_1}', {P_2}') \in \mathcal{R}^L.$$

[2]Strong bisimulation might be regarded as more fundamental than the observation one. However, it is somewhat unnatural to assume that observers have the ability to observe all events at remote locations.

where a, b may be empty action names. P_1 and P_2 are remotely bisimilar with respect on L, written $P_1 \approx_{\mathcal{T}}^{L} P_2$, if there exists a remote bisimulation with respect to L, $\mathcal{R}_L$ such that $(P_1, P_2) \in \mathcal{R}_L$. □

We here state the informal meaning of $P_1 \approx_{\mathcal{T}}^{L} P_2$. An observers at a location in L sends P_1 and P_2 any messages as they like. And then, another observer at a location in L waits messages from P_1 and P_2. If the messages which P_1 and P_2 return are the same and arrives at the observer at the same time, and P_1' and P_2' can simulate to each other in the same way, P_1 and P_2 are remotely equivalent. Note that the observers do not have any sense of messages which arrive at any locations different from their locations.

Proposition 3.4.2 (1) $\approx_{\mathcal{T}}^{L}$ is symmetric, reflexive, and transitive. (2) $\approx_{\mathcal{T}}^{L}$ is the biggest remote bisimulation with respect to L. □

Proof. Definition 3.4.1 almost directly tells us this. □

Example 3.4.3 Let $\ell_1, \ell_2 \in L$,

(1) $(\ell_1 \uparrow a.\ell_2 \uparrow b.E) : \ell \approx_{\mathcal{T}}^{L} (\ell_2 \uparrow b.\ell_1 \uparrow a.E) : \ell$

(2) $(\ell_1 \uparrow a.\ell_3 \uparrow b_1.E) : \ell \approx_{\mathcal{T}}^{L} (\ell_1 \uparrow a.\ell_3 \uparrow b_2.E) : \ell$ where $\ell_3 \notin L$

The first of this example is important to demonstrate an essential characteristic of asynchronous communication.

Proposition 3.4.4 Let $(E_1) : \ell \approx_{\mathcal{T}}^{L} (E_2) : \ell$ and $P_1 \approx_{\mathcal{T}}^{L} P_2$. Then we have the following properties:

(1) $(\downarrow a.E_1) : \ell \approx_{\mathcal{T}}^{L} (\downarrow a.E_2) : \ell$

(2) $(\ell' \uparrow a.E_1) : \ell \approx_{\mathcal{T}}^{L} (\ell' \uparrow a.E_2) : \ell$

(3) $P_1 | Q \approx_{\mathcal{T}}^{L} P_2 | Q$

(4) $P_1 \setminus N \approx_{\mathcal{T}}^{L} P_2 \setminus N$

(5) $\langle t \rangle.P_1 \approx_{\mathcal{T}}^{L} \langle t \rangle.P_2$

where $\Delta(\ell, \ell')$ is constant. □

Proof. Similar to Proposition 2.4.11. □

Proposition 3.4.5

$$P_1 \approx_{\mathcal{T}}^{L_1} P_2 \quad \textit{and} \quad L_2 \subseteq L_1 \qquad \textit{then} \qquad P_1 \approx_{\mathcal{T}}^{L_2} P_2 \qquad \square$$

Proof. We can easily prove this from Definition 3.4.1. $\square$

If two processes are equivalent by an observer with a wider scope (L_1), they can be still equivalent by another observer with a narrow scope ($L_2 \subseteq L_1$).

3.5 Speed-Sensitive Prebisimulation

This section presents an algebraic order relation over distributed processes with respect to their speeds. In asynchronous communication settings, sender processes do not have to synchronize their receiver processes and thus can send messages as soon as they can. Therefore, in the settings real-time processes do not need to completely match their own temporal specification, and need only to deliver messages to their receiver processes at earlier timings than those given in their specification. This reveals that in the verification of asynchronous communicating real-time processes, a speed-sensitive order relation is often more suitable and practical than time-sensitive equivalence relations.

Relating processes with Respect to Speed

Before defining the relations, we first illustrate the basic idea behind them. Suppose two simple processes: $A_1 \stackrel{\text{def}}{=} (\langle 1\rangle.\ell \uparrow a.E) : \ell_A$ and $A_2 \stackrel{\text{def}}{=} (\langle 3\rangle.\ell \uparrow a.E) : \ell_A$, where ℓ is an observer's location. A_1 can send message a to the observer after 1 real time, and A_2 can send the same message after 3 real time. That is, A_1 can send the message *sooner* than A_2. Therefore, we would consider process A_1 to be *faster* than process A_2. We define such a speed-sensitive order relation.

Definition 3.5.1 A binary relation $\mathcal{R} \subseteq (\mathcal{P} \times \mathcal{P}) \times \mathcal{R}^{+0} \times 2^{\mathcal{L}\text{oc}}$ is a *t-speed prebisimulation* on L if $(P_1, P_2) \in \mathcal{R}_t^L$ implies, for all $a, b \in \mathcal{A}$, $d \in \mathcal{R}^{+0}$, $\ell \in \mathcal{L}\text{oc}$, and $\ell_L^a, \ell_L^b \in L$;

(i) $\forall t_1, \forall P_1'$: $P_1 \stackrel{\langle d\rangle \ell\!\downarrow\! a}{\stackrel{}{\Longrightarrow}_{\ell_L^a}} \stackrel{\langle t_1\rangle}{\Longrightarrow} \stackrel{\uparrow b}{\stackrel{}{\Longrightarrow}_{\ell_L^b}} P_1'$ *then*

$$\exists t_2, \exists P_2' \text{:}\ P_2 \stackrel{\langle d\rangle \ell\!\downarrow\! a}{\stackrel{}{\Longrightarrow}_{\ell_L^a}} \stackrel{\langle t_2\rangle}{\Longrightarrow} \stackrel{\uparrow b}{\stackrel{}{\Longrightarrow}_{\ell_L^b}} P_2' \ \textit{and}\ (P_1', P_2') \in \mathcal{R}_{t-t_1+t_2}^L$$

(ii) $\forall t_2, \forall P_2'\colon\ P_2 \stackrel{\langle d\rangle \ell[a}{\Longrightarrow}_{\ell_L^a} \stackrel{\langle t_2\rangle}{\Longrightarrow} \stackrel{\rceil b}{\Longrightarrow}_{\ell_L^b} P_2'$ *then*

$$\exists t_1, \exists P_1'\colon\ P_1 \stackrel{\langle d\rangle \ell[a}{\Longrightarrow}_{\ell_L^a} \stackrel{\langle t_1\rangle}{\Longrightarrow} \stackrel{\rceil b}{\Longrightarrow}_{\ell_L^b} P_1' \ \textit{and}\ (P_1', P_2') \in \mathcal{R}^L_{t-t_1+t_2}$$

where $t \in \mathcal{R}^{+0}$, $L \subseteq \mathcal{L}\mathrm{oc}$ and a, b may be empty names. □

In the above definition, $\mathcal{R}_t^L$ is a family of relations indexed by a non-negative time value t. Intuitively, t is the relative difference between the time of P_1 and that of P_2; that is, it means that P_1 precedes P_2 by t real time.[3] L corresponds to the observers' locations. The following order relation $\preceq_t^L$ starts with a pre-bisimulation indexed by t (i.e., $\mathcal{R}_t^L$) and can change t as the bisimulation proceeds only if $t \geq 0$.

Definition 3.5.2 We let $P_1 \preceq_t^L P_2$ if there exists some t-speed pre-bisimulation on L such that $(P_1, P_2) \in \mathcal{R}_t^L$. We call $\preceq_t^L$ *t-speed-sensitive order* on L. We shall often abbreviate $\preceq_0^L$ as $\preceq^L$.

We here state the informal meaning of $P_1 \preceq_t^L P_2$. We first assume $(P_1, P_2) \in \mathcal{R}_t^L$. This assumption means P_1 precedes P_2 by t real time. An observer at location ℓ_L^a ($\ell_L^a \in L$) sends message a to P_1 after d real time (written as $P_1 \stackrel{\langle d\rangle \ell[a}{\Longrightarrow}_{\ell_L^a}$ in (i)). It also sends the same message to P_2 after d real time (written as $P_2 \stackrel{\langle d\rangle \ell[a}{\Longrightarrow}_{\ell_L^a}$ in (ii)). And then an observer at location ℓ_L^b ($\ell_L^b \in L$) receives return message b from P_1 after t_1 real time (written as $\stackrel{\langle t_1\rangle}{\Longrightarrow} \stackrel{\rceil b}{\Longrightarrow}_{\ell_L^b} P_1'$ in (i)) and from P_2 after t_2 real time (written as $\stackrel{\langle t_2\rangle}{\Longrightarrow} \stackrel{\rceil b}{\Longrightarrow}_{\ell_L^b} P_2'$ in (ii)). If the arrival time of the return message from P_1 is earlier than that from P_2,[4] and if P_1' and P_2' can be successfully observed in $(P_1', P_2') \in \mathcal{R}^L_{t-t_1+t_2}$ in the same way, P_1 and P_2 can perform the same behaviors but P_1 can perform the behaviors *faster* than P_2.

We show several algebraic properties of the order relation below.

Proposition 3.5.3 Let $P, P_1, P_2, P_3 \in \mathcal{P}$, $t_1, t_2 \in \mathcal{R}^{+0}$, and $L \subseteq \mathcal{L}\mathrm{oc}$ then,

(1) $\quad P \preceq_0^L P$

[3] This means that the performance of P_1 is at most t real time faster than that of P_2.

[4] Note that P_2 already precedes P_1 by t real time. Thus, the relative difference between the arrival timing of the message from P_2 and that from P_1 is $t - t_1 + t_2$.

(2) $P_1 \leq^L_{t_1} P_2$ *and* $P_2 \leq^L_{t_2} P_3$ *then* $P_1 \leq^L_{t_1+t_2} P_3$ □

Proof. Definition 3.5.1 and 3.5.2 almost directly tells us this. □

From these results, we see that $P \leq^L P$ and that if $P_1 \leq^L P_2$ and $P_2 \leq^L P_3$ then $P_1 \leq^L P_3$. Hence, $\leq^L$ is a preorder relation.

Proposition 3.5.4 Let $P_1, P_2 \in \mathcal{P}$, $L, L_1, L_2 \subseteq \mathcal{L}\mathrm{oc}$, and $t, t_1, t_2 \in \mathcal{R}^{+0}$ then,

(1) $P_1 \leq^L_{t_1} P_2$ *and* $t_1 \leq t_2$ *then* $P_1 \leq^L_{t_2} P_2$
(2) $P_1 \leq^{L_1}_{t} P_2$ *and* $L_2 \subseteq L_1$ *then* $P_1 \leq^{L_2}_{t} P_2$ □

Proof. Easy application of Definition Definition 3.5.1 and 3.5.2. □

The first half of the above proposition means that P_1 can perform at most t_1 real time faster than P_1. Then, P_1 can still perform at most t_2 real time faster than P_1, where $t_1 \leq t_2$. The second half means that two processes ordered by an observer having a scope (L_1) can be still ordered by another observer having a more limited scope (L_2). It is very convenient to develop a precongruence with respect to speeds in order to guarantee the substitutability between two ordered processes.

Proposition 3.5.5 Let $(E_1) : \ell, (E_2) : \ell, P_1, P_2 \in \mathcal{P}$ such that $(E_1) : \ell \leq^L (E_2) : \ell$ and $P_1 \leq^L P_2$ where $L \subseteq \mathcal{L}\mathrm{oc}$ and $t \in \mathcal{R}^{+0}$ then,

(1) $(\downarrow a.E_1) : \ell \leq^L (\downarrow a.E_2) : \ell$
(2) $(\ell' \uparrow a.E_1) : \ell \leq^L (\ell' \uparrow a.E_2) : \ell$
(3) $(\langle t \rangle.E_1) : \ell \leq^L (\langle t \rangle.E_2) : \ell$
(4) $P_1 \setminus N \leq^L P_2 \setminus N$

where $\Delta(\ell, \ell')$ is constant. □

Proof. As Proposition 2.4.11. □

However, there is an undesirable problem in giving a pre-congruence with respect to parallel composition. Suppose three processes: $A_1 \stackrel{\mathrm{def}}{=} \langle 1 \rangle. \uparrow a.\mathbf{0}$, $A_2 \stackrel{\mathrm{def}}{=} \langle 4 \rangle. \uparrow a.\mathbf{0}$ and $B \stackrel{\mathrm{def}}{=} ((a.B' + \downarrow b.B'') \,|\, \langle 2 \rangle. \uparrow b.\mathbf{0}) \setminus \{b\}$ where we ignore the notion of location can be ignored for simplify. We clearly have $A_1 \leq^L A_2$ but cannot expect $A_1|B \leq^L A_2|B$. This reveals that a slower process cannot always be replaced by a faster process in

a parallel composition with other processes. This anomaly is traced to contexts that restrict the capability to execute a particular computation due to the passage of time, e.g. *timeout* handling.

In order to define a rational pre-congruence with respect to speed, we here define another order relation which is a little more strict than the $\leq_t^L$ relation.

Definition 3.5.6 A binary relation $\mathcal{R} \subseteq (\mathcal{P} \times \mathcal{P}) \times \mathcal{R}^{+0} \times 2^{\mathcal{L}\mathrm{oc}}$ is a t *strict-speed prebisimulation* on L if $(P_1, P_2) \in \mathcal{R}_t^L$ implies, for all $a, b \in \mathcal{A}$, $d_1, d_2 \in \mathcal{R}^{+0}$ such that $d_1 \leq d_2 + t$, $\ell \in \mathcal{L}\mathrm{oc}$, and $\ell_L^a, \ell_L^b \in L$;

(i) $\forall t_1, \forall {P_1}' \colon P_1 \overset{\langle d_1 \rangle \ell \lfloor a}{\underset{\ell_L^a}{\Longrightarrow}} \overset{\langle t_1 \rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_1}'$ *then*

$\exists t_2, \exists {P_2}' \colon P_2 \overset{\langle d_2 \rangle \ell \lfloor a}{\underset{\ell_L^a}{\Longrightarrow}} \overset{\langle t_2 \rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_2}'$ *and* $({P_1}', {P_2}') \in \mathcal{R}_{t-t_1+t_2}^L$

(ii) $\forall t_2, \forall {P_2}' \colon P_2 \overset{\langle d_2 \rangle \ell \lfloor a}{\underset{\ell_L^a}{\Longrightarrow}} \overset{\langle t_2 \rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_2}'$ *then*

$\exists t_1, \exists {P_1}' \colon P_1 \overset{\langle d_1 \rangle \ell \lfloor a}{\underset{\ell_L^a}{\Longrightarrow}} \overset{\langle t_1 \rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell_L^b}{\Longrightarrow}} {P_1}'$ *and* $({P_1}', {P_2}') \in \mathcal{R}_{t-t_1+t_2}^L$

where $t \in \mathcal{R}^{+0}$ and $L \subseteq \mathcal{L}\mathrm{oc}$ and a, b may be an empty name. □

Definition 3.5.7 We let $P_1 \lhd_t^L P_2$ if there exists some t strict-speed prebisimulation on L such that $(P_1, P_2) \in \mathcal{R}_t^L$. We call $\lhd_t^L$ *t-strict-speed order* on L. We shall often abbreviate $\leq_0^L$ as $\leq^L$.

This relation is basically similar to $\leq_t^L$ except that the observer is a little strict. That is, whereas the observer of $\leq_t^L$ sends a message to the concerned processes after the amount of the passage of time, the observer of $\lhd_t^L$ sends a message to the second (slower) process arbitrarily later than to the (faster) process. Also, to allow easier discussion hereafter, we define a restricted expression below.

Definition 3.5.8 Let $P \in \mathcal{P}$. Then, if $P \ \lhd_t^L \ P$, we call P a *sound* expression on L. We denote $\mathcal{P}'$ to the set of all the sound expressions.

We show several algebraic properties of the order relation below.

Proposition 3.5.9 Let $P, P_1, P_2, P_3 \in \mathcal{P}'$, $t_1, t_2 \in \mathcal{R}^{+0}$, and $L \subseteq \mathcal{L}\mathrm{oc}$ then,

(1) $P \ \lhd_0^L \ P$

(2) $P_1 \trianglelefteq^L_{t_1} P_2$ *and* $P_2 \trianglelefteq^L_{t_2} P_3$ *then* $P_1 \trianglelefteq^L_{t_1+t_2} P_3$ □

Proof. Definition 3.5.6 and 3.5.7 almost directly tells us this. □

From these results, we see that $P \trianglelefteq^L P$ and that if $P_1 \leq^L P_2$ and $P_2 \trianglelefteq^L P_3$ then $P_1 \trianglelefteq^L P_3$. Hence, $\trianglelefteq^L$ is a preorder relation on $\mathcal{P}'$.

Proposition 3.5.10 Let $P_1, P_2 \in \mathcal{P}$, $L \subseteq \mathcal{L}\mathrm{oc}$, and $t \in \mathcal{T}$ then,

(1) *If* $P_1 \trianglelefteq^L_t P_2$ *then* $P_1 \leq^L_t P_2$

(2) If in Proposition 3.5.4 and 3.5.5 every $\leq^L_t$ is replaced by $\trianglelefteq^L_t$, the propositions still hold. □

The first above result shows that $\leq^L_t$ at least includes $\trianglelefteq^L_t$.

We will show that $\trianglelefteq^L_t$ is precongruent with respect to a parallel composition. We need a lemma before proving it.

Lemma 3.5.11 Let $P_1, P_2, Q_1, Q_2, P_1|Q_1, P_2|Q_1 \in \mathcal{P}'$, Then,

$$P_1 \trianglelefteq^L_t P_2 \quad \textit{and} \quad Q_1 \trianglelefteq^L_t Q_2 \quad \textit{then} \quad P_1 \,|\, Q_1 \trianglelefteq^L_t P_2 \,|\, Q_2$$

where we assume $|P_1|_{Loc}, |P_2|_{Loc}, |Q_1|_{Loc}, |Q_1|_{Loc} \subseteq L$ □

Proof. It is enough to show that $\mathcal{R}^L_t = \{\, (P_1|Q_1, P_2|Q_2) \mid P_1 \trianglelefteq^L_t P_2 \text{ and } Q_1 \trianglelefteq^L_t Q_2 \,\}$ is strict-speed prebisimulation ($t \geq 0$). Let $P_1|Q_1 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} \overset{\langle t_1\rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell^b_L}{\Longrightarrow}} P_1'|Q_1'$. It is enough to find $P_2|Q_2$ such that $P_2|Q_2 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} \overset{\langle t_2\rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell^b_L}{\Longrightarrow}} P_2'|Q_2'$ and $(P_1'|Q_1', P_2'|Q_2') \in \mathcal{R}^L_{t-t_1+t_2}$. There are three cases:

Case 1 Let $P_1|Q_1 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} P_1'|Q_1'$ then,

Subcase 1.1 Let $P_1 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} P_1'$ and $Q_1 \equiv Q_1'$. Then, from $P_1 \trianglelefteq^L_t P_2$, we have $\forall d_2 \geq d_1 + t$, $P_2 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} P_2'$ with $P_1' \trianglelefteq^L_{t-t_1+t_2} P_2'$. By choosing Q_2 to be Q_2', $(P_1'|Q_1', P_2'|Q_2') \in \mathcal{R}^L_{t-t_1+t_2}$.

Subcase 1.2 Let $P_1 \equiv P_1'$ $Q_1 \overset{\langle d_1\rangle \ell\!\!\downarrow a}{\underset{\ell^a_L}{\Longrightarrow}} Q_1'$ then, similar.

Case 2 Let $P_1|Q_1 \overset{\langle t_1\rangle}{\Longrightarrow} \overset{\Uparrow b}{\underset{\ell^b_L}{\Longrightarrow}} P_1'|Q_1'$ then,

Subcase 2.1 Let $P_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_L^b}{\Longrightarrow}} P_1'$. and $Q_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} Q_1'$. Then, from $P_1 \trianglelefteq_t^L P_2$, we have $\exists t_2 \geq t_1 + t$, $P_2 \stackrel{\langle t_2 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_L^b}{\Longrightarrow}} P_2'$ with $P_1' \trianglelefteq_{t-t_1+t_2}^L P_2'$. Also, from $Q_1 \trianglelefteq_t^L Q_2$, we have $\forall \dot{t_2} \geq t_1 + t$, $Q_2 \stackrel{\langle \dot{t_2} \rangle \epsilon \not a}{\underset{\ell_Q}{\Longrightarrow}} Q_2'$ with $Q_1' \trianglelefteq_{t-t_1+\dot{t_2}}^L \dot{Q_2}'$. By choosing $\dot{t_2}$ to be t_2 and $\dot{Q_2}'$ to be Q_2', we have $(P_1'|Q_1', P_2'|Q_2') \in \mathcal{R}_{t-t_1+t_2}^L$.
Subcase 2.2 Let $P_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} P_1'$ and $Q_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_L^b}{\Longrightarrow}} Q_1'$ then, similar.

Case 3 Let $P_1|Q_1 \stackrel{\langle t_2 \rangle}{\Longrightarrow} P_1'|Q_1'$ then,

Subcase 3.1 Let $P_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} P_1'$, and $P_2 \stackrel{\langle t_1 \rangle}{\Longrightarrow} P_2'$. Trivial.
Subcase 3.2 Let $P_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_Q}{\Longrightarrow}} P_1'$, $Q_1 \stackrel{\langle t_1 \rangle \epsilon \not a}{\underset{\ell_Q}{\Longrightarrow}} Q_1'$, and $\ell_Q \in |Q_1|_{Loc} (\equiv |Q_2|_{Loc})$ Then, from $P_1 \trianglelefteq_t^L P_2$, we have $\exists t_2 \geq t_1 + t$, $P_2 \stackrel{\langle t_2 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_Q}{\Longrightarrow}} P_2'$ with $P_1' \trianglelefteq_{t-t_1+t_2}^L P_2'$. Also, from $Q_1 \trianglelefteq_t^L Q_2$, we have $\forall \dot{t_2} \geq t_1 + t$, $Q_2 \stackrel{\langle \dot{t_2} \rangle \epsilon \not a}{\underset{\ell_Q}{\Longrightarrow}} Q_2'$ with $Q_1' \trianglelefteq_{t-t_1+\dot{t_2}}^L \dot{Q_2}'$. By choosing $\dot{t_2}$ to be t_2 and $\dot{Q_2}'$ to be Q_2', we have $(P_1'|Q_1', P_2'|Q_2') \in \mathcal{R}_{t-t_1+t_2}^L$.
Subcase 3.3 Let $P_1 \stackrel{\langle t_1 \rangle \epsilon \not a}{\underset{\ell_P}{\Longrightarrow}} P_1'$, $Q_1 \stackrel{\langle t_1 \rangle}{\Longrightarrow} \stackrel{\not b}{\underset{\ell_P}{\Longrightarrow}} Q_1'$, and $\ell_P \in |P_1|_{Loc} (\equiv |P_2|_{Loc})$ then, similar.

By a symmetric argument, we complete the proof. □

Theorem 3.5.12 Let $P_1, P_2, Q, P_1|Q, P_2|Q \in \mathcal{P}'$.

$$P_1 \trianglelefteq^L P_2 \qquad \textit{and} \qquad P_1 \,|\, Q \trianglelefteq^L P_2 \,|\, Q$$

where we assume $|P_1|_{Loc}, |P_2|_{Loc}, |Q|_{Loc} \subseteq L$ □

Proof. By Lemma 3.5.11, we can easily prove this theorem. □

Intuitively, the above result tells that a parallel composition between the faster processes can really perform faster than one between the slower ones. That is, a system when embedding the faster processes can really perform faster than when embedding the slower ones.

Proposition 3.5.13 Let $P_1, P_2 \in \mathcal{P}'$, Δ_1, Δ_2 such that $\forall \ell \in L, \forall \ell' \in |P_1|_{Loc} \cup |P_2|_{Loc}$: $\mathbf{max}\Delta_1(\ell, \ell') \leq \mathbf{min}\Delta_2(\ell, \ell')$ and $\mathbf{max}\Delta_1(\ell', \ell) \leq \mathbf{min}\Delta_2(\ell', \ell)$,

$$P_1 \trianglelefteq^L P_2 \quad \textit{assuming} \quad \Delta_1 \qquad \textit{then} \qquad P_1 \trianglelefteq^L P_2 \quad \textit{assuming} \quad \Delta_2$$ □

This proposition shows that a slower process is still slower even from an observer residing at a further location. However, the reverse implication of the proposition does not hold; That is, for all Δ_1, Δ_2 such that $\forall \ell \in L, \ell' \in |P_1|_{Loc} \cup |P_2|_{Loc} : \mathbf{max}\Delta_1(\ell, \ell') \geq \mathbf{min}\Delta_2(\ell, \ell')$ and $\mathbf{max}\Delta_1(\ell', \ell) \geq \mathbf{min}\Delta_2(\ell', \ell)$ then, from $P_1 \leq^L P_2$ assuming Δ_1, $P_1 \trianglelefteq^L P_2$ assuming Δ_2 is not usually derived. This is impossible because the observer of $\leq^L$ cannot notice any difference between its ordered processes before it receives their first messages.

Remark The expressive power of sound expressions is weaker than that of $\mathcal{P}$ because any expressions which contain the anomalous contexts mentioned above are no longer definable, for example $((a \downarrow .A_1 + b \downarrow .A_2) \,|\, \langle 2 \rangle.\ell \uparrow b.\mathbf{0}) \setminus \{b\} : \ell$ and $(\langle 2 \rangle.a \downarrow .\mathbf{0} \,|\, \langle 4 \rangle.a \downarrow .\mathbf{0}) : \ell$. However, we insist that this is not an unreasonable restriction because we never lose the expressive capability for time-dependent operations which make some other event executable due to the passing of time, in particular this restriction never affects any required expressiveness of the language in describing the real-time systems which contain no timeout handling, including hard real-time systems. On the contrary, every expression in $\mathcal{P}$ can satisfy all the propositions presented in this section, if we alter the fourth rule of Definition 2 into "$P_1 \xrightarrow{\langle t \rangle} P_1'$ and $P_2 \xrightarrow{\langle t \rangle} P_2'$ imply $P_1|P_2 \xrightarrow{\langle t \rangle} P_1'|P_2'$".[5] However, this alternation allows an executable communication to be suspended for arbitrary periods of time.

3.6 Examples

We show some examples to demonstrate the utility of the calculus and the relations presented in this chapter.

Example of Description

The following example illustrates how to describe distributed processes in $RtCCS_A$.

Example 3.6.1 We describe a simple communication protocol for an unreliable communication network. The protocol consists of a sender process at location ℓ_S and a

[5] We leave further details of this alternative semantics to another paper [67].

receiver process at location ℓ_R. The sender sends a message to the receiver ($\ell_R \uparrow send$) and waits for an acknowledgment ($\downarrow ack$). If it cannot receive the acknowledgment within 8.0 real time, it retransmits the message. The receiver accepts the message ($\downarrow send$) and then returns an acknowledgment ($\ell_S \uparrow ack$). These processes are described as follows:

$$\begin{aligned} Sender : \ell_S &\stackrel{\text{def}}{=} ((\ell_R \uparrow send \,.\, (\downarrow ack.\mathbf{0} + \downarrow timeout \,.\, Sender) \mid \\ &\qquad\qquad \langle 8.0 \rangle \,.\, \ell_S \uparrow timeout \,.\, \mathbf{0}) \setminus \{timeout\}) : \ell_S \\ Receiver : \ell_R &\stackrel{\text{def}}{=} \downarrow send \,.\, \ell_S \uparrow ack \,.\, \mathbf{0} : \ell_R \end{aligned}$$

We assume that the communication from ℓ_S to ℓ_R takes 3.0 ± 1.0 real time and may occasionally fail, and that from ℓ_R to ℓ_S takes 2.0 ± 1.0 real time and may occasionally fail.

$$\begin{aligned} \Delta(\ell_S, \ell_R) &= \{2.0, \ldots, 4.0\} \cup \{\infty\} \\ \Delta(\ell_R, \ell_S) &= \{1.0, \ldots, 3.0\} \cup \{\infty\} \\ \Delta(\ell_S, \ell_S) &= \{0\} \end{aligned}$$

where ∞ corresponds to a communication failure. By using rule $(E) : \ell$, these processes are expanded as follows:

$$\begin{aligned} (Sender) : \ell_S \longrightarrow \cdots \longrightarrow \oplus_{t \in \Delta(\ell_S, \ell_R)} \langle t \rangle . \overline{send_{\ell_R}}.\mathbf{0} \mid (ack_{\ell_S}.\mathbf{0} + timeout_{\ell_S}.(Sender) : \ell_S) \mid \\ \langle 8.0 \rangle . \overline{timeout_{\ell_S}}.\mathbf{0}) \setminus \{timeout_{\ell_S}\} \\ (Receiver) : \ell_R \longrightarrow \cdots \longrightarrow send_{\ell_R}.\langle 2.0 \rangle . \overline{ack_{\ell_S}}.\mathbf{0} \end{aligned}$$

By expanding $(Sender : \ell_S \mid Receiver : \ell_R) \setminus \{send, ack\}$ in the above transition, we can strictly analyze both the behavioral properties and the temporal properties of the entire system. When $\Delta(\ell_S, \ell_R)$ is assumed to be 3.0 real time and $\Delta(\ell_R, \ell_S)$ to be 2.0 real time, we show interactions between them as follows:

$$\begin{aligned} &(Sender : \ell_S \mid Receiver : \ell_R) \setminus \{send, ack\} \\ &\xrightarrow{\tau} (\langle 3.0 \rangle . \overline{send_{\ell_R}}.\mathbf{0} \mid (ack_{\ell_S}.\mathbf{0} + timeout_{\ell_S}.(Sender) : \ell_S) \mid \\ &\qquad \langle 8.0 \rangle . \overline{timeout_{\ell_S}}.\mathbf{0}) \setminus \{timeout\} \mid send_{\ell_R}.\langle 2.0 \rangle . \overline{ack_{\ell_S}}.\mathbf{0}) \setminus \{send, ack\} \\ &\xrightarrow{\langle 3.0 \rangle} ((\overline{send_{\ell_R}}.\mathbf{0} \mid (ack_{\ell_S}.\mathbf{0} + timeout_{\ell_S}.(Sender) : \ell_S) \mid \end{aligned}$$

$$
\begin{array}{rl}
 & \langle 5.0\rangle.\overline{timeout_{\ell_S}}.\mathbf{0}) \setminus \{timeout\} \mid send_{\ell_R}.\langle 2.0\rangle.\overline{ack_{\ell_S}}.\mathbf{0}) \setminus \{send, ack\} \\
\xrightarrow{\tau} & ((ack_{\ell_S}.\mathbf{0} + timeout_{\ell_S}.(Sender) : \ell_S) \mid \\
 & \langle 5.0\rangle.\overline{timeout_{\ell_S}}.\mathbf{0}) \setminus \{timeout\} \mid \langle 2.0\rangle.\overline{ack_{\ell_S}}.\mathbf{0}) \setminus \{send, ack\} \\
\xrightarrow{\langle 2.0\rangle} & ((ack_{\ell_S}.\mathbf{0} + timeout_{\ell_S}.(Sender) : \ell_S) \mid \\
 & \langle 5.0\rangle.\overline{timeout_{\ell_S}}.\mathbf{0}) \setminus \{timeout\} \mid \overline{ack_{\ell_S}}.\mathbf{0}) \setminus \{send, ack\} \\
\xrightarrow{\tau} & (\mathbf{0} \mid \mathbf{0}) \setminus \{send, ack\}
\end{array}
$$

Example of Verification

For the remainder of this section we will present an example of the verification of distributed processes to demonstrate how the order relation works.

Example 3.6.2 We consider two printing service systems in a distributed system. The first system consists of two remotely located processes: a printer process ($Printer_1$) at location ℓ_P and a console process ($Console_1$) at location ℓ_C. We denote the location of their environment as ℓ.

- Upon reception of a message ($\downarrow print$) from the environment, the console process queries the status of the printer process ($\ell_P \uparrow status$). If it receives a permission to send data ($\downarrow idle$), it sends data to the printer ($\ell_P \uparrow data$) after an internal execution for 5 real time and waits for a completion notice of the print ($\downarrow end$). After receiving the notice, it sends a message to the environment ($\ell \uparrow ok$).
- The printer process waits for a query message ($\downarrow status$) and then returns its status ($\ell_C \uparrow idle$) after 5 real time and waits for data transmission ($\downarrow data$). Since it takes 60 real time to print the data, it returns a print completion notice ($\ell_C \uparrow end$) after 60 real time.

The two processes are described as follows:

$$
\begin{array}{rcl}
Console_1 : \ell_C & \stackrel{\text{def}}{=} & (\downarrow print \,.\, \ell_P \uparrow status \,.\, \downarrow idle \,.\, \langle 5\rangle \,.\, \ell_P \uparrow data \,.\, \downarrow end \,.\, \ell \uparrow ok \,.\, \mathbf{0}) : \ell_C \\
Printer_1 : \ell_P & \stackrel{\text{def}}{=} & (\downarrow status \,.\, \langle 5\rangle \,.\, \ell_C \uparrow idle \,.\, \downarrow data \,.\, \langle 60\rangle \,.\, \ell_C \uparrow end \,.\, Printer_1) : \ell_P
\end{array}
$$

The first system is described as a parallel composition of the processes as follows:

$$(Printer_1 : \ell_P \mid Console_1 : \ell_C) \setminus N_1 \qquad \text{where } N_1 \stackrel{\text{def}}{=} \{status, idle, data, end\}$$

The second system consists of three remotely located processes: a printer process ($Printer_2$) at location ℓ_P, an agent process ($Agent$) at location ℓ_A, and a console process ($Console_2$) at location ℓ_C. The agent process interacts with the printer process and the console process. The printer process is identical to that in the first system except for its message destinations.

- Upon reception of a message ($\downarrow print$), the console process sends the agent process a printing request ($\ell_P \uparrow req$) and then waits for a notice of printing start ($\downarrow started$). After receiving the notice, it sends a message to the environment ($\ell \uparrow ok$).

- The agent process receives a printing request message ($\downarrow req$). After an internal execution of 20 real time, it queries the printer about its status ($\ell_P \uparrow status$). If it receives the status ($\downarrow idle$), it sends data to the printer ($\ell_P \uparrow data$) after an internal execution of 20 real time, and then sends a message to the console ($\ell_C \uparrow started$). After that, it waits for next print request ($\downarrow req$) while waiting for a print completion notice ($\downarrow end$) from the printer process.

These processes are described as follows:

$$
\begin{aligned}
Console_2 : \ell_C &\stackrel{\text{def}}{=} (\downarrow print \,.\, \ell_A \uparrow req \,.\, \downarrow started \,.\, \ell \uparrow ok \,.\, \mathbf{0}) : \ell_C \\
Agent : \ell_A &\stackrel{\text{def}}{=} (\downarrow req \,.\, \langle 20 \rangle \,.\, \ell_P \uparrow status \,.\, \downarrow idle \,.\, \langle 20 \rangle \,. \\
&\qquad\qquad (\ell_P \uparrow data \,.\, \ell_C \uparrow started \,.\, \downarrow end \,.\, \mathbf{0} \mid Agent)) : \ell_A \\
Printer_2 : \ell_P &\stackrel{\text{def}}{=} (\downarrow status \,.\, \langle 5 \rangle \,.\, \ell_A \uparrow idle \,.\, \downarrow data \,.\, \langle 60 \rangle \,.\, \ell_A \uparrow end \,.\, Printer_2) : \ell_P
\end{aligned}
$$

The whole second system is described as a parallel composition of the three process as follows:

$$
\begin{aligned}
&(Printer_2 : \ell_P \mid Agent : \ell_A \mid Console_2 : \ell_C) \setminus N_2 \\
&\qquad\qquad \text{where } N_2 \stackrel{\text{def}}{=} \{req, status, idle, started, data, end\}
\end{aligned}
$$

We here compare the performances of these systems. We first assume that the communication delay between ℓ_C and ℓ_P is 40 ± 1 real time, that between ℓ_A and ℓ_P is 20 ± 1 real time, and that between ℓ_C and ℓ_A is 30 ± 1 real time.

$$\begin{array}{rcl} \Delta(\ell_P, \ell_C) & = & \Delta(\ell_C, \ell_P) = \{39.0, \ldots, 41.0\} \\ \Delta(\ell_A, \ell_P) & = & \Delta(\ell_P, \ell_A) = \{19.0, \ldots, 21.0\} \\ \Delta(\ell_C, \ell_A) & = & \Delta(\ell_A, \ell_C) = \{29.0, \ldots, 31.0\} \end{array}$$

We also assume that the other communication channels do not exist. By using the timed order relation, the two systems are related as follows:

$$\begin{array}{l} (\mathit{Printer}_1 : \ell_P \mid \mathit{Console}_1 : \ell_C) \setminus N_1 \\ \quad \trianglelefteq^L \quad (\mathit{Printer}_2 : \ell_P \mid \mathit{Agent} : \ell_A \mid \mathit{Console}_2 : \ell_C) \setminus N_2 \\ \qquad\qquad \text{where } L \subseteq \mathcal{L}\text{oc such that } \ell, \ell_A, \ell_C, \ell_P \in L \end{array}$$

The above result shows that the two systems are behaviorally equivalent but that the first system can perform faster than the second one.

Also, by using the timed order relation, we can verify that the systems satisfy their specification. For example, let $\mathit{Spec} : \ell_C$ be a specification for the systems.

$$\mathit{Spec} : \ell_C \stackrel{\text{def}}{=} \downarrow \mathit{req} \,.\, \langle 300 \rangle \,.\, \ell \uparrow \mathit{ok} \,.\, \mathbf{0} : \ell_C$$

By using Definition 3.4.1 we conclude that:

$$\begin{array}{rcl} (\mathit{Printer}_1 : \ell_P \mid \mathit{Console}_1 : \ell_C) \setminus N_1 & \trianglelefteq^L & \mathit{Spec} : \ell_C \\ (\mathit{Printer}_2 : \ell_P \mid \mathit{Agent} : \ell_A \mid \mathit{Console}_2 : \ell_C) \setminus N_2 & \trianglelefteq^L & \mathit{Spec} : \ell_C \end{array}$$

The above inequalities show that the two systems can execute the behaviors given in the specification faster than the required execution time in the specification.

Next we will consider a reconstruction (or an improvement) of the second system. Suppose another agent process (a new implementation) described as the following Agent':

$$\begin{array}{rcl} \mathit{Agent}' & \stackrel{\text{def}}{=} & (\downarrow \mathit{req} \,.\, \langle 5 \rangle \,.\, \ell_P \uparrow \mathit{status} \,.\, \downarrow \mathit{idle} \,.\, \langle 5 \rangle \,. \\ & & \qquad (\ell_P \uparrow \mathit{data} \,.\, \ell_C \uparrow \mathit{started} \,.\, \downarrow \mathit{end} \,.\, \mathbf{0} \mid \mathit{Agent}') \end{array}$$

When we apply the two agent expressions to Definition 3.4.1, we know the following result:

$$\mathit{Agent}' : \ell_A \quad \trianglelefteq^L \quad \mathit{Agent} : \ell_A$$

The above inequality tells that *Agent* and *Agent*′ are behaviorally equivalent and *Agent* can perform faster than *Agent*′. In order to improve the performance of the whole second system, we replace *Agent*′ by *Agent* in the system. We will need to verify that the new second system really performs faster than the old one without changing any behavioral properties. From the inequality and Proposition 3.5.12, we can directly know the following desired fact:

$$\begin{aligned}(\mathit{Printer}_2 : \ell_P \mid \mathit{Agent}' : \ell_A \mid \mathit{Console}_2 : \ell_C) \backslash N_2 \\ \trianglelefteq^L \quad (\mathit{Printer}_2 : \ell_P \mid \mathit{Agent} : \ell_A \mid \mathit{Console}_2 : \ell_C) \backslash N_2\end{aligned}$$

This demonstrates that *Agent*′ can be behaviorally substituted for *Agent* in the second system and that the new second system can perform faster than the old one.[6] Moreover, by using the timed order relation we can compare the new system and the first system.

$$\begin{aligned}(\mathit{Printer}_2 : \ell_P \mid \mathit{Agent}' : \ell_A \mid \mathit{Console}_2 : \ell_C) \setminus N_2 \\ \trianglelefteq^L \quad (\mathit{Printer}_1 : \ell_P \mid \mathit{Console}_1 : \ell_C) \setminus N_1\end{aligned}$$

This holds for $\leq^L$ as well as $\trianglelefteq^L$.

3.7 Concluding Remarks

In this chapter, we proposed a process calculus for distributed processes. The calculus is characterized by having the ability to express delay, asynchrony, and locality in communication. It allows us to analyze the temporal and behavioral properties of asynchronous interactions among remotely located processes.

Also, we defined a time-sensitive equivalence relation and speed-sensitive order relations. The equivalence can equate two processes when observers at particular locations cannot distinguish between them in the messages that arrive at the observers and in the arrival timings of the messages. The order relations can decide whether two processes are behaviorally equivalent and whether one of them can perform its behaviors faster than the other. One of the order relations is proved to be precongruent with respect to parallel composition. By using the relations, we can guarantee that a

[6] Note that all the expressions in this example are sound.

faster local process can be functionally substituted for a slower one in a distributed system and that the system embedding the faster one can perform even faster than the system embedding the slower one. It offers a foundation for proving the correctness and the reusability of asynchronous interacting remotely located processes.

Chapter 4

Locality in Time

Communication delay is one of the most characteristic property of distributed systems. The previous chapter studied how the delay affects communications among distributed processes. Also, the delay may feature temporal aspects of distributed computing systems as compared to other computing systems. It prevents every processor from sharing any unique global time reference. As a result, each processor must follow its own physical clock, for example crystal-base clock, but such a clock is usually inaccurate and different from each other. This chapter addresses a formalization of distributed processes following multiple inaccurate clocks.[1]

4.1 Introductory Remarks

Distributed systems often need real-time facilities to manage time critical responses. These facilities are materialized by physical clocks on local processors instead of any global clock. However, such physical clocks are not perfect. Their measurement rates are different and may drift. Indeed, many clock synchronization techniques for compensating for clock drift have already been explored, for example see [40, 43, 45, 48]. However, these techniques are not available in all distributed systems. Even when such techniques are supported, if the required precision of time information is finer than the intervals of clock synchronizations, differences and uncertainties among clocks may lead cooperations among distributed processes to timing failures.

[1]This chapter is a modified version of an article that has been published earlier in [66, 68].

We need to analyze the influences of the inaccuracies of clocks upon the behavioral and temporal properties of distributed systems. A number of computing models for distributed systems have already been proposed. However, most of them are essentially formulated based on the assumption of the existence of a global clock. Therefore, they cannot reason about distributed processes following inaccurate clocks.

This chapter aims to develop a new process calculus for describing distributed processes following different clocks. The new calculus is formulated by extending the calculus developed in Chapter 2 with the ability to express multiple inaccurate clocks quantitatively.

4.2 Basic Framework

The new calculus is an extension of the calculus studied in Chapter 2 with the ability to express multiple inaccurate clocks. It is called $RtCCS_L$. Below we briefly survey basic ideas of the new calculus.

The Passage of Time

We assume that the passage of physical time in every processor elapses at the same rate.[2] The coordinate of the passage of physical time is dense and called *real time domain*. We assume that the precision of all local clocks can be specified based on the time coordinate. In the calculus, the coordinate plays a role of the conceptual time standard of when interpreting processes following local clocks.[3]

Local Clocks

Each local clock is introduced as an entity to measure the passage of physical time according to its own measurement speed.[4] Only the parameters of the delay operator and time restriction operator in *RtCCS* can correspond to such clocks. The parameters are decreased exactly by the amount of the passage of time. However, the

[2]This assumption is natural and reasonable, because from Einstein's Special Relativity, if relative motion of all processors is negligible as compared to the speed of light, physical time in every processor passes at the same speed.

[3]Note that this assumption does not imply the existence of any actual global clock.

[4]The word *timer* is considered to be better to refer these devices in the usual sense. However, we will use the word *clock* to refer them, because the word *clock* is usually used to refer to these devices in computer systems.

measurement speeds of real clocks, — the decrease speeds of the parameters — are not the same as the speed of the passage of physical time by virtue of their measurement errors. Therefore, we introduce a mapping that associates each clock's measurement speed with the speed of the passage of physical time and vice versa. The mapping specifies the precision of each clock. The mapping is given as a non-deterministic function to model an inaccurate clock whose measurement rate drifts within a given bound and whose errors in reading its measurement result.

Finally, we explain our way of modeling distributed processes. In the calculus, distributed processes are described as expressions with respect to their own local time coordinates. And then, the expressions are translated into ones on the real time domain by using special syntactic mappings, written as $(E)[\Phi]$. The mapping translates all local time values in the expressions into ones on the real time domain. The translated expressions can be interpreted as *RtCCS* expressions.

4.3 Definition

This section gives the definition of $RtCCS_L$. A few preliminary definitions are needed before giving the definition. We first define notations for clocks.

Notations

The passage of physical time are dense, written as $\mathcal{R}^{+0}$, whereas time values measured by local clocks are discrete.

Definition 4.3.1 Let $\mathcal{T}_i$ be the set of time values measured by i-th clock, called i-th *local time domain*,

$$\mathcal{T}_i \stackrel{\text{def}}{=} \mathcal{N} \cup \{0\} \quad \text{where } \mathcal{N} \text{ is a set of natural numbers.} \qquad \square$$

Each local clock measures the length of the passage of time according to its own measurement rate. The rate is coarser than that of the passage of the physical time, and may drift. When reading time, the clock may have non-negligible reading errors. We introduce a mapping to specify the measurement precision of each a clock.[5]

[5]Note that the mapping is a little different from that of usual functions in mathematics.

Definition 4.3.2 Let $\mathcal{T}_i$ be the local time domain of i-th clock. We call Φ_i ($\Phi_i : \mathcal{T}_i \to 2^{\mathcal{R}^{+0}}$) *clock mapping* for i-th clock, where Φ_i is given as follows:

$$\Phi_i(t) \stackrel{\text{def}}{=} \{\ \phi(t) \mid \delta_1 t + \epsilon_1 \ \leq\ \phi(t)\ \leq\ \delta_2 t + \epsilon_2\ \} \quad \text{where} \quad \Phi(0) = \{0\}$$

where we assume $\delta_1, \delta_2, \epsilon_1, \epsilon_2$ are given constants ($\delta_1, \delta_2, \epsilon_1, \epsilon_2 \in \mathcal{R}^{+0}$, $\delta_1 \leq \delta_2$ and $\epsilon_1 \leq \epsilon_2$). We denote elements of Φ_i as $\phi, \phi', \ldots$ ($\phi : \mathcal{T}_i \to \mathcal{R}^{+0}$), where for all $\phi \in \Phi_i$ and $t, \dot{t} \in \mathcal{T}_i$, there is some mapping $\dot{\phi} \in \Phi_i$ such that $\phi(t + \dot{t}) = \phi(t) + \dot{\phi}(\dot{t})$. □

In the above definition, δ_1 and δ_2 correspond to the minimal and the maximal measurement rates of i-th clock respectively, and ϵ_1 and ϵ_2 correspond to the minimal and the maximal possible offsets, when reading of i-th clock respectively. $\Phi_i(t)$ represents all the possible values on real time domain measured by i-th clock at t local time units. We classify clock mappings according to the properties of clocks.

Definition 4.3.3 Let $\mathcal{T}$ be a local time domain. We give $\Phi : \mathcal{T} \to 2^{\mathcal{R}^{+0}}$,

(1) *Variable Clock*: Φ is called a *variable* clock, where Φ can be defined as follows:

$$\Phi(t) \stackrel{\text{def}}{=} \{\ \phi(t) \,|\, \delta_1 t \ \leq\ \phi(t)\ \leq\ \delta_2 t\ \}$$

where δ_1 and δ_2 are given constants in $\mathcal{R}^{+0}$.

(2) *Synchronized Clock*: Φ is called a *synchronized* clock, where Φ can be given as follows:

$$\Phi(t) \stackrel{\text{def}}{=} \{\ \phi(t) \,|\, \delta t + \epsilon_1\ \leq \phi(t)\ \leq \delta t + \epsilon_2\ \} \quad (t > 0)$$

where we assume $\Phi(0) \stackrel{\text{def}}{=} \{\ 0\ \}$ and δ, ϵ_1 and ϵ_2 are given constants in $\mathcal{R}^{+0}$.

(3) *Monotone Clock*: Φ is called a *monotone* clock if for any $t_1, t_2 \in \mathcal{T}$ such that $t_1 < t_2$: $\mathbf{max}\Phi(t_1) < \mathbf{min}\Phi(t_2)$. □

Note that the clock given in (2) corresponds a clock adjusted by synchronization mechanisms, for example see [40, 43, 45]. Therefore, the clock can satisfy the condition of synchronized clocks given in [43] when choosing $\epsilon_1 + \epsilon_2$ to be less than $\epsilon_{\mathbf{max}}$:

$$\forall i, j \in I,\ \forall t \in \mathcal{T} : \quad |\Phi_i(t) - \Phi_j(t)| \leq \epsilon_{\mathbf{max}}$$

where Φ_i and Φ_j are clock mappings for i-th and j-th clocks, and $\epsilon_{\mathbf{max}}$ means the maximal time difference between their reading times.

We here define the inverses of clock mappings. In the calculus, all time values measured by local clocks are assumed to be discrete. If an event is executed between two consecutive time instants, the event occurs at the earlier time instant. Also, if two or more events occur between two consecutive time instants, we say that the events occur at the same time.

Definition 4.3.4 Let $\Phi_i^{-1} : \mathcal{R}^{+0} \rightarrow 2^{\mathcal{T}_i}$ be the inverse function of clock function Φ_i given as follows:

$$\Phi_i^{-1}(T) \stackrel{\text{def}}{=} \bigcup_{\phi \in \Phi_i} \{\ t \in \mathcal{T}_i \mid \phi(t) \leq T < \phi(t+1)\ \}$$

where $T \in \mathcal{R}^{+0}$. □

We show an alternative notation of the inverse mapping below.[6]

Proposition 4.3.5 Let Φ_i: $\mathcal{T}_i \rightarrow 2^{\mathcal{R}^{+0}}$ be the clock mapping of i-th local clock and Ψ: $\mathcal{R}^{+0} \rightarrow 2^{\mathcal{T}_i}$. Then, $\Psi(T)$ is equivalent to $\Phi_i^{-1}(T)$.

$$\begin{aligned} \Psi(T) &\stackrel{\text{def}}{=} \{\ t \in \mathcal{T}_i \mid \left\lfloor \frac{T-\epsilon_2}{\delta_2} \right\rfloor \leq t \leq \left\lfloor \frac{T-\epsilon_1}{\delta_1} \right\rfloor\ \} \\ \textit{where}\quad \Phi_i(t) &\stackrel{\text{def}}{=} \{\, \phi(t) \mid \delta_1 t + \epsilon_1 \ \leq \phi(t) \ \leq \delta_2 t + \epsilon_2 \,\} \end{aligned}$$

where we assume that δ_1 and δ_2 are the minimal and the maximal possible measurement rates respectively, and ϵ_1 and ϵ_2 are the minimal and the maximal errors when reading of the clock, respectively. □

Example 4.3.6

(1) Suppose a synchronized clock whose measurement rate is 8 real time and its minimal and maximal possible reading errors are from 0 to 2. The clock mapping of the clock is denoted as follows:

$$\Phi_1(t) \stackrel{\text{def}}{=} \{\ \phi(t) \mid 8t + 0 \ \leq \ \phi(t) \ \leq \ 8t + 2\ \}$$

[6] $\lfloor x \rfloor = n$ where $n \leq x < n+1$ and n is an integer.

where we assume $\Phi_1(0) = \{0\}$. Its inverse mapping is given as follows:

$$\Phi_1^{-1}(T) \stackrel{\text{def}}{=} \{\ t \mid \ \lfloor\frac{T-2}{8}\rfloor \leq t \leq \lfloor\frac{T}{8}\rfloor\ \}$$

(2) Suppose a variable clock whose measurement rate varies from 3 to 5 real time is given as follows:

$$\Phi_2(t) \stackrel{\text{def}}{=} \{\ \phi(t) \mid 3t \leq \phi(t) \leq 5t\ \}$$

Its inverse mapping is given as follows:

$$\Phi_2^{-1}(T) \stackrel{\text{def}}{=} \{\ t \mid \ \lfloor\frac{T}{5}\rfloor \leq t \leq \lfloor\frac{T}{3}\rfloor\ \}$$

Syntax

To clarify our exposition, we divide the syntax of the calculus into two groups: *sequential expressions* for describing processes on a processor (or a node) with one local clock, and *interacting expressions* for describing interactions among distributed processes following different clocks.

Definition 4.3.7 The set $\mathcal{E}$ of local process expressions, ranged over by E, E_1, E_2, is defined by the following expressions:

$$E \ ::= \ \mathbf{0} \mid X \mid \alpha.E \mid E_1 + E_2 \mid \mathbf{rec}\, X : E \mid \langle t\rangle.E \mid [t].E$$

where t is an element of a local time domain. We assume that every process variable is guarded.[7]. Hereafter we shall often use the more readable notation $X \stackrel{\text{def}}{=} E$ instead of $\mathbf{rec}\, X : E$. □

Definition 4.3.8 The set $\mathcal{P}$ of interacting expressions, ranged over by P, P_1, P_2, is defined by the following expressions:

$$P \ ::= \ (E)[\Phi] \mid P_1 \,|\, P_2 \mid P[f] \mid P \setminus L$$

where Φ is a time translation mapping. We assume that $f \in Act \to Act$ where $f(\tau) = \tau$, and $L \subseteq \Lambda$. We will often abbreviate $(E)[\Phi]$ as $E[\Phi]$. □

[7] X is *guarded* in E if each occurrence of X is only within some subexpressions $\alpha.E'$ in E where α is not an empty element; c.f. *unguarded* expressions, e.g. $\mathbf{rec}\, X : X$ or $\mathbf{rec}\, X : X + S$.

Remarks Let us give some remarks on the syntax.

- $E[\Phi]$ means a process E executed by a processor (or a node) with local clock Φ. All the constructors except for $E[\Phi]$ coincide with those in *RtCCS*.
- By the definition of $\mathcal{P}$, the expressions applicable to $(-)[\Phi]$ are restricted to sequential expressions included in $\mathcal{E}$. Therefore, the calculus might not seem to be able to describe parallel processes following the same clock. However, such parallel processes can be reduced to an equivalent sequential expression in $\mathcal{E}$ by using the expansion rules developed in Corollary 1 and Proposition 10 of the author's paper [64]. Therefore, this limitation never creates any actual problem in describing distributed systems.

Semantics

The semantics of $RtCCS_L$ is defined through two steps: *clock translation rule* written as $E[\Phi]$ which translate corresponding expressions based on local time into ones on the real time domain. And then, the translated expressions can be interpreted as *RtCCS* expressions through *RtCCS*'s operational semantics.

Clock Translation

We first define the clock translation rule. The rule maps expressions based on a local time domain into ones based on the real time domain by translating all local time values in the expressions into corresponding values on the real time domain.

Definition 4.3.9 $(E)[\Phi]$, called *clock translation rule*, is recursively defined as follows:

$$
\begin{aligned}
E[\Phi] &\longrightarrow \bigoplus_{\phi\in\Phi} E[\Phi]_\phi && (1)\\
(\mathbf{0})[\Phi]_\phi &\longrightarrow \mathbf{0} && (2)\\
(X)[\Phi]_\phi &\longrightarrow X && (3)\\
(\lambda.E)[\Phi]_\phi &\longrightarrow \lambda.E[\Phi] && (4)\\
(\tau.E)[\Phi]_\phi &\longrightarrow \tau.E[\Phi]_\phi && (5)\\
(E_1 + E_2)[\Phi]_\phi &\longrightarrow E_1[\Phi]_\phi + E_2[\Phi]_\phi && (6)\\
(\mathbf{rec}\, X : E)[\Phi]_\phi &\longrightarrow \mathbf{rec}\, X : E[\Phi]_\phi && (7)\\
(\langle t\rangle.E)[\Phi]_\phi &\longrightarrow \langle\phi(t)\rangle.E[\Phi]_{\dot\phi} && (8)\\
([t].E)[\Phi]_\phi &\longrightarrow [\phi(t)].E[\Phi]_\phi && (9)
\end{aligned}
$$

where $\forall \dot{t}, \phi(t+\dot{t}) = \phi(t) + \dot{\phi}(\dot{t})$. We write $\tau.E_1 + \cdots + \tau.E_n$ as $\bigoplus_{i\in\{1,\ldots,n\}} E_i$. □

We briefly explain the intuitive meaning of some important rules in Definition 4.3.9.

- The first rule means that $E[\Phi]$ corresponds to process E which follows an arbitrary one of all the clocks whose possible precisions are specified as Φ.
- The fourth rule defines that every process rewinds its own clock only after performing any communication action. However, such a rewinded clock does not always reproduce the same measurement errors.
- The ninth rule translates the idling time of $\langle t\rangle.E$ on the local time domain into a corresponding value on the real time domain.
- The tenth rule translates the deadline time of $[t].E$ on the local time domain into a corresponding value on the real time coordinate.

Hereafter, we will often omit the $\longrightarrow$ translation if it is directly understood from the context.

Operational Semantics

The clock translation rules can eliminate all $(-)[\Phi]$ from $RtCCS_L$ expressions. The syntax and semantics of the translated expressions on the real time domain coincide with *RtCCS* expressions. All the translated expressions can be interpreted as *RtCCS* expressions.

Definition 4.3.10 *RtCCS*$_L$ including no $(E)[\Phi]$ is a labeled transition system defined through the transition relation rules given in Definition 2.3.6.

Note that the translated expressions can enjoy all the proof techniques presented in Chapter 2.

4.4 Bisimulations with Time Uncertainties

In Chapter 2 we studied some time-sensitive equivalence relations. They can equate two processes when both their behavioral properties and their temporal properties are *completely* matched with each other. However, these relations may often be too strict in the verification of distributed processes with temporal uncertainties. It is practical that two processes following inaccurate clocks can be considered to be equivalent, only if their behaviors are completely matched and differences in their timings are within a given bound. We develop such an equivalence by extending the notion of bisimulation [58, 51] and study its theoretical properties.

Definition 4.4.1 A binary relation $\mathcal{R}_\Phi$ is a Φ-*clock bisimulation* $(\Phi : \mathcal{T}_\ell \to 2^{\mathcal{R}^{+0}})$ if $(P_1, P_2) \in \mathcal{R}_\Phi$ implies, for all $\alpha \in Act$,

(i) $\forall m_1, \forall P_1'$: $P_1 \stackrel{\langle m_1 \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} P_1'$ *then*
$\exists m_2, \exists P_2'$: $P_2 \stackrel{\langle m_2 \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} P_2'$ *and* $\Phi^{-1}(m_1) \cap \Phi^{-1}(m_2) \neq \emptyset$ *and* $(P_1', P_2') \in \mathcal{R}_\Phi$.

(ii) $\forall m_2, \forall P_2'$: $P_2 \stackrel{\langle m_2 \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} P_2'$ *then*
$\exists m_1, \exists P_1'$: $P_1 \stackrel{\langle m_1 \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} P_1'$ *and* $\Phi^{-1}(m_1) \cap \Phi^{-1}(m_2) \neq \emptyset$ *and* $(P_1', P_2') \in \mathcal{R}_\Phi$.

where $m_1, m_2 \in \mathcal{R}^{+0}$. We say P_1 and P_2 are Φ-clock bisimilar, written as $P_1 \approx_\Phi P_2$, if there exists a Φ-clock bisimulation, $\mathcal{R}_\Phi$ such that $(P_1, P_2) \in \mathcal{R}_\Phi$. □

We explain the key idea of $\approx_\Phi$. We introduce a conceptual experimenter following a non-perfect clock, written Φ, which may be coarse and may drift within a certain bound. The experimenter cannot find any temporal difference smaller than the measurement sensitivity of the clock. Therefore, if the experimenter cannot distinguish between two processes, P_1 and P_2, in their observational behaviors and their temporal differences are within the measurement sensitivity of the clock, $P_1 \approx_\Phi P_2$.

From Definition 4.4.1 we can directly prove the following properties.

Proposition 4.4.2 For all $P, P_1, P_2, P_3 \in \mathcal{P}$ we have the following properties:

(1) $P \approx_\Phi P$

(2) $P_1 \approx_\Phi P_2$, *then* $P_2 \approx_\Phi P_1$

(3) $\approx_\Phi$ is the largest Φ-clock bisimulation. □

Proof. Immediate from Definition 4.4.1. □

Proposition 4.4.3 Let $P_1, P_2, Q \in \mathcal{P}$: $P_1 \approx_\Phi P_2$ then

(1) $P_1 \setminus L \approx_\Phi P_2 \setminus L$

(2) $P_1[f] \approx_\Phi P_2[f]$

(3) $P_1|Q \approx_\Phi P_2|Q$

where we assume that $Q \in \mathcal{P}$ contains no delay operator nor time restriction operator. □

Proof. Analogous to Proposition 2.4.7. □

From $P_1 \approx_\Phi P_2$, we do not have $\langle t\rangle.P_1 \approx_\Phi \langle t\rangle.P_2$ nor $[t].P_1 \approx_\Phi [t].P_2$. We show a counterexample: $\langle 2\rangle.a.\mathbf{0} \approx_{\Phi'} \langle 4\rangle.a.\mathbf{0}$ where $\Phi'(t) = \{6t\}$, but $\langle 3\rangle.\langle 2\rangle.a.\mathbf{0} \not\approx_{\Phi'} \langle 3\rangle.\langle 4\rangle.a.\mathbf{0}$, and $[3].\langle 2\rangle.a.\mathbf{0} \not\approx_{\Phi'} [3].\langle 4\rangle.a.\mathbf{0}$.

We show a relationship between this bisimilarity and the observation bisimilarity studied in Chapter 2.

Proposition 4.4.4 For all $P_1, P_2 \in \mathcal{P}$ we have the following property:

$$P_1 \sim_{\mathcal{T}} P_2 \quad \textit{then} \quad P_1 \approx_\Phi P_2$$ □

Proof. Because of $\Phi^{-1}(m) \cap \Phi^{-1}(m) \neq \emptyset$. □

There is another relation between $\sim_{\mathcal{T}}$ and $\approx_\Phi$. A few preliminary lemmas are needed before showing the result.

Lemma 4.4.5 $\forall \phi \in \Phi, \forall m$: $E[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'[\Phi]$ *then* $E \stackrel{\langle \phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$

Proof. We proceed by transition induction on the depth of the inference of $E \stackrel{\langle \phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$. Consider the cases for E.

Case 1 $E \equiv \mathbf{0}$, obvious by definition.

Case 2 $E \equiv X$, a process variable. By a shorter inference for X from $X[\Phi]_\phi \longrightarrow\!\!\!\!\!\rightharpoonup X$.

Case 3 $E \equiv \alpha.\dot{E}$, Then we have $(\alpha.\dot{E})[\Phi]_\phi -\!\!\!\!\!\rightharpoonup \alpha.\dot{E}[\Phi] \stackrel{\langle\dot{m}\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}[\Phi]$. The result follows by choosing E' to be $\dot{E}$.

Case 4 $E \equiv E_1 + E_2$, Then either $E_1 \stackrel{\langle\phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$ or $E_2 \stackrel{\langle\phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$ by a shorter inference. We can apply the lemma in either case.

Case 5 $E \equiv \mathbf{rec}\, X : \dot{E}$, Then we have $\dot{E} \stackrel{\langle\phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}'$. By a shorter inference for $\dot{E}$, we will get the result.

Case 6 $E \equiv \langle d\rangle.\dot{E}$, By induction, $\dot{\phi} \in \Phi$ such that $\phi(d+\dot{\phi}^{-1}(\dot{m})) = \phi(d)+\dot{\phi}(\dot{\phi}^{-1}(\dot{m}))$, we have $(\dot{E})[\Phi]_{\dot{\phi}} \stackrel{\langle\dot{m}\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} (\dot{E}')[\Phi]$ then $\dot{E} \stackrel{\langle\dot{\phi}^{-1}(\dot{m})\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$. Since $(\langle d\rangle.\dot{E})[\Phi]_\phi -\!\!\!\!\!\rightharpoonup \langle\phi(d)\rangle.(\dot{E})[\Phi]_{\dot{\phi}}$ and $(\dot{E})[\Phi]_{\dot{\phi}} \stackrel{\langle\phi(d)\rangle}{\Longrightarrow} \stackrel{\langle\dot{m}\rangle}{\Longrightarrow} (\dot{E}')[\Phi]_{\dot{\phi}}$, the result follows easily by choosing m to be $\phi(d) + \dot{m}$ and E' to be $\dot{E}'$.

Case 7 $E \equiv \lfloor d\rfloor.\dot{E}$, by inductive hypothesis of E, $(\dot{E})[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} (\dot{E}')[\Phi]_\phi$ then $\dot{E} \stackrel{\langle\phi^{-1}(m)\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}'$. Since $m \leq \phi(d)$, we have $\phi^{-1}(m) \leq d$. By choosing $\dot{E}'$ to be E', we get the result. □

Lemma 4.4.6 *$\forall k$: $E \stackrel{\langle k\rangle}{\Longrightarrow} \stackrel{\lambda}{\Longrightarrow} E'$ then*
$\forall \phi \in \Phi$, $\forall m$: $k = \phi^{-1}(m)$, $E[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} \stackrel{\lambda}{\Longrightarrow} E'[\Phi]$ □

Proof. We proceed by transition induction on the depth of the inference of $E[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'$ $[\Phi]$. Consider the cases for E.

Case 1 $E \equiv \mathbf{0}$, obvious from definition.

Case 2 $E \equiv X$, a process variable. By a shorter inference for X from $X[\Phi]_\phi \longrightarrow\!\!\!\!\!\rightharpoonup X$.

Case 3 $E \equiv \alpha.\dot{E}$, Then we have $(\alpha.\dot{E})[\Phi]_\phi -\!\!\!\!\!\rightharpoonup \alpha.\dot{E}[\Phi]$. By induction, the result easily follows by choosing E' to be $\dot{E}$.

Case 4 $E \equiv E_1 + E_2$, Then, we have $(E_1 + E_2)[\Phi]_\phi \longrightarrow\!\!\!\!\!\rightharpoonup E_1[\Phi]_\phi + E_2[\Phi]_\phi$ and then either $E_1[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'[\Phi]$ or $E_2[\Phi]_\phi \stackrel{\langle m\rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E'[\Phi]$. We can apply the lemma in either case.

Case 5 $E \equiv \mathbf{rec}\, X : \dot{E}$, Then we have $\mathbf{rec}\, X : \dot{E}[\Phi] \longrightarrow\!\!\!\!\!\rightharpoonup \mathbf{rec}\, X : \dot{E}[\Phi]$. By induction we can apply the lemma to $\mathbf{rec}\, X : \dot{E}[\Phi]$ and we get the result.

Case 6 $E \equiv \langle d \rangle.\dot{E}$. Then, by induction, we have for any $\dot{\phi} \in \Phi$ such that for any $\dot{k}$, $\phi(d + \dot{k}) = \phi(d) + \dot{\phi}(\dot{k})$ For any $\dot{m}$, $\dot{E}[\Phi]_{\dot{\phi}} \stackrel{\langle \dot{m} \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}'[\Phi]$ then $\dot{k} = \dot{\phi}^{-1}(\dot{m})$ and $\dot{E} \stackrel{\langle \dot{k} \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}'$. Since $(\langle d \rangle.\dot{E})[\Phi]_{\phi} \longrightarrow \langle \phi(d) \rangle.\dot{E}[\Phi]_{\dot{\phi}} \stackrel{\langle \phi(d) \rangle}{\Longrightarrow} \dot{E}[\Phi]_{\dot{\phi}}$ and $\phi^{-1}(\phi(d) + \dot{m}) = d + \dot{\phi}^{-1}(\dot{m})$. The result follows easily by choosing $\dot{E}'$ to be E' and m to be $\phi(d) + \dot{m}$.

Case 7 $E \equiv [d].\dot{E}$. Then, we have $([d].\dot{E})[\Phi]_{\phi} \longrightarrow [\phi(d)].\dot{E}[\Phi]_{\phi}$. By inductive hypothesis of E, $\dot{E} \stackrel{\langle \dot{k} \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}$ then $k = \phi^{-1}(m)$ and $\dot{E}[\Phi]_{\phi} \stackrel{\langle m \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} \dot{E}[\Phi]$. By choosing $\dot{E}'$ to be E', the result follows. □

Lemma 4.4.7 $\forall k\colon E \stackrel{\langle k \rangle}{\Longrightarrow} \stackrel{\tau}{\Longrightarrow} E'$ *then* $\forall \phi \in \Phi,\ E[\Phi]_{\phi} \stackrel{\langle \phi(k) \rangle}{\Longrightarrow} \stackrel{\tau}{\Longrightarrow} E'[\Phi]$. □

Proof. Analogous to 4.4.6. □

We now present that there is a correlation between clock translation $E[\Phi]$ and Φ-clock bisimilarity.

Theorem 4.4.8 Let Φ be a monotone clock. For any $E_1, E_2 \in \mathcal{E}$,

$$E_1 \ \sim_{\mathcal{T}} \ E_2 \qquad \textit{iff} \qquad E_1[\Phi] \ \approx_{\Phi} \ E_2[\Phi]$$ □

Proof. ($\Longrightarrow$) By proving that $\mathcal{R}_{\Phi}$ is a Φ-clock bisimulation, where $\mathcal{R}_{\Phi} \stackrel{\text{def}}{=} \{\, (E_1[\Phi], E_2[\Phi]) \mid E_1 \sim E_2 \,\}$ Let $\forall m_1\colon E_1[\Phi] \stackrel{\langle m_1 \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E_1'[\Phi]$. When $\alpha = \tau$, it is easy. Therefore, we will prove the case of $\alpha \neq \tau$. Then, from Lemma 4.4.5, $\forall \phi_1 \in \Phi$, $\forall k\colon k = \phi_1^{-1}(m_1)$ with $E_1 \stackrel{\langle k \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E_1'$. Since $E_1 \sim E_2$, we have $E_2 \stackrel{\langle k \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} E_2'$ with $E_1' \sim E_2'$. From Lemma 4.4.6, we have $\exists m_2\colon k = \Phi^{-1}(m_2)$ and $E_2[\Phi]_{\phi} \stackrel{\langle m_2 \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} E_2'[\Phi]$. Since Φ is a monotone clock, we have $\Phi^{-1}(m_1) \cap \Phi^{-1}(m_2) \neq \emptyset$.

($\Longleftarrow$) By proving that $\mathcal{R}$ is a Φ-clock bisimulation, where $\mathcal{R} \stackrel{\text{def}}{=} \{\, (E_1, E_2) \mid E_1[\Phi] \sim_{\Phi} E_2[\Phi] \,\}$ Let $\forall k\colon E_1 \stackrel{\langle k \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E_1'$. When $\alpha = \tau$, it is easy. Therefore, we will prove the case of $\alpha \neq \tau$. Then, from Lemma 4.4.6, $\forall \phi_1 \in \Phi, \forall m_1 :\ k = \phi_1^{-1}(m_1) \wedge E_1[\Phi]_{\phi} \stackrel{\langle m_1 \rangle}{\Longrightarrow} \stackrel{\alpha}{\Longrightarrow} E_1'[\Phi]$. From $E_1[\Phi] \sim_{\Phi} E_2[\Phi]$, we have $\forall \phi_2 \in \Phi, \exists m_2 :\ E_2[\Phi]_{\phi} \stackrel{\langle m_2 \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} E_2'[\Phi]_{\phi}$ and $\Phi^{-1}(m_1) \cap \Phi^{-1}(m_2) \neq \emptyset$ and $E_1'[\Phi] \sim_{\Phi} E_2'[\Phi]$. Hence, since Φ is a monotone clock, $k = \Phi^{-1}(m_2)$ is required. From Lemma 4.4.5, we have $\exists \phi_2' \in \Phi \wedge k = \phi'^{-1}_2(m_2) :\ E_2 \stackrel{\langle k \rangle}{\Longrightarrow} \stackrel{\widehat{\alpha}}{\Longrightarrow} E_2'$ with $E_1'[\Phi] \sim_{\Phi} E_2'[\Phi]$.

By a symmetric argument, we complete the proof that $\mathcal{R}_{\Phi}$ and $\mathcal{R}$ are the bisimulation. □

The above result tells that $\approx_\Phi$ is a complement of $E[\Phi]$ and vice versa. Therefore, in equating two distributed processes following inaccurate clocks, it allows us to abstract away the influences of inaccuracies of clocks from differences between their temporal properties of them exactly. We can predicate temporal differences among two time-dependent programs, even when they executed with processors following inaccurate clocks.

Timed Bisimulations according to Clock Precisions

In the rest of this section we introduce two orders over clocks with respect to clock's sensitivity. First we define an order based on the granularity of clocks.

Definition 4.4.9 Let $\Phi_1 : \mathcal{T}_1 \to 2^{\mathcal{R}^{+0}}$ and $\Phi_2 : \mathcal{T}_2 \to 2^{\mathcal{R}^{+0}}$ be clock mappings where $\mathcal{T}_1$ are $\mathcal{T}_2$ are local time domains, $\forall t_2 \in \mathcal{T}_2, \exists t_1 \in \mathcal{T}_1 : \Phi_1(t_1) = \Phi_2(t_2)$ then, we say that Φ_1 is finer than Φ_2, written as $\Phi_1 \trianglelefteq \Phi_2$. □

This order relation relates two clocks according to their measurement granularities. If the granularity of one of them is less than that of the another, the former is finer than the latter.

Definition 4.4.10 Let $\Phi_1 : \mathcal{T} \to 2^{\mathcal{R}^{+0}}$ and $\Phi_2 : \mathcal{T} \to 2^{\mathcal{R}^{+0}}$ be clock mappings where $\mathcal{T}$ is a local time domains, $\forall t \in \mathcal{T} : \Phi_1(t) \subseteq \Phi_2(t)$ then, we say that Φ_1 is more precise than Φ_2, written as $\Phi_1 \preceq \Phi_2$. □

This order relates two clocks according to the possible measurement errors. If the possible errors of a clock is smaller than that of the another, the former is more precise than the latter.

From Definition 4.4.9 and 4.4.10, we can easily confirm that $\trianglelefteq$ and $\preceq$ are preorder relations. We show two interesting facts between $\approx_\Phi$ and these orders.

Proposition 4.4.11 $\forall P_1, P_2 \in \mathcal{P}$, $\Phi_1 \trianglelefteq \Phi_2$, $P_1 \approx_{\Phi_1} P_2$ *then* $P_1 \approx_{\Phi_2} P_2$ □

Proof. $\{\ n_1 \mid \Phi_1^{-1}(m) \cap \Phi_1^{-1}(n_1) \neq \emptyset\ \} \subseteq \{\ n_2 \mid \Phi_2^{-1}(m) \cap \Phi_2^{-1}(n_2) \neq \emptyset\ \}$. □

This proposition tells that if two processes cannot be distinguished by an observer following a clock whose time unit is one second, then they cannot be distinguished by another observer having a clock with the unit of one minute.

Proposition 4.4.12 $\forall P_1, P_2 \in \mathcal{P},\ \Phi_1 \preceq \Phi_2\ P_1 \approx_{\Phi_1} P_2 \quad then \quad P_1 \approx_{\Phi_2} P_2$ □

Proof. From for all $m, n : (\Phi_1^{-1}(m) \cap \Phi_1^{-1}(n)) \subseteq (\Phi_2^{-1}(m) \cap \Phi_2^{-1}(n))$. □

Let the measurement error of a clock (e.g. ± 1 seconds) be less than that of another clock (e.g. ± 5 seconds). If two processes cannot be distinguished from each other by an observer with the former clock, they cannot do by one with the latter clock.

4.5 Examples

In order to illustrate how to describe and verify distributed processes in $RtCCS_L$, we present some simple examples.

Example 4.5.1 Suppose interactions between a client process and a sever process executed with different processors.

- The client process (*Client*) sends a request message (action $\overline{req}$) and then waits for a return message (action ret). If the return message is not received within 6 units of local time, then it sends the request message again.
- Upon reception of a request message (action req), the server process (*Server*) sends a return message (action $\overline{ret}$) after an internal execution of 5 units of local time.

The client and server programs are denoted as follows:

$$\begin{aligned} Client &\stackrel{\text{def}}{=} \overline{req}.([6].ret.\mathbf{0} + \langle 6\rangle.Client) \\ Server &\stackrel{\text{def}}{=} req.\langle 5\rangle.\overline{ret}.Server \end{aligned}$$

We assume that the client process and the server one are allocated on different processors having different clocks, written by Φ_c and Φ_s. The measurement rate of the client's clock varies from 4 to 6 on $\mathcal{R}^{+0}$. The measurement rate of the server's clock varies from 3 to 5 on $\mathcal{R}^{+0}$. For simplify, we assume that these clocks do not have any reading errors. Φ_c and Φ_s are defined as follows:

$$\begin{aligned} \Phi_c(t) &\stackrel{\text{def}}{=} \{\ \phi_c(t) \mid 4t \le \phi_c(t) \le 6t\ \} \\ \Phi_s(t) &\stackrel{\text{def}}{=} \{\ \phi_s(t) \mid 3t \le \phi_s(t) \le 5t\ \} \end{aligned}$$

where for any $\phi \in \Phi$, $\forall t, \dot{t} \in \mathcal{T}$ there is some function $\dot{\phi} \in \Phi_i$ in $\phi(t+\dot{t}) = \phi(t) + \dot{\phi}(\dot{t})$.

By using $E[\Phi]$ mapping rule, the client and the server are mapped on the real time domain as shown below.

$$\begin{array}{lll} \mathit{Client}[\Phi_c] & \longrightarrow \cdots \longrightarrow & \overline{req}.([\phi_c(6)].\mathit{ret}.\mathbf{0} + \langle\phi_c(6)\rangle.\mathit{Client}[\Phi_c]_{\dot{\phi_c}}) \\ \mathit{Server}[\Phi_s] & \longrightarrow \cdots \longrightarrow & \mathit{req}.\langle\phi_s(5)\rangle.\overline{\mathit{ret}}.\mathit{Server}[\Phi_s] \end{array}$$

where ϕ_c and $\dot{\phi_c}$ are elements of Φ_c such that $\forall t : \phi_c(6) + \dot{\phi_c}(t) = \phi_c(6+t)$. ϕ_s is an elements of Φ_s. From the definition of Φ_c and Φ_s, there are multiple results for $\phi_c(6) \in \{24, 25, ..36\}$ and $\phi_s(5) \in \{15, 16, ..25\}$. The interactions between these processes is described as follow:

$$(\mathit{Client}[\Phi_c] \mid \mathit{Server}[\Phi_s]) \setminus \{req, ret\}$$

The result of the interactions is dependent on the evaluated values of $\phi_c(6)$ and $\phi_s(5)$. Here we show some of the possible results:

(1) In the case of $\phi_c(6) > \phi_s(5)$,

$$\begin{array}{ll} & (\mathit{Client}[\Phi_c] \mid \mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\tau} & ([\phi_c(6)].\mathit{ret}.\mathbf{0} + \langle\phi_c(6)\rangle.\mathit{Client}[\Phi_c]_{\dot{\phi_c}}) \mid \\ & \qquad \langle\phi_s(5)\rangle.\overline{\mathit{ret}}.\mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\langle\phi_s(5)\rangle} & ([\phi_c(6) \dot{-} \phi_s(5)].\mathit{ret}.\mathbf{0} + \langle\phi_c(6) \dot{-} \phi_s(5)\rangle.\mathit{Client}[\Phi_c]_{\dot{\phi_c}}) \mid \\ & \qquad \overline{\mathit{ret}}.\mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\tau} & \mathbf{0} \mid \mathit{Server}[\Phi_s]_{\phi_s}) \setminus \{req, ret\} \\ & (\mathit{success}) \end{array}$$

In the above case, the client can receive the return message before it goes into timeout.

(2) In the case of $\phi_c(6) \leq \phi_s(5)$,

$$\begin{array}{ll} & (\mathit{Client}[\Phi_c] \mid \mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\tau} & ([\phi_c(6)].\mathit{ret}.\mathbf{0} + \langle\phi_c(6)\rangle.\mathit{Client}[\Phi_c]_{\dot{\phi_c}}) \mid \\ & \qquad \langle\phi_s(5)\rangle.\overline{\mathit{ret}}.\mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\langle\phi_c(6)\rangle} & \mathit{Client}[\Phi_c]_{\dot{\phi_c}} \mid \langle\phi_s(5) \dot{-} \phi_c(6)\rangle.\overline{\mathit{ret}}.\mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ \xrightarrow{\langle\phi_s(5) - \phi_c(6)\rangle} & \mathit{Client}[\Phi_c]_{\dot{\phi_c}} \mid \overline{\mathit{ret}}.\mathit{Server}[\Phi_s]) \setminus \{req, ret\} \\ & (\mathit{failure}) \end{array}$$

In the above case, the client goes into timeout before receiving any return message ret. Thus, the processes go into a timing failure.

$RtCCS_L$ allows us to analyze how the differences among local clocks affect the result of interactions in distributed computing. □

Note that the idling time of the delay operator in *Server* means execution steps for handling the request in the server. Φ_s in $Server[\Phi_s]$ represents an index on the performance of a processor executing the server. In this contexts, when a processor's clock has a smaller measurement rate, the processor is faster. This result shows that the time translation rules $(-)[\Phi]$ allows us to represent differences among processors' performances as well as differences among processors' clocks.

Example 4.5.2 We demonstrate the utility of Φ-clock bisimulations. Suppose two client processes, written $Client_A$ and $Client_B$. $Client_A$ sends a request message (action $\overline{req}$) and then waits for a return message (action ret). It sends the request message again, if the return message is not received within 8 units of its local time. On the other hand, $Client_B$ is the same of $Client_A$ except that its deadline time is 9 units of its local time. These processes are denoted as follows:

$$\begin{aligned} Client_A &\stackrel{\text{def}}{=} \overline{req}.[8].ret.\mathbf{0} + \langle 8\rangle.Client \\ Client_B &\stackrel{\text{def}}{=} \overline{req}.[9].ret.\mathbf{0} + \langle 9\rangle.Client \end{aligned}$$

We consider a clock defined as: $\Phi_1(t) \stackrel{\text{def}}{=} \{\, \phi_1(t) \,|\, \phi_1(t) \stackrel{\text{def}}{=} 6t \,\})$. An observer following Φ_1 cannot distinguish between the two processes in their temporal properties because $1 \in \Phi_1^{-1}(8)$ and $1 \in \Phi_1^{-1}(9)$.

$$Client_A \approx_{\Phi_1} Client_B \qquad c.f. \qquad Client_A \not\approx Client_B$$

We here suppose a clock Φ_2 such that $\Phi_1 \trianglelefteq \Phi_2$, where Φ_2 is defined as $\Phi_2 \stackrel{\text{def}}{=} \{\, \phi_2(t) \,|\, \phi_2(t) \stackrel{\text{def}}{=} 12t \,\})$. By Proposition 4.4.11 we have $Clinet_A \approx_{\Phi_2} Clinet_B$.

This example shows that our bisimilarity provides a practical method to verify real-time processes with non-strict time constraints. For example, let $Client_A$ be a specification of the client process and $Client_B$ be an implementation of the process.

$Client_A \approx_{\Phi_1} Client_B$ shows that the implementation completely satisfies the behavioral requirements in the specification and that temporal differences between both of them is within a permissible bound specified in terms of Φ_1.

Example 4.5.3 Suppose a client process ($Client$) whose timeout time is 6 units of local time as follows:

$$Client \stackrel{\text{def}}{=} \overline{req}.\langle 6\rangle.ret.\mathbf{0}$$

We assume that the program is allocated on two processors with different local clocks, denoted as Φ_A and Φ_B. The measurement rates of Φ_A and Φ_B may vary from 99 to 101 and from 100 to 102 respectively. These clocks are described as follow:

$$\Phi_A(t) \stackrel{\text{def}}{=} \{\ \phi_A(t) \mid 99t \le \phi_A(t) \le 101t\ \}$$
$$\Phi_B(t) \stackrel{\text{def}}{=} \{\ \phi_B(t) \mid 100t \le \phi_B(t) \le 102t\ \}$$

By $(\cdot)[\Phi]$ mapping rules, $Client$ following clock Φ_A is mapped on the standard time domain as shown below:

$$Client[\Phi_A] \longrightarrow \cdots \longrightarrow \overline{req}.([6].ret.\mathbf{0} + \langle 6\rangle.Client[\Phi_A]_{\dot{\phi}_A})$$

An observer with the following variable clock $\Phi'(t)$ equates these processes.

$$\Phi'(t) \stackrel{\text{def}}{=} \{\ \phi'(t) \mid 69t \le \phi'(t) \le 71t\ \}$$

$$Client[\Phi_A] \approx_{\Phi'} Client[\Phi_B] \qquad c.f. \qquad Client[\Phi_A] \not\approx Client[\Phi_B]$$

This result proves that Φ'-clock bisimilarity can equate two distributed processes even if their clocks are different and are not constant.

4.6 Concluding Remarks

This chapter proposed a new calculus which is an extension of the calculus developed in Chapter 2 with the ability to express processes following multiple inaccurate clocks. The calculus allows us to describe the properties of physical clocks quantitatively

and analyze the influences of their inaccuracies upon the behavioral and temporal properties of synchronous interactions among distributed processes. We also defined a time-sensitive bisimulation and study its theoretical properties. The bisimulation equates processes with temporal uncertainties, when their behaviors are completely matched and their timings are different within a given bound. It provides a suitable method to verify distributed processes following inaccurate clocks and and real-time processes with non-strict time constraints.

The approach studied in this chapter to deal with multiple inaccurate clocks is essentially independent of the calculus. It is general to be applied this kind of real-time formalisms, such as other timed process calculi, timed automata, real-time temporal logic, timed petri net, and so on.

Chapter 5

Related Work

In recent years, a wide variety formal models have been explored for the specification and verification of distributed computing systems. Some of them have laid a foundation for our work. In this chapter we compare with other existing process calculi which have the notion of quantitative time. Next we survey existing works which intend to model some peculiar features of distributed systems.

5.1 Timed Extended Process Calculi

Process calculi are structured description languages for non-deterministic and concurrent systems. A number of process calculi has been proposed in the literature. Examples are CSP [35], SCCS [50], CCS [51], π-calculus [52], ACP [4], LOTOS [10], and so on. These calculi lack the notion of time and thus cannot cope with descriptions of time-dependent systems. Several researchers have extended these process calculi with quantitatively expressive capabilities of temporal properties of systems. We summarize these time extended process calculi below.

CCS and it Extensions

During five years, several researchers have studied timed extensions of CCS. The extensions are various but have some common features. They assume that all actions take zero duration, and introduce a kind of synchronous broadcast communication actions among all processes to represent the passage of time. They also have special

operators whose contents are dependent on the passage of time. These operators can be classified into two groups according to their expressiveness.

Timed Extension: Delayed Processing. TPA developed by Hennessy and Regan in [31] is an extension of CCS. In addition to all the standard CCS operators, the calculus has a delay operator written as $\sigma.P$ to represent the suspension of execution for a unit of time. Timed CCS developed by Wang in [80, 79] has a special delay operator, written as $\varepsilon(d).\alpha@t.P$ which represents the suspension of execution for d real time. In $\varepsilon(d).\alpha@t.P$, variable t records the time at which α is executed. TPA and Timed CCS assume that time advances only when communications are not possible, like ours. However, there is a negative side in TPA and Timed CCS. Once an action is enabled, it is not disabled directly. Therefore, these calculi are difficult to express some indispensable temporal facilities found in distributed systems such as timeout handling.

Timed Extension: Timed Restriction. Moller and Tofts in [53, 54, 75] develop a time extension of CCS. It is characterized by the following primitives: $\mathbf{0}$, $\delta.P$, $(t).P$, and $P \oplus P$. $\mathbf{0}$ corresponds to a terminate process but does not allow time to pass; $\delta.P$ corresponds to the derived arbitrary delay operator of in SCCS [50] and is willing to wait any amount of time; $(t).P$ represents an idling for t time units; and $P \oplus Q$ is a time-sensitive choice operator. The choice is made not only by an action but also by idling when either P or Q. By combining $(t).P$ and $\oplus$, the calculus can restrict actions which are already executable.

Further, Timed CCS by Chen in [14] introduces a special action prefix, written as $\alpha_{t_1}^{t_2}.P$. CCSiT (CCS with interval Time) by Daniels in [17] have a action prefix, written as $\alpha@[t_1, t_2).P$. Both these operators represent a process which can perform action α from t_1 to t_2 units of time. The both examples correspond to $\langle t_1 \rangle.[t_2].P$ in ours. TPCCS by Hannson in [28, 27] is an extension of CCS with a timeout operator and the notion of probability. The timeout operator is described as a binary operator, written $\triangleright$. For example, $P \triangleright Q$ behaves like P if P performs an action for one time unit, otherwise like Q. The timeout operator stops the passage of time at its pre-timeout process. Therefore, the timeout operator cannot apply to any time-dependent processes. Recently, Hennessy and Regan in [30] extend their TPA with a timeout

operator, written as $\lfloor P \rfloor^t(Q)$. Their operator is similar to ATP's timeout operator developed in [57] and does not stop the passage of time at its pre-timeout process It corresponds to $\lceil t \rceil.P + \langle t \rangle.Q$ in ours.

CSP and its Extensions

For about ten years, a variety of time extensions of the CSP language by C. A. R.Hoare in [35] has been studied. By incorporating enrich operators of CSP, time extended CSP provides a powerful method to describe various time-dependent systems. It first appeared in [61] and has undergone a series of enhancements by Oxford University Timed CSP Group in [19, 18] and other researchers. Several denotational semantic models have been provided for the language in [25, 38, 62]. These semantics models define the meaning of time extended CSP programs to be a set of possible behaviors and timings at which the behaviors are performed and refused. In [20, 37] time extended CSP with asynchronous communication mechanisms is formulated based on a metric temporal logic.

Synchronous CCS and its Extensions

SCCS by Milner [50] is a process calculus for synchronously computing processes based on the idea that all parallel processes proceed in lock-step. There are some process calculi derived from SCCS, such as Meije by Simone [73] and CIRCAL by Milne [49]. These calculi assume the execution time of an action to be one unit of time and thus express quantities in time as the number of sequential actions. CCSR by Gerber and Lee [23, 24] introduces into SCCS with the concepts of quantitative time, priority, and computational resource. Processes assigned to the same resource are interleaved according to their priorities. However, in these SCCS-base approaches computation among parallel processes is essentially synchronous. Therefore, they do not fit to model distributed systems, where processes proceed at indeterminate relative speeds.

Other Calculi

ATP by Sifakis and Nicollin [57] has a timeout operator, written as $\lfloor P \rfloor^t(Q)$. Intuitively, it behaves as Q after t time units unless P performs some action within t

time units. The operator is encoded as $[t].P + \langle t \rangle.Q$ in our calculi. In ATP $\alpha.P$ can only offer action α only within one time unit, whereas ours can perform action α at any time whenever $\alpha.P$ are executable. Baeten and Bergstra in [6, 5] extend ACP [4] with a special operator corresponding to our delay operator. There are several timed enhancements of LOTOS [10]. Leduce [44] extends LOTOS with a timed operator which is similar to ATP's timeout operator. Bryans, Davis, and Schneider in [12] extend LOTOS with two timed operators: $\Delta^d P$ and $a\{d\}; P$, corresponding to our delay operator and time restriction operator respectively.

Remarks

We summarize comparisons between our theories and other time-extended calculi. Among our theories, *RtCCS* is unique among other existing process calculi in the following points; the first is that it has two temporal operators: delay operator and time restriction operator, in addition to the expressiveness of CCS; the second is that its temporal semantics does not interfere with the operational semantics of CCS. Therefore, it can encompass CCS and exactly inherit all the pleasant properties of CCS, including many proof techniques studied in CCS without changing them.

5.2 Formalisms for Locality in Communication

This section compare with existing formalisms which address locality in communication.

Process Calculi with Spatial Locality

Distributed systems often consist of processors allocated remotely. The correctness of distributed computation is not independent of the locations at which processors are allocated. Several researchers propose process calculi which have the notion of spatial locality, for example see [9, 13, 39, 56]. The basic idea of these calculi is to assign localities to processes and actions. They distinguish between actions which occur at different locations, even when the actions have the same name. However, none of the previous attempts to study a relationship between spatial locality and communication delay, although spatial locality is manifested by a function of communication delay.

Process Calculi with Asynchronous Communication

Ordinary process calculi are based on synchronous communication. However, for efficiency reasons, in distributed systems communication among remote processors is often asynchronous. Recently, a few process calculi with the ability to express asynchronous communication have been explored.

Most of the calculi introduce auxiliary mechanisms: message buffering, in order to capture non-blocking message sending. For example, Begstra and Klop in [8] extend ACP with arrival ordering queue and non ordering queue for messages. De Boer, Klop, and Paramidessi in [20] have studied further refinements of the calculus developed in [8] by formulating trace and failure equivalences for asynchronous communication. However, such buffering mechanisms are not always suitable to purely computational aspects of process calculi.

On the other hand, there have been several process calculi which model asynchronous messages as newly created output processes in [6, 36, 42], like ours. On the basis of π-calculus, Honda and Tokoro develop a calculus to construct a purely theoretical foundation for asynchronous communication with port passing mechanism. Their calculus lacks any alternative choice operator and thus is difficult to express systems depending on limited computational resources.

Among them, the calculus proposed by Baeten and Bergstra in [6] is notable. The calculus is a real time extension of ACP and can express asynchronous communication channel with latency. The calculus represents an asynchronous message as a newly created process corresponding to the message, and models communication delay as a suspension of the created process for the length of the latency. It is very similar to the calculus developed in Chapter 3. However, their calculus is difficult to have inequalities for processes such as our speed-sensitive order relation, because it is essentially formulated through a set of equational laws for precess expressions.

Remarks

Asynchrony, delay, and locality in communication are closely related with one another. The calculus developed in Chapter 3 is intended to establish a theory for reasoning about these aspects in a unified way.

5.3 Formalisms for Locality in Time

The present section compares with other formalisms for reasoning about locality in time. Temporal differences and uncertainties among physical clocks may distributed computing systems to inefficient or failure. Therefore, several methods for agreement on processes' time have been proposed by using clock synchronization techniques [40, 43, 45]. These methods are useful in keeping differences among physical clocks with a specified degree of accuracy. However, these methods are not available in all distributed systems. Even when such techniques are supported, if the required precision of time information is finer than the granularity of clock synchronizations, we need to take differences among clocks into consideration again.

However, most of existing formalisms for concurrent and distributed systems essentially assume the existence of the global clock. A few models can cope with the notion of local time. Andersen and Mendler propose a calculus with the notion of local time [2]. The calculus models the passage of time as multiple synchronous communication actions. Each process puts its own clocks forward by one time unit when only particular ones of the actions are performed. Therefore, the speeds of the passage of time in all processes are different. However, the calculus lacks any expressive capability of the properties of inaccurate clocks. As we know, there is no process calculus which can specify the properties of physical clocks quantitatively other than ours.

During the last ten years, there have been various attempts to extend temporal logic with the notion of quantitative time, for example see [32, 33, 41]. Most of them are based on the existence of a global clock. Among them, Corsetti and Montanari in [16, 55] extend real-time extended temporal logic with the ability to deal with different time granularities. However, the logic does not have any expressive capability of clocks whose measurement rates may drift. Wang, Al Mok and Emerson in [78] propose a real-time extended temporal logic with the ability to reason about multiple inaccurate clocks. They present inequalities for treating inaccuracies of clocks and a real-time model checking technique for reasoning about the inequalities. Alur, Courcoubetis, and Henzinger in [1] develop time-sensitive bisimulations for descriptions in temporal logic and automata having an explicit variable to indicate the passage of time. The bisimulations equate two descriptions according to the sensitivity of multiple constant

clocks instead of any inaccurate clocks.

Remarks

Among the calculi studied in this thesis, $RtCCS_L$ is characterized by the ability to specify the properties of physical clocks including various measurement errors quantitatively. It provides a powerful framework to analyze the influence of uncertainties and differences of physical clocks upon the temporal and behavioral properties of distributed systems.

5.4 Verification Techniques

The traditional method for verification in process calculi is to establish equivalence relations between two descriptions of a system; one typically corresponding to the specification of the system and the other corresponding to an implementation of the system. The relations equate two descriptions only by checking whether they cannot be distinguished from each other in their functions. When the two descriptions are equivalent, we conclude the implementation satisfies its specification. Several researchers have explored algebraic relations with time-sensitivity for the verification of time-dependent processes. We briefly survey existing time-sensitive relations.

Time-Sensitive Equivalence Relations

Many of the existing time-extended process calculi provide special equivalence relations over process expressions. The relations are formulated based on non-timed equivalence relations such as bisimulation equivalence [58], trace equivalence [11], failure equivalence [11], and testing equivalence [21], but they cannot distinguish between the timings of processes.

Several researchers have explored time extensions of the strong bisimulation [51], for example see [27, 14, 17, 26, 54, 57, 64, 75, 79]. The extended bisimulations equate two processes if they cannot be distinguished from each other in their temporal properties as well as their behavioral one. Most of them are difficult to have the concept of observation, because their timed semantics often disable internal actions to be ignored and cannot enjoy some pleasant properties of untimed observation

equivalences. Some researchers develop time-sensitive equivalence relations with the concept of observation, on the assumption that when an internal or communication action is enabled, processes must perform the action without imposing unnecessary idling, like ours. The relations can successfully inherit all the properties of untimed observation equivalences. Hennessy and Regan introduce a similar assumption and proposed time-sensitive testing equivalences in [31, 30]. Some of time-extended CSP works have time-sensitive equivalence relations based on trace equivalence and failure equivalence, for example see [18, 62].

Time-Sensitive Order Relations

We formulated an order relation over two processes with respect to their execution speeds. We compare with other existing order relations.

Speed-Sensitive Order Relations. Moller and Tofts in [54] proposed a preorder relation over processes with respect to their relative speeds. The relation is formulated based on the bisimulation concept. However, unlike ours, their calculus assumes to permit an executable communication to be suspended for arbitrary periods of time. As a result, their relation shows only that a process may *possibly* execute faster than the other. Also, their relation is essentially dependent on synchronous communication and thus cannot cope with asynchronous interactions among distributed processes.

Recently, Vogler in [77] presents a speed-sensitive preorder relation based on the testing equivalence concept [21].[1] The relation can relate asynchronously communicating processes according to their relative speed in the worst case scenario of their execution. However, the computational semantics of the relation is formulated based on causality between events on the assumption that actions are not instantaneous, unlike ours.

Cost-Sensitive Order Relations. Arun-Kumar and Hennessy in [3] propose an approach to relate processes with respect to their relative efficiencies based on the bisimulation concept [51]. Cleaveland and Zwarico in [15] propose a similar approach based on the testing equivalence concept [21].

[1] We should emphasize that our speed-sensitive order relation was proposed in 1994, independently of Vogler's work.

These approaches are based on algebraic order relations which relate two processes only if their behavioral properties are match and one of them can perform less internal actions τ than the other. That is to say, a process which needs to take more τ actions to execute the same properties than the other, is a slower process. However, both the approaches have some problems in the analysis of the efficiency of distributed systems. It is very difficult to reflect the execution cost of real systems upon descriptions of processes exactly. The relations intend to order two processes with respect to their *energetic* execution cost, instead of execution time.

Remarks

We summarize comparisons with existing verification techniques for time-dependent processes. Most of time-extended process calculi are given a semantics — the meaning of a time-dependent process is given by a tree of possible transitions, describing the possible behaviors of the process and their timings. As mentioned previously, several researchers have explored time-sensitive equivalences which are based on trace equivalence, failure equivalence, testing equivalence, and bisimulation equivalence like ours. Such time-sensitive equivalences are concord with the semantics of process calculi, because they relate between the tree structures of two processes. The equivalences provide a basis of the verification of time-dependent processes. However, there is a problem when they used in verifying real distributed systems. The equivalences can equate two behaviorally equivalent processes only when their temporal properties *completely* match with each other. Besides, the temporal properties of any two real distributed processes may not be completely coincide with each other, by virtue of the influences of inaccurate clocks and unpredictable communication latency. Therefore, such time-sensitive equivalences may be too strict to verify distributed processes. On the other hand, we defined a pseudo time-sensitive equivalence relation, which can equate two processes when their behavioral properties are equivalent and their temporal ones are different within a specified bound. The equivalence is more practical in the verification of real distributed systems. As far as we know, there is not such a pseudo equivalence relation other than ours.

Chapter 6

Conclusion

In this thesis we have seen ways to formulate temporal aspects of distributed computing, through developing an elementary process calculus for time-sensitive computing and its two further expressive extensions.

The elementary calculus is defined on the assumption that every process communicates synchronously and shares a global clock. Its language consists of two new time-dependent constructors: delay operator and time restriction operator, in addition to operational constructors found in many non-timed process calculi: sequential execution, parallel composition, synchronous communication, message scope, and recursion definition. It has enough expressive capabilities of temporal properties of real-time systems. The semantics of the language is given by means of two labeled transition systems, corresponding to the semantics of operational behaviors and that of the passage of time. Since the semantics of the time extensions does not interfere with that of operational behaviors, the calculus can enjoy many pleasant properties of non-timed process calculi including verification techniques.

On the basis of the calculus, we developed three time-sensitive equivalence relations over communicating processes: strong equivalence, observation equivalence, and observation congruence. These relations are formulated based on the notion of bisimulation. They can equate two processes if they cannot be distinguished from one another in their time properties as well as their behavioral properties. They offer a theoretical and practical framework for verifying real-time systems in a single processor and small distributed systems.

The two further extensions reinforce the calculus with the ability to express peculiar temporal properties of distributed computing. One of the extensions addresses asynchrony and delay in communications among distributed processes following a global clock. The extension enriches the original calculus with supplement language constructors: non-blocking message send and blocking message receive. It has the notion of process location and lets us to specify the latency of interprocess communication as the temporal distance between the locations of sender and receiver processes. Its semantics can be given through a set of rules that translate these extended supplement language constructions into the original calculus.

Furthermore, we defined a time-sensitive equivalence and order relations. The equivalence is characterized by equating two distributed processes according to the results of remote interactions with testing processes. The order relations can order two behaviorally equivalent processes with respect to their relative execution speeds. The relations guarantee that in asynchronous communication settings, a system embedding a faster process can still perform faster than the system embedding its slower one, without altering any functional behaviors. Since in asynchronous communication settings, it often is necessary only to verify that real-time processes can perform faster than what is required by their specification, these relations offer a suitable method for proving the correctness and the reusability of asynchronous communicating processes.

The other extension lets the original calculus embody the expressive capability of multiple inaccurate clocks. The extension has a special mapping to specify the precision of physical clocks quantitatively. The mapping also translates descriptions of processes following local clocks into the original calculus. By incorporating the expressiveness of the original calculus, the extension allows us to analyze the influences of inaccuracies of physical clocks upon distributed computing systems. Furthermore, we developed an equality based on the notion of local time. It equates two processes when their behaviors are completely matched and their timings are different within a given bound. It is useful in verifying distributed systems with inaccurate clocks and non-strict time constraints.

Time is a crucial factor in constructing distributed systems for the sake of archiving reliability and efficient cooperation with external systems and non computational systems. Nowadays, distributed systems are being more and more opened to their external environments, where time is real. The timed formalisms studied in this thesis

are getting important increasingly.

Future Work

There are many issues that we leave in this thesis. One of the most important issues is to investigate further refinements of these calculi. Here we would like to point out other further directions of this work.

Probabilistic Models. The present calculi lack any mechanism for reasoning about probability of systems. The mechanism allows us to analyze the performance of particular systems more exactly such as communication protocols with unreliable channels. We are interested in combining probability and time as stochastic Petri net.

Task Scheduling. When the number of available processors is less than that of active processes, a scheduling policy is required to make good use of the available processors. However, for simplicity, we intended to make minimal assumptions about the scheduling in this thesis. Studying a process calculus with the notion of concrete scheduling policies is an interesting issue.

Programming Languages. The calculi are expected to support formal reasoning at every stage of the development process of programs for distributed systems. Therefore, we should bridge the gap between our process calculi and real implementation languages. We are fortune in our having a general technique to encode parallel real-time languages into *RtCCS* expressions in [70, 69]. We need to enhance the technique to distributed programs by incorporating the expressive capabilities of the calculi developed in this thesis.

Tool Supports. We wish the present calculi to be adopted by industrial users. To do this, it is necessary to support reliable software tools to manipulate formal specifications and to assist verifications. We plan to develop some automatic reasoning systems for distributed processes described in the calculi by means of model checking, theorem solving approaches, and so on.

Open Systems. There are some distributed systems where new computers and new services will always come and change dynamically, for example the Internet and Open Information System [34]. In the other word, such systems are always incomplete and open to their internal and external changings. On the other hand, the existing specification techniques, including our calculi, are intended to reason about closed systems. A theoretical framework to specify such open distributed systems attracts much interest.

Bibliography

[1] R. Alur, C. Courcoubetis, and T. Henzinger. The Observational Power of Clocks. In *Proceedings of 4th CONCUR'94*, volume 826 of *Lecture Notes in Computer Science*, pages 162–177, 1994.

[2] H. R. Andersen and Mendler. M. An Asynchronous Process Algebra with Multiple Clocks. In *Proceedings of 5th European Symposium on Programming*, volume 788 of *Lecture Notes in Computer Science*, pages 58–73, 1994.

[3] S. Arun-Kumar. An Efficiency Preorder for Processes. *Acta Informatica*, 29:737–760, 1992.

[4] J. C. M. Baeten and J. A. Bergstra. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

[5] J. C. M. Baeten and J. A. Bergstra. Asynchronous Communication in Real Space Process Algebra. In *Proceedings, Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 473–492. Springer-Verlag, 1991.

[6] J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. In *Formal Aspects of Computing*, pages 142–188, 1991.

[7] J.C.M. Baeten and J.A. Bergstra. Real Space Process Algebra. Technical report, Eindhoven University of Technology, 1992. Computing Science Note 92/03.

[8] J.A. Bergstra and J.W. Klop. Process Algebra with Asynchronous Communication Mechanisms. In *Seminar on Concurrency*, volume 197 of *Lecture Notes in Computer Science*, pages 76–95. Springer-Verlag, 1985.

[9] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing Localites. *Theoretical Computer Science*, 114:31–61, 1993.

[10] E. Brinksma. A Tutorial on LOTOS. In V. M. Diaz, editor, *Proceedings of IFIP Workshop on Protocol Specification, Testing and Verification*, pages 73–84. North-Holland, 1986.

[11] S. D. Brookes, C. A. R. Hoare, and A. D. Roscoe. A Theory of Communicating Sequential Processes. *Journal of Association for Computing Machinery*, 31:560–599, 1984.

[12] J. Bryans, J. Davis, and S. Schneider. Towards a Denotational Semantics for ET-LOTOS. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 269–283. Springer-Verlag, 1995.

[13] H. Castellani and M. Hennessy. Observing Distributed in Processes. In *Proceedings of MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 321–331, 1993.

[14] L. Chen. An Interleaving Model for Real-Time Systems. Technical Report Technical report ECS-LFCS-91-184, University of Edinburgh, 1991.

[15] R Cleaveland and A. E. Zwarico. A Theory of Testing for Real-Time. In *Proceedings, 6th IEEE Symposium on Logic in Computer Science*, pages 110–119, 1990.

[16] E. Corsetti, A. Montanari, and E. Ratto. Dealing with Different Granularities in Formal Specifications of Real-Time Systems. *Journal of Real-Time Systems*, 3(2), May 1991.

[17] M. Daniels. Modeling Real-Time Behavior with an Interval Time Calculus. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[18] J. W. Davies. *Specification and Proof in Real-Time CSP*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994.

[19] J. W. Davies and S. A. Schneider. Factorising Proofs in timed CSP. In *Proceedings, the Fifth Conference on the Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[20] F. de Boer and J. Hooman. The Real-Time Behavior of Asynchronously Communicating Processes. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[21] E. de Nicola and M. Hennessy. Testing Equivalence for Processes. *Theoretical Computer Science*, 34, 1984.

[22] E.A. Emerson. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B. Elseviser Science Publishers, 1990.

[23] R. Gerber. *Communicating Shared Resource: A Model for Distributed Real-Time Systems*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, October 1991.

[24] R. Gerber and I. Lee. A Resource-Based Prioritized Bisimulation for Real-Time Systems. Technical Report PA 19104-6389, Department of Computer and Information Science, University of Pennsylvania, October 1990. Revised Version, 1992.

[25] R. Gerth and A. Boucher. A Timed Model for Extended Communicating Processes. In *Proceedings, Automata, Language and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 95–114. Springer-Verlag, 1987.

[26] J. Groote and F. W. Vaandrager. Structured Operational Semantics and Bisimulation as a congruence. In *Proceedings, Automata, Language and Programming*, volume 372 of *Lecture Notes in Computer Science*, 1989.

[27] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. PhD thesis, Department of Computer Systems, Uppsla University, Sweden, September 1991.

[28] H. Hansson and B. Jonsson. A Calculus of Communicating Systems with Time and Probabilities. In *Proceedings, 11th IEEE Real-Time Systems Symposium*, 1990.

[29] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[30] M. Hennessy. On Timed Process Algebra: a Tutorial. Technical Report Technical Report, 2/93, University of Sussex, 1993.

[31] M. Hennessy and T. Regan. A Temporal Process Algebra. Technical Report Technical Report, 2/90, University of Sussex, 1990.

[32] T. Henzinger, Z. Manna, and A. Pnueli. Temporal Proof Methodologies for Real Time Systems. In *Proceedings of Principle of Programming Languages*, pages 353–366, 1991.

[33] X. Henzinger, T. Nicollin and J. Sifaiki. Symbolic Model Checking for Real Time Systems. *Information and Computation*, 111:193–244, 1994.

[34] C. Hewitt and P. Jong. Open Systems. In *On Conceptual Modeling*. Springer-Verlag, 1984.

[35] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[36] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In *Proceedings, ECOOP'91*, volume 512 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[37] J. Hooman. *Specification and Compositional Verification of Real-Time Systems*. PhD thesis, Computer Science Department, Eindhoven University of Technology, 1991.

[38] C. Huizing, R. Gerth, and de Roever W.P. Full Abstraction of a Real-Time Denotational Semantics for an Occam-like Language. In *Proceedings of Symposium on Principles of Programming Languages*, pages 223–237. ACM Press, 1987.

[39] A. Kiehn, I. Castellani, M. Hennessy, and G. Boudol. Observing Localities. In *Proceedings, Symposium on Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[40] H. Kopetz. Interval Measurements in Distributed Real-Time Systems. In *Proceedings of IEEE 9th Conference on Distributed Computing Systems*, 1987.

[41] R. Koymans. *Specifying Message Passing and Time Critical Systems with Temporal Logic*. PhD thesis, Eindhoven University of Technology, 1989.

[42] P. Krishnan. Distributed CCS. In *Proceedings, 2th CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.

[43] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communication of the ACM*, 21(7):558–565, July 1978.

[44] G. Leduc. An Upward Compatible Timed Extension to LOTOS. In *Formal Description Techniques IV*, pages 217–232. North-Holland, 1992.

[45] J. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62:190–204, 1984.

[46] N.A. Lynch and M.R. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. Technical Report MIT/LCS/TR-87, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1987.

[47] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991.

[48] D.L. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Transaction on Communication*, pages 1482–1493, 1985.

[49] G. J. Milne. Circal and the Representation of Communication and Concurrency. *ACM Transactions on Programming Languages and Systems*, 1985.

[50] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[51] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[52] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes Part 1 & 2. *Information and Computation*, 1992. Also in Technical report ECS-LFCS-89-85 & 86, University of Edinburgh, 1989.

[53] F. Moller and C. Tofts. A Temporal Calculus of Communicating Systems. In *Proceedings, 1th CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–415. Springer-Verlag, 1990.

[54] F. Moller and C. Tofts. Relating Processes with Respect to Speed. In *Proceeding, 2th CONCUR'91*, volume 527 of *Lecture Notes in Computer Science*, pages 424–438. Springer-Verlag, 1991.

[55] A. Montanari, E. Ratto, E. Corsetti, and A. Morzeniti. Embedding Time Granularity in Logical Specification of Real-Time Systems. In *Proceedings of EUROMICOR'91, Workshop on Real-Time Systems*, pages 88–97, June 1991.

[56] U. Montanari and D. Yenkelevich. A Parametric Approach to Localities. In *Proceedings, Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 617–628. Springer-Verlag, 1992.

[57] X. Nicollin and J. Sifakis. The Algebra of Timed Process ATP: Theory and Applications. Technical report, IMAG Technical Report, 1990.

[58] D. Park. Concurrency and Automata on Infinite Sequences. In *Proceedings, Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[59] G. D. Plotkin. Structural Approach to Operational Semantics. Technical report, Department of Computer Science, Arhus University, 1981.

[60] A. Pnueli. The Temporal Logic of Programming. *Theoretical Computer Science*, 13:46–57, 1986.

[61] G. M. Reed and A. W. Roscoe. A Timed Model for Communicating Sequential Processes. In *Proceedings, Automata, Language and Programming'86*, volume 226 of *Lecture Notes in Computer Science*, pages 314–323. Springer-Verlag, 1986.

[62] G. M. Reed and A. W. Roscoe. Analysing Mtfs: A study of nondeterminism in real-time concurrency. In *Proceedings, Concurrency: Theory, Language, and Architecture*, volume 491 of *Lecture Notes in Computer Science*, pages 36–63. Springer-Verlag, 1991.

[63] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs*. Springer-Verlag, 1985.

[64] I. Satoh and M. Tokoro. A Formalism for Real-Time Concurrent Object-Oriented Computing. In *Proceedings of Conference Object-Oriented Programming Systems, Languages, and Applications (ACM OOPSLA'92)*, pages 315–326, October 1992.

[65] I. Satoh and M. Tokoro. A Formal Description for Parallel Processes with Time Properties. *Transactions of Information Processing Society of Japan*, 35:540–548, 1993. (in Japanese).

[66] I. Satoh and M. Tokoro. A Timed Calculus for Distributed Objects with Clocks. In *Proceedings of European Conference on Object Oriented Programming (ECOOP'93)*, volume 707 of *Lecture Notes in Computer Science*, pages 326–345. Springer-Verlag, July 1993.

[67] I. Satoh and M. Tokoro. A Formalism for Remotely Interacting Processes. In *Proceedings of Workshop on Theory and Practice of Parallel Programming (TPPP'94)*, volume 907 of *Lecture Notes in Computer Science*, pages 216–228. Springer-Verlag, November 1994.

[68] I. Satoh and M. Tokoro. Modeling Distributed Processes with Local Time. *Computer Software*, 11:32–44, 1994. (in Japanese).

[69] I. Satoh and M. Tokoro. Process Algebra Semantics for a Real Time Object Oriented Programming Language. *Transactions of Information Processing Society of Japan*, 35:2355–2365, 1994. (in Japanese).

[70] I. Satoh and M. Tokoro. Semantics for a Real-Time Object-Oriented Programming Language. In *Proceedings of IEEE 5th International Conference on Computer Languages (ICCL'94)*, pages 159–170, May 1994.

[71] I. Satoh and M. Tokoro. Time and Asynchrony in Interactions among Distributed Real-Time Objects. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP'95)*, Lecture Notes in Computer Science, pages 331–350. Springer-Verlag, August 1995.

[72] F.B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach. *ACM Computing Surveys*, 22:299–319, 1990.

[73] R. Simone. Higher-level Synchronizing Devices in MEIJE-SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[74] C. Stirling. Modal and Temporal logics for Processes. In *Handbook of Logic in Theoretical Computer Science*, volume 2. Elsevier Science Publishers, 1993.

[75] C. Tofts. *Proof Methods and Pragmatics for Parallel Programming*. PhD thesis, Department of Computer Science, University of Edinburgh, 1991. Also published in as Technical Report ECS-LFCS-91-140.

[76] M. Tokoro and I. Satoh. Asynchrony and Real-time in Distributed Computing. In *Proceedings of Parallel Symbolic Computing*, volume 734 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[77] W. Volger. Faster Asynchronous Systems. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, pages 299–312. Springer-Verlag, 1995.

[78] F. Wang, A. Mok, and E. A. Emerson. Real-Time Distributed Systems Specification and Verification in APTL. *ACM Transaction on Software Engineering and Methodology*, pages 346–378, 1993.

[79] Yi Wang. *A Calculus of Real-Time Systems*. PhD thesis, Computer Science Department, Chalmers University of Technology, 1991.

[80] Yi Wang. CCS + Time = an Interleaving Model for Real Time Systems. In *Proceedings of Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.