Timed Processes:
Models, Axioms and Decidability

Liang Chen

Doctor of Philosophy

University of Edinburgh
1992

Abstract

This thesis presents and studies a timed computational model of parallelism, a Timed Calculus of Communicating Systems or Timed CCS for short. Timed CCS is an extension of Milner's CCS with time. We allow time to be discrete, such as the natural numbers, or dense, such as the non-negative rationals or the non-negative reals. We make no assumption of the Maximal Progress Principle, but the calculus is consistent with the principle. Time variables in the language allow us to express a notion of time dependency and the language is more expressive than those without time variables or infinite summation. We extend the well known notion of bisimulation to timed processes and study the abstract semantics of timed processes. We show that strong equivalence (the largest strong bisimulation) is decidable for finite processes, i.e. processes without recursion. The decidability is independent of the choice of time domain. We also present a simple proof system for strong equivalence and the proof system is again independent of the choice of time domain. We show that the proof system is sound and complete for finite processes over dense time domains, but only complete for a restricted language over discrete time domains. We discuss how to modify the definition of time expressions to get the restricted language. We also study behavioural abstraction in timed processes.

The thesis also presents and studies a general model, Timed Synchronisation Trees, for timed calculi. Timed synchronisation trees are extensions of synchronisation trees with time. All constructions on timed synchronisation trees are continuous with respect to a natural complete partial order. We can interpret a wide range of real-time process algebras in timed synchronisation trees. As an example, we give a denotational semantics for Timed CCS based on timed synchronisation

trees. We show that the denotational and operational semantics of Timed CCS coincide.

CCS is a symbolic calculus in the sense that it treats solely the observation of events of a system. The relative time, location and duration of events are abstracted away from the consideration. If we postulate that every action has a non-zero constant duration, we can observe the usual notions of causality, concurrency and conflict relations of events of a system. By interpreting CCS in Timed CCS based on a postulation that for any two events which are causally related there is at least a non-zero constant delay between them, we get a timed semantics for CCS. The timed semantics of CCS is a partial order or true concurrency semantics. As a consequence, we develop a partial order or true concurrency semantics based on an interleaving approach.

Acknowledgement

I would like first of all to thank my first supervisor Stuart Anderson and second supervisor John Power for their help and many invaluable discussions that greatly influenced my work. From them I have learned a lot about research methodology. If I had followed their advice, the contents and presentation of the thesis would be much better.

I also owe a great debt to Faron Moller for his insights, for many fruitful discussions, helpful comments and suggestions, and for reading a draft of this thesis while he was visiting Japan.

I would also like to thank Jos Baeten, Jan Friso Groote, Mark Jerrum, Kim Larsen, Robin Milner, Alistair Munro, Peter Sewell, Colin Stirling and Chris Tofts for their helpful discussions and comments, especially to Robin Milner for his suggestion on studying the models of timed calculi in my thesis proposal committee. Thanks also go to the people in concurrency club for their patience in listening even my half baked ideas and their useful comments.

I want to thank to the staff, students and friends in Edinburgh who enabled me stay and work in such a good environment. Thanks especially go to Davide Sangiorgi for organizing football matches for us and to Soren Christensen for playing badminton.

My thanks to my wife Hong is impossible to be enough in any sense.

My stay in Edinburgh was supported by a TC scholarship of British Council.

Declaration

I hereby declare that this thesis has been composed by myself, the idea, the work and result which I do not attribute to others are due to myself.

Chapter 8 is joint work with Stuart Anderson and Faron Moller [ACM92].

This is the revised version of my thesis incorporating the required corrections suggested by my examiners Kim Larsen and Faron Moller.

Table of Contents

Chapter 1

Introduction

There are two main approaches to the semantics of concurrency, namely the interleaving approach (among others [Mil80,Mil89a,Hoa85,BK84,Hen88a]) and the partial order or true concurrency approach (as in [Rei85,NPW81,BC89,DDM89, Win87]).

Among the interleaving models, Milner's CCS [Mil80,Mil89a] can be considered to be a standard representative. In [Mil80,Mil89a] Milner has developed a semantic theory of processes based on a notion of bisimulation equivalence. Bisimulation semantics is an elegant mathematical theory of processes. It provides simple and elegant proof techniques for showing that a process implementation meets its specification. However, in this framework, some researchers claim concurrency is not handled in a natural way. The parallelism among actions is described by saying that they can happen in any order. As a consequence, concurrency is reduced to nondeterminism and every concurrent process is equivalent to a nondeterministic sequential process. For any single observation of the behaviour of the system the actions are seen to occur in sequence even though some actions are not causally related. In general, the causality relation and concurrency relation are not observable in interleaving models. For example, in CCS, the process $a \mid b$ is equivalent to the process $ab + ba$. In SCCS [Mil83], multisets of actions are allowed thereby providing us a direct representation of "parallelism". As a result, the process $\underline{a} \mid \underline{b}$, where we write $\underline{a}$ in place of δa of SCCS, can be distinguished from the process $\underline{a}\,\underline{b} + \underline{b}\,\underline{a}$. However, causal dependencies are still not recoverable. As an example, process $\underline{a} \mid \underline{b} + \underline{a}\,\underline{b}$ cannot be differentiated from process $\underline{a} \mid \underline{b}$ in SCCS.

The partial order or true concurrency approach provides a more faithful account of the *causality*, *concurrency* and *conflict* or *incompatibility* relations between events (actions together with their locations) of a system. It describes concurrent computations by means of partial orders. Concurrency between events is represented by the absence of orderings. This approach deals with properties related to knowledge of concurrent activities such as deadlock, fairness, starvation, etc. However, techniques for defining and handling partial orderings are rather weak and there are no corresponding standard mathematical constructions. Among partial order or true concurrency models, Petri Nets [Rei85] and Event Structures [NPW81,Win87] can be considered to be representative.

The traditional methods for reasoning about nondeterministic or concurrent systems, such as [Mil89a,Hoa85,NPW81,Rei85], do not consider hard time aspects of systems. Instead they deal with the quantitative aspects of time in a qualitative way and abstract away explicit time information. There are many systems and applications for which purely qualitative specification and analysis are inadequate. Typical examples are real-time systems, in which the interaction with the environment must satisfy some time constraints. What is important in designing and analysing real-time systems are the significant events and their relative time information.

Real-time systems include fault-tolerant systems, such as protocols, and safety critical systems, such as flight control and radiation control systems. In these systems, it is not sufficient to only say events occur or eventually occur. There are lower and upper bounds on when events can occur relative to the other events. For example, consider an implementation of Alternating Bit Protocol which is based on unreliable transition lines. The transition lines may lose messages. After sending a message to the receiving part via a transition line, the sending part must wait for a certain time, e.g. time Δ, which is the time out of waiting for an acknowledgement before assuming the message has been lost from the transition line. If the sending part receives a correct acknowledgement within time Δ, then it becomes ready to transmit another message. Otherwise, it retransmits the message after a delay of Δ time.

Another important aspect of time critical behaviour of real-time systems is time dependency, which says that the time constraints of some actions depends on the happening time of preceding actions. For example, consider a machine which can perform action a followed by action b. Action b causally depends on action a. There is at least a delay of time Δ between the occurrences of a and b, and the machine must perform both actions a and b within 25 *seconds*.

In this thesis, we study the possibility of incorporating time explicitly to the well developed techniques, such as synchronisation trees and process algebras, to reason about quantitative as well as qualitative timing behaviour of systems. We first present a general model, Timed Synchronisation Trees, for Real-Timed Calculi. Timed synchronisation trees are extensions of synchronisation trees with time. We make no assumption about the underlying nature of time, allowing time to be discrete (such as the natural numbers $\mathbb{N}$) or dense (such as the non-negative rationals $\mathbb{Q}^{\geq 0}$ or the non-negative reals $\mathbb{R}^{\geq 0}$). All constructions on timed synchronisation trees are continuous with respect to a natural complete partial order of timed synchronisation trees. This allows timed synchronisation trees to be used to define denotational semantics for a wide range of timed calculi.

We also present a timed calculus, Timed CCS, which is an extension of Milner's CCS with time. We again make no assumption about the underlying nature of time, allowing it to be discrete or dense. Atomic actions take no time, i.e. they are instantaneous. This assumption is essential if we are to use an interleaving model for real-time systems. For an asynchronous system, any two events which are not causally related or in conflict may occur not at the same time but at times which are arbitrarily close to each other. We say two events occur at the same time if one follows the other without any non-zero delay between them. However, this assumption also means that we can define processes (usually called the *Zeno* machines) which perform an infinite sequence of events in finite time or perform infinite events without time progress. To be more realistic, we may insist that for any two events which are causally related there should be some non-zero constant delay between them. This may be understood as saying that every action has a non-zero constant duration. We have no such realistic assumption when we develop the theory of Timed CCS and try to keep the calculus as general as possible. It is a task for the user to write realistic processes.

Postulating that every atomic action has a non-zero constant duration allows the usual notions of causality, concurrency and conflict to be observed within Timed CCS in the following sense:

- two events are causally related if both of them can occur in the same run but never at the same time, where we say two events can occur in the same run if one event occurs first and then, after a sequence of events, the other event can occur;
- two events are in the concurrency relation if they can occur at the same time;
- two events are in the conflict relation if they cannot both occur in the same run.

We satisfy this condition (that every atomic action has a non-zero constant duration) by saying that for any two events which are causally related there should be at least some non-zero constant delay between them. We can interpret CCS processes within Timed CCS based on this postulation. As a consequence, we obtain a partial order or true concurrency semantics for CCS. Since Timed CCS is an interleaving model, we in fact develop a partial order or true concurrency semantics based on an interleaving model.

In the remainder of this introduction, we briefly outline the background knowledge and references for our study before giving an outline of the thesis.

1.1 Background

1.1.1 CCS

Process algebras like Milner's *Calculus of Communicating Systems* (CCS) [Mil80, Mil89a], Hoare's *Communicating Sequential Processes* (CSP) [BHR84,Hoa85] and Bergstra and Klop's *Algebra of Communicating Processes* (ACP) [BK84,BK85] view concurrent systems as structured entities which interact via synchronisation mechanisms. Among these process algebras, we regard here Milner's CCS as a standard representative. In CCS, agents (or processes) are able to perform actions and their complementary co-actions. A synchronisation between agents is achieved when a pair of complementary actions occur simultaneously. The result of a synchronisation is to eliminate the (action, co-action) pair and replace it by a silent τ action.

The Language

To give a formal description of the syntax of CCS, we presuppose a set Λ of action names which does not contain τ, ϵ, and is ranged over by a, b, c, possibly indexed. Let $Act = \Lambda \cup \{\tau\}$ which is ranged over by α, β. We also presuppose a structure on Λ that it can be partitioned into Γ, the set of names, and $\overline{\Gamma} = \{\bar{a} : a \in \Gamma\}$, the set of co-names, with the provision that $\bar{\bar{a}} = a$. We call actions a and $\bar{a}$ a pair of complementary actions which forms the basis of communications in CCS. Let V_p be an infinite set of process variables, ranged over by X, Y. The process expressions of CCS, ranged over by E, F, is defined by the following BNF expression:

$$E ::= X \;\Big|\; nil \;\Big|\; \alpha.E \;\Big|\; E + F \;\Big|\; E \mid F \;\Big|\; E\backslash a \;\Big|\; E[S] \;\Big|\; \mu X.E$$

where $S : \Lambda \longrightarrow \Lambda$ is a relabelling function which satisfies $\overline{S(a)} = S(\bar{a})$. By convention, we have $S(\tau) = \tau$.

The process *nil* cannot perform any action. The prefix $\alpha.E$ represents a process which can perform action α. After doing so, it becomes the process E. We will often omit the "*nil*" in the term "$\alpha.nil$". Also, we will often drop the "." of "$\alpha.E$". For example, we render $a.b.nil + b.a.nil$ as $ab + ba$. The summation $E + F$ represents a process which can behave as process E or process F. The choice is made at the time of the first action of E or F. Process $E \mid F$ represents parallel composition of processes E and F. Both E and F proceed independently. Synchronisation may occur when they perform a pair of complementary actions representing communication between them. The restriction $E\backslash a$ represents a process which behaves as process E except that action a and its complementary action $\bar{a}$ are forbidden. Relabelling $E[S]$ represents the process derived from process E by relabelling its action using the relabelling function S. Process $\mu X.E$ is an infinite process which is defined by an equation $X = E$. The operator μX in process $\mu X.E$ binds all free occurrences of process variable X in process E. This give us in the usual sense the notions of free and bound occurrences of process variables. We say a process is an agent if it does not contain any free occurrence of process variables. We usually use $\mathcal{P}$ to represent the set of all agents, ranged over by P, Q.

Transition Systems

The operational semantics of CCS is given in terms of labelled transition systems, which are a general model of computation described in [Kel76]. A labelled transition system is a pair $(S, \{\xrightarrow{t} \ : \ t \in T\})$ where S is a set of states, T is a set of transition labels and $\xrightarrow{t} \subseteq S \times S$ is a transition relation (for every t of T). For convenience, we write $s \xrightarrow{t} s'$ in place of $(s, s') \in \xrightarrow{t}$. Intuitively, a transition of the form $s \xrightarrow{t} s'$ indicates that the system, when in state s, may perform action t and in so doing evolves to state s'.

The operational semantics of CCS is defined by transition relations of the

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{P \xrightarrow{\alpha} P'}{Q \mid P \xrightarrow{\alpha} Q \mid P'}$$

$$\frac{P \xrightarrow{a} P', \;\; Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a} \qquad a \neq \alpha \wedge \bar{a} \neq \alpha$$

$$\frac{P \xrightarrow{\alpha} P'}{P[S] \xrightarrow{S(\alpha)} P'[S]}$$

$$\frac{E\{\mu X.E/X\} \xrightarrow{\alpha} P}{\mu X.E \xrightarrow{\alpha} P}$$

Table 1–1: Operational Rules for CCS

form $P \xrightarrow{\alpha} Q$ of Plotkin's *Structured Operational Semantics* (SOS) [Plo81] style. Table 1–1 presents all transition rules, which are in natural deduction style and have the following form:

$$\frac{T_1, \cdots, T_n}{T}$$

Transition rules of the above form have the following interpretation: T is derivable if all $T_1, \cdots, T_n$ are derivable.

A transition T is allowed if and only if it can be derived by using the transition rules of Table 1–1. The operational semantics of CCS is given by the least transition relations $\xrightarrow{\alpha}$, where $\alpha \in Act$, defined in Table 1–1.

Synchronisation Trees

Synchronisation trees [Mil80,Win84a] are a concrete model for much of the work on concurrency. They are trees with the vertices representing states and edges labelled by actions which show how to synchronise with the environment.

If we collect the derivatives of a CCS agent P into a tree form, we get a derivation tree of P. For example, the derivation tree of CCS agent $a(b+c)$ is presented in Figure 1–1. The derivation trees of CCS agents correspond to synchronisation trees.

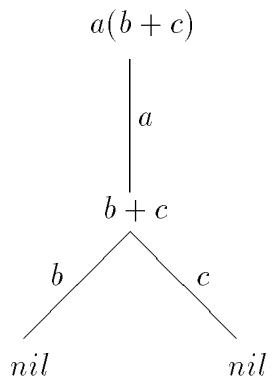


Figure 1–1: The Derivation Tree of $a(b+c)$

By introducing an appropriate ordering on synchronisation trees, we get a complete partial order of such structures. Furthermore, natural constructions defined over such trees are continuous with respect to this ordering, so we can naturally define infinite synchronisation trees. As a result we can give a denotational semantics for CCS based on synchronisation trees [Win84a].

Bisimulations

A more abstract semantics for CCS is achieved by identifying those agents which cannot be distinguished by any external observation of their behaviour. A notion of bisimulation [Mil89a,Par81] is a standard device for defining behavioural equivalence for process algebras.

We say a binary relation $\mathcal{R}$ over agents is a strong bisimulation if for any agents P and Q, $(P, Q) \in \mathcal{R}$ implies that for any $\alpha \in Act$,

(1) whenever $P \xrightarrow{\alpha} P'$, then $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{R}$ for some Q';

(2) whenever $Q \xrightarrow{\alpha} Q'$, then $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{R}$ for some P'.

Two agents P and Q are strongly bisimilar, written $P \sim Q$, if there is a strong bisimulation which contains (P, Q). The relation $\sim$ itself is a strong bisimulation, the largest strong bisimulation. It is an equivalence relation, called strong equivalence. Moreover it is a congruence relation, i.e. it is substitutive for all operators including recursive operator.

Strong bisimulation treats the internal action τ in the same way as all other actions and requires every τ action in one agent be matched by a τ action in the other. By relaxing the requirement and allowing a τ action to be matched by *zero* or *more* τ actions, we obtain the notion of weak bisimulation.

Let $\hat{\tau}$ be ϵ and $\hat{a}$, where $a \in \Lambda$, be a. We write $P \stackrel{\epsilon}{\Longrightarrow} P'$ in place of $P(\xrightarrow{\tau})^* P'$ and $P \stackrel{\alpha}{\Longrightarrow} P'$ in place of $P(\xrightarrow{\tau})^* \cdot \xrightarrow{\alpha} \cdot (\xrightarrow{\tau})^* P'$. Note that for every CCS agent P, we have $P \stackrel{\epsilon}{\Longrightarrow} P$. A binary relation $\mathcal{R}$ over agents is a weak bisimulation if $(P, Q) \in \mathcal{R}$ implies that for any $\alpha \in Act$,

(1) whenever $P \xrightarrow{\alpha} P'$, then $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{R}$ for some Q';

(2) whenever $Q \xrightarrow{\alpha} Q'$, then $P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ and $(P', Q') \in \mathcal{R}$ for some P'.

Similarly two agents P and Q are weakly bisimilar, written $P \approx Q$, if there is a weak bisimulation containing (P, Q). The relation $\approx$ itself is a weak bisimulation, the largest weak bisimulation. Moreover it is an equivalence relation, but not a congruence relation. In general, the summation operator does not preserve weak bisimilarity. For example, $a \approx \tau a$ but $a + b \not\approx \tau a + b$.

Since the relation $\approx$ is not preserved by summation, we define a relation $=^+$ by saying that $P =^+ Q$ if $P + R \approx Q + R$ for any agent R. The relation $=^+$ is a congruence relation. In fact it is the largest congruence relation contained in weak equivalence $\approx$.

1.1.2 Hennessy-Milner Logic

An alternative characterization of bisimulation is given by Hennessy and Milner [HM85] based on a simple modal logic which is often referred to as Hennessy-Milner Logic. This further confirms the naturalness of the notion of bisimulation.

The formulas of Hennessy-Milner logic, ranged over by φ, ψ, are defined by the following BNF expression:

$$\varphi \ ::= \ tt \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle a \rangle \varphi$$

where $a \in A$. The formula $\langle a \rangle \varphi$ is called a *modalized* formula with the intended meaning that after some a action φ holds. The class of modalized formulas is parameterized by a label set A. So the class of transition systems over a given label set has a Hennessy-Milner logic associated with it.

The formulas of Hennessy-Milner logic are interpreted over a transition system of the form $(S, \{\xrightarrow{a} \ : \ a \in A\})$. As CCS defines a transition system $(\mathcal{P}, \{\xrightarrow{\alpha} \ : \alpha \in Act\})$, the formulas of Hennessy-Milner logic can be interpreted over CCS as follows:

(1) for every agent P, we have $P \models tt$;

(2) $P \models \neg\varphi$ if and only if *not* $P \models \varphi$;

(3) $P \models \varphi \wedge \psi$ if and only if $P \models \varphi$ and $P \models \psi$;

(4) $P \models \langle \alpha \rangle \varphi$ if and only if *there is a transition* $P \xrightarrow{\alpha} P'$ *such that* $P' \models \varphi$.

An important derived operator is $[\alpha]$, the dual of $\langle \alpha \rangle$, i.e. $[\alpha]\varphi \stackrel{def}{=} \neg\langle \alpha \rangle \neg\varphi$. The formulas of Hennessy-Milner logic can be used to specify the desired properties of a nondeterministic or concurrent system. As an example, consider the vending machine V of [Mil89a] where

$$V \stackrel{def}{=} 2p.big.collect.V + 1p.little.collect.V$$

We can specify the following interesting properties of vending machine V:

1. after $2p$ is deposited, only the big button can be pressed

$$V \models [2p]([little]ff \wedge \langle big \rangle tt)$$

2. after $1p$ is deposited and the little button is pressed, a chocolate can be collected

$$V \models [1p][little]\langle collect \rangle tt$$

In order to support modular design and verification, a problem of compositionality of modal assertions has been raised. This problem has been successfully dealt with by the works of [Sti87,Win84b,LL90] for CCS and SCCS. In [Lar88, Cle90,BS92,SW89,Win89], proof systems, tableau methods and rewrite rules for Hennessy-Milner logic with recursion have been proposed. This allows us to reason about temporal properties of processes which are to hold invariantly or eventually.

1.1.3 Labelled Event Structures

Labelled event structures [NPW81,Win87] represent processes by means of events together with a relation of conflict, which describes how occurrences of certain events exclude others, and a relation of enabling, which describes how events are causally related. Events may occur concurrently if they are not in conflict or enabling relations. The concurrency relation between events is presented by the absence of the conflict and enabling relations.

An A-labelled event structure is a quadruple $S = (E, \prec, \#, \ell)$ where

- $(E, \prec)$ is a partial ordering of events, where $\prec$ is called the causality or precondition relation, such that for every $e \in E$, the set $\{e' \in E : e' \prec e\}$ is finite;

- $\#$, the conflict relation, is a symmetric and irreflexive binary relation on E which is hereditary in the sense that if $e_1 \# e$ and $e \prec e_2$ then $e_1 \# e_2$; and

- $\ell : E \longrightarrow A$ is a labelling function from events to actions.

Two events of E are in the concurrency relation if they are neither in the causality relation nor in the conflict relation. The concurrency relation is a symmetric and irreflexive binary relation of E and the causality, conflict and concurrency relations form a partition of $E \times E$.

For any A-labelled event structure $S = (E, \prec, \#, \ell)$, a subset $F \subseteq E$ of events is said to be a configuration or computation of S if F is conflict free and closed under preconditions. As the conflict relation $\#$ is irreflexive and hereditary, a configuration is conflict free, i.e. for any events e and e' of a configuration, relation $e \# e$ does not hold. Viewing computations of an A-labelled event structure as such subsets of events, progress in a computation can be measured by the occurrence of more events. Denoting the set of all computations of S by $\mathcal{E}(S)$, we know that $(\mathcal{E}(S), \subseteq)$ is a domain which enjoys many nice properties [Win87].

Labelled event structures are used to provide true concurrency semantics for process algebras. In [Win83], a denotational semantics of CCS based on labelled event structures has been proposed. However, the labelled event structure semantics is still very intensional (similar to the interleaving approach). Some notions of equivalence for event structures which preserve concurrency have been proposed in the literature. These include Boudol and Castellani's pomset bisimulation [BC87,BC89] and Degano, De Nicola and Montanari's partial order bisimulation [DDM89].

1.2 Layout of the Thesis

In Chapter 2, we present a mathematical model, timed synchronisation trees, for timed calculi. A timed synchronisation tree is a rooted tree with vertices representing states and edges labelled by actions to show how to synchronise with the environment. The vertices are labelled by time information which shows the maximal delay time of the vertices and the time constraints over actions labelling the path from the root to the vertices. All constructions of timed synchronisation trees are continuous with respect to a natural complete partial order. This allows us to recursively define infinite timed synchronisation trees along standard lines. Since trees are concrete interleaving models, we define an equivalence relation on timed synchronisation trees based on a notion of behaviour of timed synchronisation trees.

In Chapter 3, we give a formal description for Timed CCS, including its syntax and semantics. We present its operational and denotational semantics. The operational semantics is in Plotkin's SOS style and the denotational semantics is based on timed synchronisation trees. Time variables in the language allow us to express the notion of time dependency. We also introduce some derived operators which will be used in the thesis. We present three examples to show the utility of the calculus for the specification and verification of real-time systems. The examples illustrate three important aspects of real-time systems, i.e. time out, duration control and time dependency. In this chapter, we also discuss our decisions on the design of the calculus and compare it with related work.

In Chapter 4, we extend the notion of strong bisimulation to timed processes and show that strong equivalence enjoys many desirable properties including an expansion law. Strong equivalence is a congruence relation. We show the two semantics (operational and denotational) of Timed CCS are identical in the sense that the equivalences on Timed CCS induced by the denotational and operational semantics coincide. Equations which satisfy some condition are shown to have unique solutions up to strong equivalence. We also present a simple modal logic which is an extension of Hennessy-Milner logic with time and show a modal characterization of strong bisimulation. This not only shows that bisimulation is a natural equivalence, but also suggests that modal logic is an appropriate program logic for the specification and verification of real-time systems.

In Chapter 5, we show that strong equivalence is decidable for finite processes, i.e. those processes without recursion. The decidability is independent of the choice of time domain, allowing time to be discrete or dense.

In Chapter 6, we present a simple proof system for strong equivalence. The proof system is also independent of the choice of time domain. It is sound and complete for finite processes over dense time domains, but only complete for a restricted language over discrete time domains. We show by an example how the proof system works for recursively defined processes.

In Chapter 7, we discuss behavioural abstraction of timed processes. As in CSS, weak bisimulation in Timed CCS is not fully substitutive. It is not preserved by the summation operator. We refine weak bisimulation and define an observational congruence relation which is the largest congruence relation included in weak bisimulation. We show that equations which satisfy some conditions have unique solutions up to observational congruence.

In Chapter 8, we show that if we postulate that every atomic action has a non-zero constant duration (we satisfy this by saying that for any two events which are causally related there should be at least a non-zero constant delay between them), then we can observe the causality relation in real-time calculi. We interpret CCS processes within Timed CCS based on a postulation that for any two events which are causally related there is at least a delay Δ (where $\Delta > 0$) between them. As a result, we obtain a timed semantics for CCS. The time semantics of CCS is a partial order or true concurrency semantics. We in fact develop a partial order or true concurrency semantics for CCS based on an interleaving approach.

In Chapter 9, we summerise the thesis and discuss future research topics and directions.

Chapter 2

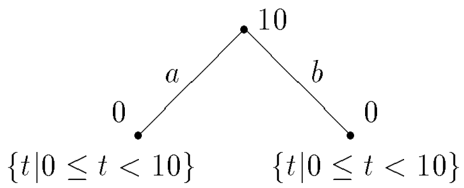
Timed Synchronisation Trees

Tree semantics arise naturally when concurrency is simulated by nondeterministic interleaving. In [Mil80], Milner uses synchronisation trees as an interleaving model for parallel computation in which processes communicate by mutual synchronisation. By further labelling the vertices of synchronisation trees with time constraints, we obtain timed synchronisation trees. The time constraints associated with the vertices of trees represent the maximal delay time of the vertices and the time at which the actions labelling the paths from the root to the vertices can occur in order to arrive at the vertices. Timed synchronisation trees are a general interleaving model of real-time systems in which processes communicate by mutual synchronisation, analogous to synchronisation trees. For example, the following tree

$$\begin{array}{l} \bullet \\ \big| \; a \\ \bullet \;\; \{t|t \geq 0\} \\ \big| \; b \\ \bullet \;\; \{(t,t')|t \geq 0 \& 0 \leq t' \leq t\} \end{array}$$

can be used to express a machine which can first perform action a at time t, where $t \geq 0$, and after doing so it can perform action b at a relative time t', where

$0 \leq t' \leq t$. However, in real-time systems, one important notion is time out. There are lower and upper bounds on when events can occur relative to the other events. Consider a machine M which can perform action a or b, but not both. Action a and b can occur before 10 minutes. If a and b do not occur, then the machine M evolves to a dead state after a delay of 10 minutes. To express the machine M, we need to represent a notion of the maximal delay time of a state of a machine. For example, the following tree can be used to represent the machine M:



2.1 A Description of the Model

In this section, we give a formal description of the *timed synchronisation trees.*

2.1.1 Timed Synchronisation Trees

Timed synchronisation trees are certain kinds of rooted trees. The vertices of trees represent states and the edges represent event occurrences which cause changes of states. Edges and vertices are both labelled. The edges are labelled by elements of a nonempty set A of actions, which the machine under consideration may perform. The labels over edges show how the machine synchronises with the environment. The vertices are labelled with time constraints which express the maximal delay time of the machine at the vertices and time at which the actions labelling the path from the root to the vertices can happen in order to arrive at the vertices. To define timed synchronisation trees, we first define graphs.

Definition 2.1.1 *A* graph *G is a pair (V, E), where V is a set of vertices and E, a subset of the set of unordered pairs of V, is a set of edges.*

An edge $\langle v,\ v' \rangle$ joins the vertices v and v', which in turn are called the *end-vertices* of $\langle v,\ v' \rangle$. Two edges are adjacent if they have exactly one common end-vertex. A *path* between vertices v_1 and v_{n+1} in G is a sequence of edges $e_1, \cdots, e_n$ of G, where $e_1 = \langle v_1, v_2 \rangle, \cdots, e_n = \langle v_n, v_{n+1} \rangle$ are all distinct. The *length* of a path p is the length of the sequence p. A path p in G is a *cycle* if its length is not 0 and it is a path between a vertex v and itself.

Definition 2.1.2 *A* tree *is a graph (V, E) with a special vertex $r \in V$,* the root of the tree*, such that*

(1) *for any $v \in V$ there is a path between r and v;*

(2) *there are no cycles in (V, E).*

Note that a tree contains at least one vertex, *the root of the tree.* We use (V, E, r) to represent a tree with the set V of vertices, the set E of edges and the root r. The *height* of a vertex v of a tree (V, E, r), written $h(v)$, is defined to be the length of the path between v and r.

Definition 2.1.3 *An A-*labelled tree *is a tree (V, E, r) together with a labelling function $l : E \longrightarrow A$.*

We use (V, E, r, l) to represent an A-labelled tree with the tree (V, E, r) and the labelling function $l : E \longrightarrow A$.

For convenience, we will say a labelled tree instead of an A-labelled tree when A is clear from the context.

To define timed synchronisation trees, we presuppose a time domain $(\mathcal{T} \cup \{\infty\}, \leq)$, where $\mathcal{T}$ contains a least element 0 which represents the starting time and $\leq$ is a linear order over $\mathcal{T}$. We introduce an infinite time ∞, $\infty \notin \mathcal{T}$, which satisfies that $u \in \mathcal{T}$ implies $u \leq \infty$. We make no assumption about the underlying nature of time, allowing $\mathcal{T}$ to be discrete, such as $\mathbb{N}$ (the set of natural numbers), or dense, such as $\mathbb{Q}^{\geq 0}$ (the non-negative rationals) or $\mathbb{R}^{\geq 0}$ (the non-negative reals).

Notation

(1) $D^0 \stackrel{def}{=} \mathcal{T} \cup \{\infty\}$

(2) $D^i \stackrel{def}{=} 2^{\mathcal{T}^i \times D^0}$ *where* $i \in \mathbb{N} - \{0\}$

where $\mathcal{T}^i$ *represents* $\underbrace{\mathcal{T} \times \cdots \times \mathcal{T}}_{i}$. $\square$

Let D^* represent the set $\bigcup_{i \in \omega} D^i$. For any set S, we will write $S_\perp$ in place of $S \cup \{\perp\}$. We will sometime abuse notations and write $((u_1, \cdots, u_n), u)$ as $(u_1, \cdot, u_n, u)$.

Definition 2.1.4 *A* timed A-labelled tree *is a labelled tree* (V, E, r, l) *together with a time function* t *which assigns to every vertex* v *of height* n *an element of* $D^n_\perp$ *such that*

(1) *if* $t(v) = \perp$ *and* $\langle v, v' \rangle \in E$, *where* $h(v') = h(v) + 1$, *then* $t(v') = \perp$. *Vertices labelled by* $\perp$ *are called open vertices.*

(2) *if* $\langle v, v' \rangle \in E$, *where* $h(v') = h(v) + 1$, *and vertex* v' *is not an open vertex, then for any* $((u_1, \cdots, u_n), u) \in t(v')$, *there is a* $u' \in D^0$ *such that* $u_n \leq u'$ *and* $((u_1, \cdots, u_{n-1}), u') \in t(v)$.

Elements of $D^*_\perp$ which are associated to vertices of a labelled tree record time constraints over actions labelling paths from the root to the vertices and the maximal delay time of the vertices. For example, if a path between the root and a vertex v are labelled by action $a_1, \cdots, a_n$ and an element of $D^*_\perp$ labelling v contains $((u_1, \cdots, u_n), u)$, then to arrive at v, action a_i may occur at time u_i relative to the previous action, where $i = 1, \cdots, n$. When the machine arrives at

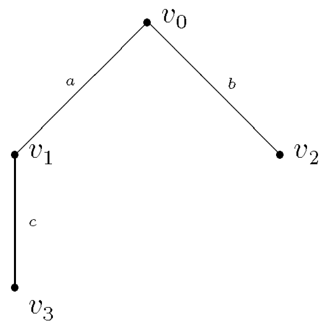


Figure 2–1: Labelled Tree T

v, its maximal delay time is u. We understand a timed labelled tree by the idea that its vertices represent states and its edges represent event occurrences. The vertices are labelled by the elements of $D_\perp^*$ which show time constraints over the actions labelling the paths from the root to the vertices and the maximal delay time of the vertices. The edges are labelled to show how to synchronise with the environment.

Open vertices describe parts of machines which are not fully defined. Time constraints associated to those open vertices cannot be elaborated. As a consequence, the successor vertices of open vertices must also be open vertices which is shown by condition (1). Condition (2) represents that time constraints associated to vertices which are in the same path starting from the root must satisfy a consistency condition. This consistency condition may be understood as: if vertices v and v' are connected by an edge $\langle v, v' \rangle$, where $h(v) + 1 = h(v')$, and $((u_1, \cdots, u_n), u) \in t(v')$; let $a_1, \cdots, a_{n-1}$ label the path from the root r to v and a label the edge $\langle v, v' \rangle$, then $a_1, \cdots, a_{n-1}$ and a may happen at times $u_1, \cdots, u_{n-1}$ and u_n, respectively. When the machine is in the state v, it may delay at least time u_n before action a, as action a occurs at time u_n. So we have $((u_1, \cdots, u_{n-1}), u') \in t(v)$ for some $u' \geq u_n$.

For example, the labelled tree T of Figure 2–1 with a time function t, where $t(v_0) = 15$, $t(v_1) = \{(2, 5)\}$, $t(v_2) = \{(15, 0)\}$ and $t(v_3) = \{((5, 6), 0)\}$, is not a timed labelled tree, as the time information tells us that to arrive at v_1 action a must occur after a delay of time 2 and to arrive at v_3 (note that to arrive at v_3, we must first arrive at v_1), action a must occur after a delay of time 5 and then action c occurs after another delay of time 6, a contradiction. Similarly, the labelled tree T with a time function t', where $t'(v_0) = 3$, $t'(v_1) = \{(10, 5)\}$, $t(v_2) = \{(15, 0)\}$ and $t'(v_3) = \{(10, 6, 0)\}$, is also not a timed labelled tree, as the time information tells us that the machine (we assume that the machine starts from v_0) can delay at most time 3 but action a can occur after a delay of time 10, a contradiction.

We write $v \xrightarrow{a}_T v'$ to represent the edge of a tree T between v and v' which is labelled by a, where $h(v) + 1 = h(v')$. We use $\mathcal{P}(T)$ to represent the set of all paths starting from the root of T. By convention, we use V, E, r, l and t to represent the sets of vertices and edges, the root, the labelling function and the time function of a timed labelled tree T, respectively.

Since an experimenter cannot see the states (only event occurrences can be observed from outside), we identify those timed labelled trees which are the same up to changes of the names of the vertices.

Definition 2.1.5 *Two timed labelled trees $T = (V, E, r, l, t)$ and $T' = (V', E', r', l', t')$ are isomorphic, written as $T = T'$, if there is a bijective map $f : V \longrightarrow V'$ such that*

(1) $f(r) = r'$;

(2) $v \xrightarrow{a}_T v'$ *if and only if* $f(v) \xrightarrow{a}_{T'} f(v')$; *and*

(3) $t = f \circ t'$.

Our timed synchronisation trees are isomorphism classes of timed labelled trees.

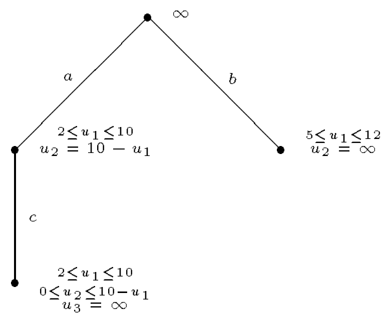


Figure 2–2: Timed Synchronisation Tree T

Definition 2.1.6 *A* timed synchronisation tree $T = [V, E, r, l, t]$ *is an isomorphism class of a timed labelled tree* (V, E, r, l, t).

Let Ts represent the set of all timed synchronisation trees, ranged over by T. For any timed synchronisation tree $[V, E, r, l, t]$, we regard the labelled tree (V, E, r, l) as its underlying tree structure.

The timed synchronisation trees are an interleaving model of real-time systems. For example, the timed synchronisation tree T of Figure 2–2, where $2 \leq u_1 \leq 10$, $0 \leq u_2 \leq 10 - u_1$ and $u_3 = \infty$ stand for the set $\{(u_1, u_2, u_3) \; : \; 2 \leq u_1 \leq 10 \;\&\; 0 \leq u_2 \leq 10 - u_1 \;\&\; u_3 = \infty\}$, represents a machine which can perform either an action a followed by an action c or an action b, but not both. Action a can happen between 2 *seconds* and 10 *seconds* (if we measure time in seconds). If action a happens at time u $(2 \leq u \leq 10)$, then action c can happen within another $10 - u$ *seconds* (inclusive). Note that the time constraints of action c depend on the happening time of action a. This represents the notion of *time dependency* of real-time systems. Action b may happen between 5 *seconds* and 12 *seconds*. After c or b happens, the machine evolves to a state in which no actions can occur, but time may proceed indefinitely. If a and b do not occur after a delay of 15 *seconds*, the machine will evolve to a state in which the machine cannot perform any action but lets time proceed.

2.1.2 A Complete Partial Order over *Ts*

In this section, we define a natural CPO on timed synchronisation trees.

A *partially ordered set* is a set D together with a binary relation $\sqsubseteq_D$ over D which is reflexive, antisymmetric and transitive. We usually write D for the partially ordered set $(D, \sqsubseteq_D)$ when the ordering $\sqsubseteq_D$ is clear from the context. An ω-chain of a partially ordered set D is an infinite sequence of elements, $d_0, \cdots, d_n, \cdots$, of D such that $d_i \sqsubseteq_D d_{i+1}$ where $i \in \omega$. An element $\perp_D$ in D is called a *least element* of D if it satisfies $\perp_D \sqsubseteq_D d$ for every $d \in D$. It follows from the antisymmetry of $\sqsubseteq_D$ that if least elements exist, then they are unique. For any subset M of D, an element a is called an *upper bound* of M if for every $d \in M$ we have $d \sqsubseteq_D a$. We say a is a *least upper bound* (lub) of M if it is an upper bound, and for any upper bound b of M we have $a \sqsubseteq_D b$. Similarly, if least upper bounds exist, then they are unique. We use $\sqcup_D M$ to represent the least upper bound of M in D, when it exists. For convenience, when D is understood from the context, we drop the subscript D.

A partially ordered set or partial order $(D, \sqsubseteq)$ is complete, called a *complete partial order* (CPO), if it contains a least element $\perp$ and any ω-chain $d_0, \cdots, d_n, \cdots$ of D has a least upper bound $\sqcup_{i \in \omega} d_i$ (we write $\sqcup_{i \in \omega} d_i$ for $\sqcup \{d_i : i \in \omega\}$).

Given two CPOs D and E, a function $f : D \longrightarrow E$ is *monotonic* if for any $d, d' \in D$, $d \sqsubseteq d'$ implies $f(d) \sqsubseteq f(d')$. If f is monotonic and for any ω-chain $d_0, \cdots, d_n, \cdots$ we have $f(\sqcup_{i \in \omega} d_i) = \sqcup_{i \in \omega} f(d_i)$, then we say f is *continuous*. We write $[D \longrightarrow E]$ for the set of all *continuous functions* from D to E. For any $f, g \in [D \longrightarrow E]$, we say $f \sqsubseteq g$ if and only if for every $d \in D$, $f(d) \sqsubseteq g(d)$. This ordering is usually known as the *induced pointwise ordering*. With this ordering, $[D \longrightarrow E]$ itself is a CPO.

The following results are standard classical results of domain theory. For more details, the reader may refer to a standard reference [Hen88a].

Lemma 2.1.7 *A function $f : D \longrightarrow D'$ is continuous if and only if it is monotonic and for every ω-chain $d_0, \cdots, d_n, \cdots$ of D*

$$f(\sqcup_{i\in\omega} d_i) \sqsubseteq \sqcup_{i\in\omega} f(d_i)$$

For any CPOs D_1 and D_2, we define an ordering $\sqsubseteq$ on the Cartesian product of D_1 and D_2 as follows: for every $\langle d_1, d_2\rangle$ and $\langle d'_1, d'_2\rangle$ of $D_1 \times D_2$, $\langle d_1, d_2\rangle \sqsubseteq \langle d'_1, d'_2\rangle$ if both $d_1 \sqsubseteq d'_1$ and $d_2 \sqsubseteq d'_2$ hold. With this ordering, $D_1 \times D_2$ is a CPO.

A function $f : D_1 \times D_2 \longrightarrow D$ is *left-continuous* if for every element d of D_2 and every ω-chain $\langle d_0, d\rangle, \cdots, \langle d_n, d\rangle, \cdots$ of $D_1 \times D_2$

$$f(\sqcup_{i\in\omega}\langle d_i, d\rangle) = \sqcup_{i\in\omega} f(\langle d_i, d\rangle)$$

Similarly we can define *right-continuous* function.

Lemma 2.1.8 *A function $f : D_1 \times D_2 \longrightarrow D$ is continuous if and only if it is both left-continuous and right-continuous.*

Definition 2.1.9 *Let $f \in [D \longrightarrow D]$. An element d of D is called a* fixpoint *of f if $d = f(d)$. It is a* least fixpoint *of f if it is a fixpoint of f and for any other fixpoint d' of f, we have $d \sqsubseteq d'$.*

Obviously if a function $f \in [D \longrightarrow D]$ has least fixpoints, then the least fixpoints are unique. Moreover continuous functions always have least fixpoints.

Theorem 2.1.10 *Every $f \in [D \longrightarrow D]$ has a least fixpoint.*

Now we consider a natural CPO over timed synchronisation trees.

Let $\sqsubseteq$ be a partial order over $D^*_\perp$ satisfying that for any S, $S' \in D^*_\perp$, $S \sqsubseteq S'$ implies $S = \perp$ or $S = S'$. A partial order $\sqsubseteq$ over the set $[V \longrightarrow D^*_\perp]$ of continuous time functions is the induced pointwise ordering.

Definition 2.1.11 (A Partial Order over Ts)
For any $T, T' \in Ts$, we say $T \sqsubseteq T'$ if for any $(V, E, r, l, t) \in T$, there is a $(V', E', r', l', t') \in T'$ such that

$$\begin{array}{llll} (1) & V \subseteq V' & (4) & l = l'|_E \\ (2) & E \subseteq E' & (5) & t \sqsubseteq t'|_V \\ (3) & r = r' & & \end{array}$$

Note that for any function f, we use $f|_S$ to represent the restriction of f to the set S.

Lemma 2.1.12 $\sqsubseteq$ *is a partial order over Ts.*

Proof: It is easy to show that $\sqsubseteq$ over Ts is reflexive, antisymmetric and transitive. □

Theorem 2.1.13 $(Ts, \sqsubseteq)$ *is a* CPO.

Proof: The timed synchronisation tree $[\{v\}, \emptyset, v, \emptyset, \{(v, \perp)\}]$ is the least element of Ts. For any ω-chain $T_0, \cdots, T_n, \cdots$, let $(V_i, E_i, r_i, l_i, t_i) \in T_i$ such that $V_i \subseteq V_{i+1}$, $E_i \subseteq E_{i+1}$ $r_i = r_{i+1}$, $l_i = l_{i+1}|_{E_i}$ and $t_i \sqsubseteq t_{i+1}|_{V_i}$, where $i \in \omega$. We may show that

$$T = [\cup_{i\in\omega} V_i, \cup_{i\in\omega} E_i, r_0, \sqcup_{i\in\omega} l_i, \sqcup_{i\in\omega} t_i]$$

is the least upper bound of the ω-chain.

We first need to show that T is a timed synchronisation tree.

(1) It is easy to check that

$$(\cup_{i\in\omega} V_i, \cup_{i\in\omega} E_i, r_0, \sqcup_{i\in\omega} l_i)$$

is a labelled tree.

(2) For any $v \in \cup_{i\in\omega} V_i$, let n be the height of v. If $\sqcup_{i\in\omega} t_i(v) \neq \perp$, then there is an m such that for all $i \geq m$, we have $t_i(v) = t_m(v)$, where $t_m(v) \in D^n$. Hence $\sqcup_{i\in\omega} t_i(v) \in D^n_\perp$.

(3) Suppose $\sqcup_{i\in\omega}t_i(v)=\perp$ and $\langle v,v'\rangle \in \sqcup_{i\in\omega}E_i$, where $h(v)+1=h(v')$. Then for every $i\in\omega$, $v\in V_i$ implies $t_i(v)=\perp$. Also there is an n such that for any $i\geq n$, $\langle v,v'\rangle\in E_i$. Hence for any $i\geq n$, we have $t_i(v')=\perp$. Therefore $\sqcup_{i\in\omega}t_i(v')=\perp$.

(4) Let $\langle v,v'\rangle \in \cup_{i\in\omega}E_i$ where $h(v)+1=h(v')$ and v' is not an open vertex. Suppose $(u_1,\cdots,u_n,u)\in\sqcup_{i\in\omega}t_i(v')$. There is an n such that for any $i\geq n$ we have $(u_1,\cdots,u_n,u)\in t_i(v')$. So for any $i\geq n$, there is a u' such that $u'\geq u_n$ and $(u_1,\cdots,u_{n-1},u')\in t_i(v)$. Therefore $(u_1,\cdots,u_{n-1},u')\in\sqcup_{i\in\omega}t_i(v)$.

Now we only need to show that T is the least upper bound of the ω-chain, which directly follows from the definition of $\sqsubseteq$ over Ts. □

2.2 Constructions

In this section, we consider some constructions over Ts.

2.2.1 Empty Constant

Let Ω be the timed synchronisation tree $[\{v\},\emptyset,v,\emptyset,\{(v,\perp)\}]$, which is the least element of Ts. We call it the empty constant. The empty element may be described diagrammatically as:

$$\bullet\ \perp$$

2.2.2 Prefix

To capture the notion of time dependency, we define a time synchronisation tree with a parameter as an isomorphism class of a labelled tree T with a time function which assigns to every vertex v of depth i a function $f_v : \mathcal{T} \longrightarrow D^i_\perp$ such that if $v\rightarrow_T v'$ then for any $u\in\mathcal{T}$, $f_v(u)$ and $f_{v'}(u)$ satisfy conditions (1) and (2) of

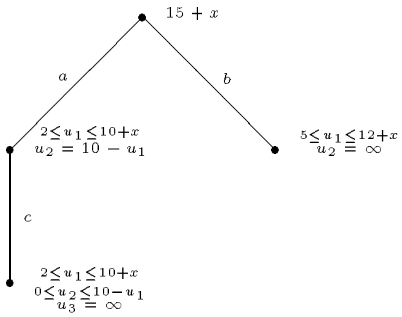


Figure 2–3: Timed Synchronisation Tree T with Parameter x

Definition 2.1.4. A timed synchronisation tree with a parameter x can be regarded as a family of timed synchronisation trees which are indexed by the parameter x. All timed synchronisation trees in the family have the same underlying tree structure, i.e. they represent the same labelled tree if we ignore their time function, and only time constraints associated to every vertex may contain the parameter x. For example, Figure 2–3 represents a timed synchronisation tree with a parameter x. If we take x to be 0, then we get the timed synchronisation tree of Figure 2–2. We use $T\{u/x\}$ to represent the timed synchronisation tree resulted by taking the parameter x to be u in T.

We can generalise the notion of timed synchronisation trees with a parameter to a notion of timed synchronisation trees with a sequence of parameters. As a special case, a timed synchronisation tree is a timed synchronisation tree with an empty sequence of parameters. Clearly a timed synchronisation tree with n parameters can also be regarded as a timed synchronisation tree with $n+1$ parameters.

Definition 2.2.1 *For any $a \in A$, $S \subseteq \mathcal{T}$ which has a least upper bound in $\mathcal{T}$, and timed synchronisation tree T, possibly with parameters, the prefix $a(S).(\lambda x.T)$ represents a new timed synchronisation tree T', possibly with parameters, such that*

- $V' = \{a\} \uplus V$

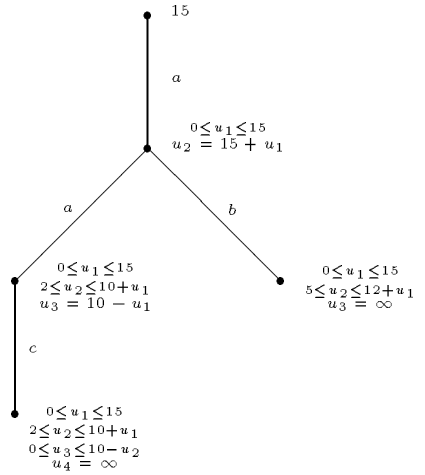


Figure 2–4: Timed Synchronisation Tree: $a([0, 15])(\lambda x.T)$

- $E' = \{\langle (v,1),(v',1)\rangle \;:\; \langle v, v'\rangle \in E\} \uplus \{\langle (a,0),(r,1)\rangle\}$
- $r' = (a, 0)$
- $l' = \lambda(e,i).\begin{cases} a & i = 1 \& e = \langle (a,0),(r,1)\rangle \\ l(\langle v, v'\rangle) & i = 0 \& e = \langle (v,1),(v',1)\rangle \end{cases}$
- $t' = \lambda(v,i).\begin{cases} Sup(S) & (v,i) = r' \\ \bot & t_u(v) = \bot \\ & \quad where\ t_u(v) = (t\{u/x\})(v)\ \&\ u \in S \\ \{\langle u\rangle s \;:\; u \in S \wedge s \in t_u(v)\} & otherwise \end{cases}$

where $Sup(S)$ represents the least upper bound of S. By convention, $Sup(\emptyset) = 0$.

Note that we use $\lambda x.T$ in the prefix $a(S)(\lambda x.T)$ to emphasize the parameter x of T which will be instantiated next.

Figure 2–4 shows the result of prefixing the timed synchronisation tree T, with a parameter x, of Figure 2–3 with an action a and an interval $[0, 15]$.

We still use Ts to represent the set of all timed synchronisation trees, possible with parameters. Let $\sqsubseteq$ over timed synchronisation trees with parameters be the induced pointwise ordering, i.e. for any timed synchronisation trees T and T', where T and T' contain at most parameters $x_1, \cdots, x_n$, we say $T \sqsubseteq T'$ when $T\{u_1/x_1, \cdots, u_n/x_n\} \sqsubseteq T'\{u_1/x_1, \cdots, u_n/x_n\}$ for any $u_1, \cdots, u_n \in \mathcal{T}$.

Theorem 2.2.2 *$a(S) : Ts \longrightarrow Ts$ is continuous with respect to $\sqsubseteq$.*

Proof: Since the partial order $\sqsubseteq$ over timed synchronisation trees with parameters is the induced pointwise ordering, we only need to consider the case where T contains at most one parameter x. We use $[a(S).V, a(S).E, a(S).r, a(S).l, a(S).(\lambda x.t)]$ to represent the timed synchronisation tree $a(S).(\lambda x.T)$. It is easy to check that $a(S)$ is monotonic. Now we only need to show that for any ω-chain $T_0, \cdots, T_n, \cdots$, we have

$$a(S).(\cup_{i\in\omega} V_i) \subseteq \cup_{i\in\omega} a(S).V_i$$
$$a(S).(\cup_{i\in\omega} E_i) \subseteq \cup_{i\in\omega} a(S).E_i$$
$$\sqcup_{i\in\omega} a(S).l_i = a(S).(\sqcup_{i\in\omega} l_i)|_{a(S).(\cup_{i\in\omega} E_i)}$$

and

$$a(S).(\sqcup_{i\in\omega} \lambda x.t_i) \sqsubseteq \sqcup_{i\in\omega} a(S).(\lambda x.t_i)|_{a(S).(\cup_{i\in\omega} V_i)}$$

We only consider the last one, as all others are similar.

For any $v \in \cup_{i\in\omega} V_i$, if $(a(S).(\sqcup_{i\in\omega}\lambda x.t_i))(v) = \perp$ then $(a(S).(\sqcup_{i\in\omega}\lambda x.t_i))(v) \sqsubseteq (\sqcup_{i\in\omega} a(S).(\lambda x.t_i))(v)$. Suppose $(a(S).(\sqcup_{i\in\omega}\lambda x.t_i))(v) \neq \perp$. If v is the root, then $(a(S).(\sqcup_{i\in\omega}\lambda x.t_i))(v) = Sup(S) = (\sqcup_{i\in\omega} a(S).(\lambda x.t_i))(v)$. Now we also suppose v is not the root. Let $s \in (a(S).(\sqcup_{i\in\omega}\lambda x.t_i))(v)$, then there is a s' and $u \in S$ such that $s = \langle u\rangle s'$ and $s' \in (\sqcup_{i\in\omega}(t_i\{u/x\}))(v)$. Hence there is a k such that for any $i \geq k$, $s' \in t_i\{u/x\}(v)$ and $s \in (a(S).(\lambda x.t_i))(v)$. So $s \in (\sqcup_{i\in\omega} a(S).(\lambda x.t_i))(v)$. □

2.2.3 Summation

The sum of two timed synchronisation trees T and T' represents a new timed synchronisation tree resulted by putting T and T' together at their roots. Figure 2–5 diagrammatically describes the summation operation.

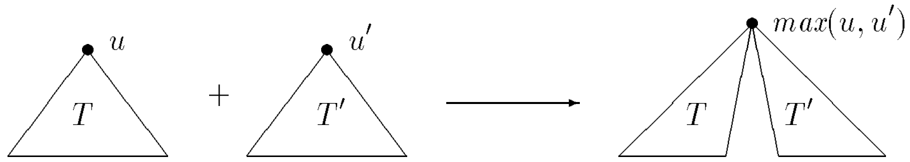


Figure 2–5: Summation Operation

Definition 2.2.3 *For any timed synchronisation trees T_1 and T_2, possibly with parameters, $T_1 + T_2$ represents a new timed synchronisation tree T, where*

- $V = (V_1 - \{r_1\}) \uplus (V_2 - \{r_2\}) \uplus \{(r_1, r_2)\}$

- $$\begin{aligned} E = & \{\langle (v,0),(v',0)\rangle \ : v \neq r_1 \& \langle v, v'\rangle \in E_1\} \\ & \uplus \{\langle (v,1),(v',1)\rangle \ : v \neq r_2 \& \langle v, v'\rangle \in E_2\} \\ & \uplus \{\langle ((r_1,r_2),2),(v,i)\rangle \ : \ \langle r_{i+1}, v\rangle \in E_1 \vee \langle r_{i+1}, v\rangle \in E_2\} \end{aligned}$$

- $r = ((r_1, r_2), 2)$

- $$l = \lambda(e,i).\begin{cases} l_1(\langle v, v'\rangle) & i = 0 \& e = \langle (v,0),(v',0)\rangle \\ l_2(\langle v, v'\rangle) & i = 1 \& e = \langle (v,1),(v',1)\rangle \\ l_{j+1}(\langle r_{j+1}, v\rangle) & i = 2 \& e = (r,(v,j)) \& v \in V_{j+1} \& j = 0,1 \end{cases}$$

- $$t = \lambda(v,i).\begin{cases} max(t_1(r_1), t_2(r_2)) & (v,i) = r \\ t_i(v) & v \in V_i \ \& \ i = 0,1 \end{cases}$$

where, by convention, $max(u, \perp) = u$.

Remark If we use *min* to replace *max* in the definition of summation, we get Moller and Tofts' strong summation [MT90]. □

Clearly the summation operation is well defined. Moreover we have the following theorem.

Theorem 2.2.4 $+ : Ts \times Ts \longrightarrow Ts$ *is continuous with respect to* $\sqsubseteq$.

Proof: We only need to show that $+$ is both left and right continuous. By symmetry, it is enough to check that $+$ is left continuous.

Clearly $+$ is left monotonic, i.e. for any T, T' and T'' of Ts, $T \sqsubseteq T'$ implies $T + T'' \sqsubseteq T' + T''$. For every ω-chain $T_1, \cdots, T_n, \cdots$ and a T of Ts, we need to show that for every $(V', E', r', l', t') \in \sqcup_{i\in\omega} T_i + T$, there is a $(V'', E'', r'', l'', t'') \in \sqcup_{i\in\omega}(T_i + T)$ such that $V' \subseteq V''$, $E' \subseteq E''$, $r' = r''$, $l' = l''|_{E'}$ and $t' \sqsubseteq t''|_{V'}$. Suppose $(V', E', r', l', t') \in \sqcup_{i\in\omega} T_i + T$. Let $v \in V'$, then $v \in V$ or there is a k such that for all $i \geq k$ we have $v \in V_i$. So there is a V'' such that $v \in V''$ and thus $V' \subseteq V''$, where V'' is the set of vertices of a timed labelled tree of $\sqcup_{i\in\omega}(T_i + T)$. In the same way we have E'', r'', l'' and t'', where $E' \subseteq E''$, $r' = r''$, $l' = l''|_{E'}$ and $t' \sqsubseteq t''|_{V'}$, such that $(V'', E'', r'', l'', t'')$ is a timed labelled tree of $\sqcup_{i\in\omega}(T_i + T)$. □

2.2.4 Restriction

The restriction of a specific action $\lambda \in A$, where $\lambda \neq \tau$, on a timed synchronisation tree T represents a new timed synchronisation tree resulted by pruning away all edges labelled by λ and $\overline{\lambda}$ together with their successors.

Definition 2.2.5 *Let* T *be a timed synchronisation tree, possibly with parameters, and* $\lambda \in A$*, where* $\lambda \neq \tau$*,* $T\backslash\lambda$ *represents a new timed synchronisation tree* T'*, where*

- $V' = \{v\ :\ there\ is\ a\ path\ e_1, \cdots, e_n\ from\ the\ root\ r\ to\ v\ such\ that\ for\ any\ i = 1, \cdots, n,\ \ l(e_i) \neq \lambda\ and\ l(e_i) \neq \bar{\lambda}\}$

- $E' = E \cap V' \times V'$
- $r' = r$
- $l' = l|_{E'}$
- $t' = t|_{V'}$

Clearly the restriction operator is well defined. Moreover we have the following theorem.

Theorem 2.2.6 $\backslash\lambda : Ts \longrightarrow Ts$ *is continuous with respect to* $\sqsubseteq$.

Proof: Similar to that of Theorem 2.2.2. $\square$

Notation $\delta \stackrel{def}{=} (a(\{0\})(\lambda x.\Omega))\backslash a$

δ may be described diagrammatically as:

$$\bullet\ 0$$

2.2.5 Parallel

We can define the parallel operation on timed synchronisation trees. Figure 2–6 and Figure 2–7 show the result of the parallel operation of two timed synchronisation trees.

Definition 2.2.7 *For any* T_0, $T_1 \in Ts$, *the interleaving* $\mathcal{I}(T_0, T_1)$ *of their vertices is defined as*

$$\mathcal{I}(T_0, T_1) = \{v\ :\ i = 0, 1 \& \pi_i(v) \in \mathcal{P}(T_i)\ \&\ v \in C^*\}$$

Figure 2–6: Timed Synchronisation Trees T_1 and T_2

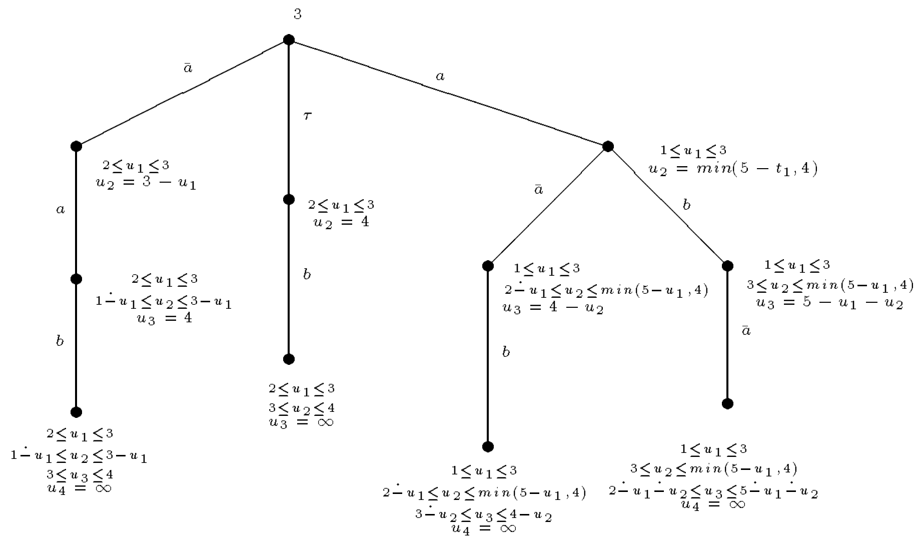


Figure 2–7: Parallel Composition of T_1 and T_2

where $C = E_0 \uplus E_1 \uplus \{(e, e') : e \in E_0 \;\&\; e' \in E_1 \;\&\; \overline{l_0(e)} = l_1(e') \neq \tau\}$, $\mathcal{P}(T)$ *is the set of all path starting from the root in* T *and we first define* $\pi_i : C \longrightarrow E_i$, $i = 0, 1$, *as*

$$\pi_i = \lambda(x, j). \begin{cases} \langle\rangle & i \neq j \& j = 0, 1 \\ \langle x \rangle & i = j \\ \langle x_i \rangle & j = 2 \& x = (x_0, x_1) \end{cases}$$

then we generalize $\pi_i : C^* \longrightarrow E_i^*$ *as follows*

$$\pi_i(\langle e_1 \cdots e_n \rangle) = \pi_i(e_1) \cdots \pi_i(e_n)$$

Clearly, for any timed synchronisation trees T_1 and T_2, the interleaving $\mathcal{I}(T_1, T_2)$ of their vertices contains the null sequence $\langle\rangle$ and is closed under the initial subsequence relation.

Definition 2.2.8 *For any* T_0, $T_1 \in Ts$, *their parallel composition* $T_0 \,|\, T_1$ *represents a new timed synchronisation tree* T, *where*

- $V = \mathcal{I}(T_0, T_1)$
- $E = \{\langle v, v' \rangle : v,\, v' \in V \;\&\; \exists \alpha \in C. v\langle \alpha \rangle = v'\}$
- $r = \langle\rangle$
- $l = \lambda e. \begin{cases} l_i(\alpha) & e = \langle v, v' \rangle \;\&\; v\langle \alpha \rangle = v' \;\&\; \alpha \in E_i \;\&\; i = 0, 1 \\ \tau & e = \langle v, v' \rangle \;\&\; v\langle \alpha \rangle = v' \;\&\; \alpha \in E_0 \times E_1 \end{cases}$
- $t = \lambda v. S$

  where if $t_i(\pi_i(v)) = \perp$ $(i = 0, 1)$ *then* $S = \perp$;

  otherwise, let $v = \langle \alpha_1, \cdots, \alpha_n \rangle$, *then* S *is the least set which satisfies*

  (1) $(u_1, \cdots, u_n, u) \in S$ *implies that*

$$(\sum_{1 \leq j \leq k_1} u_j, \cdots, \sum_{k_{l-1} < j \leq k_l} u_j, u') \in t_0(\pi_0(v))$$

$$(\sum_{1\le j\le k'_1} u_j, \cdots, \sum_{k'_{m-1}<j\le k'_m} u_j, u'') \in t_1(\pi_1(v))$$

and

$$u = min(u' - \sum_{k_l<j\le n} u_j, u'' - \sum_{k'_m<j\le n} u_j)$$

for some $u', u'' \in D^0$, *where* $\{k_1, \cdots, k_l\}$ *and* $\{k'_1, \cdots, k'_m\}$ $(k_1 \le \cdots \le k_l$ *and* $k'_1 \le \cdots \le k'_m)$ *satisfy that* $\forall j \in \{k_1, \cdots, k_l\}.\ \pi_0(\alpha_j) \ne \langle\rangle\ \&\ \forall i \in \{1, \cdots, n\} - \{k_1, \cdots, k_l\}.\pi_0(\alpha_i) = \langle\rangle$ *and* $\forall j \in \{k'_1, \cdots, k'_m\}.\pi_1(\alpha_j) \ne \langle\rangle \& \forall i \in \{1, \cdots, n\} - \{k'_1, \cdots, k'_m\}.\ \pi_1(\alpha_i) = \langle\rangle$; *and*

(2) $(u'_1, \cdots, u'_l, u') \in t_0(\pi_0(v))$ *and* $(u''_1, \cdots, u''_m, u'') \in t_1(\pi_1(v))$ *implies that there are* $(u_1, \cdots, u_n, u) \in t(v)$, *where* $n = l + m$, *such that*

$$u'_1 = \sum_{1\le j\le k_1} u_j, \cdots, u'_l = \sum_{k_{l-1}<j\le k_l} u_j$$
$$u''_1 = \sum_{1\le j\le k'_1} u_j, \cdots, u''_l = \sum_{k'_{m-1}<j\le k'_m} u_j$$

and

$$u = min(u' - \sum_{k_l<j\le n} u_j, u'' - \sum_{k'_m<j\le n} u_j)$$

where $k_1, \cdots, k_l, k'_1, \cdots, k'_m$ *are defined as in* (1).

It is easy to see that the parallel operation is well defined. Moreover we have the following theorem.

Theorem 2.2.9 $|: Ts \times Ts \longrightarrow Ts$ *is continuous with respect to* $\sqsubseteq$.

Proof: Analogous to Theorem 2.2.4. □

To present an expansion theorem which show the interleaving characterization of parallel composition of timed synchronisation trees, we first introduce two notations.

Notation For every $u \in \mathcal{T}$ and $[V, E, r, l, t] \in Ts$, the time prefix $(u)[V, E, r, l, t]$ represents a timed synchronisation tree $[V, E, r, l, t']$ where

$$t' = \lambda v. \begin{cases} \perp & t(v) = \perp \\ t(r) + u & v = r \,\&\, t(r) \in \mathcal{T} \\ \{(u_1 + u, \cdots, u_{n+1}, u') : (u_1, \cdots, u_{n+1}, u') \in t(v)\} & otherwise \end{cases}$$

□

Intuitively, the time prefix $(u)[V, E, r, l, t]$ represents the machine which, after a delay of time u, has the same behaviour as the machine represented by $[V, E, r, l, t]$.

Notation For every $u \in \mathcal{T}$ and $[V, E, r, l, t] \in Ts$, the time shift $u \gg [V, E, r, l, t]$ represents a timed synchronisation tree $[V, E, r, l, t']$ where

$$t' = \lambda v. \begin{cases} \perp & t(v) = \perp \\ u' \dot{-} u & v = r \,\&\, t(r) = u' \\ \{(u_1 - u, \cdots, u_{n+1}, u') : (u_1, \cdots, u_{n+1}, u') \in t(v) \,\&\, u_1 \geq u\} & \\ & otherwise \end{cases}$$

□

Intuitively, the time shift $u >> [V, E, r, l, t]$ represents the result of machine $[V, E, r, l, t]$ after a delay of time u without any action.

Theorem 2.2.10 (Expansion Theorem)

Let

$$T = \sum_{i \in I} a_i(S_i)(\lambda x_i.T_i) + (u)\delta$$

and

$$T' = \sum_{j \in J} b_j(S'_j)(\lambda y_j.T'_j) + (u')\delta$$

where $Sup(S_i) \leq u$, $Sup(S'_j) \leq u'$ *for any* $i \in I$ *and* $j \in J$, *then*

$T \mid T' =$

$$\sum_{i \in I} a_i(S_i \cap \{w : w \leq u'\})(\lambda z_i.(T_i\{z_i/x_i\} \mid z_i >> T'))$$

$$+$$

$$\sum_{j \in J} b_j(S'_j \cap \{w : w \leq u\})(\lambda z'_j.(z'_j >> T \,|\, T'_j\{z'_j/y_j\}))$$

$$+$$

$$\sum_{\substack{i \in I \\ j \in J}} \tau(S_i \cap S'_j)(\lambda z''_{ij}.(T_i\{z''_{ij}/x_i\} \,|\, T'_j\{z''_{ij}/y_j\}))$$

$$+$$

$$min(u, u')\delta$$

where for any $i \in I$ and $j \in J$, z_i, z'_j and z''_{ij} are fresh parameters.

Proof: Let $T_l = [V_l, E_l, r_l, l_l, t_l]$ and $T_r = [V_r, E_r, r_r, l_r, t_r]$ represent the left hand side and right hand side of the expansion theorem, respectively. Clearly $v \in V_l \Leftrightarrow \pi_0(v) \in \mathcal{P}(T) \;\&\; \pi_1(v) \in \mathcal{P}(T')$

$$\Leftrightarrow v = \begin{cases} \langle\rangle & or \\ \langle(r, r_i)\rangle v' & i \in I \;\&\; \pi_0(v') \in \mathcal{P}(T_i) \;\&\; \pi_1(v') \in \mathcal{P}(T') \\ & \qquad \& \; l(r, r_i) = a_i, \; or \\ \langle(r', r'_j)\rangle v' & j \in J \;\&\; \pi_0(v') \in \mathcal{P}(T) \;\&\; \pi_1(v') \in \mathcal{P}(T'_j) \\ & \qquad \& \; l(r', r'_j) = b_j, \; or \\ \langle((r, r_i), (r', r'_j))\rangle v' & i \in I \;\&\; j \in J \;\&\; \pi_0(v') \in \mathcal{P}(T) \;\&\; \pi_1(v') \in \mathcal{P}(T'_j) \\ & \qquad \& \; l(r, r_i) = a_i = \bar{b}_j = \overline{l(r', r'_j)} \neq \tau. \end{cases}$$

This gives us a bijection $f : V_l \longrightarrow V_r$ which satisfies $f(r_l) = r_r$ and $v \xrightarrow{a}_{T_l} v'$ implies $f(v) \xrightarrow{a}_{T_r} f(v')$. Now we only need to show that for every $v \in V_l$, $t_l(v) = t_r(f(v))$ which follows from Definition 2.2.8. □

2.2.6 Relabelling

By relabelling a tree T with a relabelling function $R : A \longrightarrow A$ we get a new tree which is resulted from T by changing the labels of the edges of T using the relabelling function R.

Definition 2.2.11 *Let $[V, E, r, l, t] \in Ts$ and $R: A \longrightarrow A$ be a relabelling function, $[V, E, r, l, t][R]$ represents a new tree $[V, E, r, l \circ R, t]$.*

It is easy to check that relabelling operation is well defined. Moreover we have the following theorem.

Theorem 2.2.12 *$[R] : Ts \longrightarrow Ts$ is continuous with respect to $\sqsubseteq$.*

Proof: Analogous to Theorem 2.2.2. □

2.3 An Equivalence over *Ts*

Trees are concrete structures and contain detailed information. They distinguish too many things. For example, a tree T in general is not equal to a tree $T + T$, although it is hard to imagine any programming contexts in which their behaviour could be distinguished. In this section, we define an equivalence on Ts based on a notion of behaviour of timed synchronisation trees. The equivalence identifies those trees which have the same behaviour.

Definition 2.3.1 *Given T, $T' \in Ts$, we write $T \xrightarrow{a}_u T'$, where $a \in A$ and $u \in \mathcal{T}$, whenever $r \xrightarrow{a}_T r'$, $u \in \{u' : (u', u'') \in t(r')\}$ and*

(1) $V' = \{v : \textit{there is a path from } r' \textit{ to } v \textit{ in graph } (V, E - \{\langle r, r' \rangle\})\}$

(2) $E' = E \cap V' \times V'$

(3) $l' = l|_{E'}$

(4) $t' = \lambda v. \begin{cases} \perp & t(v) = \perp \\ \{(u_2, \cdots, u_{n+1}, u') : (u, u_2, \cdots, u_{n+1}, u') \in t(v)\} & \textit{otherwise} \end{cases}$

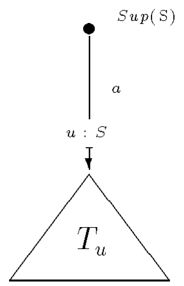


Figure 2–8: a timed synchronisation tree: T

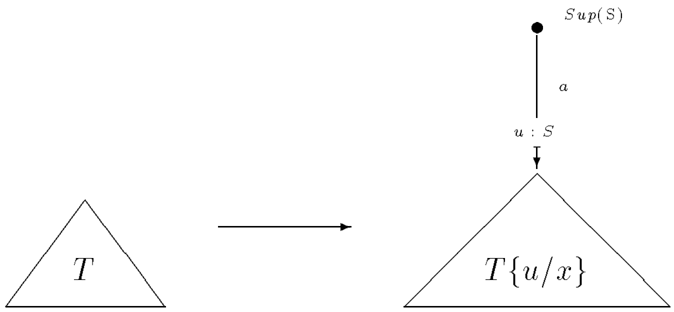


Figure 2–9: prefix operation

We can understand $T \xrightarrow{a}_u T'$ as that the machine represented by T performs an action a at time u and after doing so it evolves to a state represented by T'.

For convenience, we use Figure 2–8 to represent a timed synchronisation tree T, where $T \xrightarrow{a}_u T_u$ for every $u \in S$. As a result, the procedure of prefixing a timed synchronisation tree, possibly with a parameter x, by a specific action $a \in A$ and an interval S of $\mathcal{T}$, can be described diagrammatically as in Figure 2–9.

We say a timed synchronisation tree is closed if it contains no open vertices. The presence of an open vertex in a timed synchronisation tree indicates that the tree is not fully defined.

Lemma 2.3.2

(1) *If* $a(S)(\lambda x.T)$ *is a closed timed synchronisation tree and* $u \in S$, *then*

$$a(S)(\lambda x.T) \xrightarrow{a}_u T\{u/x\}$$

(2) *If* $T \xrightarrow{a}_u T''$, *then* $T + T' \xrightarrow{a}_u T''$ *and* $T' + T \xrightarrow{a}_u T''$.

Proof: Straightforward. □

Definition 2.3.3 *A binary relation* $\mathcal{R}$ *over closed timed synchronisation trees is a tree bisimulation if* $(T_1, T_2) \in \mathcal{R}$ *implies that for any* $a \in A$ *and* $u \in \mathcal{T}$,

(1) *if* $T_1 \xrightarrow{a}_u T_1'$, *then* $T_2 \xrightarrow{a}_u T_2'$ *and* $(T_1', T_2') \in \mathcal{R}$ *for some* T_2';

(2) *if* $T_2 \xrightarrow{a}_u T_2'$, *then* $T_1 \xrightarrow{a}_u T_1'$ *and* $(T_1', T_2') \in \mathcal{R}$ *for some* T_1'; *and*

(3) $t_1(r_1) = t_2(r_2)$.

We say two closed trees T and T' are equivalent, written as $T \simeq T'$, if there is a tree bisimulation $\mathcal{R}$ such that $(T, T') \in \mathcal{R}$.

Definition 2.3.4 $\simeq \; = \; \bigcup\{\mathcal{R} : \mathcal{R}$ *is a tree bisimulation*$\}$

Proposition 2.3.5 $\simeq$ *is the largest tree bisimulation. Moreover it is an equivalence relation.*

Proof: The result follows from the facts that the union, composition and inverse of tree bisimulations are still tree bisimulations. Also the identity relation over closed timed synchronisation trees is a tree bisimulation. □

Since $(Ts, \sqsubseteq)$ is a CPO and all operations are continuous with respect to $\sqsubseteq$, we may define infinite timed synchronisation trees along the standard line. For example, the infinite timed synchronisation tree $\mu X.T$ represents the limit of the ω-chain $T^{(0)}, \cdots, T^{(n+1)}, \cdots$ where

$$T^{(0)} = \Omega \qquad \textit{and} \qquad T^{(n+1)} = T\{T^{(n)}/X\}$$

Lemma 2.3.6 *Given timed synchronisation trees T_1 and T_2, if $T_1 \sqsubseteq T_2$ and $T_1 \xrightarrow{a}_u T_1'$, then $T_2 \xrightarrow{a}_u T_2'$ and $T_1' \sqsubseteq T_2'$ for some tree T_2'.*

Proof: The result follows from Definition 2.3.1. □

With Lemma 2.3.6, we can now define behaviours of an infinite timed synchronisation tree.

Definition 2.3.7 *For an infinite tree $\mu X.T$, we say $\mu X.T \xrightarrow{a}_u T'$ if there is a $n \in \omega$ such that for any $i \geq n$*

$$T^{(i)} \xrightarrow{a}_u T_i' \qquad \textit{and} \qquad T' = \sqcup_{\substack{i\in\omega \\ i\geq n}} T_i'$$

where $T^{(0)} = \Omega$ and $T^{(i+1)} = T\{T^{(i)}/X\}$ for every $i \in \omega$.

Proposition 2.3.8 $\mu X.T \xrightarrow{a}_u T'$ *if and only if* $T\{\mu X.T/X\} \xrightarrow{a}_u T'$

Proof: Clearly $T\{\mu X.T/X\}$ represents the limit of the ω-chain $T^1, \cdots, T^n, \cdots$, where

$$T^1 = T\{\Omega/X\} \qquad \textit{and} \qquad T^{n+1} = T\{T^n/X\}$$

The result follows from the definition of the behaviour of infinite timed synchronisation trees. □

2.4 Conclusion

In this chapter, we presented a simple interleaving model, Timed Synchronisation Trees, for parallel computation in real-time systems. To represent machines which are only partially defined, we follow Hennessy [Hen85] and introduce a notion of open vertices. The presence of open vertices allows us to consider the model, timed synchronisation trees, as a CPO. All operations on timed synchronisation trees are continuous with respect to the CPO. This allows us to define infinite timed synchronisation trees along the standard lines.

Timed synchronisation trees may be used to give denotational semantics for a wide range of timed calculi, including Baeten and Bergstra's *Real Time Process Algebra* and Moller and Tofts' TCCS. In chapter 3, we will give a denotational semantics for Timed CCS based on timed synchronisation trees.

Chapter 3

Timed CCS: A Timed Calculus of Communicating Systems

In this chapter, a formal description of Timed CCS (its syntax and semantics) is presented, together with some informal interpretations and remarks on design decisions.

3.1 Syntax

Timed CCS is an extension of Milner's CCS [Mil80,Mil89a] with time. To define Timed CCS, we presuppose a countable set of names Λ, ranged over by a, b, of atomic actions not containing τ and ϵ. Let $Act = \Lambda \cup \{\tau\}$, ranged over by α, β. As in CCS, Λ can be partitioned into Γ, the set of names, and $\overline{\Gamma} = \{\bar{a} \mid a \in \Gamma\}$, the set of co-names, with the provision that $\bar{\bar{a}} = a$. Actions a and $\bar{a}$ are called complementary actions which form the basis of communications in our language, analogous to CCS. We also presuppose an infinite set V_t of time variables, ranged over by t, s, r. Let the time domain be $(\mathcal{T} \cup \{\infty\}, \leq)$, where $\mathcal{T}$ contains the least element 0 to represent the starting time and $\leq$ is a linear order over $\mathcal{T}$. Note that we make no assumption about the underlying nature of time, allowing $\mathcal{T}$, for example, to be $\mathbb{N}$ (the set of natural numbers), $\mathbb{Q}^{\geq 0}$ (the set of non-negative rationals) or $\mathbb{R}^{\geq 0}$ (the set of non-negative reals). We introduce ∞ to represent infinite time, where $\infty \notin \mathcal{T}$. Our time expressions, ranged over by e, f, g, are defined as follows:

Definition 3.1.1

1 *For any $u \in \mathcal{T}$ and $t \in V_t$, u and t are time expressions.*

2 *For every $u \in \mathcal{T}$ and time expression e, $u \times e$ is a time expression.*

3 *If e and f are time expressions, then $e + f$, $e \dot{-} f$, $max(e, f)$ and $min(e, f)$ are all time expressions, where $e \dot{-} f$ represents the conditional expression $if\ e < f\ then\ 0\ else\ (e - f)$.*

By convention, for any time expression e, we have $e \leq \infty$, $\infty \circ e = \infty$, $max(e, \infty) = \infty$ and $min(e, \infty) = e$, where $\circ$ is $+$ or $\dot{-}$. We will write $e' - e$ in place of $e' \dot{-} e$ whenever $e \leq e'$.

Remark The decidability result of Chapter 5 will justify our decisions in the choice of time expressions. □

We also presuppose an infinite set V_p of process variables, ranged over by X, Y. The process expressions of Timed CCS, ranged over by E, F, are defined by the following BNF expression:

$$E ::= X \mid nil \mid \alpha(t)_e^{e'}.E \mid E + F \mid E \,|\, F \mid E \backslash a \mid E[S] \mid \mu X.E$$

where $S : Act \longrightarrow Act$ is a relabelling function which satisfies $\overline{S(a)} = S(\bar{a})$ and $S(\tau) = \tau$, e and e' are time expressions or e' is the infinite time ∞.

Let $\mathcal{E}$ represent the set of all processes. Process *nil* cannot do any action, but idles any time. Prefix $\alpha(t)_e^{e'}.E$ represents the process which performs action α

between time e and e' (inclusive), where e and e' are called the lower bound and upper bound of action α, respectively. Note that we only allow the upper bound to be the infinite time ∞ and therefore $\alpha(t)_{\infty}^{e}.nil$ is not a process. Any occurrences of time variable t in E refer to the happening time of action α. After α happens, it evolves to a process $E\{u/t\}$, where u is the time at which α happens and $E\{u/t\}$ is the result of substituting all free occurrence of t by u in E. Time variables in the language allow us to describe the notion of time dependency. For example, in the prefix $\alpha(t)_e^{e'}.E$, the behaviours of process E depend on the happening time of action α whenever t is a free time variable of E. Summation $E + F$ represents choice between processes E and F. The choice is made at the time of the first action of E or F, or at the time when only one process can idle (in this case the process which cannot delay is dropped from the future computation). Clearly the choice is deterministic with respect to time proceeding. Process $E \mid F$ represents the parallel composition of processes E and F. Each of them may perform actions independently, or they may synchronise on complementary actions which represent communications between them, analogous to CCS. Parallel composition is synchronous with respect to time proceeding, i.e. the parallel composition $E \mid F$ of the processes E and F can delay time u only when both E and F can.

There are two kinds of variables, time variables and process variables. The prefix operator $\alpha(t)_e^{e'}$ in $\alpha(t)_e^{e'}.E$ binds all free occurrences of time variable t in E and the recursive operator μX in $\mu X.E$ binds all free occurrences of process variable X in E. These give, in the usual sense, the notions of free and bound occurrences of time and process variables. We use $fv_t(E)$ and $fv_p(E)$ to represent the set of all free time and process variables, respectively, occurring in E. We identify those process expressions which are the same up to changes of bound time and process variables. We say a process expression E is closed with respect to time variables (or process variables) if there is no free occurrences of time variables (or process variables) in E, i.e. $fv_t(E) = \emptyset$ (or $fv_p(E) = \emptyset$). An agent is a process expression which is closed with respect to both time and process variables. Let

$\mathcal{P}$ represent the set of agents which is ranged over by P, Q, R. We occasionally allow multiple fixpoints of the form $\mu\vec{X}.\vec{E}$.

We say a process E is weakly guarded if every process variable of E is weakly guarded in E, where X is weakly guarded in E if every occurrence of X is in some subterm of form $\alpha(t)_e^{e'}.F$ of E. For example, the process $a(t)_0^{\infty}.X + b(s)_1^{10}.nil$ is weakly guarded. However the process $a(t)_2^{10}.X + X$ is not weakly guarded as the second occurrence of X is not guarded in it.

Notation For convenience, we will write $\alpha.(E\{0/t\})$ in place of $\alpha(t)_0^0.E$ and $\underline{\alpha}.E$ in place of $\alpha(t)_0^{\infty}.E$ whenever $t \notin fv_t(E)$. □

We will also regard infinite time ∞ as a time expression when it is not important to distinguish between a time expression and the infinite time ∞.

Remark For simplicity, we only consider closed intervals in Timed CCS, i.e. in prefix $\alpha(t)_e^{e'}.E$ the action α can happen at any time of the closed interval $e \leq t \leq e'$. However, it is easy to extend Timed CCS to also deal with open intervals when their least upper bounds exist. As an example, we may use four prefixes $\alpha(t)_{>e}^{<e'}.E$, $\alpha(t)_{\geq e}^{<e'}.E$, $\alpha(t)_{>e}^{\leq e'}.E$ and $\alpha(t)_{\geq e}^{\leq e'}.E$ to replace the present one. □

As in [Mil89a], we introduce a notion of sort which is needed in the definition of a duration control operator. We say a process E has a sort L ($L \subseteq Act$), or L is a sort of E, written $E : L$, if the actions of E and all its derivatives lie in $L \cup \{\tau\}$. As for CCS, there is a natural way to assign a sort $\mathcal{L}(E)$ to each process E of Timed CCS.

Definition 3.1.2 *Given sort $\mathcal{L}(X)$ for every process variable X, the syntactic sort $\mathcal{L}(E)$ of process E is inductively defined as follows:*

1 $\mathcal{L}(\tau(t)_e^{e'}.E) = \mathcal{L}(E)$

2 $\mathcal{L}(a(t)_e^{e'}.E) = \{a\} \cup \mathcal{L}(E)$

3 $\mathcal{L}(E + F) = \mathcal{L}(E) \cup \mathcal{L}(F)$

4 $\mathcal{L}(E \mid F) = \mathcal{L}(E) \cup \mathcal{L}(F)$

5 $\mathcal{L}(E \backslash a) = \mathcal{L}(E) - \{a, \bar{a}\}$

6 $\mathcal{L}(E[S]) = \{S(a) \, : \, a \in \mathcal{L}(E)\}$

7 $\mathcal{L}(\mu X.E) = \mathcal{L}(E)$

where, for every recursively defined process $\mu X.E$, the inclusion $\mathcal{L}(E) \subseteq \mathcal{L}(X)$ must hold.

It is easy to show that $\mathcal{L}(E)$ is a sort of process E, i.e. $E : \mathcal{L}(E)$ (see [Mil89a]).

3.2 Semantics

In this section, we consider an operational and a denotational semantics for Timed CCS.

3.2.1 Operational Semantics

In order to define an operational semantics for Timed CCS, we use the general notion of a labelled transition system

$$(S, \{\xrightarrow{t} \, : \, t \in T\})$$

which consists of a set S of *states*, a set T of *transition labels* and a *transition relation* $\xrightarrow{t} \subseteq S \times S$ for each $t \in T$.

In the transition system of Timed CCS, we take S to be $\mathcal{P}$, the set of agents, and T to be $(Act \cup \{\epsilon\}) \times \mathcal{T}$, the set of pairs of actions and times (recall that $\epsilon \notin Act$). For convenience, we will write $P \xrightarrow{\alpha}_u P'$ in place of $(P, P') \in \xrightarrow{(\alpha,u)}$, where $\alpha \in Act$, and $P \longrightarrow_u P'$ in place of $(P, P') \in \xrightarrow{(\epsilon,u)}$.

The understanding of transition $P \xrightarrow{\alpha}_u P'$ is that agent P performs action α at time u and then evolves to P'. The transition $P \longrightarrow_u P'$ means that agent P idles up to time u without any action and then evolves to P'. $P\{u/t\}$ represents the result of substituting all free occurrences of t in P by u.

To define the transition rules, we first define Moller and Tofts' maximal delay time of processes before any action [MT90].

Definition 3.2.1

(1) $\lvert X \rvert_{\mathcal{T}} = 0$	(5) $\lvert E \mid F \rvert_{\mathcal{T}} = min(\lvert E \rvert_{\mathcal{T}}, \lvert F \rvert_{\mathcal{T}})$
(2) $\lvert nil \rvert_{\mathcal{T}} = \infty$	(6) $\lvert E \backslash a \rvert_{\mathcal{T}} = \lvert E \rvert_{\mathcal{T}}$
(3) $\lvert \alpha(t)_e^{e'}.E \rvert_{\mathcal{T}} = e'$	(7) $\lvert E[S] \rvert_{\mathcal{T}} = \lvert E \rvert_{\mathcal{T}}$
(4) $\lvert E + F \rvert_{\mathcal{T}} = max(\lvert E \rvert_{\mathcal{T}}, \lvert F \rvert_{\mathcal{T}})$	(8) $\lvert \mu X.E \rvert_{\mathcal{T}} = \lvert E\{\mu X.E/X\} \rvert_{\mathcal{T}}$

Note that the maximal delay time $\lvert E \rvert_{\mathcal{T}}$ of process E is well defined only when E is weakly guarded. Henceforth we make this assumption about recursive terms.

All transition rules of the language are presented in Table 3–1. The rules are presented in natural deduction style which are read as follows: if the transition or transitions above the inference line can be inferred, then we can infer the transition below the line. The operational semantics of the language is then given by the least transition relations $\xrightarrow{\alpha}_u$ and $\longrightarrow_u$, where $\alpha \in$ Act and $u \in \mathcal{T}$, defined in Table 3–1.

For convenience, we introduce a notation δ to represent a *dead process.*

Notation $\delta \stackrel{def}{=} (a(t)_0^0\, nil)\backslash a$ □

$$\frac{}{\alpha(t)^{v'}_{v}.\,E \xrightarrow{\alpha}_u E\{u/t\}} \quad (v \le u \le v')$$

$$\frac{}{\alpha(t)^{v'}_{v}.\,E \longrightarrow_u \alpha(t)^{v'-u}_{v\dot{-}u}.\,(E\{u+t/t\})} \quad (u \le v')$$

$$\frac{}{nil \longrightarrow_u nil} \qquad \frac{P \longrightarrow_u P' \qquad Q \longrightarrow_u Q'}{P + Q \longrightarrow_u P' + Q'}$$

$$\frac{P \longrightarrow_u P'}{P + Q \longrightarrow_u P'} \quad (\,|\,Q\,|_{\mathcal{T}} < u) \qquad \frac{Q \longrightarrow_u Q'}{P + Q \longrightarrow_u Q'} \quad (\,|\,P\,|_{\mathcal{T}} < u)$$

$$\frac{P \xrightarrow{\alpha}_u P'}{P + Q \xrightarrow{\alpha}_u P'} \qquad \frac{Q \xrightarrow{\alpha}_u Q'}{P + Q \xrightarrow{\alpha}_u Q'}$$

$$\frac{P \longrightarrow_u P' \qquad Q \longrightarrow_u Q'}{P \,|\, Q \longrightarrow_u P' \,|\, Q'} \qquad \frac{P \xrightarrow{a}_u P' \qquad Q \xrightarrow{\bar{a}}_u Q'}{P \,|\, Q \xrightarrow{\tau}_u P' \,|\, Q'}$$

$$\frac{P \xrightarrow{\alpha}_u P' \qquad Q \longrightarrow_u Q'}{P \,|\, Q \xrightarrow{\alpha}_u P' \,|\, Q'} \qquad \frac{P \longrightarrow_u P' \qquad Q \xrightarrow{\alpha}_u Q'}{P \,|\, Q \xrightarrow{\alpha}_u P' \,|\, Q'}$$

$$\frac{P \longrightarrow_u P'}{P\backslash a \longrightarrow_u P'\backslash a} \qquad \frac{P \xrightarrow{\alpha}_u P'}{P\backslash a \xrightarrow{\alpha}_u P'\backslash a} \quad (\alpha \ne a \wedge \alpha \ne \bar{a})$$

$$\frac{P \longrightarrow_u P'}{P[S] \longrightarrow_u P'[S]} \qquad \frac{P \xrightarrow{\alpha}_u P'}{P[S] \xrightarrow{S(\alpha)}_u P'[S]}$$

$$\frac{E\{\mu X.E/X\} \longrightarrow_u P}{\mu X.E \longrightarrow_u P} \qquad \frac{E\{\mu X.E/X\} \xrightarrow{\alpha}_u P}{\mu X.E \xrightarrow{\alpha}_u P}$$

Table 3–1: Operational Rules for Timed CCS

Intuitively, the dead process δ neither performs any action nor idles. Note that we are abusing notation as δ also represents a timed synchronisation tree, but the intended meaning of it will be clear from the context.

Proposition 3.2.2 *For any weakly guarded agent P and $u \in \mathcal{T}$, $P \longrightarrow_u P'$ for some P' if and only if $u \leq \mid P \mid_{\mathcal{T}}$.*

Proof: ($\Leftarrow$) Suppose $u \leq \mid P \mid_{\mathcal{T}}$, where $u \in \mathcal{T}$, we proceed by transition induction.

Case 1 $P \equiv nil$, then $\mid P \mid_{\mathcal{T}} = \infty$ and $P \longrightarrow_u P$.

Case 2 $P \equiv \alpha(t)_v^w.Q$, then $\mid P \mid_{\mathcal{T}} = w$ and $P \longrightarrow_u \alpha(t)_{v-u}^{w-u}.(Q\{t+u/t\})$.

Case 3 $P \equiv P_1 + P_2$, then $\mid P \mid_{\mathcal{T}} = max(\mid P_1 \mid_{\mathcal{T}}, \mid P_2 \mid_{\mathcal{T}})$. Suppose $\mid P_1 \mid_{\mathcal{T}} \geq \mid P_2 \mid_{\mathcal{T}}$. By induction, $P_1 \longrightarrow_u P_1'$ for some P_1'. If $u \leq \mid P_2 \mid_{\mathcal{T}}$, then $P_2 \longrightarrow_u P_2'$ for some P_2' and $P \longrightarrow_u P_1' + P_2'$. If $\mid P_2 \mid_{\mathcal{T}} < u$, then $P \longrightarrow_u P_1'$. The case of $\mid P_1 \mid_{\mathcal{T}} \leq \mid P_2 \mid_{\mathcal{T}}$ is similar.

Case 4 $P \equiv P_1 \mid P_2$, $P \equiv Q\backslash a$ and $P \equiv Q[S]$ are similar to case 3.

Case 5 $P \equiv \mu X.E$, then $\mid E\{\mu X.E/X\} \mid_{\mathcal{T}} = \mid \mu X.E \mid_{\mathcal{T}}$. Since X is weakly guarded in E, by induction $E\{\mu X.E/X\} \longrightarrow_u P'$ for some P'. Therefore $P \longrightarrow_u P'$.

($\Rightarrow$) Suppose $P \longrightarrow_u P'$, we proceed by induction on the depth of the inference $P \longrightarrow_u P'$.

Case 1 $P \equiv nil$, then $\mid P \mid_{\mathcal{T}} = \infty$ and $u \leq \infty$.

Case 2 $P \equiv \alpha(t)_v^w.E$, then $u \leq w$ and $\mid P \mid_{\mathcal{T}} = w$.

Case 3 $P \equiv P_1 + P_2$, then $P_1 \longrightarrow_u P_1'$ and $P_2 \longrightarrow_u P_2'$ for some P_1' and P_2' such that $P' \equiv P_1' + P_2'$, or $P_1 \longrightarrow_u P'$ and $|P_2|_{\mathcal{T}} < u$, or $P_2 \longrightarrow_u P'$ and $|P_1|_{\mathcal{T}} < u$. If $P_1 \longrightarrow_u P_1'$ and $P_2 \longrightarrow_u P_2'$, where $P' \equiv P_1' + P_2'$, then by induction $u \leq |P_1'|_{\mathcal{T}}$ and $u \leq |P_2'|_{\mathcal{T}}$. So $u \leq max(|P_1|_{\mathcal{T}}, |P_2|_{\mathcal{T}}) = |P|_{\mathcal{T}}$. All other cases are similar.

Case 4 $P \equiv P_1 \mid P_2$, $P \equiv Q\backslash a$ and $P \equiv Q[S]$ are similar to case 3.

Case 5 $P \equiv \mu X.E$, then $E\{P/X\} \longrightarrow_u P'$. By induction, $u \leq |E\{P/X\}|_{\mathcal{T}} = |P|_{\mathcal{T}}$. □

Proposition 3.2.3 *For any agent P, we have*

(1) *if $P \longrightarrow_u P'$ and $P \longrightarrow_u P''$, then $P' \equiv P''$;*

(2) *if $P \longrightarrow_u P'$ and $P' \longrightarrow_v P''$, then $P \longrightarrow_{u+v} P''$;*

(3) *if $P \longrightarrow_u P'$ and $P' \xrightarrow{\alpha}_v P''$, then $P \xrightarrow{\alpha}_{u+v} P''$; and*

(4) *if $P \xrightarrow{\alpha}_u P'$, then $P \longrightarrow_u P''$ and $P'' \xrightarrow{\alpha}_0 P'$ for some P''.*

Proof: All these properties may be proved by induction on the depth of the inference of the form $P \longrightarrow_u P'$. We only consider (1), as all others are similar.

For (1), we proceed by induction on the depth of the inference of the form $P \longrightarrow_u P'$. Suppose $P \longrightarrow_u P'$ and $P \longrightarrow_u P''$

Case 1 $P \equiv nil$, then $P' \equiv nil \equiv P''$.

Case 2 $P \equiv \alpha(t)_v^{v'}.Q$, then $u \leq v'$ and $P' \equiv \alpha(t)_{v \dot{-} u}^{v' - u}.(Q\{u + t/t\}) \equiv P''$.

Case 3 $P \equiv P_1 + P_2$, then $u \leq max(|P_1|_{\mathcal{T}}, |P_2|_{\mathcal{T}})$. If $u \leq min(|P_1|_{\mathcal{T}}, |P_2|_{\mathcal{T}})$, then $P_1 \longrightarrow_u P_1'$ and $P_2 \longrightarrow_u P_2'$ for some P_1', P_2' such that $P' \equiv P_1' + P_2'$, and

$P_1 \longrightarrow_u P_1''$ and $P_2 \longrightarrow_u P_2''$ for some P_1', P_2' such that $P'' \equiv P_1'' + P_2''$. By induction, we have $P_1' \equiv P_1''$ and $P_2' \equiv P_2''$. So $P' \equiv P''$. If $\mid P_1 \mid_{\mathcal{T}} < u \leq \mid P_2 \mid_{\mathcal{T}}$ then $P_2 \longrightarrow_u P'$ and $P_2 \longrightarrow_u P''$. By induction, we have $P' \equiv P''$. Similarly, if $\mid P_2 \mid_{\mathcal{T}} < u \leq \mid P_1 \mid_{\mathcal{T}}$, we have $P' \equiv P''$.

Case 4 $P \equiv P_1 \mid P_2$, $P \equiv Q \backslash a$ or $P \equiv Q[S]$ similar to case 3.

Case 5 $P \equiv \mu X.E$, then $E\{\mu X.E/X\} \longrightarrow_u P'$ and $E\{\mu X.E/X\} \longrightarrow_u P''$. By induction, $P' \equiv P''$. □

(1) shows the property of time determinacy. (2) and (3) show the property of time continuity and (4) says that atomic actions are instantaneous.

3.2.2 Denotational Semantics

Since Ts can be consider as a CPO, we can use Ts to interpret Timed CCS. To deal with process variables, we introduce a notion of environments which are mappings from V_p to Ts. Let Env be the set of environments, ranged over by ρ. For any environment ρ, $\rho\{T/X\}$ represents a new environment which differs from ρ only at X, where it is defined to be T. The denotational semantics is given by a mapping

$$\mathcal{M} : \mathcal{P} \longrightarrow (Env \longrightarrow Ts)$$

which is defined by induction on processes.

Definition 3.2.4 *For any process $E \in \mathcal{P}$ and environment ρ, the denotation of E with respect to ρ, written as $\mathcal{M}(E)\rho$, is defined by induction on E as*

(1) $\mathcal{M}(X)\rho = \rho(X)$

(2) $\mathcal{M}(nil)\rho = (\infty)\delta$

(3) $\mathcal{M}(a(t)_e^{e'}.E) = a(\{u : e \leq u \leq e'\})(\lambda t.\mathcal{M}(E))$

(4) $\mathcal{M}(E+F)\rho = \mathcal{M}(E)\rho + \mathcal{M}(F)\rho$

(5) $\mathcal{M}(E \,|\, F)\rho = \mathcal{M}(E)\rho \,|\, \mathcal{M}(F)\rho$

(6) $\mathcal{M}(E\backslash b)\rho = (\mathcal{M}(E)\rho)\backslash b$

(7) $\mathcal{M}(E[S])\rho = (\mathcal{M}(E)\rho)[S]$

(8) $\mathcal{M}(\mu X.E)\rho = \mu T.(\mathcal{M}(E)\rho\{T/X\})$

Note that, if E is an agent, then $\mathcal{M}(E)$ is a constant function from Env to Ts, which may be regarded as an element of Ts.

Theorem 3.2.5 *If an agent P is weakly guarded, then $\mathcal{M}(P)$ contains no open vertices, i.e. $\mathcal{M}(P)$ is a closed timed synchronisation tree.*

Proof: Suppose P is weakly guarded. We show by induction on P that $\mathcal{M}(P)$ contains no open vertices. Clearly, $\mathcal{M}(nil)$ contains no open vertices and prefix, summation, restriction, parallel and relabelling operations preserve the property of no open vertices. The only nontrivial case is when P is of the form $\mu X.Q$. In this case, we use the fact that $\mathcal{M}(P) = \bigsqcup_{i\in\omega} \mathcal{M}(P^{(i)})$, where $P^{(0)} = \Omega$ and $P^{(i+1)} = Q\{P^{(i)}/X\}$. Clearly, for any vertex v of depth less than i in $P^{(i)}$, v cannot be an open vertex. It follows by the construction of limits that $\mathcal{M}(P)$ contains no open vertices. $\Box$

Definition 3.2.6 *For any weakly guarded agents P and Q, we say $P \simeq Q$ if $\mathcal{M}(P) \simeq \mathcal{M}(Q)$.*

Lemma 3.2.7 *For every weakly guarded agent P, $t(r) = \big|\, P \,\big|_T$ where $(V, E, r, l, t) \in \mathcal{M}(P)$.*

Proof: For every agent P, let $(V, E, r, l, t) \in \mathcal{M}(P)$. By transition induction, we have

Case 1 $P \equiv nil$, then $\mid nil \mid_{\mathcal{T}} = \infty$ and clearly $t(r) = \infty$.

Case 2 $P \equiv \alpha(t)_e^{e'}.Q$, then $\mid P \mid_{\mathcal{T}} = e'$ and $t(r) = Sup(\{u \,:\, u \leq e'\}) = e'$.

Case 3 $P \equiv P_1 + P_2$, then $\mid P \mid_{\mathcal{T}} = max(\mid P_1 \mid_{\mathcal{T}}, \mid P_2 \mid_{\mathcal{T}})$. Let $(V_i, E_i, r_i, l_i, t_i) \in \mathcal{M}(P_i)$ where $i = 1, 2$. By induction, we have $t_i(r_i) = \mid P_i \mid_{\mathcal{T}}$ and $t(r) = max(t_1(r_1), t_2(r_2))$.

Case 4 $P \equiv \mu X.E$. Since P is weakly guarded, we have $t'(r') = \mid E\{nil/X\} \mid_{\mathcal{T}}$, where $(V', E', r', l', t') \in \mathcal{M}(E)\rho\{\delta/T\}$ for any environment ρ. It follows by the construction of limits that $t(r) = \mid P \mid_{\mathcal{T}}$, where $(V, E, r, l, t) \in \mathcal{M}(P)$.

All other cases are similar. □

Theorem 3.2.8 *$P \simeq Q$ implies $\mid P \mid_{\mathcal{T}} = \mid Q \mid_{\mathcal{T}}$.*

Proof: The result follows from Lemma 3.2.7. □

3.3 Applications

In this section, we define some derived operators and give some examples to demonstrate the use of the calculus for the specification of real-time systems. The examples will deal with three important aspects of real-time systems: time out, duration control and time dependency.

3.3.1 Derived Operators

We will only consider those derived operators which will be used in the sequel.

Time Prefix

The first derived operator is the time prefix of [MT90].

Definition 3.3.1 *For any time expression e and weakly guarded process E, the time prefix $(e)E$ is defined inductively as follows:*

- $(e)nil = nil$
- $(e)(a(t)_{f}^{f'}.\,E) = a(t)_{f+e}^{f'+e}.\,(E\{t-e/t\})$ *where* $t \notin fv(e)$
- $(e)(E+F) = (e)E + (e)F$
- $(e)(E \,|\, F) = (e)E \,|\, (e)F$
- $(e)(E \backslash a) = ((e)E)\backslash a$
- $(e)(E[S]) = ((e)E)[S]$
- $(e)(\mu X.E) = (e)(E\{\mu X.E/X\})$

Intuitively, $(e)E$ represents a process which, after a delay of time e, has the same behaviour as process E.

For the time prefix operator, we have the following derived transition relation

$$(u+v)P \longrightarrow_{u} (v)P$$

and transition rules

$$\frac{P \xrightarrow{\alpha}_{v} P'}{(u)P \xrightarrow{\alpha}_{u+v} P'} \qquad\qquad \frac{P \longrightarrow_{v} P'}{(u)P \longrightarrow_{u+v} P'}$$

Note that ∞ is not a time expression and therefore $(\infty)\delta$ is not a process. For convenience, we allow ourselves to write $(\infty)\delta$ as syntactically identical to the process *nil* in the sequel.

Time Shift

The second derived operator is time shift which is the relative version of that of [BB91].

Definition 3.3.2 *For any time expression e and weakly guarded process E, the time shift $e >> E$ is defined inductively as follows:*

- $e >> nil = nil$
- $e >> (\alpha(t)^{f'}_{f}.E) = \alpha(t)^{f' \dot{-} e}_{(f \dot{-} e)+(max(e,f')-f')}.(E\{e+t/t\})$ *where* $t \notin fv(e)$
- $e >> (E+F) = e >> E + e >> F$
- $e >> (E \mid F) = e >> E \mid e >> F$
- $e >> (E \backslash a) = (e >> E)\backslash a$
- $e >> (E[S]) = (e >> E)[S]$
- $e >> (\mu X.E) = e >> (E\{\mu X.E/X\})$

Informally, for any process E, if $e \leq \left| E \right|_{\mathcal{T}}$, then $e >> E$ represents a state of E after a delay of time e without any action. If $e > \left| E \right|_{\mathcal{T}}$, then $e \gg E$ represents the dead process δ. Note the subtlety in the definition for $e >> (\alpha(t)^{f'}_{f}.E)$. It ensures that the lower bound $(f \dot{-} e) + (max(e, f') - f') > 0$ whenever $e > f'$.

For the time shift operator, we have the following derived transition rules

$$\frac{P \longrightarrow_{u+v} P'}{u >> P \longrightarrow_{v} P'} \qquad \frac{P \xrightarrow{\alpha}_{u+v} P'}{u >> P \xrightarrow{\alpha}_{v} P'}$$

Duration Control

In real-time systems, one important aspect of time-critical behaviour is duration control, which says that a process is allowed to run for a certain length of time, after that the control is passed to another process.

Consider a process which may behave as process E within time e and from time e it will behave as process F. We use $E\mathcal{U}_e F$ to represent such a process.

Definition 3.3.3 *For weakly guarded processes E and F, the duration control $E\mathcal{U}_e F$ is inductively defined as follows:*

- $nil\mathcal{U}_e F = (e)F$
- $(\alpha(t)_f^{f'}.E)\mathcal{U}_e F = \alpha(t)_f^{min(e,f')}.(E\mathcal{U}_{e-t}F) + (e)F$
- $(E + E')\mathcal{U}_e F = (E\mathcal{U}_e F) + (E'\mathcal{U}_e F)$
- $(E \mid E')\mathcal{U}_e F = (E\mathcal{U}_e F) \mid (E'\mathcal{U}_e nil)$
- $(E\backslash a)\mathcal{U}_e F = ((E\mathcal{U}_e(F[\lambda/a]))\backslash a)[a/\lambda]$ *where λ is a fresh action for E and F*
- $(E[S])\mathcal{U}_e F = (E\mathcal{U}_e(F[S']))[S][S'^{-1}]$ *where $S'|_{\mathcal{L}(F)} : \mathcal{L}(F) \longrightarrow \Lambda - (\mathcal{L}(E) \cup \mathcal{L}(F))$ and S' is a bijection*
- $(\mu X.E)\mathcal{U}_e F = (E\{\mu x.E/X\})\mathcal{U}_e F$

Remark For any finite subsets A and B of Λ, where we insist that Λ is countably infinite, there is a relabelling function $S : \Lambda \longrightarrow \Lambda$ which is a bijection and the range of $S|_A$ is $\Lambda - B$. Since Λ is countable, we can construct such a relabelling function S as follows: (1) for the ith element a of A, where $i = 1, 2, \cdots$, let $S(a)$ be the ith element of $\Lambda - B$; (2) for the ith element a of $\Lambda - B$, where $i \leq n$ (n is the size of A), let $S(a)$ be the ith element of A; (3) for the jth element a of $\Lambda - B$, where $j > n$, let $S(a)$ be a; and (4) for every element b of $B - A$, let $S(b)$ be b. $\Box$

For the duration control operator, we have the following derived transition rules

$$\frac{Q \xrightarrow{\alpha}_v Q'}{P\,\mathcal{U}_u Q \xrightarrow{\alpha}_{u+v} Q'} \qquad \frac{Q \longrightarrow_v Q'}{P\,\mathcal{U}_u Q \longrightarrow_{u+v} Q'} \quad v > 0$$

and

$$\frac{P \xrightarrow{\alpha}_u P'}{P\,\mathcal{U}_{u+v} Q \xrightarrow{\alpha}_u P'\,\mathcal{U}_v Q} \qquad \frac{P \longrightarrow_u P'}{P\,\mathcal{U}_{u+v} Q \longrightarrow_u P'\,\mathcal{U}_v Q}$$

Remark Process $E\,\mathcal{U}_e F$, even if E is the dead process, will behave as process F after time e. For example, process $\delta\,\mathcal{U}_{10} F$ has the same behaviour as process $(10)F$. To define a strong version of duration control operator such that process $\delta\,\mathcal{U}_{10} F$ represents the dead process δ, not $(10)F$, we can introduce a condition operator $[e = f]E$ which represents process E when condition $e = f$ holds, otherwise it represents the dead process δ.

3.3.2 Examples

In this section, we consider three examples and show how to describe the notions of time out, duration control and time dependency in Timed CCS.

Alternating Bit Protocol (Time Out)

Consider a timed implementation of the Alternating Bit Protocol which consists of four parts: *Sender*, *Receiver* and two unreliable transition lines, *Ack* and *Trans*. The unreliable transition lines may lose (but not corrupt) messages. Messages and acknowledgements are sent tagged with bits 0 and 1 alternately. Let $\hat{b} = 1 - b$, where $b = 0, 1$.

Sender, after accepting a message from the environment, sends the message together with an attached bit b (we assume the first message is tagged with 0), where $b = 0$ or 1, along the line *Trans*. Then it waits for an appropriate acknowledgement within time D, where D is the time *Sender* will wait before assuming the message had been lost. If it receives a correct acknowledgement within time D, *Sender* becomes ready to accept the next message from the environment and then transmits it with a bit $\hat{b}$. If it receives a wrong acknowledgement within time D, it retransmits the message after a delay of time Δ. If it does not receive any acknowledgement after a delay of time D, it retransmits the message.

$$Sender = Sender_0$$

$$Sender_b = \underline{accept}.(\Delta)\,\overline{send_b}.(\Delta)\,Sending_b$$

$$\begin{aligned} Sending_b = {} & ack_{\hat{b}}(t)_0^D.\,(\Delta)\,\overline{send_b}.(\Delta)\,Sending_b + \\ & ack_b(t)_0^D.\,(\Delta)\,Sender_{\hat{b}} + (D)\,\overline{send_b}.(\Delta)\,Sending_b \end{aligned}$$

Receiver, which we assume starts in a state waiting for a message tagged with 0, delivers the message after it receives a correct message along the transition line *Trans* and then transmits its acknowledgement along another transition line *Ack*. After doing so, it waits for a new message within time D, where D is the time that *Receiver* will wait before assuming its acknowledgement had been lost. If it receives a new message within time D, it again evolves to a state of delivering a message. If it receives the old message within time D, then it retransmits the acknowledgement after a delay of time Δ. If it does not receive any message after a delay of time D, it retransmits the acknowledgement.

$$Receiver = \underline{tran_0}.(\Delta)\,Deliver_0$$

$$Deliver_b = \overline{deliver}.(\Delta)\,\overline{reply_b}.(\Delta)\,Reply_b$$

$$Reply_b = tran_b(t)_0^D.(\Delta)\,\overline{reply_b}.(\Delta)\,Reply_b +$$
$$tran_{\hat{b}}(t)_0^D.(\Delta)\,Deliver_{\hat{b}} + (D+\Delta)\,\overline{reply}_b(\Delta)\,Reply_b$$

Two unreliable transition lines *Trans* and *Ack* behave as buffers of capacity one, but they may lose messages. We suppose the transition time for both transition lines is D'.

$$Trans = \underline{send_0}.(\Delta)\,(\tau(s)_0^{D'}.(\Delta)\,Trans + (D')\,\overline{tran_0}.(\Delta)\,Trans)$$
$$+\,\underline{send_1}.(\Delta)\,(\tau(s)_0^{D'}.(\Delta)\,Trans + (D')\,\overline{tran_1}.(\Delta)\,Trans)$$

$$Ack = \underline{reply_0}.(\Delta)\,(\tau(s)_0^{D'}.(\Delta)\,Ack + (D')\,\overline{ack_0}.(\Delta)\,Ack)$$
$$+\,\underline{reply_1}.(\Delta)\,(\tau(s)_0^{D'}.(\Delta)\,Ack + (D')\,\overline{ack_1}.(\Delta)\,Ack)$$

Now we may define the protocol as

$$Protocol = (Sender \mid Receiver \mid Trans \mid Ack)\backslash L$$

where $L = \{send_0,\ send_1,\ tran_0,\ tran_1,\ ack_0,\ ack_1,\ reply_0,\ reply_1\}$.

Since the least time for *Sender* to receive a correct acknowledgement after sending a message to the transition line is $2D' + 4\Delta$, where $D'(>> \Delta)$ is the transition delay of the transition lines, we should take the timeout $D > 2D' + 4\Delta$.

Login Procedure (Duration Control)

Consider a user who attempts to use computers in a computer centre through a terminal. After the terminal is connected to a system, the system waits for the user's valid login response. If the user enters a valid response r within 2 minutes, then the login procedure passes the control to a process Q which formalizes the user's task. If the user's login response is invalid, the system returns and still waits for user's valid response. After waiting for 2 minutes without any valid response, the system rejects the login request and the connection between the terminal and the system is broken.

$$LP = \underline{connect}.(\Delta)\,(P \,|\, (\underline{b}.Q \;+\; \underline{a}.nil))\backslash\{a, b\}$$

$$P = L\;\mathcal{U}_2\,\bar{a}.LP$$

$$L = \underline{r}.(\Delta)\,\bar{b}.nil + \underline{ir}.(\Delta)L$$

Radar Gun (Time Dependency)

People who live in Glasgow but work in Edinburgh always like to drive as fast as possible on M8 between Glasgow and Edinburgh. We suppose the distance between Glasgow and Edinburgh is 45 (miles) and the maximal speed of a car is 90 (mph). However traffic wardens try to stop and punish heavily those people who drive too fast. We assume the maximum speed which is allowed on M8 is 70 (mph). When people leave Glasgow, they drive at speed 90 (mph) until they find by using their radar detectors that traffic wardens are using radar guns to measure their cars' speed. After noticing the policemen's presence, they immediately change their speed to 70 (mph) to avoid punishment. When they pass the policemen, they immediately return to 90 (mph) and drive at that speed until the next appearances of policemen. If there are no police on their way, people only require half an hour to arrive in Edinburgh. Otherwise, it takes longer.

$$\begin{aligned} Car = \underline{depart}.(&(0.5)arrive.nil \\ &+ gunfire(t)_0^{0.5}.\,\overline{pass}(s)_0^{70^{-1}(45-90t)}.\,Car'(45 - 90t - 70s)) \end{aligned}$$

$$\begin{aligned} Car'(x) = (&90^{-1}x)arrive.nil \\ &+ gunfire(t)_0^{90^{-1}x}.\,\overline{pass}(s)_0^{70^{-1}(x-90t)}.\,Car'(x - 90t - 70s) \end{aligned}$$

$Police = \underline{\overline{gunfire}}.\underline{pass}.Police$

For every car which runs from Glasgow to Edinburgh, we are only interested in the time it leaves Glasgow and the time it arrives in Edinburgh. So we define the whole system as:

$Sys = (Car|Police)\backslash\{gunfire, pass\}$

3.4 Conclusion

In this chapter, we proposed a timed calculus, *Timed CCS*, which is an extension of Milner's CCS with time. We made no assumptions about the underlying nature of time, allowing it to be discrete, such as the natural numbers, or dense, such as the rationals and the reals. We also made no assumption of the *Maximal Progress Principle* which says that a process cannot delay when it may perform some internal action. Timed CCS is an interleaving model for real-time systems. Time variables in Timed CCS allow us to express the notion of time dependency in real-time systems. We gave two semantics, an operational and a denotational semantics, for Timed CCS and will show in Chapter 4 that these two semantics coincide in a sense that the equivalence on Timed CCS induced by the operational semantics is the same as the equivalence induced by the denotational semantics. We also presented some examples to show the use of the calculus for the specification of real-time systems.

Recently there have been a number of attempts to introduce time in well-developed process algebras. These include extensions of Milner's CCS with time [CAM90,HR90,MT90,Wan90,Wan91], extensions of Hoare's CSP with time [RR88, DS89,Sch91] and extensions of ACP with time [BB91,Gro90,Sif90]. Among these works, there are many features in common. For example, most works suppose atomic actions take no time, i.e. actions are instantaneous. This assumption is essential if we are to use an interleaving model for real-time systems. Also many works insist that the progress of time is deterministic which can be expressed as:

$$\text{for every agent } P,\ P \longrightarrow_u P' \text{ and } P \longrightarrow_u P'' \text{ implies } P' \equiv P''.$$

However, some work such as [Gro90] believes that the progress of time is not deterministic and idling also forces a choice of behaviour.

Some work such as [CAM90,HR90,Wan91,RR88,DS89,Sch91] is based on the assumption of the Maximal Progress Principle. We do not believe the Maximal Progress Principle is a natural assumption, as the internal behaviour of a non-deterministic machine is uncontrollable. How can we force an invisible and uncontrollable event to occur immediately?

The work such as [HR90,Gro90,Sif90] only deals with discrete time. In fact all these works introduce a special action to represent one unit time. In general these works cannot express the notion of time dependency. For example, consider a machine M which can perform an action a followed by an action b. The action a can happen at any time and if a happens at time t, then the action b must happen within another t time. We can define the machine M in discrete Timed CCS as:

$$M \stackrel{def}{=} a(t)_0^{\infty}.\, b(s)_0^{t}.\, nil$$

It is not clear how this machine M could be defined in these calculi.

In [MT90], Moller and Tofts also make no assumption of the underlying nature of time, but they only provide expansion laws for discrete time. Similarly it is not clear how to define the machine M in TCCS of [MT90].

There are two ways to deal with the time dependency property: one is to introduce an infinite choice operator and the other is to introduce time variables. The extension of the syntax to include an infinite choice operator is very convenient descriptively but the mathematical foundations are immediately more complicated.

As an example, if we allow an infinite sum in timed synchronisation trees, then a hard question of whether the construction is continuous arises immediately.

In [BB91], Baeten and Bergstra introduce a notion of integration which in fact is just a special form of infinite summation. $ACP\rho I$ is more general and complicated than Timed CCS. As pointed out by Klusener in [Klu91] that we do not know real-life processes which are ruled out by restricting to "prefixed integration" processes. However, if we restrict to those "prefixed integration" processes, the resulting language corresponds to Timed CCS.

In [Wan91], Wang also independently introduces time variables to fix a serious error in the expansion law of his early paper [Wan90] (the error was pointed out to Wang by the author of this thesis). The work of [Wan90,Wan91] is based on the assumption of the maximal progress principle and the ability of expressing the notion of time out is achieved by using the property that the internal action τ cannot delay. Due to the assumption of the Maximal Progress Principle, it is difficult to extend the work of [Wan90,Wan91] to deal with open intervals over dense time domains. As a consequence, we cannot express a notion of some action happening before a certain time or a time out happens for some action at that time by extending [Wan90,Wan91]. For example, we cannot define a machine P which is defined in a possible extension of Timed CCS as

$$P \stackrel{def}{=} a(t)^{<25}_{\geq 0}.\, nil + b(s)^{<\infty}_{\geq 25}.\, nil$$

by extending [Wan90,Wan91]. This is a drawback, as an adequate description of time out becomes impossible.

Chapter 4

Strong Equivalence

A more abstract semantics for Timed CCS is achieved by identifying those agents which cannot be distinguished by any external observation of their behaviour. The notion of bisimulation [Par81,Mil89a] is a standard device for defining behavioural equivalence for process algebras. In this chapter, we extend the notion of strong bisimulation of [Mil89a] to timed processes. Strong equivalence treats the internal action τ in the same way as all other actions and is a rather strict equivalence relation. However strong equivalence enjoys many nice properties. It is a congruence relation and many of the equational laws including an expansion law which we wish to use in practice are valid for it. It is decidable for finite processes (see Chapter 5) and there is a simple proof system (see Chapter 6) which is complete for finite processes over dense time domains.

4.1 Strong Bisimulation

We do not wish to distinguish agents which, in some sense, have the same behaviour. The notion of bisimulation between agents captures the idea of having the same behaviour. We say two agents are not equivalent if a distinction can be detected by an experimenter who interacts with each of them.

Definition 4.1.1 *A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a strong bisimulation if $(P, Q) \in \mathcal{R}$ implies that for any $\alpha \in Act$ and $u \in \mathcal{T}$*

(1) *if $P \xrightarrow{\alpha}_u P'$, then $Q \xrightarrow{\alpha}_u Q'$ and $(P', Q') \in \mathcal{R}$ for some Q';*

(2) *if $P \longrightarrow_u P'$, then $Q \longrightarrow_u Q'$ and $(P', Q') \in \mathcal{R}$ for some Q';*

(3) *if $Q \xrightarrow{\alpha}_u Q'$, then $P \xrightarrow{\alpha}_u P'$ and $(P', Q') \in \mathcal{R}$ for some P'; and*

(4) *if $Q \longrightarrow_u Q'$, then $P \longrightarrow_u P'$ and $(P', Q') \in \mathcal{R}$ for some P'.*

We say two agents P and Q are strongly bisimilar, denoted by $P \sim Q$, if there is a strong bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

Definition 4.1.2 $\sim = \bigcup\{\mathcal{R} : \mathcal{R}$ *is a strong bisimulation*$\}$.

Proposition 4.1.3 $\sim$ *is the largest strong bisimulation. Moreover, it is an equivalence relation.*

Proof: The proof follows from the facts that the union, composition and inverse of strong bisimulations are still strong bisimulations and also the identity over agents is a strong bisimulation. □

Lemma 4.1.4 *For any agents P and Q, $P \sim Q$ implies $|P|_{\mathcal{T}} = |Q|_{\mathcal{T}}$.*

Proof: The result follows from Proposition 3.2.2. □

Definition 4.1.5 *A binary relation $\mathcal{R}$ over $\mathcal{P}$ is a strong $\mathcal{T}$-bisimulation if $(P, Q) \in \mathcal{R}$ implies that for any $\alpha \in Act$ and $u \in \mathcal{T}$*

(1) *if $P \xrightarrow{a}_u P'$, then $Q \xrightarrow{a}_u Q'$ and $(P', Q') \in \mathcal{R}$ for some Q';*

(2) *if $Q \xrightarrow{a}_u Q'$, then $P \xrightarrow{a}_u P'$ and $(P', Q') \in \mathcal{R}$ for some P'; and*

(3) $|P|_{\mathcal{T}} = |Q|_{\mathcal{T}}$.

Similarly to strong bisimulation, we say two agents P and Q are strong $\mathcal{T}$-bisimilar, denoted by $P \sim_{\mathcal{T}} Q$, if there is a strong $\mathcal{T}$-bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

Definition 4.1.6 $\sim_{\mathcal{T}} = \bigcup\{\mathcal{R} \,:\, \mathcal{R}$ *is a strong* $\mathcal{T}-$*bisimulation*$\}$

"Bisimulation up to" of [Mil89a] is a useful technique for reducing the size of the relation needed to define a bisimulation. We generalize it to timed case and first define a notion of $\mathcal{T}$-bisimulation up to $\sim_{\mathcal{T}}$.

Definition 4.1.7 *We say* $\mathcal{S}$ *is a strong* $\mathcal{T}$*-bisimulation up to* $\sim_{\mathcal{T}}$ *if* $P\mathcal{S}Q$ *implies that*

(1) *whenever* $P \xrightarrow{\alpha}_u P'$ *then* $Q \xrightarrow{\alpha}_u Q'$ *and* $P' \sim_{\mathcal{T}} \mathcal{S} \sim_{\mathcal{T}} Q'$ *for some* Q'*;*

(2) *whenever* $Q \xrightarrow{\alpha}_u Q'$ *then* $P \xrightarrow{\alpha}_u P'$ *and* $P' \sim_{\mathcal{T}} \mathcal{S} \sim_{\mathcal{T}} Q'$ *for some* P'*; and*

(3) $\mid P \mid_{\mathcal{T}} = \mid Q \mid_{\mathcal{T}}$.

Similarly we can define a notion of strong bisimulation up to $\sim$.

Definition 4.1.8 *We say* $\mathcal{S}$ *is a strong bisimulation up to* $\sim$ *if* $P\mathcal{S}Q$ *implies that*

(1) *if* $P \xrightarrow{\alpha}_u P'$*, then* $Q \xrightarrow{\alpha}_u Q'$ *and* $P' \sim \mathcal{S} \sim Q'$ *for some* Q'*;*

(2) *if* $P \longrightarrow_u P'$*, then* $Q \longrightarrow_u Q'$ *and* $P' \sim \mathcal{S} \sim Q'$ *for some* Q'*;*

(3) *if* $Q \xrightarrow{\alpha}_u Q'$*, then* $P \xrightarrow{\alpha}_u P'$ *and* $P' \sim \mathcal{S} \sim Q'$ *for some* P'*; and*

(4) *if* $Q \longrightarrow_u Q'$*, then* $P \longrightarrow_u P'$ *and* $P' \sim \mathcal{S} \sim Q'$ *for some* P'.

Proposition 4.1.9

1 *If $\mathcal{S}$ is a strong $\mathcal{T}$-bisimulation up to $\sim_{\mathcal{T}}$, then $\mathcal{S} \subseteq \sim_{\mathcal{T}}$.*

2 *If $\mathcal{S}$ is a strong bisimulation up to $\sim$, then $\mathcal{S} \subseteq \sim$.*

Proof: Suppose $\mathcal{S}$ is a strong $\mathcal{T}$-bisimulation (strong bisimulation) up to $\sim_{\mathcal{T}}$ ($\sim$) . We only need to show that $\sim_{\mathcal{T}} \mathcal{S} \sim_{\mathcal{T}}$ ($\sim \mathcal{S} \sim$) is a strong $\mathcal{T}$-bisimulation (strong bisimulation), which is analogous to Lemma 4.3.5 of [Mil89a]. □

Hence, to prove $P \sim_{\mathcal{T}} Q$ ($P \sim Q$), we only need to find a strong $\mathcal{T}$-bisimulation (strong bisimulation) up to $\sim_{\mathcal{T}}$ ($\sim$) which contains (P, Q).

Lemma 4.1.10 *For weakly guarded agent P, $P \longrightarrow_u P'$ implies $P' \sim u >> P$.*

Proof: We proceed by induction on the depth of the inference of the form $P \longrightarrow_u P'$. Suppose $P \longrightarrow_u P'$.

Case 1 $P \equiv nil$, then $P' \equiv nil \sim u >> P$.

Case 2 $P \equiv \alpha(t)^{v'}_{v}.Q$, then $v \leq u \leq v'$ and $P' \equiv \alpha(t)^{v'-u}_{v\dot{-}u}.(Q\{u+t/t\}) \sim u >> P$.

Case 3 $P \equiv P_1 + P_2$, then $u \leq max(\mid P_1 \mid_{\mathcal{T}}, \mid P_2 \mid_{\mathcal{T}})$. If $u \leq min(\mid P_1 \mid_{\mathcal{T}}, \mid P_2 \mid_{\mathcal{T}})$, then $P_1 \longrightarrow_u P_1'$, $P_2 \longrightarrow_u P_2'$ and $P' \equiv P_1' + P_2'$. By induction, we have $P_1' \sim u >> P_1$ and $P_2' \sim u >> P_2$. Hence $P' \sim u >> P$. If $\mid P_1 \mid_{\mathcal{T}} < u \leq \mid P_2 \mid_{\mathcal{T}}$, then $P_2 \longrightarrow_u P'$. By induction, we have $P' \sim u >> P_2$. Since $\mid P_1 \mid_{\mathcal{T}} < u$, we have $u >> P_1 \sim \delta$. Hence $P' \sim u >> P$. The case of $\mid P_2 \mid_{\mathcal{T}} < u \leq \mid P_1 \mid_{\mathcal{T}}$ is similar to that of $\mid P_1 \mid_{\mathcal{T}} < u \leq \mid P_2 \mid_{\mathcal{T}}$.

Case 4 $P \equiv P_1 \mid P_2$, $P \equiv Q \backslash a$ or $P \equiv Q[S]$, similar to case 3.

Case 5 $P \equiv \mu X.E$, then $E\{\mu X.E/X\} \longrightarrow_u P'$. By induction,

$$P' \sim u >> E\{\mu X.E/X\} = u >> P$$

□

Due to the property of time determinacy, the equivalences $\sim$ and $\sim_{\mathcal{T}}$ coincide over weakly guarded agents. As the maximal delay time of a process is syntactically defined, it is easier to show a relation of strong $\mathcal{T}$-bisimulation than that of strong bisimulation.

Proposition 4.1.11 *For any weakly guarded agents P and Q, $P \sim Q$ if and only if $P \sim_{\mathcal{T}} Q$.*

Proof: For direction ($\Rightarrow$), it is enough to show that the relation

$$\sim \cap \{(P, Q) \,:\, P \textit{ and } Q \textit{ are weakly guarded}\}$$

is a strong $\mathcal{T}$-bisimulation. The result follows from the definitions of strong bisimulations and strong $\mathcal{T}$-bisimulations.

For direction ($\Leftarrow$), we only need to show that

$$\{(u >> P, u >> Q) \,:\, P \sim_{\mathcal{T}} Q \,\&\, u \leq \left| P \right|_{\mathcal{T}} \& \; P \textit{ and } Q \textit{ are weakly guarded}\}$$

is a strong bisimulation up to $\sim$. The result follows from Lemma 4.1.10 and Proposition 3.2.2. □

4.2 Operational Characterization of $\simeq$

In Chapter 3, we gave an operational and a denotational semantics for Timed CCS. In this section, we show that the two semantics of Timed CCS coincide over weakly guarded agents, i.e. the equivalence $\simeq$ and $\sim$ are identical over weakly guarded agents. This informally states that the model $\mathcal{T}s$ differentiates two machines if and only if they can be differentiated by an experimenter.

Lemma 4.2.1 *For any weakly guarded agent P, $P \xrightarrow{\alpha}_u P'$ implies $\mathcal{M}(P) \xrightarrow{\alpha}_u \mathcal{M}(P')$.*

Proof: See [Mil80]. □

Lemma 4.2.2 *For any weakly guarded agent P, $\mathcal{M}(P) \xrightarrow{\alpha}_u T$ implies $P \xrightarrow{\alpha}_u P'$ and $\mathcal{M}(P') = T$ for some agent P'.*

Proof: See [Mil80]. $\Box$

Theorem 4.2.3 (Operational Characterization Theorem)
For any weakly guarded agents P and Q, $P \sim Q$ if and only if $P \simeq Q$.

Proof: ($\Rightarrow$) It is enough to show that

$$\mathcal{S} = \{(\mathcal{M}(P), \mathcal{M}(Q)) \,:\, P \sim Q\}$$

is a tree bisimulation.

(1) if $\mathcal{M}(P) \xrightarrow{\alpha}_u T$, then $P \xrightarrow{\alpha}_u P'$ and $\mathcal{M}(P') = T$. Since $P \sim Q$, there is a Q' such that $Q \xrightarrow{\alpha}_u Q'$ and $P' \sim Q'$. So $\mathcal{M}(Q) \xrightarrow{\alpha}_u T'$ with $\mathcal{M}(Q') = T'$ and $(\mathcal{M}(P'), \mathcal{M}(Q')) \in \mathcal{S}$.

(2) if $\mathcal{M}(Q) \xrightarrow{\alpha}_u T$, similar to (1).

(3) $t(r) = \mid P \mid_{\mathcal{T}} = \mid Q \mid_{\mathcal{T}} = t'(r')$, where $(V, E, r, l, t) \in \mathcal{M}(P)$ and $(V', E', r', l', t') \in \mathcal{M}(Q)$.

($\Leftarrow$) It is easy to check that the relation

$$\mathcal{S} = \{(P, Q) \,:\, P \simeq Q \textit{ where } P, Q \textit{ weakly guarded}\}$$

is a strong $\mathcal{T}$-bisimulation. $\Box$

4.3 Properties of Strong Equivalence

In this section, we present some properties for strong equivalence. The first five propositions are proved in a standard way by demonstrating appropriate strong bisimulations.

Proposition 4.3.1 (Laws for Prefix)

1 $\alpha(t)_u^{u'}.E \sim \alpha(s)_u^{u'}.(E\{s/t\})$ *where s is free for t in E*

2 $\alpha(t)_u^{u'}.E \sim (u')\delta$ *where* $u > u'$

3 $\alpha(t)_u^{u'}.E \sim \alpha(t)_u^{u'}.E + (u')\delta$

4 $\alpha(t)_u^v.E + \alpha(t)_v^{u'}.E \sim \alpha(t)_u^{u'}.E$ *where* $u \leq v \leq u'$

(1) shows that α-conversion preserves strong equivalence. From (2), we see that timing can also block actions. As an example, agent $a(t)_{10}^5\, nil$ cannot perform action a, but idles time 5 before it dies.

Proposition 4.3.2 (Laws for Summation)

1 $P + \delta \sim P$

2 $P + P \sim P$

3 $P + Q \sim Q + P$

4 $P + (Q + R) \sim (P + Q) + R$

Proposition 4.3.3 (Laws for Restriction)

1 $((u)\delta)\backslash a \;\sim\; (u)\delta$

2 $(\alpha(t)_u^{u'}.E)\backslash a \;\sim\; \alpha(t)_u^{u'}.(E\backslash a)$ *where* $\alpha \neq a \wedge \alpha \neq \bar{a}$

3 $(\alpha(t)_u^{u'}.E)\backslash a \;\sim\; (u')\delta$ *where* $\alpha = a \vee \alpha = \bar{a}$

4 $(P+Q)\backslash a \;\sim P\backslash a + Q\backslash a$

The restriction operator is distributive over summation. We will see that for any finite agent P which does not contain parallel and relabelling operators, by using the above laws, we can find an agent P', where P' does not contain the restriction operator (except δ), such that $P \;\sim\; P'$.

Proposition 4.3.4 (Laws for Relabelling)

1 $((u)\delta)[S] \;\sim\; (u)\delta$

2 $(\alpha(t)_u^{u'}.E)[S] \;\sim\; (S(\alpha))(t)_u^{u'}.(E[S])$

3 $(P+Q)[S] \;\sim\; P[S]+Q[S]$

The relabelling operator is also distributive over choice and we will see that for every finite agent P which does not contain the parallel operator, by using the above laws, we can find an agent P', where P' does not contain the relabelling operator, such that $P \;\sim\; P'$.

Proposition 4.3.5 (Laws for Recursion) $\mu X.E \sim E\{\mu X.E/X\}$

Until now, we have only defined strong equivalence over agents. However, we may also extend strong equivalence to processes along the standard lines.

Definition 4.3.6 *For any processes E and F which are closed with respect to process variables and contain at most free time variables $\vec{t} = (t_1, \ldots, t_n)$, we write $E \;\sim\; F$ if for any $\vec{u} \in T^n$ we have $E\{\vec{u}/\vec{t}\} \;\sim\; F\{\vec{u}/\vec{t}\}$.*

Definition 4.3.7 *For any processes E and F which contain at most free process variables $\vec{X} = (X_1, \ldots, X_n)$, we write $E \sim F$ if for any agents $\vec{P} = (P_1, \ldots, P_n)$, we have $E\{\vec{P}/\vec{X}\} \sim F\{\vec{P}/\vec{X}\}$.*

It is easy to check that all of the above laws still hold for processes. Moreover, we have the following expansion law. To simplify the presentation, we introduce a notion of normal form.

Definition 4.3.8 *A process $E \equiv \sum_{i\in I} \alpha_i(t_i)_{e_i}^{e'_i}. E_i + (e)\delta$ is in normal form if for any $i \in I$, $e'_i \le e$ and E_i is also in normal form.*

Clearly, for any normal form $E \equiv \sum_{i\in I} \alpha_i(t_i)_{e_i}^{e'_i}. E_i + (e)\delta$, we have $\left| E \right|_{\mathcal{T}} = e$.

Proposition 4.3.9 (Expansion Law) *Given normal forms*

$$E \equiv \sum_{i\in I} \alpha_i(t_i)_{e_i}^{e'_i}. E_i + (e)\delta$$

and

$$F \equiv \sum_{j\in J} \beta_j(s_j)_{f_j}^{f'_j}. F_j + (f)\delta$$

we have

$$E \mid F \sim$$

$$\sum_{i\in I} \alpha_i(r_i)_{e_i}^{min(e_i,f)}. (E_i\{r_i/t_i\} \mid r_i >> F)$$

$$+$$

$$\sum_{j\in J} \beta_j(r'_j)_{f_j}^{min(f'_j,e)}. (r'_j >> E \mid F_j\{r'_j/s_j\})$$

$$+$$

$$\sum_{\substack{\alpha_i=\bar{\beta}_j \\ i\in I \\ j\in J}} \tau(r_{ij})_{max(e_i,f_j)}^{min(e'_i,f'_j)}. (E_i\{r_{ij}/t_i\} \mid F_j\{r_{ij}/s_j\})$$

$+$

$(min(e,f))\delta$

where for any $i \in I$, $j \in J$, r_i, r'_j and r_{ij} are fresh time variables.

Proof: We only need to consider the case in which E and F are agents and show that a relation $\mathcal{R}$, where $\mathcal{R}$ is the least relation satisfying

(1) $\sim_{\mathcal{T}} \subseteq \mathcal{R}$; and

(2) for any normal forms $P \equiv \sum_{i\in I} \alpha_i(t_i)^{v'_i}_{v_i}. E_i + (v)\delta$ and $Q \equiv \sum_{j\in J} \beta_j(s_j)^{w'_j}_{w_j}. F_j + (w)\delta$, let G be the right hand side of expansion law for P and Q, then $(P \,|\, Q,\ G) \in \mathcal{R}$,

is a strong $\mathcal{T}$-bisimulation up to $\sim_{\mathcal{T}}$.

Suppose $(S, S') \in \mathcal{R}$. The case of $S \sim_{\mathcal{T}} S'$ is trivial. Now suppose $S \equiv P \,|\, Q$, where $P \equiv \sum_{i\in I} \alpha_i(t_i)^{v'_i}_{v_i}. E_i + (v)\delta$ and $Q \equiv \sum_{j\in J} \beta_j(s_j)^{w'_j}_{w_j}. F_j + (w)\delta$ are in normal form, and $S' \equiv G$, where G is the right hand side of expansion law for P and Q. Clearly $\mid P \,|\, Q \mid_{\mathcal{T}} = min(v, w) = \mid G \mid_{\mathcal{T}}$.

Case 1 Suppose $P \,|\, Q \xrightarrow{\alpha}_u R$

Case 1.1 $P \xrightarrow{\alpha}_u P'$, $Q \longrightarrow_u Q'$ and $R \equiv P' \,|\, Q'$, then for some $i \in I$ and $j \in J$, $\alpha = \alpha_i$, $v_i \le u \le min(v'_i, w)$, $P' \equiv E_i\{u/t_i\}$ and $Q' \sim u >> Q$. So (by congruence property, which is demonstrated next) $R \sim E_i\{u/t_i\} \,|\, u >> Q$. Clearly $G \xrightarrow{\alpha}_u (E_i\{r_i/t_i\} \,|\, r_i >> Q)\{u/r_i\}$ and $E_i\{u/t_i\} \,|\, u >> Q \sim (E_i\{r_i/t_i\} \,|\, r_i >> Q)\{u/r_i\}$.

Case 1.2 $P \longrightarrow_u P'$, $Q \xrightarrow{\alpha}_u Q'$ and $R \equiv P' \,|\, Q'$. Similar to case 1.1.

Case 1.3 $\alpha = \tau$, $P \xrightarrow{a}_u P'$, $Q \xrightarrow{\bar{a}}_u Q'$ and $R \equiv P' \,|\, Q'$ for some a. Then $\alpha_i = a$, $\beta_j = \bar{a}$, $w_j \le u \le min(v'_i, w'_j)$, $P' \equiv E_i\{u/t_i\}$ and $Q' \equiv F_j\{u/s_j\}$ for some $i \in I$, $j \in J$. Clearly, $G \xrightarrow{\tau}_u (E_i\{r_{ij}/t_i\} \,|\, F_j\{r_{ij}/s_j\})\{u/r_{ij}\}$ and $E_i\{u/t_i\} \,|\, F_j\{u/s_j\} \sim (E_i\{r_{ij}/t_i\} \,|\, F_j\{r_{ij}/s_j\})\{u/r_{ij}\}$.

Case 2 Suppose $G \xrightarrow{\alpha}_u R$, similar to case 1. □

4.4 Strong Congruence

Strong equivalence is a congruence relation, i.e. it is substitutive under all operators including the recursive operator. This is shown by the following two propositions.

Proposition 4.4.1 *For any processes E and F, $E \sim F$ implies*

(1) $\alpha(t)_e^{e'}.E \sim \alpha(t)_e^{e'}.F$	(4) $E \backslash a \sim F \backslash a$
(2) $E + G \sim F + G$	(5) $E[S] \sim F[S]$
(3) $E \mid G \sim F \mid G$	

Proof: All these properties can be proved by exhibiting appropriate strong bisimulations. For example, for (3), it is enough to consider agents and show that

$$S = \{(P \mid R, Q \mid R) : P \sim Q\}$$

is a strong bisimulation. □

In the sequel, we use $\vec{E} \sim \vec{F}$ to represent componentwise strong equivalences between $\vec{E}$ and $\vec{F}$.

Proposition 4.4.2 *For any processes $\vec{E}$ and $\vec{F}$ which contain at most free process variables $\vec{X}$, $\vec{E} \sim \vec{F}$ implies $\mu\vec{X}.\vec{E} \sim \mu\vec{X}.\vec{F}$.*

Proof: Analogous to Proposition 4.4.12 of [Mil89a]. □

For any finite process E, by using the laws for restriction, the laws for relabelling and the expansion law, we can find a process E', where E' does not contain the restriction operator (except δ), relabelling operator and parallel operator, such that $E \sim E'$.

4.5 Unique Solution of Equations

In this section, we focus on the solutions of the equations $\vec{X} = \vec{E}$. We will show that under certain conditions, the equations $\vec{X} = \vec{E}$ have an unique solution up to strong equivalence.

Definition 4.5.1 *We say the equations $\vec{X} = \vec{E}$ are weakly guarded if all $\vec{X}$ are weakly guarded in $\vec{E}$.*

The most important property of a weakly guarded process is that its first possible actions are completely independent of whatever we may substitute for its process variables.

Lemma 4.5.2 *For any process E and agents $\vec{P}$, if process variables $\vec{X}$ are weakly guarded in E and $E\{\vec{P}/\vec{X}\}\xrightarrow{\alpha}_u P'$ (or $E\{\vec{P}/\vec{X}\}\longrightarrow_u P'$) , then P' has the form $E'\{\vec{P}/\vec{X}\}$ for some process E'. Moreover, for any agents $\vec{Q}$, $E\{\vec{Q}/\vec{X}\}\xrightarrow{\alpha}_u E'\{\vec{Q}/\vec{X}\}$ (or $E\{\vec{Q}/\vec{X}\}\longrightarrow_u E'\{\vec{Q}/\vec{X}\}$).*

Proof: Analogous to Lemma 4.5.13 of [Mil89a]. □

Lemma 4.5.3 *For any process E which contains at most free process variables $\vec{X}$ and every $X_i \in \vec{X}$ is weakly guarded in E, then for any agents $\vec{P}$ and $\vec{Q}$, $\mid E\{\vec{P}/\vec{X}\} \mid_{\mathcal{T}} = \mid E\{\vec{Q}/\vec{X}\} \mid_{\mathcal{T}}$.*

Proof: The result follows from Definition 3.2.1. □

For any weakly guarded equations $\vec{X} = \vec{E}$, if they have solutions, then the solutions are unique up to strong equivalence.

Proposition 4.5.4 *For any processes E_i $(i \in I)$ which are closed with respect to time variables and contain at most free process variables X_i $(i \in I)$, let all X_i $(i \in I)$ be weakly guarded in each E_j $(j \in I)$. Then*

$$\vec{P} \sim \vec{E}\{\vec{P}/\vec{X}\} \text{ and } \vec{Q} \sim \vec{E}\{\vec{Q}/\vec{X}\} \text{ implies } \vec{P} \sim \vec{Q}$$

Proof: Analogous to Proposition 4.5.14 of [Mil89a]. □

It is easy to see that any weakly guarded equations have solutions. Therefore any weakly guarded equations have an unique solution up to strong equivalence.

Corollary 4.5.5 (Unique Solution) *For any weakly guarded equations $\vec{X} = \vec{E}$, where $\vec{E}$ are closed with respect to time variables and contain at most free process variables $\vec{X}$, there is an unique solution $\vec{P}$ (up to strong equivalence) such that*

$$\vec{P} \sim \vec{E}\{\vec{P}/\vec{X}\}$$

4.6 Modal Characterization of $\sim$

In this section, we give an alternative characterization of strong equivalence based on a simple modal language. Our modal language is an extension of the well-known Hennessy-Milner Logic with time.

There are some works on introducing real-time explicitly to Hennessy-Milner Logic, possibly with recursion. Among others, we mention here the works of [CM92,HLW91,Mol90].

The formulas of the modal language are defined by the following BNF expression:

$$\psi = tt \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \langle\alpha,\ v\rangle\psi \mid \langle u\rangle\psi$$

where $\alpha \in Act$, $u, v \in \mathcal{T}$ or $v \equiv \infty$. Let $\mathcal{L}$ be the set of all formulas defined by the above BNF expression, ranged over by ψ, φ.

The intended meaning of $\langle \alpha, u \rangle \psi$ is that there is an action α which happens not later than time u and after α happens the formula ψ will hold. Note that the above defined language is not a standard extension of Hennessy-Milner logic with time. The modal formula $\langle \alpha, u \rangle \psi$ allows action al to occur at ot before time u. The formula $\langle u \rangle \psi$ means there is a possibility that after a delay of time u, ψ will hold.

The formulas of $\mathcal{L}$ are interpreted over transition systems of the form

$$(S, \{\xrightarrow{\alpha}_u \ : \ \alpha \in Act \ \& \ u \in \mathcal{T}\} \cup \{\longrightarrow_u \ : \ u \in \mathcal{T}\})$$

where $\alpha \in Act$, $u \in \mathcal{T}$. As the operational semantics of Timed CCS is defined by a transition system of the above form, we can interpret the formulas of $\mathcal{L}$ over processes of Timed CCS.

Definition 4.6.1 *We say a Timed CCS agent P satisfies a formula $\psi \in \mathcal{L}$ if $P \models \psi$, where $P \models \psi$ is inductively defined as follows:*

(1) $P \models tt$

(2) $P \models \neg\psi$ *if and only if* $\neg P \models \psi$

(3) $P \models \psi \wedge \varphi$ *if and only if* $P \models \psi \wedge P \models \varphi$

(4) $P \models \langle \alpha, u \rangle \psi$ *if and only if* $\exists v \in \mathcal{T} \exists P' (v \leq u \wedge P \xrightarrow{\alpha}_v P' \wedge P' \models \psi)$

(5) $P \models \langle u \rangle \psi$ *if and only if* $\exists P' (P \longrightarrow_u P' \wedge P' \models \psi)$

The important derived operators are the duals of $\langle \alpha,\ v \rangle$ and $\langle u \rangle$, denoted by $[\alpha,\ v]$ and $[v]$, respectively.

$$[\alpha,\ v]\psi \stackrel{def}{=} \neg\langle \alpha,\ v \rangle \neg\psi$$

$$[u]\psi \overset{def}{=} \neg\langle u\rangle\neg\psi$$

Informally the formula $[\alpha, u]\psi$ means that ψ holds after every action α which happens before or at time u. The formula $[u]\psi$ means that ψ holds after every delay of time u.

Further derived operators are:

$$ff \overset{def}{=} \neg tt$$

$$A \vee B \overset{def}{=} \neg(\neg A \wedge \neg B)$$

$$A \rightarrow B \overset{def}{=} \neg A \vee B$$

All these operators have the usual meanings.

Bisimilarity preserves modal properties of $\mathcal{L}$, i.e. bisimilar agents of Timed CCS enjoy the same modal properties of $\mathcal{L}$.

Theorem 4.6.2 *$P \sim Q$ implies that for any formula ψ of $\mathcal{L}$, $P \models \psi$ if and only if $Q \models \psi$.*

Proof: Suppose $P \sim Q$. For any formula ψ of $\mathcal{L}$, we prove $P \models \psi$ if and only if $Q \models \psi$ by induction on the structure of ψ. We only show modalities, the other parts are trivial.

(1) $\psi \equiv \langle\alpha, u\rangle\varphi$, by symmetry we only need to show that $P \models \langle\alpha, u\rangle\varphi$ implies $Q \models \langle\alpha, u\rangle\varphi$. Suppose $P \models \langle\alpha, u\rangle\varphi$, then we have $P \xrightarrow{\alpha}_v P'$ for some P' and v such that $v \leq u$ and $P' \models \varphi$. Since $P \sim Q$, we have $Q \xrightarrow{\alpha}_v Q'$ for some Q' such that $P' \sim Q'$. By induction, $Q' \models \varphi$. So $Q \models \langle\alpha, u\rangle\varphi$.

(2) $\psi \equiv \langle u\rangle\varphi$, similarly we only need to show that $P \models \langle u\rangle\varphi$ implies $Q \models \langle u\rangle\varphi$. Suppose $P \models \langle u\rangle\varphi$, then $P \longrightarrow_u P'$ for some P' such that $P' \models \varphi$. Since $P \sim Q$, we have $Q \longrightarrow_u Q'$ for some Q' such that $P' \sim Q'$. By induction, $Q' \models \varphi$. So $Q \models \langle u\rangle\varphi$. □

The converse of the proposition also holds for those processes which are called image-finite. We first introduce the notion of image-finite processes.

Definition 4.6.3 *We say an agent P is image-finite if for every action $\alpha \in Act$ and $u \in \mathcal{T}$,*

$$\mathcal{P}_\alpha = \{P' : P \xrightarrow{\alpha}_u P'\}$$

is finite and every $P' \in \mathcal{P}_\alpha$ is also image-finite.

Theorem 4.6.4 *For any agents P and Q, if P or Q is image-finite and for any formula ψ, $P \models \psi$ if and only if $Q \models \psi$, then $P \sim Q$.*

Proof: We write $P =_{\mathcal{L}} Q$ to represent that for every formula ψ of $\mathcal{L}$, $P \models \psi$ if and only if $Q \models \psi$. We only need to show that

$$\mathcal{S} = \{(P, Q) \,:\, P =_{\mathcal{L}} Q \,\&\, P \textit{ or } Q \textit{ is image} - \textit{finite}\}$$

is a strong bisimulation.

Suppose $(P, Q) \in \mathcal{S}$. We first assume Q is image-finite.

(1) Suppose $P \xrightarrow{\alpha}_u P'$. We further assume that $Q \xrightarrow{\alpha}_u Q'$ implies $(P', Q') \notin \mathcal{S}$. Let $(Q_1, \cdots, Q_n)$ is an enumeration of $\{Q' \,:\, Q \xrightarrow{\alpha}_u Q'\}$, where $Q_1, \cdots, Q_n$ are image-finite. So $P' \neq_{\mathcal{L}} Q_i$ for all $i = 1, \cdots, n$. Therefore, there are formulas $\psi_1, \cdots, \psi_n$ such that $P' \models \psi_i$ but $Q_i \not\models \psi_i$. Let $\psi = \langle u \rangle \langle \alpha, 0 \rangle (\psi_1 \wedge \cdots \wedge \psi_n)$. Then $P \models \psi$ but $Q \not\models \psi$, a contradiction. Hence there is a Q' such that $Q \xrightarrow{\alpha}_u Q'$ and $(P', Q') \in \mathcal{S}$.

(2) The case of $P \longrightarrow_u P'$ is similar to (1).

Now assume P is image-finite. Suppose $P \xrightarrow{\alpha}_u P'$, where P' is image-finite. By the above, if $Q \xrightarrow{\alpha}_u Q''$, then $P \xrightarrow{\alpha}_u P''$ and $(P'', Q'') \in \mathcal{S}$ for some P''. Let $\{Q_i : i \in I\} = \{Q' \,:\, Q \xrightarrow{\alpha}_u Q'\}$ and $\{P_1, \cdots, P_n\} = \{P'' \,:\, P \xrightarrow{\alpha}_u P'' \,\&\, (P'', Q_i) \in \mathcal{S} \,\&\, i \in I\}$. Suppose $(P', Q_i) \notin \mathcal{S}$ for all i of I. Then $(P', P_j) \notin \mathcal{S}$, where $j = 1, \cdots, n$. So there are formulas $\psi_1, \cdots, \psi_n$ such that $P' \models \psi_i$ but $P_i \not\models \psi_i$ for all $i = 1, \cdots, n$. Let $\psi = \langle u \rangle \langle \alpha, 0 \rangle (\psi_1 \wedge \cdots \wedge \psi_n)$. Then $P \models \psi$ but $Q \not\models \psi$, a contradiction. Therefore there is a Q' such that $Q \xrightarrow{\alpha}_u Q'$ and $(P', Q') \in \mathcal{S}$. The case of $P \longrightarrow_u P'$ is similar.

By symmetry, we have done. $\square$

The above two theorems together are known as the modal characterization of bisimulation, which not only show that bisimulation is a natural equivalence, but also suggest that modal logic is an appropriate program logic for real-time systems.

The formulas of $\mathcal{L}$ can also be used to describe properties of real-time systems. As an example, we consider a vending machine which, after accepting coins, waits for a user to press a choice button for tea or coffee. If the user does not press the choice button within 5 minutes, the machine will refund the coins after another delay of Δ ($\Delta < 1$ second) time and then return to its initial state waiting to serve other users. If the user presses the choice button at a right time (i.e. within 5 minutes), he can collect his tea or coffee after another Δ delay. After the user collects the drink, the machine return to its initial state. We can define the vending machine in Timed CCS as follows:

$$VM \stackrel{def}{=} coin(t)_0^{\infty}. (choice(t)_0^5\, collect(t)_{\Delta}^{\infty}. VM + refund(t)_{5+\Delta}^{5+\Delta}. VM)$$

Now we can express many interesting temporal properties of the vending machine as formulas of $\mathcal{L}$ as follows:

(1) the machine can receive coins at any time.

$$VM \models \langle coin, \infty \rangle tt$$

(2) after inserting some coin, the user can make a choice within 5 minutes.

$$VM \models [coin, \infty] \langle choice, 5 \rangle tt$$

(3) after the coins are inserted and a choice is made at a right time, the user can always get some drink and the coins will never be refunded.

$$VM \models [coin, \infty][choice, 5](\langle collect, \infty\rangle tt \;\wedge\; [refund, \infty]ff)$$

(4) after the coins are inserted and $5 + \Delta$ minutes passes, the choice is timed out and the coins are refunded immediately.

$$VM \models [coin, \infty][5 + \Delta]([choice, \infty]ff \;\wedge\; \langle refund, 0\rangle tt)$$

Chapter 5

Decidability

A variety of equivalences have been proposed in the field of process theory in order to capture the behavioural aspects of processes. Various criteria exist for comparing the merits and deficiencies of these equivalence. One of the most important criteria is decidability. For CCS, every finite process has only finite derivatives and therefore the decidability of the strong equivalence (for finite processes) is straightforward. However for real-time calculi, even simple finite processes may have infinite derivatives. In this chapter, we show that strong equivalence is still decidable over finite processes of real-time calculi. The decidability is independent of the choice of time domain, allowing time to be discrete or dense. In the rest of this chapter, we focus on finite processes, i.e. processes without recursion.

There are other works on decidability for real-time systems. In [HLW91], it is showed that strong bisimulations over regular processes without time variables is decidable. It also discusses model checking for a timed extension of Hennessy-Milner Logic. The work of [Cer92] generalises that of [HLW91] to also include parallel composition. It also discusses the decidability property for weak bisimulation. All these works are for real-time processes without time variables. In [AD90], it is showed that for timed B*ü*chi automata the language emptiness problem is decidable but language inclusion problem is not. In [ACD90], a model checking algorithm for a real-time extension of branching time temporal logic is proposed. In [CM92], a tableau system for model checking of a real-time extension of Hennessy-Milner Logic with recursion is presented.

5.1 Characteristic Formulas

For any processes E and F, we construct a formula $WC(E,F)$, the characteristic formula of E and F, of the first-order theory of some structure such that $E \sim F$ if and only if $WC(E,F)$.

Definition 5.1.1 *For any processes E and F, which contain at most free time variables $\vec{t} = (t_1, \ldots, t_n)$, and a formula A, where $fv(A) \subseteq \{t_1, \ldots, t_n\}$, we say A is a condition for E to be bisimilar to F, written as $A \models E \sim F$, if for any $\vec{u} \in \mathcal{T}^n$, $A\{\vec{u}/\vec{t}\}$ implies $E\{\vec{u}/\vec{t}\} \sim F\{\vec{u}/\vec{t}\}$.*

Definition 5.1.2 *A formula A is said to be a weakest condition for E to be bisimilar to F if $A \models E \sim F$ and for any formula B, $B \models E \sim F$ implies $B \rightarrow A$ holds.*

Proposition 5.1.3 *If A is a weakest condition for E to be bisimilar to F, then*

$$B \models P \sim Q \quad if\ and\ only\ if \quad B \rightarrow A$$

Proof: Straightforward. $\square$

Corollary 5.1.4 *If A and B are weakest conditions for E to be bisimilar to F, then $A \leftrightarrow B$ holds.*

We identify formulas which are logically equivalent. By the above corollary, we have that if weakest conditions for E to be bisimilar to F exist, then they are unique up to this equality.

Now we show that for all processes E and F, the weakest condition for E to be bisimilar to F exists. We first consider the weakest condition $WC(E, F)$ for E to be bisimilar to F, where E and F are both in normal form.

Definition 5.1.5 *For any normal forms E and F, where*

$$E \equiv \sum_{i \in I} \alpha_i(t_i)_{e_i}^{e_i'}.\, E_i + (e)\delta$$

and

$$F \equiv \sum_{j \in J} \beta_j(s_j)_{f_j}^{f_j'}.\, F_j + (f)\delta$$

let

$$WC(E,F) \stackrel{def}{=} (e = f) \wedge$$

$$\bigwedge_{i \in I}(\forall t(e_i \le t \le e_i' \rightarrow \bigvee_{\substack{\alpha_i = \beta_j \\ j \in J}} (f_j \le t \le f_j' \wedge WC(E_i\{t/t_i\}, F_j\{t/s_j\})))) \wedge$$

$$\bigwedge_{j \in J}(\forall s(f_j \le s \le f_j' \rightarrow \bigvee_{\substack{\alpha_i = \beta_j \\ i \in I}} (e_i \le s \le e_i' \wedge WC(E_i\{s/t_i\}, F_j\{s/s_j\}))))$$

We will show that $WC(E, F)$ is the weakest condition for E to be bisimilar to F, where E and F are both in normal form. We first show that $WC(E, F)$ is a condition for E to be bisimilar to F.

Proposition 5.1.6 *For any normal forms E and F, $WC(E, F) \models E \sim F$.*

Proof: For normal forms $E \equiv \sum_{i \in I} \alpha_i(t_i)_{e_i}^{e_i'}.\, E_i + (e)\delta$ and $F \equiv \sum_{j \in J} \beta_j(s_j)_{f_j}^{f_j'}.\, F_j + (f)\delta$, suppose E and F contain at most free time variables $\vec{r} = (r_1, \ldots, r_n)$. We proceed by induction on the maximum depth of E and F, where the depth $d(E)$ of E is inductively defined as follows:

$$d((e)\delta) = 0 \qquad d(\alpha(t)_e^{e'}.\, E) = 1 + d(E)$$
$$d(E + F) = max(d(E), d(F))$$

For any $\vec{u} = (u_1, \ldots, u_n) \in \mathcal{T}^n$, we suppose $WC(E, F)\{\vec{u}/\vec{r}\}$ holds.

Firstly, $E\{\vec{u}/\vec{r}\} \xrightarrow{\alpha}_v P'$, then there is $i \in I$ such that $\alpha = \alpha_i$, $e_i\{\vec{u}/\vec{r}\} \leq v \leq e'_i\{\vec{u}/\vec{r}\}$ and $P' \equiv E_i\{\vec{u}/\vec{r}\}\{v/t_i\}$. By definition of $WC(E, F)$, there is $j \in J$, such that $\beta_j = \alpha_i$, $f_j\{\vec{u}/\vec{r}\} \leq v \leq f'_j\{\vec{u}/\vec{r}\}$ and $WC(E_i\{\vec{u}/\vec{r}\}\{v/t_i\}, F_j\{\vec{u}/\vec{r}\}\{v/s_j\})$ holds. By induction, we have $E_i\{\vec{u}/\vec{r}\}\{v/t_i\} \sim F_j\{\vec{u}/\vec{r}\}\{v/s_j\})$ hold. Clearly $F\{\vec{u}/\vec{r}\} \xrightarrow{\beta_j}_v F_j\{\vec{u}/\vec{r}\}\{v/s_j\}$.

Similarly, $F\{\vec{u}/\vec{r}\} \xrightarrow{\alpha}_v Q'$ implies $E\{\vec{u}/\vec{t}\} \xrightarrow{\alpha}_v P'$ with $P' \sim Q'$ for some P'.

Finally, $e\{\vec{u}/\vec{r}\} = f\{\vec{u}/\vec{r}\}$. Therefore $E\{\vec{u}/\vec{r}\} \sim F\{\vec{u}/\vec{r}\}$. □

Now we show that $WC(E, F)$, where E and F are both in normal form, is the weakest condition for E to be bisimilar to F. To do so we first prove the following lemma.

Lemma 5.1.7 *For any normal forms E and F, $E \sim F$ implies $WC(E, F)$.*

Proof: We only need to consider the case where both E and F are agents. Let $E \equiv \sum_{i \in I} \alpha_i(t_i)_{v_i}^{v'_i}.E_i + (v)\delta$ and $F \equiv \sum_{j \in J} \beta_j(s_j)_{w_j}^{w'_j}.F_j + (w)\delta$ be normal forms. Suppose $E \sim F$. We proceed by induction on the maximum depth m of E and F.

Case $m = 0$. Then we have $E \equiv (v)\delta$ and $F \equiv (w)\delta$. Clearly $E \sim F$ implies $v = w$.

Case $m = m' + 1$.

(1) For any $i \in I$, if $v_i \leq u \leq v'_i$, then we have $E \xrightarrow{\alpha_i}_u E_i\{u/t_i\}$. So there is a $j \in J$ such that $F \xrightarrow{\beta_j}_u F_j\{u/s_j\}$ and $F_j\{u/s_j\} \sim E_i\{u/t_i\}$, where $\alpha_i = \beta_j$ and $w_j \leq u \leq w'_j$. By induction, $WC(F_j\{u/s_j\}, E_i\{u/t_i\})$ holds.

(2) Similarly, for any $j \in J$, $w_j \leq u \leq w'_j$ implies that there is a $i \in I$ such that $v_i \leq u \leq v'_i$ and $WC(F_j\{u/s_j\}, E_i\{u/t_i\})$ hold.

(3) $v = w$.

Therefore $WC(E, F)$ holds $\square$

Proposition 5.1.8 *For normal forms E, F and formula A, $A \models E \sim F$ implies $A \to WC(E, F)$ holds.*

Proof: For any normal forms $E \equiv \sum_{i\in I} \alpha_i(t_i)^{e_i'}_{e_i}. E_i + (e)\delta$ and $F \equiv \sum_{j\in J} \beta_j(s_j)^{f_j'}_{f_j}. F_j + (f)\delta$, suppose E and F contain at most time variables $\vec{r} = (r_1, \ldots, r_n)$ and $A \models E \sim F$. For any $\vec{u} = (u_1, \ldots, u_n) \in \mathcal{T}^n$, if $A\{\vec{u}/\vec{r}\}$ holds, then $E\{\vec{u}/\vec{t}\} \sim F\{\vec{u}/\vec{r}\}$. By Lemma 5.1.7, $WC(E\{\vec{u}/\vec{r}\}, F\{\vec{u}/\vec{r}\})$ holds. Hence $A \to WC(E, F)$ holds. $\square$

Corollary 5.1.9 *For normal forms E, E' and F, $E \sim E'$ implies that $WC(E, F)$ is logically equivalent to $WC(E', F)$.*

Now we consider the weakest condition for E to be bisimilar to F, for any processes E and F. We first show that for any process E, there is a normal form F such that $E \sim F$.

Lemma 5.1.10 *For any process E, there is a normal form F such that $E \sim F$.*

Proof: Since for every process E, there is a process E', where E' does not contain the restriction operator (except δ), relabelling operator and parallel operator, such that $E \sim E'$, we only need to show that for every such E' there is a normal form F such that $E' \sim F$. We proceed by induction on the structure of E'.

- $E' \equiv (e)\delta$, trivial.

- $E' \equiv \alpha(t)^{e'}_{e}. E''$. By induction, there is a normal form F'' such that $E'' \sim F''$. So we have $\alpha(t)^{e'}_{e}. E'' \sim \alpha(t)^{e'}_{e}. F'' + (e')\delta$.

– $E' \equiv E_1 + E_2$. By induction, there are normal forms $F_1 \equiv \sum_{i\in I} \alpha_i(t_i)^{e'_i}_{e_i}.F_{1i} + (e)\delta$ and $F_2 \equiv \sum_{j\in J} \beta_j(t_j)^{f'_j}_{f_j}.F_{2j} + (f)\delta$ such that $E_1 \sim F_1$ and $E_2 \sim F_2$. Clearly $F \equiv \sum_{i\in I} \alpha_i(t_i)^{e'_i}_{e_i}.F_{1i} + \sum_{j\in J} \beta_j(t_j)^{f'_j}_{f_j}.F_{2j} + (max(e,f))\delta$ is in normal form and $E' \sim F$. □

Definition 5.1.11 *For any processes E and F, let $WC(E,F)$ be $WC(E',F')$, where normal forms E' and F' satisfy $E \sim E'$ and $F \sim F'$.*

Clearly $WC(E,F)$ is well defined. We will show next that $WC(E,F)$ is the weakest condition for E to be bisimilar to F.

Proposition 5.1.12 *For any processes E and F, $WC(E,F)$ is the weakest condition for E to be bisimilar to F.*

Proof: We first show that $WC(E,F)$ is a condition for E to be bisimilar to F, i.e. $WC(E,F) \models E \sim F$ holds. Suppose E and F contain at most free time variables $\vec{t} = (t_1, \cdots, t_n)$. Let E' and F' be normal forms such that $E \sim E'$ and $F \sim F'$. For any $\vec{u} \in \mathcal{T}^n$, suppose $WC(E,F)\{\vec{u}/\vec{t}\}$ holds. Then $E'\{\vec{u}/\vec{t}\} \sim F'\{\vec{u}/\vec{t}\}$. So $E\{\vec{u}/\vec{t}\} \sim F\{\vec{u}/\vec{t}\}$ holds.

Now we suppose $A \models E \sim F$ holds. We further assume $A\{\vec{u}/\vec{t}\}$ holds, where $\vec{u} \in \mathcal{T}^n$. Then $E\{\vec{u}/\vec{t}\} \sim F\{\vec{u}/\vec{t}\}$. So $E'\{\vec{u}/\vec{t}\} \sim F'\{\vec{u}/\vec{t}\}$ and therefore $WC(E,F)\{\vec{u}/\vec{t}\}$. □

5.2 Decidability over Discrete Time Domains

In this section, we suppose time is discrete. Let $\mathcal{T}$ be $\mathbb{N}$, the set of natural numbers.

In the structure $(\mathbb{N}, +)$, we can define the following relations:

(1) Zero ($\{m : m = 0\}$): $\forall x(x + m = x)$

(2) Ordering ($\{(m,n) : m \leq n\}$): $\exists x(m + x = n)$

(3) Successor ($\{(m,n) : n = S(m)\}$):

$$\exists x \forall y(x \neq 0) \wedge (y = 0 \vee x \leq y) \wedge m + x = n$$

(4) Subtraction ($\{(l,m,n) : n = l \dot{-} m\}$): $(l < m \wedge n = 0) \vee (l \geq m \wedge l = m + n)$

(5) Max ($\{(l,m,n) : n = max(l,m)\}$): $(l \geq m \wedge n = l) \vee (l < m \wedge n = m)$

(6) Min ($\{(l,m,n) : n = min(l,m)\}$): $(l \geq m \wedge n = m) \vee (l < m \wedge n = l)$

For any time expression e over time domain $(\mathcal{T} \cup \{\infty\}, \leq)$, we have $e < \infty$. We can read the terms of the form $e \leq \infty$ as $0 = 0$ throughout $WC(E,F)$. As a result, $WC(E,F)$ is a sentence of the first-order theory over structure $(\mathbb{N}, +)$.

Theorem 5.2.1 $\sim$ *is decidable for agents.*

Proof: The result follows from the decidability of the first-order theory of the natural numbers with $+$ [End72]. □

Lemma 5.2.2 *For any processes E and F which contain at most free time variables $\vec{t} = (t_1, \ldots, t_n)$, we have*

$$E \sim F \textit{ if and only if } \forall t_1 \ldots \forall t_n WC(E,F)$$

Proof: The result follows from the definition of strong equivalence over processes. □

Lemma 5.2.3 *For any processes E and F, where E and F contain at most free time variables $t_1, \ldots, t_n$, the formula $\forall t_1 \ldots \forall t_n WC(E,F)$ is a sentence of the first-order theory over structure $(\mathbb{N}, +)$.*

Proof: Straightforward. □

Theorem 5.2.4 $\sim$ *is decidable.*

Proof: Analogous to Theorem 5.2.1. $\square$

Since the first-order theory of the natural numbers with $+$ and $\times$, i.e. $\mathrm{Th}(\mathbb{N}, +, \times)$, is undecidable, this justifies our decisions in the choice of time expressions.

5.3 Decidability over Dense Time Domains

In this section, we assume time to be dense. Let $\mathcal{T}$ be $\mathbb{R}^{\geq 0}$, the set of non-negative real numbers.

As for the structure $(\mathbb{N}, +)$, we can also define relations: Zero, Ordering, Subtraction, Max and Min in the structure $(\mathbb{R}, +, \times)$.

Lemma 5.3.1 *For any processes E and F, where E and F contain at most free time variables $t_1, \ldots, t_n$, the formula $\forall t_1 \ldots \forall t_n WC(E, F)$ is a sentence of the first-order theory over structure $(\mathbb{R}, +, \times)$.*

Proof: Straightforward. $\square$

Theorem 5.3.2 $\sim$ *is decidable.*

Proof: The result follows from the decidability of the first-order theory of the reals with $+$ and $\times$ [Col75].

Chapter 6

A Proof System

In the last chapter we have shown that for any finite processes E and F, we have a characteristic formula $WC(E,F)$ such that $E \sim F$ if and only if $WC(E,F)$ holds. We used the notation $A \models E \sim F$ to indicate that a formula A is a condition for process E to be bisimilar to process F. In this section, we study proof systems for strong equivalence.

There is a simple proof system for strong equivalence pointed out by Kim Larsen which consists of a single rule

$$\frac{WC(E,F)}{E = F}$$

The proof system is independent of the choice of time domain, allowing time to be discrete or dense. It is sound and complete for finite processes over discrete time as well as dense time domains.

In this chapter we describe a relativised compositional proof system in which we derive statements of the form $A \vdash E = F$. The proof system is sound. It is complete for finite processes over dense time domains, but only complete for a restricted language over discrete time domains. We will give some examples to show how the proof system works for finite processes as well as recursively defined processes.

6.1 The Proof System

In the rest of this chapter, we will mainly focus on finite processes (except Section 6.4 in which we will discuss how the proof system works for recursively defined processes). By Lemma 5.1.10, for every finite process E, we have a normal form F such that $E \sim F$. However, every finite normal form can be defined by the following BNF expressions:

$$E ::= \delta \,\Big|\, nil \,\Big|\, (e)E \,\Big|\, \alpha(t)_e^{e'}.\,E \,\Big|\, E + F$$

where e is a time expression and e' is a time expression or ∞.

Henceforth we will focus on the sublanguage defined by the above syntax except Section 6.4. We first present the proof system. Table 6–1 contains all axioms and Table 6–2 contains all proof rules. The proof rules are in the form

$$\frac{S_1 \ldots S_n}{S}$$

where $S_1, \ldots, S_n, S$ are statements. The rule can be read as: if all premises S_1, $\ldots$, S_n can be derived, then the conclusion S can be deduced.

We say $A \vdash X = Y$ is derivable if it can be derived from the axioms in Table 6–1 by using the proof rules in Table 6–2. For convenience, we write $A \vdash X = Y$ to assert that $A \vdash X = Y$ is derivable.

Lemma 6.1.1 *The following are derivable:*

(1) $e \le f \wedge f \le f' \wedge f' \le e' \vdash \alpha(t)_f^{f'}.\,E + \alpha(t)_e^{e'}.\,E = \alpha(t)_e^{e'}.\,E$

(2) $e \le f \wedge f \le e' \wedge e' \le f' \vdash \alpha(t)_f^{f'}.\,E + \alpha(t)_e^{e'}.\,E = \alpha(t)_e^{f'}.\,E$

Proof: We only consider (1), as (2) is similar. For (1), we have

$$e \le f \le e' \vdash \alpha(t)_e^{e'}.\,E = \alpha(t)_e^{f}.\,E + \alpha(t)_f^{e'}.\,E$$

$\text{false} \vdash X = Y$	$\text{true} \vdash X = X$
$\text{true} \vdash X = X + \delta$	$\text{true} \vdash X = X + X$
$\text{true} \vdash X + Y = Y + X$	$\text{true} \vdash X + (Y + Z) = (X + Y) + Z$
$\text{true} \vdash X = (0)X$	$\text{true} \vdash (e)(e')\delta = (e + e')\delta$
$\text{true} \vdash (e)(X + Y) = (e)X + (e)Y$	$\text{true} \vdash \alpha(t)_e^{e'}.X = \alpha(t)_e^{e'}.X + (e')\delta$
$e' < e \vdash \alpha(t)_e^{e'}.X = (e')\delta$	
$e \leq s \leq e' \vdash \alpha(t)_e^{e'}.X = \alpha(t)_e^{s}.X + \alpha(t)_s^{e'}.X$	
$\text{true} \vdash \alpha(t)_e^{e'}.X = \alpha(s)_e^{e'}.X\{s/t\}$ (s is free for t in X)	
$\text{true} \vdash (e)(\alpha(t)_f^{f'}.X) = \alpha(t)_{f+e}^{f'+e}.X\{t-e/t\}$ $(t \notin fv(e))$	

Table 6–1: Axioms

1 $$\frac{A \vdash X = Y \qquad A \vdash Y = Z}{A \vdash X = Z}$$

2 $$\frac{A \vdash X = Y \qquad B \vdash X = Y}{A \vee B \vdash X = Y}$$

3 $$\frac{B \vdash X = Y}{A \vdash X = Y} \qquad (A \rightarrow B)$$

4 $$\frac{A \vdash X = Y}{A \wedge e = f \vdash (e)X = (f)Y}$$

5 $$\frac{A \wedge e \leq t \leq e' \vdash X = Y}{A \wedge e = f \wedge e' = f' \vdash \alpha(t)_e^{e'}.\,X = \alpha(t)_f^{f'}.\,Y} \qquad (t \notin fv(A))$$

6 $$\frac{A \vdash X = Y \qquad A \vdash X' = Y'}{A \vdash X + X' = Y + Y'}$$

7 $$\frac{A \vdash X = Y \qquad B \vdash X = Z}{A \vee B \vdash X + Y + Z = Y + Z}$$

Table 6–2: Proof Rules

$$f \leq f' \leq e' \vdash \alpha(t)^{e'}_{f}.E = \alpha(t)^{f'}_{f}.E + \alpha(t)^{e'}_{f'}.E$$

Clearly

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \rightarrow e \leq f \leq e'$$

and

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \rightarrow f \leq f' \leq e'$$

So

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \vdash \alpha(t)^{e'}_{e}.E = \alpha(t)^{f}_{e}.E + \alpha(t)^{e'}_{f}.E$$

and

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \vdash \alpha(t)^{e'}_{f}.E = \alpha(t)^{f'}_{f}.E + \alpha(t)^{e'}_{f'}.E$$

Thus

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \vdash \alpha(t)^{e'}_{f}.E = \alpha(t)^{e'}_{f}.E + \alpha(t)^{f'}_{f}.E$$

and

$$e \leq f \wedge f \leq f' \wedge f' \leq e' \vdash \alpha(t)^{e'}_{e}.E = \alpha(t)^{e'}_{e}.E + \alpha(t)^{f'}_{f}.E \qquad \square$$

Lemma 6.1.2

(1) *If* $A \vdash X = Y$ *and* $t \notin fv(A)$, *then* $A \wedge (e \leq t \leq e') \vdash \alpha(t)^{e'}_{e}.X = \alpha(t)^{e'}_{e}.Y$.

(2) *If* $A \vdash X = Y$ *and* $t \notin fv(A)$, *then* $A \wedge e = f \wedge e' = f' \vdash \alpha(t)^{e'}_{e}.X = \alpha(t)^{f'}_{f}.Y$.

Proof: We only consider (2), as (1) is similar.

Suppose $A \vdash X = Y$ and $t \notin fv(A)$. Since $A \wedge e \leq t \leq e' \rightarrow A$, we have $A \wedge e \leq t \leq e' \vdash X = Y$. Thus $A \wedge e = f \wedge e' = f' \vdash \alpha(t)^{e'}_{e}.X = \alpha(t)^{f'}_{f}.Y$. $\square$

Remark Rules (1) and (2) of Lemma 6.1.2 are equipotent with the proof rule 5. In fact they were the version which I first proposed. $\square$

For convenience, we will write $\vdash E = F$ in place of $true \vdash E = F$.

Now we consider a simple example and show how the proof system works. In Section 6.4, we will consider a more interesting example and show how the proof system works for recursively defined processes.

Example Let $E \equiv b(s)_0^{t-(2\dot{-}t)}.\,\delta$, $F \equiv b(s)_0^{2t-2}.\,\delta$ and $G \equiv b(s)_0^{t}.\,\delta$. We show that

$$\vdash a(t)_1^{10}.\,(E+F+G) = a(t)_1^{10}.\,(F+G)$$

is derivable.

$$\frac{\vdash \delta = \delta}{\dfrac{t-(2\dot{-}t) = 2t-2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,\delta = b(s)_0^{2t-2}.\,\delta}{t \leq 2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,\delta = b(s)_0^{2t-2}.\,\delta} \qquad t \leq 2 \rightarrow (t-(2\dot{-}t) = 2t-2)}$$

and

$$\frac{\vdash \delta = \delta}{\dfrac{t-(2\dot{-}t) = t \vdash b(s)_0^{t-(2\dot{-}t)}.\,\delta = b(s)_0^{t}.\,\delta}{t > 2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,\delta = b(s)_0^{t}.\,\delta} \qquad t > 2 \rightarrow (t-(2\dot{-}t) = t)}$$

By rule 7

$$t \leq 2 \vee t > 2 \vdash E+F+G = F+G$$

and

$$\frac{\dfrac{t \leq 2 \vee t > 2 \vdash E+F+G = F+G}{1 \leq t \leq 10 \vdash E+F+G = F+G} \qquad 1 \leq t \leq 10 \rightarrow (t \leq 2 \vee t > 2)}{\vdash a(t)_1^{10}.\,(E+F+G) = a(t)_1^{10}.\,(F+G)}$$

6.2 Soundness

The soundness of the proof system is shown by the following proposition.

Proposition 6.2.1 (Soundness) *If $A \vdash E = F$, then $A \models E \sim F$.*

Proof: We only need to show that every instance of the axioms of Table 6–1 is valid, i.e. if $A \vdash E = F$ is an instance of an axiom, then $A \models E \sim F$, and every proof rule of Table 6–2 preserves the validity. The validity of every instance of the axioms follows from the properties of strong bisimulation (see Chapter 4 for details).

The proofs of rules (1), (2), (3) and (4) are straightforward.

For (5), we suppose $A \wedge e \leq t \leq e' \models X \sim Y$ where the formula A and the processes $\alpha(t)_e^{e'}.X$, $\alpha(t)_f^{f'}.Y$ contain at most free time variables $\vec{s} = (s_1, \cdots, s_n)$. For any $\vec{v} \in \mathcal{T}^n$, we also suppose $A\{\vec{v}/\vec{s}\}$ holds, $e\{\vec{v}/\vec{s}\} = f\{\vec{v}/\vec{s}\}$ and $e'\{\vec{v}/\vec{s}\} = f'\{\vec{v}/\vec{s}\}$. If $(\alpha(t)_e^{e'}.X)\{\vec{v}/\vec{s}\} \xrightarrow{\beta}_u P'$, then $\beta = \alpha$, and $P' \equiv X\{\vec{v}/\vec{s}\}\{u/t\}$ where $e\{\vec{v}/\vec{s}\} \leq u \leq e'\{\vec{v}/\vec{s}\}$. Clearly $(\alpha(t)_f^{f'}.Y)\{\vec{v}/\vec{s}\} \xrightarrow{\alpha}_u Y\{\vec{v}/\vec{s}\}\{u/t\}$ and $X\{\vec{v}/\vec{s}\}\{u/t\} \sim Y\{\vec{v}/\vec{s}\}\{u/t\}$ (note that $t \notin fv(A)$). By symmetry, we finish the proof of rule (5).

For (6), we suppose $A \models X \sim Y$ and $A \models X' \sim Y'$. Let the formula A, the processes X, X', Y and Y' contain at most free time variables $\vec{s} = (s_1, \cdots, s_n)$. For any $\vec{v} \in \mathcal{T}^n$, we also suppose $A\{\vec{v}/\vec{s}\}$ holds. So we have $X\{\vec{v}/\vec{s}\} \sim Y\{\vec{v}/\vec{s}\}$ and $X'\{\vec{v}/\vec{s}\} \sim Y'\{\vec{v}/\vec{s}\}$. Therefore $(X + X')\{\vec{v}/\vec{s}\} \sim (Y + Y')\{\vec{v}/\vec{s}\}$.

The proof of rule (7) is similar to that of rule (6). $\square$

Note that the side condition of rule 5 is important, as otherwise the rule is invalid. As an example, let the formula A be $1 \leq t \leq 5$, the processes X and Y be $b(s)_0^{10-t}.nil$ and $b(s)_0^t.nil$, respectively. Clearly we have $A \wedge 5 \leq t \leq 10 \models X \sim Y$, but $A \not\models a(t)_5^{10}.X \sim a(t)_5^{10}.Y$.

6.3 Completeness

The completeness result of the proof system depends heavily on the existence of normal forms.

Lemma 6.3.1 *For any process E, there is a normal form E' such that $\vdash E = E'$.*

Proof: We proceed by induction on the structure of E.

Case 1 $E \equiv \delta$. Clearly $\vdash \delta = (0)\delta$.

Case 2 $E \equiv nil$. Clearly $\vdash (\infty)\delta = (\infty)\delta$.

Case 3 $E \equiv (f)F$. By induction, there is a normal form $\sum_{i\in I} \alpha_i(t_i)_{e_i}^{e_i'}. E_i + (e)\delta$ such that

$$\vdash F = \sum_{i\in I} \alpha_i(t_i)_{e_i}^{e_i'}. E_i + (e)\delta$$

So

$$\vdash (f)F = (f)(\sum_{i\in I} \alpha_i(t_i)_{e_i}^{e_i'}. F_i + (e)\delta)$$

Therefore

$$\vdash (f)F = \sum_{i\in I} \alpha_i(t_i)_{e_i+f}^{e_i'+f}. E_i\{t_i - f/t_i\} + (e+f)\delta$$

and $\sum_{i\in I} \alpha_i(t_i)_{e_i+f}^{e_i'+f}. E_i\{t_i - f/t_i\} + (e+f)\delta$ is in normal form.

Case 4 $E \equiv \alpha(t)_e^{e'}. F$. By induction, there is a normal form F' such that $\vdash F = F'$. So $\vdash \alpha(t)_e^{e'}. F = \alpha(t)_e^{e'}. F'$ and therefore $\vdash \alpha(t)_e^{e'}. F = a(t)_e^{e'}. F' + (e')\delta$ where $a(t)_e^{e'}. F' + (e')\delta$ is in normal form.

Case 5 $E \equiv E_1 + E_2$. By induction, there is a normal form $\sum_{i\in I} \alpha_i(t_i)_{e_i}^{e_i'}. E_i' + (e)\delta$ such that

$$\vdash E_1 = \sum_{i\in I} \alpha_i(t_i)_{e_i}^{e_i'}. E_i' + (e)\delta$$

and a normal form $\sum_{j\in J}\beta_j(s_j)^{f_j'}_{f_j}.E_j''+(f)\delta$ such that

$$\vdash E_2=\sum_{j\in J}\beta_j(s_j)^{f_j'}_{f_j}.E_j''+(f)\delta$$

So

$$\vdash E_1+E_2=\sum_{i\in I}\alpha_i(t_i)^{e_i'}_{e_i}.E_i'+\sum_{j\in J}\beta_j(s_j)^{f_j'}_{f_j}.E_j''+(max(e,f))\delta$$

where $\sum_{i\in I}\alpha_i(t_i)^{e_i'}_{e_i}.E_i'+\sum_{j\in J}\beta_j(s_j)^{f_j'}_{f_j}.E_j''+(max(e,f))\delta$ is in a normal form.

□

6.3.1 Completeness in Dense Time Domains

In this section, we assume time to be dense. For example, the time domain is $(\mathbb{Q}^{\geq 0}\cup\{\infty\},\leq)$ or $(\mathbb{R}^{\geq 0}\cup\{\infty\},\leq)$. By Lemma 6.3.1, we can further assume that every process is in normal form.

We first show that for any processes E and F, the weakest condition for E to be bisimilar to F can be written in disjunctive normal form.

Lemma 6.3.2 *For any processes E and F which contain at most free time variables $(t_1,\ldots,t_n)$, $WC(E,F)$ can be written in disjunctive normal form*

$$\bigvee_{i\in I} e_{1i}\leq t_1\leq f_{1i}\wedge\cdots\wedge e_{ni}(t_1,\ldots,t_{n-1})\leq t_n\leq f_{ni}(t_1,\ldots,t_{n-1})$$

for some finite I, where $e_{ki}(t_1,\cdots,t_{k-1})$ and $f_{ki}(t_1,\cdots,t_{k-1})$ $(k=1,\cdots,n)$ contain at most variables $t_1,\cdots,t_{k-1}$.

Proof: We proceed by induction on the maximal depth of E and F.

For the base case, we have $WC(E,F)\equiv(e=f)$. If $e=e'+ut$, where u is a constant (i.e. $u\in\mathcal{T}$), $t\notin fv(e')$ and $t\notin fv(f)$, then $e=f$ can be replaced by $(f\dot{-}e')u^{-1}\leq t\leq(f-e')u^{-1}$ which clearly can be written in the above form. If $e=e'\dot{-}u\times t$, where $u\in\mathcal{T}$, $t\notin fv(e')$ and $t\notin fv(f)$, then $e=f$ can be replaced by

$((u^{-1}\times e' \leq t \leq \infty)\wedge 0 = f)\vee((0 \leq t \leq u^{-1}\times e')\wedge u^{-1}\times(e'\dot{-}f) \leq t \leq u^{-1}\times(e'-f)$

which can also be written in the above form. All other cases are similar.

For the inductive case, we have $WC(E,F) \equiv (e = f) \wedge$

$$\bigwedge_{i\in I}(\forall t(e_i \leq t \leq e'_i \rightarrow \bigvee_{\substack{j\in J\\ \alpha_i=\beta_j}} (f_j \leq t \leq f'_j \wedge WC(E_i\{t/t_i\}, F_j\{t/s_j\}))))$$

$$\bigwedge_{j\in J}(\forall s(f_j \leq s \leq f'_j \rightarrow \bigvee_{\substack{i\in I\\ \alpha_i=\beta_j}} (e_i \leq s \leq e'_i \wedge WC(E_i\{s/t_i\}, F_j\{s/s_j\}))))$$

For any $i \in I$ and $j \in J$, by induction

$$WC(E_i\{t/t_i\}, F_j\{t/s_j\}) = \bigvee_{l\in L}(g_l \leq t \leq g'_l) \wedge D^l_{ij}$$

where for every $l \in L$, $t \notin fv(D^l_{ij})$ and L is finite, and

$$WC(E_i\{s/t_i\}, F_j\{s/s_j\}) = \bigvee_{k\in K}(h_k \leq s \leq h'_k) \wedge C^k_{ij}$$

where for every $k \in K$, $s \notin fv(C^k_{ij})$ and K is finite.

So $WC(E,F) = (e = f)\wedge$

$$\bigwedge_{i\in I}\forall t(e_i \leq t \leq e'_i \rightarrow \bigvee_{\substack{j\in J\\ l\in L\\ \alpha_i=\beta_j}} (max(f_i, g_l) \leq t \leq min(f'_j, g'_l) \wedge D^l_{ij}))$$

$$\bigwedge_{j\in J}\forall s(max(0, f_j) \leq s \leq f'_j \rightarrow \bigvee_{\substack{i\in I\\ k\in K\\ \alpha_i=\beta_j}} (max(e_i, h_k) \leq s \leq min(e'_i, h'_k) \wedge C^k_{ij}))$$

For any $i \in I$, we may replace

$$\forall t(e_i \leq t \leq e'_i \rightarrow \bigvee_{\substack{j\in J\\ l\in L\\ \alpha_i=\beta_j}} (max(f_i, g_l) \leq t \leq min(f'_j, g'_l) \wedge D^l_{ij}))$$

by

$$\forall t(\bigvee_{\substack{(u_1,v_1)\in Inv\\ \vdots\\ (u_n,v_n)\in Inv}} (e_i \geq u_1 \wedge u_1 \leq v_1 \wedge u_2 \leq v_1 \leq v_2 \wedge \cdots \wedge$$

$$u_n \leq v_{n-1} \leq v_n \wedge v_n \geq e'_i \wedge \bigwedge_{\substack{l \in L' \\ j \in J}} D^l_{ij}))$$

where

$$Inv = \{(max(f_j, g_l), min(f'_j, g'_l)) \, : \, j \in J, l \in L\}$$

$$L' = \{l \, : \, max(f_j, g_l) \in \{u_1, \ldots, u_n\}\}$$

in $WC(E, F)$.

Since $t \notin fv(\{e_i, e'_i, f_j, f'_j, g_l, g'_l, D^{l'}_{ij} \; : \; j \in J, l \in L, l' \in L'\})$, the result can further be replaced by

$$\bigvee_{\substack{(u_1, v_1) \in Inv \\ \vdots \\ (u_n, v_n) \in Inv}} (e_i \geq u_1 \wedge u_1 \leq v_1 \wedge u_2 \leq v_1 \leq v_2 \wedge \cdots \wedge$$

$$u_n \leq v_{n-1} \leq v_n \wedge v_n \geq e'_i \bigwedge_{\substack{l \in L' \\ j \in J}} D^l_{ij})$$

For the other parts, we can do the same replacement. It is easy to show that the final result can be written in the required disjunctive normal form. □

Remark For discrete time domains, the lemma in general does not hold. For example, the formula $3t = 5s$ cannot be written in the required form. In the next section, we will show how to restrict the occurrences of time variables in time expressions such that we can still retain the lemma for discrete time domains. □

Proposition 6.3.3 *For any processes E and F, $WC(E, F) \vdash E = F$.*

Proof: We proceed by induction on the maximal depth of E and F.

For the base case, we have $E \equiv (e)\delta$ and $F \equiv (f)\delta$. Then $WC(E, F) \equiv (e = f)$ and $e = f \vdash E = F$.

For the inductive case, we have

$$E \equiv \sum_{i \in I} \alpha_i(t_i)_{e_i}^{e_i'}.E_i + (e)\delta \quad \text{and} \quad F \equiv \sum_{j \in J} \beta_j(s_j)_{f_j}^{f_j'}.F_j + (f)\delta$$

Suppose $I \neq \emptyset$. If $J = \emptyset$, then $F \equiv (f)\delta$ and

$$WC(E,F) \leftrightarrow (e = f) \wedge \bigwedge_{i \in I} (e_i > e_i')$$

Clearly

$$(e = f) \wedge \bigwedge_{i \in I} (e_i > e_i') \vdash E = F$$

Hence $WC(E,F) \vdash E = F$.

Now suppose $J \neq \emptyset$, for any $i \in I$, $j \in J$, by induction

$$WC(E_i\{t/t_i\}, F_j\{t/s_j\}) \vdash E_i\{t/t_i\} = F_j\{t/s_j\}$$

By the lemma, $WC(E_i\{t/t_i\}, F_j\{t/s_j\})$ can be written as

$$\bigvee_{l \in L} (g_l \leq t \leq g_l' \wedge D_{ij}^l)$$

where for any $l \in L$, $t \notin fv(D_{ij}^l)$. So, for any $l \in L$

$$g_l \leq t \leq g_l' \wedge D_{ij}^l \vdash E_i\{t/t_i\} = F_i\{t/s_j\}$$

Thus

$$(max(e_i, f_j, g_l) \leq t \leq min(e_i', f_j', g_l')) \wedge D_{ij}^l \vdash E_i\{t/t_i\} = F_i\{t/s_j\}$$

and

$$D_{ij}^l \vdash \alpha_i(t_i)_{max(e_i,f_j,g_l)}^{min(e_i',f_j',g_l')}.E_i = \alpha_i(s_j)_{max(e_i,f_j,g_l)}^{min(e_i',f_j',g_l')}.F_j$$

Let $Inv = \{(max(f_j, g_l), min(f_l', g_j')) : j \in J \& l \in L\}$. For any $(u_1, v_1), \ldots, (u_n, v_n) \in Inv$, let

$$C = e_i \geq u_1 \wedge u_1 \leq u_2 \leq v_1 \wedge \cdots \wedge u_{n-1} \leq u_n \wedge u_n \leq v_{n-1} \leq v_n \wedge v_n \geq e_i'$$

Then

$$C \vdash \alpha_i(t_i)_{e_i}^{e_i'}.E_i = \sum_{k=1}^{n} \alpha_i(t_i)_{max(e_i,u_k)}^{min(e_i',v_k)}.E_i$$

and

$$(\bigwedge_{l \in L'} D^l_{ij}) \wedge C \vdash \alpha_i(t_i)^{e'_i}_{e_i}.E_i = \sum_{\substack{k=1 \\ \alpha_i=\beta_{\pi(k)}}}^{n} \beta_{\pi(k)}(s_{\pi(k)})^{min(e'_i,v_k)}_{max(e_i,u_k)}.F_{\pi(k)}$$

where $L' = \{l : max(f_j, g_l) \in \{u_1, \ldots, u_n\}\}$ and $\pi(k) = j$ where $u_k = max(f_j, g_l)$. So

$$C \wedge \bigwedge_{l \in L'} D^l_{ij} \vdash \alpha_i(t_i)^{e'_i}_{e_i}.E_i + F = F$$

and

$$\bigvee_{\substack{(u_1,v_1)\in Inv \\ \vdots \\ (u_n,v_n)\in Inv}} (\bigwedge_{l \in L'} D^l_{ij} \wedge C) \vdash \alpha_i(t_i)^{e'_i}_{e_i}.E_i + F = F$$

where

$$C \equiv e_i \geq u_1 \wedge u_1 \leq u_2 \leq v_1 \wedge \cdots \wedge u_{n-1} \leq u_n \wedge u_n \leq v_{n-1} \leq v_n \wedge v_n \geq e'_i$$

Hence

$$\bigwedge_{i \in I}(\bigvee_{\substack{(u_1,v_1)\in Inv \\ \vdots \\ (u_n,v_n)\in Inv}} (\bigwedge_{l \in L'} D^l_{ij} \wedge C)) \wedge (e = f) \vdash E + F = F$$

and

$$WC(E, F) \vdash E + F = F$$

Similarly

$$WC(E, F) \vdash E + F = E$$

So

$$WC(E, F) \vdash E = F \qquad \square$$

Corollary 6.3.4 *$A \models E \sim F$ implies $A \vdash E = F$.*

Corollary 6.3.5 (Completeness) *For any processes E and F,*

$$E \sim F \textit{ implies} \vdash E = F$$

6.3.2 Completeness in Discrete Time Domains

In this section, we assume time to be discrete. For example, the time domain is $(\mathbb{N} \cup \{\infty\}, \leq)$, where $\mathbb{N}$ is the set of natural numbers.

As shown in the last section, Lemma 6.3.2 in general does not hold for discrete time domains. As a result, the proof system is not complete for processes over discrete time domain $(\mathbb{N} \cup \{\infty\}, \leq)$. For example, consider processes $E \equiv a(t)_{6}^{6}\, b(s)_{10}^{10}\, (3s)\delta$ and $F \equiv a(t)_{6}^{6}\, b(s)_{10}^{10}\, (5t)\delta$. Clearly we have $\models E \sim F$, but $\vdash E = F$ is not derivable using our proposed technique. However, if we restrict the occurrences of time variables in time expressions, we can still retain the lemma.

We first formalise a notion of time expressions of a process.

Notation Let e be a time expression and S be a set of time expressions. We write $e + S$ in place of the set $\{e + f \,:\, f \in S\}$ of time expressions. □

Definition 6.3.6 *For any process E, the set of time expressions $Exp_1(E)$ of E is inductively defined as follows:*

$$\begin{array}{ll}
Exp_1(\delta) = \{0\} & Exp_1(a(t)_{e}^{e'}.\, E) = \{e, e'\} \\
Exp_1(nil) = \{\infty\} & Exp_1(E + F) = Exp_1(E) \cup Exp_1(F) \\
Exp_1((e)E) = e + Exp_1(E) &
\end{array}$$

Definition 6.3.7 *For any process E, the sets $Exp_2(E)$ and $Exp_3(E)$ of time expressions of E are defined as follows:*

$$\begin{array}{ll}
Exp_2(\delta) = \emptyset & Exp_3((e)E) = Exp_1((e)E) \cup Exp_2(E) \\
Exp_3(\delta) = \emptyset & Exp_2(a(t)_{e}^{e'}.\, E) = Exp_3(E) \\
Exp_2(nil) = \emptyset & Exp_3(a(t)_{e}^{e'}.\, E) = Exp_1(a(t)_{e}^{e'}.\, E) \cup Exp_3(E) \\
Exp_3(nil) = \emptyset & Exp_2(E + F) = Exp_2(E) \cup Exp_2(F) \\
Exp_2((e)E) = Exp_2(E) & Exp_3(E + F) = Exp_3(E) \cup Exp_3(F)
\end{array}$$

Proposition 6.3.8 *For any normal form E, where $E \equiv \sum_{i\in I} a_i(t_i)_{e_i}^{e'_i}.E_i + (e)\delta$, we have*

$$Exp_3(E) = \bigcup_{i\in I} Exp_3(E_i) \cup \{e+0, e_i, e'_i \ : \ i \in I\}$$

Proof: Straightforward. □

For any set S of time expressions, let $Basic(S)$ be the set of time expressions resulted by eliminating max and min in S by the following procedure:

1 Let $Basic(S)$ be S.

2 If $max(e,f) \in Basic(S)$ or $min(e,f) \in Basic(S)$, then replace $max(e,f)$ or $min(e,f)$ by e and f in $Basic(S)$.

3 If $e \circ max(f,f') \in Basic(S)$, or $max(f,f') \circ e \in Basic(S)$, replace them by $e \circ f$ and $e \circ f'$, where $\circ$ is $+$ or $\dot{-}$.

4 If $e \circ min(f,f') \in Basic(S)$, or $min(f,f') \circ e \in Basic(S)$, replace them by $e \circ f$ and $e \circ f'$, where $\circ$ is $+$ or $\dot{-}$.

5 Repeat steps 2 to 4 until there is no occurrence of min or max in $Basic(S)$.

For any set S of time expressions, we say S only contains time expressions which have single occurrence of the same time variables if for any time expression e of $Basic(S)$ and any time variable t, if t occurs in e, then e satisfies one of the following conditions:

(1) $e = f \circ t$ or $e = t \circ f$ for some time expression f, where $t \notin fv(f)$, and $\circ$ is $+$ or $\dot{-}$.

(2) $e = e_1 \circ e_2$ for some time expressions e_1 and e_2 such that $t \notin fv(e_2)$ and e_1 satisfies one of the two conditions, or $t \notin fv(e_1)$ and e_2 satisfies one of the two conditions, where $\circ$ is $+$ or $\dot{-}$.

Let $\mathcal{E}'$ be the set of all processes such that for any $E \in \mathcal{E}'$, $Basic(Exp_3(E))$ only contains time expressions which have single occurrence of the same time variables. The sublanguage of $\mathcal{E}'$ is still very rich. In fact, we have the following property:

Proposition 6.3.9 *For any processes E and F, if $E, F \in \mathcal{E}'$, then $E+F$ is still in $\mathcal{E}'$. Also if $\{e, e'\}$ only contains time expressions which have single occurrence of the same time variables and $E \in \mathcal{E}'$, then $\alpha(t)_e^{e'}.E \in \mathcal{E}'$.*

Proof: Note that $Basic(Exp_3(E+F)) = Basic(Exp_3(E)) \cup Basic(Exp_3(F))$ and $Basic(Exp_3(\alpha(t)_e^{e'}.E)) = Basic(\{e, e'\}) \cup Basic(Exp_3(E))$. The result is straightforward. $\square$

Remark If we consider the language defined by the syntax of Chapter 3, we only need to restrict the definition of time expressions such that the new definition of time expressions only allows those time expressions which have single occurrence of the same time variables, i.e. we only allow time expressions defined by the following:

1. for any $u \in \mathcal{T}$ and $t \in V_t$, u and t are time expressions;

2. for any time expression e and time variable $t \in V_t$, if t does not occur in e, then $e+t$, $t+e$, $e \dot{-} t$, $t \dot{-} e$, $max(e, t)$, $max(t, e)$, $min(e, t)$ and $min(t, e)$ are all time expressions. $\square$

Now we can show that for the processes of $\mathcal{E}'$, Lemma 6.3.2 of the last section still holds.

Lemma 6.3.10 *For any processes E and F, where $E \in \mathcal{E}'$ and $F \in \mathcal{E}'$, $WC(E, F)$ can be written in the disjunctive normal form*

$$\bigvee_{i \in I} e_{1i} \leq t_1 \leq f_{1i} \wedge \cdots \wedge e_{ni}(t_1, \ldots, t_{n-1}) \leq t_n \leq f_{ni}(t_1, \ldots, t_{n-1})$$

for some n and I, where I is finite, and $e_{ki}(t_1, \ldots, t_{k-1})$, $f_{ki}(t_1, \ldots, t_{k-1})$ $(k = 1, \cdots, n)$ contain at most variables $t_1, \ldots, t_{k-1}$.

Proof: The proof is similar to that of Lemma 6.3.2 (note the restriction over occurrences of time variables in time expressions). □

Proposition 6.3.11 *For any processes E and F, where $E \in \mathcal{E}'$ and $F \in \mathcal{E}'$ are in normal form, $WC(E, F) \vdash E = F$.*

Proof: Analogous to Proposition 6.3.3. □

By Lemma 6.3.1 and Proposition 6.3.11, the following results are straightforward.

Corollary 6.3.12 *For any processes $E \in \mathcal{E}'$ and $F \in \mathcal{E}'$, $A \models E \sim F$ implies $A \vdash E = F$.*

Corollary 6.3.13 (Completeness) *For any processes E and F of $\mathcal{E}'$,*

$$E \sim F \text{ implies} \vdash E = F$$

Remark Although it is not known if the above relativised compositional proof system is complete for general processes over discrete time domains, the simple proof system discussed in the begining of the chapter is complete over discrete time domains.

6.4 Proofs of Recursively Defined Processes

In this section, we consider the proofs of recursively defined processes. We show by an example how the proof system works for recursively defined processes. To do so, the proof system needs to be augmented with an axiom

$$true \vdash \mu X.E = E\{\mu X.E/X\}$$

and some form of induction. We choose a very simple form of induction, namely *Unique Fixpoint Induction*:

$$\frac{true \vdash P = E\{P/X\}}{true \vdash P = \mu X.E}$$

The soundness of the axiom follows from Proposition 4.3.5. For a weakly guarded process E, the soundness of the above inductive rule follows from Proposition 4.3.5 and Corollary 4.5.5. However the inductive rule, in general, is not valid for processes which are not weakly guarded. For example, if $E \equiv X$, then for any process P we have $P \sim X\{P/X\}$ and clearly $a(t)_0^{10}.\, nil \not\sim \mu X.X$. Also, for every agent P, we have $P + a(t)_0^{10}.\, \delta \sim (a(t)_0^{10}.\, \delta + X)\{(P + a(t)_0^{10}.\, \delta)/X\}$, but $P + a(t)_0^{10}.\, \delta \not\sim \mu X.a(t)_0^{10}.\, \delta + X$ when we have $P \equiv b(s)_0^{10}.\, \delta$.

Now we consider processes

$$P \equiv \mu X.a(t)_1^{10}.\, (b(s)_0^{t-(2\dot{-}t)}.\, X + b(s)_0^{2t-2}.\, X + b(s)_0^{t}.\, X)$$

and

$$Q \equiv \mu X.a(t)_1^{10}.\, (b(s)_0^{2t-2}.\, X + b(s)_0^{t}.\, X)$$

We show how to derive

$$\vdash P = Q$$

in the extended proof system.

Since Q is recursively defined and weakly guarded, by the above induction rule we only need to show that

$$\vdash P = a(t)_1^{10}.\,(b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P)$$

However we have

$$\vdash P = a(t)_1^{10}.\,(b(s)_0^{t-(2\dot{-}t)}.\,P + b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P)$$

By rule 1 we only need to show

$$\vdash a(t)_1^{10}.\,(b(s)_0^{t-(2\dot{-}t)}.\,P + b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P) = a(t)_1^{10}.\,(b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P)$$

Clearly

$$\vdash P = P$$

$$\vdash b(s)_0^{2t-2}.\,P = b(s)_0^{2t-2}.\,P$$

and

$$\vdash b(s)_0^{t}.\,P = b(s)_0^{t}.\,P$$

By rule 5, we have

$$t \geq 2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,P = b(s)_0^{t}.\,P$$

and

$$t \leq 2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,P = b(s)_0^{2t-2}.\,P$$

By rule 7, we have

$$t \leq 2 \vee t \geq 2 \vdash b(s)_0^{t-(2\dot{-}t)}.\,P + b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P = b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P$$

Since $1 \leq t \leq 10 \rightarrow t \leq 2 \bigvee t \geq 2$, by rule 3 we have

$$1 \leq t \leq 10 \vdash b(s)_0^{t-(2\dot{-}t)}.\,P + b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P = b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P$$

By rule 5, we have the result

$$\vdash a(t)_1^{10}.\,(b(s)_0^{t-(2\dot{-}t)}.\,P + b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P) = a(t)_1^{10}.\,(b(s)_0^{2t-2}.\,P + b(s)_0^{t}.\,P)$$

Remark Even untimed CCS is Turing-powerful [Mil89a] and therefore no effective complete proof system can exist. By adding sufficiently powerful inductive methods for handling recursively defined processes, we would have a complete (and therefore ineffective) proof system for reasoning about real-timed processes.

6.5 Conclusion

In this chapter, we proposed a simple proof system for strong equivalence. The proof system is independent of the choice of time domain, allowing time to be discrete or dense. We showed the soundness of the proof system. We also showed that the proof system is complete for finite processes over dense time domains, but only complete for processes of a restricted language over discrete time domains. We discussed how to restrict the definition of time expressions to get the restricted language. There are no infinitary proof rules and the proof system is realistic and hopefully useful.

Although the proof system is presented for Timed CCS, the approach can also be used for other work. For example, the approach can be used for the restricted language of Baeten and Bergstra's $ACP\rho I$ discussed in [Klu91] (restricting to those prefixed integrations and not allowing general integration). As discussed in Chapter 3, the restricted language of $ACP\rho I$ corresponds to Timed CCS.

Recently, Klusener proposed a sound and complete proof system [Klu91] for finite processes of the restricted language of $ACP\rho I$ and the work is for absolute time. In [Che91a], the author presents a sound and complete proof system (for finite processes) for relative time. The work of [Che91a,Klu91] is based on some powerful infinitary rules. If in Timed CCS, such an infinitary rule would be

$$\frac{\forall u(e \leq u \leq e'), E\{u/t\} = F\{u/t\}}{\alpha(t)_e^{e'}.E = \alpha(t)_e^{e'}.F}$$

The presence of infinitary rules means that the proof systems, although sound and complete, are only of theoretical interest.

Chapter 7

Behavioural Abstraction in Timed Processes

Strong bisimulation does not consider the problem of abstraction. It deals with the internal action τ in the same way as all other actions and requires that every τ action in one agent to be matched by a τ action of the other. In this chapter, we define some weak equivalences which abstract away the internal actions.

7.1 Weak Bisimulation

We first define a weak bisimulation which allows every internal action τ to be matched by zero or more τ actions.

Since the internal action τ is not observable, an agent which performs some τ actions during time u cannot be distinguished from an agent which just idles time u without any action, including τ actions. We define a transition $P \overset{\epsilon}{\Longrightarrow}_u P'$ which means that agent P idles time u without any observable actions. So $P \overset{\epsilon}{\Longrightarrow}_u P'$ may involve internal silent transitions. For convenience, we also write $P \Longrightarrow_u P'$ in place of $P \overset{\epsilon}{\Longrightarrow}_u P'$.

Definition 7.1.1 *We say* $P \stackrel{\epsilon}{\Longrightarrow}_{v+u_1+\cdots+u_n} P'$ *if* $P \stackrel{\tau}{\longrightarrow}_{u_1} \cdots \stackrel{\tau}{\longrightarrow}_{u_n} \cdot \longrightarrow_v P'$ *for some* $u_1, \ldots, u_n, v \in \mathcal{T}$. *In particular,* $P \stackrel{\epsilon}{\Longrightarrow}_0 P$.

We also define a transition $P \stackrel{a}{\Longrightarrow}_u P'$ which means that an action a is observed at time u. However, before action a, there may be finitely many unobservable internal actions, i.e. τ actions. Also there may be finitely many internal actions happening at the same time as action a (we say two actions happen at the same time if one follows another without any delay between them).

Definition 7.1.2 *We say* $P \stackrel{a}{\Longrightarrow}_{u+v} P'$ *if* $P \stackrel{\epsilon}{\Longrightarrow}_u \cdot \stackrel{a}{\longrightarrow}_v \cdot \stackrel{\epsilon}{\Longrightarrow}_0 P'$ *for some* $u, v \in \mathcal{T}$.

Notation $\hat{\alpha} \stackrel{def}{=}$ if $\alpha = \tau$ then ϵ else α. □

Definition 7.1.3 *A binary relation* $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ *is a weak bisimulation if* $(P, Q) \in \mathcal{R}$ *implies that for any* $\alpha \in Act$ *and* $u \in \mathcal{T}$

1 *if* $P \stackrel{\alpha}{\Longrightarrow}_u P'$, *then* $Q \stackrel{\hat{\alpha}}{\Longrightarrow}_u Q'$ *and* $(P', Q') \in \mathcal{R}$ *for some* Q';

2 *if* $P \longrightarrow_u P'$, *then* $Q \stackrel{\epsilon}{\Longrightarrow}_u Q'$ *and* $(P', Q') \in \mathcal{R}$ *for some* Q';

3 *if* $Q \stackrel{\alpha}{\Longrightarrow}_u Q'$, *then* $P \stackrel{\hat{\alpha}}{\Longrightarrow}_u P'$ *and* $(P', Q') \in \mathcal{R}$ *for some* P'; *and*

4 *if* $Q \longrightarrow_u Q'$, *then* $P \stackrel{\epsilon}{\Longrightarrow}_u P'$ *and* $(P', Q') \in \mathcal{R}$ *for some* P'.

Remark If we replace transitions of the form $P \stackrel{\alpha}{\Longrightarrow}_u P'$ by transitions of the form $P \stackrel{\alpha}{\longrightarrow}_u P'$, we get the same weak bisimulations.

We say two agents P and Q are weakly bisimilar if there is a weak bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.

Definition 7.1.4 $\approx = \bigcup\{\mathcal{R} : \mathcal{R}$ *is a weak bisimulation*$\}$

Proposition 7.1.5 $\approx$ *is the largest weak bisimulation. Moreover it is an equivalence relation.*

Proof: The proof follows from the facts that the union, the composition and the inverse of weak bisimulations are still weak bisimulations and also the identity relation over agents is a weak bisimulation. □

Clearly, for any agents P and Q, $P \approx Q$ and $P \overset{\epsilon}{\Longrightarrow}_t P'$ implies that $Q \overset{\epsilon}{\Longrightarrow}_t Q'$ and $P' \approx Q'$ for some Q'.

Definition 7.1.6 *We say $\mathcal{S}$ is a weak bisimulation up to $\approx$ if $P\mathcal{S}Q$ implies that*

(1) *if $P \overset{\alpha}{\Longrightarrow}_u P'$, then $Q \overset{\hat{\alpha}}{\Longrightarrow}_u Q'$ and $P' \approx \cdot \mathcal{S} \cdot \approx Q'$ for some Q';*

(2) *if $P \longrightarrow_u P'$, then $Q \Longrightarrow_u Q'$ and $P' \approx \cdot \mathcal{S} \cdot \approx Q'$ for some Q';*

(3) *if $Q \overset{\alpha}{\Longrightarrow}_u Q'$, then $P \overset{\hat{\alpha}}{\Longrightarrow}_u P'$ and $P' \approx \cdot \mathcal{S} \cdot \approx Q'$ for some P'; and*

(4) *if $Q \longrightarrow_u Q'$, then $P \Longrightarrow_u P'$ and $P' \approx \cdot \mathcal{S} \cdot \approx Q'$ for some P'.*

Similarly to strong bisimulation, we show that, to establish $P \approx Q$, it is enough to show that there is a weak bisimulation up to $\approx$ which contains the pair (P, Q).

Proposition 7.1.7 *If $\mathcal{S}$ is a weak bisimulation up to $\approx$, then $\mathcal{S} \subseteq \approx$.*

Proof: Analogous to Proposition 4.1.9. □

Remark If we use transitions of the form $P \overset{\alpha}{\longrightarrow}_t P'$ to replace the transitions of the form $P \overset{\alpha}{\Longrightarrow}_t P'$ in Definition 7.1.3, then the proposition does not hold [SM92]. □

A strong bisimulation is also a weak bisimulation, as shown by the following proposition:

Proposition 7.1.8 *If $P \sim Q$, then $P \approx Q$.*

Proof: Straightforward. $\square$

All laws for strong equivalence still hold for weak equivalence. Moreover, we have some properties which distinguish $\approx$ from $\sim$. As a result, strong equivalence has more discriminating power than weak bisimulation.

Proposition 7.1.9

(1) $Q + \tau(t)_u^{u'}.(t >> (u)P) + (u)P \approx Q + \tau(t)_u^{u'}.(t >> (u)P)$
$\qquad$ *where* $\mid Q \mid_{\mathcal{T}} \leq u \leq u'$

(2) $\tau(t)_u^{u'}.(t >> ((u)P + (u')\delta)) \approx (u)P + (u')\delta \qquad$ *where* $u \leq u'$

(3) $Q + \tau(t)_u^{u'}.(t >> ((u)P + (u')\delta)) \approx Q + (u)P + (u')\delta \qquad$ *where* $\mid Q \mid_{\mathcal{T}} < u \leq u'$

Proof: All properties may be proved by exhibiting appropriate weak bisimulations. We only consider (2), as all others are similar.

For (2), it is enough to show that

$$\mathcal{S} = \{(\tau(t)_v^{v'}.(t >> ((v)P + (v')\delta)), (v)P + (v')\delta) : 0 \leq v \leq v'\} \cup \approx$$

is a weak bisimulation up to $\approx$.

- Suppose $\tau(t)_v^{v'}.(t >> ((v)P + (v')\delta)) \stackrel{\alpha}{\Longrightarrow}_u P'$. If $\alpha = \tau$, then for some w, where $v \leq w \leq v'$ and $w \leq u$, $\tau(t)_v^{v'}.(t >> ((v)P + (v')\delta)) \stackrel{\tau}{\longrightarrow}_w w >> ((v)P + (v')\delta)$ and $w >> ((v)P + (v')\delta) \stackrel{\epsilon}{\Longrightarrow}_{u-w} P'$. Clearly $(v)P + (v')\delta \longrightarrow_w Q$ for some Q such that $Q \sim w >> ((v)P + (v')\delta)$. So $(v)P + (v')\delta \stackrel{\epsilon}{\Longrightarrow}_u Q'$ for some Q' such that $P' \approx Q'$. If $\alpha \neq \tau$, then for some w, where $v \leq w \leq v'$ and $w \leq u$, $\tau(t)_v^{v'}.(t >> ((v)P + (v')\delta)) \stackrel{\tau}{\longrightarrow}_w w >> ((v)P + (v')\delta)$ and $w >> ((v)P + (v')\delta) \stackrel{\alpha}{\Longrightarrow}_{u-w} P'$. However $(v)P + (v')\delta \longrightarrow_w Q$ for some Q, where $Q \sim w >> ((v)P + (v')\delta) \stackrel{\alpha}{\Longrightarrow}_{u-w} P'$. So $(v)P + (v')\delta \stackrel{\alpha}{\Longrightarrow}_w Q'$ for some Q' such that $P' \approx Q'$.

- Suppose $\tau(t)^{v'}_{v}.(t \gg ((v)P + (v')\delta)) \longrightarrow_u P'$. Then $u \leq v'$ and $P' \equiv \tau(t)^{v'-u}_{v \dot{-} u}.(t + u \gg ((v)P + (v')\delta))$. If $u \leq v$, then

$$P' \approx \tau(t)^{v'-u}_{v-u}.(t \gg ((v-u)P + (v'-u)\delta))$$

and $(v)P + (v')\delta \longrightarrow_u P''$ for some P'', where $P'' \sim (v-u)P + (v'-u)\delta$. Clearly $P' \approx \cdot\mathcal{S}\cdot \approx P''$. If $u \geq v$, then $P' \approx \tau(t)^{v'-u}_{0}.(t \gg (u - v \gg P + (v'-u)\delta)$ and $(v)P + (v')\delta \longrightarrow_u P''$ for some P'', where $P'' \sim u - v \gg P + (v'-u)\delta$. Clearly $P' \approx \cdot\mathcal{S}\cdot \approx P''$.

- Suppose $(v)P + (v')\delta \stackrel{\alpha}{\Longrightarrow}_u P'$. Then $(v)P \stackrel{\alpha}{\Longrightarrow}_u P'$ and $u \geq v$. Clearly $\tau(t)^{v'}_{v}.(t \gg ((v)P + (v')\delta)) \stackrel{\tau}{\longrightarrow}_v \stackrel{\alpha}{\longrightarrow}_{u-v} P''$ for some P'' such that $P'' \approx P'$.

- Suppose $(v)P + (v')\delta \longrightarrow_u P'$.

Case 1 $u \leq v$, then $P' \approx (v-u)P + (v'-u)\delta$. Clearly $\tau(t)^{v'}_{v}.(t \gg ((v)P + (v')\delta)) \longrightarrow_u \tau(t)^{v'-u}_{v-u}.(t + u \gg ((v)P + (v')\delta))$ where $\tau(t)^{v'-u}_{v-u}.(t + u \gg ((v)P + (v')\delta)) \approx \tau(t)^{v'-u}_{v-u}.(t \gg ((v-u)P + (v'-u)\delta))$. So $P' \approx \cdot\mathcal{S}\cdot \approx \tau(t)^{v'-u}_{v-u}.(t + u \gg ((v)P + (v')\delta))$.

Case 2 $v < u \leq v'$, then $P' \sim u - v \gg P + (v'-u)\delta$. Clearly $\tau(t)^{v'}_{v}.(t \gg ((v)P + (v')\delta)) \longrightarrow_u \tau(t)^{v'-u}_{0}.(t + u \gg ((v)P + (v')\delta))$ where $\tau(t)^{v'-u}_{0}.(t + u \gg ((v)P + (v')\delta)) \approx \tau(t)^{v'-u}_{0}.(t \gg (u - v \gg P + (v'-u)\delta))$. So $P' \approx \cdot\mathcal{S}\cdot \approx \tau(t)^{v'-u}_{0}.(t + u \gg ((v)P + (v')\delta))$

Case 3 $v' < u$, then $P' \sim u - v' \gg P$. Clearly $\tau(t)^{v'}_{v}.(t \gg ((v)P + (v')\delta)) \stackrel{\tau}{\longrightarrow}_{v'} \longrightarrow_{u-v'} P''$ for some P'', where $P'' \sim u - v' \gg P$. So $P' \approx P''$.

All other cases are straightforward. □

Note that the side conditions in Proposition 7.1.9 are important, as seen by the following counterexamples:

- for (1), we have $a(t)_0^{10}.\,\delta + \tau(s)_0^5\,(s \gg b(r)_1^{10}.\,\delta) + b(r)_1^{10}.\,\delta \not\approx a(t)_0^{10}.\,\delta + \tau(s)_0^5\,(s \gg b(r)_1^{10}.\,\delta)$, where the left hand side delays 10 *seconds* (if time is measured in seconds) and then evolves to a state which can perform either action a or action b but the right hand side can only either do action a or do action b (without choice) after a delay of 10 *seconds* without any observable actions;

- for (2), we have $\tau(t)_8^6\,(t >> a(s)_{10}^{15}\,\delta) + a(s)_{10}^{15}\,\delta \not\approx \tau(t)_8^6\,(t >> a(s)_{10}^{15}\,\delta)$ as the left hand side can perform action a at time 10 which is not matched by right hand side; and

- for (3), we have $a(t)_0^{10}.\,\delta + \tau(s)_8^{15}.\,(s \gg (8)b(r)_0^{10}.\,\delta) \not\approx a(t)_0^{10}.\,\delta + (8)b(r)_0^{10}.\,\delta$, where the left hand side can have an internal transition at 10 *seconds* and then can only perform action b but the right hand side can always do either action a or action b after a delay of 10 seconds without any observable actions.

Similarly to CCS, weak equivalence is not a congruence relation. In general the summation operation does not preserve weak equivalence. For example, $\tau(t)_2^8\,nil \approx nil$, but $\tau(t)_2^8\,nil + a(s)_8^{\infty}.\,nil \not\approx nil + a(s)_8^{\infty}.\,nil$. However, weak equivalence is preserved by the other operations.

Proposition 7.1.10 *For any agents P and Q, $P \approx Q$ implies*

$$
\begin{array}{llll}
(1) & \alpha(t)_e^{e'}.\,P \approx \alpha(t)_e^{e'}.\,Q & (3) & P\backslash a \approx Q\backslash a \\
(2) & P \mid R \approx Q \mid R & (4) & P[S] \approx Q[S]
\end{array}
$$

Proof: All properties may be proved by exhibiting appropriate weak bisimulations. We only consider (2), as all other cases are similar.

For (2), we only need to show that the relation

$$S = \{(P \mid R, Q \mid R) : P \approx Q\}$$

is a weak bisimulation. □

As for strong bisimulation, we can also generalize the definition of weak bisimulation to any processes.

Definition 7.1.11 *For any processes E and F which are closed with respect to process variables and contain at most free time variables $\vec{t} = (t_1, \ldots, t_n)$, we say $E \approx F$ if for any $\vec{u} \in \mathcal{T}^n$ we have $E\{\vec{u}/\vec{t}\} \approx F\{\vec{u}/\vec{t}\}$.*

Definition 7.1.12 *For any processes E and F which contain at most free process variables $\vec{X} = (X_1, \ldots, X_n)$, we say $E \approx F$ if for any agents $\vec{P} = (P_1, \ldots, P_n)$, we have $E\{\vec{P}/\vec{X}\} \approx F\{\vec{P}/\vec{X}\}$.*

All properties for agents also hold for processes. For example, for any processes E and F, if $E \sim F$, then $E \approx F$.

7.2 Observational Congruence

Weak equivalence is not fully substitutive. In general, it is not preserved by summation operator. In this section, we define an equality which will be fully substitutive and also the largest congruence relation included in $\approx$. The following definition for $=^+$ is a standard CCS technique.

Definition 7.2.1 *We say $E =^+ F$ if for any agent P, we have $E + P \approx F + P$.*

Clearly $=^+$ is included in $\approx$. In fact $=^+$ lies between the strong and the weak equivalences.

Proposition 7.2.2 *$E \sim F$ implies $E =^+ F$ and $E =^+ F$ implies $E \approx F$.*

Proof: The first part follows from the facts that strong equivalence is preserved by summation and $E \sim F$ implies $E \approx F$. The second part follows by taking

$P \equiv \delta$ in the definition of $=^+$. $\square$

The converse of the above does not hold. For example, $\tau.nil \approx nil$, but $\tau.nil \neq^+ nil$. Also $a(t)_2^{10}.\tau.nil =^+ a(t)_2^{10}.nil$, but $a(t)_2^{10}.\tau.nil \not\approx a(t)_2^{10}.nil$.

We say an agent is stable if it cannot perform an internal action τ as its first action.

Definition 7.2.3 *An agent P is stable if there are no P' or $u \in \mathcal{T}$ such that $P \xrightarrow{\tau}_u P'$.*

Proposition 7.2.4 *For any stable agents P and Q, $P \approx Q$ implies $P =^+ Q$*

Proof: We only need to show that the relation

$$\mathcal{S} = \{(P + R,\, Q + R) \,:\, P \approx Q \,\&\, both\ P\ and\ Q\ are\ stable\} \cup \approx$$

is a weak bisimulation.

The case $(P, Q) \in \approx$ is trivial. Now we consider the case $(P + R, Q + R) \in \mathcal{S}$ for any agents P, Q and R, where P and Q are stable and satisfy $P \approx Q$.

- if $P + R \stackrel{\alpha}{\Longrightarrow}_u P'$, then $P \stackrel{\alpha}{\Longrightarrow}_u P'$ or $R \stackrel{\alpha}{\Longrightarrow}_u P'$.

  Case 1 $P \stackrel{\alpha}{\Longrightarrow}_u P'$, then $\alpha \neq \tau$ (P is stable). So $Q \stackrel{\alpha}{\Longrightarrow}_u Q'$ and $P' \approx Q'$ for some Q'. Hence $Q + R \stackrel{\alpha}{\Longrightarrow}_u Q'$ and $P' \approx Q'$.

  Case 2 $R \stackrel{\alpha}{\Longrightarrow}_u P'$, then $Q + R \stackrel{\alpha}{\Longrightarrow}_u P'$ and $P' \approx P'$.

- if $P + R \longrightarrow_u P''$, then $0 \leq u \leq max(\,|\,P\,|_{\mathcal{T}},\, |\,R\,|_{\mathcal{T}})$.

  Case 1 $u \leq min(\,|\,P\,|_{\mathcal{T}},\, |\,R\,|_{\mathcal{T}})$, then $P \longrightarrow_u P'$, $R \longrightarrow_u R'$ and $P'' \equiv P' + R'$. Clearly P' is still stable. So $Q \longrightarrow_u Q'$ and $P' \approx Q'$ for some Q' (Q is stable). Similarly Q' is still stable. Hence $Q + R \longrightarrow_u Q' + R'$ and $(P' + R', Q' + R') \in S$.

Case 2 $|P|_{\mathcal{T}} < u \leq |R|_{\mathcal{T}}$, then $R \longrightarrow_u P'$. Since P, Q are stable and $P \approx Q$, we have $Q + R \longrightarrow_u P'$ and $P' \approx P'$.

Case 3 $|R|_{\mathcal{T}} < u \leq |P|_{\mathcal{T}}$, then $P \longrightarrow_u P'$. So $Q \longrightarrow_u Q'$ and $P' \approx Q'$ for some Q' (Q is stable). Hence $Q + R \longrightarrow_u P'$ and $P' \approx P'$.

By symmetry, we are done. □

Proposition 7.2.5 *$E \approx F$ implies $\alpha(t)_e^{e'}.E =^+ \alpha(t)_e^{e'}.F$.*

Proof: We only need to consider the case that $\alpha(t)_e^{e'}.E$ and $\alpha(t)_e^{e'}.F$ are agents. If $\alpha \neq \tau$, the result follows from Lemma 7.1.10 and Lemma 7.2.4. Now suppose $\alpha = \tau$. It is enough to show that the relation

$$\mathcal{S} = \{(\tau(t)_v^{v'}.E + R, \tau(t)_v^{v'}.F + R) : E \approx F \,\&\, fv_t(E,F) \subseteq \{t\} \,\&\, fv_p(E,F) = \emptyset\} \cup \approx$$

is a weak bisimulation.

The case $(P,Q) \in \approx$ is trivial. Now we consider the case $(\tau(t)_v^{v'}.E + R, \tau(t)_v^{v'}.F + R) \in \mathcal{S}$ for any agents $\tau(t)_v^{v'}.E$, $\tau(t)_v^{v'}.F$ and R, where $E \approx F$.

- if $\tau(t)_v^{v'}.E + R \overset{\alpha}{\Longrightarrow}_u P$, then $\tau(t)_v^{v'}.E \overset{\alpha}{\Longrightarrow}_u P$ or $R \overset{\alpha}{\Longrightarrow}_u P$.

Case 1 $\tau(t)_v^{v'}.E \overset{\alpha}{\Longrightarrow}_u P$, then $\alpha = \tau$ and $P \equiv E\{u/t\}$ or $E\{w/t\} \overset{\alpha}{\Longrightarrow}_{u-w} P$ for some w, where $v \leq w \leq min(v', u)$. In the first case, we have $\tau(t)_v^{v'}.F + R \overset{\tau}{\longrightarrow}_u F\{u/t\}$ and $E\{u/t\} \approx F\{u/t\}$. In the second case, we have $F\{w/t\} \overset{\hat{\alpha}}{\Longrightarrow}_{u-w} Q$ for some Q such that $P \approx Q$.

Case 2 $R \overset{\alpha}{\Longrightarrow}_u P$, then $\tau(t)_v^{v'}.F + R \overset{\hat{\alpha}}{\Longrightarrow}_u P$ and $P \approx P$.

- if $\tau(t)_v^{v'}.E + R \longrightarrow_u P$, then $u \leq max(v', |R|_{\mathcal{T}})$.

Case 1 $u \leq min(v', |R|_{\mathcal{T}})$, then $P \equiv \tau(t)_{v \dot{-} u}^{v' \dot{-} u}.(E\{u+t/t\}) + R'$ where $R \longrightarrow_u R'$. Clearly $\tau(t)_v^{v'}.F + R \longrightarrow_u \tau(t)_{v \dot{-} u}^{v' \dot{-} u}.(F\{u+t/t\}) + R'$ and $E\{u+t/t\} \approx F\{u+t/t\}$. So

$$(\tau(t)_{v \dot{-} u}^{v' \dot{-} u}.(E\{u+t/t\}) + R', \tau(t)_{v \dot{-} u}^{v' \dot{-} u}.(F\{u+t/t\}) + R) \in \mathcal{S}$$

Case 2 $v' < u \leq \mid R \mid_{\mathcal{T}}$, then $R \longrightarrow_u P'$. Clearly $\tau(t)_v^{v'}.F + R \longrightarrow_u P'$ and $P' \approx P'$.

Case 3 $\mid R \mid_{\mathcal{T}} < u \leq v'$, then $P \equiv \tau(t)_{v \dot{-} u}^{v' - u}.(E\{u+t/t\})$. Clearly $\tau(t)_v^{v'}.F + R \longrightarrow_u \tau(t)_{v \dot{-} u}^{v' - u}.(F\{u+t/t\})$ and

$$\tau(t)_{v \dot{-} u}^{v' - u}.(E\{u+t/t\}) \approx \tau(t)_{v \dot{-} u}^{v' - u}.(F\{u+t/t\})$$

By symmetry, we are done. □

Proposition 7.2.6 (τ-Laws)

1 $\tau(t)_0^{e'}.(\alpha.E + F) =^+ \alpha.(E\{0/t\}) + \tau(t)_0^{e'}.(\alpha.E + F)$

2 $\alpha(t)_e^{e'}.(\tau.E + F) =^+ \alpha(t)_e^{e'}.E + \alpha(t)_e^{e'}.(\tau.E + F)$

3 $\alpha(t)_f^{f'}.(F + \tau(s)_e^{e'}.(s >> (e)E) + (e)E) =^+ \alpha(t)_f^{f'}.(F + \tau(s)_e^{e'}.(s >> (e)E))$
$\quad\quad$ *where* $\mid F \mid_{\mathcal{T}} \leq e \leq e' \;\&\; s \notin fv_t(E)$

4 $\alpha(t)_f^{f'}.(\tau(s)_e^{e'}.(s >> ((e)E + (e')\delta))) =^+ \alpha(t)_f^{f'}.((e)E + (e')\delta)$
$\quad\quad$ *where* $e \leq e' \;\&\; s \notin fv_t((e)E + (e')\delta)$

5 $\alpha(t)_f^{f'}.(F + \tau(s)_e^{e'}.(s >> ((e)E + (e')\delta))) =^+ \alpha(t)_f^{f'}.(F + (e)E + (e')\delta)$
$\quad\quad$ *where* $\mid F \mid_{\mathcal{T}} < e \leq e' \;\&\; s \notin fv_t((e)E + (e')\delta)$

Proof: Properties (3), (4) and (5) follow from Lemma 7.1.9 and Lemma 7.2.5. Properties (1) and (2) may be proved by showing appropriate weak bisimulations. We only consider (2), as (1) is similar.

For (2), it is enough to consider agents and show that the relation

$$\begin{aligned}\mathcal{S} = \{(\alpha(t)_v^{v'}.(\tau.E + F) + R, \alpha(t)_v^{v'}.E + \alpha(t)_v^{v'}.(\tau.E + F) + R) \\ : fv_t(E) \subseteq \{t\} \;\&\; fv_p(E) = \emptyset \;\&\; fv_t(F) \subseteq \{t\} \;\&\; fv_p(F) = \emptyset\} \cup \approx\end{aligned}$$

is a weak bisimulation.

- if $\alpha(t)_v^{v'}.\,E + \alpha(t)_v^{v'}.\,(\tau.E + F) + R \overset{\beta}{\Longrightarrow}_u P$

Case 1 $\alpha(t)_v^{v'}.\,E \overset{\beta}{\Longrightarrow}_u P$, then $\alpha(t)_v^{v'}.\,E \overset{\beta}{\longrightarrow}_u P$ and $P \equiv E\{u/t\}$, or $\alpha = \tau$ and $E\{w/t\} \overset{\beta}{\Longrightarrow}_{u-w} P$ for some w, where $v \leq w \leq min(v', u)$. In the first case, we have

$$\alpha(t)_v^{v'}.\,(\tau.E + F) + R \overset{\alpha}{\longrightarrow}_u \overset{\tau}{\longrightarrow}_0 E\{u/t\}$$

In the second case, we have

$$\alpha(t)_v^{v'}.\,(\tau.E + F) + R \overset{\tau}{\longrightarrow}_w \overset{\tau}{\longrightarrow}_0 E\{w/t\}$$

where $v \leq w \leq min(v', u)$ and $E\{w/t\} \overset{\beta}{\Longrightarrow}_{u-w} P$.

Case 2 $\alpha(t)_v^{v'}.\,(\tau.E + F) \overset{\beta}{\Longrightarrow}_u P$ or $R \overset{\beta}{\Longrightarrow}_u P$, straightforward.

- if $\alpha(t)_v^{v'}.\,E + \alpha(t)_v^{v'}.\,(\tau.E + F) + R \longrightarrow_u P$, then $u \leq max(v', \mid R\mid_{\mathcal{T}})$.

Case 1 $u \leq min(v', \mid R\mid_{\mathcal{T}})$, then

$$P \equiv \alpha(t)_{v\dot{-}u}^{v'-u}.\,(E\{t+u/t\}) + \alpha(t)_{v\dot{-}u}^{v'-u}.\,(\tau.E + F)\{t+u/t\} + R'$$

where $R \longrightarrow_u R'$. Clearly

$$\alpha(t)_v^{v'}.\,(\tau.E + F) + R \longrightarrow_u \alpha(t)_{v\dot{-}u}^{v'-u}.\,(\tau.E + F)\{t+u/t\} + R'$$

and

$$(\alpha(t)_{v\dot{-}u}^{v'-u}.\,(\tau.E + F)\{t+u/t\} + R', P) \in \mathcal{S}$$

Case 2 $v' < u \leq \mid R\mid_{\mathcal{T}}$, then $R \longrightarrow_u P$. Clearly

$$\alpha(t)_v^{v'}.\,(\tau.E + F) + R \longrightarrow_u P$$

Case 3 $\mid R\mid_{\mathcal{T}} < u \leq v'$, then

$$P \equiv \alpha(t)_{v\dot{-}u}^{v'-u}.\,(E\{t+u/t\}) + \alpha(t)_{v\dot{-}u}^{v'-u}.\,(\tau.E + F)\{t+u/t\}$$

Clearly

$$\alpha(t)^{v'}_{v}.(\tau.E+F)+R \longrightarrow_u \alpha(t)^{v'-u}_{v-u}.(\tau.E+F)\{t+u/t\})$$

and

$$(\alpha(t)^{v'-u}_{v-u}.(\tau.E+F)\{t+u/t\}, P) \in S$$

All other cases are straightforward. □

We can also give an alternative definition of $=^+$ via the transitions $\longrightarrow_u$, $\xrightarrow{\alpha}_0$ and $\stackrel{\alpha}{\Longrightarrow}_0$. To do so, we first define an equivalence $=^c$ and then show that $=^+$ and $=^c$ coincide.

Definition 7.2.7 *A binary relation $\mathcal{R}$ over agents is a C-bisimulation if $(P,Q) \in \mathcal{R}$ implies that for any $\alpha \in Act$ and $u \in \mathcal{T}$,*

(1) *if $P\xrightarrow{\alpha}_0 P'$, then $Q\stackrel{\alpha}{\Longrightarrow}_0 Q'$ and $P' \approx Q'$ for some Q';*

(2) *if $P\longrightarrow_u P'$, then $Q\longrightarrow_u Q'$ and $(P',Q') \in \mathcal{R}$;*

(3) *if $Q\xrightarrow{\alpha}_0 Q'$, then $P\stackrel{\alpha}{\Longrightarrow}_0 P'$ and $P' \approx Q'$ for some P'; and*

(4) *if $Q\longrightarrow_u Q'$, then $P\longrightarrow_u P'$ and $(P',Q') \in \mathcal{R}$.*

Definition 7.2.8 $=^c \stackrel{def}{=} \bigcup\{\mathcal{R} : \mathcal{R}$ *is a C-bisimulation*$\}$.

Proposition 7.2.9 *For any weakly guarded agents P and Q, $P =^c Q$ if and only if $P =^+ Q$.*

Proof: ($\Rightarrow$) It is enough to show that

$$\mathcal{S} = \{(P+R,\, Q+R) : P =^c Q\} \cup \approx$$

is a weak bisimulation.

The case of $(P,Q) \in \approx$ is trivial. We only consider here the case $(P+R, Q+R) \in \mathcal{S}$ for any agents P, Q and R, where $P =^c Q$.

- if $P + R \stackrel{\alpha}{\Longrightarrow}_u P'$

  Case 1 $P \stackrel{\alpha}{\Longrightarrow}_u P'$. If $\alpha \neq \tau$, then $Q \stackrel{\alpha}{\Longrightarrow}_u Q'$ for some Q' such that $P' \approx Q'$. Now suppose $\alpha = \tau$. Then $P \longrightarrow_w \stackrel{\tau}{\longrightarrow}_0 \stackrel{\epsilon}{\Longrightarrow}_{u-w} P'$ for some w, where $w \leq u$. Since $P =^c Q$, we have $Q \longrightarrow_w \stackrel{\tau}{\Longrightarrow}_0 \stackrel{\epsilon}{\Longrightarrow}_{u-w} Q'$ for some Q' such that $P' \approx Q'$. So $Q + R \stackrel{\alpha}{\Longrightarrow}_u Q'$ and $P' \approx Q'$ for some Q'.

  Case 2 $R \stackrel{\alpha}{\Longrightarrow}_u P'$, then $Q + R \stackrel{\alpha}{\Longrightarrow}_u P'$.

- if $P + R \longrightarrow_u P'$, then $u \leq max(\,|\,P\,|_{\mathcal{T}},\,|\,R\,|_{\mathcal{T}})$.

  Case 1 $u \leq min(\,|\,P\,|_{\mathcal{T}},\,|\,R\,|_{\mathcal{T}})$, then $P' \equiv P'' + R'$ where $P \longrightarrow_u P''$ and $R \longrightarrow_u R'$. Clearly $Q \longrightarrow_u Q''$ and $P'' =^c Q''$ for some Q''. So $Q + R \longrightarrow_u Q'' + R'$ and $(P'' + R', Q'' + R') \in \mathcal{S}$.

  Case 2 $|\,P\,|_{\mathcal{T}} < u \leq |\,R\,|_{\mathcal{T}}$, then $R \longrightarrow_u P'$. So $Q + R \longrightarrow_u P'$ and $P' \approx P'$.

  Case 3 $|\,R\,|_{\mathcal{T}} < u \leq |\,P\,|_{\mathcal{T}}$, then $P \longrightarrow_u P'$. Clearly $Q \longrightarrow_u Q'$ and $P' =^c Q'$ for some Q'. So $Q + R \longrightarrow_u Q'$. It is easy to check that $P' =^c Q'$ implies $P' \approx Q'$.

By symmetry, we are done.

($\Leftarrow$) Suppose $P \neq^c Q$ and for any $a \in Act$, $u \in \mathcal{T}$

Case 1 $P \longrightarrow_u \cdot \stackrel{\alpha}{\longrightarrow}_0 P'$, but whenever $Q \longrightarrow_u \cdot \stackrel{\alpha}{\Longrightarrow}_0 Q'$ then $P' \not\approx Q'$.

We choose $R \equiv \ell(t)^u_u.\, nil$, where ℓ is a fresh action which does not occur in P and Q. Since $P + R \longrightarrow_u \cdot \stackrel{\alpha}{\longrightarrow}_0 P'$, we show that whenever $Q + R \Longrightarrow_u \stackrel{\alpha}{\Longrightarrow}_0 Q'$ then $P' \not\approx Q'$. Clearly, $\Longrightarrow_u \;\equiv\; \longrightarrow_u$, otherwise $Q + R \Longrightarrow_u Q''$, where $Q \Longrightarrow_u Q''$, and $u >> (P + R) \not\approx Q''$. If $\alpha = \tau$ and $Q' \equiv u >> (Q + R)$, then $P' \not\approx Q'$ (Q' may perform action ℓ). In all other cases, we have $Q + R \longrightarrow_u \cdot \stackrel{\alpha}{\Longrightarrow}_0 Q'$, where $Q \longrightarrow_u \cdot \stackrel{\alpha}{\Longrightarrow}_0 Q'$, and $P' \not\approx Q'$. So we have $P \neq^+ Q$.

Case 2 $P \longrightarrow_u P'$, but $\mid Q \mid_{\mathcal{T}} < u$.

If $P' \not\approx \delta$, then we choose a v, where $\mid Q \mid_{\mathcal{T}} < v \leq u$, and let $R \equiv \ell(t)^v_v.\, nil$, where ℓ is a fresh action which does not occur in P and Q. Then $P + R \longrightarrow_v v >> (P + R)$ is not matched by $Q + R$. Now suppose $P' \approx \delta$. Let $v = \mid Q \mid_{\mathcal{T}}$ and $R \equiv \ell(t)^v_v.\, nil$, where ℓ is a fresh action which does not occur in P and Q. If $Q \xrightarrow{\tau}_v Q'$, then $Q + R \xrightarrow{\tau}_v Q'$ is not matched by $P + R$. Otherwise $P + R \longrightarrow_v P'' + \ell(t)^0_0.\, nil$, where $P \longrightarrow_v P''$ and $P'' transu - vP'$, is not matched by $Q + R$. So we have $P \neq^+ Q$.

By symmetry, we are done. $\square$

Definition 7.2.10 *We say $\mathcal{S}$ is a C-bisimulation up to $=^c$ if $P\mathcal{S}Q$ implies that*

(1) *if $P \xrightarrow{\alpha}_0 P'$, then $Q \stackrel{\alpha}{\Longrightarrow}_0 Q'$ and $P' \approx Q'$ for some Q';*

(2) *if $P \longrightarrow_u P'$, then $Q \longrightarrow_u Q'$ and $P' =^c \mathcal{S} =^c Q'$ for some Q';*

(3) *if $Q \xrightarrow{\alpha}_0 Q'$, then $P \stackrel{\alpha}{\Longrightarrow}_0 P'$ and $P' \approx Q'$ for some P'; and*

(4) *if $Q \longrightarrow_u Q'$, then $P \longrightarrow_u P'$ and $P' =^c \mathcal{S} =^c Q'$ for some P'.*

Proposition 7.2.11 *If $\mathcal{S}$ is a C-bisimulation up to $=^c$, then $\mathcal{S} \subseteq =^c$.*

Proof: Analogous to Proposition 4.1.9. $\square$

Hence, to prove $P =^c Q$, we only need to find a C-bisimulation up to $=^c$ which contains (P, Q).

Now we show that $=^+$ is a congruence relation, i.e. it is substitutive for all operators including the recursive operator.

Proposition 7.2.12 $E =^+ F$ *implies*

(1) $\alpha(t)_e^{e'}.E =^+ \alpha(t)_e^{e'}.F$ (4) $E\backslash a =^+ F\backslash a$

(2) $E + G =^+ F + G$ (5) $E[S] =^+ F[S]$

(3) $E \mid G =^+ F \mid G$

Proof: To prove (2), we only need to show that for every agent P, $(E+G)+P \approx (F+G)+P$ which follows from $E =^+ F$. All other properties can be proved by exhibiting appropriate C-bisimulations. For example, for (3), we only need to show that the relation

$$\mathcal{S} = \{(P \mid R, Q \mid R) : P =^c Q\}$$

is a C-bisimulation. □

Now we show that $=^+$ is also preserved by recursive operator.

Proposition 7.2.13 *For any processes $\vec{E}$ and $\vec{F}$ which contain at most process variables $\vec{X}$, $\vec{E} =^+ \vec{F}$ implies $\mu\vec{X}.\vec{E} =^+ \mu\vec{X}.\vec{F}$.*

Proof: Analogous to Proposition 7.2.8 of [Mil89a]. □

7.3 Unique Solution of Equations

In this section, we show that under certain conditions the solutions of equations $\vec{X} = \vec{E}$ are still unique up to observational congruence.

Definition 7.3.1 *X is sequential in E if every subterm of E which contains X, apart from X itself, is of the form $\alpha(t)_e^{e'}.F$ or $F_1 + F_2$.*

Definition 7.3.2 *X is guarded in E if every occurrence of X is within some subterm $a(t)_e^{e'}.F$ of E, where $a \neq \tau$.*

We say a process E is guarded (sequential) if all process variables of E are guarded (sequential) in E. Equations $\vec{X} = \vec{E}$ are guarded (sequential) if all processes $\vec{E}$ are guarded (sequential). Similar to weak guardedness, the guardedness ensures that process variables cannot affect the first non-τ action.

Lemma 7.3.3 *For any sequential process E and agents $\vec{P}$, if process variables $\vec{X}$ are guarded in E and $E\{\vec{P}/\vec{X}\}\xrightarrow{\alpha}_u P'$, then there is a process H such that $P' \equiv H\{\vec{P}/\vec{X}\}$ and for any agents $\vec{Q}$, we have $E\{\vec{Q}/\vec{X}\}\xrightarrow{\alpha}_u H\{\vec{Q}/\vec{X}\}$. Moreover H is also sequential and if $\alpha = \tau$, then H is still guarded.*

Proof: Suppose $E\{\vec{P}/\vec{X}\}\xrightarrow{\alpha}_u P'$. We proceed by induction on the structure of E. If E is a Recursion, a Composition, a Restriction or a Relabelling, then it contains no free process variables and the result is immediate (recall the definition of a sequential process). If E is a Summation, the result follows from induction.

If $E \equiv \alpha(t)_v^{v'}.F$, then $P' \equiv F\{\vec{P}/\vec{X}\}\{u/t\}$, where $v \leq u \leq v'$. Since $\vec{P}$ are agents, $F\{\vec{P}/\vec{X}\}\{u/t\} \equiv F\{u/t\}\{\vec{P}/\vec{X}\}$. Also, for any agents $\vec{Q}$,

$$E\{\vec{Q}/\vec{X}\}\xrightarrow{\alpha}_u F\{\vec{Q}/\vec{X}\}\{u/t\} \qquad \text{and} \qquad F\{\vec{Q}/\vec{X}\}\{u/t\} \equiv F\{u/t\}\{\vec{Q}/\vec{X}\}$$

The result follows by taking $H \equiv F\{u/t\}$. Note that H must also be guarded if $\alpha = \tau$. □

Now we can show that for any guarded and sequential equations $\vec{X} = \vec{E}$, where processes $\vec{E}$ contain at most free process variables $\vec{X}$, if the equations have solutions, then the solutions are unique up to observational congruence.

Proposition 7.3.4 *Let $\vec{E}$ be guarded and sequential processes which are closed with respect to time variables and contain at most free process variables $\vec{X}$,*

$$\vec{P} =^+ \vec{E}\{\vec{P}/\vec{X}\} \text{ and } \vec{Q} =^+ \vec{E}\{\vec{Q}/\vec{X}\} \text{ implies } \vec{P} =^+ \vec{Q}$$

Proof: Analogous to Proposition 7.3.13 of [Mil89a]. $\square$

Similarly to strong bisimulation, any guarded and sequential equations have solutions. As a result, any guarded and sequential equations have unique solution up to observational congruence.

Corollary 7.3.5 (Unique Solution) *For any guarded and sequential equations $\vec{X} = \vec{E}$, where $\vec{E}$ are closed with respect to time variables and contain at most free process variables $\vec{X}$, there is an unique solution $\vec{P}$ (up to observational congruence) such that*

$$\vec{P} =^{+} \vec{E}\{\vec{P}/\vec{X}\}$$

7.4 Conclusion

In this section, we extended the notions of weak bisimulations and observational congruence to real-time processes. In [Cer92], it is shown that weak bisimulation for parallel networks of regular timed processes without time variables is decidable. However decidability and complete axiomatization problems of weak bisimulations for general finite processes (with time variables) are still open. It is not clear whether the techniques discussed in Chapter 5 and Chapter 6 can be used for weak bisimulations or observational congruence. As $P \approx Q$ in general does not imply $\tau(t)_e^{e'}.P =^{+} Q$ or $P =^{+} \tau(t)_e^{e'}.Q$ for some e and e', this leaves the completeness problem hard.

Chapter 8

Observing Causality in Real-Timed Calculi

CCS is a symbolic calculus, in the sense that it treats solely the observation of events of a system, and not their relative time, locations or duration. All these are abstracted away from consideration. For example, the term $a.P$ in CCS represents the process which can perform an action a and evolve into the process P. The place, time and duration of the action a are not specified. The process is assumed to be idling for some amount of time before the action a occurs, and then some time later the new state P is reached. In this chapter we show that by specifying temporal aspects of processes, we can observe the usual notions of causality, concurrency and conflict within a system in the following sense:

- two events are causally related if they can both occur in the same run, but never at the same time. Here we make the assumption that a process $a.P$ must delay by some nonzero time after the event a in order to stabilize.

- two events occur concurrently if they may occur at the same time.

- two events are conflicting if they cannot both occur in the same run.

With this observation, we show that we can develop an interleaving theory for CCS which respects these relationships, i.e. we develop an interleaving theory of a true concurrency semantics for CCS.

This chapter is joint work with Anderson and Moller and the work is still in progress [ACM92].

8.1 An Event Structure Semantics for CCS

We only treat here a simple language of finite terms which are defined by the following BNF expression.

$$P \ ::= \ nil \,\big|\, \alpha.P \,\big|\, P + Q \,\big|\, P \mid Q$$

The results can be extended to the language with recursion and relabelling; their omission is only to facilitate an elegant presentation of the key ideas of the chapter.

In this section, we will propose an interpretation of CCS terms over labelled event structures, as presented in [BC89,DDM89].

Definition 8.1.1 *Two labelled event structures E and F are* isomorphic*, written $E \cong F$, if there exists a label-, order- and conflict-preserving bijection between them.*

A semantics in terms of labelled event structures is still intensional, as in the case for the interleaving semantics based on synchronization trees. In [DDM89], the notion of partial order bisimulation on labelled event structures is defined.

Configurations of event structures can be regarded as special cases of event structures, with inherited concurrency, causality and conflict relations. As a result, we can define configuration isomorphism as the event structure isomorphism.

Definition 8.1.2 *A configuration F' of an event structure S is called an S-* extension *of the configuration F of S whenever $F \subseteq F'$.*

Definition 8.1.3 *A binary relation $\mathcal{R}$ over (event structure/configuration) pairs is a* po-bisimulation *if $(S, E)\mathcal{R}(T, F)$ implies $E \cong F$ and*

1 *if E' is an S-extension of E, then there is an T-extension F' of F such that $(S, E')\mathcal{R}(T, F')$; and*

2 *if F' is an T-extension of F, then there is an S-extension E' of E such that $(S, E')\mathcal{R}(T, F')$.*

Two event structures S and T are po-bisimilar, *written as $S \sim_{PO} T$, if $\big((S, \emptyset), (T, \emptyset)\big) \in \mathcal{R}$ for some po-bisimulation $\mathcal{R}$.*

We may interpret a CCS term as an event structure using a mapping $\mathcal{S}$, where the mapping $\mathcal{S}$ is inductively defined as following:

- $\mathcal{S}(nil) = (\emptyset, \emptyset, \emptyset, \emptyset)$;
- If $\mathcal{S}(P) = (E, \prec, \#, \ell)$, then $\mathcal{S}(\alpha.P) = (E \cup \{\varepsilon\}, \prec', \#, \ell')$, where $\varepsilon \notin E$ and
  - $\prec' = \prec \cup \big\{(\varepsilon, e) : e \in E\big\}$
  - $\ell' = \ell[\varepsilon \mapsto \alpha]$
- If $\mathcal{S}(P_i) = (E_i, \prec_i, \#_i, \ell_i)$ for $i = 0, 1$, then $\mathcal{S}(P_0 + P_1) = (E, \prec, \#, \ell)$ where
  - $E = E_0 \uplus E_1$
  - $\prec = \prec_0 \uplus \prec_1$
  - $\# = \#_0 \uplus \#_1 \uplus \big\{(e_0, e_1), (e_1, e_0) : e_i \in E_i\big\}$
  - $\ell = \ell_0 \uplus \ell_1$

- If $\mathcal{S}(P_i) = (E_i, \prec_i, \#_i, \ell_i)$ for $i = 0, 1$, then $\mathcal{S}(P_0 \mid P_1) = (E, \prec, \#, \ell)$ where
  - $E = E_0 \uplus E_1 \uplus \Big\{(e_0, e_1) \;:\; e_i \in E_i \;\&\; \ell_0(e_0) = \overline{\ell_1(e_1)}\Big\}$
  - $\prec = \prec_0 \uplus \prec_1 \uplus \Big\{\big(e_j, (e_0', e_1')\big),\; \big((e_0, e_1), e_j'\big) \;:\; e_i, e_i' \in E_i \;\&\; e_i \prec_i e_i'\Big\}$
    $\uplus \Big\{\big((e_0, e_1), (e_0', e_1')\big) \;:\; e_i, e_i' \in E_i \;\&$
    $(e_0 \prec_0 e_0' \;\vee\; e_1 \prec_1 e_1') \;\&\; \neg(e_i \# e_i')\Big\}$
  - $\# = \#_0 \uplus \#_1 \uplus \Big\{\big(e_j, (e_0', e_1')\big),\; \big((e_0, e_1), e_j'\big) \;:\; e_i, e_i' \in E_i \;\&\; e_i \#_i e_i'\Big\}$
    $\uplus \Big\{\big((e_0, e_1), (e_0', e_1')\big) \;:\; e_i, e_i' \in E_i \;\&\; (e_0 \#_0 e_0' \;\vee\; e_1 \#_1 e_1')\Big\}$
  - $\ell = \ell_0 \uplus \ell_1 \uplus \Big\{\big((e_0, e_1), \tau\big) \;:\; e_i \in E_i \;\&\; \ell_0(e_0) = \overline{\ell_1(e_1)}\Big\}$

Definition 8.1.4 *For any CCS agents P and Q, we say $P \sim_{PO} Q$ if $\mathcal{S}(P) \sim_{PO} \mathcal{S}(Q)$.*

Partial order bisimulation provides us with more discriminating power than interleaving bisimulation. As an example, we have

$$a \mid b \;\not\sim_{PO}\; a \mid b + ab$$

Compared with the pomset bisimulation of [BC87], partial order bisimulation also considers the (past) history of an actual step. As a result, partial order bisimulation has more discriminating power than pomset bisimulation. For example, we have

$$a \mid b + a(b + c) + ab \;\not\sim_{PO}\; a \mid b + a(b + c)$$

but pomset bisimulation cannot distinguish these two agents. As claimed in [DDM89] these agents should be differentiated whenever causal dependencies are important.

8.2 A Timed Semantics for CCS

In this section we interpret the symbolic calculus CCS within the loose temporal calculus ℓTCCS of [MT91], a sublanguage of the calculi TCCS of [MT90] and Timed CCS of Chapter 3.

Our model is an extension of CCS with real-time. Processes are defined by the following BNF expression, where t ranges over $\mathbb{R}^+$, the positive reals. We let $\underline{\mathrm{P}}$, $\underline{\mathrm{Q}}$ range over agents of ℓTCCS to avoid confusion with CCS agents. Also, the (delayed) *nil* and (delayed) action prefix operators of ℓTCCS are written with underscores.

$$\underline{\mathrm{P}} \;\; ::= \;\; \underline{nil} \,\Big|\, \underline{\alpha.\mathrm{P}} \,\Big|\, (t).\underline{\mathrm{P}} \,\Big|\, \underline{\mathrm{P}} + \underline{\mathrm{Q}} \,\Big|\, \underline{\mathrm{P}} \mid \underline{\mathrm{Q}}$$

In Table 8–1, we present the operational rules for the language. The transitional semantics of the language is then given by the least relations

$$\longrightarrow \;\subseteq \ell\mathrm{TCCS} \times \mathrm{Act} \times \ell\mathrm{TCCS} \text{ and } \leadsto \subseteq \ell\mathrm{TCCS} \times \mathbb{R}^+ \times \ell\mathrm{TCCS}$$

written as $\underline{\mathrm{P}} \xrightarrow{\alpha} \underline{\mathrm{Q}}$ and $\underline{\mathrm{P}} \overset{t}{\leadsto} \underline{\mathrm{Q}}$ respectively, which satisfy the rules laid out in Figure 8–1. The rules for action derivations are identical to those in the calculus CCS. The only new feature is the temporal derivations.

Remark Although the time constants are taken from the positive reals, we allow ourselves to interpret $(0).\underline{\mathrm{P}}$ as syntactically identical to $\underline{\mathrm{P}}$. Equally, we allow ourselves to write $\underline{\mathrm{P}} \overset{0}{\leadsto} \underline{\mathrm{P}}$. □

Definition 8.2.1 *We write* $\underline{P} \xrightarrow{\alpha}_t \underline{P'}$ *whenever* $\underline{P} \overset{t}{\leadsto} \xrightarrow{\alpha} \underline{P'}$, *and* $\underline{P} \xrightarrow{\alpha_1 \cdots \alpha_n}_{t_1 + \cdots + t_n} \underline{P'}$ *whenever* $\underline{P} \xrightarrow{\alpha_1}_{t_1} \cdots \xrightarrow{\alpha_n}_{t_n} \underline{P'}$. *In particular,* $\underline{P} \xrightarrow{\varepsilon}_0 \underline{P}$.

Definition 8.2.2 *A binary relation* $\mathcal{R}$ *over* ℓ*TCCS is a* strong temporal bisimulation *if* $(\underline{P}, \underline{Q}) \in \mathcal{R}$ *implies that for all* $\alpha \in Act$ *and* $t \in \mathbb{R}^+$,

$$\frac{}{\underline{nil} \overset{t}{\leadsto} \underline{nil}} \qquad \frac{}{\underline{\alpha}.\underline{P} \overset{t}{\leadsto} \underline{\alpha}.\underline{P}}$$

$$\frac{}{(s+t).\underline{P} \overset{s}{\leadsto} (t).\underline{P}} \qquad \frac{}{(t).\underline{P} \overset{t}{\leadsto} \underline{P}}$$

$$\frac{\underline{P} \overset{s}{\leadsto} \underline{P'}}{(t).\underline{P} \overset{s+t}{\leadsto} \underline{P'}}$$

$$\frac{\underline{P} \overset{t}{\leadsto} \underline{P'}, \quad \underline{Q} \overset{t}{\leadsto} \underline{Q'}}{\underline{P} + \underline{Q} \overset{t}{\leadsto} \underline{P'} + \underline{Q'}} \qquad \frac{\underline{P} \overset{t}{\leadsto} \underline{P'}, \quad \underline{Q} \overset{t}{\leadsto} \underline{Q'}}{\underline{P} \mid \underline{Q} \overset{t}{\leadsto} \underline{P'} \mid \underline{Q'}}$$

$$\frac{}{\underline{\alpha}.\underline{P} \xrightarrow{\alpha} \underline{P}} \qquad \frac{\underline{P} \xrightarrow{\alpha} \underline{P'}}{\underline{P} + \underline{Q} \xrightarrow{\alpha} \underline{P'}}$$

$$\frac{\underline{Q} \xrightarrow{\alpha} \underline{Q'}}{\underline{P} + \underline{Q} \xrightarrow{\alpha} \underline{Q'}} \qquad \frac{\underline{P} \xrightarrow{\alpha} \underline{P'}}{\underline{P} \mid \underline{Q} \xrightarrow{\alpha} \underline{P'} \mid \underline{Q}}$$

$$\frac{\underline{Q} \xrightarrow{\alpha} \underline{Q'}}{\underline{P} \mid \underline{Q} \xrightarrow{\alpha} \underline{P} \mid \underline{Q'}} \qquad \frac{\underline{P} \xrightarrow{a} \underline{P'}, \quad \underline{Q} \xrightarrow{\bar{a}} \underline{Q'}}{\underline{P} \mid \underline{Q} \xrightarrow{\tau} \underline{P'} \mid \underline{Q'}}$$

Table 8–1: Operational Rules for ℓTCCS

1. *if* $\underline{P} \xrightarrow{\alpha}_t \underline{P}'$ *then* $\underline{Q} \xrightarrow{\alpha}_t \underline{Q}'$ *for some* $\underline{Q}'$ *with* $(\underline{P}', \underline{Q}') \in \mathcal{R}$ *and*

2. *if* $\underline{Q} \xrightarrow{\alpha}_t \underline{Q}'$ *then* $\underline{P} \xrightarrow{\alpha}_t \underline{P}'$ *for some* $\underline{P}'$ *with* $(\underline{P}', \underline{Q}') \in \mathcal{R}$.

Two agents $\underline{P}$ *and* $\underline{Q}$ *are* bisimilar, *denoted by* $\underline{P} \sim_T \underline{Q}$, *if there is a strong temporal bisimulation* $\mathcal{R}$ *with* $(\underline{P}, \underline{Q}) \in \mathcal{R}$.

Remark Since ℓTCCS is a sublanguage of Timed CCS, it is easily to see that $\sim_T$ coincides with strong equivalence $\sim$ of Chapter 5. Therefore $\sim_T$, the largest temporal bisimulation, is an equivalence relation and also a congruence relation, i.e. it is fully substitutive. □

We now consider an interpretation of CCS over ℓTCCS. Our interpretation is based on a postulation that between any two actions which are causally related, there is some non-zero delay Δ. We can understand the postulation as that every atomic action has a duration Δ. The interpretation is then defined by the following.

Definition 8.2.3

$$
\begin{aligned}
T(nil) &= \underline{nil} & \qquad T(P+Q) &= T(P)+T(Q) \\
T(\alpha.P) &= \underline{\alpha}.(\Delta).T(P) & \qquad T(P \mid Q) &= T(P) \mid T(Q)
\end{aligned}
$$

We now define a timed equivalence on CCS.

Definition 8.2.4 *For any CCS processes* P *and* Q, *we say* P *and* Q *are timed equivalent, written as* $P \sim_T Q$, *if* $T(P) \sim_T T(Q)$.

Timed equivalence $\sim_T$ over CCS distinguishes more things than strong equivalence. For example,

$$a \mid b \;\not\sim_T\; a \mid b \,+\, ab$$

Moreover, timed equivalence $\sim_T$ over CCS has more discriminating power than pomset bisimulation of [BC87]. As an example, we have

$$a \mid b + a(b+c) + ab \quad \not\sim_T \quad a \mid b + a(b+c)$$

We actually believe that our timed equivalence over CCS is the same as the partial order bisimulation of [DDM89] as described in Section 8.1.

8.3 Conclusion

In the usual operational semantics of CCS and the related observational equivalence, atomic actions are instantaneous indivisible events. However it can be argued that every atomic action will take some time. In this chapter, we postulated that every atomic action has a non-zero constant duration. We satisfied this by assuming that for any two atomic actions which are causally related there is at least a non-zero constant delay Δ between them. Based on this postulation, we translated CCS processes into timed processes of ℓTCCS and thus gave a timed equivalence for CCS (the induced equivalence). The resulting equivalence has more discriminating power than strong equivalence and distinguishes concurrency from nondeterminism. It is in fact a partial order or true concurrency semantics.

The importance of the timed semantics for CCS is in the use of an interleaving model for partial order or true concurrency semantics. This suggests an interleaving approach for the study of true concurrency semantics. It is well known that the interleaving approach is simple and has elegant mathematical techniques. In Chapter 6, we presented a simple proof system for deciding timed equivalence. With the result of this chapter, it can also be used for partial order or true concurrency.

In [Hen88b], Hennessy also postulated that every external action (non-τ action) has a duration. Rather than associate specific durations with actions, he introduced observers which could distinguish the inception of an action from its termination. The observational equivalence based on those new observers also distinguished concurrency from nondeterminism. However, since the internal action τ was still instantaneous, one could not differentiate the process $a|\tau$ from the process $a|\tau + \tau a$ in the strong version of Hennessy's equivalence.

Chapter 9

Conclusions and Future Work

In this thesis, we presented a timed calculus, Timed CCS, which is a general model of concurrency. We made no assumptions about the underlying nature of time, allowing it to be discrete or dense. Timed CCS can be used for the specification and verification of real-time systems. We gave some examples to show the uses of the calculus for formalising three important notions, time out, duration control and time dependency, of real-time systems. We studied the theory of the abstract semantics of Timed CCS and provided a mathematical model for a wide range of timed calculi including Timed CCS. Timed CCS is an interleaving model for real-time systems. If we postulate that every atomic action has a non-zero constant duration, then we can observe the usual notions of causality, concurrency and conflict relations between events of a system. We satisfied the postulation by insisting that for any two causally related events there is at least a non-zero constant delay between them. By interpreting CCS in a sublanguage ℓTCCS of Timed CCS and TCCS [MT90] based on this postulation, we developed a partial order or true concurrency semantics for CCS. As a consequence, we have developed a partial order semantics by using an interleaving approach.

There are a few problems which are worth future work. In Chapter 5, we presented a simple modal logic which is an extension of Hennessy-Milner logic with time. We showed the modal characterization of strong bisimulation. This in turn suggested that modal logic is an appropriate logic for the specification and verification of non-deterministic, concurrent and real-time systems. In order to support modular design and verification of systems in this framework, the problem of compositionality of modal assertions arose. This problem has been successfully dealt with by Stirling [Sti87] and Winskel [Win84b] for CCS and SCCS. Also a formula of our modal logic is only capable of describing those properties which only concern certain finite behaviours of processes. As a consequence, it is impossible to use a formula to describe properties of processes which are to hold invariantly or eventually. All these properties are clearly important in practice. By extending the modal logic with infinite connectives, these invariant and eventual properties are expressible. However we are also concerned with the problem of verifying when a process satisfies a formula. This requires us to insist that all formulas of our logic should be finitary expressions. In [Lar88], Larsen presents proof systems for Hennessy-Milner logic with recursion. As future work, we can consider how to incorporate time into modal μ-calculus and study compositional proof systems for Timed CCS.

In Chapter 8, we studied the theory of behavioural abstraction in timed processes. As the timed version of Hennessy's theorem, which says that $P \approx Q$ implies $P =^{+} Q$ or $\tau.P =^{+} Q$ or $P =^{+} \tau.Q$, no longer holds, this leaves the completeness problem of observational congruence of timed processes hard and still open. However, if we restrict ourselves to a special case and assume maximal progress, then we can retain Hennessy's theorem for timed processes and the solution to the problem is straightforward.

In [MT91], Moller and Tofts have proposed a *faster than* preorder on concurrent processes which distinguishes between functionally behaviourally equivalent processes operating at different speeds. For timed processes, timing also implicitly introduces a causality relation between events. For example, in process $a(t)_{0'}^{5}\, nil \mid b(s)_{10'}^{15}\, nil$, action b can only occur after the occurrence of action a. Since we do not distinguish between the causality introduced explicitly by the prefix operator and the causality introduced implicitly by timing, this leads to the *faster than* preorder being a precongruence only when we do not allow the notion of time out (e.g. for Timed CCS, the upper bound e' of prefix $\alpha(t)_e^{e'}.E$ must be the infinite time ∞). It would be nice if we could have some preorder which is also a precongruence for a language allowing time out notion, e.g. Timed CCS or TCCS of [MT90].

Another interesting topic is adding probabilities to Timed CCS. This may enable us to consider temporal, probabilistic and computational behaviour simultaneously. The resulting calculus would be expressively powerful to represent notions of safety, liveness and fairness. This would facilitate the unification and comparison of different models of concurrency and promote a deep understanding of concurrency.

Bibliography

[AD90] R. Alur & D. Dill, *Automata for Modelling Real-Time Systems* Proc. ICALP'90, Lecture Notes in Computer Science 443, pp 322-335, 1990

[ACD90] *Model-Checking for Real-Time Systems*, Proc. of LICS'90, pp 414-425, 1990

[ACM92] S. Anderson, L. Chen & F. Moller, *Observing Causality in Real-Timed Calculi*, Draft Note, LFCS, University of Edinburgh, 1992

[BB91] J.C.M. Baeten & J.A. Bergstra, *Real Time Process Algebra*, Formal Aspects of Computing, Vol 3, No 2, pp142-188, 1991

[BKR86] M. Ben-Or, D. Kozen & J. Reif, *The Complexity of Elementary Algebra and Geometry*, Journal of Computer and System Sciences, 32, pp 251-264, 1986

[BK84] J.A. Bergstra & J.W. Klop, *Process Algebra for Synchronous Communication*, Information and Control 60(1-3), pp. 109-137, 1984

[BK85] J.A. Bergstra & J.W. Klop, *Algebra of Communicating Processes with Abstraction*, Theoretical Computer Science 37, pp 77-12, 1985

[BC87] G. Boudol & I. Castellani, *On the Semantics of Concurrency: Partial Order and Transition Systems*, Proc. of TAPSOFT-CAAP'87, Lecture Note in Computer Science 249, pp 123-137, 1987

[BC89] G. Boudol & I. Castellani, *Permutation of Transitions: An Event Structure Semantics for CCS and SCCS*, Proc. of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science 354, pp 411-427, 1989

[BS92] J. Bradfield & C. Stirling, *Local Model Checking for Infinite State Spaces*, Theoretical Computer Science 96, pp 157-174, 1992

[BHR84] S.D. Brookes, C.A.R. Hoare & A.D. Roscoe, *A Theory of Communicating Sequential Processes*, Journal of the ACM, Vol. 31, No. 3, 1984

[Cer92] K. Cerans, *Decidability of Bisimulation Equivalences for Parallel Timer Processes*, in Proc. of the 4th Workshop on Computer-Aided Verification, 1992

[Cle90] R. Cleaveland, *Tableau-based Model Checking in Propositional mu-Calculus*, Acta Inform. 27, pp 725-747, 1990

[Che91a] L. Chen, *Specification and Verification of Real-Time Systems*, Draft Note, LFCS, University of Edinburgh, 1991

[Che91b] L. Chen, *Decidability and Completeness in Real-Time Processes*, Technical Report ECS-LFCS-91-185, Edinburgh University, 1991

[Che92] L. Chen, *An Interleaving Model for Real-Time Systems*, in Proc. of Symp. of Logical Foundations of Computer Science, Lecture Notes in Computer Science 620, pp 81-92, 1992

[Che93] L. Chen, *Axiomatising Real-Time Processes*, will appear in Proc. of MFPS IX, Lecture Notes in Computer Science, April, 1993

[CAM90] L. Chen, S. Anderson & F. Moller, *A Timed Calculus of Communicating System*, Technical Report ECS-LFCS-90-127, University of Edinburgh, 1990

[CM92] L. Chen & A. Munro, *Applications of Modal Logic for the Specifications of Real-Time Systems*, will appear in Proc. of FME'93, Lecture Notes in Computer Science, April, 1993

[Col75] G. E. Collins, *Quantifer Elimination for Real Closed Fields by Cylindric Algebraic Decomposition*, Proc. 2nd GI Conference on Automata Theory and Formal Language, pp 134-183, 1975

[DS89] J. Davies & S. Schneider, *An Introduction to Timed CSP*, PRG-75, Programming Researcg Group, Oxford University, 1989

[DDM89] P. Degano, R. De Nicola & U. Montanari, *Partial Orderings Descriptions and Observations of Nondeterministics Concurrent Processes*, Proc. of Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Lecture Notes in Computer Science 354, pp 438-466, 1989

[End72] H. B. Enderton, **A Mathematical Introduction to Logic**, Academic Press, Inc., 1972

[Gro90] J.F. Groote, *Specification and Verification of Real Time Systems in ACP*, Proc. of 10th International symp. on Protocol Specification, Testing and Verification, Ottawa, 1990

[Hen85] M. Hennessy, *Acceptance Trees*, Journal of the Association for Computing Machinery, Vol 32, No 4, pp 896-928, 1985

[Hen88a] M. Hennessy, **Algebraic Theory of Processes**, The MIT Press, 1988

[Hen88b] M. Hennessy, *Axiomatising Finite Concurrent Processes*, SIAM J. Comput. Vol 17, No 5, pp 997-1017, 1988

[Hen91] M. Hennessy, *A Proof System for Communicating Processes with Value-Passing*, Formal Aspects of Computing, Vol3, No 4, pp 346-366, 1991

[HM85] M. Hennessy & R. Milner, *Algebraic Laws for Nondeterministic and Concurrency*, J. Assoc. Comput. Mach. 32, pp 137-161, 1985

[HR90] M. Hennessy & T. Regan, *A Temporal Process Algebra*, Technical Report 2/90 University of Sussex, 1990

[Hoa85] C.A.R. Hoare, **Communicating Sequential Processes**, Prentice-Hall International, 1985

[HLW91] U. Holmer, K. Larsen & Y. Wang, *Deciding Properties of Regular Real Time Processes*, in CAV'91, Lecture Notes in Computer Science, 1991.

[Kel76] R. Keller, *Formal Verification of Parallel Programs*, Communication of the ACM, Vol. 19, No. 7, 1976

[Klu91] A.S. Klusener, *Completeness in Real Time Process Algebra*, Proceedings of CONCUR'91, Lecture Notes in Computer Science 527, pp 96-110, 1991

[Lar88] K. Larsen, *Proof systems for Hennessy-Milner Logic with Recursion*, Lecture Notes in Computer Science 299, 1988

[LL90] K. Larsen & X. Liu, *Compositionality through an Operational Semantics of Contexts*, Proc. ICALP'90, Lecture Notes in Computer Scuence 443, pp 526-539, 1990

[Mil80] R. Milner, *A Calculus of Communicating systems*, Lecture Notes in Computer Science 92, Springer-Verlag, 1980

[Mil83] R. Milner, *Calculi for Synchrony and Asynchrony*, Theoretical Computer Science, 25, pp 267-310, 1983

[Mil89a] R. Milner, **Communication and Concurrency**, Prentice-Hall International, 1989

[Mil89b] R. Milner, *A Complete Axiomatisation for Observational Congruence of finite-state Behaviours*, Information and Computation 81, pp. 227-247, 1989

[Mol89] F. Moller *Axioms for Concurrency*, Ph.D Thesis, CST-59-89, University of Edinburgh, 1989

[Mol90] F. Moller, *Process Algebra as a Tool for Time Analysis*, in IV Banff Higher Order Workshop, Springer-Verlag Workshops in Computer Science, 1990

[MT90] F. Moller & C. Tofts, *A Temporal Calculus of Communicating System*, Proc. of CONCUR'90, Lecture Notes in Computer Science 458, pp 401-415, 1990

[MT91] F. Moller & C. Tofts, *Relating Processes With Respect to Speed*, Proc. of CONCUR'91, Lecture Notes in Computer Science 527, 1991

[Sif90] X. Nicollin, J. Richier, J. Sifakis & J. Voiron, *ATP : an Algebra for Timed Processes*, Laboratoire de Genie Informatique, IMAG-Campus, 38041 Grenoble Cedex, France, 1990

[NPW81] M. Nielsen, G. Plotkin & G. Winskel, *Petri Nets, Event Structures and Domains*, Theoretical Computer Science 13, pp 85-108, 1981

[Par81] D.M.R. Park, *Concurrency and Automata on Infinite Sequences*, Proc. of ICALP'81, Lecture Notes in Computer Science 104, Springer-Verlag, 1981

[Plo81] G.D. Plotkin, *A Structured Approach to Operational Semantics*, DAIMI FN-19, Computer Science Department, Arhus University, Denmark, 1981

[RR88] R. Reed & A. W. Roscoe, *A Timed Model for Communicating Sequential Processes*, Theoretical Computer Science, 58, pp 249-261, 1988

[Rei85] W. Reisig, **Petri Nets: an Introduction**, EATCS Monographs, Springer-Verlag, 1985

[SM92] D. Sangiogi & R. Milner, *The Problem of "Weak Bisimulation Up To"*, LFCS, University of Edinburgh (submitted for publication), 1992

[Sch91] S. Schneider, *An Operational Semantics for Timed CSP*, in Proc. of Chalmers Workshop on Concurrency, 1991.

[Smy78] M. B. Smyth, *Power Domains*, Journal of Computer and Systems Science 16, 1978

[Sti87] C. Stirling, *Modal Logics for Communication Systems*, Theoretical Computer Science 49, pp. 311-347, 1987

[Sti91] C. Stirling, *Modal and Temporal Logics* in Handbook of Logic in Computer Science, edited by S. Abramsky, D. Gabbay & T. Maibaum (also as Technical Report ECS-LFCS-91-157, University of Edinburgh), 1991

[SW89] C. Stirling & D. Walker, *Local Model Checking in the Propositional μ-Calculus*, Proc. of TAPSOFT'89, Lecture Notes in Computer Science 351, pp 369-383, 1989

[Tof91] C. Tofts, *Proof Methods and Pragmatics for Parallel Programming*, Ph.D Thesis, CST-74-91,University of Edinburgh, 1991

[Wan90] Y. Wang, *Real-time Behaviour of Asynchronous Agents*, Proc. of CONCUR'90, Lecture Notes in Computer Science 458, pp 502-520, 1990

[Wan91] Y. Wang, *CCS + Time = an Interleaving Model for Real Time Systems*, Proc. of ICALP'91, Lecture Notes in Computer Science 1991

[Win83] G. Winskel, *Event Structure Semantics for CCS and Related Languages*, Lecture Notes in Computer Science 140, pp 561-576, 1983

[Win84a] G. Winskel, *Synchronization Trees*, Theoretical Computer Science 34, pp32-82, 1984

[Win84b] G. Winskel, *On the Composition and Decomposition of Assertions*, Proc. of Seminar on Concurrency, Lecture Notes in Computer Science 197, 1984

[Win87] G. Winskel, *Event Structures*, Proc. of Petri Nets: Applications and Relations to Other Models of Concurrency, Lecture Notes in Computer Science 255, pp 325-392, 1987

[Win89] G. Winskel, *Model Checking the Modal ν-Calculus*, in Proc. ICALP'89, Lecture Notes in Computer Science, 1989